Subgraph Similarity Search in Large Graphs

Kanigalpula Samanvi

Dept. of Computer Science and Engineering
Indian Institute of Technology Hyderabad, India
Email: cs13m1001@iith.ac.in

Naveen Sivadasan

TCS Innovation Labs Hyderabad, India
Email: naveen@atc.tcs.com

***Abstract*—One of the major challenges in applications related to social networks, computational biology, collaboration networks, etc., is to efficiently search for similar patterns in their underlying graphs. These graphs are typically noisy and contain thousands of vertices and millions of edges. In many cases, the graphs are unlabeled and the notion of similarity is also not well defined. We study the problem of searching an induced subgraph in a large target graph that is most similar to the given query graph. We assume that the query graph and target graph are undirected and unlabeled. We use graphlet kernels to define graph similarity. Our algorithm maps topological neighborhood information of vertices in the query and target graphs to vectors and these local information are combined to find global similarity. We conduct experiments on several real world networks and we show that our algorithm is able to detect highly similar matches when queried in these networks. Our implementation takes about one second to find matches on graphs containing thousands of vertices and million edges, excluding the time for one time pre-processing. Computationally expensive parts of our algorithm can be further scaled to standard parallel and distributed frameworks.**

Keywords–Similarity Search; Subgraph Similarity Search; Graph Kernel; Nearest Neighbors Search.

I. INTRODUCTION

Similarity based graph searching has attracted considerable attention in the context of social networks, road networks, collaboration networks, software testing, computational biology, molecular chemistry, etc. In these domains, underlying graphs are large with tens of thousands of vertices and millions of edges. Subgraph searching is fundamental to the applications, where occurrence of the query graph in the large target graph has to be identified. Searching for exact occurrence of an induced subgraph isomorphic to the query graph is known as the subgraph isomorphism problem, which is known to be NP-complete for undirected unlabeled graphs.

Presence of noise in the underlying graphs and need for searching 'similar' subgraph patterns are characteristic to these applications. For instance, in computational biology, the data is noisy due to possible errors in data collection and different thresholds for experiments. In object-oriented programming, querying typical object usage patterns against the target object dependency graph of a program run can identify deviating locations indicating potential bugs [1]. In molecular chemistry, identifying similar molecular structures is a fundamental problem. Searching for similar subgraphs plays a crucial role in mining and analysis of social networks. Subgraph similarity searching is therefore more natural in these settings in contrast to exact search. In subgraph similarity search problem, induced subgraph of the target graph that is 'most similar' to the query graph has to be identified, where similarity is defined using some distance function. Quality of the solution and computational efficiency are two major challenges in these search problems. In this work, we assume that both the underlying graph and query graph are unlabeled and undirected.

Most applications work with a distance metric to define similarity between two entities (graphs in our case). Popular distance metrics include Euclidean distance, Hamming distance, Edit distance, Kernel functions [2]–[5], etc. We use graph kernel functions to define graph similarity.

Kernels are symmetric functions that map pairs of entities from a domain to real values which indicate their similarity. Kernels that are positive definite not only define similarity between pairs of entities but also allow implicit mapping of objects to a high-dimensional feature space and operating on this space without requiring to compute explicit mapping of objects in the feature space. Kernels implicitly yield inner products between the feature vectors without explicit computation of the same in feature space. This is usually computationally cheaper than explicit computation. This approach is usually referred to as the kernel trick or kernel method. Kernel methods have been widely applied to sequence data, graphs, text, images, videos, etc., as many of the standard machine learning algorithms including support vector machine (SVM) and principle component analysis (PCA) can directly work with kernels.

Kernels have been successfully applied in the past in the context of graphs [6]–[8]. There are several existing graph kernels based on various graph properties, such as random walks in the graphs [9][10], cyclic patterns [11], graph edit distance [12], shortest paths [13][14], frequency of occurrences of special subgraphs [15]–[17] and so on.

Graphlet kernels are defined based on occurrence frequencies of small induced subgraphs called graphlets in the given graphs [18]. Graphlet kernels have been shown to provide good SVM classification accuracy in comparison to random walk kernel and shortest path kernel on different datasets including protein and enzyme data [18]. Graphlet kernels are also of theoretical interest. It is known that under certain restricted settings, if two graphs have distance zero with respect to their graphlet kernel value then they are isomorphic [18]. Improving the efficiency of computing graphlet kernel is also studied in [18]. Graphlet kernel computation can also be scaled to parallel and distributed setting in a fairly straight forward manner. In our work, we use graphlet kernels to define graph similarity.

 ISBN: 978-1-61208-486-2

A. Related Work

Similarity based graph searching has been studied in the past under various settings. In many of the previous works, it is assumed that the graphs are labeled. In one class of problems, a large database of graphs is given and the goal is to find the most similar match in the database with respect to the given query graph [19]–[24]. In the second class, given a target graph and a query graph, subgraph of the target graph that is most similar to the query graph needs to be identified [25]–[28]. Different notions of similarity were also explored in the past for these classes of problems.

In [29], approximate matching of query graph in a database of graphs is studied. The graphs are assumed to be labeled. Structural information of the graph is stored in a hybrid index structure based on B-tree index. Important vertices of a query graph are matched first and then the match is extended progressively. In [30], graph similarity search on labeled graphs from a large database of graphs under minimum edit distance is studied. In [25], algorithm for computing top-k approximate subgraph matches for a given query graph in a large labeled target graph is given. In this work, the target graph is converted into a set of multi-dimensional vectors based on the labels in the vertex neighborhoods. Only matches above a user defined threshold are computed. With higher threshold values, the match is a trivial vertex to vertex label matching. In [26], algorithm NeMa was proposed which uses a combination of label similarity and local structural similarity to search for subgraph similar to query graph in large labeled graphs. Their query time is proportional to the product of number of vertices of the query and target graph. Subgraph matching in a large target graph for graphs deployed on a distributed memory store was studied in [27]. In [28], efficient distributed subgraph similarity search to retrieve matches whose number of missing edges is below a given threshold is studied. It looks for exact matching and not similarity matching. Though different techniques were studied in the past for the problem of similarity searching in various settings, to the best of our knowledge, little work has been done on subgraph similarity search on large unlabeled graphs. In many of the previous works, either the vertices are assumed to be labeled or the graphs they work with are small with hundreds of vertices.

