Kafrelsheikh Journal of Information Sciences ISSN (Online): 2535-1478, ISSN (Print): 2537-0677 Volume 3, Issue 2, 2022, PP. 1-10

Summarizing Graph Data Via the Compactness of Disjoint Paths

Mosab Hassaan
Faculty of Science, Benha University, Egypt
E-mail: mosab.hassaan@fsc.bu.edu.eg

Abstract: Graphs are widely used to model many real-world data in many application domains such as chemical compounds, protein structures, gene structures, metabolic pathways, communication networks, and images entities. Graph summarization is very important task which searching for a summary of the given graph. There are many benefits of the graph summarization task which are as follows. By graph summarization, we reduce the data volume and storage as much as possible, speedup the query processing algorithms, and apply the interactive analysis. In this paper, we propose a new graph summarization method based on the compactness of disjoint paths. Our algorithm called DJ_Paths. DJ_Paths is edge-grouping technique. The experimental results show that DJ_Path outperforms the state-of-the-art method, Slugger, with respect to compression ratio (It achieves up to 2x better compression), total response time (It outperforms Slugger by more than one order of magnitude), and memory usage (It is 8x less memory consumption).

Keywords: Graph Data, Graph Summarization, Disjoint Paths, Compression Ratio

1. INTRODUCTION

Graphs are non-linear data structures that represent many real-world data such as chemical compounds [17], protein structures, gene structures, metabolic pathways, communication networks, social networks [2], citation networks, transaction networks, and images entities. For instance, Figure 1 shows the graphs that have been used to represent many complex data types as follows. Figure 1(a), Figure 1(b), Figure 1(c), and Figure 1(d) show the representation of protein structure, metabolic pathway, chemical compound, and gene structure respectively.

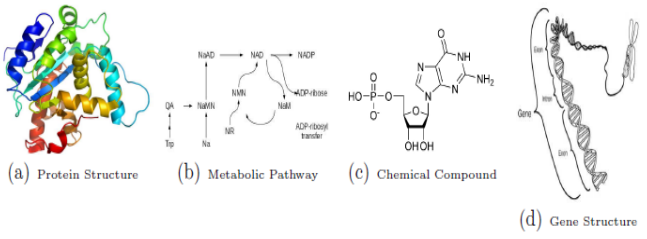


Figure 1. Representations of Graph Data

There are different meaning of the vertices and the edges of the given graph when this graph model different types of data as follows.

- In protein structure, the vertices represent the amino acids in the protein structure while the edges represent the strength of the non-covalent interactions between two amino acids.
- In Metabolic pathway, the vertices is a set of enzymes, reactions and chemicals while the edges represent the connections between them.
- In chemical compound, the vertices represent the atoms while the edges represent the bonds between the atoms.
- In gene structure, the vertices represent the gene products (mRNA transcripts) while the edges represent the pairwise relationships between genes products.

In this paper, we focus on the graph summarization problem. The objective of this problem is searching for a compact representation of the given graph which called the summary. This summary allows us to decrease the footprint of the graph and query [4, 11, 5, 3] in efficient way. In other words, Graph summarization has many benefits such as reduction of data volume and storage, speedup of graph algorithms and queries, and interactive analysis support.

There are two types of graph summarizations which are lossless graph summarization [9, 15, 6, 16] and lossy graph summarization [10, 8, 18]. In this paper, we focus on the lossless graph summarization which can reconstruct the original from summary. Here, we propose an efficient method to reduce the number of edges with lossless. This method based on the edge-merging technique which merges the edges in the given graph into virtual nodes. There are many challenges to use efficient method to summarize graphs in lossless way as follows.

1. Efficient summarized representation is essential since the memory is not free and there are tradeoffs between the fast compression and the efficiency of the space.
2. Real graphs are already highly compressible.

Our new method for summarizing graphs denoted as DJ_Paths which uses the **D**is**J**oint **Paths** to summarize the given graphs in efficient way. Here, we set the size of disjoint paths to two, i.e. each disjoint path contains only two edges. Then, we replace each disjoint path of size two by its corresponding virtual node. i.e., our summary contains a set of virtual nodes and virtual edges which represent the connections among virtual nodes. Details about disjoint paths are discussed in Definition 3.1 and Example 3.1. Our experiments on four real datasets show that DJ_Paths has the best performance compared to the state-of-the-art algorithm, Slugger [9], in terms of compression ratio, total response time and memory consumption.

Organization. This paper is organized as follows. Section 2 presents the preliminary concepts. Section 3 reports the related work. Section 4 discusses the proposed algorithm. Section 5 reports the experimental results. Finally, Section 6 concludes the paper.

2. PRELIMINARY CONCEPTS

As a general data structure, labeled graph is used to model complex structured. In labeled graph, vertices and edges represent entities and their relationships, respectively. Below, the terminology that used in this paper is discussed.

Definition 2.1. (Labeled Graph)
A labeled graph G is defined as a 4-tuple $<V_G, E_G, L_G, l_G>$, where V_G is the set of vertices (nodes), E_G is the set of edges, L_G is the labels set, and l_G is a function that maps each vertex or edge to a label in L_G.

