CMPE 492

Partitioning Undirected Graphs

Aral Dörtoğul

Advisor:

Can Özturan

November 19, 2023

TABLE OF CONTENTS

1. INTRODUCTION

Graph partitioning, a fundamental problem in computer science, involves dividing a graph into smaller, more manageable components. While heuristic algorithms have historically been employed to tackle this challenge, the aim of this research is to go beyond approximations. By leveraging modern computing capabilities, this study seeks to develop an exact algorithm for solving the NP-hard graph partitioning problem, even for smaller graphs that were once considered computationally intensive.

To address the NP-hard nature of graph partitioning, an integer linear programming approach is adopted in this research. This involves formulating the partitioning problem as a mathematical model, which is then solved using the Gurobi optimization engine in conjunction with C++ programming. This theoretical framework not only enhances the understanding of the problem but also facilitates the translation of mathematical precision into practical, computer-executable solutions.

1.1. Broad Impact

Undirected graph partitioning, while posing a computational challenge, holds immense potential for influencing various domains. This research can extend to applications such as parallel processing, image processing, and VLSI design. In the realm of parallel processing, efficient graph partitioning can significantly enhance the distribution of computing tasks across multiple processors, leading to improved performance and speed. Likewise, in image processing, the ability to accurately partition graphs can streamline the analysis of complex visual data. Moreover, in VLSI design, optimized graph partitioning contributes to the creation of more efficient and compact electronic circuits. As computers continue to play an essential role in diverse fields, the impact of exact graph partitioning solutions becomes increasingly pronounced.

1.2. Ethical Considerations

The pursuit of advanced computational solutions brings with it ethical responsibilities. In the context of graph partitioning, ethical considerations revolve around reliability, fairness, transparency, and accountability in the use of partitioning algorithms. As these algorithms influence decision-making processes and resource allocations, it is crucial to ensure that their deployment is unbiased and just. This research is committed to navigating these ethical dimensions by prioritizing responsible algorithmic deployment and promoting transparency in its application.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

As memory become cheaper and the processing power of CPUs increase with technological advancements, the complexity of the problems solved by computers also increase. The classical methods of solving problems with a single threaded fashion is sometimes ineffective in large problems. For such cases, it is much more convenient to divide the data into several partitions that have equal load so that algorithms on data can be operated in a parallel fashion by multiple CPUs. Of course, parallelism brings the problem of interprocessor communication, which is time consuming. In order to overcome this, it is crucial to divide the data in a way that the communication between processors is minimized. For this context, graphs are a great abstraction to model this problem.

Graph theory is a branch of mathematics that focuses on the examination of graphs—mathematical structures composed of "vertices" or nodes connected by "edges" or arcs, as depicted in Figure 2.1. This field finds widespread applications in diverse domains, including computer science, engineering, physics, social sciences, and operations research.

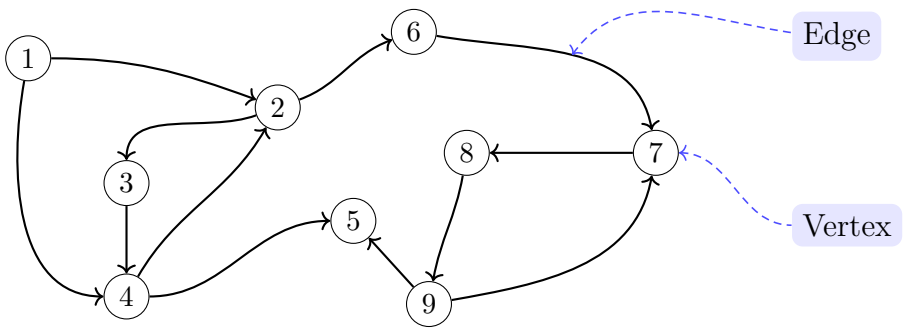

Figure 2.1. A directed graph

Graph partitioning algorithms partition a graph into equal sized subgraphs (in

terms of nodes) in such a way that the number of edges cut is minimized. For example, Figure 2.2 illustrates different ways of partitioning a graph, but not all alternatives provide an optimal solution.

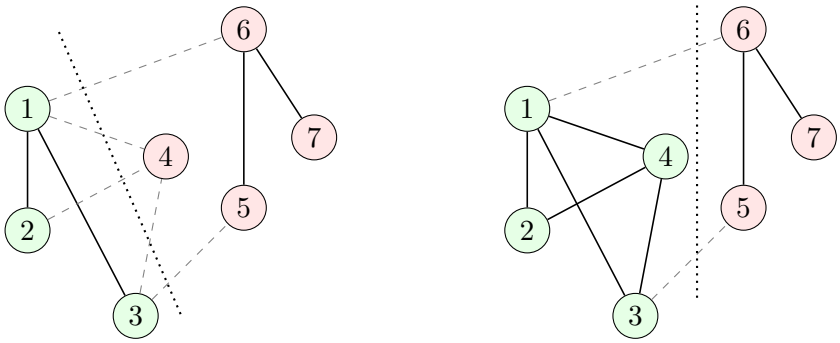

Figure 2.2. Different partitions of a graph

In the more general problem, edges can also have weights. Moreover, since this problem is a minimization problem, it can be modeled as an integer programming model, which is used commonly in operations research.

