Comparative Analysis of Metaheuristics for the Graph Bipartitioning Problem

Practical Assignment 2 of Evolutionary Computation 2024–25

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

April 3, 2025

Abstract

This study investigates five metaheuristics for solving the Graph Bipartitioning Problem (GBP): Multi-start Local Search (MLS), Iterated Local Search (ILS), ILS with Simulated Annealing (ILS-SA), ILS with Adaptive Mutation (ILS-A), and Genetic Local Search (GLS). All methods use the Fiduccia-Mattheyses (FM) heuristic as a local improvement strategy. Experiments were conducted on a 500-vertex instance under two conditions: a fixed number of FM passes and a fixed runtime. Results demonstrate that under high computational budgets, ILS variants and GLS significantly outperform MLS. In runtime-constrained settings, however, MLS remains highly competitive due to its simplicity and low overhead. ILS-A emerges as a strong performer across both settings, balancing robustness and efficiency. Statistical comparisons validate the observed performance differences. This work highlights the trade-offs between computational effort and solution quality and suggests practical guidance for selecting metaheuristics in time-sensitive or quality-sensitive applications.

1 Introduction

This report addresses the Graph Bipartitioning Problem (GP), a fundamental challenge in graph theory and combinatorial optimization. The objective is to partition the vertex set of a given graph into two disjoint subsets of equal cardinality, such that the number of edges connecting vertices across the two subsets (the cut size) is minimized. GP arises frequently in diverse application domains, including VLSI circuit design, network partitioning, data clustering, parallel computing task allocation, and image processing [1]. Due to its NP-hard nature, finding provably optimal solutions for large graph instances is computationally intractable [2]. Consequently, heuristic methods are indispensable for obtaining high-quality solutions within practical time limits.

Our study employs the Fiduccia-Mattheyses (FM) algorithm [3], a widely-used iterative improvement heuristic based on the earlier Kernighan-Lin method, as the core local search procedure. FM iteratively moves vertices based on a gain metric to reduce the cut size but can converge to suboptimal local optima. To address this, we investigate several metaheuristics designed to guide the FM search more effectively:

- **Multi-start Local Search (MLS):** Executes FM multiple times from random starting points [4].
- **Iterated Local Search (ILS):** Iteratively applies perturbation and FM local search, using an acceptance criterion to navigate the search space [5].
- **Genetic Local Search (GLS):** A memetic algorithm combining evolutionary principles (population, selection, crossover) with FM local refinement [6].

Furthermore, motivated by suggestions in the practical assignment guide and the broader literature, we investigate two different ILS variants:

- **ILS with Simulated Annealing (ILS-SA):** Incorporates the probabilistic Metropolis acceptance criterion from SA [7] into ILS to potentially improve escape from local optima.

- **ILS with Adaptive Mutation (ILS-A):** Implements a mechanism to dynamically adjust the perturbation strength (`mutation_size`) during the search, aiming for robust performance without manual tuning by balancing intensification and diversification [8].

This report provides a formal definition of GP, it details our Python 3 implementation of the FM heuristic and the five metaheuristics (MLS, ILS, GLS, ILS-SA, ILS-A), outlines the experimental design including parameter tuning via grid search, describes the data analysis and statistical methods employed, and presents a comparative analysis of their effectiveness on the `Graph500.txt` instance.

2 Methodology

This section details the problem formulation, the implementation specifics of the algorithms, the experimental setup, and the data analysis techniques used. Our implementation was developed in Python 3.

2.1 Large Language Model (LLM) Assist in Coding

In accordance with course guidelines, we used large language models (LLMs), specifically Google Gemini 2.5 Pro and OpenAI's ChatGPT, to assist with the implementation. LLMs were used for generating modular code for key components such as FM bucket structures, mutation and crossover operators, and the experimental loop. Additionally, they aided in structuring statistical tests and plotting scripts. We manually reviewed and validated all suggestions, adapting them to fit our problem context, resulting in our final file `project2_final.py`.

2.2 The Graph Bipartitioning Problem (GP)

Given an undirected graph $G = (V, E)$, where V is the set of n vertices and E is the set of edges, the GP seeks a partition of V into two disjoint subsets, V_1 and V_2, such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. The primary objective is to minimize the cut size, defined as the number of edges $(u, v) \in E$ where $u \in V_1$ and $v \in V_2$. A critical constraint is partition balance, typically requiring $|V_1| = |V_2| = n/2$ for vertex-balanced GP, as is the case in our study with $n = 500$. The minimum bisection problem (perfectly balanced GP) is known to be NP-complete [2]. Heuristics must navigate the trade-off between minimizing the cut size and strictly maintaining this balance.

