KSGM: Keynode-driven Scalable Graph Matching

Xilun Chen, K. Selçuk Candan
Arizona State University
Tempe, AZ, USA
{xilun.chen, candan}@asu.edu

Maria Luisa Sapino
University of Torino
Torino, Italy
marialuisa.sapino@unito.it

Paulo Shakarian
Arizona State University
Tempe, AZ, USA
shak@asu.edu

ABSTRACT

Understanding how a given pair of graphs align with each other (also known as the *graph matching* problem) is a critical task in many search, classification, and analysis applications. Unfortunately, the problem of *maximum common subgraph isomorphism* between two graphs is a well known NP-hard problem, rendering it impractical to search for exact graph alignments. While there are several heuristics, most of these analyze and encode global and local structural information for every node of the graph and then rank pairs of nodes across the two graphs based on their structural similarities. Moreover, many algorithms involve a post-processing (or refinement) step which aims to improve the initial matching accuracy. In this paper [1] we note that the expensive refinement phase of graph matching algorithms is not practical in any application where scalability is critical. It is also impractical to seek structural similarity between all pairs of nodes. We argue that a more practical and scalable solution is to seek structural *keynodes* of the input graphs that can be used to limit the amount of time needed to search for alignments. Naturally, these *keynodes* need to be selected carefully to prevent any degradations in accuracy during the alignment process. Given this motivation, in this paper, we first present a structural *keynode* extraction (SKE) algorithm and then use structural keynodes obtained during off-line processing for keynode-driven scalable graph matching (KSGM). Experiments show that the proposed keynode-driven scalable graph matching algorithms produce alignments that are as accurate as (or better than) the state-of-the-art algorithms, with significantly faster online executions.

1. INTRODUCTION

Graphs have been used to represent a large variety of complex data, from multimedia objects, social networks, hypertext/Web, knowledge graphs (RDF), mobility graphs,to protein interactions. Let D be a set of entities of interest, a graph, $G(V, E)$, defined over $V = D$ describes the relationships between pairs of objects in D. The elements in the set V are referred to as the nodes or vertices of the graph. The elements of the set E are referred to as the edges and they represent the pairwise relationships between the nodes of the graph. Edges can be directed or undirected, meaning that the relationship can be non-symmetric or symmetric, respectively. Nodes and edges of the graph can also be labeled or non-labeled. The label of an edge, for example, may denote the name of the relationship between the corresponding pair of nodes or may represent other meta-data, such as the certainty of the relationship or the cost of leveraging that relationship within an application.

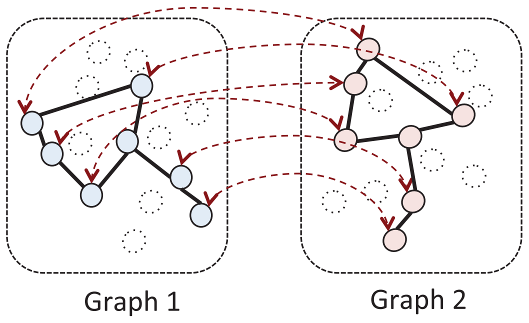

Figure 1: Graph matching/alignment problem seeks a *maximum common subgraph isomorphism* between two input graphs

Due to the success of the graph model as a powerful and flexible data representation, graph analysis and search tasks are also increasingly critical in many application domains. In particular, understanding how a given set of graphs align with each other (also known as the *graph matching/alignment* problem, Figure 1) forms the core task in many search, classification, and analysis applications. Unfortunately, the problem of *maximum common subgraph isomorphism* between two graphs is a well known NP-hard problem [24], making it impractical to search for exact or maximal graph alignments. As a result, while there are some attempts to improve the performance of exact maximum common subgraph matching solutions [23], most of the recent efforts in the area have focused on seeking approximate/inexact graph alignments [3, 18, 22, 19, 29].

While these algorithms differ in their specific techniques, most of them rely on a four phase process:

1. First, the matching algorithm analyzes and encodes the global structural information (for example a spectral signature [23]) corresponding to the nodes of the graph.
2. Secondly, the algorithm analyzes and encodes the local structural information (such as neighborhood degree distribution [29]) for the nodes of the graph.
3. Once these global and local signatures are encoded, the matching algorithm compares the signatures of pairs of nodes across the given graphs to rank these pairs of nodes (for example using a stable matching algorithm, like the Hungarian algorithm [16]) based on their overall structural similarities.

[1]This work is supported by NSF Grants #1339835 and #1318788. This work is also supported in part by NSF grant #0856090.

CIKM'15, October 19–23, 2015, Melbourne, Australia.
© 2015 ACM. ISBN 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806577.

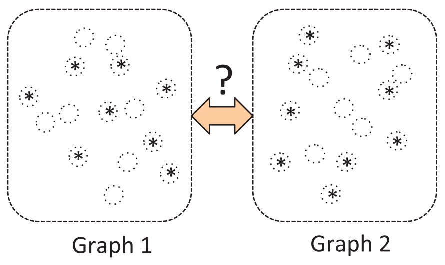

Figure 2: Keynode selection problem for scalable graph matching: the nodes marked with "*" are *keynodes* of the input graphs that can be used to reduce the amount of time needed to search for alignments

4. Finally, a post-processing, or refinement, step (involving, for example, a *vertex cover* operation) is used to improve the accuracy of the initial matching [29].

Unfortunately, many of these steps result in significant scalability challenges in terms of the matching time needed to compare the pairs of nodes:

- In particular, the expensive refinement phase of graph matching algorithms is not practical in applications where scalability of the graph matching operation is critical.
- Moreover, especially in very large graphs, it is also impractical to seek pairwise structural similarities for all node pairs during the graph matching process.

1.1 Contributions of this Paper

Based on these observations, in this paper, we argue that a more practical and scalable solution would be to seek structural *keynodes* of the input graphs that can be used to reduce the amount of time needed to search for alignments (Figure 2). Of course, these keynodes must be selected carefully to prevent any degradations in accuracy during the alignment process, especially because, as mentioned above, refinement post-processes are detrimental to scalability of matching algorithms.

Given this motivation, in this paper, we first present a highly efficient and effective *structural keynode extraction* (SKE) algorithm. The SKE algorithm, which is executed off-line, relies on a 3-step process:

1. In the first step, a PageRank algorithm [7] is ran to associate a structural score to each node in the graph.
2. In the second step, a scale-space (based on a difference-of-Gaussians (DoG) function defined over different scales of the graph) is constructed.
3. In the third step, keynode candidates are extracted by analyzing the resulting scale-space for extrema of the DoG function and a subset of these candidates are selected as structural keynodes.

We then propose a graph matching algorithm that uses these structural keynodes (obtained during off-line processing) for *keynode-driven scalable graph matching* (KSGM). In particular, KSGM extracts only *local* signatures and relies on the structural keynodes for fast node-to-node similarity searching. In addition, we also show that this keynode-driven approach not only reduces the number of comparisons that need to be performed online, but it also enables effective matching, even without having to rely on an expensive assignment algorithm, like the Hungarian algorithm (with $O(|V|^3)$ complexity). Experiment results show that the proposed structural keynode extraction and keynode-driven scalable graph matching algorithms produce alignments that are as accurate as (or better than) the state-of-the-art algorithms, while requiring significantly less online execution time without refinement.

1.2 Organization of this Paper

The paper is organized as follows: in the next section, we first introduce basic concepts and review existing graph matching algorithms. In Section 3, we provide overviews of the general graph matching process as well as the proposed keynode-driven scalable graph matching (KSGM) algorithm. Then, in Section 4, we present our structural keynode extraction (SKE) algorithm. In Sections 5 and 6, we discuss how to use these structural keynodes for obtaining graph alignments. We discuss the complexity of the proposed algorithms and parallelization opportunities in Section 7. We present experimental evaluations with various real and synthetic data sets in Section 8. These confirm that the proposed approximate graph matching algorithm is highly effective and efficient. Finally, we conclude the paper in Section 9.

2. BACKGROUND AND RELATED WORK

In this section, we review key concepts related to the graph matching problem and discuss the existing algorithms.

