Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2021 SCPE. Volume 22, Issues 4, pp. 413–424, DOI 10.12694/scpe.v22i4.1948

K-WAY BALANCED GRAPH PARTITIONING FOR PARALLEL COMPUTING[*]

SIDDHESHWAR V. PATIL[†] AND DINESH B. KULKARNI[‡]

Abstract. In modern computing, high-performance computing (HPC) and parallel computing require most of the decision-making in terms of distributing the payloads (input) uniformly across the available set of resources, majorly processors; the former deals with the hardware and its better utilization. In parallel computing, a larger, complex problem is broken down into multiple smaller calculations and executed simultaneously on several processors. The efficient use of resources (processors) plays a vital role in achieving the maximum throughput which necessitates uniform load distribution across available processors, i.e. load balancing.

The load balancing in parallel computing is modeled as a graph partitioning problem. In the graph partitioning problem, the weighted nodes represent the computing cost at each node, and the weighted edges represent the communication cost between the connected nodes. The goal is to partition the graph G into k partitions such that - I) the sum of weights on the nodes is approximately equal for each partition, and, II) the sum of weights on the edges across different partitions is minimum. In this paper, a novel node-weighted and edge-weighted k-way balanced graph partitioning (NWEWBGP) algorithm of $O(n^2)$ is proposed. The algorithm works for all relevant values of k, meets or improves on earlier algorithms in terms of balanced partitioning and lowest edge-cut. For evaluation and validation, the outcome is compared with the ground truth benchmarks.

Key words: Parallel Computing, Load Balancing, Graph Partitioning

AMS subject classifications. 68W10, 05C85

1. Introduction.

In computer science, graphs are commonly used to depict various real-world problems, including but not limited to social, biological, and information networks. The scale of such networks continues to grow with each passing day, especially in the domains like social networks, task scheduling, cloud computing, data centers, etc. In such examples, graph-based computations are central to the core functions of the applications.

In parallel computing, the problem of graph partitioning is more relevant [3]. There is a need to assign data/code equally among available processors while minimizing the communication overhead to exploit the parallelization. The goal is to balance the complete workload on available processors and reduce inter-process communication to achieve optimum efficiency and throughput. The operating costs of all of these systems are high, it is necessary to utilize all of the processing capacity optimally. So, to solve any problem on a parallel system, it should be divided into sub-problems. Each sub-problem has some computational load. There are dependencies between sub-problems due to communication. The effectiveness of parallel calculation is determined by the effective distribution of computational load across all available processors. Graph partitioning is a technique for simulating load balancing issues. The graph should be partitioned taking two measures into account. Firstly, the total weights on the number of nodes (computational load) of each part should be approximately equal. Secondly, to guarantee minimum communication overhead, the weights on the number of edges incident on the nodes of different partitions should be minimum [14].

It's worth noting that the majority of existing research focuses on graphs with either unit weighted nodes/edges or weighted graphs (only edge weighted). Such graphs cannot appropriately model the variety of partitioning problems. Few researchers use greedy optimization techniques [9, 12] while others use approximate optimization techniques, like spectral clustering [11], some use recursive algorithms like multilevel graph

[*]Submitted to the editors, 8 December 2021.

[†]Research Scholar, Department of CSE, Walchand College of Engineering, Sangli (Assistant Professor, ADCET, Ashta) (siddheshwar.patil@walchandsangli.ac.in).

[‡]Professor, Department of Information Technology, Walchand College of Engineering, Sangli (dinesh.kulkarni@walchandsangli.ac.in).

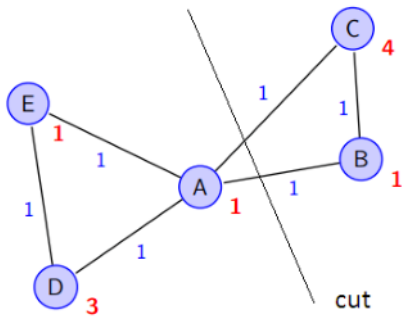

Fig. 2.1: Node-weighted balanced partitioning

partitioning [10] to solve the problem. There is little work on node-weighted graphs as compared to edge-weighted graph partitioning. Also many algorithm has running time of $O(n^3)$ and more. To address these issues, the multi-objective k-way graph partitioning (node-weighted and edge-weighted balanced graph partitioning) is proposed in this paper. This work is not similar to state-of-art approaches which use collapsing nodes and edges like multilevel graph partitioning approaches. It employs a heuristic approximation-based approach.