2.2. Project Planning

The following will be done in this study:

(i) Construction of an integer programming model for the weightless, undirected graph partitioning problem.
(ii) Development of an algorithm that solves the model using a integer programming solver. (Gurobi and C++ will be used.)
(iii) Construction of another model for the weighted version of the problem.
(iv) Development of an algorithm for the weighted version of the problem.
(v) Evaluation of the software's performance in both cases (weighted and weightless) on a range of graph sizes and complexities.

The weightless version of the problem is handled until this report, and the

weighted version of the problem will be handled until the poster session and the final report.

2.2.1. Project Time and Resource Estimation

The gantt chart for the project plan is in Figure 2.3.

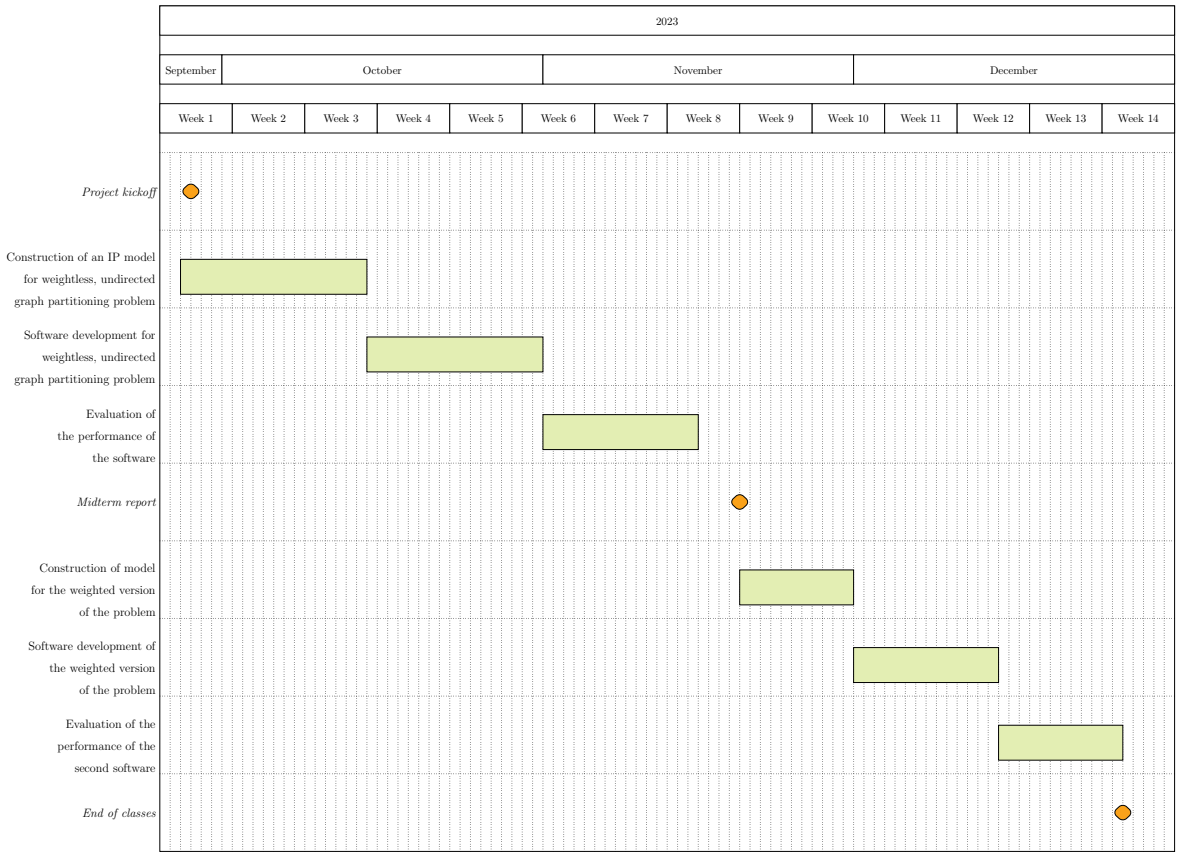

Figure 2.3. Project Plan

2.2.2. Success Criteria

The study will be considered to be successful if weightless and weighted graphs can be partitioned by software.

2.2.3. Risk Analysis

(i) Technical Risks

(a) Algorithm Complexity

Risk The complexity of the algorithm amy result in longer computation times, especially for larger graphs.

Impact Delays in delivering results and potential resource strain.

Likelihood High

Mitigation Implement algorithm optimizations, explore parallel computing options, and conduct thorough testing on a range of graph sizes and densities.

(b) Model Formulation Challenges

Risk Difficulty in accurately formulating the graph partitioning problem as an IP model.

Impact Incorrect solutions or failure to converge to an optimal solution.

Likelihood Low

Mitigation Perform rigorous testing on representative graph instances, conduct a comprehensive literature review.

(ii) Operational Risks

(a) Resource Constraints

Risk Inadequate computational resources for executing the algorithm efficiently.