Graph Isomorphism: Given two graphs G and H, G is isomorphic to H if there exists a bijective mapping from the nodes of G to the nodes H that preserves the edge structure [13]: for any two vertices that are adjacent on G, the vertices they are mapped to are also adjacent on H, and vice versa.

Subgraph Isomorphism: Subgraph isomorphism seeks a bijective function, f, such that there is a subgraph G' of G and a subgraph H' of H, such that G' is isomorphic to H', with respect to f.

Maximum Common Subgraph Isomorphism: Maximum common subgraph isomorphism seeks the largest subgraph of G isomorphic to a subgraph of H [24]. Intuitively, the larger the maximum common subgraph of two graphs is, the more similar the graphs are to each other.

One of the first exact graph matching algorithms was proposed by Ullman [24]. An alternative way to search for a matching between two graphs is to rely on *graph edit distance* algorithms: given two graphs the corresponding graph edit distance is the least cost sequence of edit operations that transforms G_1 into G_2. Commonly used graph edit operations include *substitution*, *deletion*, and *insertion* of graph nodes and edges. Unfortunately, the graph edit distance problem is also known to be NP-complete [24]. In fact, even approximating graph-edit distance is very costly; the edit distance problem is known to be APX-hard [8]. [8] shows that graph isomorphism, subgraph isomorphism, and maximum common subgraph problem are special instances of the graph edit distance computation problem. Many subgraph isomorphism search algorithms have been developed, such as [15, 29, 14].

Approximate Graph Matching: In order to be applicable to large graphs, many heuristic and approximate graph matching algorithms have been proposed.

While, as we discussed above, graph matching through edit distance computation is an expensive task, there are various heuristics that have been developed to perform this operation more efficiently. `GraphGrep` [14] is one such technique, relying on a path-based representation of graphs. `GraphGrep` takes an undirected, node-labeled graph and for each node in the graph, it finds all paths that start at this node and have length up to a given, small upper bound, l_p. Given a path in the graph, the corresponding *id-path* is the list of the ids of the nodes on the path. The corresponding *label-path* is the list of the labels of the nodes on the path. The fingerprint of the graph, then, is a hash table, where each row contains the hash of the label-path and the corresponding number of id-paths in the graph. Irrelevant graphs are filtered out by comparing the numbers of id-paths for each matching hash key and by discarding those graphs which have at least one value in its fingerprint less than the corresponding value in the fingerprint of the query. Matching sub-graphs are found by focusing on the parts of the graph which correspond to the label-paths in the query. After, the relevant id-path sets are selected and overlapping id-paths are found and concatenated to build matching sub-graphs.

A common method to obtain an approximate graph matching is to use the eigenvectors derived from the adjacency matrix of the graph [23]: intuitively, two similar graphs should have similar eigenvectors; moreover, if we construct a $|V| \times |V|$ matrix (for example the Laplacian of the graph or a matrix encoding node distances) and decompose it into three matrices of $|V| \times c$, $c \times c$, and $c \times |V|$ elements using an eigen-decomposition technique like SVD, the c-length vector corresponding the node $v \in V$ can be used as a global-signature corresponding to node v. Once node-to-node similarities are computed, an assignment is usually found using an assignment algorithm, such as the Hungarian algorithm [16], which uses a primal-dual strategy to solve this problem in $O(|V|^3)$ time. This simple observation, led to several works leveraging different global-signatures for identifying node matches across different graphs [3, 18, 22, 19, 29]. [28] formulates the labeled weighted graph matching problem in the form of a convex-concave program, which searches for appropriate permutation matrices by solving a least-square problem. In addition, feature selection techniques are used for more accurate calculation [11, 12, 20]. In order to improve matching accuracy, [29] proposes to enrich the global-signatures associated to the graph nodes with local-signatures, encoding the properties of the immediate neighborhood of each node.

3. OVERVIEW OF KEYNODE-DRIVEN GRAPH MATCHING

Given a set $\mathcal{G} = \{G_1, G_2, ..., G_g\}$ of graphs and a query graph $G_q \in \mathcal{G}$, in this paper, we seek the maximum graph matching between G_q and all $G_i \in \mathcal{G}$ ($i \neq q$). Note that the exact solution for this problem is NP-hard [24]. Since we treat *scalability* as a key constraint, we consider inexact solutions and rely on the *matching quality measure* proposed in [29] to evaluate the accuracies of the resulting alignments: Let $G_q(V_q, E_q)$ be a query graph and let $G_i(V_i, E_i)$ be a graph in $\mathcal{G}$. Let $M_{q,i}(V_{q,i}, E_{q,i})$ be a subgraph of both G_q and G_i, returned by an inexact subgraph search algorithm. [29] defines the *matching quality* function as follows:

$$quality(M_{q,i}) = \frac{|E_{q,i}|}{\min(|E_q|, |E_i|)},$$

Intuitively, the quality function describes how similar the given query graph G_q and known graph G_i are by using the ratio of matched edges and the maximum number of edges that can be possibly matched, which is equal to the minimum number of edges between two graphs. In other words, the larger the number of edges in the graph $M_{q,i}$, the better is the *quality* of the matching (or the more similar the two graphs are).

3.1 Challenges

Given a query graph G_q, our goal is to rank the graphs in $\mathcal{G}$ according to their matching similarities against G_q (and eventually return the top few matches to the user). [29] solves this problem by relying on a 6-step process, common to many graph search algorithms:

1. [29], first, analyzes the global structure of each graph through eigen-decomposition of the graph Laplacian matrix and encodes this in the form of a c-length vector associated to each node in the graph.
2. Secondly, [29] encodes the structural information local to each node, v_j, in the form of an s_j-length degree distribution vector, where s_j is the number of nodes in the k-neighborhood of the node.
3. Given the global and local signatures of all nodes in G_q and G_i, [29] then computes the global and local similarities for each pair of nodes from the two graphs, in $O(|V_q| \times |V_i| \times c)$ and $O(|V_q| \times |V_i| \times max_{v_j}(s_j))$ time, respectively. It then combines (by multiplying) the global and local similarities of each pair of nodes into a single value, thereby quantifying the overall similarity of the pair.
4. Once the overall similarities for $|V_q| \times |V_i|$ pairs of nodes are computed, [29] drops node pairs with small degrees and, then, expands the remaining set of *anchor* pairs by adding, in an iterative manner, immediate good nearby pairs to this anchor set.
5. When no more pairs can be added to the *anchor* set, [29] uses the Hungarian algorithm to identify an initial node matching in $O(max\{|V_q|, |V_i|\}^3)$ time.
6. Finally, as a post-processing step, [29] applies a vertex cover based refinement, which explores different subsets of the nodes and searches for better alignments than the one initially identified. In particular, the algorithm seeks *small vertex covers*, which are likely to give the mismatched nodes additional chances to be refined. Note that since the minimum vertex cover problem is known to be NP-hard, the algorithm searches for *minimal* vertex covers in $O(m \times n^3)$ time, where $m = min\{|E_q|, |E_i|\}$ and $n = max\{|V_q|, |V_i|\}$.

Algorithm 1 Overview of keynodes based graph matching

```
Input:
    A set $\mathcal{G} = \{G_1, G_2, ...G_g\}$ of graphs
    A query graph $G_q \in \mathcal{G}$.
Output:
    Rank $G_i \in \mathcal{G}$ in terms of matching quality
    Offline process:
 1: for all $G_i \in \mathcal{G}$ (including $G_q$) do
 2:     Perform structural keynode extraction (SKE) for $G_i$
 3:     Extract local-signatures for all nodes in $G_i$
 4:     (Optional) Extract global-signatures for all nodes in $G_i$
 5: end for
    Online process:
 6: for all $G_i \in \mathcal{G}$ do
 7:     Compute local similarities for keynode pairs from $G_i$ and
        $G_q$.
 8:     (Optional) Compute global similarities for keynode pairs
        from $G_i$ and $G_q$ and combine these with local similarities.
 9:     Select anchors to obtain a base matching
10:     Expand the base matching to obtain $M_{q,i}$
11:     Compute matching quality, $quality(M_{q,i})$
12: end for
13: Rank $G_i \in \mathcal{G}$ in terms of $quality(M_{q,i})$
```

This process includes a number of very expensive steps: The first two steps, involving global and local analysis are expensive, but can be performed off-line and indexed for later reuse assuming that the graphs are available ahead of time. The last four steps, however, need to be performed on-line, yet they consist of operations that are quadratic or higher. In particular, the last refinement step, with $O(m \times n^3)$ time cost is impractical for most large data graphs.

