Learning to Rank Graphs for Online Similar Graph Search

Bingjun Sun
Department of Computer Science and Engineering
Pennsylvania State University
University Park,PA 16802,USA
sunbingjun@hotmail.com

Prasenjit Mitra
College of Information Sciences and Technology
Pennsylvania State University
University Park,PA 16802,USA
pmitra@ist.psu.edu

C. Lee Giles
College of Information Sciences and Technology
Pennsylvania State University
University Park,PA 16802,USA
giles@ist.psu.edu

ABSTRACT

Many applications in structure matching require the ability to search for graphs that are similar to a query graph, i.e., *similarity graph queries*. Prior works, especially in chemoinformatics, have used the maximum common edge subgraph (MCEG) to compute the graph similarity. This approach is prohibitively slow for real-time queries. In this work, we propose an algorithm that extracts and indexes subgraph features from a graph dataset. It computes the similarity of graphs using a linear graph kernel based on feature weights learned offline from a training set generated using MCEG. We show empirically that our proposed algorithm of learning to rank graphs can achieve higher normalized discounted cumulative gain compared with existing optimal methods based on MCEG. The running time of our algorithm is orders of magnitude faster than these existing methods.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query formulation,Retrieval models*

General Terms

Algorithms, Design, Experimentation

Keywords

Learn to rank, graph kernel, similarity graph search

1. INTRODUCTION

Graphs have been used to represent structured data for a long time. Increasingly, massive complex structured data, such as chemical molecule structures [6], social networks [1], and XML structures [12], are identified and studied in many areas. Efficient and effective access of the desired structure information is crucial in many areas from generic and vertical research engine [8, 9, 7].

Usually a typical query to search for desired graph information is a *subgraph query* that searches for graphs containing exactly the query graph, i.e., the *support* [10]. However, sufficient knowledge to select subgraphs to characterize the desired graphs is required and sometimes no support exists, so the *similarity graph query* searching for all graphs similar to the query graph is desired to bypass the subgraph selection. To measure the similarity of two graphs, previous methods [5, 10] usually use the size of the maximum common edge subgraph (MCEG) between two graphs, i.e., the number of edges in MCEG. The crux of similarity graph search lies in the complexity of the MCEG isomorphism algorithm for similarity measurement. However, since the MCEG isomorphism problem is NP-hard [10], it is prohibitively expensive to scan all graphs in real time to find MCEG. Previous works [10, 5] use different filters to prune out unsatisfied graphs given a user specified minimum MCEG size. If users need more search results, the minimum MCEG size has to be reduced and more graphs are retrieved. However, previous methods are still slow because MCEG isomorphism tests have to be executed on the filtered graph set, which usually is still large. Rather than executing MCEG isomorphism tests on the fly, we proposed to index the graphs offline using subgraph features to enable fast graph search.

The goal of using MCEG sizes to measure the graph similarity is to rank search results. Instead of using MCEG, we propose a novel approach that uses a linear graph kernel function to rank retrieved graphs using the indexed subgraph features and feature weights learned from a training set. Our method generates a training set offline using MCEG isomorphism, a training query set, and a graph set. Our approach avoids MCEG isomorphism online and is more efficient computationally than previous methods [10, 5]. Experimental results also show that our method can achieve a reasonably high normalized discounted cumulative gain [13] in a significantly shorter time in comparison to existing methods. Moreover, because our method learns the ranking function from a training set, it can be applied to other similarity metrics, including similarity scores labeled by human experts or extracted from user logs.

2. PRELIMINARIES

In this work, we consider *labeled undirected graphs* and *connected labeled undirected subgraph* features, where a path exists for any pair of vertices on the subgraph. Notations are given as follows:

Definition 1. **Labeled Undirected Graph:** A labeled undirected graph is a 5-tuple, $G = \{V, E, L_V, L_E, l\}$, where V is a set of vertices, each $v \in V$ is an unique ID representing this vertex, $E \subseteq V \times V$ is a set of edges with each $e = (u, v) \in E, u \in V, v \in V$, L_V is a set of vertex labels, L_E is a set of edge labels, and $l : V \cup E \rightarrow L_V \cup L_E$ is a function assigning

CIKM'09, November 2–6, 2009, Hong Kong, China.

labels to vertices and edges on the graph. The size of a graph G, $|G|$, is defined as the edge count of G.