Impact Slower execution and potential limitations on the size of the graphs that can be processed.

Likelihood Low to Moderate

Mitigation Assess and secure sufficient computational resources, optimize code for resource efficiency, and maybe explore cloud computing options.

(b) Data Security Concerns

Risk Potential vulnerabilities in handling sensitive graph data.

Impact Data breaches and/or unauthorized access.

Likelihood Low

Mitigation Implement secure data handling practices, use encryption where necessary.

3. RELATED WORK

The study of graph partitioning has a long history and can be traced back to various researchers and fields. While it's challenging to pinpoint a single origin, the problem has been explored by mathematicians, computer scientists, and researchers in related disciplines for many decades.

One early mention of graph partitioning can be found in the field of spectral graph theory. In the 1970s, researchers like F. R. K. Chung [1] began investigating the relationship between graph properties and eigenvalues of certain matrices associated with graphs. This work laid the foundation for spectral graph partitioning algorithms.

In the context of computer science and algorithms, the graph partitioning problem gained attention for parallel computing applications. During the 1980s and 1990s, researchers explored methods to efficiently partition graphs for parallel processing to optimize computational tasks. The proven NP-hard nature of the problem [2] led researchers to develop heuristics as well as exact algorithms.

3.1. Exact Algorithms

A lot of research has been done on ways to solve the Generalized Partitioning Problem (GPP). Some methods are made specifically the bipartitioning case, while others work for the general GPP. Most of these methods use a common approach called the branch and bound framework.

Researchers use different strategies to set limits on the solutions. According to Buluç [3], some use a method called semi-definite programming, like Karisch [4], or Armbruster [5]. Sellman and Sensen [6, 7] use something called multi-commodity flows. Linear programming is another technique used by Brunetta [8], Ferreira [9], Lisser [10], and Armbruster [5], and others. Hager and others [11, 12] take a continuous quadratic

approach and then use the branch-and-bound method on it.

3.2. Heuristics

There are several heuristics proposed to solve graph partitioning problem. Buluç [3] lists many heuristics, including:

- Spectral Partitioning
- Graph Growing
- Flows
- Geometric Partitioning
- Streaming Graph Partitioning (SGP)
- Node-swapping Local Search
- Multilevel Graph Partitioning

3.3. Graph Partitioning Software

Several software packages implement various graph partitioning algorithms. Notable ones include: [3]

Metis (and its variants kMetis and hMetis) Known for its speed and flexibility, Metis is widely used for graph partitioning, with kMetis focusing on partitioning speed and hMetis emphasizing partition quality, especially in hypergraph scenarios.

Scotch Developed by Pellegrini, Scotch offers a comprehensive graph partitioning framework with support for both sequential and parallel techniques. It utilizes recursive multilevel bisection for effective partitioning.

KaHIP (Karlsruhe High-Quality Partitioning) Released by Sanders and Schulz, KaHIP implements various advanced methods, including flow-based techniques, more-localized local searches, and diverse parallel and sequential meta-heuristics. It has demonstrated high performance in challenges and benchmarks.

Chaco One of the early publicly available packages, Chaco, developed by Hendrickson and Leland, implements multilevel approaches, basic local search algorithms, and spectral partitioning techniques.

ParMetis A parallel implementation of the Metis GP algorithm by Karypis and Kumar, ParMetis is widely used for efficient graph partitioning in parallel computing environments.

4. METHODOLOGY

4.1. Integer Programming

Integer programming (IP) is a mathematical optimization technique used to find the optimal solution to a problem, where the decision variables are required to take integer values. It is a special case of linear programming (LP), where the decision variables can be any real number. In the context of graph partitioning, integer programming becomes a powerful tool for formulating and solving the problems. In our case, the goal is to find an optimal way to partition the graph's vertices while considering specific constraints, such as minimizing the number of edges between partitions.

4.1.1. Methods of Solving Integer Programming Models

4.1.1.1. Branch and Bound. Branch and bound is a common way of solving integer programming models. Before solving the problem, the integer constraints on decision variables are relaxed and treated as continuous variables. Then, a basic, feasible solution is found using regular linear programming solving techniques like the simplex method. If the basic solution contains originally integer variables that are fractional, then branch and bound algorithm starts.

For the variables that got fractional values, the problem is split into two branches by adding additional constraints to omit the fractional value from the set of feasible solutions: For example, if x_i is originally an integer variable but $x_i = 5.3$ in the relaxed LP formulation, one branch adds the constraint $x_i \leq 5$ and the other adds $x_i \geq 6$. For each branch, the problem is solved again with the hope of finding an integer solution. This procedure is repeated until an integer solution is found and it is made sure that no better unknown solutions exist. [13] Figure 4.1 summarizes the branch and bound method on a sample maximization problem with 2 integer variables (x_1, x_2).