In this paper, we note that Step 3 can be significantly sped up if the similarity computations are limited to only a small subset of the vertices in V_q and V_i (which we refer to as *keynodes* of V_q and V_i). However, the use of keynodes for node similarity computation is not sufficient to reduce the overall complexity as, once the keynodes are identified and the keynode pairs set is expanded, solving the assignment problem needed to return the matching would still take $O(max\{|V_q|, |V_i|\}^3)$ time, if we were to apply the Hungarian algorithm on the extracted keynodes. Therefore, we also need to reduce the time complexity of this step significantly. It is especially important that the initial keynode based similarity computation is accurate as we cannot afford a cubic algorithm like Hungarian algorithm to return a high-quality matching.

3.2 Outline of `KSGM`

Algorithm 1 illustrates an overview of the *keynode-driven scalable graph matching* (`KSGM`) process. In the rest of the paper, we study each step in detail. First, in the next two sections, we focus on the offline steps of `KSGM`, which involve identifying keynodes and extracting local-signatures. The online steps of the `KSGM` algorithm are discussed in Section 6.

4. STRUCTURAL KEYNODE EXTRACTION

In this section, we propose an off-line *structural keynode extraction* (`SKE`) algorithm which identifies $\Theta\%$ (where Θ is a user provided parameter) of the nodes in V as the keynode set, K, of a given graph, $G(V, E)$ to support scalable graph matching. The proposed `SKE` algorithm has a number of advantages: (a) First of all, the identified keynodes are robust against noise, such as random edge insertion/removal; and (b) the identified nodes represent structural features of the graph of different sizes and complexities (i.e., correspond to neighborhoods of different sizes).

4.1 Naive Solution - Selecting Structural Keynodes based on Node Significance

As described above, the keynodes of the graph need to *represent* the structural properties of the graph well (i.e., extracted keynodes need to be structurally *significant* in the graph) to support effective matching. Therefore, the first alternative is to rely on traditional node significance measures.

Measures like *betweenness* [26] and the *centrality/cohesion* [5], help quantify how *significant* any node is on a given graph based on the underlying graph topology. The *betweenness* measure [26], for example, quantifies the number of shortest paths that pass through a given node. The *centrality/cohesion* [5] measures quantify how close to a clique the given node and its neighbors are. Other *authority*, *prestige*, and *prominence* measures [4, 7, 5] quantify the significance of the node in the graph through eigen-analysis or random walks, which help measure how reachable a node is in the graph. PageRank [7] is one of the most widely-used random-walk based methods for measuring node significance and has been used in a variety of application domains, including web search, biology, and social networks. The basic thesis of PageRank is that a node is important if it is pointed to by other important nodes – it takes into account the connectivity of nodes in the graph by defining the score of the node $v_i \in V$ as the amount of time spent on v_i in a sufficiently long random walk on the graph. Given a graph $G(V, E)$, the PageRank scores are represented as $\vec{r}$, where

$$\vec{r} = \alpha \mathbf{T}_G \vec{r} + (1 - \alpha)\vec{t}$$

where $\mathbf{T}_G$ is a transition matrix corresponding to the graph G, $\vec{t}$ is a teleportation vector (such that $\vec{t}[i] = \frac{1}{\|V\|}$), and α if the residual probability (or equivalently, $(1 - \alpha)$ is the so-called teleportation probability). Unless the graph is weighted, the transition matrix, $\mathbf{T}_G$, is constructed such that for a node v with k (outgoing) neighbors, the transition probability from v to each of its (outgoing) neighbors will be $1/k$. If the graph is weighted, then the transition probabilities are adjusted in a way to account for the relative weights of the edges.

Therefore, as the first alternative, we consider a PageRank based keynode selection scheme: in this scheme, given a graph $G(V, E)$, we would (a) first identify the PageRank scores, $p(v_i)$, of all $v_i \in V$, then (b) we would rank the nodes in non-increasing order of PageRank scores, and finally, (c) we would return the top $\Theta\%$ of the nodes in V as the keynode set, K.

4.2 Proposed Solution - Robust Keynode Extraction through Scale Space Analysis

We note that the above alternative has a number of disadvantages:

- First of all, many of the structural significance measures, such as PageRank, are not entirely robust against modifications in the graph. The PageRank score of a node, for example, can jump significantly, if a new edge connects the sub-graph in which the node is contained to a high PageRank node in the graph.
- Secondly, common structural significance measures, like PageRank, capture the significance of a node in the whole graph and favor nodes that are *overall central*. However, this may be disadvantageous as there is a possibility that smaller scale, but distinct (and, therefore, useful for matching) structural features of the graph may be missed.

We therefore argue that we need a better alternative, which is both *robust* and *multi-scale*. We build the proposed `SKE` algorithm based on three key insights:

- *Robustness:* Even when the PageRank scores of the nodes themselves vary due to graph transformations, such as edge insertions and removals, a given node's PageRank score relative to the scores of the nodes in its neighborhood is likely to be stable.
- *Structural distinctiveness:* A node is structurally distinctive in its neighborhood, if "the relationship between its PageRank score to the PageRank scores of its neighbors" is different from the "relationships between the node's neighbors' PageRank scores and the PageRank scores of their own neighbors".
- *Multi-scale:* Since we do not know the scale of the structurally distinctive features of the graph, we need to search for features of potentially different sizes.

It is important to note that similar requirements also exist in other application domains. For example, algorithms for extracting such robust, local features have been developed for 2D images (SIFT [21]), uni-variate time series [9], and multi-variate time series [25]. In this paper, we argue that a similar process can be used to identify keynodes (corresponding to robust, multi-scale structural features) of a graph, if the nodes are annotated with PageRank scores ahead of the time. Let $G(V, E, p)$ be a PageRank-labeled graph, where $p()$ is a mapping from the nodes to the corresponding PageRank scores. What makes the problem of extracting local features from *PageRank*-labeled graphs challenging is that the concepts of *neighborhood*, *gradient*, and *smoothing* are not well-defined for graphs.

Therefore, before we describe the keynode extraction process, we describe how to smooth a PageRank-labeled graph. Intuitively, smoothing the graph with respect to the scores associated to the graph nodes creates versions of the given graph at different resolutions and, thus, helps identify features with different amounts of details.

4.2.1 *Gaussian Smoothing of a PageRank-Labeled Graph*

1D or 2D data are commonly smoothed by applying a convolution operation with a Gaussian window. For example, if $Y = \langle t_0, t_1, \ldots, t_l \rangle$ is a time series data and σ is a smoothing parameter, its smoothed version, $\tilde{Y}(t, \sigma)$, is obtained through $\mathcal{G}(t, \sigma) * Y(t)$ where $*$ is the convolution operation in t and $\mathcal{G}(t, \sigma)$ is the Gaussian function

$$\mathcal{G}(t, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-t^2}{2\sigma^2}}.$$

Essentially, the Gaussian smoothing process takes a weighted average of values of the points in the vicinity of a given point, t.

The closer a point to t, the higher is the weight. Therefore, in order to implement a similar Gaussian smoothing of the given graph, we first need to define a distance function to measure how close different nodes are to each other. Common applicable definitions of node distance include the hop distance (determined by the shortest edge distance between the nodes on the given graph) or hitting distance [10]. In this paper, we use hop distance to measure how far nodes are to each other:

DEFINITION 1 (NODE DISTANCE MATRIX). Let us be given a graph $G(V, E)$ with n nodes. The ordering among the nodes is described through a set of node distance matrices, $\mathbf{N_j}$, where

- $\mathbf{N_j}$, for $j \geq 0$, is an $n \times n$ 0, 1-valued matrix, where for a given node v_i in the graph, G, the i^{th} row in the matrix, $\mathbf{N_j}$ is 1 only for nodes that have node distance exactly j from the node v_i, and
- $\mathbf{N_j}$, for $j \leq 0$, is an $n \times n$ 0, 1-valued matrix, where for a given node v_i, the i^{th} column in the matrix, $\mathbf{N}(j, G) = 1$ only for nodes that have distance exactly j on the inverted graph, where all edges are inverted. ◇

Intuitively, the cell $\mathbf{N_j}[v_1, v_2] = 1$ if the node v_2 is exactly j hops from v_1. When j is positive the hop-distance is measured following outgoing edges, whereas when j is negative, incoming edges are followed. Given this, we construct multiple scales of the given graph G by using a Gaussian graph smoothing function defined as follows.