Definition 2. **Subgraph and Frequency:** A subgraph G' of a graph G is also a graph where $V_{G'} \subseteq V_G$ and $E_{G'} \subseteq E_G$, i.e. $G' \subseteq G$. G is the supergraph of G'. An embedding of a subgraph G' in a graph G, i.e., $E_{G' \subseteq G}$, is an instance of $G' \subseteq G$. We say that in a graph G, two embeddings $E_{G' \subseteq G}$ and $E_{G'' \subseteq G}$ overlap, i.e. $E_{G' \subseteq G} \cap E_{G'' \subseteq G} \neq \emptyset$, iff $\exists v, v \in G' \wedge v \in G''$. The frequency of a subgraph G' in a graph G, i.e., $F_{G' \subseteq G}$, is the embedding number of G' in G.

Definition 3. **Graph Isomorphism:** An isomorphism between two graphs G and G' is a bijective function $f : V_G \rightarrow V_{G'}$ mapping each vertex on G to a vertex on G', such that $\forall v \in V_G, l_G(v) = l_{G'}(f(v))$, and $\forall e = (u, v) \in E_G, (f(u), f(v)) \in E_{G'}$ and $l_G((u, v)) = l_{G'}((f(u), f(v))$. Since it is a bijective function, a bijective function $f' : V_{G'} \rightarrow V_G$ exists with the same of reverse one to one mapping of f.

Definition 4. **Canonical labeling:** A canonical labeling $CL(G)$ is a string to represent a graph G, where given two graphs G and G', G is isomorphic to G' iff $CL(G) = CL(G')$.

Definition 5. **Maximum Common Edge Subgraph:** A graph G' is a *common edge subgraph* of G_i and G_j, if G' is isomorphic to subgraphs of G_i and G_j. A common edge subgraph G' of G_i and G_j is a *maximum common edge subgraph*, i.e., $MCEG(G_i, G_j)$, iff no common edge subgraph G'' of G_i and G_j exists that $|E(G'')| > |E(G')|$, i.e., the edge count on G'' is larger than that on G'. The size of a MCEG, $|MCEG(G_i, G_j)|$, is defined as its edge count.

Note that an MCEG is not necessarily a connected graph. To make the similarity scores comparable between different sizes of query graphs in our research, we normalize the MCEG sizes into the interval $[0, 4]$, where 4 means the query graph is a subgraph of the retrieved graph, while 0 means no edge matched. These *normalized MCEG sizes* are used as the similarity scores for training and test in our experiments.

Discounted cumulative gain (DCG) is the most widely used metric to evaluate the performance of ranking functions. Given a query q and n ordered results , it is computed as follows [3],

$$DCG = \sum_{i=1}^{n} c_i f(y_i), \tag{1}$$

where $y_i, i = 1, ..., n$ are the real relevance scores of the n ordered results, c_i is a non-increasing function of i, typically $c_i = 1/log(i + 1)$, and $f(y_i)$ is a non-decreasing function of y_i, typically $f(y_i) = 2^{y_i} + 1$, or sometimes $f(y_i) = y_i$. If y_i is higher, the result i is more relevant. If $y_i \in \{0, 1\}$, only relevance and irrelevance are considered. Normalized discounted cumulative gain (NDCG) is a score that normalize DCG scores into the interval of $[0, 1]$ using the maximum DCG that can be achieved. The average NDCG, $NDCG_Q$, for the whole query set Q is used for evaluation.

3. LEARNING TO RANK GRAPHS

In this section, we describe our search algorithm and the weighted linear graph kernel to measure the graph similarity. Then we describe how to learn the weights.