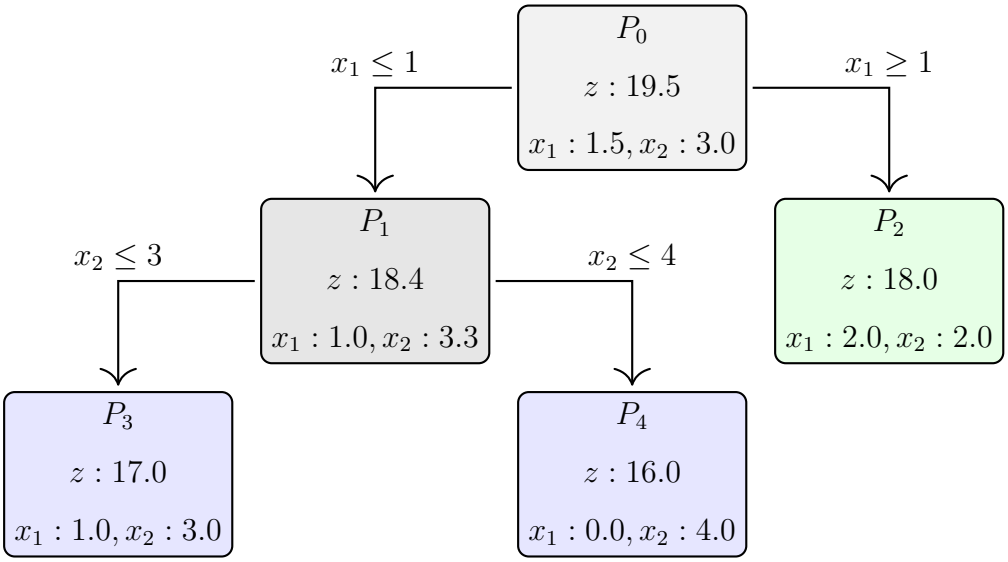

Figure 4.1. Branch and bound method

4.1.1.2. Cutting Plane. Cutting plane method, is similar to branch and bound. Initially, the relaxed version of the problem is solved using simplex. (Integer constraints are discarded.) Then, if the solution includes variables violating the integer constraint, additional cutting constraints called "Gomory cuts" are added to tighten the relaxed problem. The Gomory cuts are special kinds of cuts in the feasible region of the problem that omit the previously found optimal (but violating integrality constraint) solution without eliminating any other integer solutions of the problem. Cuts are added iteratively to achieve an integer solution. [14] An overview of the cutting plane algorithm can be seen in Figure 4.2

4.2. Gurobi

Gurobi is a cutting-edge optimization engine designed for solving complex mathematical optimization problems, including integer programming. [15] Gurobi uses branch and bound and cutting plane methods together while solving an integer model, and the joint use of these methods is called "branch and cut". One of Gurobi's notable features is its ability to perform parallel optimization, harnessing the power of modern computing architectures to accelerate problem-solving. By efficiently distributing computation across multiple processors, Gurobi enhances the speed and scalability of

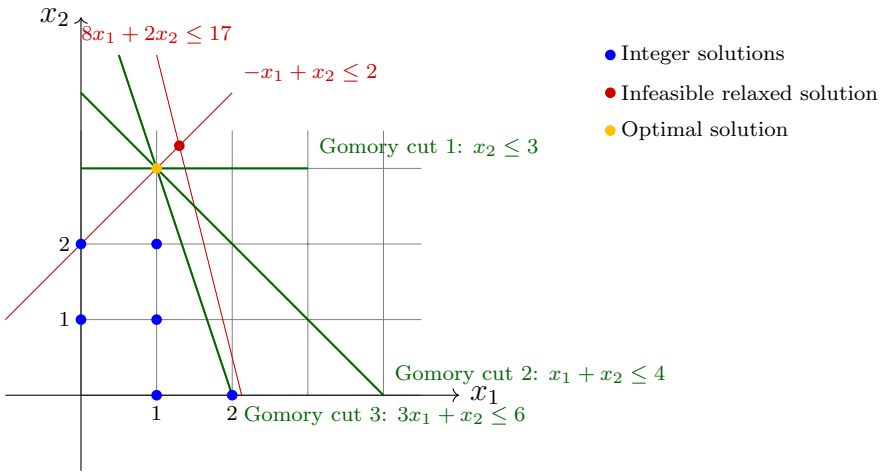

Figure 4.2. Cutting plane method

integer programming solutions. Because of these reasons, Gurobi will be utilized in the software. Gurobi has several APIs for different programming environments. Since the computation speed is a concern for this study, C++ is chosen as the programming language, because it is much faster than interpreted languages such as Python.

5. REQUIREMENTS SPECIFICATION

(i) Functional Requirements

(a) Algorithm

i. The algorithm **MUST** partition the graph into two equal subgraphs with minimum edge cuts.

ii. The algorithm **MUST** use the methodologies for solving integer programming models.

iii. The algorithm **MUST** be able to receive a graph from the user as input.

iv. The algorithm **MUST** be able to give the output of edges cut and the partition information back to the user.

(b) Implementation

i. The software **SHALL** utilize integer programming model solvers like Gurobi, CPLEX when solcing the IP model.

(c) Performance

i. The software **MUST** be able to partition graphs of different sizes and densities.

6. DESIGN

6.1. Information Structure

This project does not use any database. The input is taken from the user in the form as an input file (.txt file). The output is given in the terminal (console).

