班级 031111

学号 03111002

本科毕业设计（论文）

外文资料翻译

毕业设计题目 大规模图数据库的图的相似性搜索

外文资料题目 G-Hash: Towards Fast Kernel-base Similarity Search in Large Graph Databases

学　　院 计算机学院

专　　业 计算机科学与技术

学 生 姓 名 贾新禹

指导教师姓名 霍红卫

G-Hash: Towards Fast Kernel-based Similarity Search in Large Graph Databases

Xiaohong Wang[1],Aaron Smalter[1],Jun Huan[1], Gerald H. Lushington[2]
[1]Department of Electrical Engineering and Computer Science,
University of Kansas,Lawrence, KS, USA
[1]{xwang85,asmalter,jhuan}@eecs.ku.edu
[2]Molecular Graphics and Modeling Laboratory,
University of Kansas,Lawrence, KS, USA
[2]glushington@ku.edu

ABSTRACT

Structured data including sets, sequences, trees and graphs, pose significant challenges to fundamental aspects of data management such as efficient storage, indexing, and similarity search. With the fast accumulation of graph databases, *similarity search* in graph databases has emerged as an important research topic. Graph similarity search has applications in a wide range of domains including cheminformatics, bioinformatics, sensor network management, social network management, and XML documents, among others.

Most of the current graph indexing methods focus on subgraph query processing, i.e. determining the set of database graphs that contains the query graph and hence do not directly support similarity search. In data mining and machine learning, various graph kernel functions have been designed to capture the intrinsic similarity of graphs. Though successful in constructing accurate predictive and classification models for supervised learning, graph kernel functions have (i) high computational complexity and (ii) non-trivial difficulty to be indexed in a graph database.

Our objective is to bridge graph kernel function and similarity search in graph databases by proposing (i) a novel kernel-based similarity measurement and (ii) an efficient indexing structure for graph data management. Our method of similarity measurement builds upon local features extracted from each node and their neighboring nodes in graphs. A hash table is utilized to support efficient storage and fast search of the extracted local features. Using the hash table, a graph kernel function is defined to capture the intrinsic similarity of graphs and for fast similarity query processing. We have implemented our method, which we have named G-hash, and have demonstrated its utility on large chemical graph databases. Our results show that the G-hash method achieves state-of-the-art performance for k-nearest neighbor (k-NN) classification. Most importantly, the new similarity measurement and the index structure is scalable to large database with smaller indexing size, faster indexing construction time, and faster query processing time as compared to state-of-the-art indexing methods such as C-tree, gIndex, and GraphGrep.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM.
EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.
Copyright 2009 ACM 978-1-60558-422-5/09/0003 ...$5.00

Keywords

graph similarity query, graph classification, hashing, graph kernels, k-NNs search.

1. INTRODUCTION

Structured data including sets, sequences, trees and graphs, pose significant challenges to fundamental aspects of data management such as efficient storage, indexing, component search (e.g.subgraph/supergraph search) and similarity search. With the fast accumulation of graph databases, *similarity search* in graph databases has emerged as an important research topic. Graph similarity search has applications in a wide range of domains including cheminformatics, bioinformatics, sensor network management, social network management, and XML documents, among others. For example, in chemistry and pharmacy, there is a fast accumulation of chemical molecule data. Once a new chemical is synthesized, the properties of the chemical may be revealed through querying existing chemicals with known properties. Fast similarity search in large graph databases enable scientists and data engineers to build accurate models for graphs, identify the intrinsic connections between graph data, and reduce the computational cost of processing large databases.

Graph queries can be classified into two categories: (i) subgraph query and (ii) similarity query. *Subgraph query* aims to identify a set of graphs that contain a query graph [12]. *Similarity query* aims to identify similar graphs in a graph database to a query, according to a distance metric. There are two types of similarity query, i.e. k-NNs query and range query. In k-NNs query, the k most similar graphs are reported. In range query, all graphs within a predefined distance to the query graph are reported. In this paper we address the problem of k-NN similarity search in large databases of graphs.

Similarity search on graphs is challenging. We argue that an ideal design of fast similarity search should achieve the following three related (sometimes contradicting) objectives: accurate, running-time efficient, space efficient. By accurate, we emphasize that the similarity measurement should capture the intrinsic similarity of objects. By running-time

efficient, it is well-known that operations on graphs, such as subgraph isomorphism, are NP-complete problems [6] which require us to design efficient algorithms to avoid exhaustive search as much as possible. By space efficient, the index structure should not add a significant storage overhead to graph databases.

The most straightforward approach for similarity measurement is to embed a graph in a high dimensional Euclidian space, known as the *feature space*, and use spatial indexing techniques for similarity search. Current feature extraction methods operate in two different ways: (i) enumeration of substructures in each graph separately (e.g. generating a set of random walks from a graph) [15], and (ii) enumeration of substructures from a set of graphs (e.g. mining frequent patterns) [30, 4, 32, 29]. Though widely used, there are several limitations of adapting the feature extraction and feature indexing strategy for similarity search. First, the feature extraction process is computational intensive, especially for mining large graph databases. Second, feature extraction may produce many features that occupy a large amount of memory. Different feature selection methods have been devised to identify "discriminative" features [30]. However features that are efficient for database search may not be equally good for similarity search and a trade-off is needed.

Here we explore a new way of graph similarity search where similarity is defined using graph kernel functions. Rather than extracting features explicitly, a *kernel function* maps data objects to a high-dimensional functional space and measure the similarity of objects by computing the inner product of the objects in the functional space. The advantage of kernel function based similarity measurement is that kernel function usually have high statistical power, i.e. affording high classification accuracy. The major difficulty of applying kernel function for database search is that (i) kernel functions for graphs are expensive to compute and (ii) there is no clear way to index the kernel function computation over a large graph database.

Our approach, called G-hash, aims to devise a kernel function that can be efficiently computed over a large graph database. In our model, graphs are reduced to point sets that are compared directly via a kernel function. Typically such an approach would lose a great deal of information in the rich graph structure, but we avoid this by compressing much of the topological information into feature vectors describing each graph vertex. This approach provides a compact graph representation that is information rich, yet easy to compare. We then hash graph objects using the compressed set representation. The hash keys are based these sets and hence similar objects in the hash table are positioned in the same or nearby cells. Once we have hashed graphs in a database into the table, we can find all similar nodes and then calculate the distances between the query graph and the graphs in the database based on them and kernel function to obtain the k-NNs of the query graph.

In summary, our contributions in this papers are:

- Devised a graph kernel function and related index structure for fast graph similarity search
- Our index has linear time to compute (in terms of total number of nodes in a graph database) and may be constructed on-line with dynamic insertion and deletion
- We have proved that the new graph kernel function and its related index structure achieved a better trade-off between capturing the intrinsic similarity of graphs and fast computation for large graph databases.

This paper is organized as follows. We review related work in the areas of hashing, indexing, and kernels for graphs in section 2. Next, we formally define graphs and graph similarity search in section 3. We discuss the details of our index structure and kernel function in section 4. Finally we present a comprehensive experimental study using our methods and competing methods, and conclude with a few remarks on the study and future work.

2. RELATED WORK

In this section we discuss related work, starting from indexing in graph databases in general, including subgraph search, approximate subgraph search, and graph similarity search, and move to graph kernel functions.

2.1 Subgraph Search

Many of the recent methods for subgraph search adopt a similar framework, decomposing graphs into a set of smaller pieces, treating each piece as a feature, and building a feature-based index structure for subgraph query. Methods that belong to this category include *GraphGrep* [21], *gIndex* [30], *FG-Index* [4], *Tree+Delta* [32], and *GDIndex* [29].

The simplest type of feature in use for graph indexing is walks (including path as special cases) as pioneered in *GraphGrep* [21]. Path are easy to retrieve and easy to work with. The simplicity of paths limit their expressiveness. For example, using paths, we could not distinguish the topology of a ring and a chain (where all paths from the two graphs are paths with different sizes).

Recognizing the limitation of paths, *gIndex* [30] and *FG-Index* [4] build indices using general subgraphs, which can easily distinguish between paths and cycles in graphs and hence are more powerful. The limitation of subgraph features is that subgraph enumeration and matching are computational intensive procedures. In order to manage these obstacles, these methods extract only *frequent* subgraph features. Similar methods, *Tree+Delta* [32] and *TreePI* [31] use frequent tree patterns (as well as some discriminative graph features) instead of frequent subgraph patterns.

The method *GDIndex* [29] also uses subgraphs as the basic index feature, but does not restrict itself to *frequent* subgraph features. In addition to a subgraph-based index, this method incorporates a hash table of subgraphs for fast isomorphism lookup. While this method's focus is subgraph search, it supports similarity search as well.

2.2 Approximate Subgraph Search

Besides strict subgraph search, some methods relax the isomorphism matching constraint and allow partial or approximate matches. This is a relatively new direction, and hence not many methods currently address the problem. One such method, *SAGA* [24], was designed for biological pathway analysis. First, it builds an index based on graph fragments. It then uses a graph distance measure to allow for vertex mismatches and gaps when when matching candidate graphs to a query.

Another method *gApprox* [3] is similar to the *gIndex* [30] method, in spirit and name, as well as authors. This approach seeks to mine frequent *approximate* patterns from a

graph database and use these for indexing. They also explore the notion of *approximately frequent*.

The method TALE [25] is also designed for approximate graph matching. It's focus, however, is on handling large graphs with thousands of vertices and edges.

