On Dimensionality Reduction of Massive Graphs for Indexing and Retrieval

Charu C. Aggarwal *1, Haixun Wang #2

*IBM T. J. Watson Research Center
Hawthorne, NY 10532, USA
1 charu@us.ibm.com

Microsoft Research Asia
Beijing, China
2haixunw@microsoft.com

***Abstract*—In this paper, we will examine the problem of dimensionality reduction of massive disk-resident data sets. Graph mining has become important in recent years because of its numerous applications in community detection, social networking, and web mining. Many graph data sets are defined on massive node domains in which the number of nodes in the underlying domain is very large. As a result, it is often difficult to store and hold the information necessary in order to retrieve and index the data. Most known methods for dimensionality reduction are effective only for data sets defined on modest domains. Furthermore, while the problem of dimensionality reduction is most relevant to the problem of massive data sets, these algorithms are inherently not designed for the case of disk-resident data in terms of the order in which the data is accessed on disk. This is a serious limitation which restricts the applicability of current dimensionality reduction methods. Furthermore, since dimensionality reduction methods are typically designed for database applications such as indexing, it is important to design the underlying data reduction method, so that it can be effectively used for such applications. In this paper, we will examine the difficult problem of dimensionality reduction of graph data in the difficult case in which the underlying number of nodes are very large and the data set is disk-resident. We will propose an effective sampling algorithm for dimensionality reduction and show how to perform the dimensionality reduction in a limited number of passes on disk. We will also design the technique to be highly interpretable and friendly for indexing applications. We will illustrate the effectiveness and efficiency of the approach on a number of real data sets.**

I. Introduction

The area of graph mining has numerous applications in a number of domains such as computational biology, chemical applications, the web, and social networking. In recent years, a number of data mining and management applications have been designed in the context of graphs and structural data [1], [3], [4], [8], [5], [12], [14], [23], [24], [25]. A detailed discussion of graph mining algorithms may be found in [3].

The problem of dimensionality reduction has been widely studied in the multi-dimensional domain [6], [10], [18]. Dimensionality reduction is a useful tool for reducing the size of the data for various database applications such as indexing and retrieval. In this paper, we will examine the problem of dimensionality reduction of massive disk-resident graphs. Graph applications typically arise in one of two scenarios:

- In one scenario, the different graphs may be drawn on a limited domain, and the size of the graph data is quite modest. Such cases are common in applications such as chemical analysis or biological compound analysis. Many of the currently developed mining algorithms [23], [24] correspond to this scenario. However, this scenario is not quite as relevant to the problem of dimensionality reduction, which is designed for graphs that are drawn on a massive domain.
- In a second and more challenging scenario, the *base node domain* is drawn on a massive set of nodes. In this case, node labels are typically key identifiers drawn across a very large universe of possibilities. For example, the node labels may correspond to URL addresses in a web graph [15], the IP-addresses in a communication network, or the user identifiers in a social network. The number of graphs can be very large, and are typically drawn over different subsets of nodes from the base domain. Since the number of edges can be the square of the number of nodes, the total number of distinct edges (over the different graphs) may be too large to store effectively. Currently developed mining algorithms are not very effective for this scenario.

In this paper, we study the second scenario in which the graphs may be drawn on a *massive domain* of nodes. Furthermore, it is assumed that the data set is too large to be stored in main memory. Therefore, it is extremely important to design the algorithm effectively in order to work well in the disk-resident scenario. Such cases are very common in many web applications such as social networking or communication network analysis, in which the nodes are associated with *a massive domain of* unique identifiers. Examples of such identifiers could be a web address URL, an IP-address, or a user identifier in a social network. Real data sets are often *sparse*, which implies that the individual data sets satisfy the sparsity property. Our goal is to develop a reduction which can be performed efficiently in the massive-domain and disk-resident scenario, and which continues to retain its usefulness for indexing and retrieval applications. A common solution to graph dimensionality reduction is that of Singular Value

Decomposition (SVD) [18], in which the edge correlation structure is used in conjunction with matrix decomposition. However, this approach does not work very well with massive graphs because of the following reasons:

- Since the number of nodes is very large, it is extremely difficult to use techniques such as SVD, which requires diagonalization of an extremely large covariance matrix.
- The interpretability of the reduced representation is very low. This is a disadvantage in many real applications in which it is desirable to retain the interpretability of the reduced representation with respect to the original data.
- Many applications such as communication analysis and social networking create a huge volume of data which needs to be stored on disk. The dimensionality reduction technique must continue to retain its efficiency and usability in such scenarios. This can be achieved only by designing the algorithm such that the number of passes over the data is limited.

Some recent methods attempt to perform matrix-decomposition and dimensionality reduction [16], [21], [20] of large graphs. However, these methods are designed for **single graphs** rather than generating an index-friendly representation of multiple graphs in a massive domain. Furthermore, these methods are not designed for the disk-resident and massive-domain case. The disk-resident assumption is especially important, since all large databases are stored with a disk-resident representation.

In this paper, we will design an efficient algorithm for dimensionality reduction of massive graph data sets. We will mine important structural concepts from the data, and the entire graph is represented as a function of these concepts. Furthermore, the transformed data has the property that it activates only a small proportion of these concepts. Therefore, most of the graphs can be efficiently maintained in this representation. Furthermore, this representation also maintains its interpretability in terms of the original graph. We will show that such a representation can be effectively used in the context of a variety of database applications.

This paper is organized as follows. In the next section, we will propose a sampling-based algorithm for graph dimensionality reduction. We will present theoretical results which bound the disk efficiency of the method, and present results which indicate its effectiveness. We will also discuss how the dimensionality reduction technique can be used for applications such as indexing. Section 3 contains the experimental results. Section 4 contains the conclusions and summary.

II. Dimensionality Reduction of Massive Graphs

Before describing the dimensionality reduction algorithm we will define some notations and definitions. We assume that the data contains graphs which are denoted by $G_1 \ldots G_r \ldots$. The labels of the nodes in the graphs are defined over a node label set $\mathcal{N}$, which is assumed to be massive. The number of nodes in $\mathcal{N}$ is denoted by N. We assume that the edges on the graphs are undirected, though the broad approach can be easily generalized to the case of directed graphs. While the base set $\mathcal{N}$ may be very large, each individual graph may be defined only over a subset of the node set $\mathcal{N}$. This is often the case in many real applications in which the *base network* may be very large, but the underlying graphs may be defined over a pattern of activity in a local region in the graph. For example, in a social network, a small set of users may interact with one another at a given time in a local region of the graph. Thus, in practical applications, such graphs satisfy the *sparsity property* [20], which is common to many application domains. We will see that this sparsity property is important from a practical perspective.

In this paper, we will mine the underlying *structural concepts* from the graph representation. We will show that such concepts will create a representation which reflect the broad characteristics of a given graph in the data set with the use of a multi-dimensional format. Furthermore, this multi-dimensional representation retains the sparsity property in the sense that only a small fraction of the multi-dimensional values take on non-zero values. This ensures that it is much simpler to use a variety of sparsity-based data structures such as the inverted index in order to perform effective storage and retrieval of the data. Next, we will define the *basis structure* for the underlying data representation.

Definition 1: The basis structure is defined as a set of graphs $H_1 \ldots H_l$, the edge sets of which are disjoint from one another. We assume that the edges in the graphs $H_1 \ldots H_l$ are weighted, and the weights correspond to the relative edge frequencies. The frequency of the edge with node labels X and Y in $\mathcal{N}$ in the graph H_j is given by $F(X, Y, H_j)$. In the event that the edge (X, Y) does not exist in H_j, we assume that the corresponding frequency is 0.