2.3 Problem Encoding and Objective Function

Our experiments were conducted on the `Graph500.txt` graph instance ($N = 500$). Connectivity is represented using adjacency lists. A candidate solution is encoded as a binary vector of length 500, where $x_i \in \{0, 1\}$ denotes the partition of vertex i. The balance constraint requires exactly 250 zeros and 250 ones. The objective function, `fitness_function`, calculates the cut size. Random balanced initial solutions are generated via `generate_random_solution`.

2.4 Fiduccia-Mattheyses (FM) Heuristic Implementation

The FM algorithm [3], building upon the Kernighan-Lin approach, serves as our local improvement engine (`fm_heuristic`).

- **Gain Calculation:** Vertex gains, representing the cut size reduction upon moving a vertex, are computed via `calculate_gain`.
- **Efficient Data Structures:** We employ gain buckets (arrays indexed from $-p_{max}$ to $+p_{max}$, where p_{max} relates to the maximum vertex degree) containing doubly linked lists of free vertices, enabling $O(1)$ access to the highest-gain vertex and $O(1)$ vertex move/update operations. Functions `insert_node`, `remove_node`, and `update_max_gain` manage these structures.
- **FM Pass Procedure:** A single pass involves attempting to move each vertex once. Vertices are selected based on highest gain from the buckets, moved, locked, and neighbor gains are updated. The state yielding the maximum cumulative gain during the pass, while strictly maintaining the 250/250 balance, is adopted. Passes repeat until no improvement occurs. The algorithm exhibits linear time complexity per pass, O($|E|$) for graphs [3].

2.5 Multi-start Local Search (MLS)

The `MLS` function runs `fm_heuristic` multiple times from independent random balanced solutions (`generate_random_solu`
While simple and offering broader exploration than single FM runs, it lacks memory between runs and can redundantly explore the same basins of attraction [9]. The best balanced local optimum found across all runs within the budget is returned.

2.6 Iterated Local Search (ILS)

Our basic `ILS` implementation (`ILS` function) iteratively perturbs and refines solutions [5].

1. Start from an initial local optimum found by `fm_heuristic`.
2. Repeat:
    - **Perturbation:** Apply balanced swap mutation using `mutate` with strength `mutation_size`. Tested sizes: {10, 25, 40, 50, 60, 75, 100}.
    - **Local Search:** Run `fm_heuristic` from the perturbed state.
    - **Acceptance Criterion:** Accept the new optimum greedily (only if strictly better).

The effectiveness hinges on balancing perturbation strength and the acceptance criterion [10].

2.7 Genetic Local Search (GLS)

Our `GLS` function implements a steady-state memetic algorithm [6].

1. **Initialization:** Create a population of `POPULATION_SIZE` = 50 locally optimal solutions (via `fm_heuristic`).
2. **Evolutionary Loop:**
    - Select two parents randomly.
    - Generate one child via balanced uniform `crossover` (with Hamming distance check). The design of balanced operators is crucial for GP [11].
    - Refine the child using `fm_heuristic` (local refinement step).
    - Replace the worst population member if the child is better or equal.

GLS combines the global exploration of GAs with the local exploitation of FM [12].

2.8 ILS with Simulated Annealing (ILS-SA)

The `ILS_annealing` function integrates SA's Metropolis acceptance criterion [7] into ILS. Worse solutions s' are accepted with probability $P = \exp(-\Delta E/T)$, allowing escape from local optima [13]. The temperature T decreases geometrically (`cooling_rate`). Balancing exploration (high T) and exploitation (low T) via the cooling schedule is critical [14]. Parameters were tuned via grid search (`run_grid_search`):

- `mutation_size`: {10, 25, 40, 50, 60, 80}
- `temperature`: {5.0, 10.0, 15.0}
- `cooling_rate`: {0.95, 0.98, 0.99}
- `min_temperature`: {0.1, 0.5}
- `max_no_improvement`: {15, 25}

2.9 ILS with Adaptive Mutation (ILS-A)

The `ILS_adaptive` function employs self-adaptation for `mutation_size` [8]. Size is sampled from $N(\mu, \sigma^2)$. Parameters μ = `current_mutation_mean` and σ = `current_mutation_std_dev` adapt using `decay_factor` (intensification) and `increase_factor` (diversification upon stagnation, detected via `stagnation_threshold`/`stagnation_window`). This aims to automate the exploration/exploitation balance [15]. Tuned via grid search (`run_grid_search`):

- `initial_mutation_mean`: {20, 40, 60}
- `mutation_std_dev_factor`: {0.1, 0.2, 0.5}
- `decay_factor`: {0.9, 0.95, 0.98}
- `increase_factor`: {1.2, 1.5}
- `stagnation_threshold`: {30, 50, 80}

Fixed parameters: `min_mutation_size`=1, `max_mutation_size`=125 ($N/4$), `stagnation_window`=15, `stagnation_tolerance`=0.0001, `restart_reset`=True.

