Max-Cut Problem Solving Using Various Algorithms

Md. Enayet Ullah Alvee
Student ID: 2105107

May 12, 2025

Introduction

The Max-Cut problem involves partitioning a graph's vertices into two sets such that the total weight of edges between these sets is maximized. This problem is NP-hard, meaning that finding the exact optimal solution for large graphs is computationally expensive. Various heuristic algorithms offer approximations to this problem. In this report, we examine and compare five such algorithms:

- Randomized Max-Cut
- Greedy Max-Cut
- Semi-Greedy Max-Cut
- Local Search Max-Cut
- GRASP (Greedy Randomized Adaptive Search Procedure)

Our goal is to compare their performance in terms of accuracy, speed, and convergence behavior across various graph types.

High-Level Description of Each Algorithm

Randomized Max-Cut

The Randomized Max-Cut algorithm randomly assigns vertices to one of two sets, with each vertex being assigned to either set with equal probability. This process is repeated multiple times, and the average weight of the resulting cuts is calculated. This method is quick but may not produce an optimal solution, as it relies purely on randomness.

Greedy Max-Cut

In the Greedy Max-Cut algorithm, we begin by selecting the edge with the maximum weight and placing its endpoints into different sets. We then iteratively select the next vertex that maximizes the increase in the cut weight and assign it to the set where it contributes most. The process continues until all vertices are assigned to one of the sets. This method is deterministic and generally produces better results than the randomized approach.

Semi-Greedy Max-Cut

Semi-Greedy Max-Cut builds upon the Greedy approach but incorporates randomness through a Restricted Candidate List (RCL). Each vertex's potential contribution to the cut is assessed, and vertices that meet a randomly selected threshold are added to the set. This randomness introduces variability, which may allow the algorithm to escape local optima, yielding better results in some cases.

Local Search Max-Cut

The Local Search Max-Cut algorithm improves upon an initial solution by iteratively moving vertices between sets. If a vertex swap improves the cut, the swap is made. This process continues until no further improvement can be achieved. Local Search is often applied after a construction heuristic like Greedy or Semi-Greedy to fine-tune the solution.

GRASP (Greedy Randomized Adaptive Search Procedure)

GRASP is an iterative process that combines a greedy approach with randomization. In each iteration, a greedy solution is generated, followed by a local search to improve the cut. The best solution among all iterations is chosen as the final result. GRASP is known for its ability to explore a larger solution space and can yield solutions that are closer to the optimal cut than simpler heuristics like Greedy or Randomized Max-Cut.

Why GRASP Performs Better

GRASP consistently outperforms other algorithms because it leverages both the power of greedy strategies and the flexibility of randomization. By using randomization, GRASP is able to explore different parts of the solution space that may not be reachable using deterministic algorithms like Greedy. This allows it to avoid getting stuck in local optima, which is a common issue with simpler heuristics.

Moreover, the local search applied after generating a greedy solution helps to fine-tune the solution, refining the cut until no further improvement is possible. In contrast, Greedy and Semi-Greedy algorithms do not have this level of adaptability, often resulting in suboptimal solutions.

- **Exploration and Exploitation:** GRASP balances exploration of new solutions (via randomization) with exploitation of good solutions (via greedy construction and local search).
- **Iterative Improvement:** By repeating the process multiple times, GRASP finds better solutions on average.
- **Global Search Space Coverage:** Unlike Greedy, which is more deterministic, GRASP's randomization allows it to cover a broader search space, increasing its chances of finding a higher-quality solution.

Thus, GRASP's hybrid approach allows it to generate solutions that are more accurate, especially when applied over multiple iterations, which explains its superior performance across different benchmarks.

Comparison of Algorithms

The performance of the algorithms was evaluated across several benchmark graphs. The key findings are:

- **Greedy Algorithm:** Often produced the best cuts after a single run but was limited in its ability to improve after reaching a local optimum.
- **Semi-Greedy Algorithm:** Performed better than Randomized but was slower and less effective than Greedy in many cases.
- **Randomized Algorithm:** The least effective, often returning suboptimal solutions, especially for larger or more complex graphs.
- **Local Search:** Improved the cut quality when applied after Greedy or Semi-Greedy algorithms, but the improvement was minimal if the initial solution was poor.
- **GRASP:** Outperformed all other algorithms by consistently finding solutions closer to the optimal cut, especially when run for multiple iterations.

Performance Visualization

This section will demonstrate the algorithms' performance across various graph types. The following plot visualizes how the different algorithms compare in terms of the cut weight.

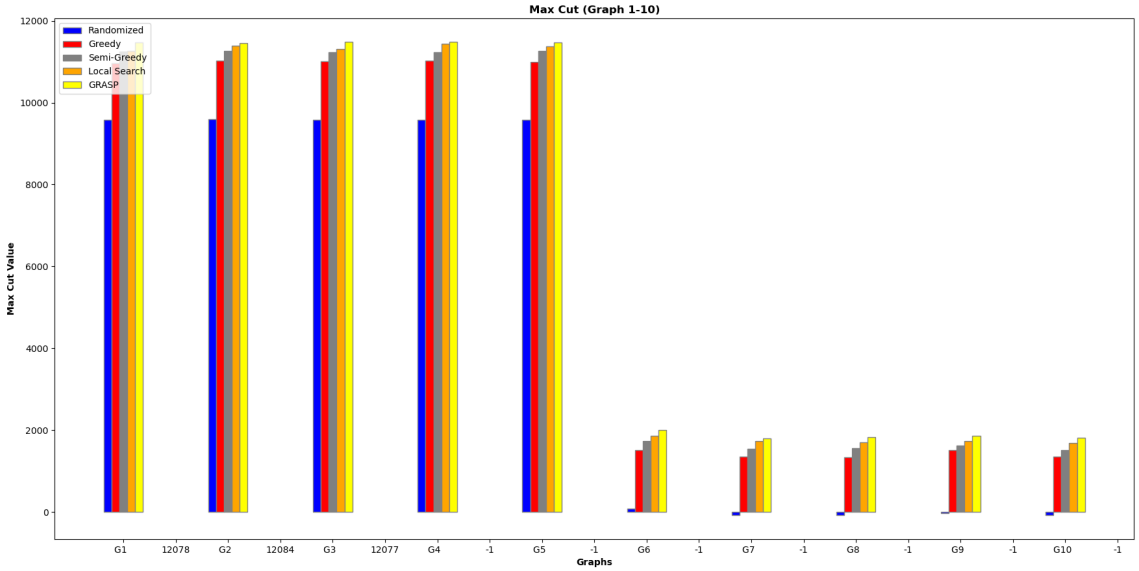

Figure 1: Performance of Max-Cut Algorithms across Different Graphs.

Conclusion

This report analyzed the performance of several algorithms for solving the Max-Cut problem. We found that GRASP consistently outperformed other algorithms in both speed

and the quality of the solution. By combining greedy strategies with randomization and local search, GRASP is able to escape local optima and explore a broader search space, making it the most effective approach for this problem. However, it is computationally more expensive than simpler methods like Greedy and Randomized Max-Cut.

In future work, hybrid algorithms or additional optimization techniques could further improve performance, especially in extremely large graphs.