We note that this basis is orthonormal, when the graphs $H_1 \ldots H_l$ are *edge-disjoint*. When the graphs are edge-disjoint, the dot product on the corresponding edge frequencies is 0. In other words, we have:

$$\sum_{(X,Y) \in H_i \cup H_j} F(X, Y, H_i) \cdot F(X, Y, H_j) = 0 \qquad (1)$$

In addition, for ease in interpretability, we assume that the graphs $H_1 \ldots H_l$ are node disjoint. This also provides a clear understanding of the different identifiers included in a particular basis, and can be easily expressed in terms of localized regions of the graph. The ease in interpretability can also be useful for a number of applications in the dimensionality reduction process. Next, we define the coordinates of a graph in this basis representation. Let $n(G_j)$ be the number of edges in the graph G_j. Then, the coordinate $c(G_j, H_i)$ of the graph G_j along the concept H_i is defined as follows:

$$c(G_j, H_i) = \frac{\sum_{(X,Y) \in G_j} F(X, Y, H_i)}{\sqrt{n(G_j)}} \qquad (2)$$

Thus, the coordinate of the graph G_j along a concept is simply the sum of the corresponding edge frequencies of the concept graph along the edges included in G_j. In addition, a normalization factor of $\sqrt{n(G_j)}$ is used in the denominator.

This is essentially a representation of the dot product between the graph G_j and the concept H_i. We define the corresponding *conceptual representation* as follows:

Definition 2: The conceptual representation of the graph G_j along the conceptual basis $\{H_1 \ldots H_l\}$ is defined by the coordinate set $(c(G_j, H_1) \ldots c(G_j, H_l))$. Each of the coordinates $c(G_j, H_i)$ is defined by Equation 2.

We note that in many real applications, the individual graph G_j may be sparse, and may therefore represent a small portion of the underlying domain. In such cases, only a small fraction of the coordinates take on non-zero values. Therefore, in order to improve the storage requirements for the reduced representation of the graph, it is possible to store the concept identifiers together with their coordinate values. Furthermore, the interpretability of the concepts is clearer in this case than that of matrix decomposition methods, since the concepts are defined in terms of a base set of graphs $\{H_1 \ldots H_l\}$.

A. Constructing the Basis Structure by Sampling

In this section, we will discuss the process of construction of the basis structure for dimensionality reduction. Let us consider the construction of the basis structure of a data set of fixed size n containing the graphs $\{G_1 \ldots G_n\}$. The value of n may typically be quite large. Furthermore, the node set size $N = |\mathcal{N}|$ may be very large, and therefore the data may need to be stored on disk. Before discussing details of the construction, we will discuss the most important desiderata for a basis structure, which are those of *space-requirements* and *basis locality*. Clearly, we would like to retain only a small subset of representative edges from the original graph, so as to optimize the space requirements for the basis structure. While the total number of distinct edges can be as many as the square of the number of nodes, we would like the space requirements for the basis structure to be significantly lower. The notion of basis locality corresponds to the fact that we would like each graph G_i to be described completely by as few components from the basis as possible. We refer to an edge in G_i as a *bridge edge*, if one end of the edge lies in one partition, and the other end lies in a different partition. We also count an edge as bridge edge, if one of the nodes at the end of the edge is not contained in any partition. We need to specify this special case, since our sampling algorithm may sometimes not pick some of the nodes in the graph. Clearly bridge edges result from the graph being defined by multiple components in the basis. Thus, we would like to choose a basis which minimizes the number of bridge edges in $G_1 \ldots G_n$:

Definition 3: **Coverage:** Create a graph-partitioning $H_1 \cup H_2 \ldots \cup H_l$, which minimizes the number of bridge edges (counting duplicates in the different graphs as distinct edges) in $G_1 \cup \ldots \cup G_n$, defined by the basis $H_1 \ldots H_l$.

We note that this is a multi-graph variation on the graph partitioning problem. While the graph-partitioning problem is well known to be NP-hard, this particular variation is even more difficult since the data is not available in main memory, and the nodes cannot be accessed randomly without increasing the cost of the algorithm significantly. While graph-partitioning is a widely studied problem [13], the massive size of the graph and the disk-resident scenario pose significant challenges which are not encountered in the standard version of the problem. Therefore, we will define a Monte Carlo sampling-based algorithm which can be effectively implemented for a disk-resident graph. Then, we will show how to use this reduced graph effectively for indexing. Next, we will provide an overview and description of the dimensionality reduction algorithm.

B. Dimensionality Reduction for Massive Graph Data Sets

We will design our dimensionality reduction algorithm, with an eye towards limiting the number of passes over the data set. This is particularly important in the case of structural data, since random access to edges would result in a very high disk I/O cost. Let n be the number of graphs in the data set, and M be the total number of edges over all graphs. We assume that duplicates are counted distinctly. Clearly, since we assume that each graph must contain at least one edge, and the number of edges is at least equal to the number of nodes. Therefore, we have $M \geq n$ and $M \geq N$. The process of determining the total number of edges and creating a sample of edges requires one pass over the data. Since the basis is assumed to contain l components, we use a *contraction based sampling approach* in order to determine the best basis. A straightforward contraction-based approach [22] is often used for determining minimum 2-ways cuts of memory-resident graphs. This technique is however not relevant to the disk-resident case, since it makes random accesses to disk. Therefore, we will design a new contract-based approach which is able to apply this broad class of techniques to the problem of basis construction in *disk resident graphs*. Our approach will carefully reconstruct the contraction process into *sequential phases* in order to limit the number of passes over the disk-resident data. We will then use a theoretical potential function argument in order to bound the number of sequential phases over the data set.

The overall approach is as follows. Let E be the union of the edges in $G_1 \ldots G_n$ for the nodes set $\mathcal{N}$ with cardinality N. We assume that E is allowed to contain duplicates (or appropriately weighted edges). The algorithm proceeds in a number of *sequential phases*, each of which requires a pass over the disk. In each sequential phase, we sample a set of N edges, where N is the total number of nodes in the current graph. We construct the set of connected components induced by this set of N edges. We contract each such connected component into a single node. Clearly, the process of contraction can create self-edges. Self-edges are those edges for which both ends are the same (contracted) node. We eliminate all "self-edges" after the sequential phase of contracting the underlying connected components. We however allow duplicate edges which are created by the contraction. We note that duplicate

Algorithm *ContractBasis(Input: $G_1 \ldots G_n$, Target Dim: l)*
begin
$E =$ Union of edges in $G_1 \ldots G_n$
{ N is the number of nodes }
$S_0 = \{\}$;
repeat
S= Sample of N edges from E in random order;
Construct graph defined by edge sample S from E;
while graph defined by S has less than l components
{ Executed only in last iteration in case over-contraction
has occurred to less than l components and
we want a basis of size exactly l; }
randomly delete an edge from S;
Contract each component in sub-graph defined by
S into one node;
Eliminate any self edges in contracted nodes;
Reset N to new number of reduced nodes;
Reset E to new set of reduced edges;
$S_0 = S_0 \cup S$;
{ In the event that an edge in S corresponds to
one between contracted node(s), we retain the mapping
to original node set in order to add to S_0. Therefore,
S_0 is defined in original node set $\mathcal{N}$; }
until l nodes remain;
return the l-component sub-graph $H_1 \ldots H_l$
defined by edge set S_0;
end

Fig. 1. Creating a basis structure

edges[1] result in an implicit bias in the sampling during future iterations. After the contraction process, let $N_1 < N$ nodes remain. Then, we sample N_1 edges and repeat the contraction approach. We repeat the process until at most l connected components remain. These l connected components constitute the basis for the algorithm. The overall algorithm *ContractBasis* is illustrated in Figure 1. The approach in the algorithm *ContractBasis* is repeated k times and the optimal basis is picked according to the condition of Definition 3. We will discuss how the value of k is picked later. First, we will analyze the efficiency of the *ContractBasis* algorithm with the use of this phased approach.

