Graph Summarization with Bounded Error

Kamalendu Nayek

University of Paderborn

Abstract. This paper is presented to propose a solution for large graphs and present them as a highly compact two-part representation. Large graphs such as the World Wide Web, Social Networks, Basket Data, etc which are huge and so complicated that they are difficult to fit into a single screen and take huge space in the main memory. Thus, a summary which is an aggregate graph representation is created for such huge graphs. Each node in the summary is a collection of nodes from the original graph G, and each edge represents a collection of edges from G. The correction C specifies the list of edge-corrections to be applied to the summary for recreating G. The benefit of the solution is that it combines with the MDL principle and presents highly understandable coarse-level summaries of the input graph G. The algorithms developed in this paper are to construct highly compressed graph representations with small sizes and maximum accuracy.

Keywords: Algorithms · Graph Compression · Minimum Description Length · Approximation.

1 Introduction

1.1 Motivation

In the real world, there are a huge number of large-scale systems and applications that store a massive amount of data that interact with each other. They can be called interaction between the entities and this data can be well-presented graphs. Below is the list of some application domains.

1. **World Wide Web.** The WWW can be represented as a graph structure in which each page can be considered as a node and the hyperlink connection leading to another web page can be considered as an edge. This linking structure is been used by Google [4] to improve search quality. Recent studies from the search engine estimate the size of the WWW with 3 billion nodes and more than 50 billion arcs [3].
2. **Social Networking.** Social networking sites like Facebook, LinkedIn, Instagram etc have millions of users and each user can be represented as a node and the friend lists can be represented as the arcs. Analysis of the social network can provide with valuable information like the social relationship of users, the music they like and the web pages with common interests.

3. **Market Basket Data.** Market basket data contain information about the products that are purchased by the customers on a regular basis and can help in establishing a buying pattern of the customers. These patterns can help in data analytics and predict future buying patterns of customers and thus help a retail market to predictive analysis.

In all the above examples it is very challenging to visualize such huge graphs due to difficulty in fitting them into a single screen or developing graph mining algorithms to scale such graphs. It is also an overloading task to the main memory of a computer.

The scientific contribution of this paper is discussed below which solves the above discussed issues:

1. The summary S is a graph which is a compact version of the original graph and can be fit into the memory easily. It also provides clarity to the high-level graph structure and techniques for analysis of customer data, social network communities. Other clustering algorithms [17,8] are based on the similarity and distance but here the focus is on computed information and theoretic principles.
2. The representation also allows high degree compression for general graphs along with an ϵ parameter that can be tuned for lossy compression with bounded errors. The ϵ parameter helps in getting more accuracy for good compression. This helps in saving up some space in the main memory and analyse the graphs efficiently using algorithms.

1.2 Background and Fundamentals

1.2.1 Representation of a Generic Graph Suppose there is a web graph G (Let us take an example of friends on a social network) where the friends are nodes and denoted by V_G and the connecting edges (friendship amongst them) are denoted by E_G (G = (V_G, E_G)). This graph will be very huge to be presented in a single screen as it will have billions of nodes that will be connected with billions of edges. Most of these nodes will be connected to each other in one way or the other due to the link between web pages through hyperlinks. These types of graphs also require a large amount of main memory to be stored.

Rather than representing graph G as a whole, we can represent it as R = (S, C). This representation consists of a graph summary S = (V_s, E_s) with some edge corrections C. This will be a compressed version of the graph G with some exceptions.

Here, summary S will be the aggregated graph in which each node v belongs to V_s which is called the supernode. These supernodes are part of set A_v of nodes in G. The edges (u, v) belongs to E_s is called the superedge. The superedges represents the edges between A_u and A_v.

Correction C will be the set of edge corrections denoted by either '$+ve$' or '$-ve$' corrections. We can use this correction to recreate the original graph by expanding the summary S. For the supernodes v belongs to V_s each node from

the set A_v can be created and for each superedge *(u, v)* belongs to E_s all the edges between the nodes should be added. But there will be few edges those will either be left or are added extra. These types of errors with the edges in the summary can be fixed with the help of correction C.

[FIGURE]

The corrections can be applied to reconstruct the original graph G.