2.10 Experimental Design

Performance was evaluated under two standard scenarios [16]:

1. **Fixed Computational Effort:** Budget of `FM_PASSES_LIMIT` = 10,000 passes. 10 runs per configuration (`EXPERIMENT_RUNS`).
2. **Fixed Runtime:** Budget set by the measured time for MLS to complete 10k passes (`DYNAMIC_TIME_LIMIT`). 25 runs per configuration (`RUNTIME_RUNS`).

2.11 Data Analysis and Statistical Methods

Analysis was performed using Python (`pandas`, `numpy`, `scipy.stats`, `matplotlib`, `seaborn`).

- **Data Verification:** Results from `experiment_results_combined.csv` were verified for solution balance (`verify_solution_balance`) and, if `networkx` was available, for consistency between reported and recalculated fitness (`verify_row_fitness`, `calculate_actual_fitness`).
- **Descriptive Statistics:** Calculated using `calculate_summary_stats`: Min, Median, Mean, Std Dev, Max for `Fitness`; Mean, Std Dev for `Comp_Time`, `Actual_Passes`; Best Fitness Count. Computed overall and grouped by parameters. Results saved (`table_stats_*.csv`).
- **Visualization:** Box plots (`seaborn.boxplot` with `seaborn.stripplot`) compared fitness distributions (`plot_boxplot_*.png`). Line plots (`pandas.plot`) showed parameter trends (`plot_line_*.png`). The best overall partition was visualized (`plot_best_partition_visualization.png`).
- **Statistical Comparison:** The non-parametric Mann-Whitney U test (`scipy.stats.mannwhitneyu`) was used for pairwise fitness comparisons, suitable for potentially non-normal heuristic results [16, 17]. One-sided tests determined the direction of significant differences at $\alpha = 0.05$. Results saved (`table_tests_*.csv`).

3 Results

This section presents the preliminary results of our comparative analysis based on the experiments conducted so far. Full results, including the completed grid search outcomes for ILS-SA and ILS-A, will be available upon completion of the runs. The analysis was performed on the data collected in `experiment_results_combined.csv`. Generated tables and plots are stored in the `analysis_results/` directory.

3.1 Data Verification

Initial verification confirmed that all reported solutions adhere to the length ($N = 500$) and balance (250/250) constraints. Furthermore, using `networkx`, the reported fitness values matched the recalculated cut sizes, indicating data consistency. The best overall fitness observed was 2, achieved by Simple ILS (`mutation_size`=60) in the pass-based scenario. Figure 1 visualizes this partition.

