Scalable Graph Similarity Search in Large Graph Databases

P. Kiran
NSS College Of Engineering, Palakkad, India
pkiran.kala@gmail.com

Naveen Sivadasan
TCS Innovation Labs, Hyderabad, India
naveen@atc.tcs.com

***Abstract*—We consider the problem of searching a collection of graphs $\mathcal{D}$ to find graphs that are most similar to a query graph Q. This has several applications in areas like computational biology, drug design, computational chemistry, collaborative networks, social networks etc. We use graphlet kernel to define similarity between graphs. In order to make the similarity search faster, we build an efficient nearest neighbor data structure on the graph collection using locality sensitive hashing technique. The graphs are embedded into a vector space and these vectors are used to build the nearest neighbor data structure. Computing the vector space embedding is the most compute intensive part in our algorithm. To scale our algorithm to large graph collections, we give an efficient Map-Reduce implementation for vector space embedding. We perform experiments on real world datasets (AIDS dataset) and synthetic datasets to show the effectiveness of our algorithm.**

I. Introduction

Similarity based graph searching is a fundamental problem in several applications related to social networks, collaboration networks, software testing, computational biology, molecular chemistry etc. In similarity search problem, given a large collection (database) of candidate graphs and a query graph, graphs from the collection that are most similar to the query graph needs to be identified. For instance, in drug design applications, the graph collection corresponds to several thousands of candidate drug molecules from which molecules most similar to given query molecule needs to be searched.

In most of these applications, underlying graphs can be noisy and graphs with similar patterns needs to be identified. For instance, in computational biology, the data is noisy due to possible errors in data collection and different experimental thresholds. In object-oriented programming, querying object usage patterns against the target object dependency graph of a program execution can help in identifying potential bugs [21]. In molecular chemistry, identifying similar molecular structures is a fundamental problem. Searching for similar subgraphs is central to social network mining. Similarity searching is therefore more natural in these applications in comparison to exact search, also known as the graph isomorphism. Quality of the solution and computational efficiency are two major challenges in similarity search problems. In this work, we assume that graphs in the collection and query graphs are all connected, undirected and unweighted with no edge or vertex labels.

Most applications work with a distance metric to define similarity between two graphs. Some of the popular distance metrics are Euclidean distance, Hamming distance, Edit distance and Kernel functions [6], [17], [29]. In this work, we use graph kernel functions to define graph similarity.

978-1-4673-6670-0/15/$31.00

Kernels are real valued symmetric functions used to define similarity between pairs of objects from a domain. Positive definite kernels are not only useful in defining similarity but also allow implicit mapping of objects to a high-dimensional feature space and operating on this space without requiring explicit mapping to feature space. Kernel value corresponds to inner product of these implicit feature vectors. This is usually computationally cheaper than explicit computation. This approach is usually called the kernel trick or kernel method. Many standard machine learning algorithms like support vector machine (SVM), principle component analysis (PCA) etc., can directly work with kernels. Kernel methods have been applied successfully in the case of sequence data, text, images, graphs, videos etc.

Several kernels have been investigated in the past in the context of graphs [9], [11], [25]. Most of these graph kernels are defined based on graph properties such as random walks [8], [15], cyclic patterns [12], subtree patterns [14], graph edit distance [20], shortest paths [3], [4], occurrence frequency of special subgraphs [7], [18], [23] and so on.

Graphlet kernels are defined based on occurrence frequencies of small induced subgraphs called *graphlets* in the given graph [26]. In [26], it was shown that graphlet kernels provide better SVM classification accuracy in comparison to random walk kernel and shortest path kernel for various datasets including protein and enzyme data [26]. Graphlet kernels are also of theoretical interest. It is known that under certain restrictions, if two graphs have distance zero with respect to their graphlet kernel value then they are isomorphic [26]. Improving the efficiency of computing graphlet kernel is also studied in [26]. In our work, we use graphlet kernels to define graph similarity.