The NWEWBGP provides following main contributions:

1. Literature is critically reviewed and problems are identified.
2. The best partitioning initialization is selection of balanced factor that helps to improve the accuracy and decrease the computations.
3. The partitioning algorithm, NWEWBGP that address the node balance and edge-cut problem.
4. Performance evaluation of the proposed algorithm and comparison with standard benchmark tools such as Metis.

The remaining sections of the paper are organised as follows.
Section 2 introduces the graph partitioning problem. Section 3 provides state-of-the-art algorithms related to the scope of our work. The implementation, results of the proposed algorithm are discussed in Section 4. Concluding remarks are finally stated in Section 5.

2. Graph Partitioning Preliminaries. A k-way partitioning of G divides V into k disjoint parts V_0, V_1, ...,V_k. The total node weights for each subset should be approximately equal, and the total weights of the edges linking the subsets are minimal. The edges cut (cut-cost) are the edges that connect V_0 , V_1, ...V_k and it is denoted by E_c. Cut-weight is the overall weight on E_c. The weight of a subset V_i denoted by $w(V_i)$ is the total weight of the nodes/vertices in V_i i.e., $w(V_i) = \sum_{v \subset V_i} w(v)$ for $i = 0, 1, \ldots k$. Figure 2.1 shows example of balanced partitioning. There are two equal partitions ($[A, D, E]$ and $[B, C]$) with total node-weight as 5 per partition and cut-cost is 2.

3. Related Work. Graph partitioning has been the subject of significant research over many years, which was less attempted to be comprehensively reviewed. The k-way balanced graph partitioning is practically important to any application which requires large graph data. So, even though it is NP-hard and available approximation s are too expensive, various practical solutions have been proposed using heuristics [18, 1, 2]. A survey of partitioning techniques is presented in the following subsections.

3.1. Local Graph Partitioners. The category of local graph partitioners includes any partitioner which makes use of local search, which is also known as iterative vertex swapping or local refinement. Simply put, local search aims to improve an existing graph partitioning by swapping vertices between partitions to minimize some objection function, usually, minimum edge-cut. Local graph partitioners vary in how they select which vertices to swap and which partitions they will consider sending vertices between (e.g. adjacent partitions, or highly imbalanced partitions). Although the global partitioning method already produces a balanced partition, the local method tries to further improve it. The potential for improvement depends upon the difference between current cut-cost and minimum cut-cost.

3.1.1. Random Vertex Distribution. To solve the graph partitioning problem, one simple way is to randomly distribute V vertices among p parts [17]. This method starts with all parts being empty. The vertices are one after another assigned to a part by randomly choosing one part which has less than V/p vertices. The random partitioning method produces partitions with the cut size (edge-cut) of approximately $(1 - 1/p) * E$. This cut size is usually much higher than the lowest possible one. Although this method is not a good approach for graph partitioning, it may be used as starting point for efficient local partitioning methods. Furthermore, this method can be used to compare the solution calculated by the weak method to the solution calculated by a sophisticated one.

3.1.2. Greedy Methods. Greedy methods [13] are commonly used for graph partitioning. Unlike random methods, the greedy method considers adjacency information. There are many different variations of this type of approach. They all start with empty partitions and according to a certain order, they handle one vertex after the another and assign it to a part of the partition. Once an assignment for a part is done, the decision will not be changed later. A common variation is the breadth-first search which is a greedy approach based on breadth-first strategy. It starts with assigning a vertex with the minimum degree to V_0. The method proceeds in a breadth-first manner. It considers all the vertices which are not already assigned to any part. Among them, it adds all vertices to V_0 which are adjacent to any vertex already in V_0. Thus, it proceeds in levels of vertices with the same distance to the initial vertex. The method assigns vertices to V_0 until V_0 of size V/p. Another vertex from the remaining graph which is adjacent to a vertex of V_0 is taken as a new seed for V_1. V_1 and all following parts are filled in the same manner as V_0. The time required for this method is linear to the graph size. The solutions are partitions with compact parts and fairly low cuts. The disadvantage is that the last part consists of all the leftover vertices. They may form elongated or even disconnected parts. To overcome this issue, start growing the parts simultaneously at several vertices, one for each part. The calculation of the two vertices with maximum distance is very time-consuming. Therefore, several heuristics are known to calculate two vertices with high distances.

