CSE 318

Assignment-02:

Solving the Max-Cut Problem by GRASP

April 28, 2025

# Introduction

The maximum cut (MAX-CUT) problem involves an undirected graph G = (V, U), where V is the set of vertices and U is the set of edges with weights wuv. The goal of the MAX-CUT problem is to find a non-empty subset of vertices S ⊂ V such that the weight of the cut (S, S¯) is maximized. The cut weight is calculated as w(S, S¯) = Σwuv for all edges (u, v) where u ∈ S and v ∈ S¯.

# Algorithm 1: GRASP (Greedy Randomized Adaptive Search Procedure)

The GRASP algorithm consists of two phases: the construction phase, where a feasible solution is generated, and the local search phase, which aims to improve the solution by iterating and finding a locally optimal solution.

The procedure for GRASP is as follows:

1. procedure GRASP(MaxIterations)  
2. for i = 1 to MaxIterations do  
3. Build a greedy randomized solution x using semi-greedy heuristic  
4. x ← LocalSearch(x)  
5. if i = 1 then  
6. x\* ← x  
7. else if w(x) > w(x\*) then  
8. x\* ← x  
9. end if  
10. end for  
11. return x\*  
12. end procedure

# Algorithm 2: Randomized-1 Heuristic for MAX-CUT

The Randomized-1 heuristic generates a cut by randomly assigning vertices to two partitions (X and Y).

The procedure for the Randomized heuristic is as follows:

1. procedure RandomizedMaxCut(G = (V, E), w, n)  
2. totalCutWeight ← 0  
3. for i = 1 to n do  
4. Initialize partitions: X ← ∅, Y ← ∅  
5. for all vertex v ∈ V do  
6. if Random choice with probability >= 1/2 then  
7. X ← X ∪ {v}  
8. else  
9. Y ← Y ∪ {v}  
10. end if  
11. end for  
12. cutWeight ← Σ (u,v) ∈ E, u ∈ X, v ∈ Y or u ∈ Y, v ∈ X wuv  
13. totalCutWeight ← totalCutWeight + cutWeight  
14. end for  
15. averageCutWeight ← totalCutWeight / n  
16. return averageCutWeight  
17. end procedure

# Algorithm 3: Greedy Heuristic for MAX-CUT

The Greedy Heuristic starts by placing the two vertices connected by the edge with the largest weight into different partitions, and then adds the remaining vertices in a greedy manner.

The procedure for the Greedy Heuristic is as follows:

1. procedure GreedyMaxCut(G = (V, E), w)  
2. Initialize partitions: X ← ∅, Y ← ∅  
3. Find edge (u, v) ∈ E with maximum weight wuv  
4. X ← X ∪ {u}, Y ← Y ∪ {v}  
5. U ← V \ {u, v} ▷ Unassigned vertices  
6. for all vertex z ∈ U do  
7. Compute weight if z ∈ X: wX = Σ y ∈ Y wzy  
8. Compute weight if z ∈ Y: wY = Σ x ∈ X wzx  
9. if wX > wY then  
10. X ← X ∪ {z}  
11. else  
12. Y ← Y ∪ {z}  
13. end if  
14. end for  
15. return partitions X, Y  
16. end procedure

# Algorithm 4: Semi-Greedy Heuristic for MAX-CUT

The Semi-Greedy Heuristic introduces randomness in the candidate selection process by evaluating the greedy function and selecting candidates from a Restricted Candidate List (RCL). The parameter α determines the randomness.

1. procedure SemiGreedyMaxCut(G = (V, E), w, α)

2. Initialize partitions: X ← ∅, Y ← ∅

3. Evaluate greedy function for each unassigned vertex v

4. Construct a Restricted Candidate List (RCL) based on the values

5. Randomly select a vertex from the RCL

6. Move the selected vertex to the partition that maximizes the cut value

7. Repeat until all vertices are assigned to partitions

8. return partitions X, Y

9. end procedure

I have tweaked the value of alpha to find a balance between randomness and greediness to find the sweet spot for it.

# Algorithm 5: Local Search for MAX-CUT

The local search phase explores neighboring solutions by moving vertices between the two partitions, improving the solution until no better solution is found.