2.3 Graph Similarity Search

There are three commonly used ways to measure graph similarity. The first is *edit distance*. That is, given a set of operations on graph vertices and edges (such as insertion, deletion, relabeling), how many many such operations are required to transform graph G into another graph, G'. We can parameterize the method by assigning different costs to different operations and summing over the total cost of all operations. Edit distance is an intuitively attractive approach to graph similarity, but unfortunately in practice it is costly to compute (NP-hard). *C-Tree* [12] is a widely used graph indexing scheme that also does not use graph pieces as features. Instead, it organizes database graphs in tree based structure, where interior nodes are *graph closures*, and leaf nodes are database graphs. Importantly, *C-Tree* also supports similarity queries where the previous two methods, *GraphGrep* and *gIndex*, do not.

One method, *GString* [13] is a subgraph similarity query method and uses graph fragments as features as well. The approach is somewhat different than the previous two feature-based subgraph search methods. Complex graphs are first reduced into connected graphs of fewer nodes, each of which represents a specific fragment. Canonical node numbering is used to create a string representation for each graph in a database. An index that supports similarity search is then constructed in the form of a suffix tree. This method combines the expressive power of subgraphs and simplified graphs with the speed of string querying and matching.

In addition, maximal common subgraph [2] and graph *alignment* [9, 27] are used to measure graph similarity. Unfortunately, there is no easy way to index both measurements for large graph databases.

2.4 Graph Kernel Functions

Several graph kernel functions have been studied. The pioneering work was done by Haussler in his work on *R-convolution* kernel, providing a framework for many current graph kernel functions to follow [11]. The R-convolution kernel is based on the notion of decomposing a discrete structure (e.g. a graph) into a set of component objects (e.g. subgraphs). We can define many such decompositions, as well as kernels between pairs of component objects. The R-convolution framework ensures that no matter the choice of decompositions or component kernels, the result is always a symmetric, positive semi-definite function, or a kernel function between compound objects. This key insight allows the problem of finding kernel functions for discrete structures to be reduced to those of finding decompositions and kernel functions between component objects. The R-convolution kernel can be extended to allow weighting of the kernel between various components, via the Weighted Decomposition Kernel [18].

Recent progresses of graph kernel functions could be roughly divided into two categories. The first group of kernel functions consider all possible components in a graph (e.g. all possible paths) and hence measure the global similarity of two graphs. These include product graph kernels [10], random walk based kernels [15], and kernels based on shortest paths between pair of nodes [1]. The second group of kernel functions try to capture the local similarity of two graphs by specifying a (finite) subset of components and counting the shared components only according to the finite subset of components. These include a large class of graph kernels called spectrum kernels [8] and recently frequent subgraph kernels [23]. The most efficient kernel function that we notice is proposed by Vishwanathan [28] for global similarity measurement with complexity $O(n^3)$ where n is the maximal number of nodes in graphs. Different from global similarity measure, local similarity capturing is known to be expensive since subcomponent matching (e.g. subgraph isomorphism) is an NP-hard operation.

We adopt a recently develop graph wavelet matching kernel and make it scalable for large databases.

3. BACKGROUND

Before we proceed to discuss the algorithmic details, we present some general background regarding a computational analysis of graphs which includes (i) graph kernel functions, and (ii) graph wavelet analysis.

3.1 Graphs

A *labeled graph* G is described by a finite set of nodes V and a finite set of edges $E \subset V \times V$. In most applications, a graph is labeled, where labels draw from a label set λ. A labeling function $\lambda : V \cup E \rightarrow \Sigma$ assigns labels to nodes and edges. In *node-labeled graphs*, labels are assigned to nodes only and in *edge-labeled graphs*, labels are assigned to edges only. In *fully-labeled graphs*, labels are assigned to nodes and edges. We may use a special symbol to represent missing labels. If we do that, node-labeled graphs, edge-labeled graphs, and graphs without labels are special cases of fully-labeled graphs. Without loss of generality, we deal with fully-labeled graphs only in this paper. For the label set Σ we do not assume any structure of Σ now; it may be a field, a vector space, or simply a set.

Following convention, we denote a graph as a quadruple $G = (V, E, \Sigma, \lambda)$ where V, E, Σ, λ are explained before. A graph $G = (V, E, \Sigma, \lambda)$ is a *subgraph* of another graph $G' = (V', E', \Sigma', \lambda')$, denoted by $G \subseteq G'$, if there exists a 1-1 mapping $f : V \rightarrow V'$ such that

- for all $v \in V, \lambda(v) = \lambda'(f(v))$
- for all $(u, v) \in E, (f(u), f(v)) \in E'$
- for all $(u, v) \in E, \lambda(u, v) = \lambda'(f(u), f(v))$

In other words, a graph is a subgraph of another graph if it preserve the node labels, edge relations, and edge labels.

A *walk* of a graph is a list of node $v_1, v_2, \ldots, v_n$ such that v_i and v_{i+1} is connected for all $i \in [1, n-1]$. A *path* is a walk which contains no repeated nodes, i.e. for all $i \neq j$ we have $v_i \neq v_j$

3.2 Reproducing Kernel Hilbert Space

Kernel functions are powerful computational tools to analyze large volumes of graph data [11]. The advantage of kernel functions is due to their capability to map a set of data to a high dimensional Hilbert space without explicitly computing the coordinates of the data. This is done through a special function called a *kernel* function.

A binary function $K : X \times X \rightarrow \mathbb{R}$ is a *positive semi-definite* function if

$$\sum_{i,j=1}^{m} c_i c_j K(x_i, x_j) \geq 0 \quad (1)$$

for any $m \in \mathbb{N}$, any selection of samples $x_i \in X$ ($i = [1, n]$), and any set of coefficients $c_i \in \mathbb{R}$ ($i = [1, n]$). In addition, a binary function is *symmetric* if $K(x, y) = K(y, x)$ for all $x, y \in X$. A symmetric, positive semi-definite function ensures the existence of a Hilbert space $\mathcal{H}$ and a map $\Phi : X \rightarrow \mathcal{H}$ such that

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \quad (2)$$

for all $x, x' \in X$. $\langle x, y \rangle$ denotes an inner product between two objects x and y. The result is known as the Mercer's theorem and a symmetric, positive semi-definite function is also known as a Mercer kernel function [19], or *kernel* function for simplicity.

By projecting the data space to a Hilbert space, kernel functions provide a uniformed analytical environment for various data types including graphs, regardless of the fact that the original data space may not look like a vector space at all. This strategy is known as the "kernel trick" and it has been applied to various data analysis tasks including classification [26], regression [5] and feature extraction through principle component analysis [20], among others.

3.3 Graph Wavelets Analysis

Wavelet functions are commonly used as a means for decomposing and representing a function or signal as its constituent parts, across various resolutions or scales. Wavelets are usually applied to numerically valued data such as communication signals or mathematical functions, as well as to some regularly structured numeric data such as matrices and images. Graphs, however, are arbitrarily structured and may represent innumerable relationships and topologies between data elements. Recent work has established the successful application of wavelet functions to graphs for multiresolution analysis. Two examples of wavelet functions are the *Haar* and the *Mexican hat*.

Crovella et al. [7] have developed a multi-scale method for network traffic data analysis. For this application, they are attempting to determine the scale at which certain traffic phenomena occur. They represent traffic networks as graphs labeled with some measurement such as bytes carried per unit time.

Maggioni et al. [17] demonstrate a general-purpose biorthogonal wavelet for graph analysis. In their method, they use the dyadic powers of an diffusion operator to induce a multiresolution analysis. While their method applies to a large class of spaces, such as manifolds and graphs, the applicability of their method to attributed chemical structures is not clear. The major technical difficulty is how to incorporate node labels in a multiresolution analysis.

4. FAST GRAPH SIMILARITY SEARCH WITH HASH FUNCTIONS

As discussed above, current graph query methods provide fast query time but not good similarity measurements. Kernel functions can provide better similarity measurement but the kernel matrix calculation is time-consuming so it is hard to build index structure by using them directly. To address this problem, we propose a new method, G-hash. Current methods usually focus on either accuracy or speed. Our proposed method defines similarity based on Wavelet Graph matching kernels (WA) and uses hash table as index structure to speed up graph similarity query. Below we first give an introduction to WA method.

4.1 Introduction to Wavelet Graph matching kernels

The idea behind WA method is to first convert the graph into sets by compressing property information in the neighborhood around each vertex, and then apply non-recursive alignment kernel to compute similarity between graphs. This method contains two important concepts: *h-hop neighborhood* and *discrete wavelet functions*. The *h-hop neighborhood*of one node v, denoted by $N_h(v)$, refers to a set of nodes which are h hops away from the node v according to the shortest path. *Discrete wavelet functions* refer to the defined wavelet functions, shown in equation 3, applying to h-hop neighborhood.

$$\psi_{j,k} = \frac{1}{h+1} \int_{j/(k+1)}^{(j+1)/(k+1)} \varphi(x) dx \quad (3)$$

where $\varphi(x)$ is *Haar* or *Mexican Hat* wavelet function and h is the hth partition after $\varphi(x)$ is partitioned into h+1 intervals on the domain [0,1) and j is between 0 and h.