The edges count ($|E_G|$) in the given graph G is called its size.

Definition 2.2. (Path)
A path from a vertex v_1 to a vertex v_k in a labeled graph G is a sequence of vertices in the following order: $< v_1, v_2, v_3, \ldots v_{k-1}, v_k >$ such that $\forall\ v_{i-1}$ and v_i we have $(v_{i-1}, v_i) \in E_G$.

Definition 2.3. (Vertex Neighborhood)
Given a graph G, the neighborhood of $u \in V_G$ is the set $N_G(u) = \{v \in V_G \mid (u, v) \in E_G\}$.

The degree of a vertex $v \in V_G$ is defined as $deg(v) = |N_G(v)|$.

Problem Definition: Given graph $H(V, E)$, the objective is to search for a summary graph $H^S(V^S, E^S)$ of H such that H^S conserves all characteristics of H and the size of H^S is sharply less than the size of H.

3. RELATED WORK

Recall, there are two types of graph summarizations which are lossless graph summarization [9, 15, 6, 16] and lossy graph summarization [10, 8, 18]. In lossless graph summarization, we enable to reconstruct the original graph while in lossy graph summarization, we do not enable to reconstruct the original graph. In this paper, we focus on lossless graph summarization. Many methods have been proposed for lossless graph summarization. These methods are divide into two categories. The first category is the grouping-based category (which contains edge-grouping methods and node-grouping methods) and the second category is the bit compression-based category. We discuss these categories in details as follows.

In node-grouping methods, some methods apply clustering techniques to catch a clusters that will be mapped to supernodes. Others recursively merge nodes into supernodes, which connected via superedges. This is done based on the function of application-dependent optimization. In Randomized [15] repeats (a) randomly catching a vertex v and (b) merging the vertex v with a vertex in the 2-hop

neighborhood of v such that the cost of encoding is minimized.

The authors of Slugger [9] proposed hierarchical graph summarization model, which is an expressive graph representation model that contains the previous one [15]. This model represents an unweighted graph via positive edges (P^+) and negative edges (P^-) among hierarchical supernodes (S), each of which can include others. In this algorithm, given the graph $G(V, E)$, the objective of this method is searching for hierarchical graph summarization model $M(S, P^+, P^-, H)$ with minimum cost. For more details about S, P^+, P^-, and H (set of hierarchy edges among supernodes), please see [9].

SWeG [16] improves Randmomized [15] by adding a dividing phase that divides the vertices into smaller groups prior to merging and proposing an approximation metric for catching the vertices to merge. Another method called SAGS [6] selects the vertices to be merged via the locality sensitive hashing.

In edge-grouping methods, edges are merged to virtual nodes to minimize the number of edges in a graph in lossless way. For instance, the authors of [14] present a dedensification method (edge-grouping-based method) that compresses the neighbourhood around high-degree vertices in lossless way. They proposed a query processing technique over the compressed graph without decompressing the graph.

In bit compression-based category, the objective is to minimize the count of bits needed to describe the given graph, where the summary graph includes the model of the given graph and its unmodeled pieces. For instance, SlashBurn [12] proposed for this category. At beginning, it searches for a nodes that connected with many edges which called hubs. Then, it removes these hubs to result in catching a large connected subgraphs which called spokes. At end, based on the hubs and spokes, it gains a good compression by resorting and transforming the representation of the adjacency matrix of the graph.

More details about these methods are reported in the survey [13].

4. PROPOSED ALGORITHM

In this section, we propose new algorithm called DJ_Paths for summarizing graphs using disjoint paths. At beginning, we define the disjoint paths (disjoint paths appeared in our previous paper [4]) and discuss how construct them from a given graph.

Definition 4.1. (Disjoint Paths)
Paths in a given graph H are disjoint if they are edge disjoint only.

Setting the size of disjoint paths is very important step. Therefore, we should be carefully when set this size. In this paper, we set the size of disjoint paths to two, i.e. each disjoint path contains only two edges. The reasons of selection this size are as the follows.

1. In any graph H, the number of compact disjoint paths with size two in H is known which is $|E_H| / 2$ if $|E_H|$ is even otherwise it is $(|E_H| - 1) / 2$ as we will see.

2. When we join two disjoint paths with size two that are connected by some nodes in the given graph, the result is one of five distinct substructures only. Note that, large size of these disjoint paths, the more of these distinct substructures. If the number of these distinct substructures is increased, this will lead to poor summarization.

For abbreviation, each following disjoint path has size two namely, p_k and we denote the set of all disjoint paths of size two in a given graph as P.

Now, we discuss how construct the disjoint paths from a given graph as follows. We propose an efficient method to construct the disjoint paths from a given graph. In this method, we iteratively remove one disjoint path from the given graph such that the remaining graph is connected. This iteration process terminates when the remaining graph is empty or the remaining graph has only one edge. Algorithm 1 outlines this method.