The procedure for local search is as follows:

1. For each vertex v ∈ V, calculate δ(v) based on the current solution  
2. Move the vertex to the partition that improves the cut weight  
3. Repeat until no improving move is found

For the sake of implementation, I have chosen the semi-greedy algorithm for the construction phase and then ran the Local Search algorithm.

# Graph Plots for Benchmark Problems

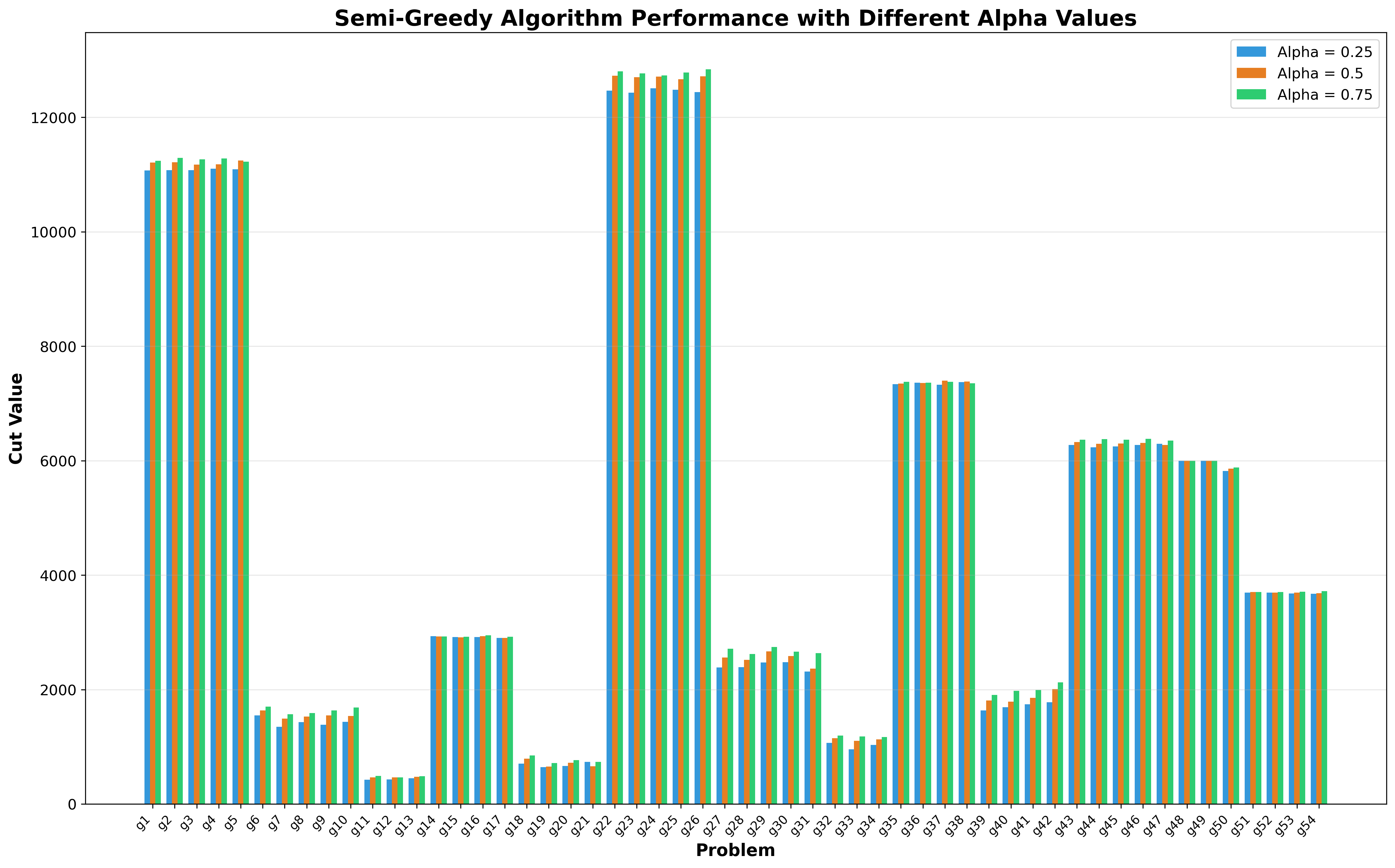
# Results

# Comparison of Algorithms

From the graphs mentioned we can easily compare the performance of various algorithms and say confidently various facts about it.