DEFINITION 2 (GAUSSIAN GRAPH SMOOTHING FUNCTION). Let us be given a labeled graph $G(V, E, x)$ and let

- σ be a smoothing parameter.
- $X = \langle x(v_1), x(v_2), ..., x(v_n)\rangle$ be a vector encoding the labels associated with the nodes, $v_i \in V$.

Then, if G is a directed graph, the *non-normalized Gaussian graph smoothing function*, $S^{\oslash}_{G,\sigma}()$ is defined as

$$S^{\oslash}_{G,\sigma}(X) = \mathcal{G}(0,\sigma)\mathbf{I}X + \sum_{j=1}^{n} \mathcal{G}(j,\sigma)\mathbf{N_j}X + \sum_{j=1}^{n} \mathcal{G}(j,\sigma)\mathbf{N_j}X,$$

where $\mathcal{G}(0, \sigma)$ is a Gaussian function with zero mean and σ standard deviation. If, on the other hand, G is an undirected graph, then the *non-normalized Gaussian graph smoothing function* is

$$S^{\oslash}_{G,\sigma}(X) = \mathcal{G}(0,\sigma)\mathbf{I}X + \sum_{j=1}^{n} 2\mathcal{G}(j,\sigma)\mathbf{N_j}X.$$

Intuitively, $S^{\oslash}$ applies Gaussian-based weighted averaging to the entries of vector X based on the hop-distances[2]. However, unlike the basic Gaussian smoothing, during (non-normalized) relationship smoothing, there may be more than one node at the same distance and all such nodes have the same degree of contribution. As a consequence, the sum of all contributions may exceed 1.0. Therefore, the *normalized Gaussian graph smoothing function*, $S(G, \sigma)$, discounts weights based on the number of nodes at a given distance:

$$S_{G,\sigma}(X) = \left(S^{\oslash}_{G,\sigma}X\right) \div \left(S^{\oslash}_{G,\sigma}1_{(n)}\right),$$

where $1_{(n)}$ is an n-vector such that all values are 1 and "$\div$" is a pairwise division operation. ◇

[2]In practice, since the Gaussian function drops fast as we move away from the mean, we need to consider only a small window, w, of hops

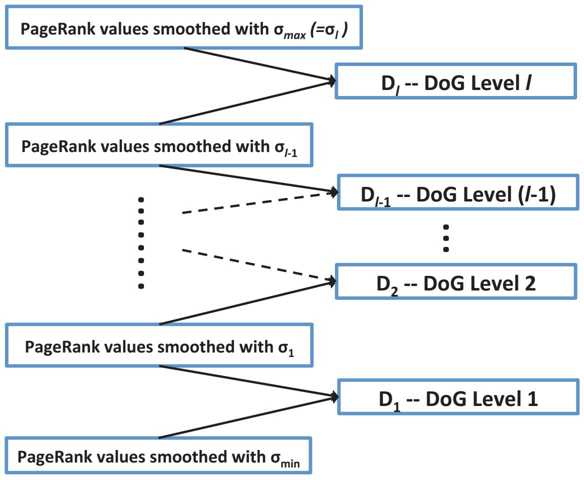
PageRank values smoothed with σ_{max} (=σ_l)
D_l -- DoG Level l
PageRank values smoothed with σ_{l-1}
D_{l-1} -- DoG Level (l-1)
D_2 -- DoG Level 2
PageRank values smoothed with σ_1
D_1 -- DoG Level 1
PageRank values smoothed with σ_{min}

Figure 3: Computing the Difference-of-Gaussian (DoG) series of the PageRank values of a graph

Intuitively, the smoothing function S applies Gaussian smoothing on the X values (associated with the nodes, $v_i \in V$) based on the hop-distances between nodes and returns a vector

$$S_{G,\sigma}(X) = \langle \tilde{x}(v_1), \tilde{x}(v_2), \ldots, \tilde{x}(v_n)\rangle$$

encoding the smoothed X values associated with the graph nodes. Note that, since at a given hop distance there may be more than one node, all the nodes at the same distance have the same degree of contribution and the degree of contribution gets progressively smaller as we get further away from the node for which the smoothing is performed.

Therefore, given a PageRank-labeled graph, $G(V, E, p)$, and a corresponding PageRank vector, $P = \langle p(v_1), p(v_2), ..., p(v_n)\rangle$, encoding PageRank scores associated with the nodes, $v_i \in V$, the vector

$$S_{G,\sigma}(P) = \langle \tilde{p}(v_1), \tilde{p}(v_2), \ldots, \tilde{p}(v_n)\rangle,$$

encodes the σ-smoothing of the PageRank-annotated graph, $G(V, E, p)$. We also say that $S_{G,\sigma}(P)$ encodes the PageRank scores of G at scale σ.

We next describe how to construct a scale-space for the given graph through an iterative smoothing process leveraging the PageRank vector and the structure of the graph.

4.2.2 Graph Scale-Space Construction

The first step in identifying robust graph features is to generate a scale-space representing versions of the given graph with different amounts of details. In particular, building on the observation that features are often located where the differences between neighboring regions (also in different scales) are large, we seek structural features of the given graph at the extrema of the scale space defined by the difference-of-the-Gaussian (DoG) series. More specifically, given

- a PageRank-labeled graph, $G(V, E, p)$,
- the corresponding vector, $P = \langle p(v_1), p(v_2), ..., p(v_n)\rangle$ encoding the scores associated with the nodes, $v_i \in V$,
- a minimum smoothing scale, σ_{min},
- a maximum smoothing scale, σ_{max},
- the number, l, of levels of the scale space,

then, we compute a difference-of-Gaussians (DoG) series, $\mathbb{D}(G, P, \sigma_{min}, \sigma_{max}, l) = \{D_1, D_2, ..., D_l\}$, where each D_i encodes the differences of two nearby scales separated by a multiplicative factor k:

$$D_i = S_{G,k^i\sigma_{min}}(P) - S_{G,k^{i-1}\sigma_{min}}(P),$$

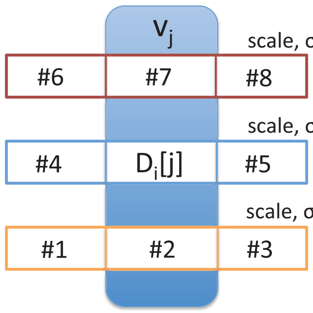

Figure 4: Extrema detection

where $k = \sqrt[l]{\frac{\sigma_{max}}{\sigma_{min}}}$. Figure 3 visualizes the process:

- On the left hand side of the figure, we have the incrementally smoothed versions of the PageRank vector, P. Here, the lowest level, $S_{G,\sigma_{min}}(P)$, corresponds to the most detailed version of the graph (with the least amount of smoothing), whereas $S_{G,\sigma_{max}}(P)$ corresponds to the least detailed (most smoothed) version of the graph. In other words, σ_{min} determines the sizes of the smallest structural features we can locate and $\sigma_{max} = k^l \sigma_{min}$ determines the sizes of the largest structural features we can identify. In particular, since under Gaussian smoothing, a diameter of 6σ would cover $\sim 99.73\%$ of the weights, the diameter of the smallest structural feature that can be identified using SKE is $\sim 6\sigma_{min}$ hops, whereas the diameter of the largest feature would be $\sim 6\sigma_{max}$ hops.

  The number of levels, l, denotes the number of detail levels (or scales) we explore between σ_{min} and σ_{max}. Intuitively, each of these levels corresponds to a different target size for the structural features of the graph.