Based on the above two definitions, we can now apply wavelet analysis to graphs. Wavelet functions are used to create a measurement summarizing the local topology of a node. Equation 4 shows such a wavelet measurement, denoted by $\Gamma_h(v)$, for a node v in a graph G.

$$\Gamma_h(v) = C_{h,v} \times \sum_{j=0}^{k} \psi_{j,k} \times \bar{f}_j(v) \quad (4)$$

where $C_{h,v}$ is a normalization factor with

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

and $\bar{f}_j(v)$ is the average feature vector value of atoms that are at most j-hop away from v with

$$\bar{f}_j(v) = \frac{1}{|N_j(v)|} \sum_{u \in N_j(v)} f_u. \quad (6)$$

and f_u denotes the feature vector value of the node v. Such feature vector value can be one of the following four types:nominal, ordinal, internal and ratio. For ratio and internal node features, we directly apply the above wavelet analysis to get local features. For nominal and ordinal node features, we could first build a histogram and then use wavelet analysis to extract local features. After the node v is analyzed, a list of vectors $\Gamma^h(v) = \{\Gamma_1(v), \Gamma_2(v), \cdots, \Gamma_h(v)\}$, called wavelet measurement matrix, can be obtained. In this way, a graph can be decomposed into a set of node vectors. Since the wavelet has strongly positive and strongly negative regions, these wavelet-compressed properties represent a comparison between the local and distant vertex

neighborhood. Structural information of a graph has therefore been compressed into the vertex properties through wavelets. Hence, we can now ignore the topology and focus on matching vertices. The kernel function is defined on these sets. Given two graphs G and G' for example, the graph matching kernel is

$$k_m(G, G') = \sum_{(u,v) \in V(G) \times V(G')} K\left(\Gamma^h(u), \Gamma^h(v)\right), \quad (7)$$

$$K(X, Y) = e^{\frac{-\|X-Y\|_2^2}{2}}. \quad (8)$$

The WA methods shows a good definition of similarity between graphs through kernel functions, as validated in the experimental part [22]. One issue, however, is that the overall time complexity of the wavelet-matching kernel is $O(m^2)$, and that of the kernel matrix is $O(n^2 \times m^2)$, where n is the size of the database and m is the average node number of all graphs. When the size of the database increases, the kernel matrix calculation time grows very quickly.

4.2 Fast graph similarity search with hash functions

Following the idea of using a function to map each node in a graph to a feature space, we may design a kernel function for fast similarity search. Specifically, we have the following two observations.

- When the node vector of the node u in the graph G is dramatically different from that of the node v in the graph G', the RBF kernel value between the node u and node v is small and has little contribution to the graph kernel. So if we just count those pairs of nodes which have similar node vectors, the kernel matrix will reflect the similar similarity measurement between two graphs to that of WA method and the time will be saved.
- Similar objects in the hash table are positioned closer if the hash keys are based on the node vectors. So the hash table can help us to find similar node pairs rapidly. In addition, if all graphs in the database are hashed into the table, one cell may contain many similar nodes which belong to different graphs. Since these nodes are all similar, only one time RBF kernel calculation using two nodes of them is enough. Node overlay provides another chance to save time.

Based on the above two observations, we introduce our method, called hash table based k-NNs query (G-hash). G-hash is based on WA method to provide an accurate similarity measurement and use hashing to improve time complexity. The entire process is described as follows.

4.2.1 Index construction

First, we need to decompose all graphs in the library database into node vectors by using wavelet transformation same as that in the method of WA. Since the WA method is relatively insensitive to small perturbation of the hop distance parameter and the use of different wavelet functions makes little difference [22], h is picked as 2 and the *Haar* wavelet is used for simplification.

After node vectors are obtained, the hash table will be built using the graphs in the database. At this time, a hash function needs to be constructed to make sure that similar nodes can be hashed into the same cell. That means the hash keys should be associated with node vectors. Since we obtains the node vectors according to the node label and the neighboring information, we can construct hash keys in the same idea. We discretize each feature in the node vector to an integer. We encode a node label directly as a n bit-string with all zeros except a single 1 which indicate the label. Other features are rounded to the nearest integer. After the node vector is changed to a list of integer numbers, we then convert a node vector to a string by represent each integer number using its binary format and concatenate these obtained bit strings delimited by underscore. Such string is the hash key of the corresponding node. Take the graph P shown in figure 1 for example.

EXAMPLE 4.1. *We pick node labels and the number of neighboring nodes with different labels as node features. So there are a total of four node features. The node features for this graph just belong to one types of value: nominal. To get the node vectors, we first obtain histogram of node features. The histogram is shown in figure 1. Then we use wavelet function to extract local features. Take node P_3 for example, the local obtained feature vector after using wavelet analysis for h=0 is $[b, 2, 0, 1]$. The sample hash table is shown in figure 1.*

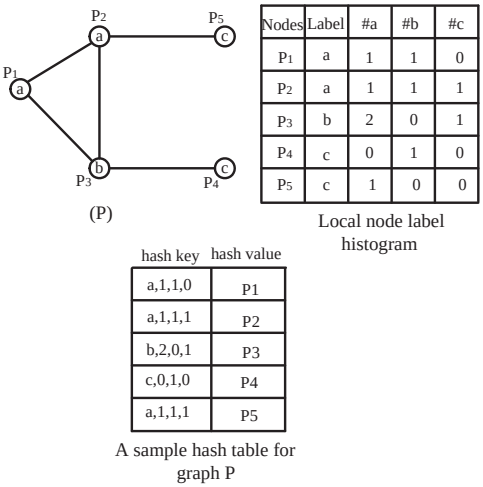


Figure 1: Example graph.

Notice: In this framework, we assume no information for query graphs when we build the hash table for indexing.

4.2.2 k-NNs Query Processing

To obtain the k-NNs of a given query graph, we need to calculate the distance between it and all graphs in the database. Here the distance is defined using the kernel function between these two graphs.

$$\begin{aligned} d(G, G') &= \sqrt{\|\phi(G) - \phi(G')\|_2^2} \\ &= \sqrt{\langle \phi(G) - \phi(G'), \phi(G) - \phi(G') \rangle} \\ &= \sqrt{\langle \phi(G), \phi(G) \rangle + \langle \phi(G'), \phi(G') \rangle - 2 \langle \phi(G), \phi(G') \rangle} \\ &= \sqrt{k_m(G, G) + k_m(G', G') - 2k_m(G, G')}. \end{aligned} \quad (9)$$

where $k_m(G,G)$ is the kernel function between the graph G and itself, $k_m(G',G')$ is the kernel function between the graph G' and itself, and $k_m(G,G')$ is the kernel function between the graph G and G'). In the following section, we will discuss how to calculate all of them.

Though hashing the nodes of the query graph into the hash table, we can get the kernel function

$$k_m(G,G') = \sum_{v\in G', u\in simi(v)} K\left(\Gamma^h(u), \Gamma^h(v)\right), \qquad (10)$$

Where $simi(v)$ are a set containing the nodes in graph G hashed to the same cell as the node v does. We use the following decoding to get the graph number containing these nodes and the node number.

Clearly, the similarity of two graphs is determined only by similar node pairs instead of all node pairs, which will save computational time. Since similar nodes also may be hashed into the neighboring cells, to increase the accuracy, we also count the nodes in neighboring cells when the size of graph is a larger (e.g. greater than 40).

Since only similar nodes are involved into the kernel calculation, $K\left(\Gamma^h(u), \Gamma^h(v)\right) \approx 1$ if RBF kernel is used. So the equation 10 can be written into

$$k_m(G,G') \approx \sum_{v\in G', u\in simi(v)} 1 = \sum_{v\in G'} |simi(v)|, \qquad (11)$$

where $|simi(v)|$ is the number of nodes contained in $simi(v)$. That means that we only need to count the number of similar nodes, belonging to the graph G, of each node in the query graph G' and sum them to get the kernel. Similarly, we can calculate the kernel between each graph and itself.

After the above calculations, we obtain a distance vector with each value corresponding to the distance between the query graph and a graph in the database. Through sorting this distance vector, we can obtain the k-NNs of this given query graph.

4.2.3 Dynamic insertion and deletion

To insert a new graph into the database, we hash all nodes of the new graph into the hash table. After insertion, only those cells associated with these nodes contain more nodes and all other cells has no changes. In addition, since the new graph has a limited number of nodes, insertion operations involve less time and the size of index little. Deletion operations are similar to insertion. To delete a graph from the database, we calculate the key corresponding to each node in this graph and then delete each node from the cell containing them.

5. EXPERIMENTAL STUDY

We have performed a comprehensive evaluation of our method by evaluating its effectiveness (in classification), efficiency, and scalability. We will apply our methods on chemical databases. For chemical compounds, the node features include numeric features and boolean atom features. Numeric features include element type, atom partial charge, atom electron affinity, atom free electrons count and atom heavy valence,etc. Boolean atom features include atom in acceptor, atom in terminal carbon, atom in ring,atom is negative, atom is axial,etc. Here, we just use a single of atomical feature: element type.

We have compared our methods with the Wavelet Alignment Kernel [22], C-tree [12],GraphGrep [21] and gIndex [30] as performance benchmarks. Our method, WA method, GrapGrep and gIndex are developed in C++ and compiled using g++. C-tree was developed in Java and compiled using Sun JDK1.5.0. All experiments were done on an Intel Xeon EM64T 3.2GHz, 4G memory cluster running Linux.

The parameters for WA, G-hash, C-tree, GraphGrep and gIndex are set in the following way. we set $h = 2$ and use *haar* wavelet function for WA and G-hash. For C-tree, we choose the default values, namely, setting the minimum number of child node $m = 20$, the maximum number $M = 2m - 1$ and the NBM method [12] is used for graph mapping. For GraphGrep and gIndex, we use default parameters.

5.1 Data sets

We chose a number data sets for our experiments. The first five data sets are established data taken from Jorisson/Gilson Data Sets[14]. The next six data sets are manually extracted from BindingDB data sets [16]. The last one is NCI/NIH AIDS Antiviral Screen data set (`http://dtp.nci.nih.gov/.`). Table 1 shows these data sets and their statistical information.

