Journal of Visual Language and Computing

journal homepage: www.ksiresearch.org/jvlc

GraPH: Graph Partitioning Based on Hotspots

Hiba G. Fareed[a], Isam A. Alobaidi[b,c,*], Jennifer L. Leopold[d] and Andrea E. Smith[d]

[a]Mathematics Department, Mustansiriyah University, Baghdad, Iraq
[b]School of Computer Science and Information Systems, Northwest Missouri State University, Maryville, MO, USA
[c]Department of Computer Engineering, Al Farabi University College, Baghdad, Iraq
[d]Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA

ARTICLE INFO

Article History:
Submitted 10.25.2023
Revised 11.5.2023
Accepted 12.5.2023

Keywords:
graph partitioning
graph data mining
structures
hotspot.

ABSTRACT

Graphs have long been used to model relationships between entities. For some applications, a single graph is sufficient; for other problems, a collection of graphs may be more appropriate to represent the underlying data. Many contemporary problem domains, for which graphs are an ideal data model, contain an enormous amount of data (e.g., social networks). Hence, researchers frequently employ parallelized or distributed processing. But first the graph data must be partitioned and assigned to the multiple processors in such a way that the work load will be balanced, and inter-processor communication will be minimized. The latter problem may be complicated by the existence of edges between vertices in a graph that have been assigned to different processors. Herein we introduce a strategy that combines vocabulary-based summarization of graphs (VoG) and detection of hotspots (i.e., vertices of high degree) to determine how a single undirected graph should be partitioned to optimize multi-processor load balancing and minimize the number of edges that exist between the partitioned subgraphs. We benchmark our method against another well-known partitioning algorithm ($METIS$) to demonstrate the benefits of our approach.

1. Introduction

Graphs are frequently used as an abstraction to model the real-world data; such as in chemical, or biological networks. The nature of the application problem will determine whether these data will be represented in a single or a collection of graphs. This diversity has contributed to the proper representation of the underlying data. Some of these graphs may contain an enormous amount of data (e.g., social networks). Hence, parallelized or distributed processing often is employed. Before the analysis commences, typically the graph dataset is partitioned, and a subset of data is assigned to each processor. The partitioning should be done in such a way that the ensuing work load will be balanced and inter-processor communication will be minimized. These tasks can be particularly challenging for a single graph; consideration must be given to which vertices are assigned to which partitions (i.e., processors) and what edges originally existed between those vertices.

Ideally, partitions should be of approximately equal size, and the number of edges between vertices that are in different partitions should be minimized. The problem of finding good partitions in these respects has been studied in graph theory. Despite the numerous algorithms that have been proposed and implemented, the complexity of this problem is still considered NP-complete.

In general, most graph partitioning algorithms utilize either edge-cut partitioning or vertex-cut partitioning. Edge-cut partitioning splits the vertices of a graph into disjoint sets of approximately equal size considering the minimum number of cut-edges (e.g., PowerGraph [3], Spark GraphX [4], and Chaos [14]). In contrast, vertex-cut partitioning splits the edges of a graph into equal-sized sets. In this approach, the partitioning of a single graph must satisfy two require-

*Corresponding author
hf_math@uomustansiriyah.edu.iq (H.G. Fareed); ialobaidi@nwmissouri.edu (I.A. Alobaidi); leopoldj@mst.edu (J.L. Leopold); aes7dc@mst.edu (A.E. Smith)
https://uomustansiriyah.edu.iq/e-learn/profile.php?id=5609 (H.G. Fareed); https://www.nwmissouri.edu/csis/directory/alobaidi.htm (I.A. Alobaidi); https://cs.mst.edu/people/faculty-directory/ (J.L. Leopold)
ORCID(s): 0000-0002-6508-2495 (H.G. Fareed); 0000-0001-6329-2440 (I.A. Alobaidi)

DOI reference number: 10-18293/JVLC2023-N2-039

ments: the quality graph partitioning criterion (which guarantees no lost data) and load balancing. Many studies have shown that edge-cut partitioning produces more accurate results on large real-world graphs [3, 4].

Herein we introduce a novel vertex-cut partitioning strategy that determines how a single, undirected graph should be partitioned to optimize multi-processor load balancing and minimize the number of edges that exist between the partitioned subgraphs. Our approach, $GraPH$, first uses vocabulary-based summarization [9] to identify the most highly connected structures that exist in the graph (e.g., cliques, stars, and chains). We then find the vertices in those structures that have the highest degree; these are called hotspots. The hotspots become the starting points from which subgraph partitions are formed.