- On the right hand side of the figure, we have the resulting Difference-of-Gaussian (DoG) series, consisting of vectors, D_1 through D_l.

Note that, intuitively, $D_i[j]$ measures how different the PageRank values of the neighborhood around v_j at scale $\sigma_{i-1}(= k^{i-1}\sigma_{min})$ are from the PageRank values of the neighborhood around v_j at scale $\sigma_i(= k^i \sigma_{min})$.

Therefore, a large $D_i[j]$ value would indicate a major structural change when neighborhoods of different size around v_j are considered (e.g., a node with a high PageRank score is included when considering a neighborhood of larger scale). In contrast, a small $D_i[j]$ indicates that there is minimal structural change when considering neighborhoods of different scales.

4.2.3 Identifying Keynode Candidates

As we mentioned earlier, our intuition is that a graph node is structurally distinctive in its neighborhood, if "the relationship between its PageRank score to the PageRank scores of its neighbors" is different from the "relationships between the node's neighbors' PageRank scores and the PageRank scores of their own neighbors, at multiple scales". Therefore, to locate the keynode candidates, we focus on the local extrema of the difference-of-Gaussian (DoG) series $\mathbb{D}$. More specifically, we identify $\langle v_j, \sigma_i \rangle$ pairs where the DoG value for node v_j at scale, $\sigma_i = k^i \sigma_{min}$, is an extremium (maximum and/or minimum) with respect to the neighbors of v_j in the same scale as well as neighbors in the previous and next levels of the scale space.

In order to verify if the pair $\langle v_j, \sigma_i \rangle$ is an extremium or not, we compare $D_i[j]$ with the values corresponding to eight DoG-neighbors in the scale-space, as visualized in Figure 4:

- DoG-neighbors numbered #2 and #7 correspond to the DoG values of the same node at the previous and next levels of the scale space. Therefore, we have

$$N_{\langle v_i,\sigma_j \rangle}[2] = D_{i-1}[j] \text{ and } N_{\langle v_i,\sigma_j \rangle}[7] = D_{i+1}[j].$$

- In contrast, DoG-neighbors #3, #5, and #8 correspond to the (average) DoG values of the forward neighbors of the node v_j, at the previous, current, and next levels of the scale space, respectively. Therefore, we have

$$N_{\langle v_i,\sigma_j \rangle}[3] = (\mathbb{F}D_{i-1})[j],$$
$$N_{\langle v_i,\sigma_j \rangle}[5] = (\mathbb{F}D_i)[j], \quad N_{\langle v_i,\sigma_j \rangle}[8] = (\mathbb{F}D_{i+1})[j],$$

  where, $\mathbb{F}$ is a row-normalized adjacency matrix accumulating the (averaged) contributions of the nodes to their neighbors along the forward edges.

- Similarly, DoG-neighbors #1, #4, and #6 correspond to the (average) DoG values of the backward neighbors at the previous, current, and next levels of the scale space. Therefore, we have

$$N_{\langle v_i,\sigma_j \rangle}[1] = (\mathbb{B}D_{i-1})[j],$$
$$N_{\langle v_i,\sigma_j \rangle}[4] = (\mathbb{B}D_i)[j], \quad N_{\langle v_i,\sigma_j \rangle}[6] = (\mathbb{B}D_{i+1})[j],$$

  where, $\mathbb{B}$ is a row-normalized backward-adjacency matrix (where all edges are reversed) accumulating the (averaged) contributions of the nodes to their neighbors along the backward direction of the edges.

Given these, the pair $\langle v_j, \sigma_i \rangle$ is an extremum (i.e., v_j is a keynode candidate at scale σ_i), iff $D_i[j]$ is a local maximum

$$D_i[j] \geq \underset{1 \leq h \leq 8}{MAX} N_{\langle v_j,\sigma_i \rangle}[h]$$

or it is a local minimum

$$D_i[j] \leq \underset{1 \leq h \leq 8}{MIN} N_{\langle v_j,\sigma_i \rangle}[h].$$

Intuitively, since $D_i[j]$ measures how different *the PageRank values of the neighborhood around* v_j *at scale* σ_i are from *the PageRank values of the neighborhood around* v_j *at scale* σ_{i-1}, a local maximum corresponds to a *highly scale-sensitive* region (amidst relatively *scale-insensitive* regions), whereas a local minimum corresponds to a *scale-insentive* region (amidst more *scale-sensitive* regions), of the graph.

4.2.4 Selecting the Best Keynodes

In the final step, we need to rank the keynode candidates and return the top $\frac{\Theta}{100} \times |V|$ of them, where Θ is a user provided parameter, as the keynode set, K. We propose *Extremum Ranking* to select the best keynodes.

Since keynodes are located at the local extrema of the DoG series, we can rank the keynode candidates based on their *extremum score* defined as follows: Let the pair $\langle v_j, \sigma_i \rangle$ be a local extremum. The corresponding extremum score, $\xi(\langle v_j, \sigma_i \rangle)$, is defined as

$$\begin{cases} D_i[j] - \left(\underset{1 \leq h \leq 8}{MAX} N_{\langle v_j,\sigma_i \rangle}[h] \right) & \text{if } \langle v_j, \sigma_i \rangle \text{ is max.} \\ \left(\underset{1 \leq h \leq 8}{MIN} N_{\langle v_j,\sigma_i \rangle}[h] \right) - D_i[j] & \text{if } \langle v_j, \sigma_i \rangle \text{ is min.} \end{cases}$$

Intuitively, the higher the extremum score is, the better local extremum (and, thus, a better keynode) is $\langle v_j, \sigma_i \rangle$.

5. LOCAL NODE SIGNATURES

The next step in the process is to extract the local signatures (to be used to compute local node similarities) for the nodes in the graph. Note that this process is also offline.

While there are different local signatures proposed in the literature, in our work we build on the k-neighborhood degree distribution based local signature proposed in [29] (both because it is simple and effective and also because this helps us compare our keynode-driven approach to the approach proposed in [29] more directly). Briefly, for each node $v_j \in V$ and for a user provided k, [29] first identifies the set, $N_k(v_j) \subseteq V$, of nodes that are at most k hops from v_j and extracts a subgraph, $G_k(v_j) \subseteq G$, induced by v_j and its k-hop neighbors. Then the degree sequence,

$$\kappa_j = [d_{j,1}, d_{j,2}, \ldots, d_{j,|N_k(v_j)|}]$$

consisting of the degrees of nodes in $G_k(v_j)$ (excluding v_j), sorted in non-increasing order, along with the degree of the node v_j and the numbers of vertices and edges in its k-hop neighborhood, form the local signature of node v_j:

$$local\_signature(v_j) = \langle \kappa_j, degree(v_j), \nu_j, \varepsilon_j \rangle,$$

where $\nu_j = |N_k(v_j) \cup \{v_j\}|$ is the number of nodes in $G_k(v_j)$ and $\varepsilon_j = |E_k(v_j)|$ is the number of edges.

Note that, while we use a local signature similar to that proposed in [29], we extend the node pair ranking function to better account for the node degrees as discussed later in Section 6.1.1. As we see in Section 8, this extension provides a significant boost in accuracy.

6. (KEYNODE-BASED) GRAPH MATCHING

As discussed in Section 3 and outlined in Algorithm 1, once the keynodes are extracted and local signatures are computed offline, the next steps of the algorithm are to

- compare the signatures of pairs of nodes across the given graphs to rank these pairs of nodes,
- select a set of pairs of keynodes (we refer it as anchor set) that serve as the base matching, and
- expand this base matching to obtain $M_{q,i}$.

We now describe how these steps are implemented in the keynode-driven scalable graph matching (`KSGM`) algorithm.

6.1 Anchor Set Selection

Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two graphs and let $K_1 \subseteq V_1$ and $K_2 \subseteq V_2$ be the corresponding keynodes identified by the `SKE` algorithm proposed in Section 4. The next step is to select a subset, A, of the pairs of nodes in $K_1 \times K_2$ as the anchor set of alignments based on a ranking function (a) evaluating how structurally similar a pair of nodes are and (b) how likely they are to lead to an effective expansion process to discover other alignments.

