The Graph Partitioning Algorithms: A Comparative Analysis of Local Heuristic Methods

Ziad Eliwa 900246124 - Mostafa Abdelwahed 900243064
Ali Mohamed 900246190 - Farouk Youssef 900243941

***Abstract*—Graph partitioning is a fundamental NP-hard problem with critical applications in parallel computing, high-performance computing, load balancing, and VLSI design. As the emergence of Big Data intensifies the demand for efficient graph processing systems, the need for high-quality partitioning approaches becomes increasingly important. This paper presents a comparative analysis of two prominent local heuristic methods for the graph bipartitioning problem: the Kernighan-Lin (KL) algorithm and the Fiduccia-Mattheyses (FM) algorithm. We examine the theoretical foundations, algorithmic designs, and computational complexities of these classical approaches. The KL algorithm, performs iterative pairwise swaps to refine an initial partition, while the FM algorithm achieves linear-time performance through single-vertex moves and bucket-based data structures. Through experimental evaluation on benchmark datasets, we assess the trade-offs between solution quality, computational efficiency, and scalability of these foundational algorithms. Our findings provide insights into the practical applicability of local-search heuristics and their evolution from exhaustive combinatorial methods to computationally feasible approaches for graph partitioning.**

I. Introduction

Abstraction is one of the most important ideas of computer science. Computer scientists and researcher across the span of the study and modelling of real-world complex entities have utilized modelling them as *graphs*. In each case, there is an underlying network that encapsulate the important properties that help us understand these entities and draw meaningful insights from them. In addition to the huge contribution that graphs brought to the advancment in computer science, it brought with its complexities. Due to the enormous data graphs can represent, they often fall in the area of *NP Hard Problems*, starting from the most famous Travelling Salesman Problem (TSP) to graph coloring (k-coloring for k $\geq$ 3), Hamiltonian paths and more. Graph Partitioning stands as one of the most challenging problems in computer science. Graph partitioning is essential for the fields of parallel computing, high performance computing (HPC) and load balancing in networks, and VLSI in digital logic design where vertices resemble logical units and edges resemble wires. This paper goes into depth of some heurisitcs and greedy algorithms that allow local use of the power of graph partitioning. [1]

"The graph partitioning problem is increasing with the emergence of Big Data. Handling tremendous volumes of graph data requires an efficient graph processing system and especially a high-quality graph partitioning approachs to cope with graph application needs." [2].

II. Background

A. Theoritical Background

***Graph**:* Let graph $G = (V, E)$ be weighted, undirected graph with vertices $v_i = 1, 2, \ldots n$. Define w_{ij} as the non-negative weight between v_i and v_j, $i \neq j$. If $(v_i, v_j) \in E$, then $w_{ij} > 0$, otherwise $w_{ij} = 0$.

***Graph partition / k-partition**:*

***Cut**:* A cut C is a partition of V which seperate into S and $V - S$. A cut-set of $C = (S, V - S)$ is defined as the set of edges $\{(u, v) | u \in S, v \in V - S\}$.

***External and Internal Cost**:* In a cut C of V in $G = (V, E)$, the external cost of the vertex a is the number of the edges that cross C and the internal cost of the vertex a is the sum of the edges that do not cross C.

***P vs NP**:* The P vs NP Problem is one of the mysteries of theoritical computer science and one of the seven millenium problems. P is the complexity class of problems that are solvable in polynomial time through deterministic algorithms. NP (Non-deterministic polynomial time) is a complexity class of problems that have no deterministic algorithms to solve it in polynomial time (Unless P = NP), while there exists exponential time algorithm through brute force and complete search methods, but these are not solvable by today's machines, unless perhaps solved with novel ideas such as quantum computation.

***NP-Hard Problems**:* In the class of NP Problems, some problems are called as *NP-Complete*. All NP-Complete problems are decision problems (i.e. with Yes or No answer) such as the existence of a hamiltonian path. Problems are called *NP-Hard* if they are at least as hard as NP-Complete problems. NP-Hard can be present in its own complexity class in which NP-Complete problems is a subset of it, but it remains undecidable.