6.2. Information Flow

The flow of the software is summarized in Figure 6.1.

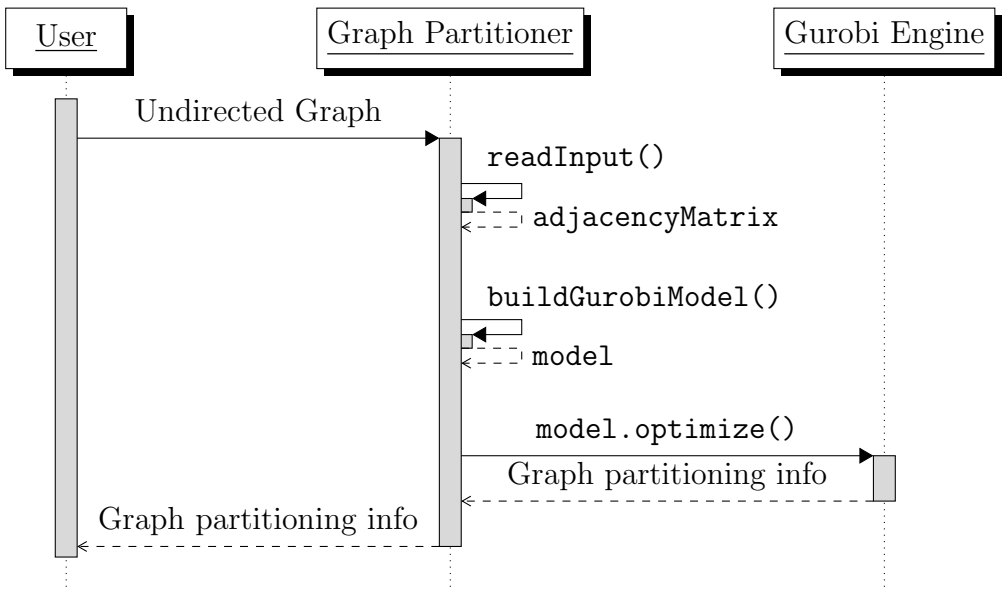

Figure 6.1. Sequence Diagram

6.3. System Design

6.3.1. Integer Programming Model (Weightless)

Let's define $V = \{1, \ldots, n\}$ as the set of vertices in the graph, and E as the adjacency matrix of the graph. $E_{ij} = 1$ if there is an edge from vertex i to j, and 0 otherwise. For each vertex $i \in V$, we need to determine the node's partition. We name

the partitions as partition #0 and partition #1 for convenience.

We can denote the partition of vertex i as p_i. p_i can be either 0 or 1, so it is a binary decision variable of our model. Furthermore, when partitioning the graph, if two nodes in different partitions are adjacent (i.e. has an edge between them), the edge between those vertices need to be "cut". Our aim is to minimize the "cuts" that we have to make. For each $i, j \in V$ and $E_{ij} = 1$, we define x_{ij} as a decision variable that is 1 if an edge will be cut between i and j, and 0 otherwise. So, x_{ij} will be our second binary decision variable which is not defined for pairs of vertices that are not adjacent.

On top of these, it is worth mentioning that we will use only the lower triangle of the adjacency matrix because the adjacency matrix is symmetric, and it is enough to build our model.

The objective function of this problem will be the sum of x_{ij}'s.

In order for this model to work, we have to define the constraints of a model:

(i) We can cut an edge only if it exists. (x_{ij} is defined only when $E_{ij} = 1$.)
(ii) If nodes i and j are connected, they have to be in the same partition. ($p_i = p_j$)
(iii) Partition sizes must be equal.

So, with everything set, here is the summary of our integer programming model:

6.3.1.1. Sets.

- $V = \{1, \ldots, n\}$: the set of vertices.

6.3.1.2. Parameters.

- $E_{ij} = \begin{cases} 1, & \text{if there is an edge from vertex } i \text{ to } j \\ 0, & \text{otherwise} \end{cases}, \forall i, j \in V, i < j$

6.3.1.3. Decision Variables.

- $x_{ij} = \begin{cases} 1, & \text{if edge between vertex } i \text{ and } j \text{ is "cut"} \\ 0, & \text{otherwise} \end{cases}, \forall i, j \in V, E_{ij} = 1$
- $p_i = \begin{cases} 1, & \text{if vertex } i \text{ is in partition \#1} \\ 0, & \text{if vertex } i \text{ is in partition \#0} \end{cases}, \forall i \in V$

$$\text{minimize} \quad \sum_{i=2}^{n} \sum_{j \in V | E_{ij}=1} x_{ij}$$

subject to

$$p_i - p_j \leq x_{ij} \qquad \forall i, j \in V, E_{ij} = 1, \text{ (Adjacent nodes are in the same partition)},$$

$$p_i - p_j \geq -x_{ij} \qquad \forall i, j \in V, E_{ij} = 1, \text{ (Adjacent nodes are in the same partition)},$$

$$\sum_{i=1}^{n} p_i = \left\lfloor \frac{n}{2} \right\rfloor \qquad \text{(Equal partition sizes)},$$

$$x_{ij} = \{0, 1\} \qquad \forall i, j \in V, E_{ij} = 1, \text{ (Binary variable constraint)},$$

$$p_i = \{0, 1\} \qquad \forall i \in V, \text{ (Binary variable constraint)}$$

7. IMPLEMENTATION AND TESTING

7.1. Implementation

The integer programming model for the weightless graph partitioning is programmed in C++ using Gurobi's C++ API. The repository of the software can be obtained from [16].