An immediate question is to why the contraction algorithm should work well. We note that we would like a basis which has the least number of bridge edges across the partition. Since each edge is sampled uniformly at random, partitions which have a large number of bridge edges between them are much more likely to be forced to contract the corresponding nodes in them into a single node. Intuitively, such partitions will not survive the contraction process. The use of k instances of *ContractBasis* further eliminates the poorly designed basis structures. In a later section, we will provide a formal argument, which quantifies the expected effectiveness of the contraction-based method.

Furthermore, the *ContractBasis* algorithm has a number of structural properties which can be used to probabilistically bound the number of passes over the data set. The *ContractBasis* algorithm performs multiple passes over the data, and in each pass, the algorithm performs a contraction. Therefore, the number of passes over the data is equal to the number of contractions. Restricting the number of passes over the data is critical in improving the I/O efficiency of the algorithm. This is extremely important in large data sets. We will use a *potential function argument* in order to bound the number of passes over the data set. Potential function arguments are frequently used in combinatorial optimization to bound the efficiency of different kinds of algorithms.

We note that each contraction results in reduction of both the number of nodes and edges in the contracted graph. The reduction of the number of nodes is because of the contraction process, and the reduction of the number of edges is because of the elimination of self-edges. Therefore, we define the potential function Φ on the edge set E and the cardinality N of the node set as follows:

$$\Phi = |E| \cdot N \tag{3}$$

The note that E and N are defined on the graph $G_1 \cup \ldots \cup G_n$, and count the duplicate edges between nodes (created by the contraction process) distinctly. However, they do not include self-edges, since they are eliminated by contraction. Note that in our randomized algorithm, the values of Φ, $|E|$, and N are all random variables which monotonically reduce because of the contraction process. We make the following claim.

Lemma 1: The expected value of the potential function Φ at the end of one contraction iteration is less than half of its value at the beginning of the iteration from one contraction to the next.

Proof: In each iteration, we sample N edges, where N is the number of nodes at the beginning of an iteration. Let us order these edges in any way, and add them to the basis sequentially, thereby reducing the number of connected components. We note that the process of adding an edge may not necessarily result in the reduction of the number of connected components. In the event that an edge is sampled between two nodes which are already part of the same connected component (based on edges sampled in current iteration), the number of nodes does not reduce. On the other hand, if the edges are part of different connected components, the number of nodes reduces by 1. Therefore, the probability of the number of nodes reducing by 1 depends upon the fraction of edges which are bridge-edges between the connected components at the time of addition. We consider two cases:

Case I: *Throughout the contraction iteration, (during the sequential addition of the N edges), at least 50% of the edges in $G_1 \cup \ldots \cup G_n$ are bridge-edges, when defined across the connected components induced by the current set of added edges:* In this case, the probability of a reduction in the number of connected components from each edge addition is greater that 0.5, since all of the N edges were sampled uniformly at random from $G_1 \cup \ldots \cup G_n$. Therefore, the expected number of nodes reduced by the addition of N

[1]If desired, duplicate edges can be replaced with an edge with weight corresponding to the number of duplicates. The weight can be used in order to bias the sampling process.

edges is at least $N/2$. This means that the expected value of the potential function Φ is also reduced by a factor of at least 2, since the total number of edges E also reduces in a contraction phase because of elimination of self-edges.

Case II: *At some point in the contraction iteration, (during the sequential addition of the N edges), less than 50% of the edges in $G_1 \cup \ldots \cup G_n$ are bridge-edges, when defined across the connected components induced by the current set of added edges:* We note that any edge which is not a bridge-edge will become a self-edge after the contraction. Such edges are eliminated at the end of a contraction phase. Since greater than 50% of the edges are non-bridge edges, this means that the value of E will reduce by a factor of at least 2 after the contraction phase. Furthermore, since N reduces because of the contractions, it follows that the value of Φ will reduce by a factor of at least 2 in this iteration. We note that in this second case, the reduction by a factor of at least 2 is not just in terms of expected value, but is deterministically guaranteed.

This completes the proof. ■

The above behavior provides us with an idea of the number of iterations in which the expected number of components reduce to the value of l.

Theorem 1: At most $\log(N) + \log(M) - log(l)$ iterations are required in order to reduce the *expected* number of nodes to the basis size of at most l by successive contractions.

Proof: We note that the value of Φ is bounded above by $N \cdot M$, where M is the number of edges (counting duplicates distinctly). Furthermore, if the value of Φ is less than l, this would imply that there are fewer than l connected components. Since the expected value of Φ reduces by a factor of 2 in each iteration, it follows that after $\log(N \cdot M/l) = \log(N) + \log(M) - log(l)$ iterations, the expected value of Φ is less than l. Therefore, the expected number of components after such iterations is also at most l. ■

The above result is important, since each contraction phase requires two passes over the data (one for sampling, and one for contraction and elimination of self-edges). Therefore, the expected number of passes is at most $O(\log(N)+\log(M)) = O(\log(M))$. Note that these are worst-case results in expectation, and therefore the algorithm is likely to be much more efficient in practice. While the above results are in expectation, we would also like to provide a probabilistic *guarantee* on the maximum number of passes required in order to reduce to the required number of connected components.

Theorem 2: Let δ is any arbitrarily small probability value. After at most to $\log(N)+\log(M)-log(l)+\log(1/\delta)$ iterations, the number of contracted nodes is at most l with probability at least $(1-\delta)$.

Proof: Let V be the random variable which represents the number of connected components after $\log(N) + \log(M) - log(l) + \log(1/\delta)$ contraction iterations. By using the methodology of the proof of Theorem 1, we can show that the $E[V] < l \cdot \delta$. By using the Markov inequality, we have:

$$P(V > l) \leq E[V]/l < \delta \tag{4}$$

The result follows. ■

The above results bound the number of passes to $O(\log(M))$ with high probability. A second question which arises is the computational complexity in terms of the CPU time. The main bottlenecks in a contraction phase are the time required for sampling and the time required for contraction of the nodes. The contraction process requires us to eliminate self-edges and re-write the adjacency matrices of the different graphs. Therefore, then a given contraction phase requires $O(|E|) \leq O(M)$ time. This means that the computational complexity over all passes is $O(M \cdot \log(M))$ with high probability. We summarize as follows:

Lemma 2: The computational complexity of the *Contract-Basis* algorithm is $O(M \cdot \log(M))$ with high probability, where M is the sum of the number of edges in the graphs $G_1 \ldots G_n$.