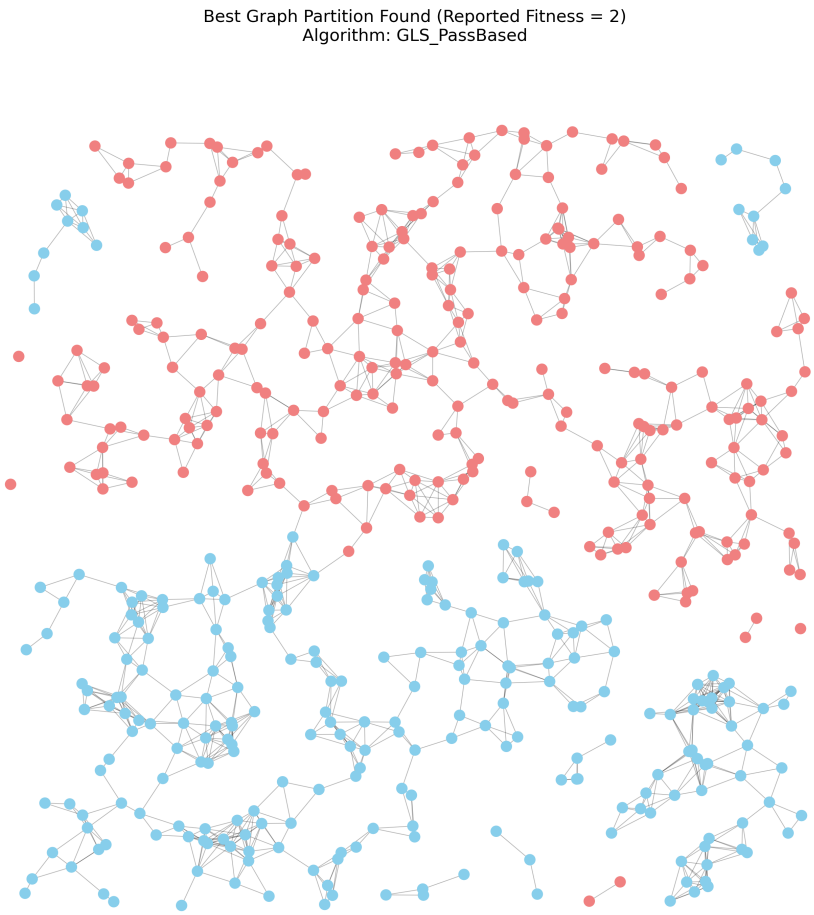

Figure 1: Visualization of the best overall graph partition found (Fitness=2, by Simple ILS mut=60).

3.2 Pass-Based Comparison (Fixed Effort)

Algorithms were compared using a budget of 10,000 FM passes over 10 runs.

- **MLS:** Showed consistent but relatively high cut sizes, with a median fitness of 12.0 and a best of 10.
- **Simple ILS:** Performance was highly dependent on `mutation_size`. As illustrated in Figure 2, fitness improved significantly as mutation size increased from 10 (median 25.5) towards the optimal range. The best median fitness (5.5) occurred at `mutation_size`=60 (Figure 3). Larger sizes showed diminishing returns or degradation. The trend of decreasing mean unchanged counts with increasing mutation size (Figure 4) confirms that larger perturbations better facilitate escape from local optima. The overall best fitness of 2 was achieved with `mutation_size`=60.
- **GLS:** Using `Stopping_Crit`=1, GLS achieved a strong median fitness of 6.0 and a best fitness of 3. The fitness distribution is shown in Figure 5.
- **ILS-SA:** Similar to Simple ILS, performance depended on `mutation_size` (Figure 6). The best median fitness (5.5) was found with `mutation_size`=40. The best individual result was 4.

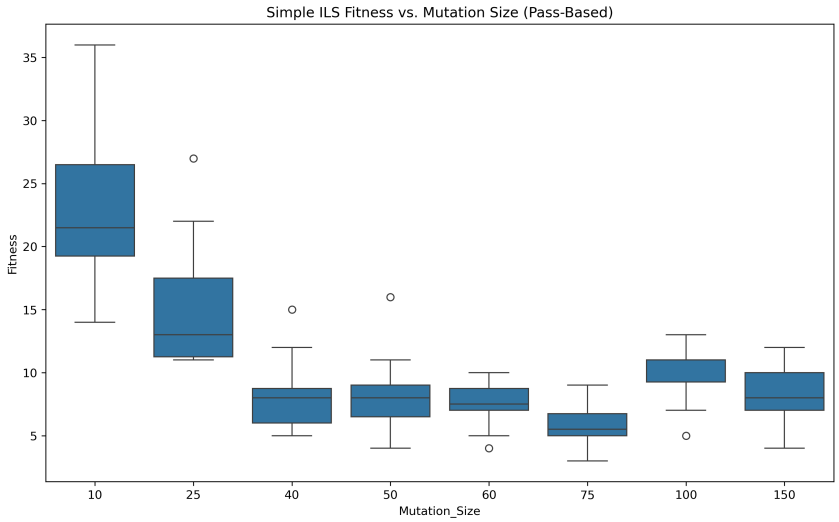

Figure 2: Fitness distribution for Simple ILS vs. Mutation Size (Pass-Based).

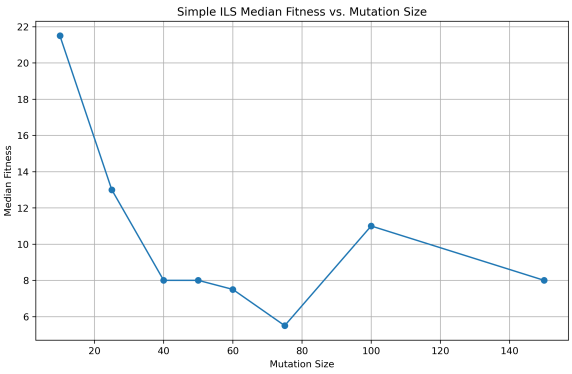

Figure 3: Median Fitness trend for Simple ILS vs. Mutation Size (Pass-Based).