This paper is organized as follows. In Section 2 we briefly discuss some of the related work in graph partitioning. We present the $GraPH$ algorithm in Section 3, and include a discussion of the VoG summarization algorithm. In Section 4 we experimentally evaluate our proposed algorithm ($GraPH$) to expound its benefits. Concluding remarks and a discussion of future work are provided in Section 5.

2. Related Work

In this section, we briefly review some of the research that has been done in graph partitioning. Despite the numerous sequential, distributed, and parallel algorithms that have been developed, the complexity of this problem is still considered to be NP-complete. One of the most significant challenges of the problem continues to be minimizing the loss of information (from the original graph dataset) when the partitions are formed; that is, the goal is to minimize the number of edges (from the original graph) that exists between vertices that are in different partitions, a situation which is more likely to occur as the number of partitions increases.

Some heuristic methods for sequential graph partitioning of a single graph are discussed in [6, 2]. One offline method (wherein the entire graph is resident in memory), $METIS$, is proposed in [6]. This method produces high-quality partitions in terms of uniformity of partition size and minimization of "lost" edges. However, because of the offline setting, it cannot handle large graphs. The $METIS$ algorithm consists of three phases: coarsening, partitioning, and refinement. During each phase, a sequence of specialized algorithms is applied. These algorithms help in selecting the maximal matchings in the coarsening phase, partitioning of the coarse graph in the partitioning phase, and projecting the graph back to the original graph in the refinement phase. An extension to $METIS$ (Streaming $METIS$ Partitioning method (SMP)) is proposed in [2], replacing the offline setting of $METIS$ by an online setting. SMP provides the ability to adjust the memory capacity, and subsequently decrease computational requirements by applying the partitioning method to small subgraphs.

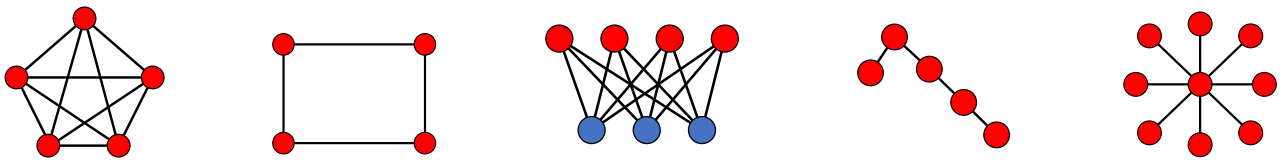
Some graph partitioning techniques are designed for specific application problems. Another technique for local (i.e., memory-resident, sequential processing) graph partitioning [1] specifically targets fixed cardinality problems such as k-densest subgraph and max k-vertex cover. The authors developed a fixed parameter algorithm using a greediness-for-parameterization technique. Clustering systems are used as a base in [17]. In this research, the authors propose a heuristic graph edge partitioning strategy, Neighbor Expansion (NE), with polynomial running time. Their goal was to reduce the running time and communication cost for some specific applications such as triangle counting and PageRank.

The graph partitioning problem in a distributed environment is addressed in [12, 11, 8, 15, 7]. The authors in [12] propose a fully distributed algorithm called JA-BE-JA. This algorithm is built on two types of partitioning: vertex-cut and edge-cut partitioning; the absence of central coordination and the processing of each vertex independently make this algorithm well-designed for distributed processing. Another distributed algorithm, $PACC$ (Partition-Aware Connected Components), based on graph partitioning for edge-filtering and load-balancing, is proposed in [11]. The authors of [15] propose a multi-level label propagation (MLP) method that uses distributed memory of several machines for partitioning the graphs. Another distributed partitioning algorithm is discussed in [10], PARallel Submodular Approximation algorithm ($Parsa$), also configures the partitions to fit the storage and computation ability of each machine.

One important characteristic of graph partitioning algorithms is the strategy employed for selecting the vertex around which the subgraph will be built for each partition. Many algorithms select such vertices randomly. Our approach was motivated by $MELT$ [16], MapReduce-based Efficient Large-scale Trajectory anonymization. The main objective of that work was to examine paths traveled by people in a geographical space, and then partition the space into regions around popular locations (e.g., a coffee house, an exercise center, etc.); those locations are referred to as hotspots. As will be discussed later in this paper, the utilization of hotspots as a basis for forming partitions is a novel feature of our partitioning strategy.

3. Methodology

In this section, we present the $GraPH$ strategy for partitioning a single, undirected graph. We begin with some preliminary definitions that will facilitate this discussion. An explanation of the vocabulary-based summarization of graphs (VoG) technique developed in [9] then follows; this is a key component for our approach as it is used to determine subgraphs of high connectivity (e.g., cliques, stars, and chains). Finally, our complete set of algorithms is presented, detailing how the vocabulary-based summarization and identification of hotspots lead to the creation of optimal partitioning.