In practice, the computational complexity is much lower. This is because the first contraction phase takes the most time, and is the real bottleneck for the algorithm. In subsequent iterations, the size of the super-graph $G_1 \cup \ldots G_n$ reduces geometrically because of contractions. In each iteration, the value of $|E|$ reduces from the last iteration. The overall reduction of the graph in terms of number of nodes also typically reduces the number of edges by the same factor or greater. This is because when the number of nodes reduce by a factor of 2 (Case I of Lemma 1), the number of edges typically reduce by a factor of much greater than 2 because of elimination of self-edges. In Case II, the number of edges are guaranteed to reduce by a factor of 2, and this case is the more likely case in skewed graphs. This essentially means that the running time in each contraction phase is less than half the running time of the previous phase, and the overall computational time often turns out to be linear in the number of edges M in practice. This is the best case practical behavior of the running time, since M is also the asymptotic size of the input, and forms a lower bound on the computational complexity.

C. Qualitative Issues

This section will discuss the qualitative analysis of the contraction technique. Since the problem of optimal basis construction is NP-hard, we do not expect our probabilistic algorithm to be optimal in expectation on general graphs. However, we can show that in the case of very skewed graphs containing clear correlations between groups of nodes, it is possible to obtain effective results. This is because of the nature of the contraction approach which quickly destroys cuts with a large number of edges, because each edge is sampled uniformly. On the other hand, the contraction approach probabilistically spare cuts with few edges. In skewed graphs, the ratio of the edge cut of the optimal basis to the total weight M of the edges is very low. Intuitively, such graphs are also very effective for the contraction approach. While this section does not provide a formal proof to this effect, we will provide an

intuitive analytical argument of why such an algorithm should work well. We will also provide experimental results which show that the algorithm does indeed work well.

The *ContractBasis* algorithm proceeds in a series of $t = O(\log(M))$ phases, during which the sizes of the node and edge sets become progressively smaller. Let the sizes of the node and edge sets at the beginning of these t phases be given by $N_1 \ldots N_t$ and $M_1 \ldots M_t$ respectively. Let us examine a particular basis which contains a total of at most C^0 edges. Then, the probability that none of the edges from the optimal basis are contracted in the ith phase is given by at most $(1 - C^0/M_i)^N$. Therefore, the probability that none of the edges from the optimal basis are contracted in any phase is given[2] by

$$\pi_{i=1}^{t}(1 - C^0/M_i)^{N_i} \approx \pi_{i=1}^{t} e^{-C^0 \cdot N_i/M_i}$$

Here e is the base of the natural logarithm. Let us denote

$$F = \sum_{i=1}^{t} C^0 \cdot N_i/M_i$$

Therefore, the corresponding probability is e^{-F}.

We note that this probability can be quite small, since the value of F occurs in an exponent. This is to be expected, since the problem is NP-hard. However, in a practical application, it is acceptable to obtain an *approximately adjacent cut* to the optimal one. Two cuts are said to be r-adjacent, if after at most r node transfers from the node set from one basis to the other, the optimal basis can be constructed. If the data is sufficiently skewed (as if often the case for real data), then approximately adjacent cuts can provide very effective results. Let us assume that C^0 represents an upper bound on the quality of any r-adjacent cut to the optimal cut. While C^0 does not represent the quality of an optimal cut, it is good enough for most practical purposes in a real applications. For an l-partitioning, there are $(N \cdot l)^r$ possible r-adjacent partitions. Then, we can show that at least one of these r-adjacent cuts are arrived at by the algorithm with probability at least $e^{-F} \cdot (N \cdot l)^r$. If we repeat the process over k applications of the *ContractBasis* algorithm, then the probability that we find a cut which is at least as good as some r-adjacent cut is given by at least $1-(1-e^{-F} \cdot (N \cdot l)^r)^k$. We note that if the value $e^{-F} \cdot (N \cdot l)^r$ is as close to 1 as possible, then by using a modest number k of applications of this procedure, we can guarantee a cut with quality at least C^0 with high probability. We would like to find the range of values of r over which a modest probabilistic guarantee is possible. In order to find the minimum value of r for which this is true, we need the following relationship to hold true:

$$(N \cdot l)^r = \alpha \cdot e^F \tag{5}$$

Here α is some modest fraction (say 0.5), so that the overall probability of success is at least $1 - (1 - \alpha)^k$. We note that even for modest values of k, this probability can be quite high because of the exponential dependence on k. By taking the logarithm of both sides, we obtain the following value for r:

$$r = (F + \ln(\alpha))/(\ln(N) + \ln(l)) \tag{6}$$

In order to obtain an typical idea of the typical value of r in a large scale application, let us consider a graph with $N = 10^5$ nodes and $M = 10^8$ edges. In such a case, the value of t is at most $\log_2(10^8 \cdot 10^5) = 43$. Let us consider graphs in which C^0 is relatively small and is given by $C^0 = 10^6$. We approximate F assuming that $N_i/M_i \approx 10^{-3}$. In such a case $F = 10^6 \cdot 10^{-3} \cdot 43 = 43000$. Let us also assume that we wish to determine a basis with $l = 10^3$ nodes. In such a case, the above computation yields $r = 2330$. To put this in perspective, approximately 2.33% of the nodes are displaced from their correct basis on the average. Since we use $l = 1000$, each basis will contain 100 nodes on the average, and approximately two or three of those nodes may not be the correct nodes on the average. This level of inaccuracy is quite acceptable in many practical scenarios.

D. Indexing with the Reduced Representation

Unlike other dimensionality reduction methods, this technique naturally lends itself to indexing. This is because such a technique yields a basis which is such that the corresponding projected coordinates are sparsely populated. This means that only a small number of coordinates take on non-zero values in this system. This allows us to use inverted representations in order to construct an index on the underlying data. For each graph G_i, we construct the conceptual representations, as suggested by Definition 2. Since each graph G_i is drawn over only a modest subset of the nodes from a massive domain, and the basis structure $H_1 \ldots H_l$ also creates a partitioning, it follows that the corresponding conceptual cooordinates $(c(G_j, H_1) \ldots c(G_j, H_l))$ take on non-zero values (or highly positive values) on only a small fraction of the nodes. Some of the non-zero values may be created by the noise in the individual graphs. In order to create a more efficient representation, we create the conceptual indexing representations with a noise threshold ϵ.

We can construct an inverted representation of the graph data, in which for each possible basis set of nodes, we have a list of graph identifiers along with the corresponding conceptual coordinate values along that basis. In a given inverted list, we include only those graph identifiers for which the corresponding coordinate value is at least ϵ. This inverted representation can be used to resolve queries in a very efficient way. First, we note that the inverted representation is very compact, because it compresses the structural information conceptually without holding information about individuals. Standard query processing techniques in information retrieval [19] which are used with the inverted representation can also be used in this case. If desired, the approach can also be used to filter the query down to a small number of candidate graphs on which the similarity is measured explicitly.

[2]The last approximation arises from the fact that $(1 - 1/n)^n = 1/e$.

III. Experimental Results

In this section, we evaluate the effectiveness and the performance of our approach. We implemented our algorithm in C++. In addition, we adapted a well known, memory-based graph partitioning algorithm (the FM algorithm [7]) for disk-based data. The results presented here are obtained from experiments conducted on a Windows machine with a 1.7 GHz CPU and 1 GBytes of memory.

A. The Baseline Approach

Our technique uses a graph partitioning approach for dimensionality reduction. To the best of our knowledge, there is no previous work on partitioning disk-resident graphs. On the other hand, a lot of research has been devoted to partitioning memory resident graphs. For performance comparison purpose, we adapt a widely-used, memory-based graph partitioning algorithm for disk-resident data.