7.2. Testing

The performance of the program is tested with various graphs on a MacBook Pro with the following hardware:

Processor Name Quad-Core Intel Core i5
Processor Speed 1.4GHz
Number of Processors : 1
Total Number of Cores : 4
L2 Cache (per Core) 256 KB
L3 Cache 6 MB
Hyper-Threading Technology Enabled
Memory 16 GB

The results of the test is discussed in Section 8

7.3. Deployment

Deployment diagram, building instructions, docker/kubernetes, readme, system manual and user manual

To be able to run the program, the user should obtain a license from Gurobi's

website. After installing the license in a computer, the program's repository can be cloned with the following command:

```
git clone https://github.com/araldortogul/CMPE492_Partitioning_Undirected_Graphs.git
```

After cloning, the program can be built by opening a terminal in the repository and typing

```
make
```

which will apply the commands in the Makefile provided in the repository. This command will produce an executable called "`weightless_graph_partitioner`". Before using the program, the user must provide the lower triangle of the adjacency matrix of his/her graph in a .txt file.

Here is the usage of `weightless_graph_partitioner` in a terminal opened in the folder containing it:
Usage: `./weightless_graph_partitioner <option(s)>`
Options:

-i, --input The input file's destination path.
-s, --size The size of the graph(s).
-n, --number The number of graphs.
Required if the input graphs are going to be random.
Not used if `-i` is given.
-l, --log The log file's destination path.
Default: `weightless_graph_partitioner.log`
-d, --density The density of the Erdös-Rényi random graphs.
Required if the input graphs are going to be random.
Not used if `-i` option is given.

8. RESULTS

The graph partitioning software has been run on hundreds of graphs with different size and densities.

8.1. Graph Size = 20

When the size is 20, Gurobi can solve the problem quite fast. Figure A.1 shows the statistics obtained from different densities when the graph size is 20.

For sparse graphs (density = 0.2), Gurobi finds the optimal solution approximately in 0.02 seconds, with less than 1000 iterations. Usually, the optimal solution is between 5 to 15 cuts.

According to Figure A.1, the average optimal solution increases to somewhere between 30 and 40 when the density is increased to 0.5. This is expected because these graphs have more edges than the graphs with 0.2 density. In response, the average time spent in finding the optimal solution increased by 20 times, the solution can be found in 0.4 seconds on average. Also, the number of simplex iterations increased in parallel with the runtime.

When the density is increased from 0.5 to 0.8, the runtime increased, but the increase is not as dramatic as the increase from density = 0.2 to 0.5.

8.2. Graph Size = 30

The performance of the program is also tested with graphs of size 30. The statistics can be examined in Figures A.2 in Appendix A for densities 0.2, 0.5, and 0.8. A similar behavior is observed when the density is increased. But for density 0.8, the runtime increases critically. On the average, it took 20 seconds for Gurobi to find the

best partition. The reason behind this is that the IP model introduces $|V|+|E|$ binary decision variables. Since E is proportional to $|V|^2$, the density of the graph ultimately determines the size and the complexity of the model. When the density is 0.2, Gurobi is able to compute the optimal solution in less than a second.

8.3. Graphs of Size Greater Than 30

For graphs of size greater than 30, the integer programming model becomes so complex that the program cannot find an optimal solution for dense graphs. However, the software can generate output for large, sparse graphs.

9. CONCLUSION

The integer programming model for the weightless graph partitioning problem solves the problem successfully, but the complexity of the problem increases very quickly when the size of the graph increases. For large graphs, it is able to find a solution in a reasonable time only when the density of the graph is very low.

The weighted version of the problem will be modeled as an integer program, and the software will be updated according to that. The performance evaluation will also be done for the weighted version.

REFERENCES

1. Chung, F., *Spectral Graph Theory*, Conference Board of Mathematical Sciences, American Mathematical Society, 1997, https://books.google.com.tr/books?id=4IK8DgAAQBAJ.

2. Feldmann, A., "Fast Balanced Partitioning Is Hard Even on Grids and Trees", *Theoretical Computer Science*, Vol. 485, 11 2011.

3. Buluç, A., H. Meyerhenke, I. Safro, P. Sanders and C. Schulz, *Recent Advances in Graph Partitioning*, pp. 117–158, Springer International Publishing, 2 2016, https://doi.org/10.1007/978-3-319-49487-6_4.

4. Karisch, S., F. Rendl and J. Clausen, "Solving Graph Bisection Problems with Semidefinite Programming", *INFORMS Journal on Computing*, Vol. 12, pp. 177–191, 08 2000.