II. Related Work

Similarity based graph searching has been researched in the past under various settings. Graphs are assumed to be labeled in many of the previous studies. In one class of problems, a large database of graphs is given and the goal is to find most similar matches in the database with respect to given query graphs [19], [24], [30]–[33]. In another setting, given a target graph and a query graph, the goal is to identify subgraph of the target graph that is most similar to the query graph [16], [27]. Different notions of similarity were also explored

for these classes of problems. In [26], graphlet kernel based similarity search was studied. The main focus in [26] was to establish the effectiveness of using graphlet kernel to identify similar graphs from a collection. They give an $O(nd^{k-1})$ time algorithm to compute k sized graphlet kernel for graphs with n vertices and maximum degree d, which is still prohibitively expensive for large dense graphs. The similarity search is then performed by a sequential scan of the collection. In [28], approximate matching of query graph in a database of graphs was investigated for labeled graphs. For this, structural information of the graph was stored using B-tree index and important vertices of a query graph were matched first and then the match was extended progressively. In [34], graph similarity search on labeled graphs from a large database of graphs under minimum edit distance was studied. In [10], authors proposed a method based on the *C-tree*, where each node in tree will have an information of its descendants. They then performed pairwise graph comparison using heuristics. In [16], computing top-k approximate subgraph matches for a given query graph was studied. Subgraph matching in a large target graph for graphs deployed on a distributed memory store was studied in [27]. It looks for exact matching and not similarity matching. Though different techniques were studied in the past, very little work exists for unlabeled case.

III. Our Contribution

We give a similarity search algorithm for finding graphs from a given graph collection that are most similar to a given query graph. We propose a modified version of graphlet kernel in [26] in order to define similarity between graphs. The kernel is based on embedding of graphs to a vector space. In order to make the similarity search faster, we build an efficient nearest neighbor data structure on the graph collection using locality sensitive hashing technique. The vectors obtained by embedding graphs into vector space are used to build the nearest neighbor data structure. Computing the vector space embedding is the most compute intensive part in our algorithm. To scale our algorithm to large graph collections, we give an efficient Map-Reduce implementation for vector computation. We perform experiments on real world datasets (AIDS dataset) and synthetic datasets to show the effectiveness of our algorithm.

IV. Preliminaries

Graph is an ordered pair $G = (V, E)$ where V is the set of vertices and E is the set of edges. To avoid ambiguity, we also use $V(G)$ and $E(G)$ to denote the vertex and edge set of G. We consider only connected, undirected and unweighted graphs with no vertex or edge labels. A subgraph H of G is a graph whose vertices are a subset of V, and whose edges are a subset of E and is denoted as $H \subseteq G$. An induced subgraph G' is a graph whose vertex set V' is a subset of V and whose edge set is the set of all edges present in G between vertices in V'.

Definition 1 (Graph Isomorphism)**.** *Graphs G_1 and G_2 are isomorphic if there exists a bijection $b : V(G_1) \to V(G_2)$ such that any two vertices u and v of G_1 are adjacent in G_1 if and only if $b(u)$ and $b(v)$ are adjacent in G_2.*

Definition 2 (Graph Similarity Searching)**.** *Given a collection of graphs and a query graph, find graphs in the collection that are closest to the query graph with respect to a given distance function between graphs.*

A. Graphlet Vector

Graphlets are fixed size non isomorphic induced subgraphs of a large graph. Typical graphlet sizes considered in applications are $3, 4$ and 5. For example, Figure 1 shows all possible non isomorphic size 4 graphlets. There are 11 of them of which 6 are connected. We denote by D_l, the set of all size l graphlets that are connected. The set D_4 is shown in Figure 2.

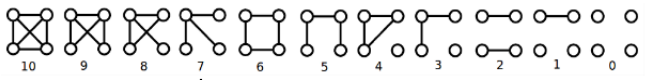

Fig. 1. Set of all non isomorphic graphlets of size 4

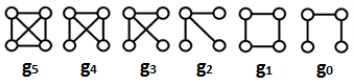

Fig. 2. Non isomorphic connected graphlets of size 4

Definition 3 (Graphlet Vector)**.** *For a given l, the graphlet vector f_G corresponding to a graph G is a frequency vector of dimension $|D_l|$ whose ith component corresponds to the number of occurrences of the ith graphlet of D_l in G.*

B. Graphlet Kernel

In this work, we use a graphlet kernel which is a modification of the kernel defined in [26]. In order to define the modified graphlet kernel, we first define vertex level graphlet vector that captures topological neighborhood information of vertices in the graph. These vectors are in turn used to define our modified graphlet kernel. Given a fixed positive integer t and graph G, let $N(v)$ denote the depth t neighbors of vertex v in G. That is, $N(v)$ is the subset of all vertices in G (including v) that are reachable from v in t or less edges. Let H_v denote the subgraph induced by vertices $N(v)$ in G. We denote by $f_G(v)$, the graphlet vector corresponding to graph H_v, with respect to size l graphlets for some fixed l. We note that for defining the graphlet vector $f_G(v)$ for a vertex, there are two implicit parameters l and t. To avoid overloading the notation, we assume them to be some fixed constants and specify them explicitly when required. Values of l and t are parameters to our final algorithm.