There are a class of local refinement algorithms that bisect a graph into even size partitions. Most of them originated from the Kerninghan-Lin (KL) partitioning refinement algorithm [13]. The KL algorithm incrementally swaps vertices among partitions of a bisection to reduce the edge-cut of the partitioning, until the partitioning reaches a local minimum. There are many variations based on the KL algorithm, including the FM algorithm [7], and the multi-level partitioning algorithm [11].

The alternative algorithm we choose to compare with our approach in this paper is the FM (Fiduccia-Mattheyses) algorithm [7]. FM is an improved version of the KL algorithm, and is very commonly used for bisection refinement. The algorithm works as follows. For each vertex v, the FM algorithm computes the gain achieved by moving v to the other partition. These vertices are maintained by two hash tables (the key of the hash table is their gains) one for each partition. An important improvement of the FM algorithm over the KL algorithm is that it does not recompute the gains of each node after each round of moving. It is for this reason we choose FM over KL, as recomputing the gains will be extremely costly if the data is disk resident. Initially all vertices are free to move to the other partition. The algorithm iteratively selects a free vertex v with the largest gain from the larger partition, and moves it to the other partition. When a vertex v is moved, it is "locked" and the gain of v's neighboring vertices are updated, which means they will be hashed into new positions in the hash table. In each pass, once a vertex is moved, it is not allowed to move again, since this may result in threshing (i.e., repeated movement of the same vertex that has the highest gain). After each vertex movement, the algorithm also records the size of the cut achieved at this point. A single pass of the algorithm ends when there are no more free vertices (i.e., all the vertices have been moved). Then, FM selects the point where the minimum cut was achieved, and all vertices that were moved after that point are moved back to their original partition. This becomes the initial partitioning for the next pass of the algorithm.

We modify the FM algorithm in two aspects. First of all, the FM algorithm is a bisection algorithm. To use FM algorithm to partition a graph into arbitrary number of partitions, we invoke it recursively. For instance, to partition a 100-node graph into 5 parts, we first bisect the graph into a 40-node subgraph and a 60-node subgraph. Then, we recursively partition the 40-node subgraph into 2 parts, and the 60-node subgraph into 3 parts. Second, we adapt the FM algorithm so that it works with disk resident data. We describe the data sets below.

B. Datasets

In our experiments, we use three datasets. Each data set contains a set of graphs, and is represented by a sequence of edges in the form of $\langle gid, from, to, weight \rangle$, where gid is the id of the graph the edge belongs to, $from$ and to are the node id of beginning and the ending node of the edge, and $weight$ is the weight of the edge. The set of edges is disk-resident.

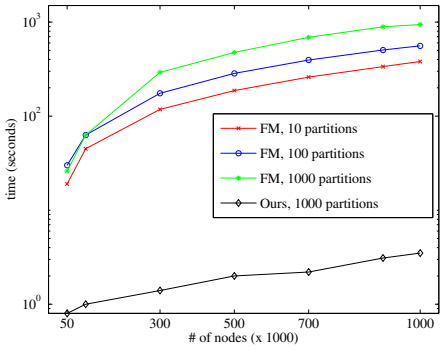
Synthetic Data Set: We use a synthetic data generator to produce a set of graphs whose nodes are in the domain of $1 \cdots N$, where N is a parameter given by the user. To simulate concepts in the graph, we partition all possible edges among the N nodes into a number of groups, each representing a concept. When generating a graph, we first pick k concepts. Then, we add edges one by one to the graph. With high probability p (p is close to 1), each edge belongs to one of the k concepts, and with probability $1 - p$, the edge is outside the k concepts. The number of graphs and the number of edges in each graph determines the density of the synthetic data set, which is also controlled by the user.

DBLP: We convert DBLP [3] into a set of graphs, where each paper constitutes a graph. In the graphs, authors were defined as nodes, and co-authorship in a particular paper was defined as an edge. The entire set contains 357,126 nodes (unique authors), 975,951 edges (co-authorship), and 355,278 groups (papers). Thus, this is a particularly sparse graph because the total number of edges is small with regard to total number of nodes. Furthermore, each individual graph also contains very few nodes. Yet, the data set may contain some concepts; for example, certain authors may specialize in research in certain topics.

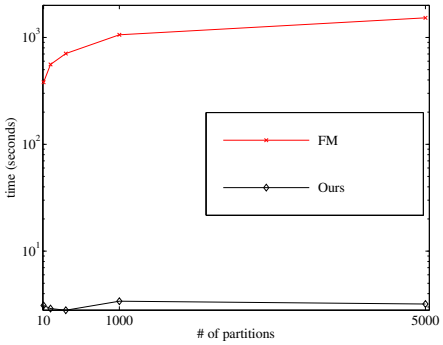
Network Data: Another real life data set we use is sensor data set with intrusion alerts. It contains information about local traffic on a sensor network which issued a set of intrusion attack types. Each graph constituted a local pattern of traffic in the sensor network. The nodes correspond to the IP-addresses, and the edges correspond to local patterns of traffic. We note that each intrusion typically caused a characteristic local pattern of traffic, in which there were some variations, but also considerable correlations. Each graph was associated with a particular intrusion-type. We have obtained

[3] http://www.informatik.uni-trier.de/ ley/db/

two such datasets. The data set **SENS1** contained a stream of intrusion graphs from June 1, 2007 to June 3, 2007, and the data set **SENS2** contains a stream of intrusion graphs from June 4, 2007 to June 6, 2007. These two data sets have very different characteristic from the DBLP data set. For instance, SENS1 has 68,386 nodes, 1,573,669 edges, and 2,250 graphs; SENS2 has 95,065 nodes, 1,535,036 edges, and 2,757 graphs. Thus, they are much more dense then the DBLP data, and furthermore, each graph is much bigger.



(a)Varying data size



(b) Varying number of partitions

Fig. 2. Running time over synthetic datasets

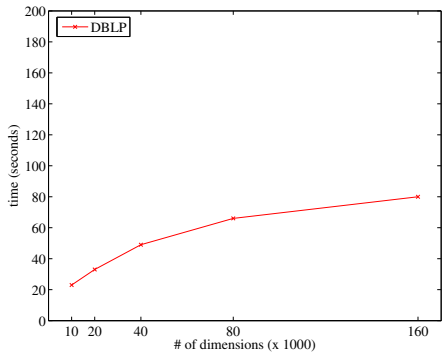
C. Running time

The running time of the algorithm consists of two parts: partitioning graphs and representing them using the new basis. Because the cost of finding the coordinate values for a graph using the new basis is O(1), we focus on the running time for disk-based graph partitioning. Figure 2 and 3 compare our partitioning algorithm with the FM algorithm that has been adapted for disk-based graphs.

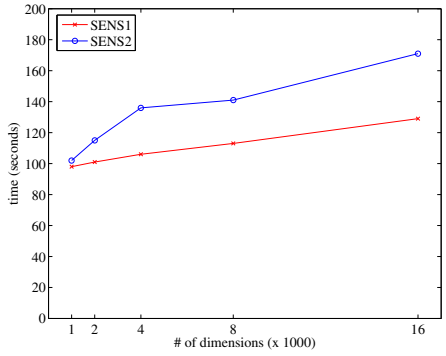
In Figure 2(a), we can see that our algorithm is 2 to 3 orders of magnitude more efficient than the disk-based FM algorithm. The synthetic dataset we use contains 20,000 graphs, and the total number of nodes increases from 50K to 1 million. As we increase the number of nodes in the synthetic dataset, we also increase the size of each graph so that the dataset maintains the same density.