B. Our Contribution

We consider undirected graphs with no vertex or edge labels. We use graphlet kernel to define similarity between graphs. We give a subgraph similarity matching algorithm that takes as input a large target graph and a query graph and identifies an induced subgraph of the target graph that is most similar to the query graph with respect to the graphlet kernel value.

In our algorithm, we first compute vertex labels for vertices in both query and target graph. These labels are vectors in some fixed dimension and are computed based on local neighborhood structure of vertices in the graph. Since our vertex labels are vectors, unlike many of the other labeling techniques, our labeling allows us to define the notion of similarity between vertex labels of two vertices to capture the topological similarity of their corresponding neighborhoods in the graph. We build a nearest neighbor data structure for vertices of the target graph based on their vertex labels. Computing vertex label for target graph vertices and building the nearest neighbor data structure are done in the pre-processing phase. Using nearest neighbor queries on this data structure, vertices of the target graph that are most similar to the vertices of the query graph are identified. Using this smaller set of candidate vertices of target graph, a seed match is computed for the query graph. Using this seed match as the basis, our algorithm computes the final match for the full query graph. By using vertex level vector labels based on graphlet distribution in the local neighborhood of vertices, we are able to extend the power of graphlet kernels, which was shown to perform well for graph similarity search on smaller graphs, to subgraph similarity search on much larger graphs.

We study the performance of our algorithm on several real life datasets including facebook network, google plus network, youtube network, road network, amazon network provided by the Stanford Large Network Dataset Collection (SNAP) [31] and Digital Bibliography & Library Project (DBLP) network [32]. We conduct number of experimental studies to measure the search quality and run time efficiency. For instance, while searching these networks with their communities as query graphs, the computed match and the query graph has similarity score close to 1, where 1 is the maximum possible similarity score. In about 30% of the cases, our algorithm is able to identify the exact match and in about 80% of the cases, vertices of exact match are present in the pruned set computed by the algorithm. We validate our results by showing that similarity scores between random subgraphs and similarity scores between random communities in these networks are significantly lower. In other words, similarity score obtained by chance is significantly lower. We also query communities across networks and in noisy networks and obtain matches with significantly high similarity scores. We use our algorithm to search for dense subgraphs and identify subgraphs with significantly high density. We also conduct experiments to compare performance of our algorithm with NeMa [26], which is a subgraph similarity search algorithm that uses both structural and label similarity. We use graphs with uniform label for this purpose.

Computationally expensive parts of our algorithm can be easily scaled to standard parallel and distributed computing frameworks such as map-reduce. Most of the networks in our experiments have millions of edges and thousands of vertices. We use multi-threaded implementation for the one time pre-processing phase. Single threaded implementation of our search algorithm takes close to one second. This excludes time taken by the pre-processing phase.

C. Paper Organization

In Section II, we present the preliminaries including graphlet kernels and the problem statement. In Section III, we present the details of the vertex labeling technique. In Section IV, we present the details of our algorithm including the pre-processing phase and the matching phase. In Section V, we present the experimental results. In Section VI, we present our conclusions and directions for future research.

II. PRELIMINARIES

Graph is an ordered pair $G = (V, E)$ comprising a set V of vertices and a set E of edges. To avoid ambiguity, we also use $V(G)$ and $E(G)$ to denote the vertex and edge set. We consider only undirected graphs with no vertex or edge labels.

 ISBN: 978-1-61208-486-2

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 $b : V(G_1) \rightarrow V(G_2)$ 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 (Subgraph Isomorphism). Graph G_1 is isomorphic to a subgraph of graph G_2, if there is an induced subgraph of G_2 that is isomorphic to G_1.

DEFINITION 3 (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/similarity function between graphs.

DEFINITION 4 (Subgraph Similarity Searching). Given graphs G_1 and G_2, determine a subgraph $G^* \subseteq G_1$ that is closest to G_2 with respect to a given distance/similarity function between graphs.

A. Graphlet Kernel

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.

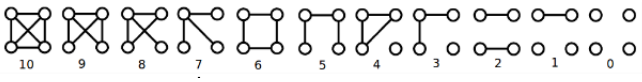

Figure 1. Set of all non isomorphic graphlets of size 4

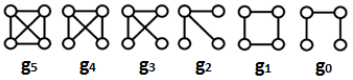

Figure 2. Non isomorphic connected graphlets of size 4

DEFINITION 5 (Graphlet Vector). For a given l, the graphlet vector f_G for a given graph G is a frequency vector of dimension $|D_l|$ where its ith component corresponds to the number of occurrences of the ith graphlet of D_l in G. We assume the graphlet vector f_G to be normalized by the L_2 norm $||f_G||_2$.

If graphs G and G' are isomorphic then clearly their corresponding graphlet vectors f_G and $f_{G'}$ are identical. But the reverse need not be true in general. But, 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 f_G is identical to $f_{G'}$ [18]. The conjecture has been verified for $n \leq 11$ [18]. Kernels based on similarity of graphlet vectors provide a natural way to express similarity of underlying graphs.