Definition 4 (Modified Graphlet Kernel $K(G, G')$)**.** *Given two graphs G and G', let vectors h_G and $h_{G'}$ denote $\sum_{u \in V(G)} f_G(u)$ and $\sum_{u \in V(G')} f_{G'}(u)$ respectively after L_2 normalization. The kernel $K(G, G')$ is given by the dot product $K(G, G') = h_G^T h_{G'}$.*

The original graphlet kernel of [26] is the dot product of graphlet vectors f_G and $f_{G'}$ of G and G' respectively after L_1 normalization. We use L_2 normalization in our kernel as L_2 norm is directionally invariant. The vector h_G for a graph G can be interpreted as a weighted version of original graphlet vector f_G in the following sense. Consider an induced graphlet D in G that is in the vicinity of vertices V' of G. That is, each vertex of D is reachable from any vertex of V' in t or less

edges. In this case, D contributes to vertex level graphlet vector $f_G(v)$ for each vertex v in V'. Contribution of D towards the graphlet vector f_G is therefore proportional to the size of V'. Hence, larger the neighborhood of a graphlet, higher its contribution to f_G. This in contrast to the graphlet vector f_G where each induced graphlet vector has equal contribution. In our experiments it was observed that for graphs whose f vectors are nearly identical, their h vectors were better at capturing their topological differences.

If graphs G and G' are isomorphic then clearly their corresponding graphlet vectors f_G and $f_{G'}$ are identical. It is conjectured that given two graphs G and G' of n vertices and their corresponding graphlet vectors f_G and $f_{G'}$ with respect to $n-1$ sized graphlets D_{n-1}, graph G is isomorphic to G' if and only if f_G is identical to $f_{G'}$ [26]. The conjecture has been verified for $n \leq 11$ [26].

Kernels based on similarity of graphlet vectors provide a natural way of expressing similarity of underlying graphs. Values of $K(G, G')$ lie in the range $[0, 1]$ and larger values of $K(G, G')$ indicate higher similarity between G and G'. We note that vector h_G is in fact an explicit embedding of graph G to a vector space of dimension $|D_l|$.

PROBLEM STATEMENT. *Let $\mathcal{D}$ be a collection of graphs let Q be a query graph. For a given k, find k graphs from $\mathcal{D}$ having highest kernel value with Q.*

C. Locality Sensitive Hashing

In locality sensitive hashing (LSH), similar items are hashed to the same bucket in the hashtable with a high probability [13]. This is different from the conventional hashing where elements are randomly distributed across the hashtables. Locality sensitive hashing has been successfully used to solve approximate nearest neighbor problem [2], [5], [13], [22]. A family of hash functions $\mathcal{F}$ is said to be (d_1, d_2, p_1, p_2)-*sensitive* if for every $f \in \mathcal{F}$:

1) if $d(x, y) \leq d_1$, then the probability that $f(x) = f(y)$ is at least p_1.
2) if $d(x, y) \geq d_2$, then the probability that $f(x) = f(y)$ is at most p_2.

The basic idea of locality-sensitive hashing is that similar objects are more likely to map to same location. Similarity between the objects are defined based on distance between the objects. There are several ways for defining distance between objects such as Euclidean distance, Jaccard distance, Hamming distance, edit distance, cosine distance, angle between vectors etc. We use locality-sensitive hashing technique to build nearest neighbor search data structure for the input graph collection $\mathcal{D}$. For this, the set of vectors $\{h_G\}$ corresponding to collection $\mathcal{D}$ are stored in the LSH data structure. The distance in our case is cosine distance between vectors. For our LSH data structure, we use angle between vectors instead of cosine distance. This is because, for unit vectors, cosine distance is known to be well approximated by angle in case of small angles. We use the standard random projection [13] based hash functions for our LSH data structure.

D. Hadoop and Map-Reduce

Hadoop is an open source framework used to perform data intensive tasks on a computing cluster. Each node in the cluster could be of commodity hardware. Hadoop framework handles fault tolerance, job allocation, migration etc. Two major components of the Hadoop framework are Hadoop Distributed File System(HDFS) and the Map-Reduce programming model.