While our approach takes about 3 seconds to partition a graph of 1 million nodes, the FM algorithm, when partitioning graphs of the same size, takes almost 1,000 seconds. Both the total number of nodes and the edges affect the cost of the FM algorithm. Since the partitioning algorithm is node-based, it implies that increasing the total number of nodes increases the size of the problem to the algorithm. On the other hand, for a dense graph, whenever a node is moved, the cost of updating the gains of its neighboring nodes also increase, as a node has more neighbors in a sensor graph. This cost is negligible for memory resident graphs, but for disk-based graphs it requires repeated random accesses to disk. This becomes a major source of running time. Still, the FM algorithm is much more efficient than the KL algorithm, which requires updating the gain for all the nodes in the graph instead of just neighboring nodes.

In Figure 2(b), we study the impact of number of partitions on running time. It shows that for the FM algorithm, the running time increases as the number of partitions increases. This is so because in order to divide a graph into many parts, FM makes deep recursions, which involves repeated accesses to disk-based graphs. On the other hand, our algorithm has no random access cost as it makes linear scan of disk-resident edges for partitioning, and it is also less affected by the number of partitions.



(a) DBLP



(b) SENS1, SENS2

Fig. 3. Running time over real datasets

We also record the running time of our algorithm on the real data sets. These data sets are however too large for the FM algorithm. Unlike the FM algorithm, which recursively partitions the data set, our algorithm partitions the data in a

global manner, which means it should be less affected by the number of partitions. This is indicated by the result shown in Figure 3 for both the DBLP and the sensor data sets.

D. Compression rate

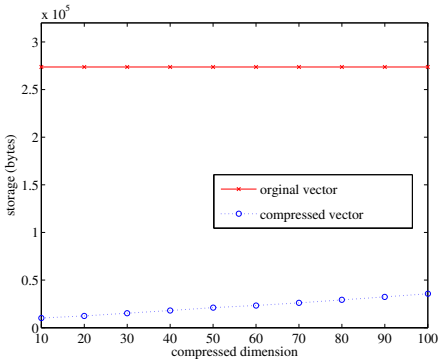
Second, we study the compression rate of our approach. For any graph, its basis representation is of size l, which is the number of structural concepts in the data. We can also represent a graph by a raw vector $\langle v_1, \cdots, v_N \rangle$, where N is the number of nodes, and $v_i = 1$ if the graph contains node i. Thus, the compression rate, l/N, is the ratio of the number of concepts to the total number of different nodes in all the graphs. The ratio can be extremely small for graphs of large nodes domain.

However, this ratio is misleading, as both representations, especially the raw vector, contain many zero entries, which means they can be further compressed. In our approach, we store only non-zero coordinate values, and each base representations is a list of (concept-identity, coordinate-value) pairs. The raw vector can be stored in the same form. We compute the compression-rate based on this format.

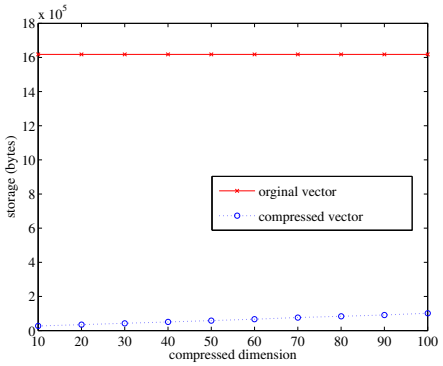
Figure 4 shows the compression rate for two synthetic datasets, the DBLP dataset, and the sensor dataset. The synthetic dataset in Figure 4(a) is relatively sparse as it contains 2000 graphs, while Figure 4(b) is more dense with 5000 graphs. The size of the nodes domain and the number of embedded concepts are the same for both datasets. It shows that the denser graph has a higher compression rate. This is because for dense graphs, the raw representation will have fewer non-zero entries, which increases the size of the storage. Since the embedded concepts keep the same, a good conceptual representation should not increase in size. Thus, compression rate goes up as graphs become denser. The same phenomenon happens for the real life datasets. As we know, the DBLP data set is very sparse (the number of nodes and the number of edges are roughly the same) while the sensor datasets are quite dense, thus the sensor data sets the compression rate is much higher. Overall, our approach achieves a compression factor around 3 for the DBLP data, and almost 100 for the sensor datasets.

E. Indexing Efficiency

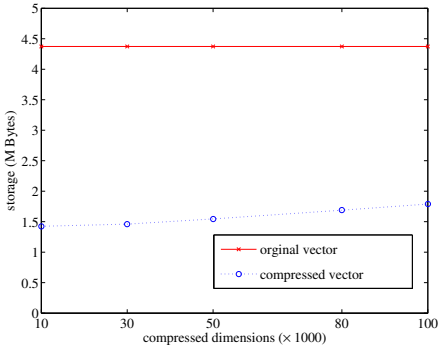
We maintain an inverted list for all the concepts. For each possible basis, we have a list of triples in the form of $\langle gid, v, S_{gid} \rangle$, where gid is the identifier of the graph whose conceptual coordinate value v for that basis is non-zero or larger than a user-provided threshold ϵ, and S_{gid} is the set of nodes that belong to the graph identified by gid. Storing S_{gid} in the inverted list might introduce redundancy, as a graph can represent multiple concepts. However, storing the node set in the inverted list enables us to answer a large variety of queries directly out of the inverted list without accessing the original graph data in a random fashion, and since each graph is typically small (e.g., the number of co-authors for a paper), it is worthwhile to include the limited redundancy.