(a) Full Clique. (b) Near Clique. (c) Full Bipartite. (d) Chain. (e) Star.

Figure 1: Types of Structures.

3.1. Preliminaries

Definition 1. ***Graph:*** *A graph G is a tuple (V, E, L) where V is a finite set of nodes called the vertex set of G, and E is a set of 2-element subsets of V ($E \subseteq V \times V$) called the edge set of G. The nodes and edges are labeled by the function L.*

Definition 2. ***Graph partitioning:*** *A graph $G = (V, E)$ will be partitioned into k subgraphs $G'_{sub} = (V', E')$, $sub = 1, ..., k$. Each $V'_{subset} \subset V_{set}$ where $V_i \cap V_j = 0$ for $i \neq j$, and each $E'_{subset} \subset E_{set}$.*

Definition 3. ***Full-clique:*** *Let $G = (V, E)$ be an undirected graph. A set FC of vertices in G is called a Full-clique if any two distinct vertices in FC are adjacent in G, when $k \geq 1$. The Full-clique term may refer to the subgraph in some cases. If several edges are missing, this will be defined as a* ***Near-clique****.*

Definition 4. ***Full bipartite core:*** *Let $G = (V, E)$ be an undirected graph. A set Fb of vertices in G is called Full-bipartite if two sets of vertices S_1 and S_2, $S_1 \cap S_2 = \emptyset$, have edges between them, where each vertex in S_1 will be connected to every edge in S_2 but not within the same set. When the core is not fully connected this will be defined as a* ***Near-bipartite core****.*

Definition 5. ***Star:*** *A* ***Star*** *consists of one internal vertex in set S_1 connected to k edges of other sets S_{i+1} (spokes). A Star is considered as a special case of a Full bipartite core.*

Definition 6. ***Chain:*** *A* ***Chain*** *is a sequence of vertices such that all vertices have degree 2, except two of them have degree 1.*

Figure 1 shows examples of these structure types.

3.2. *VoG* Graph Summarization

The ability to summarize information about highly connected subgraphs contained within a large graph can greatly facilitate understanding of the graph as a whole. Vocabulary-based summarization of Graphs (VoG) [9] is a formal methodology developed for this purpose. Using a set of terms (i.e., a vocabulary) like full-cliques, near-cliques, full-bipartite core, near-bipartite core, stars, and chains, VoG provides a summary of the most highly connected and frequently occurring structures in a graph. For problem domains like social networks and communication networks, these are typically the structures of most interest.

Algorithm 1 outlines the main steps that are performed in VoG; see [9] for a more detailed discussion. Using graph decomposition methods, candidate subgraphs are first generated. They are then classified as various connected structures such as cliques, stars, and chains; if a subgraph qualifies as more than one of these structure types, a scoring method (based on minimum description length (MDL)) is used to determine which structure type that subgraph best fits. VoG then uses another scoring system to determine which collection of those structures best characterizes the graph as a whole. This is called the summary model, and could include all of the structures (PLAIN), just the k structures with the best scores (TOP10, TOP100), or a combination of structures whose total score is best (GREEDY'nFORGET).

Algorithm 1 VoG

1: **Input** Graph G.
2: **Output** Graph summary M, encoding cost.
3: **Subgraph Generation.** Using graph decomposition methods, produce a set of candidate subgraphs, which may overlap with each other.
4: **Subgraph Labeling.** Characterize each subgraph as one of the vocabulary structure types.
5: **Summary Assembly.** From the candidate structures, select a non-redundant subset to instantiate the graph model M. Utilizing a heuristic model (e.g., PLAIN, TOP10, TOP100, GREEDY'nFORGET), the set of structures with the lowest description cost will be selected.

3.3. Proposed Algorithm

Two algorithms have been proposed here one dealing with Sequential processing while the other dealing with Parallel processing

3.3.1. Sequential Algorithm

In $GraPH$, we first use VoG to identify the most highly connected, and frequently occurring, subgraphs. That produces a set of structures (i.e., the model summary), S. Algorithm 2 is then used to select a subset of S which we call the majority structures, $MajS$. The number of majority structures depends on the desired number of partitions, n. The n structures in S that have the largest number of vertices become the majority structures.

For each majority structure, Algorithm 3 is applied to identify the vertex that has the highest degree; in the case of a tie, an arbitrary choice between those qualifying vertices is made. These vertices of highest degree are called hotspots.