Theorem 4.1.
The proposed method for constructing the disjoint paths returns a compact set of disjoint paths with size two.

Algorithm 1: Constrution Disjoint Paths (*H*)

Input: Graph *H*.
Output: The set of disjoint paths, *P* and each $p_k \in P$ with size two.

1. **while** ($\exists\ p_k \in H$ such that $H \setminus p_k$ is connected)
   // p_k is a disjoint path with size two.
2. $P = P \cup p_k$
3. $H = H \setminus p_k$ // The remaining graph
4. **if** H != NULL // i.e. This case occurs when H has only one edge, *e*.
5. **Store** the edge *e*.
6. **return** *P*

The set of disjoint paths is compact when the size of the set of disjoint paths is maximized as possible. In other words, we divide the graph into a set of disjoint paths of size two with one edge (This occurs when the size of the given graph is odd) or without any edge (This occurs when the size of the given graph is even). See next example.

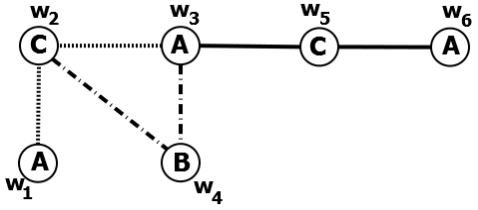
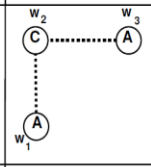
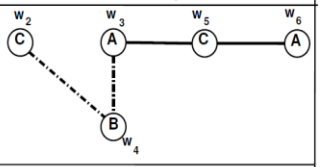
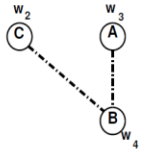
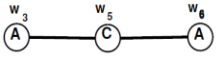
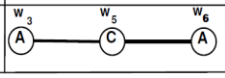


Figure. 2. Running Example (Graph *H*)

Example 4.1. Given the graph *H* in Figure 2. Based on our construction method, the disjoint paths of *H* and the remaining graphs are listed in Table 1. In this case, there are three iterations and the set of disjoint paths is compact since it contains three disjoint paths i.e. $|P| = 3$ (maximum value). In contrast, if we remove the disjoint path $<w_2,w_3,w_5>$ from *H* in the first iteration, then the remaining graph is disconnected as in Figure 3(a). In the second iteration, we can remove the disjoint path $<w_2,w_4,w_3>$ from *H*, then the remaining graph is also disconnected as in Figure 3(b). The second iteration is last one since the remaining graph contains two disconnected edges. In other words, it has not any disjoint path of size two. In this case, the set of disjoint paths is not compact since it contains two disjoint paths only i.e. $|P| = 2$. Here, in this inefficient case, we store two remaining edges. Note that, in the first case (the efficient one that based on Algorithm 1), we do not store any edges since the size of graph *H* is even (it has six edges).

Table 1. The Set of Compact Disjoint Paths of the Graph *H*

Iteration	Disjoint Path: p_j	$H = H / p_j$
j = 1		
j = 2		
j = 3		NULL

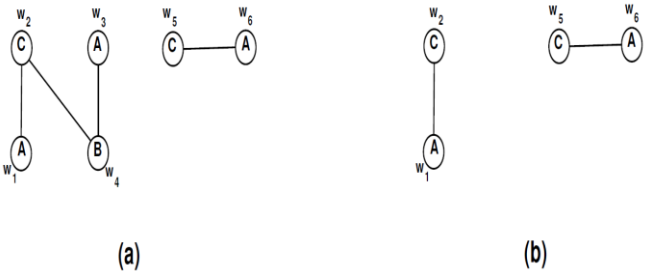


Figure. 3. Two Disconnected Remaining Graphs

In previous, we constructed the set of disjoint paths of size two (*P*) of a given graph *H* that based on Algorithm 1. Thereafter, we discuss how summarize the graph *H(V, E)* into its summary graph denoted as $H^S(V^S, E^S)$ with respect to *P* with lossless way as follows.

In the first phase (Construction the vertices of H^S, V^S), we replace each disjoint path $p_k \in P$ with its corresponding virtual vertex denoted as $v^S(p_k)$ in the summary graph, H^S. Note that, we should preserve the connections in the summary graph to enable us to conserve all characteristics of the original graph as we see in the second phase.

In the second phase (Construction the edges of H^S, E^S), if we have two disjoint paths p_i and $p_j \in P$ such that p_i and p_j are joint with some vertices in *H*. Then their corresponding virtual vertices $v^S(p_i)$ and $v^S(p_j) \in V^S$ will be connected with an edge $e^S \in E^S$.

Definition 4.2 outlines the previous two phases for summarizing the graph *H*.

Definition 4.2. (The Summary Graph H^S)
Given graph $H(V,E)$ with disjoint path set P. The summary graph of H is $H^S(V^S,E^S)$ where $V^S = \{ f(p_k) \; \forall \; p_k \in P \}$ where f is a function that maps each p_k to its corresponding virtual vertex $f(p_k) = v^S(p_k)$ (i.e. $|V^S| = |P|$) and $E^S = \{ e^S \; \forall \; e^S = (v^S(p_i), v^S(p_j))$ such that $f^{-1}(v^S(p_i))$ and $f^{-1}(v^S(p_j))$ are joinable $\}$.