3.1.3. Kernighan-Lin. Kernighan and Lin [8] proposed a classic example of a local search . Indeed perhaps the first example of a graph partitioner in general. Their key intuition was as follows: given two partitions forming halves of a balanced graph bisection $P_2(G) = \{V_1, V_2\}$, there exist subsets of vertices $A \subset V_1$, $B \subset V_2$ which may be swapped between partitions to generate the arrangement which is globally optimal for some objective function (minimum edge-cut). The Kernighan-Lin (KL) operates in iterations, selecting vertex sets to swap which will result in the greatest improvement (which is objective function gain). Within each iteration, the considers every vertex v_i from a partition (say V_1), then calculates the potential gain when swapping v_i with each vertex v_j from partition V_2. The pair (v_i, v_j) with the highest gain is marked for swapping (i.e. v_i, v_j are added to A, B respectively) and the next vertex in V_1 is considered with each vertex in V_2. Note that when considering swapping the neighbors of vertices already marked in this iteration, the potential objective gain for those neighbors is calculated as if marked vertices have already been swapped. When every vertex has been considered, sets A; B are swapped between partitions and the next iteration may begin. Iterations continue until the total gain for all suggested swaps is ≤ 0.

There are two main issues with the KL as a graph partitioner. The first is that it is limited to improving the partitioning quality of graph bisections, rather than k-way partitioning. The second is that it is highly expensive, with a single iteration of the having complexity $O(n^3)$ with $n =| V |$.

3.1.4. Fiduccia-Mattheyses. To address critical issues with the KL , several improvements have been proposed. Perhaps most significantly, Fiduccia and Mattheyses [16] present a modified (KL/FM) which is significantly less expensive. The iteration complexity for KL/FM is $O(m)$ with $m =| E |$ and upper bound for m is $(n - 1)^2$. There are two major differences between the KL/FM and KL s. Firstly, the vertex swapping between bisections is asymmetric, i.e. vertices are marked for transfer individually, rather than as a pair with a vertex from the other partition. This means that for each vertex considered for swapping, it is not necessary to consider every vertex from the other partition. Secondly, Fiduccia and Mattheyses use a data structure called a bucket queue for efficiently updating neighbor objective function gain after marking vertices for swapping. Despite the improvement that the KL/FM represents, it shares KL's original limitation of being applied only to graph bisections. However, there are extensions of local search techniques which generalize for improving k-way partitioning.

3.1.5. Simulated Annealing. The optimization technique, Simulated Annealing (SA) [5] is a local search statistical technique used to get the answer by local arrangements. The description of a solution and neighborhood relation between solutions is important. A cost function assigns a specific cost value to each solution and the cost function should be minimized by changing from one solution to another according to the neighborhood relation. SA iteratively proposes a new solution and evaluates its cost value. The new answer is preserved if the new value is lower than the previous one. Otherwise, a new solution is kept with some probability. This process is repeated till the desired solution can be found or a maximum number of iterations are exceeded.

3.2. Global Graph Partitioners. In general, global graph partitioners refer to those partitioners which, unlike their local counterparts, take an entire unpartitioned graph as input. By default, such partitioners are also executed within the confines of a single machine. They are the most commonly used family of techniques due to their effectiveness. Indeed Karypis and Kumar's Metis [7] is considered the de-facto standard for partitioning quality. However, it is also likely that the popularity of such techniques is at least partially due to their being relatively simple to use and available in several robust software packages.

3.2.1. Spectral Techniques. Donath and Hoffman [4] suggested spectral graph partitioning for computing bisections of a graph G . Firstly, the Laplacian matrix L is computed by subtraction of $G's$ adjacency matrix A from its degree matrix D, $L = D - A$. The second step is to compute the eigenvector associated with $G's$ second smallest eigenvalue. This relies upon the intuition of eigenvector, called the Fiedler vector, which contains an integer value for each vertex, which corresponds to its connectedness in the graph. Using this value as an ordering, the then divides the vertices of G around the median, into two sets of equal size. These sets represent a bisection that is good for minimum edge-cut. Note that, this generalizes to producing k-way partitioning of graphs through recursively bisecting generated partitions. All other spectral partitioning techniques extend this core algorithm. The k-way spectral graph partitioning is achieved in the following way - It finds first k eigenvectors of a Laplacian matrix L and uses k-means algorithm for the final partitioning. i. e. it partitions data set into clusters based on grouping of eigenvectors.