5. Armbruster, M., "Branch-and-Cut for a Semidefinite Relaxation of Large-scale Minimum Bisection Problems", , 06 2007.

6. Sellmann, M., N. Sensen and L. Timajev, "Multicommodity Flow Approximation Used for Exact Graph Partitioning", G. Di Battista and U. Zwick (Editors), *Algorithms - ESA 2003*, pp. 752–764, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

7. Sensen, N., "Lower Bounds and Exact Algorithms for the Graph Partitioning Problem Using Multicommodity Flows", , 06 2001.

8. Brunetta, L., M. Conforti and G. Rinaldi, "A branch-and-cut algorithm for the equicut problem", *Mathematical Programming*, Vol. 77, pp. 243–263, 08 1997.

9. Ferreira, C., A. Martin, C. Souza, R. Weismantel and L. Wolsey, "The node capac-

itated graph partitioning problem: A computational study", *Mathematical Programming*, Vol. 81, pp. 229–256, 01 1998.

10. Lisser, A. and F. Rendl, "Rendl, F.: Graph partitioning using linear and semidefinite programming. Math. Program. Ser. B 95(1), 91-101", *Mathematical Programming*, Vol. 95, pp. 91–101, 01 2003.

11. Hager, W. W., D. T. Phan and H. Zhang, "An exact algorithm for graph partitioning", *Mathematical Programming*, Vol. 137, No. 1, pp. 531–556, 2013, https://doi.org/10.1007/s10107-011-0503-x.

12. Hager, W. W. and Y. S. Krylyuk, "Graph Partitioning and Continuous Quadratic Programming", *SIAM J. Discret. Math.*, Vol. 12, pp. 500–523, 1999, https://api.semanticscholar.org/CorpusID:14431056.

13. Wolsey, L. A., *Branch and Bound*, chap. 7, pp. 113–138, John Wiley & Sons, Ltd, 2020, https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119606475.ch7.

14. Wolsey, L. A., *Cutting Plane Algorithms*, chap. 8, pp. 139–166, John Wiley & Sons, Ltd, 2020, https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119606475.ch8.

15. Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual", , 2023, https://www.gurobi.com.