5.1.1 Jorissen sets

The Jorissen data sets contain information about chemical-protein binding activity. The target values are drug's binding affinity to a particular protein. There are five proteins for which 100 chemical structures are selected with 50 chemical structures clearly bind to the protein(called "active" ones) and the other 50 ones similar to the active ones but clearly not bind to the target protein. See [14] for the further details.

5.1.2 BindingDB sets

The BindingDB database contains data for proteins and chemicals that bind to the proteins. We manually selected 6 proteins with a wide range of known interacting chemicals (ranging from tens to several hundreds). For the purpose of classification, we convert the real-valued binding activity measurements to binary class labels. This is accomplished by dividing the data set into two equal parts according to the median activity reading (we also deleted compounds whose activity value is equal to zero).

5.1.3 NCI/NIH AIDS Antiviral Screen data set

NCI/NIH AIDS Antiviral Screen data set contains 42,390 chemical compounds retrieved from DTP's Drug Information System. There is a total 63 types of atoms in this data set; the most frequent ones are C, O, N, and S. The data set contains three types of bonds: single-bond, double-bond and aromatic-bond. We selected all chemicals to build our graph database and randomly sampled 1000 chemicals as the query data set.

5.2 Similarity Measurement Evaluation with Classification

we compared classification accuracy using k-NN classifier on Jorissen sets and BindingDB sets with difference similarity measurement. For the WA method, we use wavelet matching kernel function to obtain kernel matrix, and then calculate distance matrix to obtain k nearest neighbors. For

Table 1: Data sets statistics.#S:total number of compounds, #P:number of positive compounds,#N:number of negative compounds,#Node: average number of nodes, #Edge: average number of edges.

data set	#S	#P	#N,	#Node	#Edge
PDE5	100	50	50	44.7	47.2
CDK2	100	50	50	38.4	40.6
COX2	100	50	50	37.7	39.6
FXa	100	50	50	45.75	48.03
AIA	100	50	50	48.33	50.61
AChE	183	94	89	29.1	32.0
ALF	151	61	60	23.8	25.2
EGF-R	497	250	247	24.6	27.1
HIV-P	267	135	132	43.0	46.2
HSP90	109	55	54	29.84	32.44
MAPK	336	168	168	28.0	31.1
HIV-RT	482	241	241	22.18	24.39

G-hash, we compute graph kernel according to our algorithmic study section and then calculate the k nearest neighbors. For C-tree, we directly retrieve the nearest neighbor. We use standard 5-fold cross validation to obtain classification accuracy, which is defined as $(TP+TN)/S$ where TP stands for true positive, TN stands for true negative and S is the total number of testing samples. We report the average accuracy. In our experiments, we set $k = 5$.

The accuracy results are shown in figure 2. The accuracy, precision, and recall statistical information is shown in Table 2, 3 and 4. From figure 2, we know that G-hash outperforms C-tree on all twelve data sets, with at least 8% improvement on all of them. The average accuracy difference between G-hash and C-tree is about 13%. WA method outperforms G-hash, the average difference between them is about 2% because, most likely because we adopt some simplifications on distance matrix calculation. From what is discussed above, it is clear that kernel based similarity measurement is better than edit distance based similarity measurement. Since the accuracy of k-NN classifier is associated with the value of k, we also study the accuracy with respect to the value of k on these data sets to test whether the parameter k has any effect on accuracy performance comparison. Results show that accuracy performance comparison is insensitive to the parameter k.

Table 2: Accuracy results statistical information for G-hash, C-tree and WA on all data sets.

method	G-hash	C-tree	WA
average	64.55	51.64	66.23
derivation	2.68	5.95	4.83

5.3 Scalability

5.3.1 Index Construction

In this section, we apply G-hash, WA [22], C-tree [12], GraphGrep [21] and gIndex [30] on NCI/NIH AIDS Antiviral Screen data set.

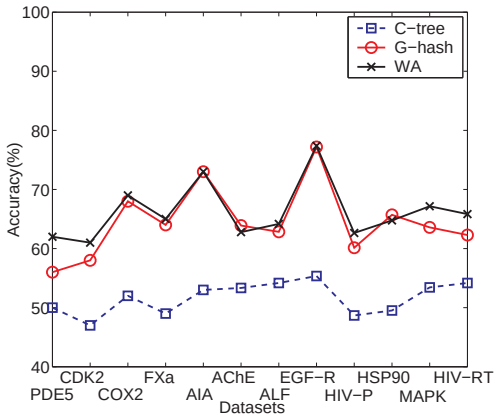


Figure 2: Comparison of averaged classification accuracy over cross validation trials.

Table 3: Average Precision for different data sets. Asterisk ($*$) denotes the best precision for the data sets among WA, G-hash and C-tree methods.

dataset	WA	G-hash	C-tree
PDE5	83.16*	75.78	31.2
CDK2	73.81*	67.42	51.82
COX2	75.88*	66.98	54.85
FXa	95.78*	91.19	29.36
AIA	98.93*	98.33	36
AChE	66.46	73.59*	62.63
ALF	72.14*	69.82	32.59
EGF-R	72.75	80*	55.41
HIV-P	56.9	64.64*	40.81
HSP90	58.19	73.63*	48.72
MAPK	66.31*	66.21	53.25
HIV-RT	69.38*	61.87	54.11

We compare index size and average index construction time for different methods. Towards that end, we sampled different number of graphs ranging from 10,000 to 40,000. Figure 3 shows the index construction time in milliseconds with respect to the size of database for G-hash, C-tree, GraphGrep and gIndex. The construction time for G-hash is much lower than those for other three methods because of the adoption of a hash table. In addition, when the data set size increases, the construction time for C-tree, GraphGrep and gIndex grows faster than that for G-hash since the construction of C-tree, GraphGrep and gIndex involve relatively complicated index structure. So G-hash outperforms C-tree, GraphGrep and gIndex on index construction time.

Figure 4 shows index size with respect to database size. The index size of G-hash shows a steady growth with increasing database size while that of C-tree increases sharply since C-tree need to save the whole tree structure while G-hash just need to save the hash table.

5.3.2 Query Processing Time

Figure 5 shows the query time in milliseconds with respect to the size of database. When the size of database increases,

Table 4: Average recall for different data sets. Asterisk (∗) denotes the best recall for the data sets among WA, G-hash and C-tree methods.

dataset	WA	G-hash	C-tree
PDE5	58.06*	56.2	46.93
CDK2	55.87*	53.54	46.7
COX2	63.57	68.06*	51.46
FXa	58.23	62.41*	42.06
AIA	64.81	66.27	55.33
AChE	63.63*	62.82	44.15
ALF	61.25	66.16*	53.83
EGF-R	79.64*	77.51	55.81
HIV-P	63.4*	61.96	47.62
HSP90	63.4*	61.96	47.62
MAPK	70.52	73.6*	72.16
HIV-RT	67.78*	66.83	56.78

G-hash scales better than C-tree. There is no direct way that we could compare G-hash and subgraph indexing methods such as G-index and Graphgrep since G-hash search for similar graph and G-index (and Graphgrep) searches for the occurrences of a subgraph in a database.

In the following, we sketch one way to use subgraph indexing methods for similarity search. This method contains three steps: (i) randomly sample subgraphs from a query, (ii) use those subgraphs as features and compute the occurrences of the subgraphs in graph databases, and (iii) search for nearest neighbors in the obtained feature space. Clearly the overall query processing time depends on (i) how many subgraphs we use and (ii) how fast we can identify the occurrences of the subgraphs in a graph database. We estimate the lower bound of the overall query processing time by randomly sampling a SINGLE (one) subgraph from each of the 1000 querying graph and use subgraph indexing method to search for the occurrence of the subgraph. We record the average query processing time for each query. This query processing time is clearly the lower bound since we use only one subgraph from the query graph.

Figure 5 shows the experimental results of comparing C-tree, GraphGrep and gIndex. When the size of database is 40,000, the query time for C-tree, Graphgrep and gIndex are nearly 8 times,10 times and 100 times as that for G-hash respectively.

Finally, we compared C-tree and G-hash with varying k values for k-NN search. the results are shown in Figure 6. The query time of C-tree increases with the increasing k and the running time of G-hash is insensitive to the number of k.

6. CONCLUSIONS AND FUTURE WORKS

Graphs are a kind of general structural data have been widely applied in many fields such as cheminformatics and bioinformatics, among others. A lot of significant researchers have been attracted to current data management and mining technique. Proposing an efficient similarity graph query method is a significant challenge since most existed methods focus on speed and provide poor accuracy. In order to address this problem, we have presented a new graph query method, G-hash. Through our experimental study, we have

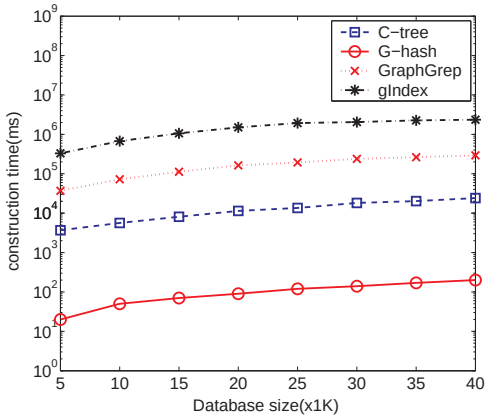


Figure 3: Index construction time for NCI/NIH AIDS data set.