DEFINITION 6 (Graphlet Kernel). Given two graphs G and G', let f_G and $f_{G'}$ be their corresponding graphlet frequency vectors with respect to size l graphlets for some fixed l. The graphlet kernel value $K(G, G')$ is defined as the dot product of f_G and $f_{G'}$. That is, $K(G, G') = f_G^T f_{G'}$

Graphlet vectors are in fact an explicit embedding of graphs into a vector space whose dimension is $|D_l|$ if size l graphlets are used. Graphlet kernels have been shown to give better classification accuracies in comparison to other graph kernels like random walk kernel and shortest path kernel for certain applications [18]. Values of $K(G, G') \in [0, 1]$ and larger values of $K(G, G')$ indicate higher similarity between G and G'. In this work, we use kernel function $K(G, G')$ to represent similarity between graphs G and G'. Exact problem statement is given below.

PROBLEM STATEMENT. Let $K(\cdot, \cdot)$ be graphlet kernel based on size l graphlets for some fixed l. Given a large connected graph G of size n and a connected query graph Q of size n_q with $n > n_q$, find a subset V^* of vertices in G such that its induced subgraph G^* in G maximizes $K(Q, G^*)$.

III. Graphlet Vector Based Vertex Labeling

Computing vertex labels that capture topological neighborhood information of corresponding vertices in the graph and comparing vertex neighborhoods using their labels is crucial in our matching algorithm. Our vertex labels are graphlet vectors of their corresponding neighborhood subgraphs.

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_v, the graphlet vector corresponding to the graph H_v, with respect to size l graphlets for some fixed l. We note that for defining the graphlet vector f_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.

For each vertex v of the graph, its vertex label is given by the vector f_v. Given vertex labels f_u and f_v for vertices u and v, we denote by $s(u, v)$ the similarity between labels of f_u and f_v, given by their dot product as

$$s(u, v) = f_u^T f_v \tag{1}$$

Values of $s(u, v) \in [0, 1]$ and larger values of $s(u, v)$ indicate higher topological similarity between neighborhoods of vertices u and v. Computing the vertex labels of the target graph is done in the pre-processing phase. Implementation details of the vertex labeling algorithm are discussed in the next section.

IV. Our Algorithm

Our subgraph similarity search algorithm has two major phases: one time pre-processing phase and the query graph matching phase. Each of these phases comprise sub-phases as given below. Details of each of these subphases is discussed in the subsequent sections.

A. Pre-processing Phase:

This phase has two subphases:

1) In this phase, vertex labels f_v of all the vertices of the target graph G are computed.
2) k-d tree based nearest neighbor data structure on the vertices of G using their label vectors f_v is built.

B. Matching Phase: This phase is further divided into four subphases:

1) Selection Phase: In this phase, vertex labels f_v for vertices of the query graph Q are computed first. Each vertex u of the query graph then selects a subset of vertices from the target graph G closest to u based on their Euclidean distance.
2) Seed Match Generation Phase: In this phase, a one to one mapping of a subset of query graph vertices to target graph vertices is obtained with highest overall similarity score. Subgraph induced by the mapped vertices in the target graph is called the seed match. The seed match is obtained by solving a maximum weighted bipartite matching problem.
3) Match Growing Phase: The above seed match is used as a basis to compute the final match for Q.
4) Match Completion Phase: This phase tries to match those vertices in Q that are still left unmatched in the previous phase.

A. Pre-processing Phase

1) Computation of vertex labels f_v: In this phase, vertex label f_v for each vertex v of the target graph G is computed first. To compute f_v, we require parameter values t and l. These two values are assumed to be provided as parameters to the search algorithm. For each vertex v, a breadth first traversal of depth t is performed starting from v to obtain the depth t neighborhood $N(v)$ of v. The graph H_v induced by the vertex set $N(v)$ is then used to compute the graphlet vector f_v as given in [33]. The algorithm is given in Figure 3.

Major time taken by the pre-processing phase is for computing the graphlet vector for H_v. In [18], methods to improve its efficiency including sampling techniques are discussed. We do not make use of sampling technique in our implementation. We remark that finding the graphlet frequencies can easily be scaled to parallel computing frameworks or distributed computing frameworks such as map-reduce.

Algorithm 1

Input: Graph G, vertex v, BFS depth t, graphlet size l
Output: Label vector f_v

1: Run BFS on G starting from v till depth t. Let $N(v)$ be the set of vertices visited including v.
2: Identify the induced subgraph H_v of G induced by $N(v)$.
3: Compute graphlet vector f_v for graph H_v.
4: Normalize f_v by $||f_v||_2$.
5: **return** f_v

Figure 3. Algorithm for computing label f_v for vertex v

2) Nearest neighbor data structure on f_v: After computing vertex labels for G, a nearest neighbor data structure on the vertices of G based on their label vectors f_v is built. We use k-d trees for nearest neighbor data structure [34]. k-d trees are known to be efficient when dimension of vectors is less than 20 [34]. Since the typical graphlet size l that we work with are $3, 4$ and 5, the dimension of f_v (which is $|D_l|$) does not exceed 10.

B. Matching Phase

In the following we describe the three subphases of matching phase.

1) Selection Phase: The vertex labels f_v for all vertices of the query graph Q are computed first using Algorithm 1. Let R_v denote the set of k vertices in G that are closest to v with respect to the Euclidean distance between their label vectors. In our experiments, we usually fix k as 10. For each vertex v of Q, we compute R_v by querying the k-d tree built in the pre-processing phase. Let R denote the union of R_v for each vertex v of the n_q vertices of Q. For the subsequent seed match generation phase, we will only consider the vertex subset R of G. Clearly size of R is at most $k.n_q$ which is typically much smaller than the number of vertices in G.