6.1.1 Node Similarity Matching and Node Pair Ranking

As we discussed in Section 5, `KSGM` uses a local node signature similar to the one proposed by [29]: $\langle \kappa_j, degree(v_j), \nu_j, \varepsilon_j \rangle$, where ν_j is the number of nodes in the neighborhood of v_j, ε_j is the number of neighborhood edges, and $\kappa_j = [d_{j,1}, d_{j,2}, \ldots, d_{j,|N_k(v_j)|}]$ consists of the degrees of nodes in the k-neighborhood of v_j (excluding v_j), sorted in non-increasing order.

Local Neighborhood Similarity.

Let v_i and v_j be two nodes (from two different graphs) and let $G_k(v_i)$ and $G'_k(v_j)$ be the corresponding induced $k-$neighborhood graphs. Then, local similarity function $nbhd\_sim(v_i, v_j)$ proposed by [29] accounts for the alignment between the degree distributions in these neighborhood graphs[3]:

$$nbhd\_sim(v_i, v_j) = \frac{n_{min} + D(v_i, v_j)}{(\nu_i + \varepsilon_i)(\nu_j + \varepsilon_j)},$$

where

$$d_{min} = min\{degree(v_i), degree(v_j)\}$$
$$n_{min} = min\{\nu_i, \nu_j\}$$
$$D(v_i, v_j) = \frac{d_{min} + \sum_{h=1}^{n_{min}-1} min\{d_{i,h}, d_{j,h}\}}{2}.$$

Node Pair Ranking with Extended Similarity.

While the local neighborhood similarity computation we use is similar to the one proposed in [29], we rank pairs of nodes differently. Let v_i and v_j be two nodes (from two different graphs). In particular, [29] ranks the pair $\langle v_i, v_j \rangle$ of nodes based on their neighborhood similarities, $nbhd\_sim(v_i, v_j)$. We, however, argue that neighborhood similarity is not sufficient for accounting for how effective the node pair is in supporting expansion. More specifically, we observe that a pair, $\langle v_i, v_j \rangle$, is likely to be a better anchor for expansion than the pair $\langle v_a, v_b \rangle$ if not only (a) the neighborhoods of v_i and v_j are more similar to each other than the pair, v_a and v_b, but also (b) if v_i and v_j have degrees that are more aligned with each other than v_a and v_b. Based on this observation, instead of applying a degree threshold, we propose that the pair $\langle v_i, v_j \rangle$ should be ranked based on the ranking function

$$\rho(v_i, v_j) = nbhd\_sim(v_i, v_j) \times \frac{min\{degree(v_i), degree(v_j)\}}{max\{degree(v_i), degree(v_j)\}}.$$

Note that [29] simply drops node pairs where the minumum of the two node degrees is smaller than the larger average degree of the two input graphs. We, however, argue that such node pairs may be useful, especially if the degrees in the graph are not uniformly distributed and the maximum matching occurs at the sparse portions of the graph. Therefore, we keep such pairs as long as they rank highly based on $\rho()$. We evaluate this ranking function in Section 8.

6.1.2 Keynode pair Selection

[29] uses the Hungarian algorithm to identify an initial node matching in $O(n^3)$ time, where $n = max\{|V_1|, |V_2|\}$. To reduce the execution time, [29] prunes those node pairs for which the similarity is ≤ 0.5. Since, instead of considering the node pairs in $V_1 \times V_2$, we only need to consider pairs of nodes in $K_1 \times K_2$, and since $|K_1| \ll |V_1|$ and $|K_2| \ll |V_2|$, keynode-driven processing is likely to be faster even without using the threshold. However, the cubic time of the Hungarian algorithm is still prohibitive and impractical for scalable graph matching. Therefore, we propose a greedy anchor selection algorithm, which (as we see in Section 8) performs very well when used along with keynodes selected in Section 4 and the proposed ranking function, $\rho()$. In particular, we first include all keynode pairs in $K_1 \times K_2$ into a queue in the order of their ranks based on $\rho()$, then, until the queue is empty, we remove and consider the keynode pair, $\langle v, u \rangle$ at the head of the queue. If neither v nor u has been marked `anchored`, we include $\langle v, u \rangle$ as an anchor and we mark v and u as `anchored`, otherwise, we drop the pair, $\langle v, u \rangle$.

Note that this process has $O((|K_1| \times |K_2|) \times log(|K_1| \times |K_2|))$ time cost (instead of the cubic cost of the Hungarian algorithm) and,

[3]In addition to using local similarities, [29] also extracts global signatures along with the local-signatures to compute node similarities. As we see in Section 8, the proposed keynode-driven graph matching algorithm achieves good results without having to rely on such global-signatures.

as we see in Section 8, performs very well in practice. Furthermore, the nature of Hungarian algorithm, which forces to pair all possible nodes to produce the optimal bipartite matching for the given two sets of nodes, is not guaranteed to provide a better matching in this case. Since the extracted keynodes are not all necessarily perfectly paired with each other, some keynodes can be a unique feature of the given graph, which does not align with other graphs, by forcing them to pair with other keynodes, it in fact introduces a bad initial base matching, and thus expand into an even worse matching. The proposed greedy matching algorithm, however, only consider the highly aligned keynodes, which in practice provides better results than the optimal bipartite matching.

6.2 Matching List Expansion

Because keynodes are inherently sparsely localized, the anchor set, A, is not necessarily a good final matching for graphs G_1 and G_2. We therefore need to expand this anchor list. Here, we follow [29]'s recommendation and expand the list incrementally by considering the neighbors (and their neighbors) until no effective expansion is possible (but we use the ranking function $\rho()$ instead of the node similarity function):

1. we first include all node pairs in A into a ranked queue (i.e., max-heap) in the order of their ranks based on the ranking function, $\rho()$,
2. then, for each node pair $\langle v, u\rangle \in A$, we also include the node pairs in $neighbors(u) \times neighbors(v)$ in the same ranked queue
3. then, until the ranked queue is empty, we remove and consider the node pair, $\langle v, u\rangle$ at the head of the ranked queue
   (a) if either v or u has not yet been marked `matched`, then
      i. we include the pair, $\langle v, u\rangle$, in the expanded matching list, L,
      ii. we mark both v and u as `matched`, and
      iii. then, the pairs in $neighbors(u) \times neighbors(v)$ are included in the ranked queue
   (b) otherwise, we drop the pair, $\langle v, u\rangle$

Once the anchor list is expanded, [29] relies on a post-process, with time complexity, $O(m \times n^3)$, where $m = min\{|E_1|, |E_2|\}$ and $n = max\{|V_1|, |V_2|\}$. This step is not scalable due to its prohibitive time complexity. Therefore, the proposed keynode-driven scalable graph matching (`KSGM`) algorithm omits this refinement post-process, due to its high time complexity[4]. Instead, the set, L, of node pairs remaining after the expansion process is directly returned as the aligned nodes of the matching, $M_{1,2}$, for the input graphs, G_1 and G_2.

7. TIME COMPLEXITY ANALYSIS

7.1 Offline Time Complexity

Let $G(V, E)$ be a graph to be indexed in the database.

7.1.1 *Structural Keynode Extraction*

The first step in structural keynode extraction is to obtain the PageRank scores for the nodes of the two graphs. While, this is an expensive operation (involving a matrix inversion with $O(|V|^{2.373})$ complexity for a graph with $|V|$ nodes), there are many efficient, approximate implementations of PageRank, including sublinear approximations [6].

The second step is the creation of an l-layer scale-space for G. To construct the scale space, we first construct a node distance matrix, which requires an all-pairs shortest path computation, with complexity $O(|V|^3)$ for a graph with $|V|$ nodes, but more efficient randomized algorithms exist [27]. Once the node distances have been computed, we construct the scale-space in $O(l \times |V| \times max\_w\_nbhd\_size)$, as for each of the l scales, the score of each node needs to be smoothed considering the scores of the vertices in its w-hop neighborhood (w is the Gaussian window size and $max\_w\_nbhd\_size$ is the size of the largest w-hop neighborhood in G).

Once the scale-space is constructed, next, we identify the keynode candidates. This involves $O(l \times |V|)$ time, because for each of the l scales, each node needs to be compared with a constant number (8) of DoG-neighbors in the scale-space.