The graph partition problem is considered NP-Hard as it can be reduced from Max-Cut problem which itself is NP-Hard. This paper discusses the approximation and heurisitcs that was motivated by the computational complexity of the problem.

***Heuristics**:* Heuristic methods are greedy and approximate algorithms that are concerned to use the available resources to reach an enough solution to the problem. Exact algorithms allow for optimal solutions but are highly limited to small graphs.

B. Problem Definition

A graph partition or a k-partition of graph $G = (V, E)$ is partitioning V into k disjoint parts that cover V.

$$V_1 \cup V_2 \cup \cdots \cup V_k = V.$$

For a partition to be balanced, we have to satisfy the Balance constraint $|V_i| \in \{\lfloor n/k \rfloor, \lceil n/k \rceil\}, |V| = n$ and the number of edges with endpoints in different parts is minimized.

The optimization problem is to find such a partition:

$$\min_{V_1,\ldots,V_k} \text{cut}(V_1, \ldots, V_k) \quad \text{subject to the balance constraint.}$$

III. Literature Survey

Throughout the past decades, several methods have been developed to solve the problem of constrained graph partitioning. Methods like Multilevel, Greedy, Classical Heuristic, and Spectral are the most common methods [3]. The multilevel method is a class of algorithms that compile in moderate time but give excellent graph partitioning [4]. This method is constructed of three phases; the first phase is called the coarsening phase. In this phase, a set of smaller graphs $G_i = (V_i, E_i)$ are produced from the original graph $G_0 = (V_0, E_0)$. The smaller graphs are often produced by combining the vertices of G_i into a single vertix that is then merged into the successive graph G_{i_+1}. This phase is often implemented by a variety of algorithms such as heavy match, light match, and random match [5]. The heavy-matching technique prioritizes grouping the vertices with the edges that have the highest weight. On the other hand, Light-matching groups the vertices with edges that have the lightest weight together. Random matching, as the name suggests, groups the vertices randomly. The second phase, which is the initial partitioning phase, is where a balanced bisection is computed on the coarsest graph G_k, which is the smallest graph constructed during the coarsening phase. The last phase is the uncoarsening phase. During this phase, we recursively move back from the coarsest graph to the original graph [4].

$$G_k \to G_{k-1} \to G_{k-2} \to \cdots \to G_1 \to G_0$$

There exists a special type of algorithm related to the multilevel method called Louvain. The main concept of the Louvain algorithm is to maximize the modularity of a community partition. This can be achieved by initializing each vertix as a separate community and then moving the vertices to other communities, which results in the modularity reaching its maximum state. This step occurs iteratively until all vertices are merged into a community. The iterative step helps obtain the multilevel community partition with the highest modularity [6]. The multilevel method is characterized by its exceptional speed and high-quality graph partitioning; however, it can be complex to design a multilevel method as it requires careful design of the three mentioned phases. Another popular method that is used to solve the graph partitioning problem is the classical heuristic. Classical heuristic methods are approximation algorithms that tend to find the best possible solution rather than the exact solution [5]. Among the known algorithms that utilise the heuristic method are Kernighan-Lin and Fiduccia-Mattheyses. The KL and FM algorithms proceed by starting with a certain partition and moving on from there. On the other hand, greedy methods start the algorithm with an empty partition and fill the partition vertix by vertix. It builds the partition by making decisions that are the best possible choice at the point. Among the algorithms that implement this method are Standard Greedy, Min-Max Greedy, Diff Greedy, and K-Greedy [5]. For the K-Greedy algorithm, the graph is split into k-parts rather than splitting into two parts.