2) Seed Match Generation Phase: In this phase, we obtain a one to one mapping of a subset of vertices of the query graph Q to the target graph G with highest overall similarity score. We call the subgraph induced by the mapped vertices in G as the seed match. To do this, we define a bipartite graph $(V(Q), R)$ with weighted edges, where one part is the vertex set $V(Q)$ of the query graph Q and the other part is the pruned vertex set R of G obtained in the previous step. The edges of the bipartite graph and their weights are defined as follows. Each vertex v in the part $V(Q)$ is connected to every vertex w in $R_v \subseteq R$, where R_v is the set of k nearest neighbors of v in G as computed in the previous step.

The weight $\lambda(v, w)$ for the edge (v, w) is defined in the following manner. Let $0 < \alpha < 1$ be a fixed scale factor which is provided as a parameter to the search algorithm. We recall that vertex v belongs to query graph Q and vertex w belongs to target graph G and $s(v, w)$ given by equation (1) denote the similarity between their label vectors f_v and f_w. Let V_w denote the neighbors of vertex w in graph G including w. Let Q' denote the subset of $V(Q)$ excluding v such that each vertex in Q' is connected to at least one vertex in V_w in the bipartite graph $(V(Q), R)$. In particular, for each vertex $u \in Q'$, let $s(u)$ denote the maximum $s(u, z)$ value among all its neighbors z in V_w in the bipartite graph. Now the weight $\lambda(v, w)$ for the edge (v, w) of the bipartite graph is given by

$$\lambda(v,w) = \frac{\left(s(v,w)^\alpha + \sum_{u \in Q'} s(u)^\alpha\right)^{1/\alpha}}{(|Q'|+1)} \qquad (2)$$

We now solve maximum weighted bipartite matching on this graph to obtain a one to one mapping between a subset of vertices of Q and the vertices of G. Defining edge weights $\lambda(v, w)$ to edge (v, w) in the bipartite graph in the above fashion not only takes into account the similarity value $s(v, w)$, but also the strength of similarity of neighbors of w in G to remaining vertices in the query graph Q. By assigning edge weights as above, we try to ensure that among two vertices in G with equal similarity values to a vertex in Q, the vertex whose neighbors in G also have high similarity to vertices in Q is preferred over the other in the final maximum weighted bipartite matching solution.

Let M denote the solution obtained for the bipartite matching. Let Q_M and G_M respectively denote the subgraphs

 ISBN: 978-1-61208-486-2

induced by the subset of matched vertices from graphs Q and G under the matching M. The connectivity of Q_M and G_M may differ. For instance, the number of connected components in G_M and Q_M could differ. Therefore, we do not include all the vertices of G_M in the seed match. Instead, we use the largest connected component of G_M as a seed solution. That is, let $S_G \subset V(G)$ denote the subset of vertices in G_M corresponding to a maximum cardinality connected component. Let S_Q denote their corresponding mapped vertices in Q_M. We call S_G as a seed match. The pseudo code for seed match computation is given in Algorithm 2.

Algorithm 2

Input: Vertex sets $V(Q)$, R and R_v for each $v \in V(Q)$ and their labels f_v, parameter α

Output: S_G and S_Q

1: Construct bipartite graph $(V(Q), R)$ with edge weights given by $\lambda(v, w)$.
2: Compute maximum weighted bipartite matching M on $(V(Q), R)$
3: Let Q_M and G_M respectively denote the subgraphs induced by vertices from Q and G in the matching M.
4: Compute largest connected component in G_M. Let S_G denote the vertices in that component. Let S_Q denote its mapped vertices in Q_M under the bipartite matching M.
5: **return** S_G and S_Q

Figure 4. Computing seed match S_G in G and its mapped vertices S_Q in Q

3) Match Growing Phase: After computing the seed match S_G in G and its mapped vertices S_Q in Q, we use this seed match as the basis to compute the final match. The final solution is computed in an incremental fashion starting with empty match. In each iteration, we include a new pair of vertices (v, w) to the solution, where v and w belongs to G and Q respectively. In order to do this, we maintain a list of candidate pairs and in each iteration, we include a pair with maximum similarity value $s(v, w)$ to the final solution. We use a max heap to maintain the candidate list. The candidate list is initialized with the mapped pairs between S_G and S_Q as obtained in the previous phase. Thus, the heap is initialized by inserting each of these mapped pairs (v, w) with corresponding weight $s(v, w)$.

We recall that the mapped pairs obtained from previous phase have stronger similarity with respect to the modified weight function $\lambda(v, w)$. Higher value of $\lambda(v, w)$ indicates that not only $s(v, w)$ is high but also their neighbors share high $s()$ value. Hence, they are more preferred in the solution over other pairs with similar $s()$ value. By initializing the candidate list with these preferred pairs, the matching algorithm tries to ensure that the incremental solution starts with these pairs first and other potential pairs are considered later. Also, because of the heap data structure, remaining pairs are considered in the decreasing order of their similarity value. Moreover, as will be discussed later, the incremental matching tries to ensure that the partial match in G constructed so far is connected. For this, new pairs that are added to the candidate list are chosen from the neighborhood of the partial match between G and Q.

The incremental matching might still match vertex pairs with low $s()$ value if they are available in the candidate list. Candidate pairs with low $s()$ values should be treated separately as there could be genuine pairs with low $s()$ value. For instance, consider boundary vertices of an optimal subgraph match in G. Boundary vertices are also connected to vertices outside the matched subgraph. Hence, their local neighborhood structure is different from their counterpart in the query graph. In other words, their corresponding graphlet vectors can be very dissimilar and their similarity value $s()$ can be very low even though they are expected to be matched in the final solution. In order to find such genuine pairs, we omit pairs with similarity value below some fixed threshold h_1 in this phase and such pairs are handled in the next phase.

