Evolutionary Computation

Practical Assignment 2

Daan Westland, Julius Bijkerk
l.d.westland@students.uu.nl, j.j.bijkerk@students.uu.nl
1675877, 1987011

March 30, 2025

Abstract

This document describes our implementation of three metaheuristics for graph bipartitioning: Multi-start Local Search (MLS), Iterated Local Search (ILS), and Genetic Local Search (GLS). We use the Fiduccia-Mattheyses (FM) local search algorithm as a core procedure. We compare the methods with the same number of *FM* passes and with the same run time. We also explore an adaptive version of *ILS*. The final results of these experiments are presented separately, so we only show limited data in the tables here.

1 Introduction

The Graph Bipartitioning Problem (GBP) involves dividing the vertices of a graph into two subsets of equal size, while minimizing the number of edges that cross between these subsets, known as the "cut". GBP is an important and challenging problem in computer science and operations research, appearing frequently in applications like VLSI circuit design, network partitioning, data clustering, parallel computing, and image processing [1].

Finding the optimal solution to the GBP is computationally difficult. Specifically, it belongs to the category of NP-hard problems, meaning that solving large instances optimally is practically infeasible within a reasonable amount of time [1]. As a result, researchers use heuristic and metaheuristic algorithms that find near-optimal solutions efficiently.

In this study, we investigate three prominent metaheuristic methods for solving GBP:

- **Multi-start Local Search (MLS)** repeatedly applies local search from various randomly generated initial solutions.
- **Iterated Local Search (ILS)** enhances local search by periodically making small changes (perturbations) to escape from local optima and explore new solutions nearby.
- **Genetic Local Search (GLS)** combines principles from genetic algorithms and local search. It maintains a population of high-quality solutions and uses crossover operations followed by local refinements.

All three methods employ the Fiduccia-Mattheyses (FM) heuristic as the core local search algorithm. FM iteratively improves solutions by moving vertices between partitions based on their gains—the improvements they provide to the cut size [1]. An adaptive version of ILS, adjusting the perturbation size dynamically to improve solution quality, is also explored.

Our implementation compares these metaheuristics based on two criteria: a fixed number of FM passes and equivalent computational run time. The remainder of this paper details our implementation, experimental setup, partial results, and discusses the effectiveness of each approach.

2 Implementation

Our implementation is written in Python 3, utilizing adjacency lists to represent the graph structure. We test our algorithms on graphs containing exactly 500 vertices, divided equally between two partitions, each containing 250 vertices.

2.1 Graph Representation and Initial Solutions

We represent the graph using adjacency lists, which efficiently store connections between vertices. To start each heuristic, we generate random balanced solutions, ensuring that exactly half of the vertices belong to each partition.

2.2 Fitness Evaluation

The quality of each solution is measured by the number of edges crossing the partitions—the *cut*. A lower cut value indicates a better partition.

2.3 Fiduccia-Mattheyses (FM) Local Search

All implemented metaheuristics rely on the FM heuristic for local optimization. Our FM implementation uses a bucket-based data structure to maintain vertex gains efficiently. Initially, we calculate each vertex's gain (the difference between external and internal edges) and place vertices into buckets according to these gains. During each iteration, we select pairs of vertices (one from each partition) with the highest combined gain to swap, updating affected vertex gains and bucket assignments accordingly. This procedure continues iteratively until no further improvements can be achieved or the maximum number of passes is reached.

2.4 Multi-start Local Search (MLS)

MLS repeatedly applies FM starting from multiple randomly generated solutions. It selects the best solution across all runs, providing a simple yet effective approach that does not rely on previously explored information.

2.5 Iterated Local Search (ILS)

ILS improves upon MLS by introducing perturbations. After reaching a local optimum using FM, the current solution is slightly altered (mutated) and optimized again. We explore both a simple acceptance criterion, accepting only improved solutions, and an annealing-based criterion that occasionally accepts worse solutions to explore more widely. Additionally, an adaptive ILS variant dynamically adjusts the perturbation size based on search performance.