Map-Reduce programming model is suitable for large scale data processing. Typical Map-Reduce process consists of three phases, namely, (a) map phase where the data is filtered into smaller processing units in parallel (b) shuffle phase where the output of the map phase is redistributed according the output keys produced by the map phase such that all data with same output key are collected together for the subsequent reduce phase, (c) reduce phase where the grouped data are again processed in parallel.

V. Our Algorithm

Our algorithm consists of a one time preprocessing phase and a querying phase. In the preprocessing phase, vector h_G for each graph G in the collection $\mathcal{D}$ is first computed. The LSH based nearest neighbor data structure is then built on these h_G vectors. Querying phase uses this nearest neighbor data structure to identify k nearest neighbors in the graph collection for the given query graph. In the following we discuss the details of our algorithm.

A. Computing Vectors h_G for the Collection $\mathcal{D}$

We compute vector h_G for each graph G in $\mathcal{D}$. For this, we first compute vector $f_G(v)$ for each vertex v of G. The pseudo code for this is given in Algorithm 1.

Algorithm 1 Computation of vector $f_G(v)$ for vertex v of G

Input: Graph G, vertex v, depth t, graphlet size l
Output: Vector $f_G(v)$

```
Run breadth first traversal on G starting from v till depth
t. Let N(v) be the set of visited vertices including v.
Let H_v be the subgraph of G induced by N(v).
Compute graphlet vector f_G(v) for graph H_v.
Normalize f_G(v) by ||f_G(v)||_2.
return f_G(v)
```

In the above algorithm, computing graphlet vector $f_G(v)$ for H_v is done by incrementing the frequency component of $f_G(v)$ corresponding to each l sized induced subgraph of H_v. The value of l is usually $3, 4$ or 5. We fix l to be 4 in our implementation. We use a precomputed hash table to efficiently identify target graphlet isomorphic to the induced subgraph under consideration. The keys in the hash table are all possible adjacency matrices for graphs on l vertices and values are their corresponding graphlets. After computing $f_G(v)$ for each vertex in G, h_G is obtained by L_2 normalization of $\sum_{v \in V(G)} f_G(v)$. Let $\mathcal{H} = \{h_G \mid G \in \mathcal{D}\}$ denote the set of h_G vectors.

B. Computing $\mathcal{H}$ using Map-Reduce

Computation of h_G vectors for the graphs as discussed above is the most compute intensive part in our algorithm.

In order to scale our algorithm to large graph collection, we computed h_G vectors using Map-Reduce on Hadoop cluster.

In the map phase, there are n map jobs, one for each vertex of G. Each map job computes vector $f_G(v)$ for its designated vertex v. For this a BFS from a designated vertex v is first performed in G and the induced subgraph H_v is computed as given in Algorithm 1. The map job inspects all subsets of l vertices in H_v and emits $\langle graphlet\_id, 1\rangle$ for each graphlet detected. Here $graphlet\_id$ is the unique number assigned to each graphlet in D_l. The map job is implemented as multithreaded. In the reduce phase, there is one reduce job deployed for each $graphlet\_id$. The reduce job updates the corresponding component of vector h_G based on the sequence of 1s it received.

Simultaneous Map-Reduce jobs are run in the cluster for all graphs in the collection $\mathcal{D}$. The resulting vector collection $\mathcal{H}$ is subsequently used to build the LSH data structure.

C. Building Nearest Neighbor Data Structure on $\mathcal{H}$

We use the random projection based hashing given in [13] to build LSH data structure for vectors in $\mathcal{H}$. Our data structure consists of l hash tables $T_1, T_2, \ldots, T_w$, where value of w is a parameter to the algorithm. Every vector h_G in $\mathcal{H}$ is stored in each of the w hash tables. Table T_i uses hash function $g_i : R^{|D_l|} \rightarrow \{0,1\}^b$, where $|D_l|$ is the dimension of h_G vectors, such that $g_i(h_G)$ maps h_G to a b-bit binary number. Value of b is a parameter to our algorithm. Function g_i is defined as follows. Let $r_1, r_2, \ldots, r_b$ be b independent random vectors such that components of each vector r_j are drawn independently from the normal distribution $N(0,1)$. The i^{th} bit (from left) of $g_i(h_G)$ is 0 if the sign of $h_G^T r_i$ is negative and 1 otherwise. Table T_i has 2^b buckets and all vectors that are hashed to the same bucket are stored in the form of a list.