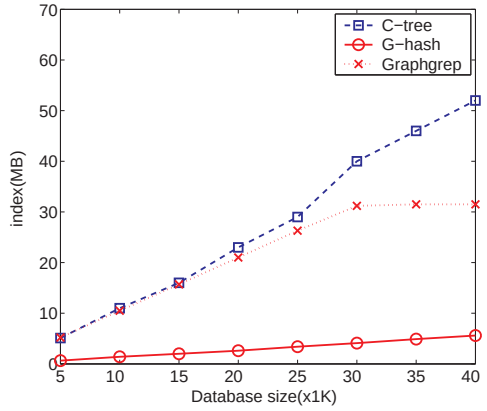


Figure 4: Index size for NCI/NIH AIDS data set.

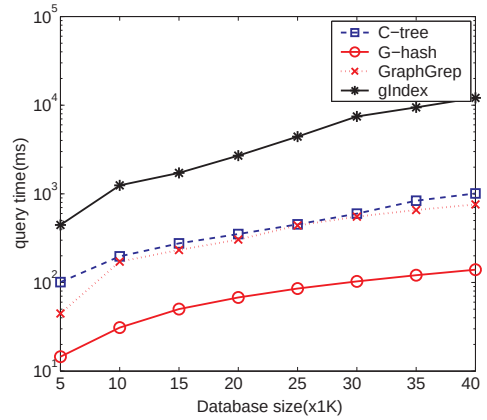


Figure 5: Query time for NCI/NIH AIDS data set.

shown that compared to C-tree [12], G-hash provides about a 13% improvement to accuracy. The query time for G-hash is much less than that for C-tree [12], GraphGrep[21] and gIndex[30] especially when the database size becomes larger. In addition, G-hash shows a better scalability on index con-

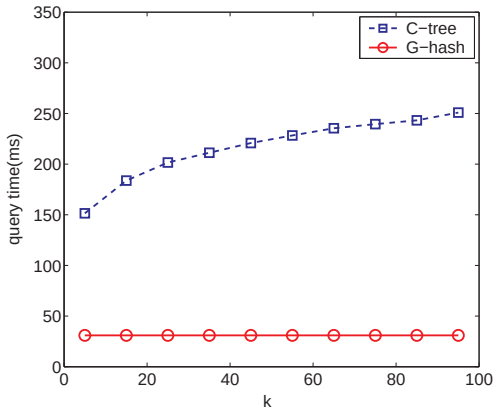


Figure 6: Query time with respect to different k for k-NN query.

struction time and efficiency in space usage. This means G-hash can support large graph database search.

Acknowledgments

We thank H. He and A. K. Singh for sharing the code of C-tree,D. Shasha, J.T.L. Wang and R. Giugno for sharing the code of GraphGre, X. Yan, P. S. Yu and J. Han for sharing the code of GIndex. This work was supported by the KU Center of Excellence for Chemical Methodology and Library Development (NIH/NIGM award #P50 GM069663)and an NIH grant #R01 GM868665.

7. REFERENCES

[1] K. Borgwardt and H. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the International Conference on Data Mining (ICDM)*, 2005.

[2] Y. Cao, T. Jiang, and T. Girke. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics*, 24(13):i366–74, 2008.

[3] C. Chen, X. Yan, F. Zhu, and r. Jiawei Han. gapprox: Mining frequent approximate patterns from a massive network. In *Proc. 2007 Int. Conf. on Data Mining (ICDM'07)*, 2007.

[4] H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, 2007.

[5] R. Collobert and S. Bengio. Svmtorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 21, 2001.

[6] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, 1971.

[7] M. Crovella and E. Kolaczyk. Graph wavelets for spatial traffic analysis. *Infocom*, 3:1848–1857, 2003.

[8] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 2005.

[9] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the 22nd international conference on Machine learning*, 2005.

[10] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Sixteenth Annual Conference on Computational Learning Theory and Seventh Kernel Workshop*, 2003.

[11] D. Haussler. Convolution kernels on discrete structures. *Technical Report UCSC-CRL099-10, Computer Science Department, UC Santa Cruz*, 1999.

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

[13] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. Gstring: A novel approach for efficient search in graph databases. In *Proc. International Conference on Data Engineering'07 (ICDE)*, 2007.

[14] R. Jorissen and M. Gilson. Virtual screening of molecular databases using a support vector machine. *J. Chem. Inf. Model.*, 45(3):549–561, 2005.

[15] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[16] T. Liu, Y. Lin, X. Wen, R. N. Jorrisen, and M. Gilson. Bindingdb: a web-accessible database of experimentally determined protein-ligand binding affinities. *Nucleic Acids Research*, 35:D198–D201, 2007.

[17] M. Maggioni, J. B. Jr, R. Coifman, and A. Szlam. Biorthogonal diffusion wavelets for multiscale representations on manifolds and graphs. In *Proc. SPIE Wavelet XI*, volume 5914, 2005.

[18] S. Menchetti, F. Costa, and P. Frasconi. Weighted decomposition kernels. In *Proceedings of the Twenty-second International Conference on Machine Learning*, pages 585–592, 2005.

[19] B. Schölkopf and A. J. Smola. *Learning with Kernels*. the MIT Press, 2002.

[20] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. *Advances in kernel methods: support vector learning*, pages 327–352, 1999.

[21] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proceeding of the ACM Symposium on Principles of Database Systems (PODS)*, 2002.

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

[23] A. Smalter, J. Huan, and G. Lushington. Structure-based pattern mining for chemical compound classification. In *Proceedings of the 6th Asia Pacific Bioinformatics Conference*, 2008.

[24] Y. Tian, R. C. McEachin, D. J. States, and J. M.Patel. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(20:232–239, 2007.

[25] Y. Tian and J. Patel. TALE: a tool for approximate large graph matching. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2008.

[26] V. Vapnik. *Statistical Learning Theory*. John Wiley, 1998.

[27] J.-P. Vert. The optimal assignment kernel is not positive definite. Technical Report HAL-00218278, French Center for Computational Biology, 2008.

[28] S. V. N. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In *In Advances in Neural Information Processing Systems*, 2006.

[29] D. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, 2007.

[30] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, 2004.

[31] S. Zhang, M. Hu, and J. Yang. treepi: A new grah indexing method. In *ICDE*, 2007.

[32] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta $\geq$ graph. In *VLDB*, 2007.

目　录

第一章　引言

Introduction ····· **5**

第二章　相关研究

Related Work ····· **9**

2.1　子图搜索

Subgraph Search ····· 9

2.2　模糊子图搜索

Approximate Subgraph Search ····· 10

2.3　图相似性搜索

Graph Similarity Search ····· 10

2.4　图核函数

Graph Kernel Functions ····· 11

第三章　背景知识

Background ····· **13**

3.1　图

Graphs ····· 13

3.2　再生核 Hilbert 空间

Reproducing Kernel Hilbert Space ····· 14

3.3　图小波分析

Graph Wavelets Analysis ····· 14

第四章　基于哈希的快速图相似性搜索

Fast Graph Similarity search with hash function ····· **17**

4.1　小波图匹配核简介

Introduction to Wavelet Graph matching kernels ····· 17

4.2 Hash 函数实现快速图相似性搜索

Fast graph similarity search with hash functions ····· 18

4.3 索引构建

Index construction ····· 19

4.4 *K-NNs* 查询过程

K-NNs Query Processing ····· 21

4.5 动态插入与删除

Dynamic insertion and deletion ····· 22

第五章 实验研究

Experimental Study ····· 23

5.1 数据集

Data Sets ····· 23

5.1.1 Jorissen 数据集 ····· 23

5.1.2 NCI/NIH AIDS 抗体数据库 ····· 24

5.2 利用分类做节点相似性度量

Similarity Measurement Evaluation with Classification ····· 24

5.3 伸缩性

Scalability ····· 25

5.3.1 索引构建时间

Index Construction ····· 25

5.3.2 查询时间

Query Processing Time ····· 25

第六章 总结与展望

Conclusions and Future Works ····· 27

第七章 致谢

Acknowledgements ····· 29

参考文献 ····· 31

摘　要

目前，集合，列表，树和图之类的结构化数据给数据管理这一基础领域提出了一个严峻的挑战。如何对图数据有效存储，索引，搜索都面临着一系列问题。随着图数据库的快速增多，图相似性搜索成为了一个热门的话题。图相似性搜索在多个领域都有应用，像化学结构，分子结构，传感器网络，XML 文档等等。

现在大多的图搜索算法都是为了解决精确搜索的，即找到一组包含精确查询图的图集合。因此，我们不能直接利用这些算法进行相似性搜索。在数据挖掘和机器学习领域有很多图核函数来判断图之间的固有相似度，但在图相似性搜索上却很罕见。因为尽管图核函数对于监督学习领域的准确预测和分类模型效果很好，但是用于图相似度查询则会存在两个关键问题:（i）计算复杂度非常高（ii）在图搜索上的应用复杂度也是非平凡的。

而本文中，我们打算利用图核函数进行相似性搜索。为此，我们提出了两个关键概念（i）一种新的图相似度度量方法（ii）一种新的图数据索引策略。我们以每个节点和其邻接节点特征为相似性度量依据，并利用哈希表来进行高效存储和快速查询。我们这种利用图核函数抓住图相似性的本质特征，并利用哈希表加速查询过程的新算法，我们称之为 G-Hash。本文中我们详细介绍了这种方法，并在大规模的化学结构数据库上进行了实验。结果表明 G-Hash 在 k 个最相近邻居问题上已经达到了业界的最高水平，更重要的是，我们这种新的相似性度量方法和索引结构比现有的算法（C-tree，gIndex，GraphGrep 等）具有更小的索引尺寸和更快的查询速度。