| **Algorithm** | **Performance** | **Reliability** | **Time Complexity** | **Use Case** |
| --- | --- | --- | --- | --- |
| **Randomized** | Good in simpler problems, inconsistent in complex ones | Low reliability (randomness leads to variability) | Fast, but suboptimal | Suitable for positive graphs or when quick approximate solutions are needed but underperforms in negative edged graphs. |
| **Greedy** | Good for small to medium graphs, starts to lose in complex ones | Moderate reliability (prone to local optima) | Fast | Best for simple problems with good local optima, fast and simple |
| **Semi-Greedy** | Generally, provides better performance than Greedy and Randomized | More reliable than Greedy and Randomized | Moderate (due to RCL construction) | Balanced between exploration and exploitation, works well on complex graphs |
| **Local Search** | 2nd best as it is not stuck in the a higher value but optimized to the local optima. | More reliable than semi-greedy | Moderately slow (due to iterations) | Better performance in a large graph as it has a lot of maxima as we can see by the plots. |
| **GRASP** | Best performance across most problem sets | Highly reliable, stable, robust | Slow (due to multiple iterations) | Best for high-quality solutions, especially on large and complex graphs |

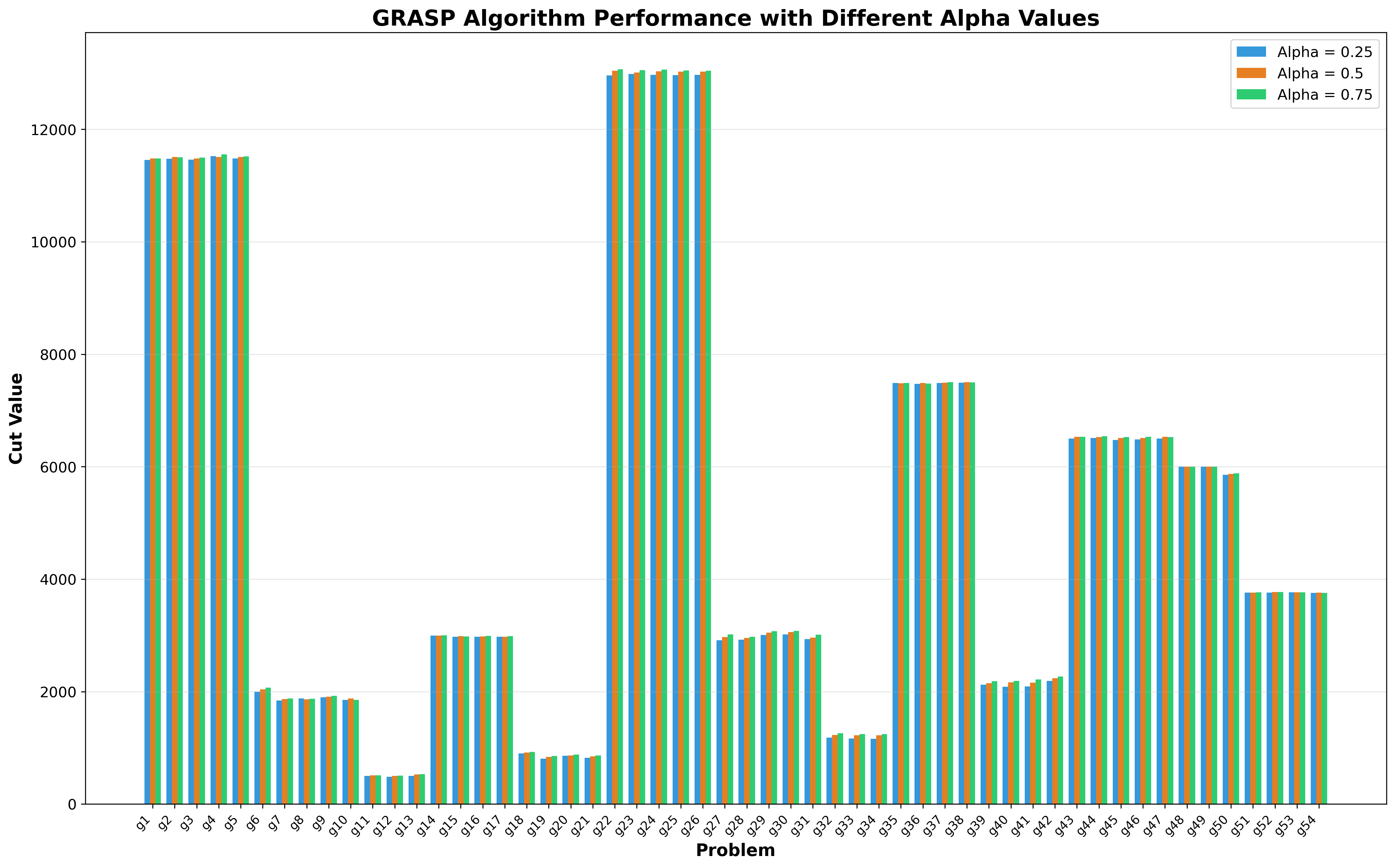
# Comparison of Alpha



### ****Effect of Different α Values****:

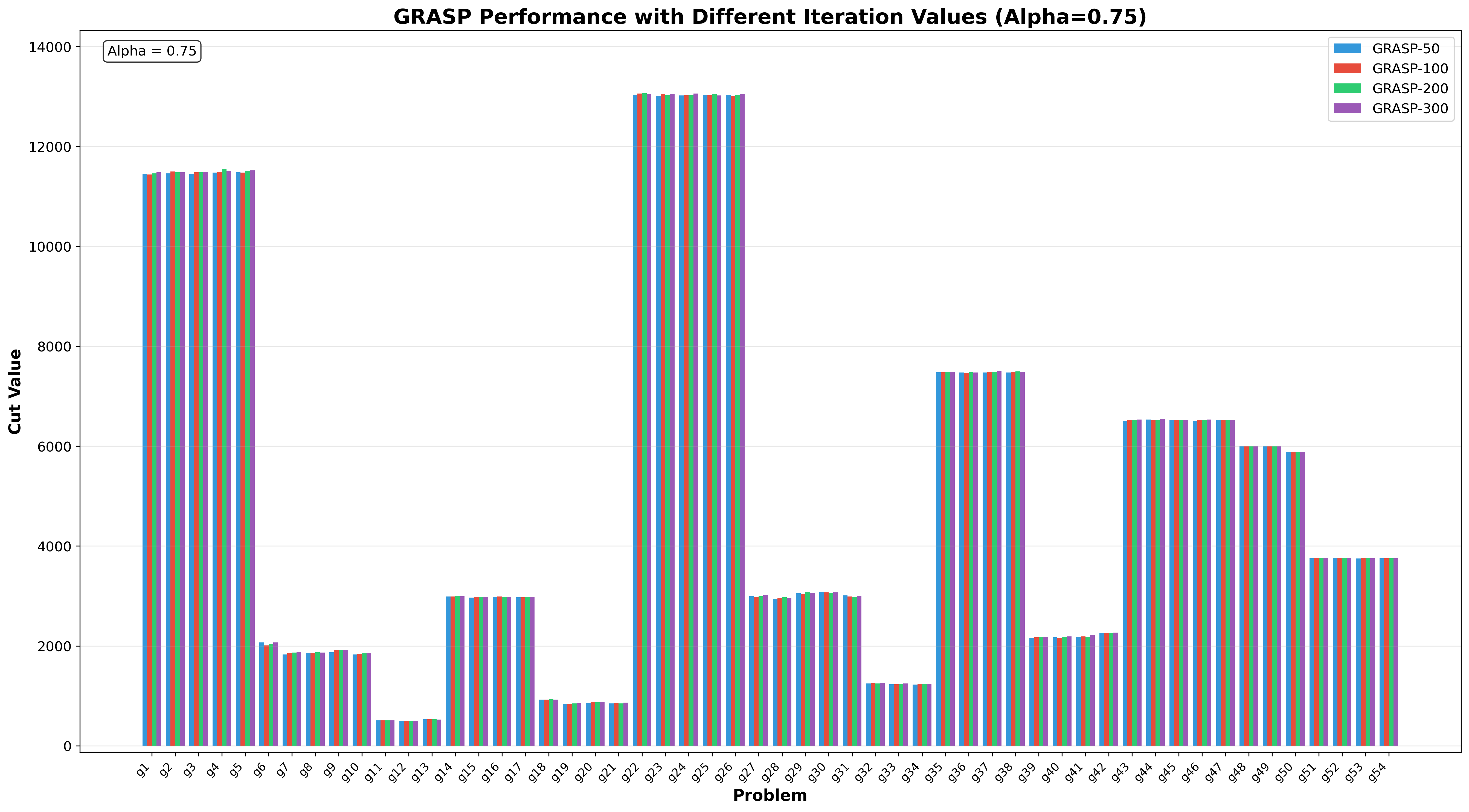
* **When α = 0.25**:
  + **RCL is large**: α = 0.25 implies that the threshold μ is much closer to **w\_min**, meaning the RCL will contain a **large number of candidates**.