D. Computing k Nearest Neighbors

In order to compute the k nearest neighbors in collection $\mathcal{D}$ that are most similar to the given query graph Q, first we compute the vector h_Q for Q. We use the previous Map-Reduce implementation for this. Having computed h_Q, we computed pruned subset $\mathcal{D}'$ of $\mathcal{D}$ as follows [13]. Initially $\mathcal{D}' = \emptyset$. For each hash table $T_i \in \{T_1, \ldots, T_w\}$, consider the bucket $g_i(h_Q)$ and include the graphs hashed to $g_i(h_Q)$ in $\mathcal{D}'$. Our implementation is a simplified version of [13]. In [13], a cascade of approximate R near neighbor problems are solved for different values of R to solve the final approximate nearest neighbor problem with provable error guarantees. We use a single instance of LSH instead and it performed well in our experiments in identifying the nearest neighbors. After computing $\mathcal{D}'$, its elements are inspected and top k elements having maximum kernel value with the query graph Q are identified. The size of $\mathcal{D}'$ is usually much smaller than $\mathcal{D}$. Therefore, when size of $\mathcal{D}$ is very large, LSH based nearest neighbor search is expected to be faster than naive k nearest neighbor search that searches sequentially through the entire collection $\mathcal{D}$.

VI. Experimental Results

In our experiments, we used both real world datasets and synthetic datasets. For real world dataset we used AIDS antiviral screen public data from Developmental Therapeutics Program in NCI/NIH [1]. For synthetic data, we generated graphs under the Erdös-Rényi $G(n,p)$ model for varying values of n and p. The value of l (graphlet size) is fixed as 4 and BFS depth t was fixed as 4 in our implementation.

TABLE I. Preprocessing Time

n	Preprocessing time
50	14sec
100	18sec
150	19sec
200	31sec
250	49sec
300	1min17sec

Experimental Setup: We used a Hadoop cluster of 3 nodes, where each node is a 24-core machine. All computing cores are used in our multithreaded implementation. We *mapred.map.tasks.maximum* parameter, which decides the number of map tasks that need to be run simultaneously across the cluster, to be 30 based on our experiments.

Experiment 1: Preprocessing time

In this experiment, we measured time taken for computing vector h_G of random graph G drawn as per $G(n,p)$ model. Value of p was fixed as 0.01. Value of n varied from 50 to 300. The results are summarized in Table 1. The results indicate that even for 300 vertex graphs, the preprocessing time was reasonable. For graphs with smaller number of vertices (less than 150), preprocessing on a single compute node may be preferred in order to avoid the overhead of Hadoop.

Experiment 2: Similarity search in AIDS dataset

The AIDS dataset contains thousands of compounds for evidence of anti-HIV activity. The dataset contained 10,746 compounds in MDL SD format. The MDL SD format contains information about atom, bonds and connectivity and coordinates of molecules. The data was processed to obtain graph representation of the compounds. Hence size of our graph collection $\mathcal{D}$ was 10,746 and the number of vertices of these graphs varied from 50 to 200. Some of the compounds from the collection were chosen as query graph for finding k nearest neighbors. We chose $k = 7$. The LSH parameters were fixed as $b = 20$ and $w = 7$. In the querying phase, only computation of the h_Q vector for query graph Q was done in the Hadoop cluster and the subsequent LSH look up was performed in a single compute node. For all the queries, the exact match was present in the k nearest neighbours returned by the algorithm. Time taken for a query was within 16 seconds, which includes compute intensive h_Q vector computation.

Experiment 3: Similarity search in synthetic dataset

In this experiment, we generated 1500 random graphs of under $G(n,p)$ model with n = 100 and p = 0.01. To perform a query, we first selected a graph randomly from this collection and made few (about 4) random modification to it by removing and inserting few random edges. The resulting graph was used as the query graph Q. Parameters used were identical to Experiment 2. In all the queries, our algorithm was able to retrieve the original graph from which the query graph was

created. The query time including h_Q computation took about 15 seconds.

VII. Conclusion

We gave a nearest neighbor search algorithm to search in a graph collection for graphs similar to the query graph. This has several applications in areas like computational biology, drug design, social networks etc. We used a variant of graphlet kernel for similarity. Locality sensitive hashing was used to build the nearest neighbor search data structure. Using Hadoop and Map-Reduce paradigm, we were able to scale our implementation to large graph collections. Our experiments on real world and synthetic datasets showed that our algorithm was able to efficiently retrieve similar graphs from the collection. Future research directions would include incorporating graph attributes such as edge/vertex labels and edge/vertex weights in the search algorithm, investigating other graph kernels, extending to subgraph similarity search etc.