Example 4.2 Given the graph H in Figure 1. Recall, there are three disjoint paths of H which are $p_1 = < w_1,w_2,w_3 >$, $p_2 = < w_2,w_4,w_3 >$, and $p_3 = < w_3,w_5,w_6 >$. Since we have three disjoint paths in H then there are also three virtual vertices in H^S (According to Definition 4.2). In other words, $V^S = \{ v^S(p_1) \; , v^S(p_2), v^S(p_3) \}$. Note that p_1, p_2, and p_3 are joinable to each other. Then, there are three edges in H^S. In other words, $E^S = \{e_1^S , e_2^S , e_3^S\}$ where $e_1^S = (v^S(p_1), v^S(p_2))$, $e_2^S = (v^S(p_1), v^S(p_3))$, and $e_3^S = (v^S(p_2) \; , v^S(p_3))$. The summary graph of H (H^S) is shown in Figure 4.

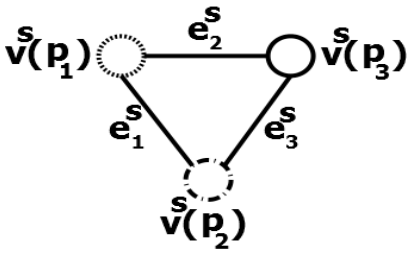


Figure 4. The Summary Graph, H^S

Note that, the join between any two disjoint paths in H will produce a subgraph g with size equals to 4. The subgraph g has five possible distinct structures namely, S_1, S_2, S_3, S_4, and S_5. For simplicity, we draw these structures without labels, see Figure 5.

To conserve all characteristics of the original graph, we should do the following. The edge $e^S = (v^S(p_i), v^S(p_j)) \in E^S$ will represent the structure that will be produced when joining p_i and p_j. Therefore, we associate with each e^S a number k that represents the resulted structure S_k where $1 \leq k \leq 5$. At the same time, the edge e^S must determine which vertices in V (at most two vertices) contributing the join between p_i and p_j. For more details, see the following.

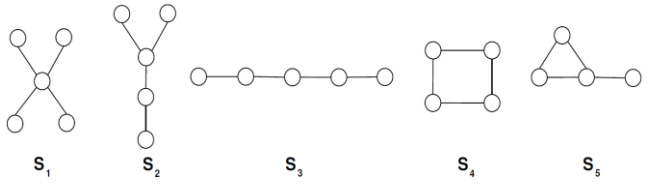


Figure 5. The Five Distinct Structures with Size Four

For each disjoint path $p_t \in P$, we label its three vertices as left vertex, middle vertex, and right vertex namely, w^l_t, w^m_t, and w^r_t respectively. Next, we discuss all possible joins between any two disjoint paths $p_i = < w^l_i, w^m_i, w^r_i >$ and $p_j = < w^l_j, w^m_j, w^r_j >$ to result S_k where $1 \leq k \leq 5$. Recall, the count of vertices that contributing the join is at most two as we will see.

For the first three structures S_1, S_2, and S_3, only one vertex contributing the join between p_i and p_j . Next, we determine this vertex for each structure.

- In S_1, the vertex $w^m_i = w^m_j$ contributing the join between p_i and p_j. Then there is only one possible combination.

- In S_2, the vertex $w^m_i = w^l_j$ or $w^m_i = w^r_j$ or $w^l_i = w^m_j$ or $w^r_i = w^m_j$ contributing the join between p_i and p_j . Then there are four possible combinations.

- In S_3, the vertex $w^l_i = w^l_j$ or $w^l_i = w^r_j$ or $w^r_i = w^l_j$ or $w^r_i = w^r_j$ contributing the join between p_i and p_j . Then there are four possible combinations.

While for the last two structures S_4 and S_5, two vertices contributing the join between p_i and p_j. Next, we determine these vertices for each structure.

- In S_4, the two vertices $w^l_i = w^l_j$ and $w^r_i = w^r_j$ (or $w^l_i = w^r_j$ and $w^r_i = w^l_j$) contributing the join between p_i and p_j. Then there are two possible combinations.

- In S_5, the two vertices $w^r_i = w^r_j$ and $w^l_i = w^m_j$ (or $w^l_i = w^r_j$ and $w^r_i = w^m_j$ or $w^r_i = w^m_j$ and $w^l_i = w^l_j$ or $w^l_i = w^m_j$ and $w^r_i = w^l_j$ or $w^r_i = w^l_j$ and $w^m_i = w^r_j$ or $w^r_i = w^r_j$ and $w^m_i = w^l_j$ or $w^m_i = w^l_j$ and $w^l_i = w^r_j$ or $w^m_i = w^r_j$ and $w^l_i = w^l_j$) contributing the join between p_i and p_j. Then there are eight possible combinations.