3.2.2. Multilevel Approach. Multilevel technique [15] is effective for effectively partitioning graphs. The tactics for coarsening and local improvement dominate the efficiency of the multilevel method. The vertices' weights are the sum of the vertices' weights in the coarsened graph's. As a result, for both the initial and coarsened graphs, the sum of all vertices' weights is equal. Furthermore, the coarsened graph's partitioning corresponds to the initial graph's partitioning. The edges that were connected to the vertices before coarsening do not appear in the coarsened network since they connect the corresponding vertices. The multi-edges are merged into a single edge which has an edge-weight equal to total weight of all merged edges' weight. The cut-size for partitioning before coarsening and partitioning after coarsening should be the same.

The best example of multilevel techniques is Metis [6]. In the coarsening stage, Metis compresses a graph G by computing maximal edge matching. An edge matching is defined as a set of edges E_M from G, such that no two edges in E_M are incident upon the same vertex. Given such matching, a single level of the compressed graph is computed by combining vertices connected by an edge $e \in E_M$, treating each pair as a single "multi" vertex. Compression is performed using these matching rather than, for example, arbitrarily combining vertices as it prevents anyone "multi" vertex form containing many more elements than another. As a result, a balanced partitioning of the compressed graph will match the original graph's balanced partitioning. Next, when computing an initial partitioning for the compressed graph G_M, Metis uses a technique based on spectral recursive bisection. Finally, in the uncoarsening stage of partitioning, Metis uses the Greedy Refinement local algorithm to move "multi" vertices between partitions, improving the partitioning after each step of match/combine compression has been reversed. Table 3.1 shows the comparative study of prominent algorithms with NWEWBGP.

Table 3.1: Comparison of different graph partitioning techniques

Technique	**Weighted Nodes**	**Weighted Edges**	**Method used**	**Best used for**	**Key points**
KL	No	No	Kernighan Lin algorithm	Bipartitioning	1) Handles only unit edge weights 2) Difficult to extend to k-way graph partitioning 3) Can't partition for varying node and edge weights 4) Need dummy nodes in imbalance situation 5) A pass's time complexity is high, $O(n^3)$ 6) Execution time is longer
FM	No	No	Fiduccia Mattheyses algorithm	Bipartitioning	1) Handles only unit edge weights 2) A pass's time complexity is high, $O(n^3)$ 3) Execution time is longer
MGP	Yes	Yes	Multilevel graph partitioning	k-way partitioning	1) Recursive in nature 2) Edge weighted centric 3) A pass's time complexity is high, $O(n^3)$
Spectral	No	Yes	Laplacian, Eigen Computations	k-way partitioning	1) Can't handle node-weighted graph 2) Time complexity is high, $O(n^3)$ 3) Computationally expensive
Proposed Algorithm	Yes	Yes	Sub-graph computations	k-way partitioning	1) Handles both node and edge weighted graph 2) Flexible, Good quality solutions 3) Time complexity is low, $O(n^2)$

Table 4.1: NWEWBGP Algorithm Conventions

Variable	**Definition**
G	Graph
V	Set of Nodes (Vertices)
E	Set of Edges
ϵ	Maximum ratio
Max_weight	Maximum weight
Min_weight	Minimum weight
k	Number of partitions
w	Sum of weights of all nodes
c	Sum of weights of all edges
Nbr	Neighbor node

4. Node Weighted and Edge Weighted k-way Balanced Graph Partitioning (NWEWBGP).

This work presents the NWEWBGP algorithm (Algorithm 1). The objective is to generate graph partitioning with approximately balanced node weights. The other objective function is to minimize cut-size (edge-cut). Table 4.1 shows the conventions used for the NWEWBGP algorithm.

4.1. Problem Formulation. Let undirected, connected graph $G = (V, E)$ has non-negative weights on nodes w_v, non-negative weights on edges w_e and partitioning criteria, $k >= 2$. A balance requirement necessitates that all blocks be around the same size. In this work, it's worth mentioning that obtaining a uniform balanced graph division isn't always attainable. As a result, the ϵ option to evaluate a partition's balancing factor is introduced. The term completely balanced will be achieved if $\epsilon = 0$. In our work, this factor varies from 0 to 1 with step of 0.1. The step is incremented by 0.1 if the following requirement will not fulfill.