关键词: 图相似性搜索，图分类，哈希，图核，k-NNs 查询

第一章　引言

Introduction

目前，形如集合，序列，树和图等结构化数据给像高效存储，索引，部分查询 (如子图/超图搜索) 和相似性搜索之类的传统数据管理领域提出了一个巨大的挑战。随着图数据库的迅速发展，在图数据库中的**图相似性搜索**成为了一个日益重要的研究课题。图相似性搜索已经多个领域进行了应用，例如化学，分子信息学，传感器网络管理，社交网络管理，XML 文档等等。在化学与制药领域，每天会产生大量的化学分子结构数据。一旦一个新的化学结构被合成后，这个结构的特性就可能通过查询已知的分子结构，通过其特性来预测。大规模图数据中的相似性搜索让科研人员和工程师们可以对图精确建模，认识到图数据间的固有联系，减少大数据库的计算代价。

图查询需要分成两种：(i) 子图查询 (ii) 相似度查询。**子图查询**目的是确定一个包含着查询图的集合。**相似性查询**是根据差异距离来找出数据库中和查询相似的图，如 k-NNs 查询 (最相近 k 查询) 和范围查询。k-NNs 查询是为了得到最相近的 k 个图。范围查询，是为了得到差异距离小于预设值的所有图。在本文中，我们只研究在大规模图数据库中的 k-NNs 相似性搜索。

在图上进行相似性搜索是十分富有挑战的。我们认为一个理想相似性搜索设计应该达到以下三个相关的 (有时是相反的) 目标: 准确度高，时间短，空间小。为了准确度高，我们要求相似性度量方法应该抓住数据的固有相似度。众所周知，一些图上的操作，像子图同构，都是 NP 问题。所以为了时间短，我们需要设计一个高效的算法来尽量避免全图搜索。为了空间小，所以索引结构不能对图数据库开销太大。

许多简单的图相似性度量方法是将图转换为在高维欧几里得空间 (**特征空间**) 中的表示，然后利用空间上的索引技术来进行相似度搜索。现有的特征提取方法大多数都可以归为两类:(i) 每幅图的子结构单独计数（如从一幅图中生成随机的通路），(ii) 图集合中的子结构一起计数（如挖掘频繁子结构）。尽管相似性搜索已经被广泛应用了，但是在特征提取的适配性和特征的索引策略上仍有很多限制。首

先，图数据库（尤其是大规模的图数据库）的特征提取是需要大量运算量的。其次，特征提取过程可能产生很多特征值，需要占用大量内存空间。学术界曾提出过许多不同的特征选取方法来确定“有辨识度的”特征。但是，对于数据库搜索表现优异的特征选取方法在图相似性上不一定会很好，所以我们还需要自行去判断与权衡。

在本文中，我们探索了一种新的图相似性搜索方法。我们通过核函数来定义相似度。和通常不同的是，*核函数*没有直接提取特征值，而是将数据映射到一个高维的函数空间，并利用在这个函数空间中数据的内积来计算其相似度。基于核函数的图相似度测量方法的优势在于核函数统计学表现非常好，例如可以高精度的分类。但是将核函数用于数据库搜索也有些问题，主要的难点在于（i）核函数用在图上计算量很大（ii）目前为止，没有一个明确的方法来标引大规模图数据库上的核函数计算。

我们的方法称作 G-hash。我们致力于提出一种新的核函数来高效计算大规模图数据库上的特征。在我们的模型中，图被用核函数直接压缩成一个节点集。传统上，对于复杂的图结构，这样的压缩方法会丢失大量的信息。但是我们将大多数拓扑信息压缩到了每个节点的特征向量上来避免大量信息的丢失。这种方法提供了一个对于信息量丰富的图的简单表示，并且也很好进行比较。我们接着利用压缩集表示法将图节点哈希化。哈希的键值是基于这些表示集的，所以相似的节点会被哈希到同一个格子或者相邻的格子中。一旦我们将数据库中所有图哈希成一个表后，我们可以找到所有相似的节点，然后利用它们计算查询图和数据库中图的距离，并找出查询图的 k-NNs(k 个相近图)。

总体而言，我们这篇文章的主要贡献有:

- 提出了一个图核函数和用来进行快速图相似性搜索的索引结构
- 我们的索引只需线性时间去计算（对数据库中的总结点数而言），并且可以通过动态增删来在线构建。
- 我们证明了新的图核函数和相应的索引结构可以更好地在抓取固有图相似性和对于大数据库的快速计算上平衡。

本文组织如下，我们首先在第二章回顾了一些相关领域，如哈希，索引，图核函数等等。在第三章，我们正式定义了图和图相似性搜索。在第四章我们讨论了索

引结构和核函数的细节。最后我们对我们的算法进行了广泛的实验，并与现有算法进行对比，并得出了一些结论来指导我们的研究。

第二章　相关研究

Related Work

在本章中，我们讨论一些相关的研究。从基本的图数据库索引，子图搜索，模糊子图搜索和图相似性搜索到图核函数

2.1　子图搜索

Subgraph Search

现在许多的子图搜索都采用了一个相似的框架，先把图分解为一系列小片段，然后将其作为特征，并基于其建立基于特征的索引结构来进行子图查询。属于这类的方法有 *GraphGrep,gIndex,FG-Index,Tree+Delta* 和 *GDIndex*。

在图索引中，最简单的特征就是通路 (walk)。一些情况下，我们可以用路径作为特征，就像这个领域的先驱方法 *GraphGrep*. 路径很好检索特性，并且很好处理。但是，路径的简单性却制约着这种方法的性能。举例而言，即便两图的所有路径都是不同长度的，我们利用路径也不能区分出环和链的拓扑结构。

当意识到路径的局限性后，*gIndex* 和 *FG-Index* 便应运而生了。这两种方法利用子图结构来有效分辨路径和环。但是，基于子图结构的索引也有一些限制，就是子图的枚举和匹配都是运算量非常大的过程。为了克服这些障碍，这些方法采用只提取频繁子结构来作为索引特征。还有些相似的方法，像 *Tree+Delta* 和 *TreePi* 则用频繁树结构来代替频繁子图结构来克服这些障碍。

GDIndex 也采用子图结构算法作为基本索引特征，但是并不使其局限于频繁子图结构，除此之外，这个算法还采用了哈希表来加速图同构查询。尽管这个算法设计目的是为了子图查询，但也同样支持相似查询。

2.2　模糊子图搜索

Approximate Subgraph Search

除了精确图匹配，也有些其他算法放宽了同构的限时，允许部分匹配或者模糊匹配。这是一个相对较新的领域，因此并没有太多的算法针对这个问题。有一个针对这个问题的算法，*SAGA*，它被设计用来作为生物路径分析。首先，建立一个基于图片段的索引。当候选图与查询图失配时，我们用一个图距离度量方法来度量差异。

另一个算法叫做 *gApprox* 和 *gIndex* 相似。从思想到名字，包括作者都很相似。这种算法力求从数据库中挖掘频繁模糊模式，并以此作为索引特征。他们还提出一个概念，称为模糊频繁。

TALE 算法也是用来做模糊匹配的一种算法。但是它着重于处理有上千节点的大图。

2.3　图相似性搜索

Graph Similarity Search

通常，我们有很多方法去测量图之间的相似度。第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 (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费用，然后用费用总和当做距离来调整这个方法的准确度。虽然编辑距离是一种非常直观的图相似性测度方法，但是实际上我们难以计算它 (是个 NP-hard 问题).*C-Tree*[1] 是一种被广泛使用的图索引模型。它没有使用图的片段信息作为特征值，而是把数据图组织在一种内部节点是图闭包 *(graph closures)*, 叶节点是数据图的树形结构中。相比于前两种方法 *GraphGrep* 和 *gIndex*,*C-Tree* 的最大优势在于其支持相似性搜索，而前两个并不支持。

还有一种名为 *GString* 的子图相似性查询方法也和 *GraphGrep* 一样是用图片段作为特征值的。当然，这种方法与前面两种基于特征值的子图查询还是不同的。在这个方法中，首先我们分解复杂图为节点数较少的连通图，得到的这些连通图每个都是一个特定的图片段。随后，我们用一种标准的编号方式把数据库中的所有图都转化为一个个字符串。并用这些字符串构建一颗后缀树来支持相似性搜索。这种方法融合了子图的数据表达能力 (信息完整) 和用字符串匹配来查询图的速度

(速度快)。

另外, 最大公共子图 *(maximal common subgraph)*[2] 和图配对 *(graph alignment)*[3,4] 这两种方法也常被用来定义图相似度。虽然有这么多方法，但是不幸的是迄今为止我们仍没有一种简单的方法来索引或者度量大图数据库。

2.4　图核函数
Graph Kernel Functions

目前业界有很多图核函数，而开创性的一个图核函数是 Haussler 在他对 *R 卷积核 (R-convolution kernel)* 的研究中提出的。现在大多数图核函数都遵循它提出的这种框架。R 卷积核是基于把离散的结构 (如图) 分解成一系列的组成元素 (子图) 这个思想的。我们可以定义许多这样的分解, 就像组成结构中的核心一样。R 卷积核保证无论选择怎样的分解方式或者结构核心，都能得要一个对称半正定的函数，或者一个结构间的核函数。这个关键性质将寻找离散结构核函数的问题简化为寻找分解方式和结构间的核函数。R 卷积核可以通过加权分解核来允许组件结构间的加权核。