IV. Experiment

A. Methodology

We performed the experiment on a computer with Intel Core i7-13700H CPU, 14C (6P + 8E)/20T, 3.7 GHz, 16 GB of memory DDR5-5200 on a Linux operating system with Ubuntu 24.04.3 LTS. The implementation was done in Python 3.12.3 and you can find the source code in the following repository: github.com/ziad-eliwa/CSCE2211-Paper.

The experiment data was originated from the following github repository: mirkat1206/Fiduccia-Mattheyses. The data was directly processed with the algorithms without any cleaning, manipulation or preprocessing. [1]

Due to limits in computational power and human resources for implementation, we considered three algorithms for the comparative analysis: Kernighan–Lin algorithm and Fiduccia–Mattheyses algorithm. We also limit the problem into only bipartitioning (k = 2), so it can adapt with the nature of the design both algorithms.

Kernighan–Lin algorithm

Kernighan–Lin (KL) algorithm was one of the first efficient problems designed to tackle the graph partitioning problem. It marked the shift from exhaustive combinatorial methods to efficient heurisitcs for relatively large graphs. KL's significance lies in its balance between solution quality and computational feasibility, making it a foundational algorithm in graph partitioning heuristics. [9]

Its idea lies in the refinement of a previosly obtained 2-partition. Therefore, it is considered a local optimization algorithm. The algorithm is to find to two partitions of B such that the cut gain is minimized as possible. It performs swaps between subsets of the partitions until no more cut reduction can be obtained. Variants were later developed for multi-way partitioning and integration with other heuristics like Fiduccia–Mattheyses, which optimized for faster updates and weighted graphs.

We define the following functions:

The difference between the external cost and internal cost of the vertex v as: $D(v) = E(v) - I(v)$.

The reduction cost for swapping a and b as: $\Delta g_k = D(a) + D(b) - 2c(a, b)$. Where $c(a, b)$ is the number of edges between a and b and a and b are vertices.

[1] Distributed under the MIT License, allowing unlimited free use, modification, and redistribution.

Algorithm 1:

```
step 0:
    V = 2n vertices
    V = {A,B} is an initial
    2-partition of V
step 1:
    k = 1
    compute D(v) for all vertices v in V
    where D(v) = E(v) - I(v)
step 2:
    - choose a and b such that g_k = D(a)
     + D(b) - 2c(a,b) is maximized.
    - Swap and mark a and b as fixed.
step 3:
    - if all vertices are fixed, go
    to step 4.
    Otherwise
        - Compute and update D values
        for all vertices connected to
        a and b, which are not fixed.
        - k = k + 1
        - go to step 2
step 4:
    - Find the move sequence 1 ... m
    (1 <= m <= k) such that
    G_m = \sum_{k = 0}^{m} g_k
    is maximized.
    If G_m > 0
        go to step 5
    Otherwise,
        End
step 5:
    - Execute m swaps, reset
    remaining vertices
    - Go to step 1.
```

***Complexity Analysis**:* The complexity of the KL algorithm is determined by two major factors: Gain Updates and Pair Selection.

Gain updates are calculated for unfixed vertices and for every iteration we decrease the number of unfixed vertices by 2. Therefore the number of iterations of gain updates,

$$T(n) = \sum_{k=0}^{n} (2n - 2k) = O(n^2).$$

Pair selections are calculated for unfixed vertices and a reduction of 2 happens after swapping and fixing and comparing pairs of the unfixed vertices. Therefore number of comparisons of pairs,

$$T(n) = \sum_{k=0}^{n} (n - i)^2 = O(n^3)$$

Therefore, the complexity of the KL algorithm is $O(n^3)$. Further optimizations can happen to reduce to $O(n^2 \log n)$.

The algorithm was adapted from IIT Roorkee Lectures by Professor Bishnu Prasad Das on VLSI Physical Design with Timing Analysis [10].