Finally, we rank the keynode candidates to select the top $K = \frac{\Theta}{100}V$ many as the keynodes to bootstrap the online matching process. Let there be C many keynode candidates. Computing the ranking scores for these takes $O(C)$ time, because each keynode candidate needs to be compared with a constant number of DoG-neighbors and obtaining the top K takes $O(C \times log(K))$ time.

7.1.2 *Local Signature Extraction*

Since the local signature extraction process needs to extract the k-hop neighborhoods around the nodes, the complexity of this step is $O(|V| \times max\_k\_nbhd\_size)$, where $max\_k\_nbhd\_size$ is the size of the largest k-hop neighborhood in G. Note that this step can also leverage the node distance matrix constructed during the offline keynode extraction process.

7.2 Online Time Complexity

Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two graphs. The online process includes the following operations.

7.2.1 *Local Similarity Computation for Keynodes*

This process has $O(|K_1| \times |K_2| \times compare\_length)$ complexity, where
$compare\_length = min\{max\_k\_nbhd\_size_1, max\_k\_nbhd\_size_2\})$
since signatures (of length are compared for each pair of nodes in the keynode sets K_1 and K_2.

7.2.2 *Anchor Set Selection*

This greedy process has $O((|K_1| \times |K_2|) \times log(|K_1| \times |K_2|))$ time cost as each pair off nodes among the keynode sets need to be considered only once in ranked order.

7.2.3 *Anchor Set Expansion*

This has $O((|V_1| \times |V_2|) \times log(|V_1| \times |V_2|))$ worst case time cost, as in the worst case, all pairs of vertices across the two graphs may need to be considered for expansion in ranked order.

8. EXPERIMENTS

In this section, we present experimental evaluations of the proposed *keynode-driven scalable graph matching* (`KSGM`) algorithm. In particular, we compare `KSGM` to the graph matching algorithm presented in [29] in terms of efficiency and accuracy.

8.1 Data Sets

8.1.1 *Facebook Data Graph*

The first data set we used is the Facebook social circles data graph obtained from the Stanford Large Network Dataset Collection [2]. This is a connected graph with 4039 nodes and 88234 edges. The graph has a diameter of 8 and a 90-percentile effective diameter of 4.7. For the experiments, we constructed 10 subgraphs by uniformly sampling connected subsets, containing $60 - 70\%$ of the original graph nodes. Once the subgraphs are obtained, each of the subgraphs is used as a query against the rest. We report the averages of execution time and accuracy.

8.1.2 *Synthetic Graph Data Sets*

In addition to the Facebook graph, we also used synthetic graphs, where we controlled the topology, size, and node degree to explore

[4]Though, in cases where scalability is not critical, this refinement can be implemented without any change in the rest of the algorithm.

Table 1: Synthetic graph topologies and configurations

Graph topology	Number of nodes	Average degree
Erdos-Renyi (ER)	5000, 7500 (plus 1 to 10%)	4, 8, 16
Power law (PL)	5000, 7500 (plus 1 to 10%)	4, 8, 16

the advantages and disadvantages of the algorithms under different scenarios.

We generated the synthetic graphs using the well known random graph generating tool, NetworkX [1]. We consider two common graph topologies: the Erdos-Renyi (ER) model and the power law topology (PL) under the Barabasi-Albert model. Table 1 lists the number of nodes and average degree settings that we used for assessing our algorithms. For each configuration, we generated 10 graphs. Note that, in addition to the base sizes (of 5000 and 7500), we randomly created an additional 1 to 10% more nodes to ensure that the different graphs in the data set have slightly different numbers of nodes. As before, once the 10 graphs are obtained for each configuration, each of the subgraphs is used as query against the rest. We report the averages of execution time and accuracy.

8.2 Evaluation Criteria

All experiments were conducted using a 4-core Intel Core i5-2400, 3.10GHz, machine with 8GB memory, running 64-bit Windows 7 Enterprise. The codes were executed using Matlab 2013b and Visual Studio 2012. To evaluate accuracy, we use the matching quality defined in Section 3.

8.2.1 Execution Time

We report both offline and online execution times. As shown in Algorithm 1 in Section 3, for `KSGM`, the offline execution includes structural keynode and local-signature extraction steps. Online execution includes similarity computation, anchor selection, and expansion steps. [29] does not perform structural keynode extraction; instead, offline execution includes eigen-decomposition for global signatures.

For both `KSGM` and [29], we omit the refinement step as its complexity is prohibitive for scalable graph matching. For instance, for the Facebook graph for which `KSGM` takes ~ 6 seconds for online processing for a pair of graphs, refinement takes ~ 30 minutes – i.e., it causes a $\sim 260\times$ slowdown[5].

8.3 Experiment Parameters

The default parameters for the structural keynode extraction (`SKE`) algorithm are as follows:

- PageRank teleportation probability $(1-\alpha)$ = 0.15 (as is commonly assumed),
- least smoothing factor (σ_{min}) = 0.275, corresponding to $\sim$ 2-hop neighborhoods,
- maximum smoothing factor (σ_{max}) = 0.777, corresponding to $\sim$ 5-hop neighborhoods, and
- number (l) of smoothing levels = 6.

In addition, for `KSGM`, the default percentage (Θ) of keynodes selected from the graph was set to 3%. Also, for all algorithms, local signatures were extracted from 2-hop neighborhoods (i.e., $k = 2$), as recommended by the authors of [29].

8.4 Results for the Facebook Graph

8.4.1 Default Configuration

Table 2 lists the online and offline processing times and accuracy for the Facebook graph under the default parameter settings. As we see here, while `KSGM` spends more time in one-time, offline processing, its online matching time is $3\times$ faster than that of [29]. Moreover, the matching accuracy of `KSGM` is $1.2\times$ better than that of the competitor, through it does not use global signatures, nor it relies on the optimal Hungarian algorithm for anchor selection.

The table also lists the performance of `KSGM` when using top PageRank (PR) scored nodes instead of those returned by the `SKE` algorithm. As we see here, while the offline process is faster when using PageRank scoring nodes, the runtime performance (both in terms of execution time and accuracy) is worse when using `SKE` keynodes. In addition, to see whether it is possible to achieve a competitive accuracy if we were to select a similar percentage of node randomly, in Table 2, we also include results where random keynodes are used in the matching online phase. As we can see, the accuracy drops significantly when we use random keynodes instead of using robust structural keynodes extracted by the proposed `SKE` algorithm[6]. These indicate that `SKE` is indeed effective in extracting structurally distinct and useful keynodes.

Table 2: Experiment results for the Facebook Graph (default parameters)

	[29]	`KSGM`	PR	Random
Matching time (online, sec.)	19.2	**6.4**	7.25	6.0
Accuracy	35.4%	**38.0%**	34.12%	15.4%
Extraction time (offline, sec.)	11.5	110.4	0.39	-

Table 3: Impact of the keynode percentage, Θ, for the Facebook Graph

	2%	3%	4%	6%	8%
Matching time (online, sec.)	6.5	6.4	6.3	6.4	7.2
Accuracy	37.6%	38.0%	38.7%	36.4%	33.2%

Table 4: Impact of the node-pair ranking function, $\rho()$, for the Facebook Graph

	$\rho()$	w/o Degree	with Global
Matching time (online sec.)	6.4	6.0	4.0
Accuracy	38.0%	31.8%	22.9%

Table 5: Impact of the local-signature neighborhood size, k, for the Facebook Graph

	2 hops (default)	3 hops	4 hops
Matching time (online, sec.)	6.4	5.5	4.3
Accuracy	38.0%	33.9%	23.3%

8.4.2 Impact of the Keynode Percentage

Table 3 studies the impact of the percentage, Θ, of the nodes used as keynodes. As we see, up to a point, the more keynodes we use, the more accurate and faster the matching becomes. Beyond that point, however, additional keynodes become disadvantageous. This indicates that top keynodes are the most effective in serving as good starting points and, as expected, below a certain rank they loose distinctiveness, resulting in increased cost and loss in accuracy.