目前图核函数的研究可以大致分为两类：第一类是考虑图中的所有可能组成结构（例如所有路径），然后以此来计算两图之间的相似度。这一类的算法有 product graph kernel,random walk based kernel 和基于点对间最短路径的核。第二类核函数是尝试通过一组特殊的 (有限的) 结构来计算局部相似度，并且值在这有限的结构中统计共享的结构。这种方法包含一大类图核算法，叫做 spectrum 核，还有最近频繁子图核。我们发现最有效的核函数是 Vishwanathan 提出的用于全局相似性度量的方法。全局度量需要 $O(n^3)$ 的时间复杂度，n 是图中最大的节点数。众所周知，不同于全局相似度测量，局部相似性度量时间代价是非常昂贵的。因为子结构匹配（如子图同构）是一个 NP-hard 的问题。

我们采用一个最近提出的图小波匹配核，并将其扩展使其能在大数据库运行。

第三章　背景知识

Background

在我们详细介绍算法细节之前，让我们先了解一下关于图分析计算的所需的基本背景。这章包含 (i) 图核函数，(ii) 图小波分析两部分。

3.1　图

Graphs

一个**标号图**可以被一个有限的节点集合 V 和一个有限的边集合 $E \in V \times V$ 所描述。在大多数应用中，图都是有标号的。这些标号都是从一个标号集选取的，我们用一个标号函数 $\lambda : V \cup E \to \Sigma$ 来给各个节点和边分配标号。在**标号点图**中只有点有标号，同样，在**标号边图**中只有边有标号。在**全标号图**中，边点都有标号。如果用一种特殊的标号来表示未标号的点和边，那么标号点图和标号边图都可以被看做是全标号图的特殊形式。所以在此文中我们只考虑全标号图来简化问题而又不失一般性。对于标号集 Σ 我们并不指定具体结构，可以是一个字段, 一个向量，也可以是很简单的是个集合。以下我们约定，一幅图用一个四元组 $G = (V, E, \Sigma, \lambda)$ 表示,V, E, Σ, λ 都如上文所述。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**，用 $G \in G'$ 表示。1-1 映射可以有这么几种

- 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，如果一幅图保持着另一幅图的所节点标签，边关系，和边标签，那么这副图就是另一幅图的子图。一个**通路** ***(walk)*** 是一个节点列表 $v_1, v_2, ..., v_n$, 对于所有 $i \in [1, n-1]$,v_i 和 v_{i+1} 有边连接。一个**路径** ***(path)*** 是一个不含重复节点的通路, 即对于所有 $i \neq j, v_i \neq v_j$。

3.2　再生核 Hilbert 空间

Reproducing Kernel Hilbert Space

对于大量图数据的分析，核函数是一个很强大的处理工具。核函数的优势在于它无需精确计算对应点对就可以把一组数据放入一个高维的 Hibert 空间。我们用一个叫做核的函数来处理这个过程。一个二元函数 $K : X \times X \to \mathbb{R}$ 如果符合以下公式，则它是一个半正定函数。

$$\sum_{i,j=1}^{m} c_i c_j K(x_i, x_j) \geq 0 \tag{3-1}$$

上式中 $m \in \mathbb{N}$, 例子 $x_i \in X(i = [1, n])$, 系数集 $c_i \in \mathbb{R}(i = [1, n])$。另外，如果 $x, y \in X, K(x, y) = K(y, x)$ 那么这个二元函数就是对称的。一个对称半正定函数保证存在一个 Hilbert 空间 $\mathcal{H}$ 和一个映射 $\Phi : X \to \mathcal{H}$, 例如

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \tag{3-2}$$

对于所有的 $x, x' \in X$。$\langle x, y \rangle$ 表示 x 与 y 的内积。这个结果就是我们所熟悉的 Mercer 定理。一个对称半正定函数又称为 Mercer 核函数简称为核函数。通过将数据空间变为 Hilbert 空间，核函数提供了一种对包括图在内的不同数据的统一分析环境。即使一开始的数据空间根本不像一个向量空间，也可以统一化。我们称这种归一化方法为" 核诀窍"，并将其应用在许多不同的数据分析任务上，包括分类，回归，通过准则分析的特征提取等。

3.3　图小波分析

Graph Wavelets Analysis

小波函数常被当做一种用于将一个函数或者信号通过不同的变换分解和表示为它的子结构的方法。小波常用在数字化的数据上，例如通信信号或者数学函数，也可以用在一些规律的数值结构上，像矩阵和图像。但是图是一种随意的结构，并且可能表示非数值化的关系，在数据元素之间也可能存在拓扑关系。近期的一些研究证实了利用小波函数对图做多解析度分析的可行性。两个小波函数的典型就是 $Haar$ 和 $Mexicanhat$。Crovella 和其他人开发了一种对于交通网络数据分析的多尺度方法。在这个应用中，他们尝试去确定一个交通现象发生的规模。他们

将交通网络表示为一个标号图，并且用一些测量方法像每个单位时间的比特携带数来做标号。Maggioni 和其同事证实了一个通用的双正小波分析可用于图分析。在他们的方法中，他们将利用扩散操作的二元性来激发多解析度分析。他们的方法主要适用于大空间的分类，例如流形和图, 对于带属性的分子结构的适应性尚不清楚。在这里主要的技术问题就是如何在多解析度分析中包含节点标签。

第四章　基于哈希的快速图相似性搜索

Fast Graph Similarity search with hash function

正如之前所说，现在的图查询算法查询时间很快但是没有好的相似性度量方法。核函数可以提供一个好的相似性度量方法，但是核函数的矩阵运算需要大量的时间，所以我们很难直接利用其来建立索引。为了解决这个问题，我们提出了一种新的算法，G-Hash。现在的算法常常关注速度或精度的一种, 而 G-Hash 在两者效果均很好。我们利用小波图匹配核 (WA) 来定义相似性，并用 Hash 表作为索引结构来加速图相似性查询。下面我们先介绍下 WA 方法。

4.1　小波图匹配核简介

Introduction to Wavelet Graph matching kernels

WA 方法的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(4-1) , 也用到了 h 跳邻域。

$$\psi_{j,k} = \frac{1}{h+1} \int_{j/(k+1)}^{(j+1)/(k+1)} \varphi(x)dx \tag{4-1}$$

$\varphi(x)$ 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(4-2)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。

$$\Gamma_h(v) = C_{h,v} \times \sum_{j=0}^{k} \psi_{j,k} \times \bar{f}_j(v) \tag{4-2}$$

其中,$C_{h,v}$ 是一个归一化因子

$$C_{h,v} = (\sum_{j=0}^{h} \frac{\psi_{j,h}^2}{|N_h(v)|})^{-1/2}, \tag{4-3}$$

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

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

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

$$k_m(G, G') = \sum_{(u,v) \in V(G) \times V(G')} K(\Gamma^h(u), \Gamma^h(v)), \tag{4-5}$$

$$K(X, Y) = e^{\frac{-\|X-Y\|_2^2}{2}}. \tag{4-6}$$

就像在后文实验部分我们验证的一样，WA 方法展示了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。

4.2　Hash 函数实现快速图相似性搜索

Fast graph similarity search with hash functions

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。

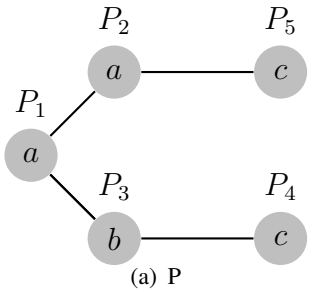
基于以上两条发现，我们引入我们的方法:*k*-NNs 查询 (G-hash).G-Hash 是一种利用 WA 方法进行精确相似度度量，并利用哈希来降低时间复杂度。具体算法见下文。

4.3　索引构建
Index construction

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[5] ，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一种类型: 定类。我们首先写出节点特征的柱形图 *(*每个数字就是一个高度*)* 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 4.1 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。

4.4　*K-NNs* 查询过程

K-NNs Query Processing

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

$$\begin{aligned} d(G,G') &= \sqrt{\|\phi(G)-\phi(G')\|_2^2} \\ &= \sqrt{\langle\phi(G)-\phi(G'),\phi(G)-\phi(G')\rangle} \\ &= \sqrt{\langle\phi(G),\phi(G)\rangle+\langle\phi(G'),\phi(G')\rangle-2\langle\phi(G),\phi(G')\rangle} \\ &= \sqrt{k_m(G,G)+k_m(G',G')-2k_m(G,G')} \end{aligned} \tag{4-7}$$

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。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

$$k_m(G,G') = \sum_{v\in G', u\in simi(v)} K(\Gamma^h(u),\Gamma^h(v)) \tag{4-8}$$

$simi(v)$ 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

$$K(G,G') \approx \sum_{v\in G', u\in simi(v)} 1 = \sum_{v\in G'} |simi(v)| \tag{4-9}$$

$|simi(v)|$ 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 $K-NNs$。

4.5　动态插入与删除

Dynamic insertion and deletion

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。

第五章　实验研究

Experimental Study

我们对我们的算法进行了广泛的实验研究来评估其有效性, 高效性及扩展性。我们在化学分子结构上测试我们的算法。对于化学结构，节点特征包括数值特征和原子布尔特征。数值特征包括元素种类, 原子部分电荷，原子电子亲和势，原子自由电子数目和原子价态等等。布尔特征包括原子是否在供体中，是否在末端碳中，是否在环中，是否为负，是否是轴向的等等。在实验中，我们仅用一个原子特征: 元素种类。

我们将我们的方法和小波分配核，C-tree,GraphGrep 还有 gIndex 进行比对。我们的算法，WA 算法，GraphGrep 和 gIndex 是基于 C++ 实现的，用 g++ 进行编译。C-tree 是用 Java 实现的，用 Sun JDK 1.5.0 编译。所有的实验都是在 Intel Xeon EM64T 3.2GHz，4G 内存，Linux 系统这一平台上测试的。