Table 2 contains a summary for each structure S_k with respect to the count of the vertices that contributing the join to result S_k and the count of all possible combinations.

Table. 2. A summary for Structure S_i

Structure	# Vertices Contributing the Join	# Possible Combinations
S_1	1	1
S_2	1	4
S_3	1	4
S_4	2	2
S_5	2	8

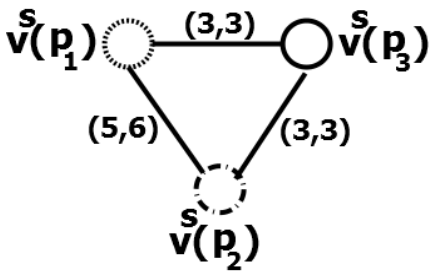


Figure 6. The Final Summary Graph, H^S

At this moment, for the summary graph $H^S = (V^S,E^S)$, we can label each edge $e^S = (v^S(p_i), v^S(p_j)) \in E^S$ with the pair *(k, l)* where *k* is the index of the produced structure and *l* is the index of the used combination. For instance, if we label e^S with (5, 3) then this means that the edge e^S with the two virtual vertices $v^S(p_i) \in V^S$ and $v^S(p_j) \in V^S$ constructed the structure S_5 that produced using the third possible combination. By this, the summary graph H^S conserves all characteristics of the original graph *H*. In other word, we can decode H^S to get *H* with lossless way. Figure 6 shows the final form of the summary graph that reported in Figure 4. From Definition 4.2, we have $|V^S| = |P|$. Since each path $p \in P$ contains two edges, then $|P| = |E|/2$. In other words, the count of virtual vertices in H^C equals to the half of the count edges in *H*, formally, $|V^C| = |E| / 2$. While the count of edges in H^C, $|E^C|$ is based on the joins among paths in *P*. Algorithm 2 outlines our method, DJ_Paths, for summarizing a given graph *H*.

In the experimental evaluation section, we will show the better summarization of DJ_Paths against the state-of-art algorithm (Slugger) on many real datasets. To achieve this, we used the compression ratio to measure how well the graph is summarized. Here, the larger compression ratio is the better summarization we have. The compression ratio equations of the two methods are as follows.

- In our proposed method, DJ_Paths, the compression ratio calculated by dividing $|V| + |E|$ (before summarization) with $|V^S| + |E^S|$ (after summarization). For instance, from Figure 2, the count of vertices and edges in *H* are 6 and 6 respectively (i.e. $|V| + |E| = 12$). While, from Figure 6, the count of vertices and edges in H^S are 3 and 3 respectively (i.e. $|V^S| + |E^S| = 6$). Then the compression ratio is $(|V| + |E|) / (|V^S| + |E^S|) = 12 / 6 = 2$.
- In Slugger method, the compression ratio calculated by $(|V| + |E|) / (|V| + |P^+| + |P^-| + |H|) = (6+6) / (6 + 6 + 0 + 0) = 12 / 12 = 1$.

Here, our method DJ_Paths achieves the best summarization which has the higher compression ratio.

Algorithm 2: DJ_Paths (*H*)

Input: Graph $H = (V, E)$.
Output: The summarized graph $H^S = (V^S,E^S)$.
1. P = Construction Disjoint Paths(*H*) // Algorithm 1
2. **Construct** $|P|$ virtual vertices for H^S (i.e. $|V^S| = |P|$) such that $\forall\ p_m \in P$ we have $v^S(p_m) \in V^S$ and $v^S(p_m) = f(p_m)$
3. **for** $i = 1$ to $|V^S|$
4. **for** $j = i + 1$ to $|V^S|$
5. **if** $f^{-1}(v^S(p_i)) = p_i$ and $f^{-1}(v^S(p_j)) = p_j$ are joinable
6. **Construct** the edge $e^S = (v^S(p_j)\ ,\ v^S(p_j))$ where $e^S \in E^S$
7. **Label** e^S with the pair *(k, l)*, where *k* is theindex of the resulted structure when joining p_i and p_j and *l* is the index of the used combination.
8. **return** H^S

5. EXPERIMENTAL EVALUATION

This section shows the results of experiments on four real datasets. We compare the performance of DJ_Paths against Slugger algorithm [9]. The code of Slugger was downloaded from https://github.com/KyuhanLee/slugger. We used Slugger for comparison since it outperforms the four algorithms in [15, 16, 1, 7].

DJ_Paths is implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments were run on laptop with Intel i3 2.4 GHz and 8G memory running Linux. In next section, we demonstrates the datasets.

5.1. Datasets

Experimental evaluation are performed on a set of real datasets as follows. We used four real datasets as follows.

-AIDS: https://wiki.nci.nih.gov/display/NCIDTPdata
-Chemical: https://pubchem.ncbi.nlm.nih.gov/
-Protein: https://fki.tic.heia-fr.ch/databases/iam-graph-database
-Enzymes: https://networkrepository.com/networks.php

For the first three datasets, we select from each dataset five graphs, namely G_1, G_2, G_3, G_4, and G_5 where the average number of vertices in these datasets is 153, 124, and 80 respectively and the average number of edges in these datasets is 155.2, 126.4, and 100 respectively. For the fourth dataset, we select three graphs, namely G_1, G_2, and G_3 where the average number of vertices in this dataset is 103.33 and the average number of edges in this dataset is 128.66.

