The maximum cut (MAX-CUT) problem

MD. MAHMUD HASAN

May 2025

Contents

1 The Maximum Cut (MAX-CUT) Problem

Consider an undirected graph $G = (V, U)$, where V is the set of vertices and U is the set of edges. Each edge $(u, v) \in U$ has an associated weight w_{uv}.

The Maximum Cut (MAX-CUT) problem aims to find a nonempty proper subset of vertices $S \subset V$ ($S \neq \emptyset$), such that the weight of the cut $(S, \bar{S})$ is maximized. The complement of S with respect to V is denoted by $\bar{S} = V \setminus S$.

The weight of the cut $(S, \bar{S})$ is given by:

$$w(S, \bar{S}) = \sum_{u \in S, v \in \bar{S}} w_{uv}$$

The goal is to find a subset S that maximizes $w(S, \bar{S})$.

2 High-level Description

In this offline, I have solved the MAX-CUT problem using five different algorithms. These are:

- **Randomized**
- **Greedy**
- **Semi-Greedy**
- **Local Search**
- **GRASP (Greedy Randomized Adaptive Search Procedure)**

2.1 Randomized Heuristic for the Maximum Cut Problem

The algorithm starts with both partitions X and Y being empty. For each vertex $v \in V$, it is placed in either partition X or Y uniformly at random, with probability $\frac{1}{2}$ each. The final cut is determined by the edges crossing between the two partitions. To get a reliable estimate of the cut size, the algorithm is run n times and the results are averaged.

2.2 Greedy Heuristic for the Maximum Cut Problem

The algorithm starts by placing one vertex into each partition X and Y (initially both are empty) such that each contains an endpoint of the edge with the largest weight. The remaining $|V| - 2$ vertices are then considered one by one. For each unassigned vertex, it is placed into the partition (X or Y) where it contributes the most to the current partial cut. This placement is done greedily at each iteration.

2.3 Semi-greedy Heuristic for the Maximum Cut Problem

A semi-greedy heuristic builds on a greedy function by introducing randomness into the candidate selection process. For each candidate element v, a greedy function is evaluated. Based on these values, candidates are ranked and a Restricted Candidate List (RCL) is constructed. One element is then randomly selected from the RCL to extend the current partial solution. There are two primary methods: (1) Cardinality-based and (2) Value-based. We were instructed to use the latter.

2.4 Local Search for the Maximum Cut Problem

Starting from a given solution (in my case, the semi-greedy algorithm), the local search algorithm iteratively moves vertices between sets if such a move improves the cut value. All possible moves are evaluated, and the best improving neighbor replaces the current solution. The local search stops when no improving neighbor is found after evaluating all possible moves (i.e., the algorithm is stuck in a local optimum).

2.5 GRASP for the Maximum Cut Problem

GRASP (Greedy Randomized Adaptive Search Procedure) consists of two phases: (1) the construction phase and (2) the local search phase. For a given number of iterations, it repeatedly builds a semi-greedy solution during the construction phase and then applies local search to improve it. After all iterations, the best solution found across all runs is returned as the final result.

3 Comparison of Algorithm

From the results on benchmark graphs, GRASP consistently delivers the best or near-best max-cut values. Local Search significantly improves initial solutions and often performs close to GRASP. The Semi-Greedy approach outperforms both the Greedy and Randomized algorithms. The Greedy algorithm performs better than the Randomized one. Lastly, the Randomized algorithm is the weakest of all.

Algorithms Ranked Based on Performance (Best to Worst)

