Heuristic Approaches to the MAX-CUT Problem

Arnob Biswas
ID: 215015
Department of CSE, BUET

May 12, 2025

Introduction

The *Maximum Cut (MAX-CUT)* problem is a fundamental optimization challenge in graph theory: given an undirected weighted graph $G = (V, E)$, partition V into two disjoint subsets so as to maximize the sum of weights of edges crossing the cut. Because MAX-CUT is NP-hard, exact algorithms become infeasible even for moderately sized graphs. Heuristic and metaheuristic methods offer practical alternatives, trading optimality for speed. In this work, we implement and evaluate five main approaches:

1. A **Randomized** baseline that assigns vertices uniformly at random and averages over trials,
2. Two variants of the **Greedy** heuristic (naive scan vs. priority-queue accelerated),
3. A **Semi-Greedy** (value-based RCL) method introducing controlled randomness,
4. Two variants of **Local Search** (full neighborhood scan vs. delta-update),
5. The **GRASP** metaheuristic that repeatedly combines semi-greedy construction with local refinement.

We test these on a suite of 54 benchmark graphs, comparing average cut weights, runtime behavior, and—where available—known best cuts.

Algorithms and Time Complexities

1. Randomized Heuristic

Assign each $v \in V$ to partition X or Y with probability $\frac{1}{2}$. Repeat for T trials and take the mean cut weight.

$$\text{Complexity per trial: } O(|V| + |E|), \quad \text{Total: } O(T\,(|V| + |E|)).$$

Pros: trivial to implement, unbiased baseline. **Cons:** high variance, poor quality vs. informed heuristics.

2. Greedy Heuristic

We have implemented two variants:

(a) **Naive Greedy:** Traverse each vertex once. For each, evaluate the gain by summing weights of incident edges already assigned, and place it to the partition yielding the higher gain.

Time Complexity: $O(|V| + |E|)$.

Pros: simple, fast, and memory-efficient. **Cons:** decisions are locally optimal and globally myopic; quality depends on input order.

(b) **Improved Greedy:** Maintain a priority queue (max-heap) to always place the vertex with maximum cut gain next. After each placement, update neighbors' scores in the heap.

Time Complexity: $O((|V| + |E|) \log |V|)$.

Pros: more informed placement decisions; typically higher cut than naive. **Cons:** extra time and space overhead; still greedy and prone to early commitment.

3. Semi-Greedy Heuristic

Compute for each candidate v its greedy gain $\Delta(v)$. Build a Restricted Candidate List of size determined by $\mu = \min \Delta + \alpha(\max \Delta - \min \Delta)$. Select one at random.

$$O(|V|^2 + |E|)$$

per construction. **Pros:** balances greediness and randomness; **Cons:** still quadratic, sensitive to α.

4. Local Search

Starting from an initial cut (constructed via greedy or semi-greedy), local search refines the solution by flipping vertices between partitions to improve the total cut.

We implemented two variants:

(a) **Naive (Best Improvement):** In each round, recompute gain $\delta(v)$ for all vertices and select the one with the highest positive gain. This requires evaluating the full vertex set repeatedly.

Time Complexity: $O(V(V + E))$

Pros: Always chooses the most beneficial move; deeper local search. **Cons:** Computationally expensive; scales poorly with large graphs.

(b) **First Improvement:** Vertices are scanned in order. The first vertex found with positive gain is moved, and the process restarts. Only one gain check per move is needed.

$$\text{Time Complexity: } O(I \cdot E)$$

Pros: Faster in practice; exits early once improvement is found. **Cons:** May settle in weaker local optima compared to best improvement.

5. GRASP

Repeat for M iterations:

$$\text{construct via semi-greedy } + \text{ local search}$$

and keep best.

$$O\Big(M\big(|V|^2 + |E| + I \cdot |E|\big)\Big).$$

Pros: high solution quality, robust; **Cons:** highest runtime.

Note: For all experiments involving the semi-greedy heuristic and GRASP construction phase, the value of the RCL threshold parameter was fixed at $\alpha = 0.85$. This value was chosen empirically to balance greedy exploitation with randomized diversification.

Benchmark Results on 54 Graphs

Table 1: Benchmark results for 54 graphs using selected heuristics.