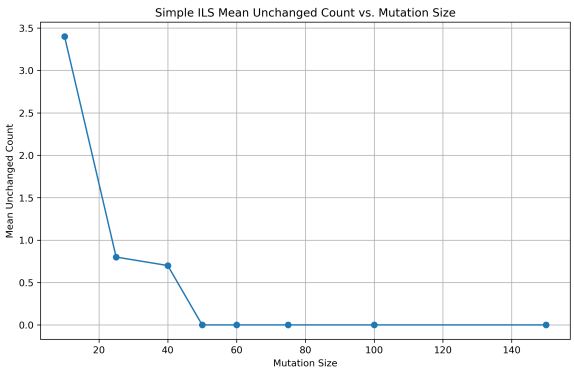

Figure 4: Mean Unchanged Count trend for Simple ILS vs. Mutation Size (Pass-Based).

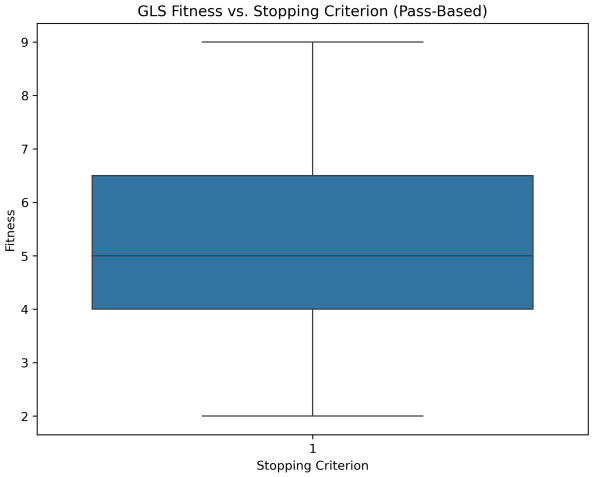

Figure 5: Fitness distribution for GLS (Stopping Crit=1) (Pass-Based).

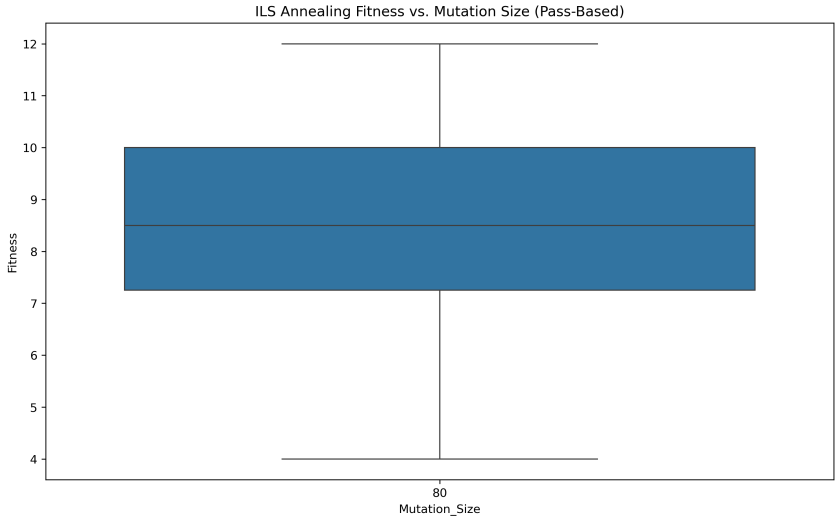

Figure 6: Fitness distribution for ILS-SA vs. Mutation Size (Pass-Based).

- **ILS-A:** Preliminary analysis (grouped by final mean mutation size) indicated a best median fitness of 7.0 (near mean size 29). A definitive assessment awaits completed grid search results.
- **Overall Comparison (Pass-Based):** Figure 7 compares the best configurations found for each algorithm based on median fitness. Simple ILS (mut=60) and ILS-SA (mut=40) achieved the lowest median (5.5), closely followed by GLS (sto=1) at 6.0. ILS-Adaptive (mean=29) was next at 7.0, while MLS lagged significantly (median 12.0). Statistical tests (Mann-Whitney U, $\alpha = 0.05$) indicated that GLS, Simple ILS (mut=60), and ILS-SA (mut=40) were all significantly better than MLS ($p < 0.001$). However, based on these preliminary 10 runs, no statistically significant difference was detected between GLS, Simple ILS (mut=60), ILS-SA (mut=40), and ILS-Adaptive (mean=29). Full pairwise test results are available (`table_tests_BestConfigs_PassBased.csv`).

