Max-Cut Algorithm Report

MD MEHEDI HASAN MIM
ID: 2105142

May 12, 2025

1 Introduction

The Max-Cut problem is a classical NP-hard combinatorial optimization problem that involves partitioning the vertices of a graph into two disjoint subsets such that the sum of the weights of the edges between the subsets is maximized. In this report, we explore and evaluate five heuristic algorithms to approximate solutions to this problem across 54 benchmark graphs.

2 Algorithm Descriptions

2.1 Randomized Max-Cut

This algorithm assigns each vertex randomly to one of the two sets and calculates the resulting cut weight. It is repeated multiple times (e.g., 100 iterations), and the best result is selected. This method is simple but can vary widely in result quality.

2.2 Greedy Max-Cut

The greedy algorithm initializes the cut with the edge of maximum weight, assigning its endpoints to different sets. It then iteratively places each unassigned vertex to the side that maximizes the gain in cut weight. It is efficient but often gets stuck in local optima.

2.3 Semi-Greedy Max-Cut

This approach uses a value-based restricted candidate list (RCL). For each unassigned vertex, a greedy score is computed based on its edge weights to the current partitions. Candidates with a score greater than a threshold $\mu = w_{\min} + \alpha(w_{\max} - w_{\min})$ are added to the RCL, from which a vertex is selected randomly. This balances greediness and randomness.

2.4 Local Search

Local search improves a given solution by moving a vertex to the opposite subset if it increases the cut weight. The process is repeated until no further improvement is possible. This method is essential for refining solutions obtained from constructive heuristics.

2.5 GRASP (Greedy Randomized Adaptive Search Procedure)

GRASP is a multi-start metaheuristic where each iteration involves a semi-greedy construction followed by local search. The best solution across all iterations is retained. This method combines diversification and intensification effectively.

3 Experimental Setup

All algorithms were implemented in C++17. Graphs were read from files in the format: number of vertices, number of edges, followed by edge list with weights. Each algorithm was tested across 54 benchmark graphs. For timing and iterations:

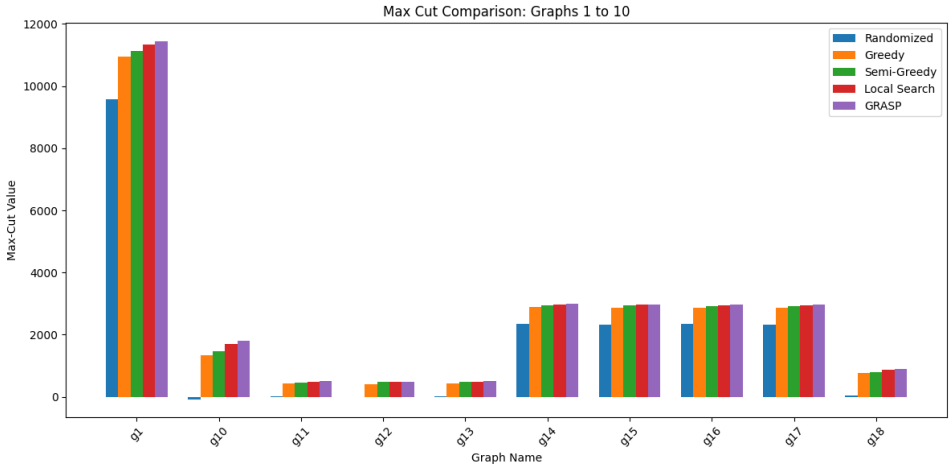
- Randomized: 100 iterations
- GRASP: 50 iterations
- Alpha for semi-greedy: 0.5