5.2. Performance of DJ_Paths Against Slugger

The proposed method, DJ_Paths is evaluated according to the following criteria:

1. ***Compression Ratio:*** To measure how well the graph is summarized using DJ_Paths.
2. ***Total Response Time:*** To measure the efficiency of DJ_Paths.
3. ***Memory Usage:*** To show the memory consumption of DJ_Paths.

5.2.1. Compression Ratio

Figure 7 shows the compression ratio of the two methods (DJ_Paths and Slugger) on the four real datasets. Recall, the larger compression ratio is the better summarization we have. DJ_Paths shows a better compression ratio in all datasets. It achieves up to 2× better compression than Slugger.

5.2.2. Total Response Time (MSec)

Figure 8 shows the total response time (MSec) of the two algorithms (DJ_Paths and Slugger) on the four real datasets. DJ_Paths has the best execution time on the most datasets. It outperforms Slugger by more than one order of magnitude. For example, with G_1 in Enzymes dataset**,** DJ_Paths takes only 41 MSec while Slugger takes 476 MSec. Except G_2 and G_3 in AIDS dataset, Slugger outperforms DJ_Paths by approximately two factors. For example, with G_3 in AIDS dataset**,** DJ_Paths takes 651 MSec while Slugger takes 383 MSec. Also, at G_3 and G_5 in Chemical dataset, Slugger outperforms DJ_Paths by approximately 1.5 factors.

5.2.3. Memory Usage (MB)

Figure 9 shows the memory consumption in MB of the two algorithms (DJ_Paths and Slugger) on the four datasets. This figure plots the peak of the memory consumption during execution (the memusage command in Linux was used for measure). At all four real datasets, we can note that Slugger generally has more than 8× higher memory consumption than DJ_Paths. In other words, DJ_Paths has the best memory consumption on all datasets. For example, with G_1 in Enzymes dataset**,** the memory consumption of DJ_Paths is 2.14 MB while the memory consumption of Slugger is 17.22 MB.

6. CONCLUSIONS

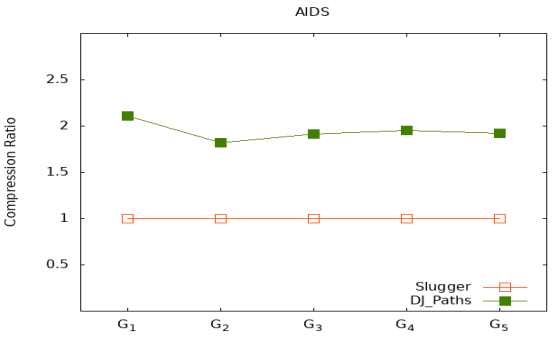
In this paper, we focus on the graph summarization task. Here, we propose a new algorithm for graph summarization called DJ_Paths. DJ_Paths is edge grouping-based method. It replaces the set of disjoint paths with size two in the given graph by a set of virtual nodes. In other words, our summary graph contains a set of virtual nodes and a set of virtual edges that represent the connections among virtual nodes. Note that, this summary conserves all characteristics of the original graph. Experimental results show that DJ_Paths has the best performance compared the state-of-the-art method (Slugger) in terms of compression ratio, total response time, and memory usage. As future work, we plan to adapt DJ_Paths method for subgraph search problem and similarity search problem.

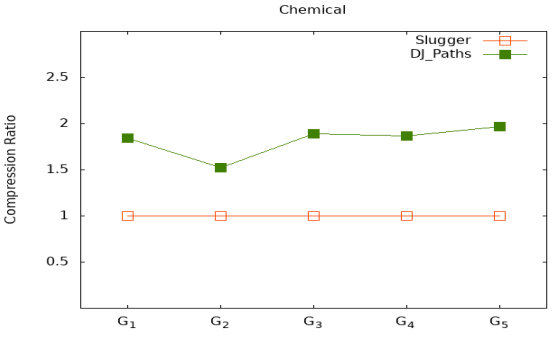
Acknowledgments

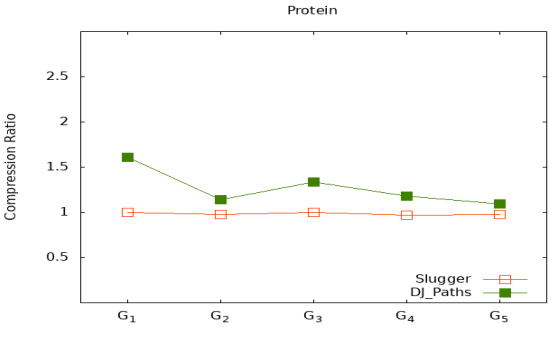
I wish to express my deep gratitude to my mentor Prof Dr. Karam Gouda. I am very grateful to my parents, my wife, my brother, and my sisters for their continuous moral support and encouragement.