In each iteration of the incremental matching, a pair (v, w) with maximum $s(v, w)$ value is removed from the candidate heap and added to the final match. After this, the candidate list is modified as follows. We recall that v and w belong to G and Q respectively. We call a vertex unmatched if it is not yet present in the final match. The algorithm maintains two invariants: (a) the pairs present in the candidate list are one to one mappings and (b) a query vertex that enters the candidate list will stay in the candidate list (possibly with multiple changes to paired partner vertex) until it is included in the final match. Let U_v denote the unmatched neighbors of v in G that are also not present in the candidate list. Let U_w denote the unmatched neighbors w in Q. For each query vertex y in U_w, let x be a vertex in U_v with maximum similarity value $s(x, y)$. We add (x, y) to the candidate list if y is absent in the list and $s(x, y) \geq h_1$. If y is already present in the candidate list, then replace the current pair for y with (x, y) if $s(x, y)$ has a higher value. The incremental algorithm is given in Algorithm 3. The candidate list modification is described in Algorithm 4.

Algorithm 3

Input: Seed match S_G and its mapped vertices S_Q, threshold h_1

Output: Partial match F

1: Initialize F to empty set.
2: Initialize the candidate list max heap with mapped pairs (v, w) of the seed match where $s(v, w) \geq h_1$.
3: **while** candidate list is not empty **do**
4: Extract maximum weight candidate match (v, w)
5: Add (v, w) to F
6: **updateCandidateList**(candidate list, (v, w), h_1,F)
7: **end while**
8: **return** F

Figure 5. Incremental Matching Algorithm.

4) Match Completion Phase: In this phase, vertices of the query graph Q that are left unmatched in the previous phase due to similarity values below the threshold h_1 are handled. Typically, boundary vertices of the final matched subgraph in G remain unmatched in the previous phase. As discussed earlier, this is because, such boundary vertices in G and their matched partners in Q have low $s()$ value as their local neighborhood topologies vastly differ. Hence, using neighborhood similarity for such pairs is ineffective. To handle them, we try to match unmatched query vertices with unmatched neighbors of the current match F in G. Since the similarity function $s()$ is ineffective here, we use a different similarity function to compare potential pairs. Let X denote

 ISBN: 978-1-61208-486-2

Algorithm 4

Input: candidate list, (v, w), h_1 and F

1: Compute U_v which is the set of unmatched neighbors of v in G that are also not present in candidate list.
2: Compute U_w which is the set of unmatched neighbors of w in Q.
3: **for all** vertex $y \in U_w$ **do**
4: Find $x \in U_v$ with maximum $s(x, y)$ value.
5: **if** y does not exist in candidate list **then**
6: Include (x, y) in the candidate list if $s(x, y) \geq h_1$.
7: **else**
8: Replace existing pair for y in the candidate list with (x, y) if $s(x, y)$ has higher value.
9: **end if**
10: **end for**

Figure 6. Algorithm for updateCandidateList

the set of unmatched neighbors of the current match F in G. Let Y denote the set of unmatched query vertices. Let $v \in X$ and let $w \in Y$. We define the similarity $c(v, w)$ as follows. Let Z_v denote the matched neighbors of v in target graph G and let Z_w denote the matched neighbors of w in query graph Q. Let Z'_v denote the matched partners of Z_v in Q. We now define $c(v, w)$ using the standard Jaccard similarity coefficient as

$$c(v, w) = \frac{|Z'_v \cap Z_w|}{|Z'_v \cup Z_w|} \quad (3)$$

We use a fixed threshold h_2 that is provided as parameter to the algorithm. We now define a bipartite graph (X, Y) with edge weights as follows. For each $(v, w) \in X \times Y$, insert an edge (v, w) with weight $c(v, w)$ in the bipartite graph if $c(v, w) \geq h_2$. Compute maximum weighted bipartite graph matching on this bipartite graph and include the matched pairs in the final solution F. In our experiments, size of Y (number of unmatched query graph vertices) is very small. The pseudo code is given in Algorithm 5.

Algorithm 5

Input: Partial match F and threshold h_2
Output: Final match F

1: Let X denote the set of unmatched neighbors of the match F in G.
2: Let Y denote the set of unmatched vertices in Q.
3: Construct bipartite graph (X, Y) by introducing all edges (v, w) with edge weight $c(v, w)$ if $c(v, w) \geq h_2$.
4: Compute maximum weighted bipartite matching.
5: Add each of these matches to F
6: **return** F

Figure 7. Match completion algorithm

We remark that our searching algorithm finds the matched subset of vertices in G and also their corresponding mapped vertices in the query graph Q.

V. Experimental Results

In this section, we conduct experiments on various real life graph datasets [31] including social networks, collaboration networks, road networks, youtube network, amazon network and on synthetic graph datasets. We also conduct experiments to compare performance of our algorithm with NeMa [26], which is a subgraph similarity search algorithm that uses both label similarity and structural similarity to find subgraph similar to query graph in large labeled graphs.

A. Experimental Datasets

Social Networks: We conduct experiments on facebook and google plus undirected graphs provided by Stanford Large Network Dataset Collection (SNAP) [31]. Facebook graph contains around 4K vertices and 88K edges. In this graph, vertices represent anonymized users and an undirected edge connects two friends. google plus graph contains 107K vertices and 13M edges. google plus graph also represents users as vertices and an edge exists between two friends. The dataset also contains list of user circles (user communities), where user circle is specified by its corresponding set of vertices. We use these user circles as query graphs and they are queried against the entire facebook network. We also query facebook circles against google plus network to find similar circles across networks. We also experiment querying facebook circles against facebook network after introducing random noise to the facebook network.

DBLP Collaboration Network: We use the DBLP collaboration network downloadable from [32]. This network has around 317K vertices and 1M edges. The vertices of this graph are authors who publish in any conference or journal and an edge exists between any two co-authors. All the authors who contribute to a common conference or a journal form a community. The dataset provides a list of such communities by specifying its corresponding set of vertices. We use such communities as query graphs.