1.2.2 Graph Compression using MDL Representation Compression of graphs generally works with the MDL [28] principle. It uses the principle to (a) reduce the size of the graph with the minimum usage of data without much loss of the meaning of the message. It focuses on (b) reduction with the use of some theory. From the above example the original graph G is on the left side. The cost of representation is given by R = (S, C). On the right side is the summary S of the original graph. The representation cost (R) = $|E_s| + |C|$. In this the $|E_s|$ represents (a) and $|C|$ represents (b). The MDL representation for minimum cost is given by $\widehat{R} = (\widehat{S}, \widehat{C})$. Every supernode v is the set of nodes of A_v. And every superedge *(u, v)* between supernode u and v is $\prod_{uv} = A_u + A_v$. Cost without the superedge is $|A_{uv}|$. The decision to choose amongst the edges is $min\{1 + |\prod_{uv} - A_{uv}|, |A_{uv}|\}$.

1.2.3 Approximate Representations To recreate the original graph is not always necessary. Certain approximations can be made and are sufficient to recreate the original graph. This graph may not be the exact graph but will be close enough to the original. Certain approximations can be used for further cost reduction. ϵ representation R_ϵ can be used for this purpose. The bounded error $\epsilon(0 <= \epsilon <= 1)$ is taken as the approximation. For a given node v the neighbours are Nv in G. The approximate neighbours N_v are computed with ϵ error. This bounded error is $|N'_v - N_v| + |N_v - N'_v| < \epsilon\,|N_v|$ (Equation 1). The number of neighbours added or deleted s at most ϵ fraction of the true neighbours. Let us go back to the above example G in Figure 1 and S in Figure 2 the corrections are - *(n,a)* and + *(n,v)*. Since there are less corrections the ϵ will be small. Let this value be 1/3. If we try to remove the entry + *(n,v)* the approximate neighbours of n : $N'_n = p, k, y$ and actual neighbours $N_n = p, k, y, v$and the approximate neighbours of n: $N'_v = y, a$ and actual neighbours $N_v = y, a, n$. Bounded error: $error(v) = |N'_n - N_n| + |N_n - N'_n| < \epsilon\,|N_n| = |0| + |4 - 3| < 1/3\,|4| = 1 < 4/3$. Similarly, for N'_v and N_v. Hence, we can remove + *(n,v)* from the correction C and reduce the size without violating the equation above.

1.2.4 Contributions The greatest highlights of this paper are as follows:

1. The graph representation is very small and accurate and can be used for both lossless and lossy graphs. There will be minimum bounded errors. The algorithms developed are used with the MDL principle to get accurate graph summaries.

2. Two algorithms GREEDY and RANDOMIZED are developed to calculate the MDL representation with less size.
3. Two more schemes APXMDL And APXGREEDY are developed to compute the minimum cost representation. APXMDL is used with a matching algorithm to remove highest possible edge corrections from MDL representations while fulfilling some constraints. On the other hand, APXGREEDY is used with the ϵ error with each step performed by GREEDY.

2 Related Work

The graph compression problem has been worked on by different groups of researchers:

1. **Web Graph Compression** [1] - To Optimize overhead for link graphs with URLs which are not applicable for other type of graphs. They mostly focus on bit compression of given graph data
2. **Approximate Query Processing** [6] - Provide approximate answers to relational queries. Histograms and wavelets do not produce high-level graph summaries that can be visualized to find interesting patterns.
3. **Network Visualization** [14] - Work on extracted data from Network traffic which is an extended work on this paper. The main focus is on performing this role-classification, and not to achieve compression.

3 Computing MDL Representation

There are two algorithms those have been presented for calculating MDL representations $\widehat{R}$.

1. GREEDY – In this algorithm the nodes are merged into supernodes that can give the maximum cost reduction.
2. RANDOMIZED – This is a lightweight algorithm in which nodes are picked randomly from the neighbourhood and merged together.