3.1 Similarity Graph Search

A naive approach to similarity graph search is to scan all the graphs to find MCEGs of the query and each graph, which is prohibitively expensive to be executed in real time. Usually previous methods first filter out graphs with lower MCEG sizes than a given threshold. Then they determine the size of the MCEG between the query graph hand each candidate graph. This size is used as the similarity score [10, 5] for ranking the result graphs. Detecting MCEG isomorphism is NP-hard [10], and all existing algorithms for MCEG isomorphism are extremely expensive. This makes online similarity graph search prohibitively slow for large graph databases. We propose a new efficient similarity graph search algorithm shown in Algorithm 1. It first returns all the graphs in the support of the query that have the maximum MCEG size, and then use a fast graph ranking function to compute a heuristic similarity score. To return the support of the query, subgraph isomorphism tests are required. Algorithms for subgraph isomorphism are significantly faster than those for MCEG isomorphism [10]. Our proposed fast graph ranking function uses a weighted kernel between vectors constructed from subgraph features. Thus, our proposed method is significantly faster for online queries in comparison with methods using MCEG.

First, we assume we have built an index of graphs using their subgraph features. Subgraph features can be discovered from those graphs using any previous methods [10, 4]. Then, as illustrated in Algorithm 1, given a query graph G_q, the algorithm finds the *support* of G_q, D_{G_q} (Line 1-11). Thus, all the graphs in the support should be returned as the top-most candidates in the result list. If G_q is indexed, it is simple to find D_{G_q} using the index. Otherwise, candidates containing all the indexed subgraph features of G_q is returned and subgraph isomorphism is performed to remove graphs that do not contain G_q. Second, if more results are required, similar graphs with lower similarity scores are returned (Line 12-19). Our proposed method uses a weighted kernel as the similarity function. All the graphs containing at least one indexed subgraph feature of G_q is returned as candidates except support graphs found at the first stage. For each candidate and the query G_q, a similarity score is computed using a weighted linear graph kernel based on the indexed subgraph features and corresponding weights. This similarity score computation is fast and can be computed during search. Finally, graphs are sorted based on the similarity scores and the top results are returned.

3.2 Graph Kernels

A graph kernel is defined as follows,

Definition 6. **Graph Kernel:** Let X be a set of graphs, $\mathcal{R}$ denotes the real numbers, $\times$ denotes set product, the function $K : X \times X \rightarrow \mathcal{R}$ is a kernel on $X \times X$ if K is *symmetric*, i.e. $\forall G_i$ and $G_j \in X$, $K(G_i, G_j) = K(G_j, G_i)$, and K is *positive semi-definite*, i.e. $\forall N \geq 1$ and $\forall G_1, G_2, ..., G_N \in X$, the N by N matrix K defined by $K_{ij} = K(G_i, G_j)$ is *positive semi-definite*, i.e. $\sum_{ij} c_i c_j K_{ij} \geq 0, \forall c_1, c_2, ..., c_N \in \mathcal{R}$. Equivalently, a symmetric matrix is positive semi-definite if all its eigenvalues are nonnegative [2].

The MCEG sizes of two graphs is also a graph kernel, but expensive to compute. We define a time-efficient and learnable weighted linear graph kernel based on indexed subgraph features and corresponding frequencies as follows,

$$K(G_i, G_j) = \sum_{G' \in S} W(G') min(F_{G' \subseteq G_i}, F_{G' \subseteq G_j}). \tag{2}$$

$W(G')$ are the learnable parameters in this kernel. Thus, our goal is to learn the kernel function to approximate a target function for ranking, but not necessarily the same as the function of MCEG sizes.