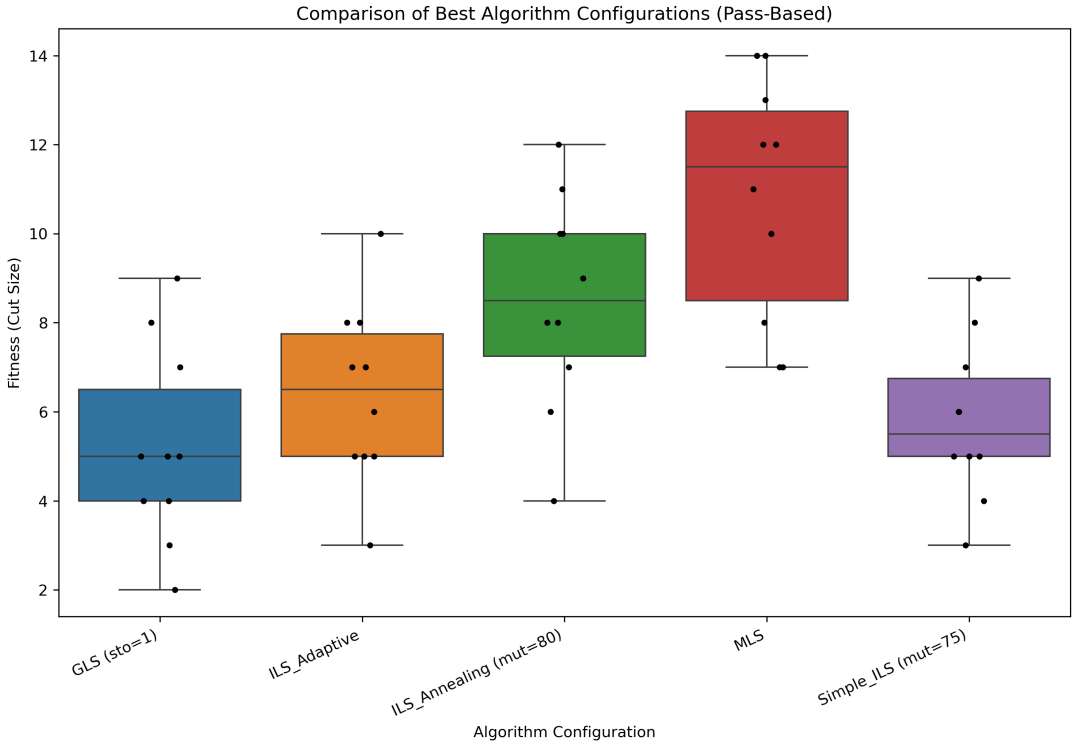

Figure 7: Comparison of best configurations found for each algorithm (Pass-Based, 10k FM passes).

3.3 Runtime-Based Comparison (Fixed Time)

Algorithms were run 25 times each within a fixed time limit ($\approx$ 60 seconds). The performance ranking differed notably from the pass-based scenario, as shown in Figure 8.

MLS achieved the best median fitness (11.0), significantly outperforming all other methods ($p < 0.001$). ILS-Adaptive was the second-best performer (median 12.0), significantly better than the remaining algorithms ($p < 0.001$). Simple ILS (median 28.0), ILS-SA (median 28.0), and GLS (median 25.0) showed statistically indistinguishable performance from each other under this time constraint. Detailed statistics and test results are available (`table_stats_Runtime.csv`, `table_tests_Comparison_Runtime.csv`).