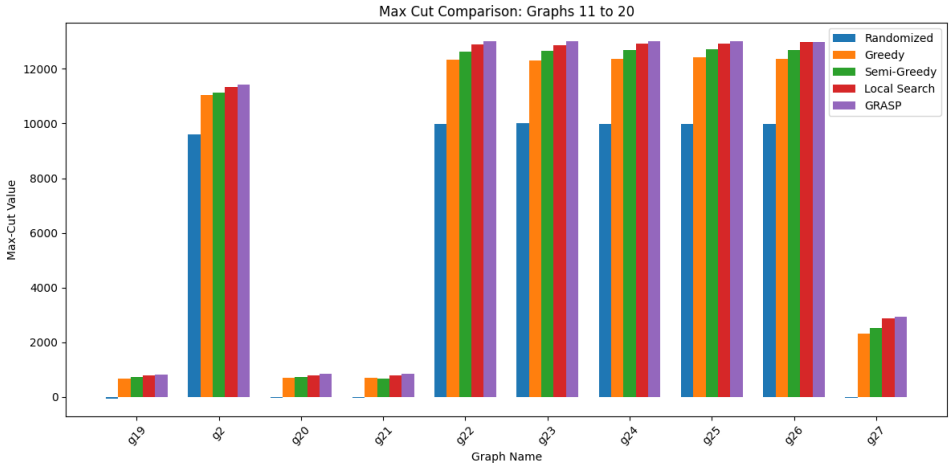
Results were saved into a CSV file named `2105142.csv`.

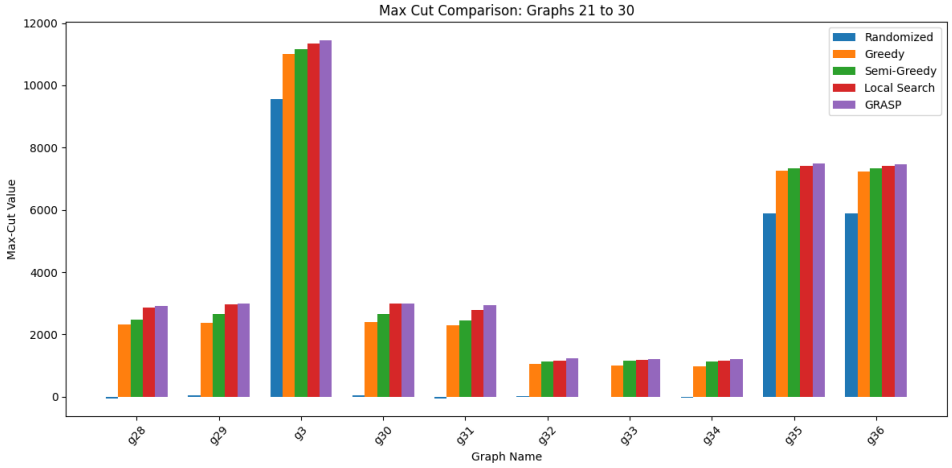
4 Performance Comparison

- **GRASP** provided the highest cut values consistently.
- **Greedy** was fast but often produced suboptimal cuts.
- **Semi-Greedy** outperformed pure greedy due to RCL.
- **Randomized** results were inconsistent but fast.
- **Local Search** greatly improved semi-greedy base solutions.

5 Plots and Visualizations


Max Cut Comparison: Graphs 1 to 10
Randomized
Greedy
Semi-Greedy
Local Search
GRASP
Max-Cut Value
0
2000
4000
6000
8000
10000
12000
g1
g10
g11
g12
g13
g14
g15
g16
g17
g18
Graph Name


Max Cut Comparison: Graphs 11 to 20
Randomized
Greedy
Semi-Greedy
Local Search
GRASP
Max-Cut Value
0
2000
4000
6000
8000
10000
12000
g19
g2
g20
g21
g22
g23
g24
g25
g26
g27
Graph Name


Max Cut Comparison: Graphs 21 to 30
Randomized
Greedy
Semi-Greedy
Local Search
GRASP
Max-Cut Value
0
2000
4000
6000
8000
10000
12000
g28
g29
g3
g30
g31
g32
g33
g34
g35
g36
Graph Name