Conflicts of Interest

The author declares that I don't have any conflict of interest regarding this article.







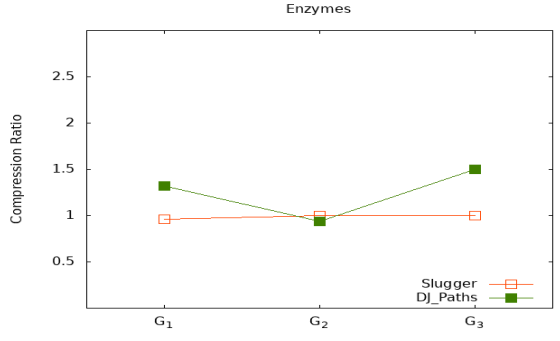
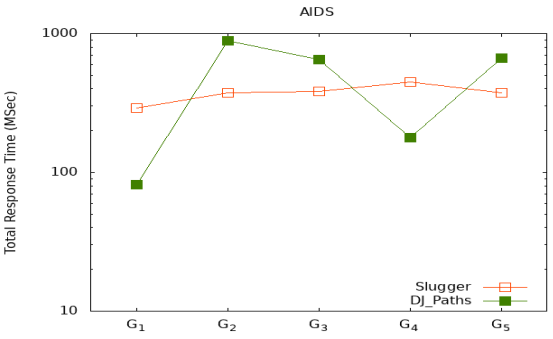
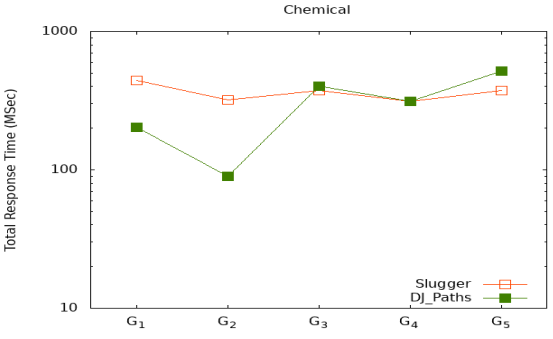
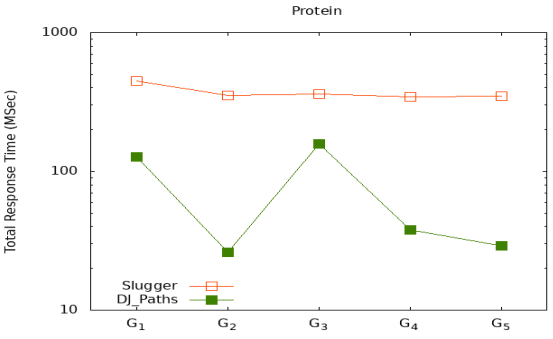


Figure 7. Compression Ratio







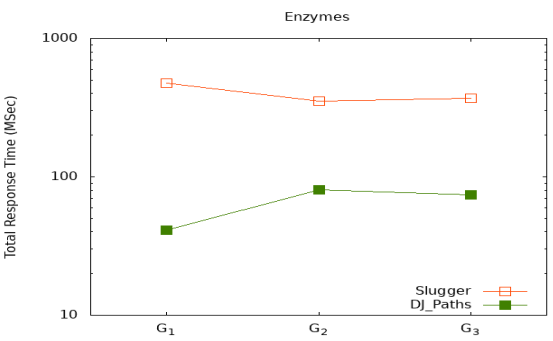


Figure 8. Total Response Time (MSec)