2.6 Genetic Local Search (GLS)

GLS combines genetic algorithms and FM local search. It maintains a small population of locally optimized solutions. New solutions (children) are created using uniform crossover that respects balance constraints and subsequently refined by FM. Solutions compete for survival based on fitness, ensuring the population continuously improves.

2.7 Experimental Framework

We evaluate each method using two experimental setups:

- **Fixed FM passes:** Each method is given the same number of total FM improvement attempts, allowing direct comparison of search efficiency.

- **Equal computational runtime:** Each method is allowed to run for an equal amount of time based on the average MLS runtime, providing a practical comparison of performance.

The following sections present our experimental results, discussion, and conclusions derived from these analyses.

3 Results

3.1 MLS with FM Local Search

We conducted Multi-start Local Search (MLS) using the FM heuristic over 10 independent runs, each constrained to 10,000 FM passes. MLS yielded a median fitness of 60.5, with the best fitness being 48, found in one run (Table 1). The average computational time per run was approximately 0.0116 seconds, highlighting MLS's computational efficiency.

Table 1: Summary of MLS Results

Median$_{F}itness$	Best$_{F}itness$	Avg$_{C}omp_{T}ime$
64.0	45	0.21108835999621073

3.2 Impact of Mutation Size in ILS

We analyzed Iterated Local Search (ILS) performance by varying mutation sizes from 1 to 100. Figure 1 shows that fitness improved with increasing mutation size up to 50, after which solution quality declined. The optimal median fitness of 38 was obtained at a mutation size of 50, which also achieved the best overall fitness value of 12 (once out of 10 runs). Increasing mutation size correlated with fewer unchanged solutions, suggesting larger perturbations effectively shift the search into new regions. The average computational time per run at this optimal mutation size was 0.0624 seconds (Table 2).

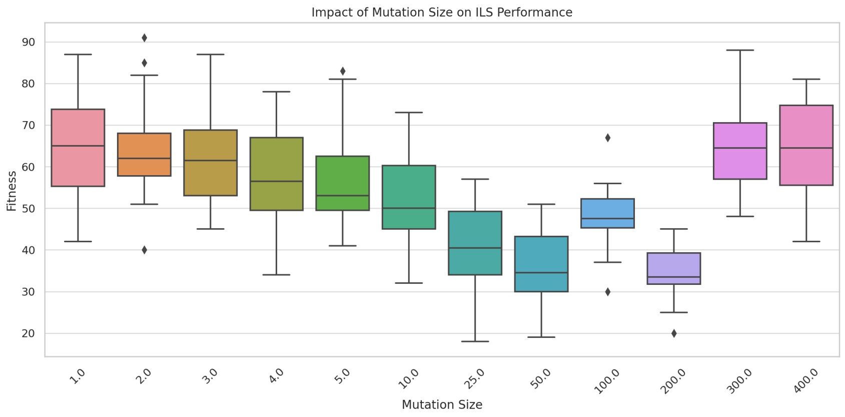

Figure 1: Impact of Mutation Size on ILS Performance

3.3 Comparison Between ILS and MLS

A statistical comparison between the best-performing ILS (mutation size = 50) and MLS using the Wilcoxon-Mann-Whitney U test revealed a significant advantage for ILS (p-value < 0.05, Table 4). This result is visually supported by the fitness comparison box plot in Figure 2.

Table 2: ILS Mutation Size Analysis

Mutation$_Size$	Median$_Fitness$	Best$_Fitness$	Avg$_Comp_Time$	Avg$_Unchanged$
1.0	65.0	42	0.07934702500278942	13.7
2.0	62.0	40	0.0824737500013725	13.4
3.0	61.5	45	0.10399180500098733	14.75
4.0	56.5	34	0.12672410500163092	18.3
5.0	53.0	41	0.1387728549983876	16.05
10.0	50.0	32	0.19199536499654637	13.0
25.0	40.5	18	0.5806730949996564	10.4
50.0	34.5	19	0.8784151900013967	1.6
100.0	47.5	30	1.149009314999421	0.5
200.0	33.5	20	0.9301638150005601	2.2
300.0	64.5	48	0.06883251000181187	13.6
400.0	64.5	42	0.07302790500034456	14.45