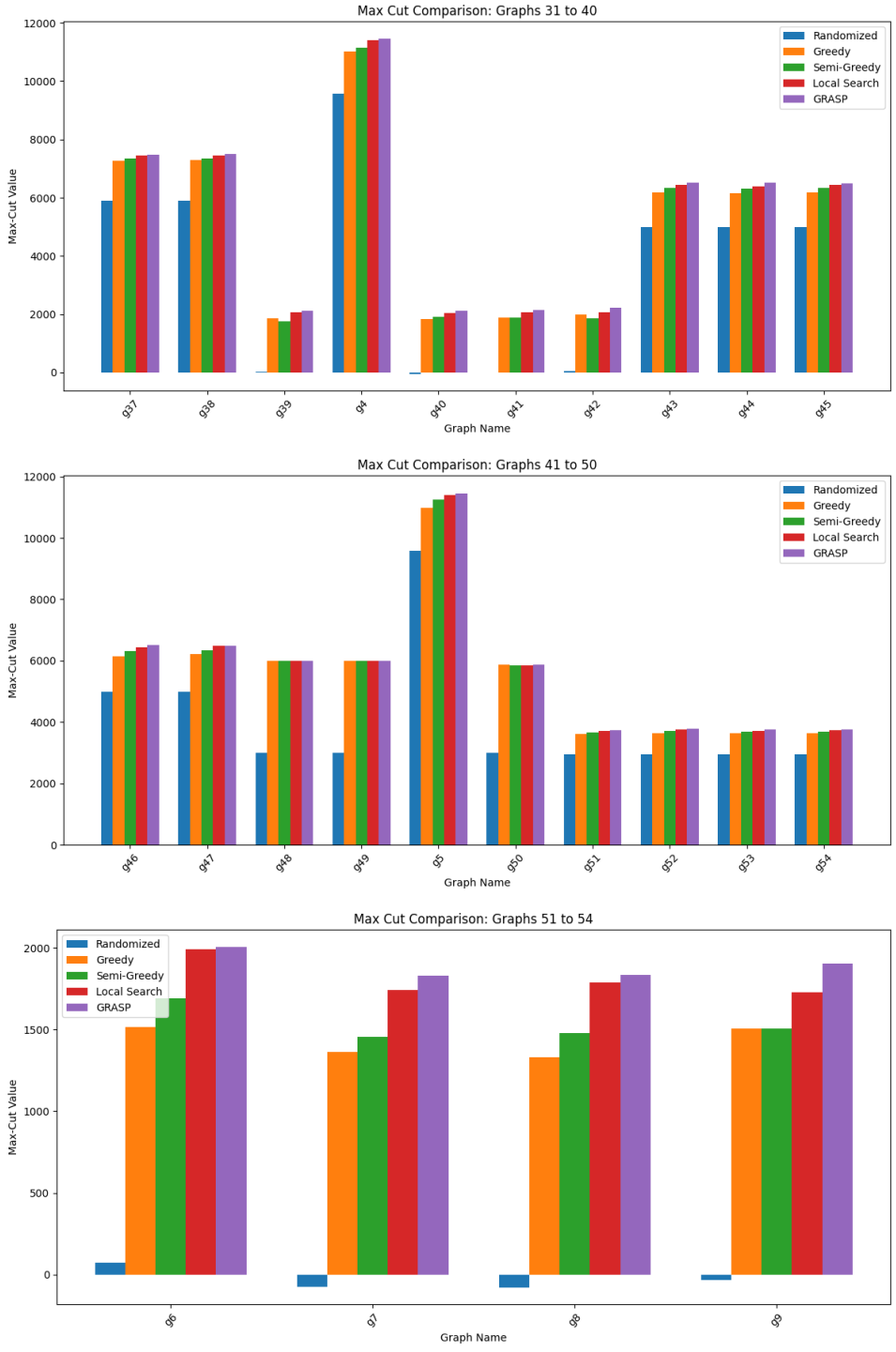


Figure 1: Max-Cut values for Graphs 1–10 by algorithm

6 Conclusion

This report demonstrates the effectiveness of combining greedy construction with local refinement. GRASP emerged as the most reliable method. Semi-greedy with value-based RCL offers a strong starting point, and local search ensures improvement. Randomized serves as a useful baseline.