Youtube Network: Youtube network is downloaded from [31]. Network has about 1M vertices and 2M edges. Vertices in this network represent users and an edge exists between two users who are friends. In youtube, users can create groups in which other users can join. The dataset provides a list of user groups by specifying its corresponding set of vertices. We consider these user-defined groups as our query graphs.

Road Network: We use the road network of California obtained from [31] in our experiments. This network has around 2M vertices and 3M edges. Vertices of this network are road endpoints or road intersections and the edges are the roads connecting these intersections. We use randomly chosen subgraphs from this network as query graphs.

Amazon Network: Amazon network is a product co-purchasing network downloaded from [31]. This network has around 334K vertices and 925K edges. Each vertex represents a product and an edge exists between the products that are frequently co-purchased [31]. All the products under a certain category form a product community. The dataset provides a list of product communities by specifying its corresponding set of vertices. We use

 ISBN: 978-1-61208-486-2

product communities as query graphs and we query them against the amazon network.

The statistics of the datasets used are listed in Table I.

TABLE I. DATASET STATISTICS

DataSet	#vertices	#edges
Facebook	4039	88234
Google Plus	107614	13673453
DBLP	317080	1049866
Amazon	334863	925872
Youtube	1134890	2987624
Road Network	1965206	2766607

B. Experimental Setup

All the experiments are carried out on a 32 core 2.60GHz Intel(R) Xeon(R) server with 32GB RAM. The server has Ubuntu 14.04 LTS. Our implementation uses Java 7.

The computationally most expensive part of our algorithm is the computation of vector labels for all vertices of a graph. The pre-processing phase that computes label vectors for each vertex of the graph is multi-threaded and thus executes on all 32 cores. Similarly, in the matching phase, computing label vectors for all vertices of the query graph is also multi-threaded and uses all 32 cores. Remaining phases use only a single core.

C. Results

To evaluate the accuracy of the result obtained by our similarity search algorithm, we compute the graphlet kernel value $K(Q, G^*)$ between the query graph Q and the subgraph G^* of G induced by the vertices V^* of the final match F in G. We use this value to show the similarity between the query graph and our obtained match and we refer to this value as *similarity score* in our experiments. We recall that similarity score lies in the range $[0, 1]$ where 1 indicates maximum similarity.

There are six parameters in our algorithm: (1) graphlet size l, (2) BFS depth t for vertex label computation, (3) value of k for the k nearest neighbors from k-d tree, (4) value of α in the edge weight function λ and (5) similarity thresholds h_1 for match growing phase and h_2 for match completion phase. In all our experiments we fix graphlet size l as 4. We performed experiments with different values of k, α, h_1 and h_2 on different datasets. Based on the results, we chose ranges for these parameters. The value of k is chosen from the range 5 to 10. Even for million vertex graphs, $k = 10$ showed good results. We fix scaling factor α to be 0.3 and the thresholds h_1 and h_2 to be 0.4 and 0.95 respectively.

Experiment 1: This experiment shows the effect of bfs depth t on the final match quality. We performed experiments with different values of t. We observed that after the depth of 2, there is very little change in the similarity scores of the final match. But as the depth increases the time to compute graphlet vectors also increases. Thus, the bfs depth t was taken to be 2 for most of our experiments. Table II shows the similarity scores of querying amazon communities on amazon network and and DBLP communities on DBLP collaboration network for different values of t. These results are averaged over 150 queries.

Experiment 2: For each of the datasets discussed earlier, we perform subgraph querying against the same network. For each network, we use the given communities as query graphs and measure the quality of the search result. That is, we query facebook communities against facebook network, DBLP communities against DBLP network, youtube groups against youtube network and amazon product communities against amazon network. For road network, we use randomly chosen induced subgraphs from the network as query graph. Second column of Table III shows the similarity score of the match. All the results are averages over 150 queries. The average community (query graph) size is around 100 for facebook, around 40 for DBLP, around 50 for youtube and around 300 for amazon. Query graphs for road network have about 500 vertices.

TABLE II. EXPERIMENT 1 : SIMILARITY SCORE VS. t

Dataset	t=2	t=3	t=4
Amazon	0.9999823	0.9999851	0.9999858
DBLP	0.9999942	0.9999896	0.9999917

To validate the quality of our solution, we do the following for each of the network. We compute the similarity score between random induced subgraphs from the same network. These random subgraphs contain 100 vertices. We also compute the similarity score between different communities from the same network. All results are averaged over 150 scores. Table III shows these results. Second column in the table shows the average similarity score between query graph and the computed match. The query graphs are the given communities. Third column in the table shows the average similarity score between random subgraphs. Fourth column shows the average similarity score between communities. The results show that the similarity score of our match close to 1 and is significantly better than scores between random subgraphs and scores between communities in the same network. For road network, the third column shows the average similarity between its query subgraphs.

TABLE III. EXPERIMENT 2 : SIMILARITY SCORES.

DataSet	Query graph & Final Match	Between Random Subgraphs	Between Communities
Facebook	0.944231	0.702286	0.787296
DBLP	0.975137	0.443763	0.6144779
Amazon	0.999982	0.663301	0.624756
Youtube	0.998054	0.311256	0.524779
Road Network	0.899956	0.770492	0.599620

Table IV shows the $\#exactMatches$ which is the number of queries that yielded the exact match out of the 150 queries (query graph is a subgraph of the network), and $\#inPruned$ - the percentage of queries where the vertices of the exact target match are present in the pruned subset of vertices R of target graph G obtained after the selection phase. Table IV shows that, for about 30% of the query graphs, our algorithm identifies the exact match. Also, for about 75% of the queries, vertices of the ideal match are present in our pruned set of vertices R in the target graph after selection phase.