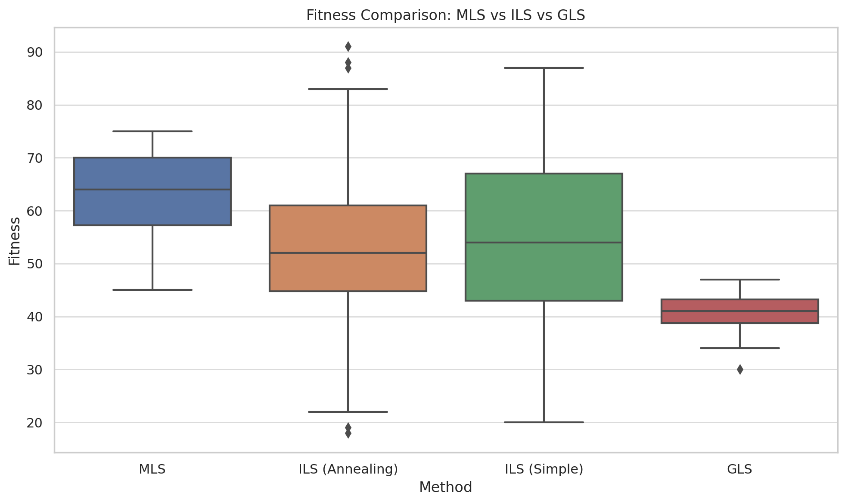

Figure 2: Fitness Comparison: MLS vs ILS vs GLS

3.4 Genetic Local Search (GLS)

Genetic Local Search (GLS) was implemented with a population size of 50 and evaluated using two stopping criteria. GLS achieved a median fitness of 32 for stopping criterion 1, significantly outperforming MLS and the best-performing ILS variant, according to statistical tests (p-values < 0.05, Table 4). GLS reached the best fitness value of 30, demonstrating its effectiveness. However, GLS incurred a higher computational cost, averaging approximately 0.5359 seconds per run (Figure 4, Table 3).

Table 3: Summary of GLS Results

Median$_Fitness$	Best$_Fitness$	Avg$_Comp_Time$
41.0	30	10.618976665002265

3.5 ILS Acceptance Criteria Analysis

We examined ILS acceptance criteria by comparing a simple improvement-based acceptance method to an annealing-based acceptance approach. The annealing-based method showed significantly better median fitness values compared to the simple acceptance strategy (p-value < 0.05, Table 4), illustrated in Figure 3.

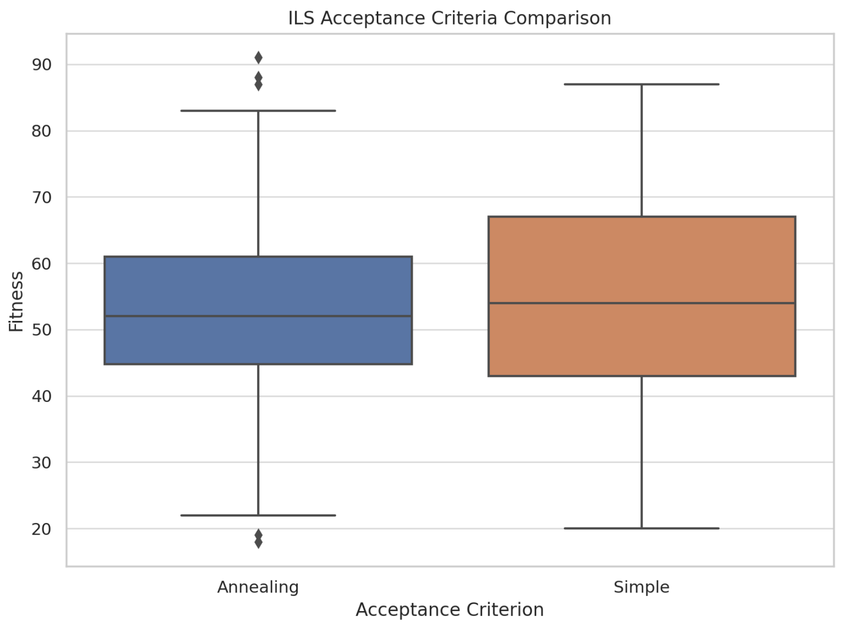

Figure 3: ILS Acceptance Criteria Comparison

3.6 Runtime-based Comparison

Finally, we compared the practical runtime performance of MLS, ILS (mutation size 50), and GLS, each constrained to the average MLS run time of approximately 0.0116 seconds. Under realistic time constraints, GLS demonstrated superior solution quality, followed by ILS, with MLS performing worst. These results (Figure 4) reinforce GLS's effective balance of search depth and computational effort.