Algorithm 2 Select the Majority Structures

Input S is set of structures produced by VoG,
  n is number of desired partitions
Output $MajS$ contains n structures in S that have the largest number of vertices
$SortedS$ = Sort structures in S in descending order by number of vertices;
  for $i = 1$ to n **do**
    $MajS[i] = SortedS[i]$
  end-for
return $MajS$

Algorithm 3 Assign the HotSpot

Input $S = (V_S, E_S)$ is a structure
Output $HotSpot$ is a vertex in V_S that is the hotspot vertex for structure $S = (V_S, E_S)$
  for $i = 1$ to $|V_S|$ **do**
    $degree[i] = 0$
  end-for
  for $i = 1$ to $|V_S|$ **do**
    for $j = 1$ to $|V_S|$ **do**
      if there is an $edge(i, j)$ in E_S
        then $degree[i] = degree[i] + 1$
      end-if
    end-for
  end-for
  $HotSpot = 1$
  for $i = 2$ to $|V_S|$ **do**
    if $degree[HotSpot] <= degree[i]$
      then $HotSpot = i$
    end-if
  end-for
return $HotSpot$

After assigning the hotspots, the actual partitioning commences. The subgraph that will be assigned to a partition will consist of all the vertices in a hotspot's structure unless that number of vertices exceeds the total number of vertices in the graph divided by the number of desired partitions; that is considered the ideal partition size. In Algorithm 4, we start a depth-first search from a hotspot vertex (denoted as Hotspot). The $MajS$ denoted in the algorithm is the set of structures from which the hotspot was selected. There are two discontinuation criteria for building a subgraph partition; the expansion will stop when either of those conditions is satisfied:

1. The current size of a partition subgraph has reached the ideal partition size.
2. The path length from the current vertex to the hotspot has reached a maximum threshold (i.e., the total number of desired partitions).

Some vertices from the original graph may not be included in any partition using these conditions. To handle those cases, we perform a breadth-first search starting from each hotspot until all nodes are included in some partition.

Algorithm 4 GraPH

Input Graph $G = (V, E)$ and $HotSpot$ and $MajS$
  $MajS$ is a set contains structures that have the largest number of vertices
  $HotSpot$ is a vertex in the structure connected to the largest number of edges
  n is the number of partitions
Output All $SubGraph$s of G, where $|V|$ of each subgraph $\geq PartitionSize$
$PartitionSize = |V| \ / \ n$
  if $|MajS_i| \leq PartitionSize$ **then**
  Include all nodes of $MajS_i$ in $Partition_i$
  end-if
  Perform DFS starting from each $HotSpot$
  $SubGraph_{DFS} \leftarrow DFS$ result
  Perform BFS starting from each $HotSpot$
  $SubGraph_{BFS} \leftarrow BFS$ result
$SubGraph \leftarrow SubGraph_{DFS} \cup SubGraph_{BFS}$
return $SubGraph$

3.4. Computational Complexity

The complexity of one well-known partitioning method that is considered to produce high-quality partitions, $METIS$ [6] (implemented as kmetis), is approximately $O(V + E + k\log(k))$ where V is the number of nodes, E the number of edges, and k is the number of partitions [5]. In contrast, the complexity of $GraPH$ is approximately $O(V + E + n\log(n))$ where V is the number of nodes, E is the number of edges, and n is the number of structures. Contributing to the overall complexity of $GraPH$ is the complexity of BFS and DFS, which are $O(V + E)$, and the complexity of sorting n structures, which is $O(nlog(n))$. We are not including the complexity of the VoG processing, which has not been published by its authors.

4. Results and Analysis

In this section we compare the results of partitioning three datasets using $GraPH$ and another well-known partitioning method, $METIS$, which was discussed in Section 2. The $GraPH$ algorithms presented in Section 3.2 and 3.3.1 were (collectively) implemented in Matlab and C++. A C++ implementation of $METIS$ was downloaded from the Karypis Lab website [5]. Our experiments were executed on an Intel(R) Core(TM) i7-6700 CPU@3.40GHz computer with 32 GB memory.

4.1. Data Description

Three single undirected graphs were used to evaluate our approach. Table 1 lists descriptive information about the graphs. One graph was synthetically generated; a second graph represented a two-dimensional finite element mesh; the third graph represented a three-dimensional finite element mesh. The last three graphs were obtained from the Network Repository,a large comprehensive collection of network graph data [13].