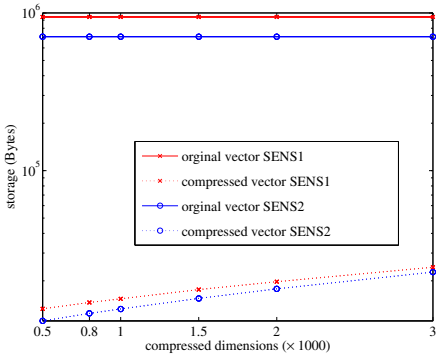
(a) Compressing sparse synthetic data



(b) Compressing dense synthetic data

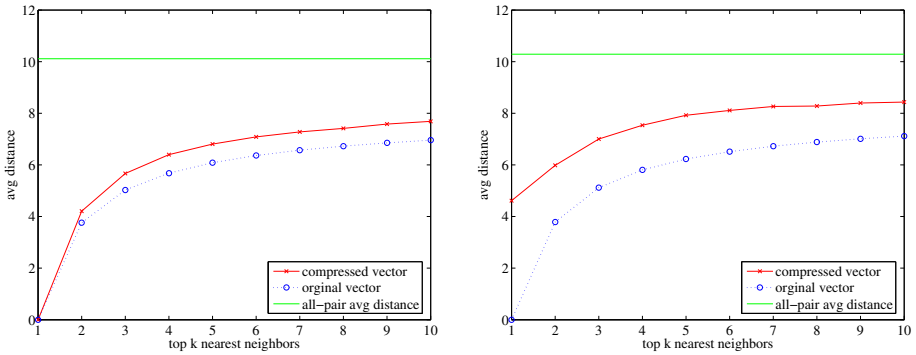


(c) Compressing DBLP



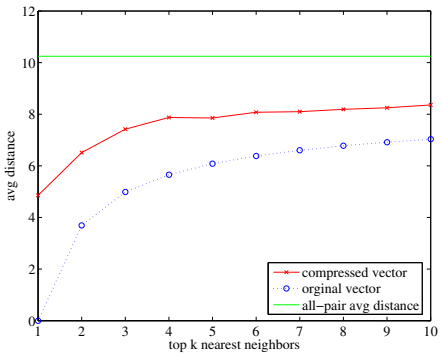
(d) Compressing SENS1 & SENS2

Fig. 4. Compression Rate

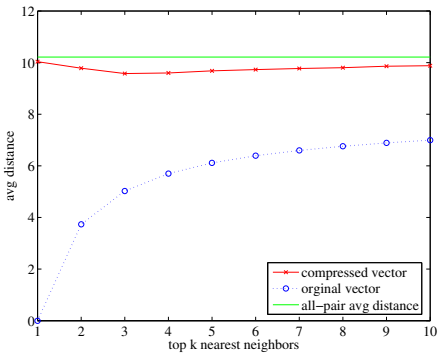


(a) 100 partitions

(b) 50 partitions



(c) 30 partitions



(d) 10 partitions

Fig. 6. Compression quality based on nearest neighbor search over synthetic data

We evaluate the indexing efficiency of the inverted list. Assume we are given a query which involves k concepts. We only need the inverted lists for that k concepts to answer the query directly. Analytically, the size of the data we need to access is:

$$k * (2 + \bar{G}) * \bar{C} * N_g/N_c \tag{7}$$

where $\bar{G}$ is the average nodes in each graph, $2+\bar{G}$ is the size of each inverted list entry (in terms of number of integers), $\bar{C}$ is the average number of concepts in each graph, N_g is the total number of graph, and N_c is the total number of concepts. On the other hand, if the index is not available, we need to scan the entire original data to answer the query. Thus, with the inverted list, the data we need to access in term of its size is a very small percentage of the original data. This is demonstrated in Figure 5 for both the DBLP data and the sensor data. Still, the result we obtained is larger than the analytical average given by Eq 7, mostly because concepts are of different sizes, with popular concepts containing a much larger number of nodes than other concepts.

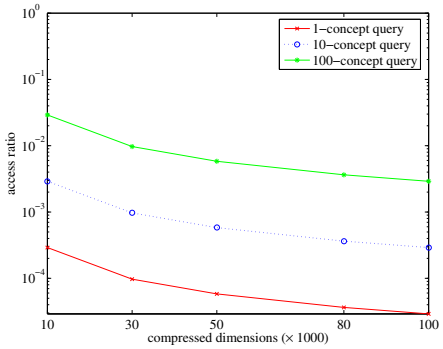
F. Quality of the Compression

Last, we evaluate the quality of the compression. The compression is apparently lossy. Thus we want to find out how accurately the conceptual representation approximates the original data.

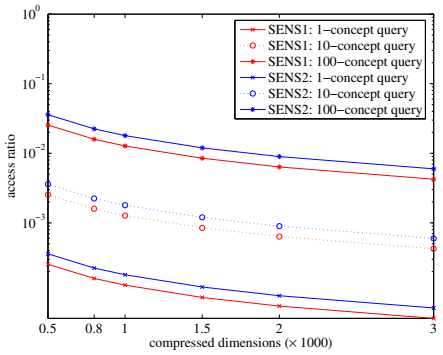
To do this, we use the k nearest neighbors of graphs as our criterion. For any given graph, we obtain its raw vector representation v, and its conceptual representation c. Then, we find v's top k nearest neighbors $N_k(v) = \{v_1, \cdots, v_k\}$ in the raw vector space, and we find c's top k nearest neighbors $N_k(c) = \{c_1, \cdots, c_k\}$ in the conceptual vector space. We compute the average distance between v and $N_k(v)$: $\frac{1}{k}\sum_i^k dist(v, v_i)$, as well as the average distance between c and $N_k(c)$: $\frac{1}{k}\sum_i^k dist(c, c_i)$. Note that the distance measure $dist(\cdot, \cdot)$ is for the raw vector space, and in our experiment, we used the Euclidean distance. Clearly, if the compression is very lossy, then there will be a big divergence between the two average distances. We also use the average distance between any two graphs in the data set as a baseline of comparison.

Figure 6 shows the result for a synthetic data set. The data set contains 2,000 graphs, each graph having about 20 edges, and the total number of different nodes in all the graphs is 2,000. Thus, the graph is relatively sparse. In addition, we embed 40 "concepts" in the graph. In the experiment, we used basis of dimensions 100, 50, 30, and 10. We randomly pick 100 graphs from the data set, find the top-k nearest neighbors ($k = 1 \cdots 10$) for each of them, and compute the average distance to the top-k nearest neighbors.

As we show in Figure 6(a), when the basis has 100 dimensions (in comparison with the raw data that has 2,000 dimensions), the average distances to top-k nearest neighbors show no big difference in the two vector spaces, which means the quality of the compressed representation is good even at such high compression rate. However, as we further increase the compression rate by reducing the number of dimensions,