Table V shows the timing results corresponding to *Experiment 2*. The timing information is only for the matching phase and it excludes the one time pre-processing phase. Here δ denotes time taken (in secs) to compute the label vectors for all vertices of the query graph and τ the time taken (in secs)

 ISBN: 978-1-61208-486-2

TABLE IV. EXPERIMENT 2 : EXACT MATCH STATISTICS

Dataset	$\#exactMatches$ out of 150 (percentage)	$\#inPruned$ (percentage)
Facebook	53 (35.3)	83
DBLP	47 (31.3)	82
Amazon	60 (40.0)	72

for the entire matching phase (including δ). We recall that the label vector computation is implemented as multi-threaded on 32 cores and the remaining part is executed as a single thread. It can be seen that the label vector computation is the computationally expensive part and the remaining phases take much lesser time.

TABLE V. EXPERIMENT 2 : TIMING RESULTS

DataSet	δ(in sec)	τ(in sec)
Facebook	0.213596	0.253706
DBLP	0.159492	0.777687
Amazon	0.199767	0.781500
Youtube	0.225131	0.989452
Road Network	0.216644	1.437619

Experiment 3: In all previous experiments, query graphs were induced subgraphs of the target network. In this experiment, we evaluate the quality of our solution when the query graph is not necessarily an induced subgraph of the target graph. For this, we conduct two experiments. In the first experiment, we use facebook communities as query graphs and query them against google plus network. To validate the quality of our solution, we measure the similarity score of the query graph with a random induced subgraph in the target graph with same number of vertices. In the second experiment, we create a modified facebook network by randomly removing 5% its original edges. We use this modified network as the target graph and query original facebook communities in this target graph. Here also, we validate the quality of our solution by measuring the similarity score for the query graph with a random induced subgraph of same number of vertices in the target graph. Table VI shows these results. Second column in the table shows the similarity score between query graph and match. Third column shows the score between query graph and a random subgraph. Values shown for both experiments are averaged over 150 scores. The results show that similarity score of our match is close to 1 and is significantly better than a match by chance.

TABLE VI. EXPERIMENT 3 : SIMILARITY SCORES

DataSet	Final Match	Random Subgraph
Google Plus	0.912241	0.600442
Facebook with random noise	0.933662	0.701198

Experiment 4: We use our matching algorithm to identify dense subgraphs in large networks. In particular, we search for dense subgraphs in DBLP and google plus networks. For this, we first generate dense random graphs using the standard $G(n,p)$ model with $n = 500$ and $p = 0.9$. We now use these random graphs as query graphs and query them against the DBLP and google plus networks. We use the standard definition of density ρ of a graph $H = (V, E)$ as

$$\rho = \frac{2|E|}{|V| * |V-1|} \in [0,1] \quad (4)$$

The average density of our random query graphs is 0.9. We queried these dense random graphs against DBLP and google plus networks. Table VII shows the results. Column 2 shows the similarity score between query graph and obtained match. Column 3 shows the density ρ for the obtained match. The results are averaged over 150 queries. Results show that the similarity score with matched result is close to 1 for google plus. For DBLP the score is close to 0.8 primarily because DBLP does not have dense subgraphs with about 500 vertices. Also, the density of the obtained match is close to that of the query graph, which is 0.9.

TABLE VII. EXPERIMENT 4 : DENSE SUBGRAPH MATCH RESULTS

DataSet	Similarity Score	ρ for the match
Google Plus	0.926670	0.812
DBLP	0.799753	0.730

Experiment 5 - Comparison with NeMa: We conducted experiments to compare performance of our algorithm with NeMa [26]. NeMa uses combination of label similarity and structural similarity to search similar subgraphs in large labeled graphs. NeMa was shown to find high quality matches efficiently compared to state-of-the-art graph querying algorithms. Similarity search using structural similarity alone is harder as label similarity helps in pruning the search space considerably. For comparing performance of NeMa with our algorithm, we considered query and target graphs with same label for all vertices, which is similar to unlabeled setting. In particular, we used YAGO and IMDB datasets for our experiments which were used also for experimental evaluation of NeMa in [26]. Both datasets were modified to make all vertices to have the same label and all edges unlabeled. IMDB (Internet Movie Database) dataset consists of relationships between movies, directors, producers and so on. YAGO entity relationship graph is a knowledge base containing information from Wikipedia, WordNet and GeoNames. IMDB dataset consists of about 3 million vertices and 11 million edges and YAGO dataset consists of about 13 million vertices and 18 million edges. Induced subgraphs from target graphs were used as query graphs. Query graph size was restricted to 7 vertices as in [26].

We considered only search time for comparison and excluded time taken for one time pre-processing/indexing from our comparison. For a single query, NeMa ran for more than 13 hours and aborted on these datasets. This is in contrast to fraction of a second that NeMa takes for similar queries in the labeled setting. For our algorithm, we considered 50 queries separately on IMDB and YAGO. For IMDB, average similarity score achieved by our algorithm was 0.91 and 41 out of 50 results were exact matches. For YAGO, average similarity score achieved was 0.89 and 37 out of 50 results were exact matches. Average search time in both cases was less than 0.5 seconds. Upon restricting target graphs to 1000 node induced subgraphs of IMDB and YAGO graphs, NeMa took 2.5 hours for searching. We finally used 100 node induced subgraph of IMDB graph as target graph for NeMa. For 50 queries, average search time for NeMa was 8 minutes and 23

out of 50 results were exact matches. For same experiment, our algorithm achieved average similarity score of 0.93 and 40 out of 50 results were exact matches. Average search time for our algorithm was 0.03 seconds.