Table 1
Description of the Graphs Tested

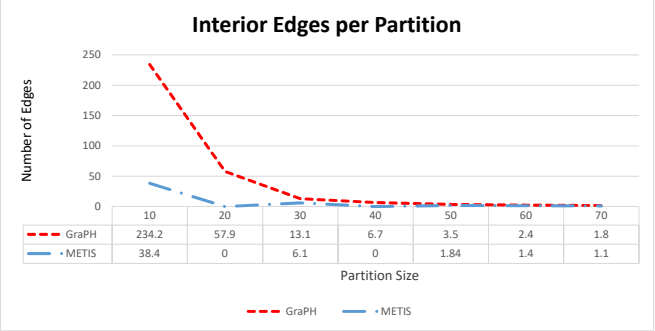
Graph Name	Number of Nodes	Number of Edges	Description
Synthetic	1565	3561	Synthetically generated
4ELT	15606	45878	2D Finite element mesh
COPTER2	55476	352238	3D Finite element mesh
web-wikipedia-link-fr	4.9M	113.1M	Power-Law
road-road-usa	23.9M	28.8M	Low-Degree
soc-sinaweibo	58.6M	261.3M	Long-Tailed

4.2. Experiment and Results

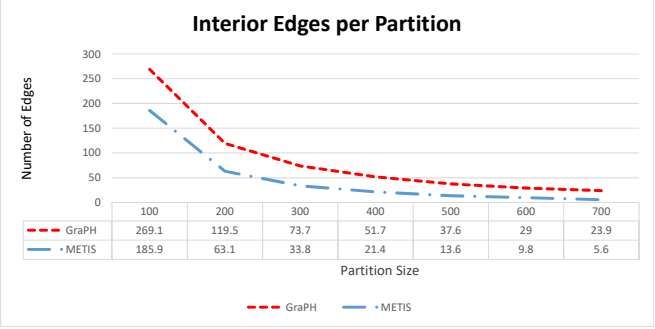
We executed *GraPH* and *METIS* on each of the graphs listed in Table 1, testing seven different numbers of partitions for each graph. The results from each test were analyzed in terms of three different metrics: the number of *interior edges* per partition (i.e., edges in a partition's graph), the number of *exterior edges* per partition (i.e., edges between vertices in a partition and vertices assigned to other partitions), and the total number of *edges lost* (i.e., edges from the original graph that were not represented in any of the partition graphs).

Seven tests were conducted to create 10, 20, 30, 40, 50, 60, and 70 partitions, respectively, of the Synthetic graph. *METIS* failed to partition this graph into either 20 or 40 partitions; the program simply failed to return any results. *GraPH* produced results for all of the tested numbers of partitions for this graph. The representation of edges amongst partitions was not well distributed when 10 partitions were requested. Specifically, the number of interior edges in one of those partitions was much higher than in the other partitions, which was not an optimal partitioning. This was likely due to the fact that when a hotspot is selected from a structure, if the structure can fit entirely into a partition, all nodes from that structure automatically will be added to the partition before the depth-first search algorithm is run. This can then prevent other partitions from growing during depth-first search (as would be the case in unconnected components), encouraging disproportionate partition sizes.

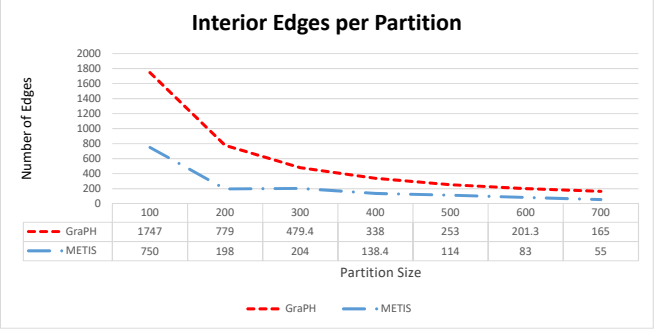
Because the 4ELT and COPTER2 graphs were much larger than the Synthetic graph, we tested larger numbers of partitions for those graphs, namely: 100, 200, 300, 400, 500, 600, and 700. For all three of the graphs listed in Table 1, in the majority of the tests, the partitions produced by *GraPH* had a higher number of interior edges in each partition than the partitions produced by *METIS*. It can be seen in Figure 2 that more edges from the original graph were retained within the partitions produced by



(a) 1565 Nodes - 3561 Edges.



(b) 15606 Nodes - 45878 Edges.



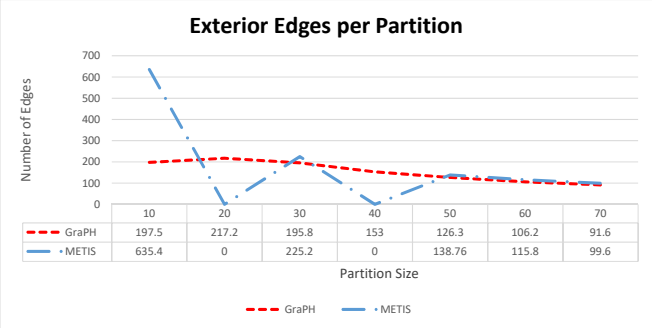
(c) 55476 Nodes - 352238 Edges.

