CSE 318 Assignment-02

Solving the Max-Cut Problem

Student ID: 2105161

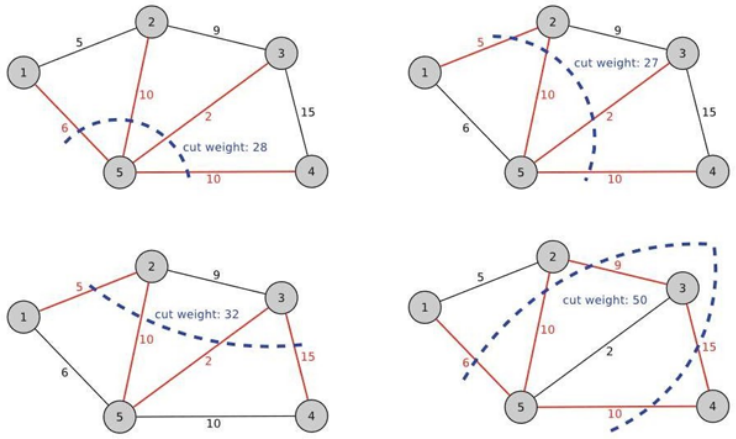
1 Introduction

The **Maximum Cut (Max-Cut)** problem is a fundamental problem in graph theory and combinatorial optimization. Given an undirected graph $G = (V, E)$ with edge weights, the goal is to partition the vertex set V into two disjoint subsets X and Y such that the total weight of edges between the two sets is maximized.

Formally, for a graph $G = (V, E)$, we want to maximize:

$$\text{Cut}(X, Y) = \sum_{\substack{(u,v)\in E \\ u\in X, v\in Y}} w(u, v)$$

This problem is known to be NP-Hard, so heuristic or approximate algorithms are often used for large instances.



Example of the maximum cut problem on a graph with five vertices and seven edges.
Four cuts are shown. The maximum cut is $(S, \bar{S}) = (\{1, 2, 4\}, \{3, 5\})$ and has a weight $w(S, \bar{S}) = 50$.

Figure 1: Illustration of the Max-Cut Problem

This report explores 5 heuristic algorithms to solve the Max-Cut problem efficiently on a set of 54 benchmark graphs.

2 Algorithm Descriptions

2.1 Randomized Algorithm

This is the simplest heuristic. Each node is assigned randomly to either subset X or Y. The algorithm is run multiple times, and the average cut weight is returned.

Algorithm 1 Randomized Max-Cut (averaged over n runs)

procedure RandomizedMaxCut($G = (V, E), n$)
    $totalCut \leftarrow 0$
    for $i = 1$ to n **do**
        Randomly assign each vertex to X or Y
        $cutWeight \leftarrow$ weight of edges crossing (X, Y)
        $totalCut \leftarrow totalCut + cutWeight$
    end for
    return $totalCut/n$
end procedure

2.2 Greedy Algorithm

The greedy heuristic starts by placing the heaviest edge's endpoints into different sets. Then, remaining vertices are added one by one to maximize the current cut value.

Algorithm 2 Greedy Max-Cut

procedure GreedyMaxCut($G = (V, E)$)
    Initialize X, Y with endpoints of max-weight edge
    for each unassigned vertex v **do**
        Compute w_X and w_Y as edge weight sums to X and Y
        if $w_X > w_Y$ **then**
            Assign v to Y
        else
            Assign v to X
        end if
    end for
    return final assignment and cut weight
end procedure

2.3 Semi-Greedy Algorithm (Value-Based RCL)

A hybrid approach that introduces randomness into the greedy algorithm. For each unassigned vertex, a greedy value is computed, and a Restricted Candidate List (RCL) is created using:

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

A vertex from RCL is selected randomly and added to the partition that yields higher gain.