References

[1] Aids dataset. http://dtp.nci.nih.gov/docs/aids/aids_data.html.

[2] P. Indyk A. Gionis and R. Motwani. "similarity search in high dimensions via hashing". In *Proceedings of 25th Very Large Database Conference*, VLDB '99, 1999.

[3] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[4] Razvan C Bunescu and Raymond J Mooney. A shortest path dependency kernel for relation extraction. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 724–731. Association for Computational Linguistics, 2005.

[5] Moses S. Charikar. "similarity estimation techniques from rounding algorithms". In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, STOC, 2002.

[6] Frédéric Desobry, Manuel Davy, and William J Fitzgerald. A class of kernels for sets of vectors. In *ESANN*, pages 461–466. Citeseer, 2005.

[7] Holger Fröhlich, Jörg K Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the 22nd international conference on Machine learning*, pages 225–232. ACM, 2005.

[8] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pages 129–143. Springer, 2003.

[9] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[10] Huahai He and A.K. Singh. Closure-tree: An index structure for graph queries. In *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, pages 38–38, April 2006.

[11] Shohei Hido and Hisashi Kashima. A linear-time graph kernel. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 179–188. IEEE, 2009.

[12] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158–167. ACM, 2004.

[13] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[14] J.Ramon and T.Gartner. Expressivity versus efficency of graph kernels. 2003.

[15] Hisashi Kashima and Akihiro Inokuchi. Kernels for graph classification. In *ICDM Workshop on Active Mining*, volume 2002. Citeseer, 2002.

[16] Arijit Khan, Nan Li, Xifeng Yan, Ziyu Guan, Supriyo Chakraborty, and Shu Tao. Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 901–912. ACM, 2011.

[17] Risi Kondor and Tony Jebara. A kernel between sets of vectors. In *ICML*, volume 20, page 361, 2003.

[18] Sauro Menchetti, Fabrizio Costa, and Paolo Frasconi. Weighted decomposition kernels. In *Proceedings of the 22nd international conference on Machine learning*, pages 585–592. ACM, 2005.

[19] Misael Mongiovi, Raffaele Di Natale, Rosalba Giugno, Alfredo Pulvirenti, Alfredo Ferro, and Roded Sharan. Sigma: a set-cover-based inexact graph matching algorithm. *Journal of bioinformatics and computational biology*, 8(02):199–218, 2010.

[20] Michel Neuhaus and Horst Bunke. Edit distance based kernel functions for attributed graph matching. In *Graph-Based Representations in Pattern Recognition*, pages 352–361. Springer, 2005.

[21] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H Pham, Jafar M Al-Kofahi, and Tien N Nguyen. Graph-based mining of multiple object usage patterns. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 383–392. ACM, 2009.

[22] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[23] Jan Ramon and Thomas Gärtner. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

[24] Dennis Shasha, Jason TL Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–52. ACM, 2002.

[25] Nino Shervashidze and Karsten M Borgwardt. Fast subtree kernels on graphs. In *Advances in Neural Information Processing Systems*, pages 1660–1668, 2009.

[26] Nino Shervashidze, Tobias Petri, Kurt Mehlhorn, Karsten M Borgwardt, and SVN Vishwanathan. Efficient graphlet kernels for large graph comparison. In *International conference on artificial intelligence and statistics*, pages 488–495, 2009.

[27] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. Efficient subgraph matching on billion node graphs. *Proceedings of the VLDB Endowment*, 5(9):788–799, 2012.

[28] Yuanyuan Tian and Jignesh M Patel. Tale: A tool for approximate large graph matching. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 963–972. IEEE, 2008.

[29] SVN Vishwanathan and Alexander Johannes Smola. Fast kernels for string and tree matching. *Kernel methods in computational biology*, pages 113–130, 2004.

[30] Xiaohong Wang, Aaron Smalter, Jun Huan, and Gerald H Lushington. G-hash: towards fast kernel-based similarity search in large graph databases. In *Proceedings of the 12th international conference on extending database technology: advances in database technology*, pages 472–480. ACM, 2009.

[31] Xifeng Yan, Philip S Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 335–346. ACM, 2004.

[32] Shijie Zhang, Shirong Li, and Jiong Yang. Gaddi: distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 192–203. ACM, 2009.

[33] Shijie Zhang, Jiong Yang, and Wei Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *Proceedings of the VLDB Endowment*, 3(1-2):1185–1194, 2010.

[34] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. Graph similarity search with edit distance constraint in large graph databases. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1595–1600. ACM, 2013.