  + **High randomness**: Because the RCL is large, the algorithm is more random in selecting vertices. While this provides **more exploration**, it also means that many of the candidates may not contribute much to the cut value, and the algorithm often selects **poor candidates**.
  + **Result**: The algorithm struggles to improve the cut, as it introduces too much randomness and selects suboptimal vertices.
* **When α = 0.50**:
  + **Moderate RCL size**: α = 0.50 implies μ is closer to the middle of the greedy function values, leading to an **intermediate-sized RCL**.
  + **Balanced exploration and exploitation**: With α = 0.50, the algorithm strikes a balance between exploration (randomness) and exploitation (greedy), but it’s not fully focusing on the best candidates, so it can still include **suboptimal candidates** that dilute the quality of the cut.
  + **Result**: The algorithm gives **moderate performance**, but some of the exploration still leads to suboptimal cuts, and it doesn’t fully exploit the best options available.
* **When α = 0.75**:
  + **Smaller RCL, focused selection**: α = 0.75 brings the threshold μ closer to **w\_max**, so the RCL becomes smaller, focusing only on the **best candidates** (those closer to **w\_max**). This means that the algorithm **prioritizes better choices** for the cut.
  + **Less randomness**: While there's still some randomness in selecting candidates from the smaller RCL, the algorithm is now more focused on high-quality vertices that will contribute significantly to the cut.