The $(k, 1 + \epsilon)$-balanced partitioning is the problem of solving G into k partitions such that:

1. Maximum weight of $(1 + \epsilon) * w/k$ in each part
2. Minimum weight of $w/(1 + \epsilon) * k$ in each part

4.2. NWEWBGP Algorithm Pseudocode. A summary of the findings of our NWEWBGP research is presented in this section. Various graph partitioning cases are evaluated and compared with the Metis, a standard benchmark tool. We assess the proposed algorithm's performance on the weighted graphs.

The number of partitions needed, k and ratio factor ϵ is taken as an input. The minimum and maximum weights are calculated as shown in step 1 of Algorithm 1. i.e. Node weight of each subgraph may differ from the average by no more than a factor of max_ratio. In step 2 of the algorithm, all partitions of the graph into subgraphs within weight limits are enumerated. It will return the list of partitions, each partition a list of subgraphs, each subgraph a list of nodes. Firstly it finds the node with the highest weight. It will find all subgraphs containing the heaviest node, within weight limits. For subgraph in subgraphs, check if the subgraph splits the graph into disconnected parts. If so, check parts for min_weight and discard subgraph if any are underweight. While doing so, the remainder graph is checked, if it is empty, then add a 1-part partition(subgraph) otherwise add subgraph to each subpartition.

4.3. Experimental Results and Discussion. The proposed algorithm was implemented in python. The Metis library of python (pmetis) was also used for comparison and validation of the proposed algorithm. This work was carried out on the system having 24GB RAM and P100 NVIDIA's GPGPU (3600 CUDA Cores). Consider the graph examples shown in Figure 4.1 and respective results are shown in Table 4.2. The input graphs used in this study have been generated using a graph generator function. The number of nodes, number of edges, weights on the nodes/edges are randomly generated to test the performance of the heuristics under different conditions. The input was a graph with node-weights, edge-weights, and partitioning criteria. In all the graph examples shown in Figure 4.1 , the values present in the circle are node-weight and the values present outside the circle are node indices. The values at every edge represent edge-weight. The partitioning criteria allows the user to set the number of approximately balanced sub-graphs to be made.

Algorithm 1: NWEWBGP Algorithm Pseudocode

```
Step 1: Estimate the minimum and maximum weights for partitioning graph into $k$ number of parts
        Based on $\epsilon$ ranging from 0 to 1 with step of 0.1, the maximum ratio (threshold) of maximum
    weight to the minimum weight is considered.
        A partitioning of $G$ into $k$ sections at a low cost, such that
            $Min\_weight \leq \sum v \in V_i \quad w(V)/k \leq Max\_weight$
            $Min\_weight = (w/(k * (1+\epsilon)))$
            $Max\_weight = (1+\epsilon)(w/k)$
Step 2: Enumerate all partitions of graph into sub-graphs within weight limits
    While (Remainder Graph $!= 0$)
        Step a): Find a node with highest weight, $\{v_i \mid v_i \in V \ \& \ v_i^w$ is maximum$\}$
        Step b): Find all sub-graphs containing heaviest node within weight limits,
                 $\{G' \mid G' \in G \ \& \ min\_wt \leq G' \leq max\_wt\}$
                 Check if sub-graph splits the graph into disconnected parts?
                 If so, check parts for $Min\_weight$,
                 Discard sub-graph if any are underweight.
Step 3: Find all sub-graphs of graph which contain sub-graph, within limits.
        Sub-graphs must have weight (sum of node weights) within limits. Ignore any nodes in ignore.
        Find all neighbors of sub-graph,excluding nodes in ignore.Use neighbors of the forward node.
        Try adding each neighbor node to sub-graph
        if($G'(w) > max\_wt$) ignore neighbor (nbr too heavy to add to subgraph, skip it later)
        else, New Subgraph, $G'' = G'(w) + Nbgr(w)$
Step 4: Min-cut function: $W(G_1, G_2) = \sum_{x_i \in G_1, x_j \in G_2} w_{ij}$ and $\bar{G}_1$ as the complement of $G_1$.
        $cut(G_1, \ldots, G_k) = \frac{1}{2} \sum_{i=1}^{k} W(G_i, \bar{G}_i)$.
```