Figure 2: Interior Edges per Partition.

GraPH. As shown in Figure 3, the *GraPH* partitioning resulted in fewer exterior edges (between partitions) than what occurred in the *METIS* partitioning. Additionally, as shown in Figure 4, *GraPH* outperformed *METIS* in terms of reducing the total number of edges lost from the original graph. It should be noted that as the desired number of partitions grew, the difference in partition quality (in terms of the three metrics) obtained from both methods became less distinct.

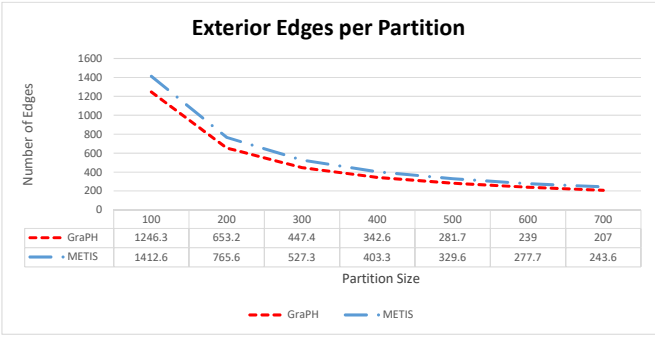
Because of the use of two methods (depth-first/breadth-first search) in *GraPH* for the extension process that include vertices in/out of partition boundaries, we also evaluated different variations of our method. We ran *GraPH* on the three test graphs using four different orders of processing:

- Depth-first search extension for vertices inside the partition boundaries followed by breadth-first search ex-



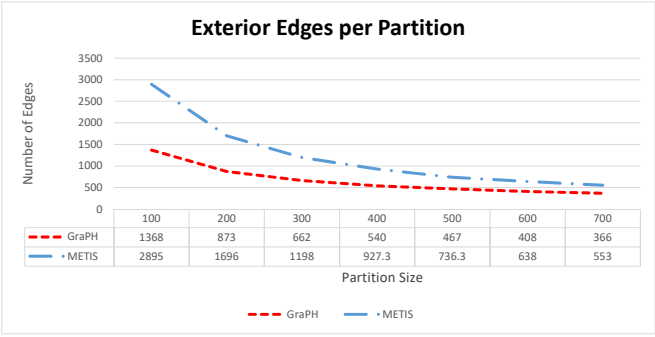
	10	20	30	40	50	60	70
GraPH	197.5	217.2	195.8	153	126.3	106.2	91.6
METIS	635.4	0	225.2	0	138.76	115.8	99.6

(a) 1565 Nodes - 3561 Edges.



	100	200	300	400	500	600	700
GraPH	1246.3	653.2	447.4	342.6	281.7	239	207
METIS	1412.6	765.6	527.3	403.3	329.6	277.7	243.6

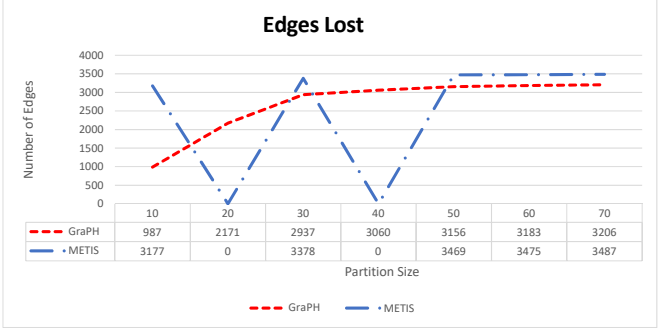
(b) 15606 Nodes - 45878 Edges.



	100	200	300	400	500	600	700
GraPH	1368	873	662	540	467	408	366
METIS	2895	1696	1198	927.3	736.3	638	553

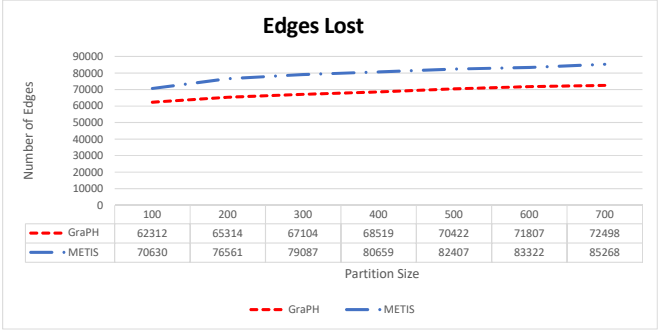
(c) 55476 Nodes - 352238 Edges.