3.1 Greedy Algorithm

[1] /* Initialization phase */ $V_S = V_G$; $H = \emptyset$; all pairs $(u, v) \in V_S$ that are 2 hops apart s(u, v) ¿ 0 insert (u, v, s(u, v)) into H; /* Iterative merging phase */ $H \neq \emptyset$ Choose pair $(u, v) \in H$ with the largest $s(u, v)$ value; $w = u \cup v$; /* merge supernodes u and v */ $V_S = V_S\ u, v\ u\ w$; all $x \in$ VS that are within 2 hops of u or v Delete (u, x) and (v, x) from H; w$s(w, x) > 0$ insert $(w, x, s(w, x))$ into H all pairs (x, y), such that x or y is in N_w Delete (x, y) from H; $s(x, y)$ ¿ 0 insert $(x, y, s(x, y))$ into H /* Output phase */ E_S = C = $\emptyset$; all pairs (u, v) such that u, v $\in V_S$ —A_{uv} — ¿ (—π_{uv} — + 1)/2 Add (u, v) to E_S ; Add $-(a, b)$ to C for all (a, b) $\in \Pi_{uv}$ - A_{uv} ; Add $+(a, b)$ to C for all $(a, b) \in A_{uv}$; return representation $R = (S = (V_S, E_S), C)$; Greedy Algorithm

The GREEDY algorithm merges the nodes into a supernode with a maximum cost reduction. If any two nodes those share a common neighbour, then they will be merged. A point to be kept in mind is that more the no of common neighbours more will be the cost reduction. The cost reduction is defined as *s(u, v)* where u and v are the pair to be merged.

With GREEDY the set of supernodes are maintained as V_s to form the summary S of the graph. Let one set of nodes which combines to form a supernode v belongs to V_s, and the neighbour set is N_v which consists of a set of other supernodes u belongs to V_s. There will be an edge from node a to node b for both present in the set A_v and A_u respectively. The cost of the superedge from supernode v to its neighbour x belongs to N_v is denoted by $C_{vx} = min\{|\prod_{vx}| - |A_{vx}| + 1, |A_{vx}|\}$. The cost c_v is the cost of all the superedges *(v, x)* to its neighbour x belongs to N_v. The cost reduction for any two supernodes *(u, v)* in V_x is calculated by

$$s(u, v) = (c_u + c_v - c_w)/(c_u + c_v) \tag{1}$$

where w is the supernode formed by the merging of u and v and the ratio is cost reduction after merging of two nodes into a supernode to the cost of the two nodes before the merge.

The GREEDY algorithm works as follows:

1. Initialization
2. Iterative
3. Output

In the Initialization phase we calculate the positive cost reduction of nodes that are 2 hops apart and have a common neighbour. The s(.) value of such pairs is calculated in this part. 2 hops neighbours are considered as they will give the maximum cost reduction due to the presence of common neighbour. Also, it is observed that the maximum cost reduction after the merge can be 0.5 for a pair of nodes having a common neighbour whereas the minimum value can be a negative value. All the node pairs with s(.) value greater than 0 are maintained in a Heap structure for later use in the Iterative phase.

In the Iterative merging phase, the pair with the highest s(.) value is picked from the heap. Once the higher value is determined for the pair, the pairs are removed from the supernode V_s and merged into a new supernode and put back in V_s. The values in the heap must be changed accordingly as the new graph does not contain the supernodes before the merge. So, the new supernode must be inserted into the Heap. Other supernodes those are not part of the nodes before and after the merge may have a change in their s(.) values. The edge cost for the neighbour node x belongs to N_v may have changed due to the merge.

Let us take the below example : [FIGURE]

From our above example the top contender for the highest cost reduction are nodes p and k as both have a common neighbour n. Both p and k have two edges associated with them so each will have a cost of 2. Once merged to supernode p_k it will also have a cost of 2 since it has a self-edge along with a superedge connection to n. The cost reduction for p_k will be $s(p, k) = (2 + 2 + 2)/(2 + 2) = 0.5$. Other

node pairs to be considered are *(y, a)* with $s(.) = 0.6 and (s, v) = 0.6$. Both are having the same value as both pairs have same number of incident edges in combination. Suppose we take the pair *(y, a)*, then $s(y, a) = (3 + 2–(3 - 1))/(3 + 2) = 0.6$ with one negative edge correction. If we take the pair (s, v), then $s(s, v) = (2+3–(3–1))/(3+2) = 0.6$. Let us choose the pair (y, a) for further proceeding of the algorithm. With pair (y, a) the no of incident edges on y and a are 3 and 2 respectively. After the merge the $s(y, a) = (3+2–(3-1))/(3+2) = 0.6$ with one edge correction $–(n, a)$. In the next step we will consider the pair (s, v) and the $s(s, v) = 2/3$ as s has 1 incident edge and v has 2 incident edge on it. So, $s(s, v) = (1 + 2–(1 + 1–1))/(1 + 2) = 2/3$. After all the merges all the pairs will have negative cost reduction so the GREEDY will terminate.