Algorithm 1 Similarity Graph Search

Algorithm: SGS(G_q,S,$Index_D$,n):
Input: Query Subgraph G_q, indexed subgraph set S, index of the graph set D, $Index_D$, and the number of returned results, n.
Output: A sorted list of n graphs similar to G_q, $List_{G_q}$.
1. **if** G_q is indexed,
2. find all $G \supseteq G_q$ using $Index_D$, i.e., the support of G_q, D_{G_q};
3. **else**
4. $D_{G_q} = \{\emptyset\}$;
5. find all subgraphs of G_q, $G'_q \in S$ with $F_{G' \subseteq G_q}$;
6. **for all** G'_q **do**
7. Find $D_{G'_q}$, where $\forall G \in D_{G'_q}, F_{G'_q \subseteq G} \geq F_{G'_q \subseteq G_q}$,
8. Then $D_{G_q} = D_{G_q} \cap D_{G'_q}$;
9. **for all** $G \in D_{G_q}$ **do**
10. **if** subgraphIsomorphism(G_q, G)==false, remove G;
11. **if** $|D_{G_q}| \geq n$ **return** $List_{G_q}$ = top n graphs $G \in D_{G_q}$;
12. $S_{G_q} = \{\emptyset\}$;
13. find all subgraphs of G_q, $G'_q \in S$;
14. **for all** G'_q **do**
15. Find $D_{G'_q}$, where $\forall G \in D_{G'_q}, F_{G'_q \subseteq G} \geq 0$,
16. Then $D_{G_q} = D_{G_q} \cup D_{G'_q}$;
17. $S_{G_q} = S_{G_q} - D_{G_q}$
18. **for all** $G \in S_{G_q}$ **compute** $similarity(G_q, G)$;
19. sort S_{G_q} in terms of $similarity(G_q, G)$;
20. **return** $List_{G_q} = D_{G_q}$+ top $(n - |D_{G_q}|)$ graphs $G \in S_{G_q}$;

Our learning task also suffers from the data sparsity problem [11], i.e., many features appearing in the test set may not have appeared in the training set. With the goal to make the space dense, we use a feature extraction method to generate features from subgraphs, and cluster subgraphs with the same feature vector together into a single dimension. Let us denote the many-to-one mapping function from a subgraph G' to a subgraph cluster using the proposed feature exaction method as $Clu(G')$. Then, we can rewrite the linear graph kernel as follows,

$$K(G_i, G_j) = \sum_{G' \in S} W(Clu(G'))min(F_{G' \subseteq G_i}, F_{G' \subseteq G_j}). \quad (3)$$

We extract the following features of a subgraph: the number of edges, the number of vertices with a specific label, the number of branches, and the number of cycles.

3.3 Kernel Learning using Regression

Suppose we have a training set with N instances, $T = \{G_{(q,n)}, G_n, y_n\}_{n=1}^N$, where each instance is a pair of a query graph $G_{(q,n)}$ and a retrieved graph G_n, and y_n is the similarity score between them. As mentioned before, if $y_n \in \{1, 0\}$, it represents only relevance or irrelevance between $G_{(q,n)}$ and G_n; Otherwise, it represents the similarity between $G_{(q,n)}$ and G_n. This training set can be generated by arbitrary similarity functions that take in two graphs $G_{(q,n)}$ and G_n as inputs and output a similarity score y_n. In our work, we use the normalized MCEG sizes as the similarity scores, y_n.

Our eventual goal is to find the optimal linear weighted graph kernel that maximizes the NDCG function that is the metric to evaluate the ranked retrieved results. However, the objective function of NDCG cannot be represented by the parameters of the graph kernel in a closed form, so we cannot optimize the NDCG function directly and find the optimal graph kernel. Instead, we optimize a specific loss function $f(y_n)$, the non-increasing function in Equation 1, using regression. Previous work [3] showed that regression on $f(y_n)$ can achieve a better NDCG of the ranked search results than regression on y_n. Thus, one of the key issue is to choose the loss function. We choose a weighted $L2$ loss function,

$$L_w = \sum_{n=1}^{N} w_n(f(y_n) - f(\hat{y}_n))^2, \quad (4)$$

where $f(y_n) - f(\hat{y}_n)$ is the error of the instance n, w_n is the weight of Instance n, and $\hat{y}_n$ is the predicted value of y_n. Instances with higher relevance scores are considered more important, so that they have higher weights. However, no previous work determined that what the value of the instance weights should be. Empirically we define the weights as the normalized MCEG sizes. In our work, we use an unweighted loss function but a weighted sampling method to generate a training set rather than using the weighted loss function. Using this method, we can have a smaller training set than using uniform sampling but weighted loss function.