Figure 3: Exterior Edges per Partition.



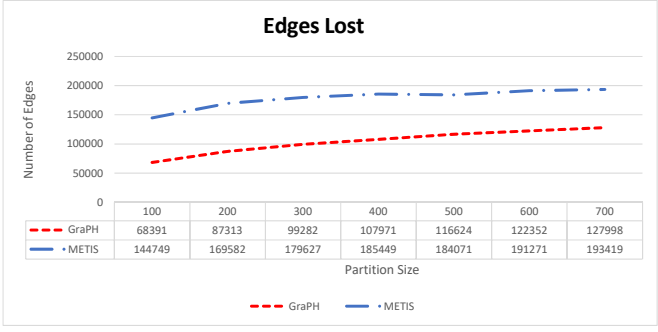
	10	20	30	40	50	60	70
GraPH	987	2171	2937	3060	3156	3183	3206
METIS	3177	0	3378	0	3469	3475	3487

(a) 1565 Nodes - 3561 Edges.



	100	200	300	400	500	600	700
GraPH	62312	65314	67104	68519	70422	71807	72498
METIS	70630	76561	79087	80659	82407	83322	85268

(b) 15606 Nodes - 45878 Edges.



	100	200	300	400	500	600	700
GraPH	68391	87313	99282	107971	116624	122352	127998
METIS	144749	169582	179627	185449	184071	191271	193419

(c) 55476 Nodes - 352238 Edges.

Figure 4: Total Edges Lost.

tension for vertices outside the partition boundaries.

- Breadth-first search extension for vertices inside the partition boundaries followed by depth-first search extension for vertices outside the partition boundaries.
- Depth-first search extension for vertices inside the partition boundaries followed by depth-first search extension for vertices outside the partition boundaries.
- Breadth-first search extension for vertices inside the partition boundaries followed by breadth-first search extension for vertices outside the partition boundaries.

We found that more consistent partitions were obtained (in terms of more interior edges and fewer external edges per partition) when we utilized the depth-first search extension process for vertices inside the boundaries followed by breadth-first search extension processing for vertices outside the boundaries. We also tested random assignment of hotspots. This was found to be unreliable in generating high-quality partitions. Interestingly, although the number of internal edges was not balanced across partitions utilizing randomization, $GraPH$ still outperformed $METIS$ in terms of producing partitions with more internal edges and fewer external edges.

5. Conclusion and Future work

With the proliferation of data in our technological world and the usefulness of modeling some problems using graphs, it is becoming increasingly difficult to process an entire graph dataset in memory. It is more efficient to partition a single large graph, and process multiple smaller subgraphs. However, in doing so, the partitioning of what may be highly interconnected data must be done in such

as way as to balance the work load amongst the individual processes, minimize inter-process communication, and minimize loss of information from the original dataset. The latter problems can occur if, in the original graph, there is an edge that exists between vertices assigned to different partitions.

Herein we have presented an algorithm, $GraPH$, for partitioning a single, undirected graph. Our algorithm strives to produce quality partitions in terms of: uniformity of the size of each partition, maximization of the number of edges from the original graph that are included in each partition, and minimization of the number of edges from the original graph that effectively exist between partitions. Our approach is novel; we first utilize vocabulary-based summarization (VoG) to find the most highly connected structures, and then find the vertices of highest degree (known as hotspots) within those structures. A benchmark comparison of $GraPH$ with another well-known, high-quality partitioning algorithm ($METIS$) demonstrated the benefits of our strategy.

In the future, we plan to explore ways to distribute or parallelize the $GraPH$ algorithms so that we can process even larger graphs than those tested for this study. To that end, we also may explore the use of some approximation (e.g., sampling) methods that may increase the efficiency of the assignment of vertices to partitions after identification of structures and hotspots.

6. Acknowledgments

The authors thank Dr. Danai Koutra for her assistance in executing the VoG software.

References

[1] Bonnet, É., Escoffier, B., Paschos, V.T., Tourniaire, É., 2015. Multi-parameter Analysis for Local Graph Partitioning Problems: Using Greediness for Parameterization. Algorithmica 71, 566–580. URL: https://doi.org/10.1007/s00453-014-9920-6, doi:10.1007/s00453-014-9920-6.

[2] Echbarthi, G., Kheddouci, H., 2016. Streaming METIS Partitioning, in: Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM '16), IEEE Press, Piscataway, NJ, USA. pp. 17–24. URL: http://dl.acm.org/citation.cfm?id=3192424.3192429, doi:10.1109/ASONAM.2016.7752208.