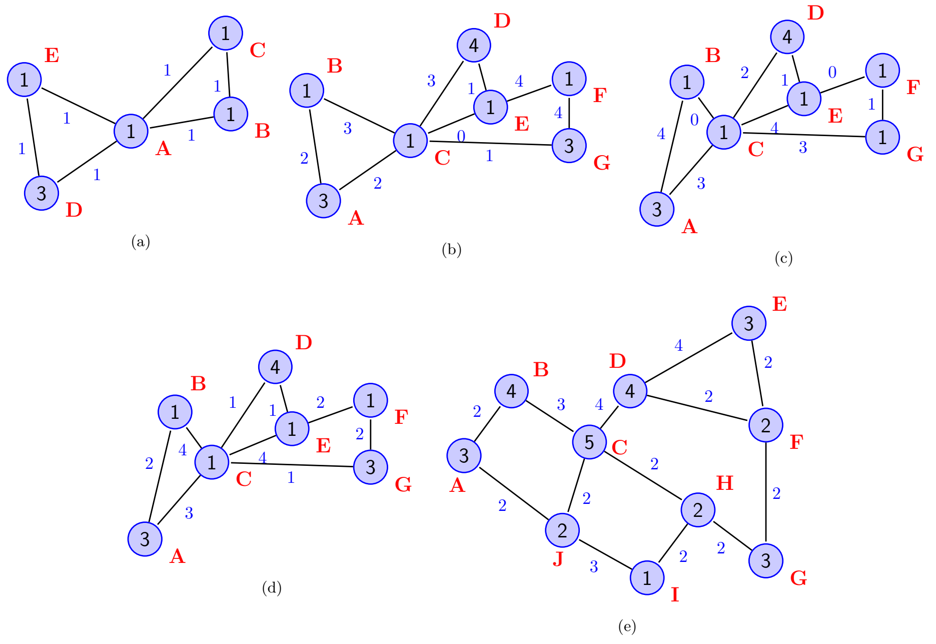

Fig. 4.1: Graph Examples for Experimentation

Table 4.2: NWEWBGP Results and Comparison with Metis

Fig.2	**Max. Ratio**	**Max. Weight**	**Min. Weight**	**No. of Parts**	**NWEWBGP Result**		**Metis Result**	
					Partitions	**Cut-cost**	**Partitions**	**Cut-cost**
(a)	1.2	4.2	2.9	2	[D,E], [A,B,C]	2	[A,B,C,D,E]	0
(b)	1.2	8.4	5.83	2	[A,B,C,G], [D,E,F]	7	[A,B,C,D,E,F,G]	0
(c)	1.4	5.6	2.85	3	[A,B], [C,D], [E,F,G]	11	[], [D,F,G], [A,B,C,E]	6
(d)	1.2	5.6	3.88	3	[A,B], [C,D], [E,F,G]	13	[], [D,G], [A,B,C,E,F]	5
(e)	1.1	15.95	13.18	2	[A,B,C,I,J], [D,E,F,G,H]	8	[A,B,C,I,J], [D,E,F,G,H]	8
(e)	1.2	11.6	8.05	3	[A,G,H,I,J], [B,C], [D,E,F]	12	[A,G,I,J], [B,C,H], [D,E,F]	14
(e)	1.3	9.425	5.58	4	[A,I,J], [D,E], [F,G,H], [B,C]	16	[A,I,J], [D,E], [F,G,H], [B,C]	16

Table 4.3: Selection of balanced factor on graph (e) of Figure 4.1

Balanced Factor (ϵ)	**No. of Parts**	**Max. Wt.**	**Min. Wt.**	**Sub-graphs formed**	**Partitions**	**Cut-cost**
0.1	2	15.95	13.18	14	[3,4,5,6,7], [0,1,2,8,9]	8
0.2	2	17.4	12.08	27	[0,4,5,6,7,8,9], [1,2,3]	12
0.3	2	18.85	11.15	61	[3,4,5,6,7,8,9], [0,1,2]	10
0.1	3	10.63	8.78	4	-	-
0.2	3	11.6	8.05	5	[1,2], [3,4,5], [0,6,7,8,9]	12
0.3	3	12.56	7.43	15	[0,1,2], [3,4,5], [6,7,8,9]	12
0.4	3	13.53	6.9	38	[1,2], [3,4], [0,5,6,7,8,9]	14

The output provides a summary of the graph partitions which includes the total weight of each partition and the cut-cost on the set of examples. 3 levels of partitions were tested in this work, particularly $k = 2, 3, 4$. The results further compare and evaluate the performance of Metis and NWEWBGP.