4. EXPERIMENTS

In this section, we evaluate our proposed approach by comparing with two heuristics and the method using MCEG isomorphism in terms of NDCG and response time of queries. We use the real data set and test query set used by Yan, et al., [10]. It is a NIH HIV antiviral screen data set that contains 43905 chemical structures. The experimental subset contains 10000 chemical structures selected randomly from the whole data set and the query set for evaluation contains 6000 randomly generated queries, i.e., 1000 queries per query size, where $Size(G_q) = \{4, 8, 12, 16, 20, 24\}$. Although we only use chemical structures for experiments, our approach is applicable to any structures that can be represented by graphs, such as DNA sequences and XML files.

In our experiment, rather than using a weighted loss function, we use a weighted sampling method to generate a training set off-line based on the MCEG isomorphism algorithm. We first generate 6000 queries with the same distribution of the test query set described above. Then for each query graph, we randomly select graphs from the 10000 chemical structures with corresponding conditional sampling probability given the normalized MCEG sizes (as mentioned before, they are normalized between [0,4]) between the query and the graph. Finally we use the normalized MCEG sizes as the target similarity scores y_n for the n^{th} query-graph pair. Since we only care top 20 search results, we remove all the query-graph pairs with low normalized MCEG sizes. We also remove query-graph pairs where the query is a subgraph of the graph. Since finding the MCEG between the query and the selected graph is time-consuming, to speed up the training instance generation, we do as follows: 1) given a query, to search all graphs using the Algorithm 1, and the similarity function is to use the linear graph kernel with uniform feature weights, 2) to pick only the top 1000 returned graphs and remove graphs among them that are supergraphs of the query, and 3) to compute the normalized MCEG sizes y_n between each survived graph and the query and sample the pair using the probability of $(y_n/4)/10$.

The final training set contains instances of query and graph pairs with a similarity score y_n, and each instance has a subgraph feature vector where each entry is the minimum one of the subgraph frequency on the query and on the graph (shown in Equation 2). Finally, in our experiment, we generate a training set with a total of 459,047 pairs of queries and graphs. Any previous subgraph feature selection methods can be applied to select a dense subset of frequent

Table 1: Average NDCGs

Method	NDCG 1	NDCG 3	NDCG 10	NDCG 20
learn	**94.224%**	**94.842%**	**95.648%**	**96.308%**
size	93.259%	93.896%	94.716%	95.336%
uniform	93.403%	94.043%	94.898%	95.570%
$size_L$	93.140%	93.793%	94.687%	95.318%
$uniform_L$	93.208%	93.872%	94.807%	95.470%

subgraphs [10, 4]. Then we cluster subgraphs using feature extraction to get 300 features finally.

Besides comparing different feature weights, we also use two different sizes ($|S| = 9855$ subgraph features v.s. $|S| = 50475$ subgraph features) of indexed subgraph sets to show the effect of the number of the indexed subgraph set, S. In the experiments, we compare the following methods: 1) linear graph kernel with subgraph feature weights learned using regression on $f(y_n)$ with the $L2$ loss function and weighted sampling (*learn* in Table 1), 2) linear graph kernel using subgraph sizes as feature weights (*size* in Table 1), 3) linear graph kernel with uniform subgraph feature weights (*uniform* in Table 1), 4) linear graph kernel using subgraph sizes as feature weights with a larger subgraph feature set ($size_L$ in Table 1), and 5) linear graph kernel with uniform subgraph feature weights with a larger subgraph feature set ($uniform_L$ in Table 1). Note that the method using MCEG always has the perfect NDCG, because it is assumed to be the gold standard. For the query response time in Figure 1, since the proposed method has similar online response time no matter what kind of subgraph feature weights it uses, we only evaluate the learned weights and called it *graph kernel*. We applied the techniques in [5] to optimize the algorithms of the MCEG isomorphism.

In the experiments, we evaluate all queries for different query sizes together. Average experimental NDCG results of top 1, 3, 10, and 20 search results are shown in Table 1. We can see all the methods achieve NDCGs above 93%, which are significantly higher than the NDCGs for web search [3]. the average NDCGs are improved by about 1% for all queries. Especially the 1% improvement is based on such high NDCGs above 93%. From the previous work [3], for the case of a standard deviation = 24 and a sample size = 10000, roughly speaking, the difference of two NDCGs is considered as "significant" if it is larger than 0.47%. Hence, the improvements of NDCGs after learning are roughly statistically significant for all NDCGs.