D. Scalability

Computationally most expensive parts of our algorithm are the vertex label computation for vertices of query and target graphs. Since this is a one time pre-processing for the target graph, it can be easily scaled to a distributed framework using the standard map-reduce paradigm. Vertex label computation for each vertex can be a separate map/reduce job. Vertex label computation for query graph is performed for every search. This can also be parallelized using the standard OpenMP/MPI framework as each vertex label computation can be done in parallel. As shown in the experimental results, remaining phases take much lesser time even with serial implementation. Parts of them can also be parallelized to further improve the search efficiency. Our algorithm can also support dynamic setting since computation of vertex level label vectors uses only local structural information. Edge and vertex modifications can therefore affect only label vectors of vertices in their local neighborhood and these label vectors can be recomputed efficiently and the pre-computed index can be modified accordingly.

VI. Conclusions

In this paper, we propose an algorithm that performs subgraph similarity search on large undirected graphs solely based on structural similarity. In the pre-processing step, our algorithm computes multi-dimensional vector representation for vertices in the target graph based on graphlet distribution in their local neighborhood. These local topological information are then combined to find a target subgraph having highly similar global topology with the given query graph. We tested our algorithm on several large real world graphs and was shown to obtain high quality matches efficiently. Size of these graphs ranged from thousand vertices to million vertices. By using vertex level vector labels based on graphlet distribution in the local topological neighborhood of vertices, we are able to extend the power of graphlet kernels, which was shown to perform well for graph similarity search on smaller graphs, to subgraph similarity search on much larger graphs. Local nature of vector label pre-computation of vertices makes our algorithm amenable to parallelization and to handle dynamic setting. Efficient parallel/distributed implementations of label vector pre-computation and matching to handle massive graphs on billions of vertices, large query graphs and massive graph streams would be future work.

References

[1] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. 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. ACM, 2009, pp. 383–392.

[2] D. Haussler, "Convolution kernels on discrete structures," University of California at Santa Cruz, Tech. Rep., 1999.

[3] F. Desobry, M. Davy, and W. J. Fitzgerald, "A class of kernels for sets of vectors." in ESANN, 2005, pp. 461–466.

[4] R. Kondor and T. Jebara, "A kernel between sets of vectors," in ICML, vol. 20, 2003, pp. 361–368.

[5] S. Vishwanathan and A. J. Smola, "Fast kernels for string and tree matching," in Kernel methods in computational biology. MIT Press, 2004, pp. 113–130.

[6] S. Hido and H. Kashima, "A linear-time graph kernel," in ICDM. IEEE, 2009, pp. 179–188.

[7] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," Applied and Computational Harmonic Analysis, vol. 30, no. 2, 2011, pp. 129–150.

[8] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs," in Advances in Neural Information Processing Systems, 2009, pp. 1660–1668.

[9] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in Learning Theory and Kernel Machines. Springer, 2003, pp. 129–143.

[10] H. Kashima and A. Inokuchi, "Kernels for graph classification," in ICDM Workshop on Active Mining, 2002.

[11] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004, pp. 158–167.

[12] M. Neuhaus and H. Bunke, "Edit distance based kernel functions for attributed graph matching," in Graph-Based Representations in Pattern Recognition. Springer, 2005, pp. 352–361.

[13] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in ICDM. IEEE, 2005, pp. 74–81.

[14] R. C. Bunescu and R. 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. Association for Computational Linguistics, 2005, pp. 724–731.

[15] 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. ACM, 2005, pp. 225–232.

[16] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in First International Workshop on Mining Graphs, Trees and Sequences, 2003, pp. 65–74.

[17] S. Menchetti, F. Costa, and P. Frasconi, "Weighted decomposition kernels," in Proceedings of the 22nd international conference on Machine learning. ACM, 2005, pp. 585–592.

[18] N. Shervashidze, T. Petri, K. Mehlhorn, K. M. Borgwardt, and S. Vishwanathan, "Efficient graphlet kernels for large graph comparison," in International conference on artificial intelligence and statistics, 2009, pp. 488–495.

[19] D. Shasha, J. T. Wang, and R. 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. ACM, 2002, pp. 39–52.

[20] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in Proceedings of the ACM SIGMOD international conference on Management of data. ACM, 2004, pp. 335–346.

[21] S. Zhang, S. Li, and J. 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. ACM, 2009, pp. 192–203.

[22] M. Mongiovi, R. Di Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan, "Sigma: a set-cover-based inexact graph matching algorithm," Journal of bioinformatics and computational biology, vol. 8, no. 02, 2010, pp. 199–218.

[23] S. Zhang, J. Yang, and W. Jin, "Sapper: Subgraph indexing and approximate matching in large graphs," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, 2010, pp. 1185–1194.

[24] X. Wang, A. Smalter, J. Huan, and G. 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. ACM, 2009, pp. 472–480.

[25] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao, "Neighborhood based fast graph search in large networks," in Proceedings of

 ISBN: 978-1-61208-486-2

the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011, pp. 901–912.

[26] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan, "NeMa: Fast graph search with label similarity," in Proceedings of the VLDB endowment, 2013, pp. 181–192.

[27] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," Proceedings of the VLDB Endowment, vol. 5, no. 9, 2012, pp. 788–799.

[28] Y. Yuan, G. Wang, J. Y. Xu, and L. Chen, "Efficient distributed subgraph similarity matching," VLDB journal, vol. 24, 2015, pp. 369–394.

[29] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in ICDE. IEEE, 2008, pp. 963–972.

[30] W. Zheng, L. Zou, X. Lian, D. Wang, and D. 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. ACM, 2013, pp. 1595–1600.

[31] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, [retrieved: May, 2016].

[32] "DBLP Network," http://dblp.uni-trier.de/db/, [retrieved: May, 2016].

[33] N. Przulj, D. Corneil, and I. Jurisica, "Efficient estimation of graphlet frequency distributions in protein-protein interaction networks," Bioinformatics, vol. 22, no. 8, 2006, pp. 974–980.

[34] G. T. Heineman, G. Pollice, and S. Selkow, Algorithms in a Nutshell. O'Reilly Media, Inc., 2008.

 ISBN: 978-1-61208-486-2