8.4.3 Impact of the Node-Pair Ranking Function

Table 4 studies the impact of the node-pair ranking function, $\rho()$. In particular, we compare the performance of the ranking function proposed in Section 6.1.1, to the ranking function without degree extension and ranking function including additional global-signature similarity as proposed in [29]. As we see here, the proposed node-pair ranking function provides the best expansion opportunities (and thus provides the highest accuracy, with slight expansion time overhead). Also, the "with Global" optional provides a much worse matching. Thus, while the algorithm allows, we encourage the users not to use "with Global" option.

[5]For this data configuration, when using expensive refinement post-processing, `KSGM` and [29]'s accuracies are 0.72 and 0.696, respectively.

[6]The slight time gain when using random keynodes is due to the fact that random keynodes are not good starting points for expansion and, thus, the expansion process ends earlier.

Table 6: Experiment results for the synthetic data sets (avg. degree=4, varying models and number of nodes)

	PL(5000)	PL(7500)	ER(5000)	ER(7500)
KSGM *Online time (sec.)*	6.9	13.9	6.5	14.5
[29] Online time (sec.)	99.3	223.7	746.7	745.1
KSGM *Accuracy*	45.3%	45.1%	45.7%	51.2%
[29] Accuracy	42.9%	42.8%	46.3%	53.0%
KSGM *Offline time (sec.)*	130.0	284.7	134.8	270.8
[29] Offline time (sec.)	23.8	82.8	24.9	84.5

Table 7: Impact of the average node degree (number of nodes=5000, power law model)

	degree=4	degree=8	degree=16
KSGM *Online time (sec.)*	6.9	7.5	9.3
[29] Online time (sec.)	99.3	116.7	141.2
KSGM *Accuracy*	45.3%	25.2%	13.9%
[29] Accuracy	42.9%	25.1%	14.1%
KSGM *Offline time (sec.)*	130.0	199.1	396.7
[29] Offline time (sec.)	23.8	27.9	53.2

8.4.4 Impact of the Neighborhood Size, k, for Local-Signatures

Table 5 studies the impact of the neighborhood size, k, for local-signature extraction. As we see in the table, the highest accuracy is at 2 hops[7], increasing the neighborhood size negatively affects the accuracy, indicating that unless locally meaningful signatures are used, the resulting node-pair ranking is not effective for expansion. This shows the keynode matching process is more accurate when keynodes are easy to localize and this requires them to be distinct and locally representative. Large neighborhoods potentially violate both. Note that this is in line with the observation in Table 4.

8.5 Results for the Synthetic Data Sets

In this subsection, we consider the impacts of graph topology, size, and node degree using ER and PL topologies. We omit discussions of the impacts of the other parameters, as they mirror those presented in Tables 3 through 5.

8.5.1 Default Configurations

Table 6 lists the performances of KSGM and [29] for synthetic graphs for different topologies and numbers of nodes under the default parameter settings. As we see here, the online execution time of KSGM is significantly ($10\times$ to $115\times$) faster than that of [29], especially for the ER topology. Moreover, on both Erdos-Renyi (ER) and power law (PL) topologies, the accuracy is highly competitive, with KSGM providing non-negligible accuracy gains for the PL model (where it is relatively easier to identify effective keynodes).

8.5.2 Impact of Average Node Degree

Table 6 studies the impact of average node degree on matching accuracies for the power law graph. As we see in the table, both algorithms see a drop in the matching accuracy with larger node degrees. However, KSGM stays competitive in terms of accuracy, whereas it provides more gains in terms of online execution time.

9. CONCLUSIONS

Noticing that existing solutions to the graph matching problem face major scalability challenges, we argue that it is impractical to seek alignment among all pairs of nodes. Given these observations, in this paper, we first presented an offline *structural keynode extraction* (SKE) algorithm and then discussed how to use these structural keynodes in a novel *keynode-driven scalable graph matching* (KSGM) algorithm. *Keynodes* are selected carefully especially because a post refinement step is not feasible due to scalability requirements. Experiment results show that the proposed KSGM algorithm works faster than the state-of-the-art algorithms without refinement, yet produces alignments that are as good or better.

[7]Coincidentally, this also is the scale at which the SKE algorithm located an overwhelming majority of the keynodes for this graph.

Acknowledgments

We thank the authors of [29] for sharing their source code and data.

10. REFERENCES

[1] http://networkx.github.io/
[2] http://snap.stanford.edu/index.html
[3] X. Bai, H. Yu, and E. R. Hancock. Graph matching using spectrament. ICPR 2004.
[4] A. Balmin, et al. ObjectRank: Authority-based keyword search in databases. VLDB, 2004.
[5] M.G. Borgatti, et al. Network measures of social capital. Connections 21(2):27-36, 1998.
[6] C. Borgs, M. Brautbar, J. T. Chayes, S.-H. Teng. Multiscale Matrix Sampling and Sublinear-Time PageRank Computation. Internet Mathematics 10(1-2): 20-48, 2014.
[7] S. Brin, et al. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30: 107-117, 1998.
[8] H. Bunke. Error correcting graph matching: On the influence of the underlying cost function. IEEE TPAMI, 21(9):917–922, 1999.
[9] K. S. Candan, R. Rossini, M. L. Sapino, X. Wang. sDTW: Computing DTW Distances using Locally Relevant Constraints based on Salient Feature Alignments. PVLDB, 1519-1530, 2012.
[10] M. Chen, J. Liu, and X. Tang. Clustering via random walk hitting time on directed graphs. AAAI 2008.
[11] Xilun Chen, K. Selcuk Candan. LWI-SVD: Low-rank, Windowed, Incremental Singular Value Decompositions on Time-Evolving Data Sets. KDD 2014.
[12] Xilun Chen, K. Selcuk Candan. GI-NMF: Group Incremental Non-Negative Matrix Factorization on Data Streams. CIKM 2014.
[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. 2001.
[14] R. Giugno and D. Shasha. Graphgrep: A fast and universal method for querying graphs. ICPR, pp. 112-115, 2002.
[15] W. S. Han, J. Lee, and J. H. Lee. TurboISO: Towards ultrafast and robust subgraph isomorphism search in large graph databases. SIGMOD 2013
[16] R. Jonker and T. Volgenant.Improving the Hungarian assignment algorithm. Oper. Res. 171-175. 1986.
[17] G. Karypis and V. Kumar "A fast and high quality multilevel scheme for partitioning irregular graphs". SIAM Journal on Scientific Computing 20 (1), 1999.
[18] D. Knossow, A. Sharma, D. Mateus, and R. Horaud. Inexact matching of large and sparse graphs using laplacian eigenvectors. GbRPR, 2009.
[19] W.-J. Lee and R. P. W. Duin. An inexact graph comparison approach in joint eigenspace. In SSPR/SPR, 35-44, 2008.
[20] Jundong Li, Xia Hu, Jiliang Tang, Huan Liu. Unsupervised Streaming Feature Selection in Social Media. CIKM 2015
[21] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. Int. Journal of Computer Vision, 60, 2, 2004.
[22] K. Riesen, X. Jiang, and H. Bunke. Exact and inexact graph matching: Methodology and applications. Managing and Mining Graph Data, pages 217-247, 2010.
[23] S. Umeyama. An eigen decomposition approach to weighted graph matching problems. IEEE TPAMI, 10(5):695-703, 1988.
[24] J. R. Ullman. An algorithm for subgraph isomorphism, JACM Vol. 23, No. 1, pp. 31-42. 1976
[25] X. Wang, K. S. Candan, M. L. Sapino: Leveraging metadata for identifying local, robust multi-variate temporal (RMT) features. ICDE, 2014.
[26] White D.R., et al. Betweenness centrality measures for directed graphs. Social Networks, 16, 335-346,1994.
[27] R. Williams. Faster all-pairs shortest paths via circuit complexity. STOC, 664-673. 2014.
[28] M. Zaslavskiy, F. R. Bach, and J.-P. Vert. A path following algorithm for the graph matching problem. IEEE Trans. Pattern Anal. Mach. Intell., 31(12):2227-2242, 2009.
[29] Y. Zhu, L. Qin, J. X. Yu, et al. High Efficiency and Quality: Large Graphs Matching. CIKM, pp. 1755-1764. 2011.