Finally we compare the average online response time for using the proposed linear graph kernel and MCEG isomorphism. As in the proposed Algorithm 1, to return top n similar graphs using MCEG isomorphism, two cases exist: 1) If the top n similar graphs all contain the query, only subgraph isomorphism tests are executed rather than running MCEG isomorphism tests. In this case, the response time of a query is the same as our proposed method. 2) If only part of or none of the top n similar graphs contain the query, the MCEG isomorphism algorithm has to be executed to find more similar graphs. However, applying the MCEG isomorphism test to scan all the graphs is prohibitively expensive. As mentioned above, previous methods [10, 5] use filters to remove part of graphs containing smaller MCEGs than the MCEG size threshold before preforming the MCEG isomorphism algorithm. However, no previous work proposed methods to find top n similar graphs containing the largest

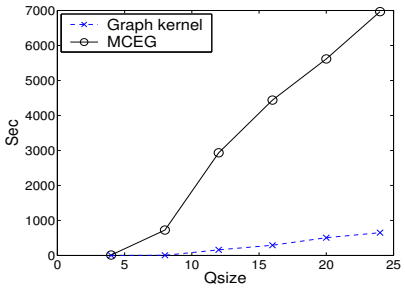

Figure 1: Response time of graph search

MCEG sizes. To simplify the situation for time complexity comparison, we assume that we have a filter to return only 100 graph candidates to execute the MCEG isomorphism test. That is, the curve in Figure 1 is the response time that at most 100 MCEG isomorphism tests are performed. Actually for most cases, more than 100 graph candidates are returned to perform MCEG isomorphism tests [10], which means in practice, using the MCEG isomorphism algorithm requires even a longer average response time than the cases shown in our experiments. Figure 1 shows the curves of average response time of similarity graph queries using two ranking methods: *graph kernel* using weighted linear graph kernel, and *MCEG* using the MCEG isomorphism test to rank graphs. It shows that our proposed method *graph kernel* is significantly more time efficient than *MCEG*, and can achieve high NDCGs above 94%.

5. ACKNOWLEDGMENTS

We acknowledge the partial support of NSF Grant 0535656 and 0845487.

6. REFERENCES

[1] B. Chen, Q. Zhao, B. Sun, and P. Mitra. Temporal and social network based blogging behavior prediction in blogspace. In *Proc. ICDM*, 2007.

[2] D. Haussler. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, 1999.

[3] P. Li, C. J. Burges, and Q. Wu. Learning to rank using classification and gradient boosting. In *Proc. NIPS*, 2007.

[4] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *Proc. SIGKDD*, 2004.

[5] J. W. Raymond, E. J. Gardiner, and P. Willet. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6):631–644, 2002.

[6] B. Sun, P. Mitra, and C. L. Giles. Mining, indexing, and searching for textual chemical molecule information on the web. In *Proc. WWW*, 2008.

[7] B. Sun, P. Mitra, H. Zha, C. L. Giles, and J. Yen. Topic segmentation with shared topic detection and alignment of multiple documents. In *Proc. SIGIR*, 2007.

[8] B. Sun, Q. Tan, P. Mitra, and C. L. Giles. Extraction and search of chemical formulae in text documents on the web. In *Proc. WWW*, 2007.

[9] B. Sun, D. Zhou, H. Zha, and J. Yen. Multi-task text segmentation and alignment based on weighted mutual information. In *Proc. CIKM*, 2006.

[10] X. Yan, F. Zhu, P. S. Yu, and J. Han. Feature-based substructure similarity search. *ACM Transactions on Database Systems*, 2006.

[11] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proc. SIGIR*, 2001.

[12] Q. Zhao, L. Chen, S. S. Bhowmick, and S. Madria. Xml structural delta mining: issues and challenges. *Data and Knowledge Engineering*, 2006.

[13] Z. Zheng, H. Zha, K. Chen, and G. Sun. A regression framework for learning ranking functions using relative relevance judgments. In *Proc. SIGIR*, 2007.