The third phase which is the output phase all the summary edges with the correction edges for the pairs are created. Superedge will be present in the summary if $|A_{uv}| > (|\prod_{uv}| + 1)/2$. We add -ve edge corrections in this case for all node pairs $(a, b) \in \prod_{uv} - A_{uv}$; We add $+ve$ edge corrections otherwise.

3.1.1 Time Complexity The total time complexity for GREEDY depends on the neighbours of the supernode w. Each neighbour x of w will have pairs containing x and such pairs will be approximately 2 Hops from x giving us $2Hop(x)$. Adding up all such neighbours of w will become 3 $Hop(w)$ including w The time required for computing the cost and considering all the edges of both the nodes and making an update to each pair in H will be $O(log\,|H|)$. If there are n nodes in G, then the size of the heap $|H| = n.2Hop(w)$. Therefore, the total time complexity of every merge step is $O(4Hop(w) + 3Hop(w).(logn + log2Hop(w)))$. The average degree of $dav \leq n$ for each node is $O(d3av(dav + logn + logdav))$.

3.1.2 Optimizations As per experiments conducted by the authors of this paper due to the large size of the graphs the time taken to update the Heap structure is more than the running time of the algorithm. Due to this the processing time increases. To overcome this one approach can be to fix a threshold value τ. This value will be used to only compute only those node pairs those have a cost reduction value $s(.) > \tau$. Thus, fixing this value we will insure to process only those nodes in the Heap with greater value than the threshold value. Once an iteration is finished the τ value will be reduced with every round until it becomes 0. With experiments conducted it is found that fixing the τ to .25 and decreasing it by .05 in very round yields good result. This approach reduces the running time while approximately giving the same result as when run with the original GREEDY algorithm.

3.2 Randomized Algorithm

[ALGORITHM]

Greedy Algorithm is slow as it needs to find all pairs with max $s(.)$ value at 2 distance Hop. It performs a cost reduction for nodes in the 3-hop neighbourhood

of the merged pair. GREEDY has to choose the best pair to perform the merge in each step. To reduce the computational complexity a new algorithm called as the RANDOMIZED algorithm. Unlike GREEDY this algorithm randomly picks a node and merges it with the node in its 2-hop distance. Using this approach, it makes it very easy to compute large scale graphs as it saves the initial time for computing the best pair for merging to happen.

With this algorithm it merges the nodes randomly to make supernodes V_s. These supernodes are then put into two categories U (unfinished) or Finished (F). The finished category nodes do not give any further cost reduction when processed with other nodes. On the other hand, the unfinished nodes do give cost reduction when merged with other nodes. In the beginning all the nodes are assigned to the unfinished category. In every step a random node u is picked, and a neighbour node v is found such that it gives the largest $s(u, v)$ value when for all the pair containing u. If after merging the cost reduction becomes negative, then the node will be moved to F. The merging will take place until all the nodes are in F. The output phase remains same as the GREEDY algorithm.

3.2.1 Time Complexity Each merge steps for 2 hops will take $2Hop(v)$. The time complexity for the merge step for all the 2 hop pairs with v is $O(3Hop(v))$.

4 COMPUTING ϵ - REPRESENTATION

The above algorithms discuss to minimise or compress the size of the original graph G by merging of nodes to supernodes and edges to superedge and making some corrections which can be applied to the summary to recreate the original graph. In this section we will see two more schemes to lower the cost and size of corrections of the ϵ -representation $R\epsilon$.

4.1 The APXMDL Algorithm

In this approach the representation R of a graph is taken which is computed by using GREEDY or RANDOMIZED discussed in the above sections. With this R the $R\epsilon$ is calculated by removing edges from C and S satisfying the equation 1. We have two steps to be followed. First, we remove unnecessary edge corrections from C and second remove edges from S. For a graph G, a new graph G' is constructed which has same number of nodes as in G and edges as a subset of edges of $G(E'_G \subseteq E_G)$. The neighbours of G' should satisfy equation 1. We find a set of edges M from the graph G using b – matching problem where the at most number of edges incident on a node is b_v. Removing these edges from the graph does not violate the approximation discussed above. A new graph H is constructed with the edges present in C. Thus $C_\epsilon = C–M$. Now to calculate S_ϵ the superedge (u, v) must be selected in S without correction C_ϵ and with increasing $|\prod_{uv}|$ value. The superedge (u, v) can be discarded without violating the ϵ – constraint for any node $(A_u \cup A_v)$.