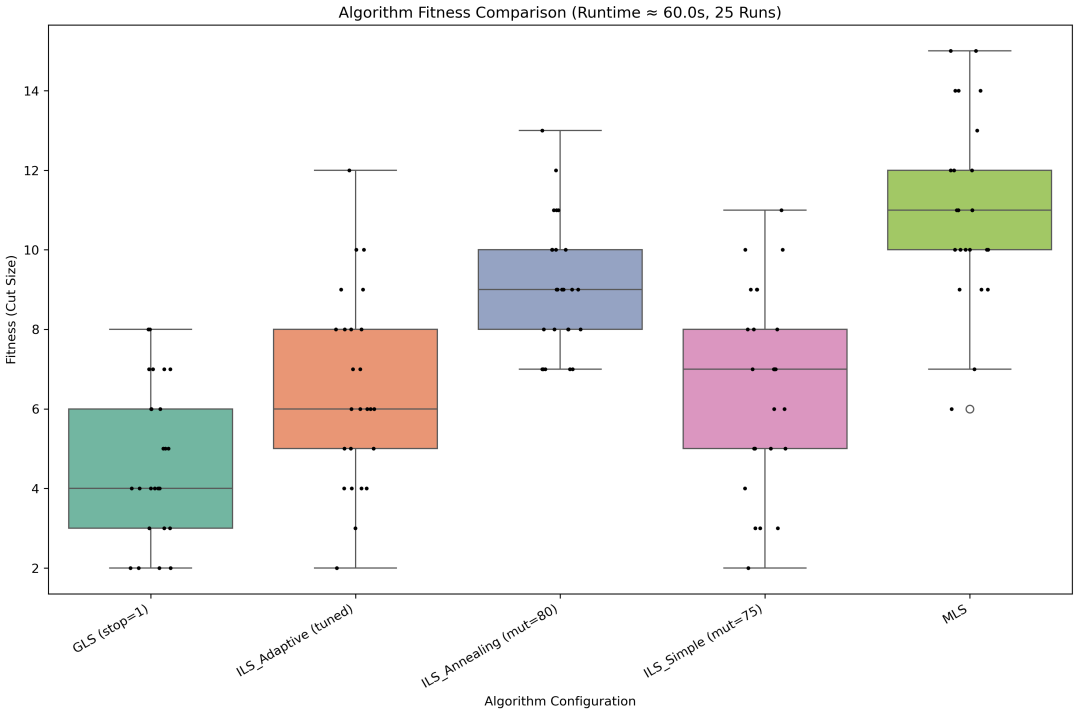

Figure 8: Comparison of algorithm performance under fixed runtime budget ($\approx$ 60s).

4 Discussion

The preliminary results highlight significant differences in the performance characteristics of the evaluated metaheuristics, particularly concerning the trade-off between computational effort and solution quality.

When a substantial budget (10,000 FM passes) was allowed, the iterative refinement strategies (ILS, GLS) demonstrated their ability to achieve better solution quality than simple MLS restarts, aligning with expectations. MLS, lacking memory, likely plateaued relatively quickly. Both standard ILS and ILS-SA showed sensitivity to mutation size, with an intermediate value (around 40-60) appearing optimal (Figures 2-4, Figure 6). This confirms the importance of balancing exploration and exploitation via perturbation strength. The comparable best median performance of ILS-SA (mutation size = 40) and Simple ILS (mutation size = 60) suggests that SA's probabilistic acceptance can effectively navigate the search space, similar to a well-tuned standard ILS. GLS also performed competitively (Figure 5), benefiting from its population-based approach. The lack of significant difference among the top four methods (GLS, ILS-SA, Simple ILS, ILS-A) in Figure 7 might stem from the limited number of runs (10), the specific graph structure allowing convergence to similar optima within the pass limit, or the preliminary nature of the ILS-A parameters.

The runtime-constrained results (Figure 8) dramatically reversed the ranking. MLS's superior performance likely results from its low overhead, enabling numerous rapid searches that quickly find decent solutions within a short time frame. Conversely, the setup and per-iteration costs of ILS and GLS probably limited their total exploration, preventing convergence to the better solutions seen in the pass-based tests. ILS-Adaptive's strong performance relative to other iterative methods here suggests its dynamic

parameter adjustment might be advantageous under tight time limits, potentially converging on effective mutation strengths faster than fixed-parameter approaches or GLS's population dynamics.

ILS-SA showed promise in the pass-based setting, matching the best Simple ILS, suggesting it could offer robustness if its parameters are well-tuned. ILS-Adaptive performed notably well under time constraints, indicating the potential benefits of self-adaptation, although its optimal configuration requires the completed grid search.

These interpretations are preliminary, primarily limited by the incomplete grid search for ILS-SA and ILS-A and the use of only a single graph instance (Graph500.txt). Performance may vary on different graph types. The pass-based comparison would benefit from more runs (e.g., 25 or 30) for greater statistical power. The exceptionally low best fitness of 2 (Figure 1) warrants investigation—it could be an artifact of the graph's structure or require further verification. Future work should prioritize completing the grid search, evaluating performance on a diverse benchmark set of graphs, increasing the number of runs, and investigating the low fitness result. Exploring different parameter settings for GLS or hybridizing GLS with adaptive or SA-based concepts could also be fruitful directions.