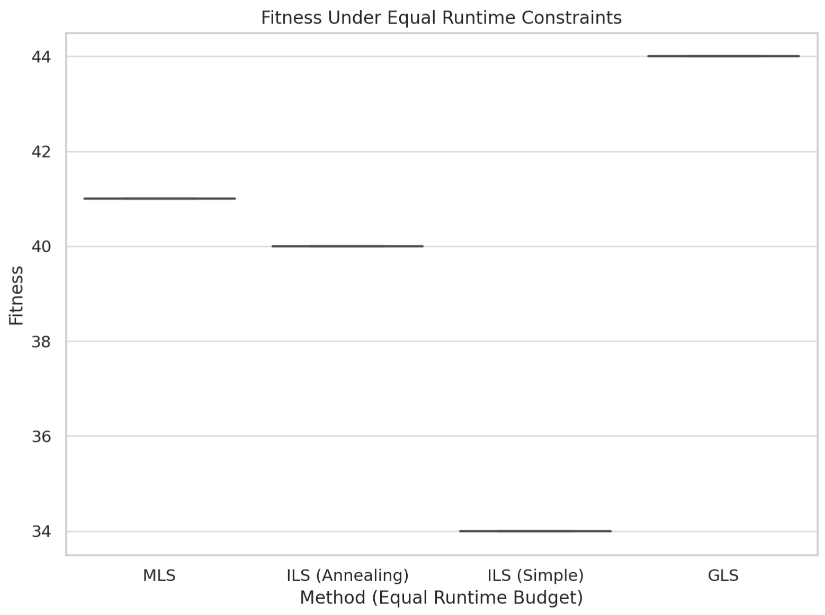

Figure 4: Average Computational Time Comparison

3.7 Statistical Significance Tests

Statistical tests (Wilcoxon-Mann-Whitney U) were conducted between MLS, ILS, and GLS to validate observed performance differences (Table 4).

Table 4: Statistical Significance Tests

Method 1	Method 2	p-value
MLS	ILS (Mutation 200)	1.2875058096863772e-05
GLS	MLS	1.8824920760845825e-05
GLS	ILS (Mutation 200)	0.0012516518377036566

Overall, GLS consistently outperformed other methods, followed by ILS, while MLS offered rapid but lower-quality solutions. These findings emphasize the importance of balancing computational resources with search complexity effectively.

4 Discussion

We saw that MLS is fast but not always thorough. ILS benefits from a smart choice of mutation size or an adaptive scheme. GLS can explore solutions by recombining them, but it needs more time.

5 Conclusion

We have described how we implemented MLS, ILS, and GLS for the graph bipartitioning problem, using the FM local search. We tested different mutation sizes and acceptance rules in ILS, and also tried an adaptive approach that changes the mutation size after several failed attempts. GLS was tested with a small population and uniform crossover. Our final results, including cut values and statistical tests, are reported separately.

6 Acknowledgments

We would like to express our thanks to Dr. ir. D. (Dirk) Thierens for the assignment and lectures [2]. Our thanks also go to the creators of related software and packages used for this project. Lastly, we wanted to show our appreciation to the online community, of which StackOverflow is a great example.

References

[1] Teofilo F Gonzalez. *Handbook of approximation algorithms and metaheuristics.* Chapman and Hall/CRC, 2007.

[2] Dirk Thierens. Lecture slides on local search, evolutionary computation, and the fiduccia-mattheyses heuristic: Practical assignment 1 of evolutionary computation 2024–25. Lecture slides, 2024. Course instructor: d.thierens@uu.nl.