  + **Result**: The algorithm performs **better** because it focuses on better candidates while still retaining a bit of randomness to avoid getting stuck in a local optimum. It **finds better cuts** by avoiding suboptimal vertices and concentrating on those that maximize the cut value.

This ensures a better grasp value when used in the construction phase



A slight increase in the performance for a higher alpha value but as we move from **α = 0.25** to **α = 0.75**, we're gradually improving the **quality of the initial solution**, but the **improvement in the final solution** may not be very large. At a certain point, the **local search** might make up for any initial shortcomings from the construction phase, **diluting the benefit** of having a larger α in the case of GRASP.

# Comparison of Number of Iterations



We can see that, not much change or improvements happen for the number of iterations of GRASP algorithm as it has similar solution ranges so variation in the iterations doesn’t guarantee a different value. Moreover after certain iterations it will get stuck in the best value so much improvement is not possible and for a reasonable small iteration number the solution is guaranteed to be optimal.

# Conclusion

The GRASP algorithm and its components (Greedy, Randomized, Semi-Greedy, and Local Search) provide a robust method for solving the MAX-CUT problem. The results and comparisons indicate the effectiveness of GRASP in achieving high-quality solutions.

# References

1. Feo, T. A., & Resende, M. G. (1995). Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, 6(2), 109-133.  
2. Fischetti, M., & Lodi, A. (2003). Local search algorithms for the max-cut problem. INFORMS Journal on Computing, 15(4), 349-366.