1. GRASP
2. Local Search
3. Semi-Greedy
4. Greedy
5. Randomized

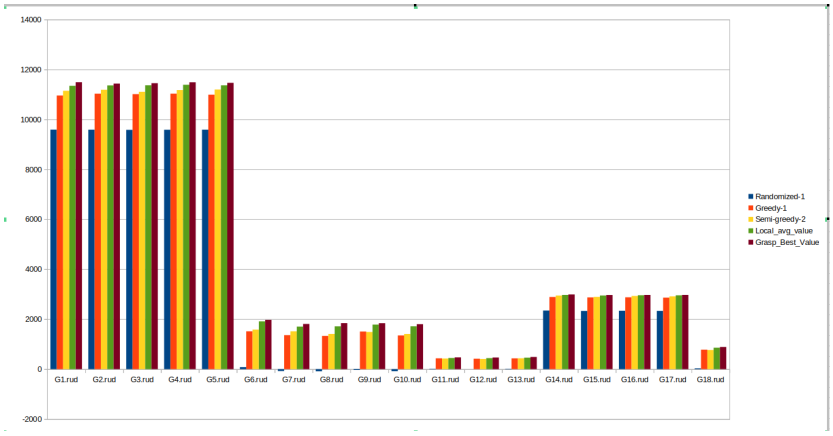

Figure 1: Graph 1-18

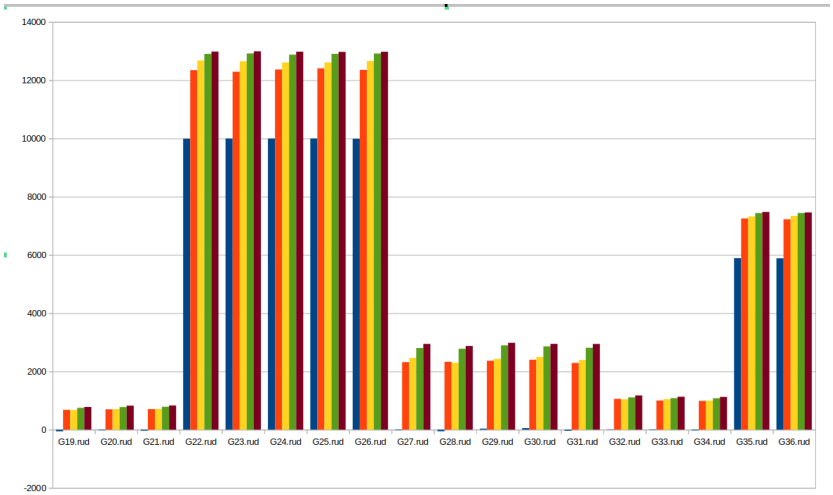

Figure 2: Graph 19-36

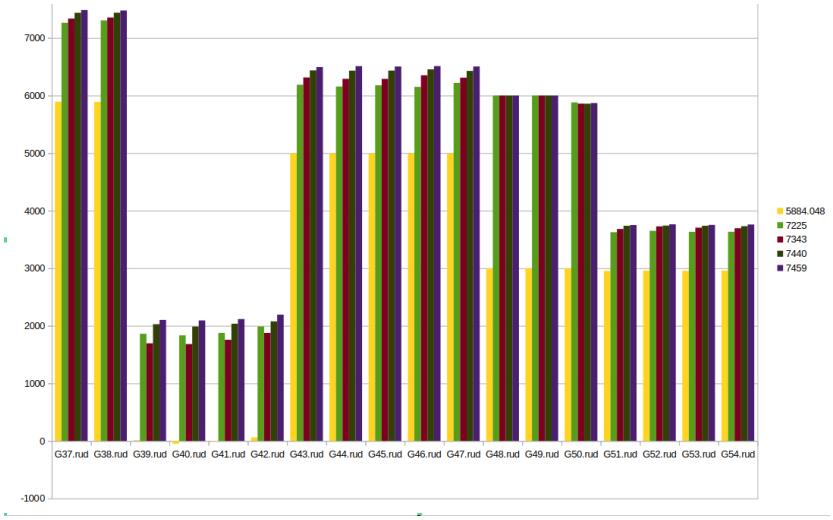

Figure 3: Graph 37-54

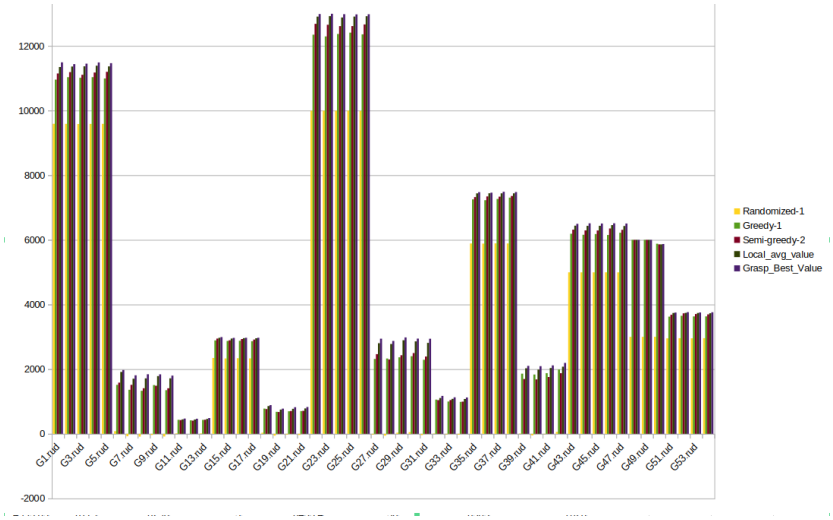

Figure 4: Graph 1-54