Algorithm 3 Semi-Greedy Max-Cut (Value-based RCL)

procedure SemiGreedyMaxCut($G = (V, E), \alpha$)
  Start with max-weight edge endpoints in X and Y
  while unassigned vertices remain **do**
    Compute $\sigma_X(v)$ and $\sigma_Y(v)$ for candidates
    $greedyScore(v) \leftarrow \max(\sigma_X, \sigma_Y)$
    Determine w_{min}, w_{max} among scores
    $\mu \leftarrow w_{min} + \alpha \cdot (w_{max} - w_{min})$
    Build RCL: vertices with score $\geq \mu$
    Randomly choose from RCL and assign to partition
  end while
  return final assignment and cut value
end procedure

2.4 Local Search

This algorithm iteratively improves a solution by moving a single vertex to the opposite partition if it improves the cut. It stops when no improvement is possible.

Algorithm 4 Local Search

procedure LocalSearch($G = (V, E), initialAssignment$)
  while improving neighbor exists **do**
    for each vertex v **do**
      Simulate flipping v's partition
      if cut value increases **then**
        Apply the move
        Break loop
      end if
    end for
  end while
  return locally optimized assignment
end procedure

2.5 GRASP (Greedy Randomized Adaptive Search Procedure)

GRASP repeatedly constructs a solution using Semi-Greedy heuristic and refines it via Local Search. The best solution over all iterations is returned.

Algorithm 5 GRASP

procedure GRASP($G = (V, E), maxIter, \alpha$)
  for $i = 1$ to $maxIter$ **do**
    $x \leftarrow$ SemiGreedyMaxCut(G, α)
    $x \leftarrow$ LocalSearch(x)
    if x better than current best **then**
      Update best solution
    end if
  end for
  return best solution found
end procedure

3 Performance Comparison

We evaluated all algorithms over 54 benchmark graphs. Metrics such as average cut weight, iteration count, and best known solutions were recorded. The CSV file '2105XXX.csv' contains all results.

4 Plots and Visualizations

Below are plots comparing the performance of all five algorithms on chunks of graphs:

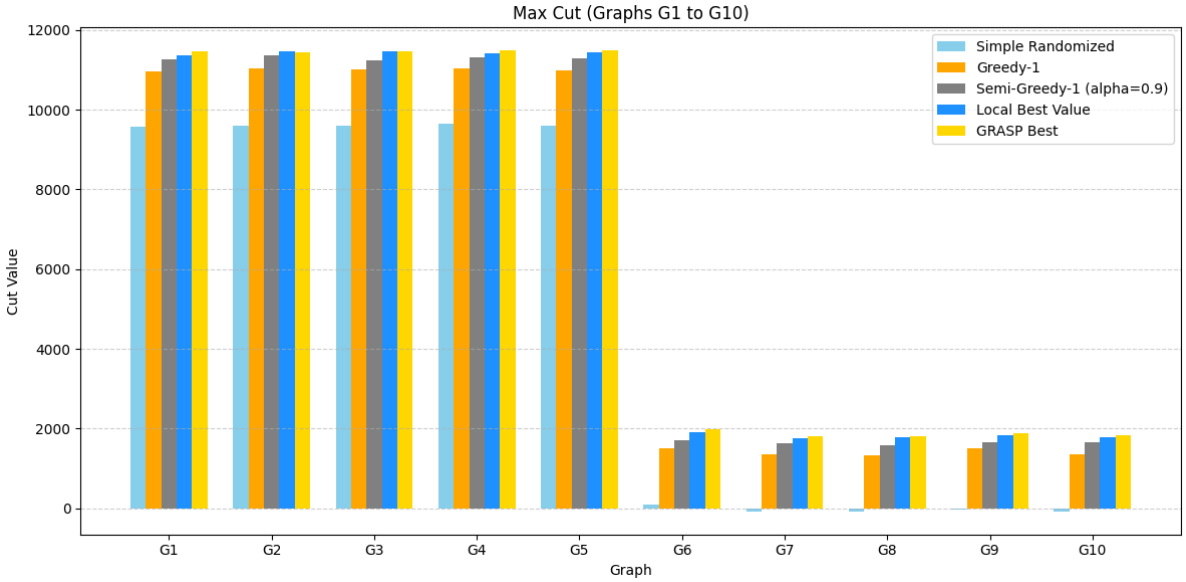


Figure 2: Graphs G1 to G10

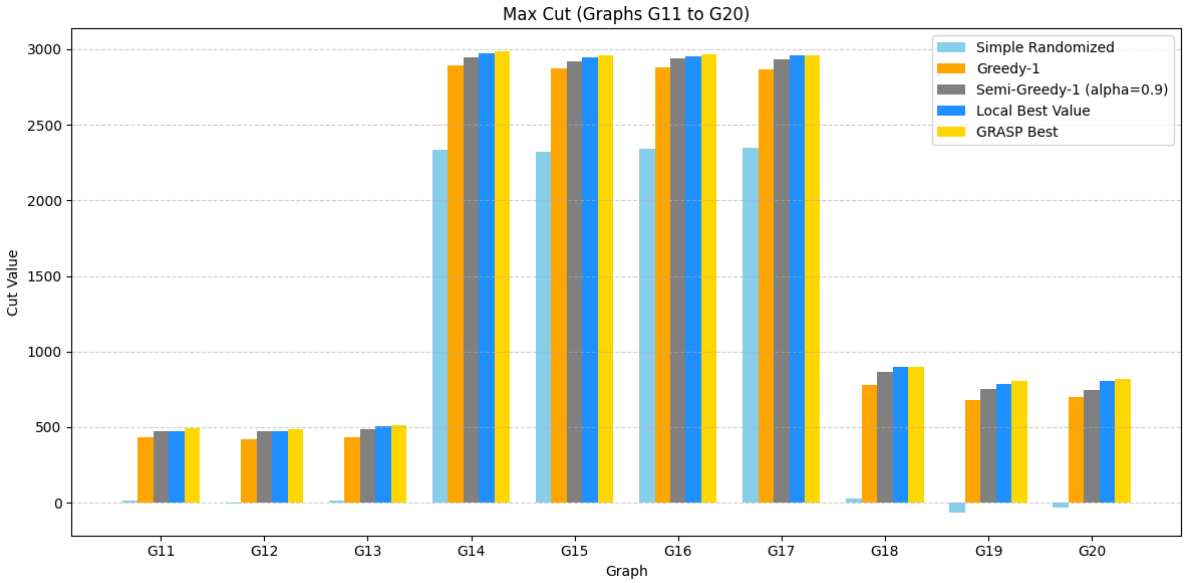


Figure 3: Graphs G11 to G20

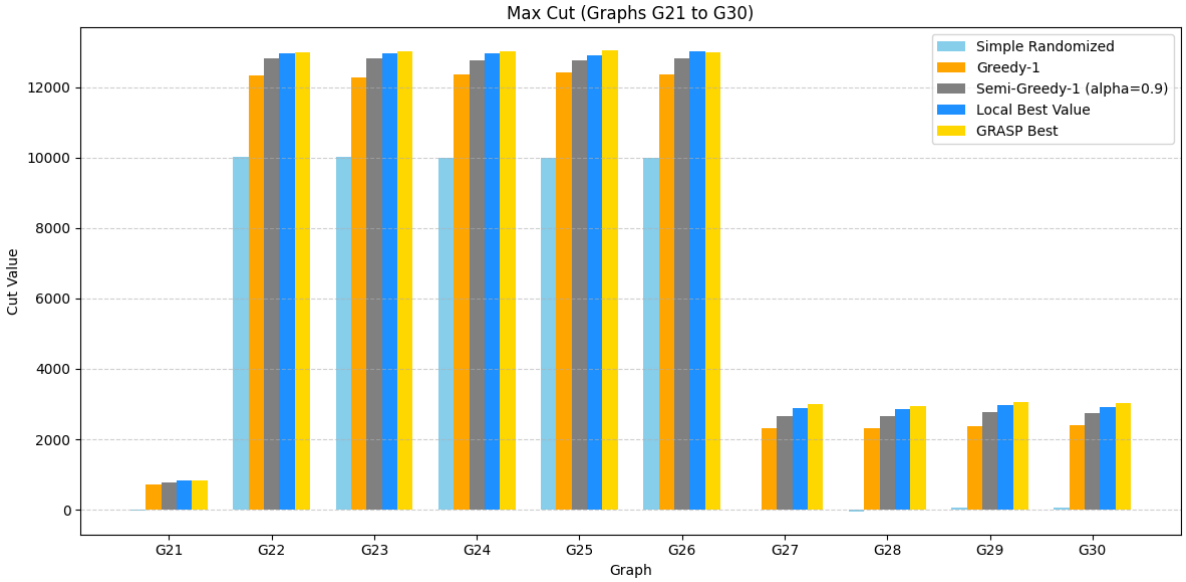


Figure 4: Graphs G21 to G30

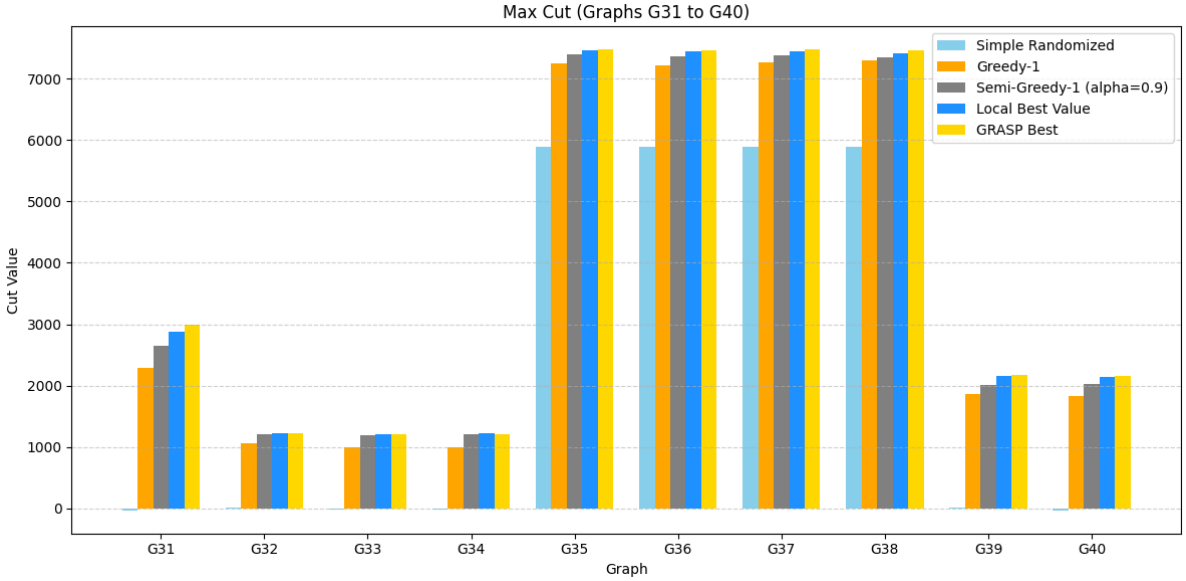


Figure 5: Graphs G31 to G40

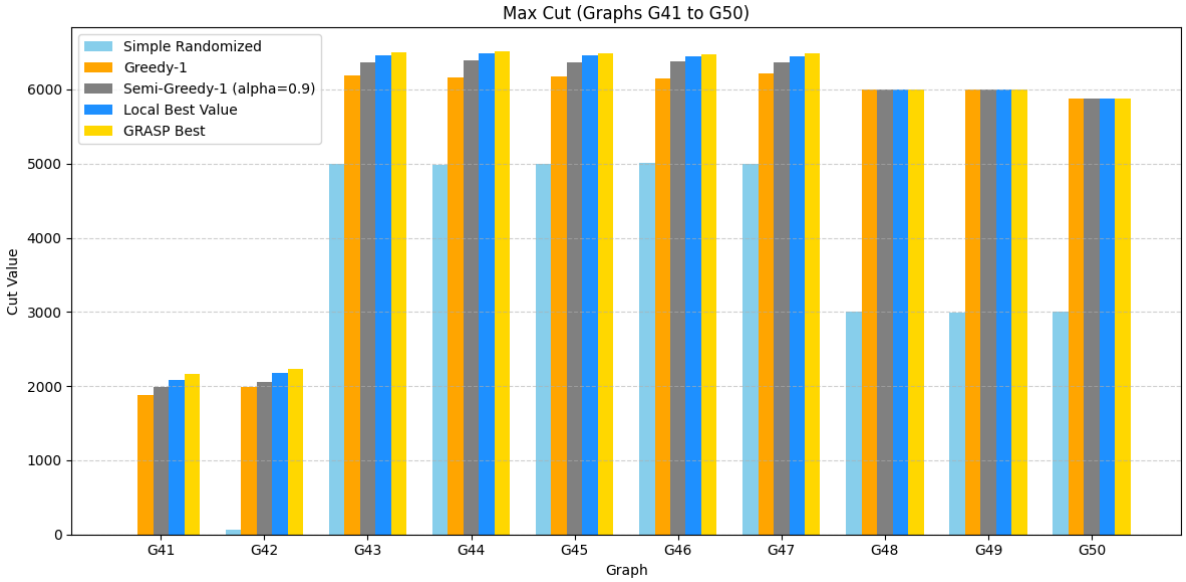


Figure 6: Graphs G41 to G50

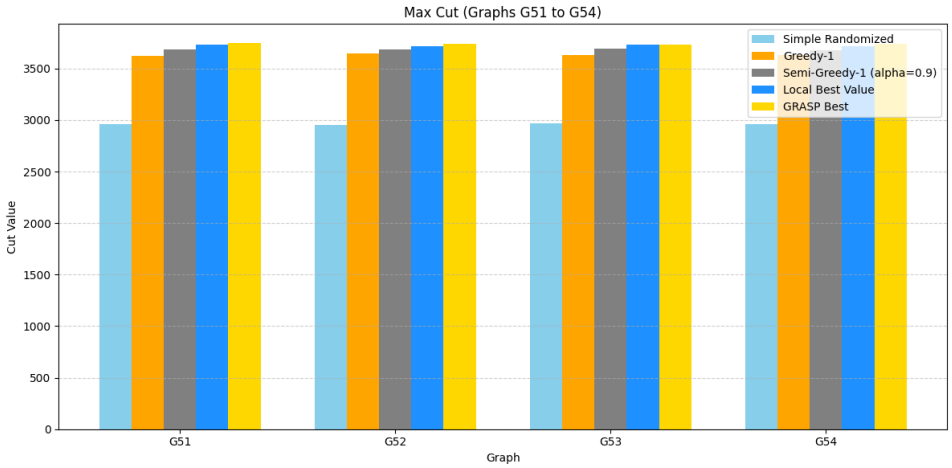


Figure 7: Graphs G51 to G54

5 Observations

- **GRASP** consistently outperforms all other heuristics, especially on large graphs.
- **Semi-Greedy** performs better than Greedy and Randomized due to its balance of exploitation and exploration.
- **Greedy** provides a decent baseline by deterministically assigning vertices based on maximum immediate gain, but it often gets stuck in local optima without further improvement.
- **Local Search** significantly improves initial solutions when applied over Greedy/Semi-Greedy assignments.
- **Randomized** performs the worst due to its lack of structure, but can provide diverse initial solutions for GRASP.

6 Conclusion

This report implemented and evaluated 5 heuristics for Max-Cut. GRASP, combining Semi-Greedy and Local Search, gave the best results across most benchmark instances.