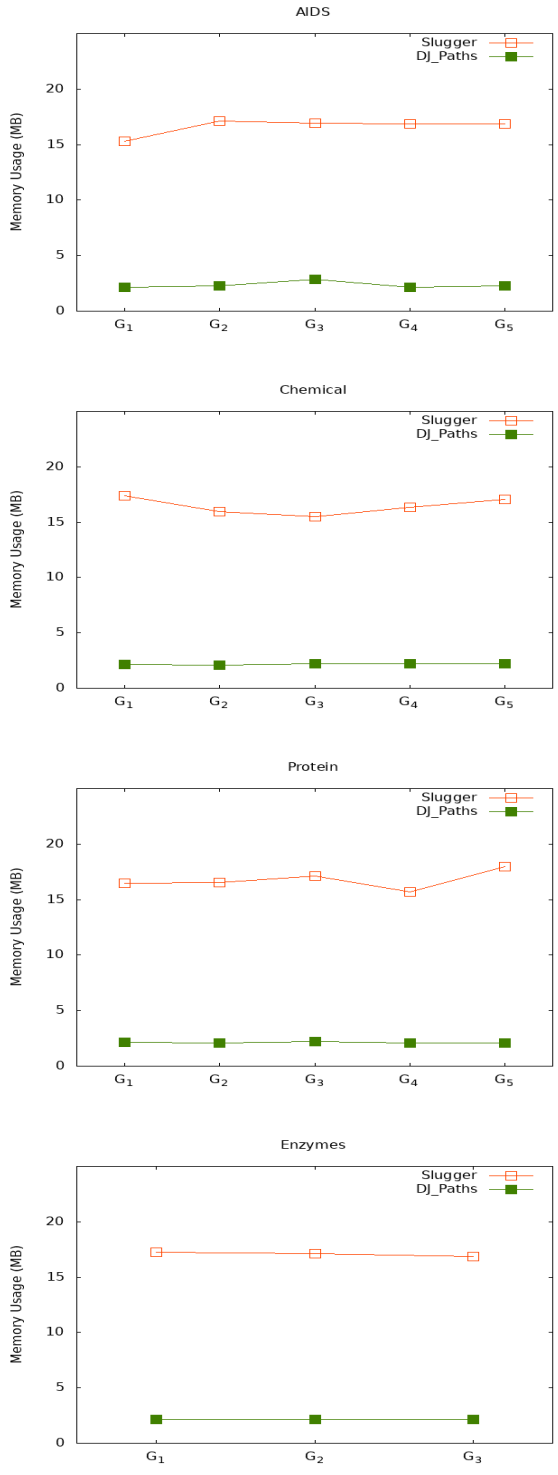


Figure 9. Memory Usage (MB)

REFERENCES

[1] Beg, M., Ahmad, M., Zaman, A. & Khan, I. (2018). Scalable approximation algorithm for graph summarization. *Proc. of PAKDD*. https://doi.org/10.1007/978-3-319-93040-4_40

[2] Cai, D., Shao, Z., He, X., Yan, X., & Han, J. (2005) Community mining from multi-relational networks. Proc. of PKDD.

[3] Chang, L., Feng, X., Yao K., Qin, L. & Zhang, W. Accelerating Graph Similarity Search via Efficient GED Computation (2022). *IEEE Trans. Knowl. Data Eng.* . https://doi.org/10.1109/TKDE.2022.3153523.

[4] Gouda, K. & Hassaan, M. (2013). Compressed feature-based filtering and verification approach for subgraph search. Proc. of EDBT. https://doi.org/10.1145/2452376.2452411

[5] Gouda, K. & Hassaan, M. (2019). A novel edge-centric approach for graph edit similarity computation. *Information Systems 80*, 91–106. https://doi.org/10.1016/j.is.2018.10.003

[6] Khan, K. U., Nawaz, W. & Lee, Y., -K. (2015). Set-based approximate approach for lossless graph summarization. *Computing, 97*(12), 1185–1207. https://doi.org/10.1007/s00607-015-0454-9

[7] Ko, J., Kook, Y. & Shin, K. (2020). Incremental lossless graph summarization. *Proc. of KDD*. https://doi.org/10.1145/3394486.3403074

[8] Lee K., Jo H., Ko J., Lim S. & Shin K. (2020). Ssumm: Sparse summarization of massive graphs. *Proc. of KDD*. https://doi.org/10.1145/3394486.3403057

[9] Lee, K., Ko, J. & Shin, K. (2022). SLUGGER: Lossless Hierarchical Summarization of Massive Graphs. *Proc. of ICDE*.

[10] LeFevre, K. & Terzi, E. (2010). Grass: Graph structure summarization. *Proc. of SDM*.

[11] Licheri, N., Bonnici, V., Beccuti, M. & Giugno R. (2021). GRAPES-DD: exploiting decision diagrams for index-driven search in biological graph databases. BMC Bioinformatics 22, 209. https://doi.org/10.1186/s12859-021-04129-0

[12] Lim, Y., Kang, U. & Faloutsos, C. (2014). SlashBurn: Graph compression and mining beyond caveman communities. *IEEE Trans. Knowl. Data Eng. 26*(12), 3077–3089. https://doi.org/10.1109/TKDE.2014.2320716

[13] Liu, Y., Safavi, T., Dighe, A. & Koutra, D. (2018). Graph summarization methods and applications: A survey. *CSUR, 51*(3), 1–34. https://doi.org/10.1145/3186727

[14] Maccioni, A. & Abadi, D., J. (2016). Scalable pattern matching over compressed graphs via dedensification. *Proc. of KDD*. https://doi.org/10.1145/2939672.2939856
[15] Navlakha, S., Rastogi, R. & Shrivastava, N. (2008). Graph summarization with bounded error. *Proc. of SIGMOD*.
[16] Shin, K., Ghoting, A., Kim, M. & Raghavan, H. (2019). Sweg: Lossless and lossy summarization of web-scale graphs. *Proc. of WWW*. https://doi.org/10.1145/3308558.331340
[17] Willett, P. (1998). Chemical similarity searching. J. Chem. Inf. Computer Science, 38(6). https://doi.org/10.1021/ci9800211
[18] Zhou, H., Liu, S., Lee, K., Shin, K., Shen, H. & Cheng X. (2021). Dpgs: Degree-preserving graph summarization. *Proc. of SDM*. https://doi.org/10.1109/HPEC49654.2021.9622846