Fiduccia–Mattheyses algorithm

Fiduccia–Mattheyses(FM) algorithm is an iterative graph partitioning heuristic algorithm that is based on insights from KL algorithm. The most significant part of FM is that the worst case grows linearly with the size of the graph and with small number of passes ($n <= 10$) over the graph, we can obtain a significantly more optimal results than KL algorithm. The General FM algorithm also focuses on the bipartitioning case, however, it elevates the use of efficient searching and updating using bucket-based data structures, which eliminates unnecessary checks which significantly improve performance with sacrificing storage, making it unfeasible for very large graphs [8]. The FM algorithm allows for different partition sizes and can be extended to hypergraphs.

While KL algorithm utilizes the swaps between subsets of the initial partitions, the FM algorithm moves only a single vertex at a time between the two partitions to maximize the gain and decrease the cut as much as possible. In each pass, we start from a random of two partitions. We keep moving the highest vertices with the highest potential gain to the other partition with updating of neighbours gains at each movement. After each vertex's movement, it gets locked and removed from the buckets with storing the cut size for each movement. After the pass finished, the algorithm rolls back to the minimal cut size and proceed to other pass till it converges and no full pass achieves a positive result. It is enough for most engineering uses [8].

Algorithm 2:

```
step 0:
    -V = 2n vertices
    -V = {A,B} is an initial
    partition of V
    -Calculate gains for each vertex
    in each partition and store it in
    a bucket list.
for 1 to num_passes = 10
        -Select the maximum gain node
        -move it to the other partition
        -recompute the neighbours gains
        -lock the moved vertex and remove
        it from the bucket list.
        -save the cut size for each
        movement
        -repeat till all the vertices
        are locked
    -roll back till you find the minimal
    cut size
    -if maximum gain > 0,
        -add the vertices back to the bucket list
        -continue
    otherwise, END
```

***Complexity Analysis**:* The complexity of the FM algorithm can be determined through tracking the number of operations in each pass:

Node Selection: $O(1)$ Amortized, due to utilizing bucket data

structures.

Neighbour updates: $O(E)$.

So, the overall complexity of the FM algorithm is $O(P \cdot E)$, where P is the number of passes until no gain improvement.

B. *Experiment*

Metrics:

V. Results & Discussion

VI. Conclusion & Future Work

References

[1] Graph partition problem - an overview — sciencedirect topics. (2025). https://www.sciencedirect.com/topics/computer-science/graph-partition-problem

[2] Sakouhi, C., Khaldi, A. & Ghezala, H. Distributed framework for high-quality graph partitioning. J Supercomput 81, 1418 (2025). https://doi-org.libproxy.aucegypt.edu/10.1007/s11227-025-07907-2

[3] Bichot, C. E., & Siarry, P. (Eds.). (2013). Graph partitioning. John Wiley & Sons.

[4] Karypis, G., & Kumar, V. (1995, December). Analysis of multilevel graph partitioning. In Proceedings of the 1995 ACM/IEEE conference on Supercomputing (pp. 29-es).

[5] Siqueira, R.M.S., Alves, A.D., Carpinteiro, O.A.O., Moreira, E.M. (2024). Graph Partitioning Algorithms: A Comparative Study. In: Latifi, S. (eds) ITNG 2024: 21st International Conference on Information Technology-New Generations. ITNG 2024. Advances in Intelligent Systems and Computing, vol 1456. Springer, Cham. https://doi.org/10.1007/978-3-031-56599-165

[6] An improved louvain algorithm for community detection. (2021). Mathematical Problems in Engineering, https://doi.org/10.1155/2021/1485592

[7] V.D. Blondel, J.L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks. J. Stat. Mech. Theory Experiment 2008(10), P10008 (2008)

[8] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in Design Automation, 1982. 19th Conference on (IEEE, 1982), pp. 175–181.

[9] Bichot, C.-E., & Siarry, P. (2013a). 2.5.2 The Kernighan–Lin Algorithm. In Graph Partitioning (pp. 22–26). essay, Wiley.

[10] Das, B. P. (2024, February 9). Lecture 22: Kernighan – Lin (KL) Algorithm. YouTube. https://youtu.be/95OB005rdtA