WA,G-Hash,C-tree,GraphGrep 和 gIndex 的参数是这样设置的。对于 WA 和 G-hash,h 取 2，用 *haar* 小波函数，对于 C-tree，用默认值即将最小子节点数 m 设为 20，最大 M 设为 $2m-1$，用 NBM 方法进行图映射。对于 GraphGrep 和 gIndex，全部采用默认参数。

5.1　数据集

Data Sets

我们选用许多数据集来进行试验。前五个数据集是从从 Jorisson/Gilson 数据集获得的已有数据。接下来六个是从 BindingDB 数据集中抽取的，最后一个是 NCI/NIH 艾滋病筛选集里的，表 1 显示了这些数据集和其基本情况。

5.1.1　Jorissen 数据集

Jorissen 数据集主要为一些包含活动蛋白质的的化学结构信息。我们以药物对特定蛋白质的吸引力作为的目标值。我们选取了 5 个包含 100 个分子结构的蛋白

质作为测试目标，其中 50 个化学结构连接着蛋白质，另外 50 个没连接到蛋白质上。参考文献 14 来查看详细信息。

5.1.2　NCI/NIH AIDS 抗体数据库

NCI/NIH 艾滋病抗体数据库包含 42390 个化学结构，总共有 63 种分子，最常见的是，C,O,N,S。数据集包括三种边，单边，双边，芬香边。我们选取了所有结构作为数据库，并随机选取 1000 个化学结构作为查询集合。

5.2　利用分类做节点相似性度量
Similarity Measurement Evaluation with Classification

我们用不同的相似性度量方法比较 k-NN 分类器在 Jorissen 数据集和 BindingDB 数据集上的分类精确度。对于 WA 算法，我们采用小波匹配核函数来获得核矩阵，然后计算距离矩阵来获得最近的 k 个最相似的图。对于 G-Hash，我们根据之前描述的算法计算核函数，然后找出最相似的 k 个图。对于 C-tree，我们直接找回最相似的子图。我们用标准的 5-fold 交叉验证来获得分类准确度，$(TP+TN)/S$，其中 TP 表示真正结果的数目，TN 表示真负的数目，S 是全部的数据图数目。在本实验中，我们取 $k=5$。

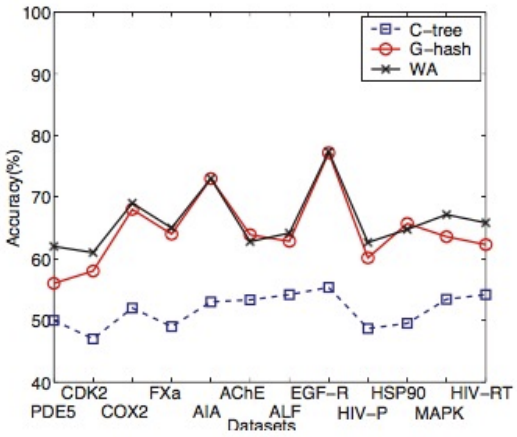


图 5.1 精确性对比图

精确性对比，实验结果如图5.1所示。可以看出，G-Hash 在所有数据集中都比 C-tree 表现优秀，至少有 8% 的提高。G-Hash 和 C-tree 的平均差异大概在 13%。而 WA 算法较 G-Hash 表现更为优异，大概有 2% 的提升。主要原因是我们简化了

距离度量公式。从实验来看，可以明显看出基于核函数的相似性度量方法比基于编辑距离的相似性方法更有优势。因为 k-NN 分类器的精确度和 k 有关，所以我们也研究了不同 k 下算法表现。实验表明精确度对比实验结果不受参数 k 影响。

5.3　伸缩性

Scalability

5.3.1　索引构建时间

Index Construction

我们探究了 G-Hash，WA 和 C-tree，GraphGrep 还有 gIndex 在 NCI/NIH 艾滋病抗体数据库上的表现。我们比较了不同算法下索引构建时间和索引大小。实验结果如图5.2,5.3所示。

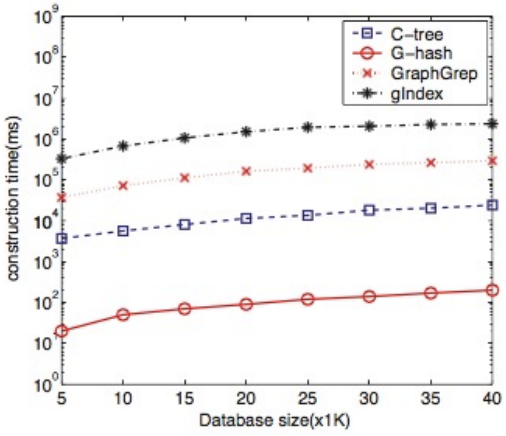


图 5.2 索引构造时间对比图

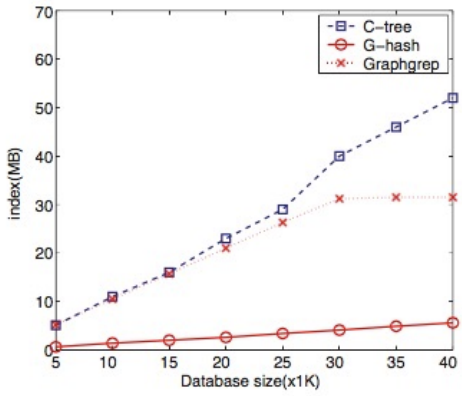


图 5.3 索引大小对比图

可以看出，无论构建时间，还是索引大小，G-Hash 算法均优于其他算法。

5.3.2　查询时间

Query Processing Time

我们探究了不同数据库大小对查询时间的影响，实验结果如图5.4所示。

显然，G-Hash 比其他算法表现均好，当数据库达到 4000 时,C-tree,GraphGrep 和 gIndex 的查询时间是 G-Hash 的 8 倍，10 倍和 100 倍。

最后我们比较了不同 k 值下查询时间，如图5.5所示。

实验表明 G-Hash 的查询时间与 k 值的大小没有影响。

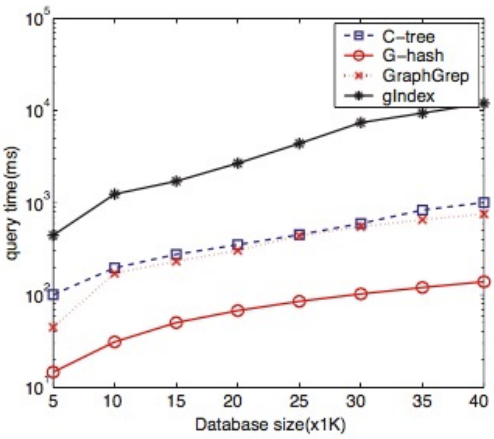


图 5.4 查询时间和数据库大小关系对比图

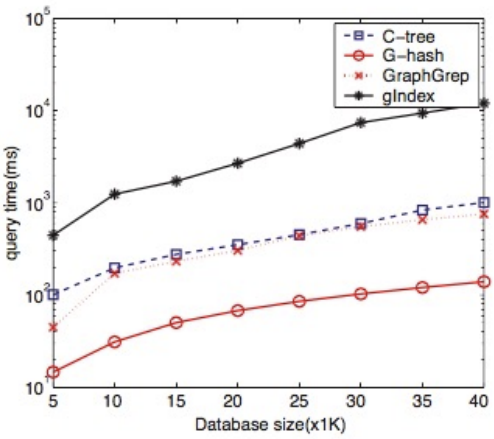


图 5.5 查询时间和 k 值关系对比图

第六章　总结与展望

Conclusions and Future Works

图作为一种通用的数据结构已被广泛运用于大量领域，像化学分子学，生物信息学等等。目前，大量知名学者都专注于数据管理方法和数据挖掘技术。而如何提出一种有效的图相似性搜索方法一直是一个重大的难题，因为现在大量算法都只专注于速度却不关心精确度。为了解决这个问题，我们提出了一种新的图查询算法 G-Hash。通过我们的实验研究，我们在构造时间和效率上都明显有优势，这表明 G-Hash 已经可以支持大规模图数据库搜索。

第七章　致谢

Acknowledgements

感谢 H.He 和 A.K.Singh 提供 C-tree 源代码。感谢 D.Shasha,J.T.L.Wang 和 R.Giugno 提供的 GraphGrep。感谢 X.Yan，P.S.Yu 和 J.Han 提供 GIndexy 源代码。我们的所有工作均是在 KU 中心支持下完成的，对他们表示由衷的感谢。

参考文献

[1] HE H, SINGH A K. Closure-tree: an index structure for graph queries[J]. Proc. International Conference on Data Engineering'06 (ICDE), 2006.

[2] Y. CAO T J, GIRKE T. A maximum common substructure-based algorithm for searching and predicting drug-like compounds[J]. Bioinformatics, 24(13), 2008.

[3] HLICH H F, WEGNER J K, SIEKER F, et al. Optimal assignment kernels for attributed molecular graphs[J]. Proceedings of the 22nd international conference on Machine learning, 2005.

[4] VERT J-P. The optimal assignment kernel is not positive definite.[J]. Technical Report HAL-00218278, French Center for Computational Biology, 2008.

[5] SMALTER A, HUAN J, LUSHINGTON G. Graph wavelet alignment kernels for drug virtual screening[J]. Proceedings of the 7th Annual International Conference on Computational Systems Bioinformatics, 2008.