Solving the Maximum Cut Problem using GRASP

Rageeb Hasan Shafee (2105175)

May 12, 2025

1 Introduction

The Maximum Cut (MAX-CUT) problem is a classical combinatorial optimization problem. Given an undirected graph $G = (V, U)$, where V represents vertices and U represents edges with weights, the task is to find a non-empty subset of vertices $S \subset V$, such that the weight of the cut $(S, S^{)}$, defined as the sum of the edge weights between the vertices in S and those in S, is maximized. This problem is NP-hard and has applications in various fields such as VLSI design, statistical physics, and machine learning.

2 Heuristic Methods

We have implemented several heuristic methods to approximate the solution for the MAX-CUT problem: Randomized, Greedy, Semi-Greedy, Local Search, and GRASP. Below is a detailed explanation of each heuristic.

2.1 Randomized Heuristic

The Randomized Heuristic starts by randomly assigning each vertex in the graph to one of two partitions, X or Y, with equal probability. The cut weight is then calculated by summing the weights of the edges that cross between the two partitions. The algorithm is run multiple times, and the average cut weight is taken as the result. This heuristic provides a simple approach but does not guarantee a good solution.

2.2 Greedy Heuristic

The Greedy heuristic improves upon the Randomized heuristic by iteratively assigning each unassigned vertex to the partition that maximizes the cut weight. Initially, the first two vertices are assigned to the partitions, and subsequent vertices are placed in the partition that maximizes the cut weight, considering the edges to already-assigned vertices.

2.3 Semi-Greedy Heuristic

The Semi-Greedy heuristic combines greediness with randomness. It builds on a standard greedy function but introduces an element of randomness in the selection process.

The algorithm begins by calculating a greedy function for each vertex, which measures the potential contribution of the vertex to the current cut. Based on these values, a Restricted Candidate List (RCL) is created. The RCL is a subset of candidate vertices that are ranked by their greedy function values. Only vertices within the RCL are considered for inclusion in the solution.

There are two main methods for constructing the RCL: - Cardinality-based method: Selects the top-k candidates (e.g., the 5 best candidates). - Value-based method: Selects vertices whose greedy function values are above a threshold, μ, defined as:

$$\mu = w_{\min} + \alpha \cdot (w_{\max} - w_{\min})$$

where $w_{\min}$ and $w_{\max}$ are the minimum and maximum greedy function values across all unassigned vertices, and α is a tunable parameter ($0 \leq \alpha \leq 1$).

Once the RCL is constructed, a vertex is randomly selected from it and assigned to one of the two partitions, either X or Y. This randomness allows the algorithm to explore different solutions and avoid getting stuck in local optima, which is a key advantage of the Semi-Greedy heuristic over the standard Greedy approach.

2.4 Local Search

Local Search further refines the solutions produced by the previous heuristics. Here it has applied on graph applying *Randomized Heuristic*. The algorithm examines all possible single vertex swaps between the two partitions. If swapping a vertex results in a better cut weight (i.e., increases the weight of the cut), the swap is performed. This process continues until no further improvements can be made.

2.5 GRASP (Greedy Randomized Adaptive Search Procedure)

GRASP combines the Semi-Greedy heuristic with Local Search in an iterative process. In each iteration, a solution is constructed using the Semi-Greedy heuristic, followed by a Local Search phase to refine the solution. The best solution found across all iterations is selected as the final result.

3 Experimental Results

The heuristics were tested on a set of benchmark graphs. For each graph, we measured the performance of the Randomized, Greedy, Semi-Greedy, Local

Search, and GRASP algorithms in terms of the cut weight achieved. The results were compared against known best solutions.

Problem	Known Best Solution	Problem	Known Best Solution	Problem	Known Best Solution	Problem	Known Best Solution
G1	12078	G14	3187	G32	1560	G43	7027
G2	12084	G15	3169	G33	1537	G44	7022
G3	12077	G16	3172	G34	1541	G45	7020
G11	627	G22	14123	G35	8000	G48	6000
G12	621	G23	14129	G36	7996	G49	6000
G13	645	G24	14131	G37	8009	G50	5988

Figure 1: Known best solutions or upper bounds for selected benchmark instances.

3.1 Comparison of Algorithms

Figure 2 shows the comparison of the algorithms on the benchmark dataset. As can be seen, the Randomized algorithm generally performs the worst, followed by Greedy and Semi-Greedy. The GRASP algorithm consistently performs the best, approaching or achieving the known optimal solutions.

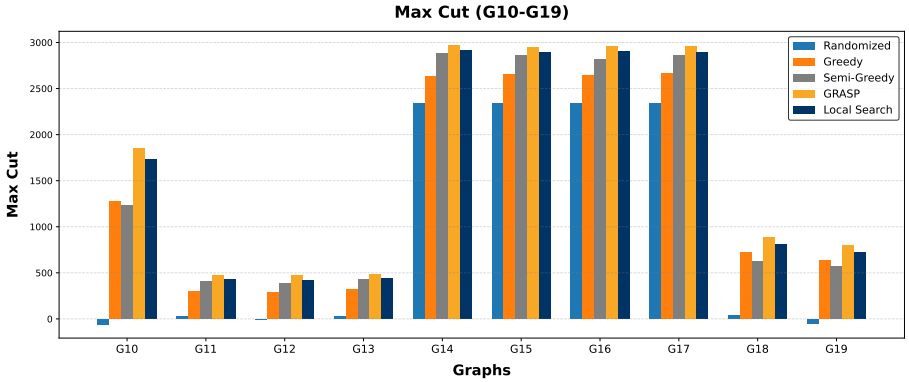

Figure 2: Performance comparison of the algorithms.

4 Analysis of Results

The performance of the Randomized, Greedy, Semi-Greedy, and GRASP algorithms shows a clear trend. The Randomized algorithm, by nature, produces solutions that are less reliable and often far from optimal. Greedy and Semi-Greedy provide better results by considering the local structure of the graph, but they still struggle to find the optimal cut. GRASP, by combining the semi-greedy approach with local search, converges more quickly to high-quality solutions, often achieving results close to the known best solutions.

5 Conclusion

In this report, we implemented several heuristic methods to solve the Maximum Cut problem, with GRASP showing the best performance. By combining randomized construction with local search, GRASP is able to achieve near-optimal solutions. The results demonstrate the effectiveness of GRASP in solving combinatorial optimization problems and highlight the importance of iterative refinement methods in reaching high-quality solutions.