16. Dörtoğul, A., "Graph Partitioner", https://github.com/araldortogul/CMPE492_Partitioning_Undirected_Graphs.

APPENDIX A: Statistics Obtained by Testing the Software

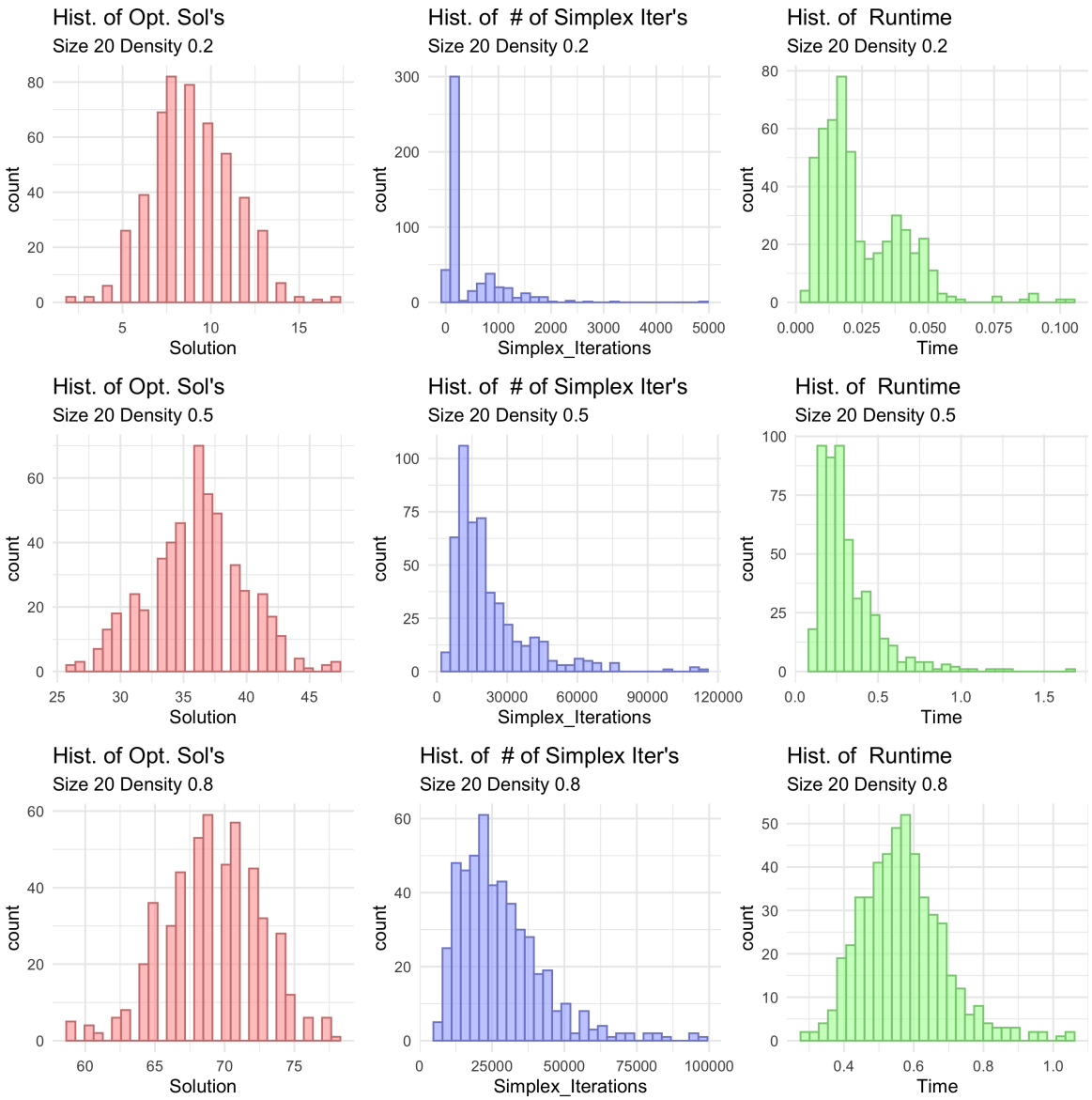

Figure A.1. Statistics of the optimal solution when graph size = 20

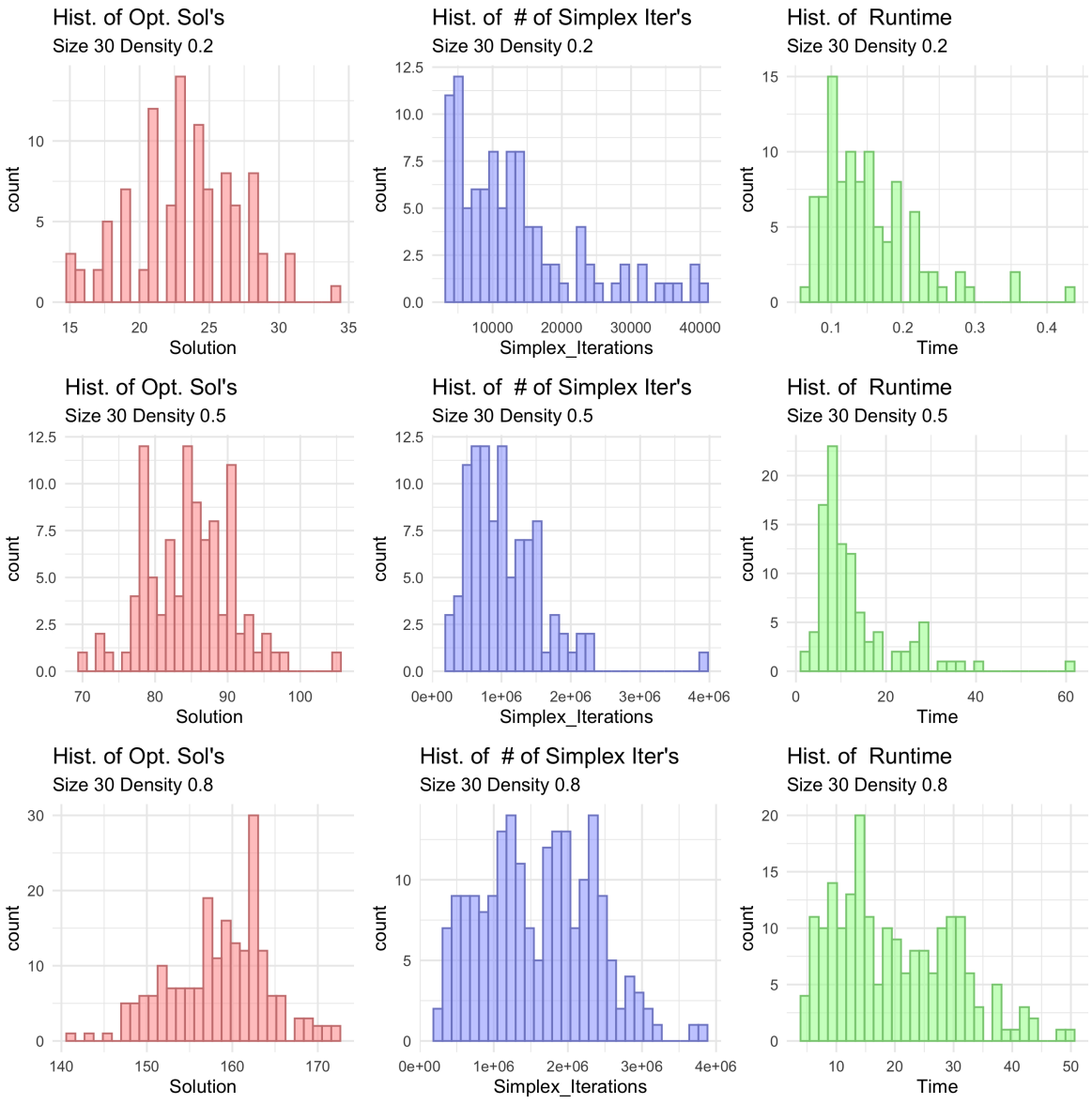

Figure A.2. Statistics of the optimal solution when graph size = 30