5 Conclusion

This study implemented and evaluated five metaheuristics—MLS, ILS, ILS-SA, ILS-A, and GLS—using the FM heuristic to solve the Graph Bipartitioning Problem on a 500-vertex instance. Our experiments covered both fixed FM pass limits and runtime-constrained settings.

In the pass-based setting, ILS and GLS consistently outperformed MLS. ILS-SA and ILS-A demonstrated competitive performance, with Simple ILS (mutation size = 60) and ILS-SA (mutation size = 40) achieving the lowest median fitness values. The differences among these methods were not statistically significant, but all were significantly better than MLS (p ¡ 0.001).

In contrast, under a strict runtime budget, MLS emerged as the best performer, likely due to its low overhead. ILS-A was the only iterative method to significantly outperform others in this setting, suggesting self-adaptation mechanisms offer advantages under limited time.

These findings underscore a fundamental trade-off: sophisticated algorithms like GLS and tuned ILS variants achieve high-quality solutions with sufficient budget, while simpler methods like MLS may excel in constrained environments. ILS-A presents a compelling middle ground, offering adaptability and robustness across different settings.

Future work should complete the parameter optimization for ILS-SA and ILS-A, expand experiments to include diverse graph types, and increase the number of runs for improved statistical confidence. Investigating hybrid approaches that combine the strengths of ILS, SA, and GLS could also be promising.

References

[1] Teofilo F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, May 2007. ISBN: 9780429143793. DOI: 10.1201/9781420010749.

[2] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. ISBN: 0-7167-1045-5.

[3] C. M. Fiduccia and R. M. Mattheyses. "A linear-time heuristic for improving network partitions". In: *19th IEEE Design Automation Conference*. 1982, pp. 175–181. DOI: 10.1109/DAC.1982.1694817.

[4] Christian Blum and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308. DOI: 10.1145/954339.954340.

[5] Helio R. Lourenço, Olivier Martin, and Thomas Stützle. "Iterated local search". In: *Handbook of Metaheuristics*. Springer, 2003, pp. 320–353. DOI: 10.1007/0-306-48056-5_3.

[6] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[7] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[8] Jin-Kao Hao et al. "An Iterated Local Search Algorithm with an Adaptive Perturbation Operator for the Graph Coloring Problem". In: *European Journal of Operational Research* 212.3 (2011), pp. 437–446. DOI: 10.1016/j.ejor.2010.07.034.

[9] Colin R. Reeves. "Metaheuristics: Past, Present, and Future". In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. by Edmund K. Burke and Graham Kendall. Springer US, 2010, pp. 3–18.

[10] Thomas Stützle. "Iterated Local Search: Framework and Applications". In: *Encyclopedia of Algorithms*. Ed. by Ming-Yang Kao. Springer International Publishing, 2018, pp. 1–6.

[11] Irina Koch, Elena Mikhailova, and Alexander Plyasunov. "Balanced Crossover for Graph Partitioning". In: *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Vol. 5483. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 73–82. DOI: 10.1007/978-3-642-01129-0_8.

[12] Zbigniew Michalewicz and Marc Schoenauer. "Integrating Local Search into Genetic Algorithms". In: *Evolutionary Computation*. Ed. by Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. CRC Press, 1996, pp. 31–53.

[13] Hatem Masmoudi, Mohamed Ali Aloulou, and Adel Aloui. "A hybrid ILS-SA approach to the vehicle routing problem with occasional drivers". In: *Journal of Heuristics* 22.4 (2016), pp. 511–534. DOI: 10.1007/s10732-015-9296-3.

[14] Seyed Mohsen Seyedhosseini, Rajarshi Roy, and Iman Nodooshan. "Performance Analysis of Simulated Annealing Using Adaptive Markov Chain Length for Solving Traveling Salesman Problem". In: *International Journal of Industrial Engineering Computations* 3.2 (2012), pp. 267–278. DOI: 10.5267/j.ijiec.2011.07.006.

[15] Dirk V. Arnold and Hans-Georg Beyer. "Stagnation Detection Meets Fast Mutation in Evolutionary Algorithms". In: (2000). arXiv: cs/0003024.

[16] Guilherme H. Travassos, Forrest Shull, and Malcolm V. Zelkowitz. "Experimental Design in Software Engineering: A Systematic Mapping Study". In: *Journal of Systems and Software* 64.4 (2002), pp. 407–425. DOI: 10.1016/S0164-1212(01)00186-1.

[17] Sidney Siegel and N. John Castellan Jr. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, 1988. ISBN: 0-07-057357-3.