As explained in Algorithm 1, based on the balanced factor, maximum weight and minimum weight is calculated. The results show that choosing the best-balanced factor helps to divide the graph into more appropriate balanced partitioning. Based on the number of partitions (k) to be made, the proposed algorithm shows an approximate balanced partitioning with cut-cost. For the same examples, the Metis results are also calculated. The NWEWBGP either gives the results same as Metis or improves in some cases. In few of the scenarios, Metis doesn't give the partitions as expected while the NWEWBGP works well by providing an appropriate balance of both node-weighted balance constraint and minimum edge-cut constraint. The NWEWBGP running time is $(O(n)^2)$, which is better than Metis running time of $(O(n)^3)$.

4.4. Selection of appropriate balanced ratio. The Algorithm 1 presented above is able to compute the partitions of high quality in a reasonable amount of time with balanced ratio (balanced factor), ϵ close to 0. The Table 4.3 shows the 2-partitions and 3-partitions obtained for various values of ϵ on the graph example (e) of Figure 4.1. It is seen that, as we increase the value of ϵ, the number of sub-graphs computed will also increases and the quality of partitions will decreases (Outcome: imbalanced partitions). So, selection of ϵ is more important for graph partitioning problem. From the Table 4.3, in case of $(2, 1 + \epsilon)$-balanced partitioning, the best balanced factor is 0.1. In case of $(3, 1 + \epsilon)$-balanced partitioning, the best balanced factor is 0.2.

4.5. Output Balance Metric. Different partitioning tools provide different output metrics. For instance, the output metrics provided by some graph partitioning are the set size, edge cuts etc. The output metrics provided by our approach are 'Balance' and 'Cut edges'. 'Cut edges' or 'Cut-cost' is similar to the edge cuts as measured by Metis. 'Largest' is the total weight of nodes present in the largest set and similarly 'Smallest' is the total weight in the smallest set. The metric 'Balance' is important for understanding the load balance obtained from the partition. In this work, 'Balance' metric is calculated using the following formula.

$$Balance = S_{max}/S_{opt}$$

where S_{max} is the weight of the largest subgraph and S_{min} is the optimum subgraph (ideal) size given by:

$$S_{opt} = G/k$$

where G is the total weight of nodes in the graph and k is the number of partitions. Table 4.4 shows 'Balance' and 'Cut-cost' metric comparison between proposed work and Metis. The Cut-cost comparison is also shown.

The Figures 4.2 and 4.3 shows the comparison for the 'Balance' metric and 'Cut-cost' metric, respectively. It is seen that, the proposed work on NWEWBGP gives better results than the Metis.

Table 4.4: Output balance metric and cut-cost

Sr. No.	**No. of Nodes**	**No. of Edges**	**Parts**	**Proposed method**		**Metis result**	
				Balance Metric	**Cut-cost**	**Balance Metric**	**Cut-cost**
1	2000	2000	3	1.23	341	1.44	366
2			4	1.14	447	1.39	482
3	4000	4500	3	1.76	589	1.97	634
4			4	1.23	665	1.51	738