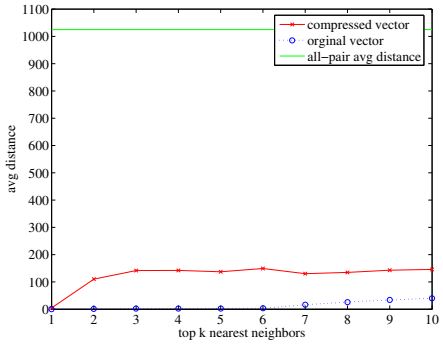


(a) Data access ratio for DBLP

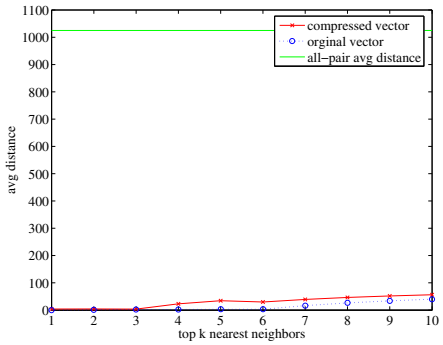


(b) Data access ratio for SENS1 & SENS2

Fig. 5. Indexing Efficiency

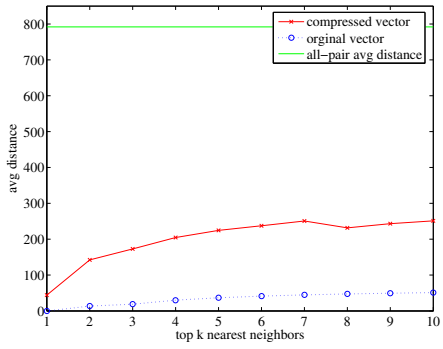


(a) dimension = 10,000

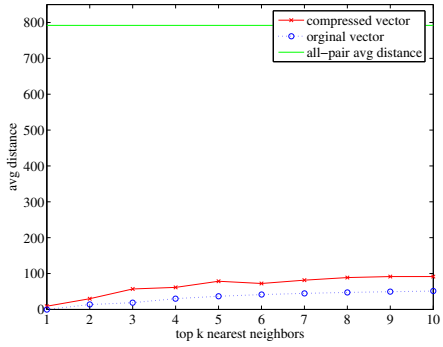


(b) dimension = 50,000

Fig. 7. Compression quality based on nearest neighbor search over SENS1

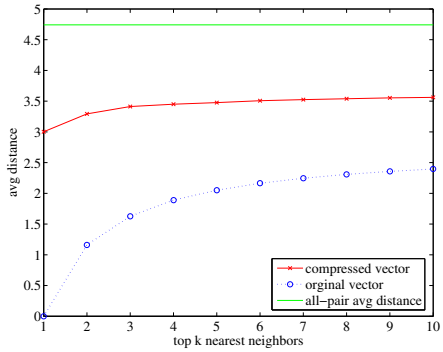


(a) dimension = 10,000

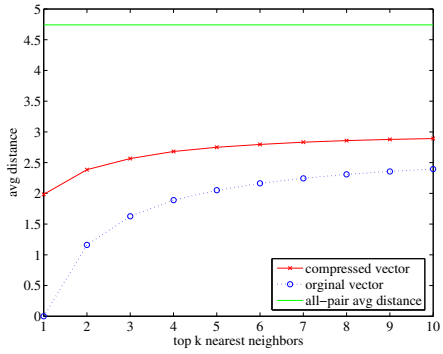


(b) dimension = 50,000

Fig. 8. Compression quality based on nearest neighbor search over SENS2



(a) dimension = 10,000



(b) dimension = 50,000

Fig. 9. Compression quality based on nearest neighbor search over DBLP

the average distance to the nearest neighbors found in the conceptual vector space start to show divergence from that in the raw vector space. In the extreme case, when the basis has only 10 dimensions (a reduction of 200 times in dimensionality) the divergence is big to the point that the average distance to the top-k nearest neighbor is not too much different from the all-pair average distance. This is expected, as we are aiming at very extremely high compression rate, which reduces the original 2,000 dimensions into only 10 dimensions.

We also evaluate the quality of compression using the real data sets. As we show in Figure 7 and Figure 8, for the sensor dataset, we achieve very good accuracy when the basis dimension is 50,000. The raw data (SENS1) only has 68,386 nodes, which seems that the compression rate is low (from 68,386 dimensions to 50,000 dimensions). In fact, because the graph is dense, the total number of non-zero coordinate-values in the original 68,386 dimensions is much higher than in the conceptual dimension, which means the real compression rate is very high. In fact, as we show in Figure 4(d), the original data is several orders of magnitude larger than the compressed data. This means we achieve high compression and high accuracy at the same time.

On the other hand, the DBLP dataset has very different characteristic from the sensor dataset, Figure 9 shows the result for the DBLP dataset. As we know, the DBLP data, which has 357,126 nodes and 975,951 edges, is very sparse. Furthermore, it contains 355,278 graphs (papers), which are almost as many as the total number of nodes, and each graph contains a very small number of nodes (co-authors). When the raw dimensionality is as high as in the DBLP data, the nearest neighbor search has the curse of the dimensionality problem, as any pair of points is as far apart as other pairs. Furthermore, because each graph is very small, it usually falls into a single concept. Thus, in nearest neighbor search, the distance between any two graphs will be 0 as long as they belong to the same concept and have the same size. Clearly, a large number of graphs will qualify. However, since their raw representations are different (because author list is different), their distances in the raw space is different. This means for a graph such as DBLP, the nearest neighbor search is not very meaningful, which may lead to large divergences between the two average distances, as we show in Figure 9.

IV. Conclusions and Summary

In this paper, we presented an algorithm for dimensionality reduction of massive disk-resident graphs with applications to indexing. The reduction is specifically designed to be friendly for applications such as indexing. To the best of our knowledge, this is the first approach for dimensionality reduction of disk-resident graphs. We design a contraction-based algorithm in order to reduce the size of the underlying graphs. We present theoretical results which bound the number of passes required on the underlying data. This reduced graph is utilized for efficient and effective indexing and retrieval. Furthermore, our reduced conceptual representation retains its interpretability with respect to the original graph. We present experimental results on real data sets illustrating the effectiveness and efficiency of our method.

References

[1] C. Aggarwal, N. Ta, J. Feng, J. Wang, and M. J. Zaki, "XProj: A Framework for Projected Structural Clustering of XML Documents," in *KDD Conference*, 2007.

[2] C. Aggarwal, Y. Xie, and P. Yu, "GConnect: A Connectivity Index in Massive Disk-Resident Graphs," in *PVLDB*, vol. 2(1), pp. 862–873, 2009.

[3] C. Aggarwal and H. Wang, *Managing and Mining Graph Data*, Springer, 2010.

[4] C. Aggarwal, *Social Network Data Analytics*, Springer, 2011.

[5] T. Dalmagas, T. Cheng, K.-J. Winkel, and T. Sellis, "Clustering XML Documents using Structural Summaries," in *Lecture Notes in Computer Science*, vol. 3268, pp. 547–556, 2005.

[6] C. Faloutsos, and K.-I. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," in *SIGMOD Conference*, pp. 163–174, 1995.

[7] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *25 years of DAC: Papers on Twenty-five years of electronic design automation*, pp. 241–247, 1988.

[8] G. Flake, R. Tarjan, and M. Tsioutsiouliklis, "Graph Clustering and Minimum Cut Trees," in *Internet Math.*, vol. 1(4), pp. 385–408, 2003.

[9] H. N. Gabow, "An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems," in *STOC*, pp. 448-456, 1983.

[10] K. V. R. Kanth, D. Agrawal, and A. K. Singh, "Dimensionality reduction for similarity searching in dynamic databases," in *SIGMOD Conference*, pp. 166-176, 1998.

[11] B. Hendrickson, and R. W. Leland, "A multi-level algorithm for partitioning graphs," in Supercomputing, 1995.

[12] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized Kernels between Labeled Graphs," in *ICML Conference*, pp. 321–328, 2003.

[13] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," in *Bell Sys. Tech. Journal*, vol. 49, pp. 291-307, 1970.

[14] T. Kudo, E. Maeda, and Y. Matsumoto, "An Application of Boosting to Graph Classification," in *NIPS Conference*, pp. 729–736, 2004.

[15] R. Kumar, P Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, "The Web as a Graph," in *PODS Conference*, pp. 1–10, 2000.

[16] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, "Incremental spectral clustering with application to monitoring of evolving blog communities," in *SDM Conference*, pp. 261–272, 2007.

[17] M. Rattigan, M. Maier, and D. Jensen, "Graph Clustering with Network Sructure Indices," in *ICML Conference*, pp. 783–790, 2007.

[18] C.-H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent Semantic Indexing: A Probabilistic Analysis," in *PODS Conference*, pp. 159-168, 1998.

[19] G. Salton, and M. J. McGill, *Introduction to Modern Information Retrieval*, Mc Graw Hill Inc., 1986.

[20] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos, "Less is more: Compact matrix decomposition for large sparse graphs," in *Statistical Analysis and Data Mining*, vol. 1, pp. 6–22, 2008.

[21] H. Tong, S. Papadimitriou, J. Sun, P. Yu, and C. Faloutsos, "Colibri: Fast Mining of Large Static and Dynamic Graphs," in *KDD Conference*, pp. 686-694, 2008.

[22] A. A. Tsay, W. S. Lovejoy, and D. R. Karger, "Random Sampling in Cut, Flow, and Network Design Problems," in *Mathematics of Operations Research*, vol. 24(2), pp. 383-413, 1999.

[23] X. Yan and J. Han, "CloseGraph: Mining Closed Frequent Graph Patterns," in *KDD Conference*, pp. 286–295, 2003.

[24] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining Significant Graph Patterns by Scalable Leap Search," in *SIGMOD Conference*, pp. 433–444, 2008.

[25] M. J. Zaki, and C. C. Aggarwal, "XRules: An Effective Structural Classifier for XML Data," in *KDD Conference*, pp. 316–325, 2003.