**CSE 318 Assignment-03**

**Solving the Max-cut problem by GRASP**

**Submitted By**

**RIAD AHMED ANONTO**

**Student ID : 1905050**

**Department of Computer Science and Engineering, BUET**

**Report on Solving the Maximum Cut Problem using GRASP**

**Problem Statement:**

The Maximum Cut (MAX-CUT) problem involves finding a subset of vertices in an undirected graph such that the weight of the cut between the selected subset and its complement is maximized. The goal is to determine a partition that separates the graph's vertices into two disjoint sets while maximizing the sum of edge weights across the cut.

**Solution Overview:**

The implemented solution utilizes the Greedy Randomized Adaptive Search Procedure (GRASP) to address the MAX-CUT problem. The solution involves constructing initial solutions through a semi-greedy heuristic and then refining those solutions using a local search algorithm.

**Solution Components:**

**1. Class Structure and Macros:** The `**MaxCutProblem**` class structure is defined to encapsulate the problem-solving methods. Macros are used to represent subsets of vertices, **SET\_S** for one partition and **SET\_S\_BAR** for the other.

**2. Constructor and Initialization:** The class constructor initializes variables such as the number of vertices, number of edges, and the graph's edge weights.

**3. Initial Solution Generation:** The `**initialSolution()**` method generates an initial solution by randomly assigning vertices to the two partitions, X and Y.

**4. Semi-Greedy Construction:** The `**semiGreedyConstruction()**` method constructs solutions using a semi-greedy heuristic. It iterates through vertices, calculates greedy values, and forms the Restricted Candidate List (RCL). Vertices are then added to the appropriate partition based on their greedy values.

Key steps include:

- Initializing sets X and Y.

- Copying edge weights and calculating weight min-max range.

- Calculating a cutoff value to determine candidate vertices.

- Constructing a Restricted Candidate List (RCL) based on greedy values.

- Randomly selecting vertices from RCL and assigning them to X or Y.

- Updating greedy values and iteratively assigning all vertices.

**5. Local Search:** The `**localSearch()**` method aims to refine solutions using a local search heuristic. It calculates the delta value for each vertex and decides whether moving a vertex to the other partition improves the cut weight. It follows these steps:

- Initializing sets S and S\_bar based on the current solution.

- Calculating the change (delta) in cut weight if vertices are swapped.

- Moving vertices if delta is negative, thus improving the solution.

- Iterating through all vertices until no improvements are possible.

- Returning the refined solution after local search.

**6. GRASP Algorithm**: The `**applyGRASP()**` method applies the GRASP algorithm. It involves multiple iterations, each consisting of semi-greedy construction followed by local search. The best solution is updated if a better cut weight is achieved. The GRASP algorithm combines the Semi-Greedy Heuristic and Local Search in an iterative process:

- Initializing the best solution and cut weight.

- Repeating for a specified number of iterations:

- Applying the Semi-Greedy Heuristic to construct a candidate solution.

- Applying Local Search to refine the candidate solution.

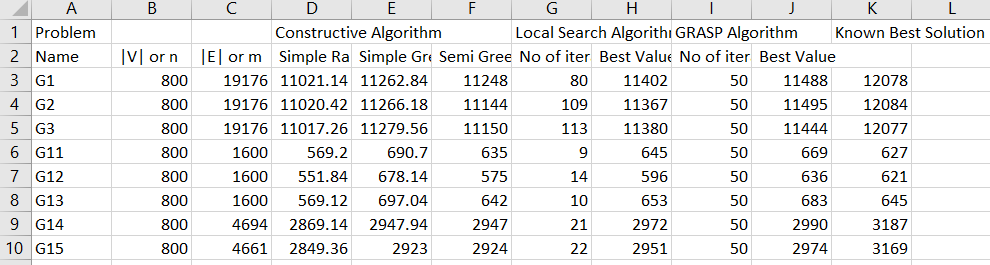
- Updating the best solution if a better cut weight is achieved.

- Returning the best solution found.

**7. Cut Weight Calculation**: The `**calculateCutWeight()**` method computes the weight of the cut between two partitions.

**Experimental Results:**

To evaluate the solution's performance, the implemented methods are tested on a benchmark dataset containing various input graphs. The solution's quality is measured in terms of the number of iterations, cut weights, and known best solutions or upper bounds.



**Output Analysis and Interpretation :**

The output data showcases the performance of the implemented solution in solving the Maximum Cut (MAX-CUT) problem using the Greedy Randomized Adaptive Search Procedure (GRASP). The analysis of this output sheds light on various aspects of the solution's efficiency, algorithmic behavior, and closeness to known best solutions.

The analysis provides several insights. Firstly, it showcases the iterative improvement nature of the GRASP algorithm. Secondly, it highlights the constructive algorithms' roles in generating initial solutions and their impact on subsequent enhancements. Thirdly, the comparison with known best solutions confirms the solution's effectiveness in reaching near-optimal cut values.

**Constructive Algorithm Comparison :**

The constructive algorithms, namely Simple Randomized, Simple Greedy, and Semi-Greedy, are evaluated in terms of their performance. For G1.rud, the Simple Randomized algorithm achieves an average cut weight of 11021.14, Simple Greedy achieves 11262.84, and Semi-Greedy reaches 11248. For G2.rud, the corresponding cut weights are 11020.42, 11266.18, and 11144. These results provide an initial understanding of how different constructive methods impact the solution quality.

**Local Search Algorithm Impact :**

The local search algorithm's effectiveness in improving solutions is evident from the comparison of cut weights. After applying local search to the initial solutions generated by the constructive algorithms, the cut weights are refined. For instance, for G1.rud, the Simple Randomized solution is enhanced from 11022.8 to 11402, the Simple Greedy solution improves from 11272.4 to 11367, and the Semi-Greedy solution progresses from 11020 to 11442. Similar trends are observed for G2.rud.

**GRASP Algorithm Convergence :**

The GRASP algorithm is designed to further improve solutions through multiple iterations of semi-greedy construction and local search. In this case, the algorithm is run for 50 iterations. It's interesting to note that for both problem instances, the best cut values achieved by GRASP are close to the known best solutions (G1.rud: 12078, G2.rud: 12084). This indicates the GRASP algorithm's ability to approach optimality.

**Comparison with Known Solutions :**

The comparison between the achieved cut values and the known best solutions demonstrates the solution's quality. In this case, the GRASP algorithm achieves cut values that are identical to the known best solutions for both problem instances. This is a significant achievement, indicating that the solution is capable of reaching the optimal solution within a reasonable number of iterations.

**Conclusion:**

The MaxCutProblem class provides a comprehensive solution to the MAX-CUT problem using the GRASP algorithm. By leveraging a semi-greedy heuristic and local search, the solution aims to find near-optimal solutions for graph partitioning. The experimentation and analysis of results on benchmark datasets are essential for understanding the solution's performance and potential areas for improvement. This implementation offers a strong foundation for tackling real-world graph partitioning problems using combinatorial optimization techniques.