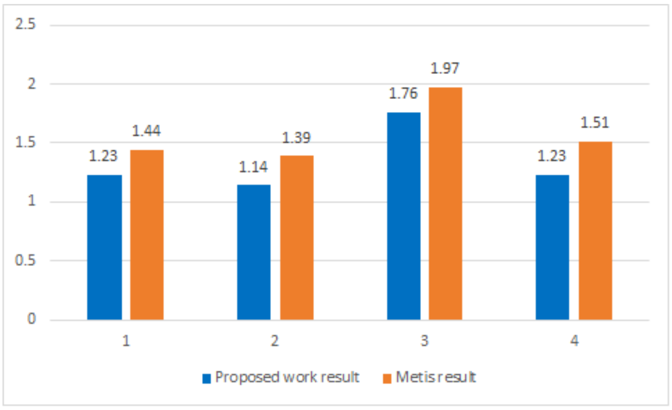

Fig. 4.2: Output Balance Metric

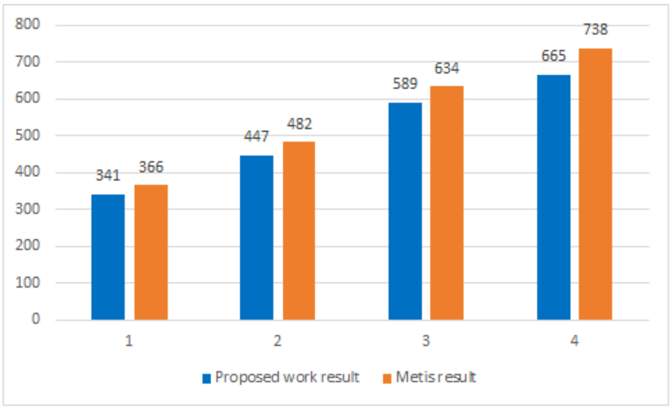

Fig. 4.3: Cut-cost

5. Conclusion. In this paper, the existing graph partitioning techniques are studied and the necessity of proposed graph partitioning in the context of parallel computing is discussed. The graph partitioning problem can be defined with several objective functions, however, the objective discussed in this study emphasizes the need in parallel computing domain - the balanced partitioning with minimum cut-cost.

The proposed node-weighted and edge-weighted k-way balanced graph partitioning algorithm shows an approximately balanced partitioning and can aid in achieving better utilization of processors in parallel computing. The results demonstrate that the algorithm succeeds in effectively balancing the node weights across partitions and minimizing the cut-cost (weighted edge-cut) as compared to the results of Metis, a benchmark tool for graph partitioning. The proposed NWEWBGP method also outperforms the existing state-of-the-art algorithms in terms of running time.

REFERENCES

[1] K. Andreev and H. Racke, *Balanced graph partitioning*, Theory of Computing Systems, 39 (2006), pp. 929–939.

[2] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, *Recent advances in graph partitioning*, Algorithm engineering, (2016), pp. 117–158.

[3] J. Doe, *Load balancing strategies in parallel computing: Short survey*, (2015).

[4] W. E. Donath and A. J. Hoffman, *Lower bounds for the partitioning of graphs*, in Selected Papers Of Alan J Hoffman: With Commentary, World Scientific, 2003, pp. 437–442.

[5] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, *Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning*, Operations research, 37 (1989), pp. 865–892.

[6] G. Karypis and V. Kumar, *Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*, (1997).

[7] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on scientific Computing, 20 (1998), pp. 359–392.

[8] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell system technical journal, 49 (1970), pp. 291–307.

[9] M. Kushwaha and S. Gupta, *Various schemes of load balancing in distributed systems-a review*, International Journal of Scientific Research Engineering & Technology (IJSRET), 4 (2015), pp. 741–748.

[10] M. Leng, S. Yu, and Y. Chen, *An effective refinement algorithm based on multilevel paradigm for graph bipartitioning*, in International Conference on Programming Languages for Manufacturing, Springer, 2006, pp. 294–303.

[11] R. Peng, H. Sun, and L. Zanetti, *Partitioning well-clustered graphs: Spectral clustering works!*, in Conference on Learning Theory, PMLR, 2015, pp. 1423–1455.

[12] V. Prasad, *Load balancing and scheduling of tasks in parallel processing environment*, Int. J. Inf. Comput. Technol, 4 (2014), pp. 1727–1732.

[13] M. Predari and A. Esnard, *A k-way greedy graph partitioning with initial fixed vertices for parallel applications*, in 2016 24^{th} Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), IEEE, 2016, pp. 280–287.

[14] C. Sakouhi, A. Khaldi, and H. B. Ghezal, *An overview of recent graph partitioning algorithms*, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), The Steering Committee of The World Congress in Computer Science, Computer ..., 2018, pp. 408–414.

[15] P. Sanders and C. Schulz, *Engineering multilevel graph partitioning algorithms*, in European Symposium on Algorithms, Springer, 2011, pp. 469–480.

[16] M. Sheblaev and A. Sheblaeva, *A method of improving initial partition of fiduccia–mattheyses algorithm*, Lobachevskii Journal of Mathematics, 39 (2018), pp. 1270–1276.

[17] I. Stanton and G. Kliot, *Streaming graph partitioning for large distributed graphs*, in Proceedings of the 18^{th} ACM SIGKDD international conference on Knowledge discovery and data mining, 2012, pp. 1222–1230.

[18] J. Sun, H. Vandierendonck, and D. S. Nikolopoulos, *Graphgrind: Addressing load imbalance of graph partitioning*, in Proceedings of the International Conference on Supercomputing, 2017, pp. 1–10.