[3] Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C., 2012. Powergraph: Distributed Graph-parallel Computation on Natural Graphs, in: Proceedings of the 10th. USENIX Symposium on Operating Systems Design and Implementation (OSDI '12), USENIX Association, Hollywood, CA, USA. pp. 17–30.

[4] Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I., 2014. Graphx: Graph Processing in a Distributed Dataflow Framework, in: Proceedings of the 11th. USENIX Symposium on Operating Systems Design and Implementation (OSDI '14), USENIX Association, Broomfield, CO, USA. pp. 599–613.

[5] Karypis, G., 2007. Complexity of pmetis and kmetis Algorithms. http://glaros.dtc.umn.edu/gkhome/node/419. Accessed: 2019-22-01.

[6] Karypis, G., Kumar, V., 1998a. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing 20, 359–392. URL: https://doi.org/10.1137/S1064827595287997, doi:10.1137/S1064827595287997.

[7] Karypis, G., Kumar, V., 1998b. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. Journal of Parallel and Distributed Computing 48, 71–95. URL: http://www.sciencedirect.com/science/article/pii/S0743731597914039, doi:https://doi.org/10.1006/jpdc.1997.1403.

[8] Kiveris, R., Lattanzi, S., Mirrokni, V., Rastogi, V., Vassilvitskii, S., 2014. Connected Components in Mapreduce and Beyond, in: Proceedings of the ACM Symposium on Cloud Computing (SOCC '14), ACM, New York, NY, USA. pp. 18:1–18:13. URL: http://doi.acm.org/10.1145/2670979.2670997, doi:10.1145/2670979.2670997.

[9] Koutra, D., Kang, U., Vreeken, J., Faloutsos, C., 2015. Summarizing and Understanding Large Graphs. Statistical Analysis and Data Mining: The ASA Data Science Journal 8, 183–202. doi:10.1002/sam.11267.

[10] Li, M., Andersen, D.G., Smola, A.J., 2015. Graph Partitioning via Parallel Submodular Approximation to Accelerate Distributed Machine Learning. CoRR 1505.04636. URL: http://arxiv.org/abs/1505.04636.

[11] Park, H.M., Park, N., Myaeng, S.H., Kang, U., 2016. Partition Aware Connected Component Computation in Distributed Systems, in: Proceedings of the 16th. IEEE International Conference on Data Mining (ICDM '16), IEEE. pp. 420–429. doi:10.1109/ICDM.2016.0053.

[12] Rahimian, F., Payberah, A.H., Girdzijauskas, S., Jelasity, M., Haridi, S., 2015. A Distributed Algorithm for Large-Scale Graph Partitioning. ACM Trans. Auton. Adapt. Syst. 10, 12:1–12:24. URL: http://doi.acm.org/10.1145/2714568, doi:10.1145/2714568.

[13] Rossi, R., Ahmed, N., 2015. The network data repository with interactive graph analytics and visualization. Proceedings of the AAAI Conference on Artificial Intelligence 29. URL: https://ojs.aaai.org/index.php/AAAI/article/view/9277.

[14] Roy, A., Bindschaedler, L., Malicevic, J., Zwaenepoel, W., 2015. Chaos: Scale-out Graph Processing from Secondary Storage, in: Proceedings of the 25th. Symposium on Operating Systems Principles (SOSP '15), ACM, New York, NY, USA. pp. 410–424. URL: http://doi.acm.org/10.1145/2815400.2815408, doi:10.1145/2815400.2815408.

[15] Wang, L., Xiao, Y., Shao, B., Wang, H., 2014. How to Partition a Billion-Node Graph, in: Proceedings of the 30th. IEEE International Conference on Data Engineering (ICDE '14), IEEE. pp. 568–579. doi:10.1109/ICDE.2014.6816682.

[16] Ward, K., Lin, D., Madria, S., 2017. MELT: Mapreduce-based Efficient Large-scale Trajectory Anonymization, in: Proceedings of the 29th. International Conference on Scientific and Statistical Database Management (SSDBM '17), ACM, New York, NY, USA. pp. 35:1–35:6. URL: http://doi.acm.org/10.1145/3085504.3085581, doi:10.1145/3085504.3085581.

[17] Zhang, C., Wei, F., Liu, Q., Tang, Z.G., Li, Z., 2017. Graph edge partitioning via neighborhood heuristic, in: Proceedings of the 23rd. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17), ACM, New York, NY, USA. pp. 605–614. URL: http://doi.acm.org/10.1145/3097983.3098033, doi:10.1145/3097983.3098033.