4.2 The APXGREEDY Algorithm

The APXMDL fails to maximize the use of the approximate constraints. APX-GREEDY take the original graph and follows the same approach as GREEDY. However, the approach to compute the node cost c_v is different. It uses the knowledge of the ϵ to compute c'_v of the node v. The ϵ constraint v'_s is calculated with the minimum use of edges from summary S and correction C. From the above example in Figure 1 and Figure 2 for $\epsilon = 1/3$, updated cost of node y is $c'_y = 2$ but not 3 as 1 incident edge on y can be removed satisfying the ϵ constraint. Deleting the correction $-(n, a)$ makes the new cost of the supernode (ya) becomes 1 i.e. $c'_{(ya)} = 1$ and not 2 without violating the ϵ constraint of n and a. New cost reduction becomes $s'(u, v) = ((c'_u + c'_v - c'_w)/(c'_u + c'_v))$. GREEDY is run now but with the updated $s'(.)$ value. While computing the new cost of the c'_v it is very important to check which edge to be deleted from the correction between 2 nodes. The neighbouring nodes must not be affected while deleting the correction edges. To compute C'_v efficiently ϵ – constraints of $V's$ neighbours should not be considered. There will be slight impact on the cost c'_v but the computation will be fast. Let e_a be the number of edge correction edges removed for a node $\in A_v then C'_v = C_v - \sum_{a \in A_v} e_a$.

5 Conclusion and Highlights

The main points to be noticed and to be highlighted are as follows:

1. The authors of this paper have extensively worked on highly compression of a graph G in two-part representation technique i.e. $R = (S, C)$.
2. The compression is done w.r.t to the MDL Principle.
3. In R, the S represents the aggregated graph with High level summary of G.
4. In R, the C represents the corrections using which the graph G can be recreated.
5. Computed representation for both lossy and lossless reconstruction of graphs with bounded error.
6. Algorithms to compute these representations with minimum cost.
7. Allow self-edges which are not present in most of the graph compression techniques.
8. Graphs can be either directed or undirected but because of the easy understanding only undirected graphs are considered.

6 Future Work

1. Many real-life graphs can have various attributes on nodes and edges.
2. Web graphs with URLs, Basket data with monetary values and IP traffic with port numbers and type of traffic.
3. Work can be done to see how to handle such graphs with these attributes carried with them.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.
Displayed equations are centered and set on a separate line.

$$x + y = z \tag{2}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 1).

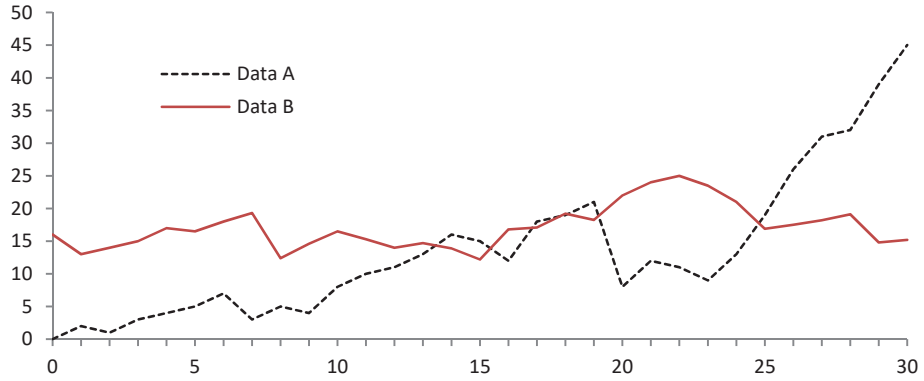

Fig. 1. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [1], an LNCS chapter [2], a book [3], proceedings without editors [4], and a homepage [5]. Multiple citations are grouped [1–3], [1, 3–5].

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017