Problem	—V—	—E—	Randomize	Greedy	Semi-greedy Weight	Simple LS Iterations	LS Avg. Value	GRASP Iters	GRASP Best	Known Best
G1	800	19176	9603	11247	11274	20	11336	50	11469	12078
G2	800	19176	9618	11272	11304	20	11342	50	11470	12084
G3	800	19176	9607	11255	11214	20	11334	50	11466	12077
G4	800	19176	9585	11255	11212	20	11348	50	11470	
G5	800	19176	9602	11229	11208	20	11343	50	11475	
G6	800	19176	101	1550	1772	20	1876	50	2040	
G7	800	19176	-80	1373	1582	20	1738	50	1831	
G8	800	19176	-99	1426	1527	20	1732	50	1877	
G9	800	19176	-16	1429	1624	20	1786	50	1885	
G10	800	19176	-107	1297	1580	20	1710	50	1827	
G11	800	1600	11	392	480	20	414	50	500	627
G12	800	1600	8	378	460	20	402	50	500	621
G13	800	1600	10	400	480	20	423	50	522	645
G14	800	4694	2346	2896	2933	20	2908	50	2987	3187
G15	800	4661	2323	2881	2898	20	2892	50	2971	3169
G16	800	4672	2358	2908	2915	20	2899	50	2970	3172
G17	800	4667	2337	2890	2924	20	2896	50	2970	
G18	800	4694	44	732	828	20	803	50	908	
G19	800	4661	-73	546	715	20	724	50	824	
G20	800	4672	-40	624	773	20	749	50	847	
G21	800	4667	-36	646	732	20	751	50	848	

Problem	—V—	—E—	Randomize	Greedy	Semi-greedy Weight	Simple LS Iterations	LS Avg. Value	GRASP Iters	GRASP Best	Known Best
G22	2000	19990	10002	12738	12752	20	12805	50	13056	14123
G23	2000	19990	9983	12765	12712	20	12789	50	13036	14129
G24	2000	19990	9910	12736	12762	20	12782	50	13039	14131
G25	2000	19990	10016	12790	12691	20	12777	50	13019	
G26	2000	19990	9987	12725	12788	20	12773	50	13000	
G27	2000	19990	-28	2255	2637	20	2768	50	2953	
G28	2000	19990	2	2239	2612	20	2737	50	2936	
G29	2000	19990	4	2304	2711	20	2833	50	3041	
G30	2000	19990	76	2318	2759	20	2836	50	3035	
G31	2000	19990	-28	2253	2638	20	2765	50	2967	
G32	2000	4000	11	942	1170	20	1012	50	1232	1560
G33	2000	4000	-17	940	1142	20	996	50	1214	1537
G34	2000	4000	-29	914	1168	20	986	50	1214	1541
G35	2000	11778	5884	7277	7394	20	7307	50	7474	8000
G36	2000	11766	5872	7309	7362	20	7294	50	7472	7996
G37	2000	11785	5944	7287	7396	20	7308	50	7471	8009
G38	2000	11779	5863	7283	7400	20	7306	50	7476	
G39	2000	11778	24	1625	1919	20	1949	50	2161	
G40	2000	11766	-35	1552	1973	20	1947	50	2159	
G41	2000	11785	-21	1586	2018	20	1934	50	2150	
G42	2000	11779	65	1728	2057	20	2018	50	2229	
G43	1000	9990	5027	6371	6371	20	6379	50	6502	7027
G44	1000	9990	4993	6372	6334	20	6389	50	6492	7022
G45	1000	9990	5001	6341	6408	20	6382	50	6535	7020
G46	1000	9990	4955	6396	6374	20	6373	50	6492	
G47	1000	9990	5005	6411	6343	20	6384	50	6509	
G48	3000	6000	2986	6000	5822	20	4831	50	5986	6000
G49	3000	6000	3019	6000	5744	20	4824	50	5936	6000
G50	3000	6000	2969	5880	5760	20	4835	50	5844	5988
G51	1000	5909	2962	3637	3710	20	3657	50	3754	
G52	1000	5916	2940	3640	3720	20	3666	50	3755	
G53	1000	5914	2954	3670	3684	20	3659	50	3752	
G54	1000	5916	2967	3657	3673	20	3655	50	3756	

Heuristic Comparison on 15 Sampled Graphs

To evaluate general performance trends, 15 graphs were uniformly sampled from the benchmark set. Their results across four heuristics—Randomized, Greedy, Semi-greedy, and GRASP—are visualized in the bar chart below.

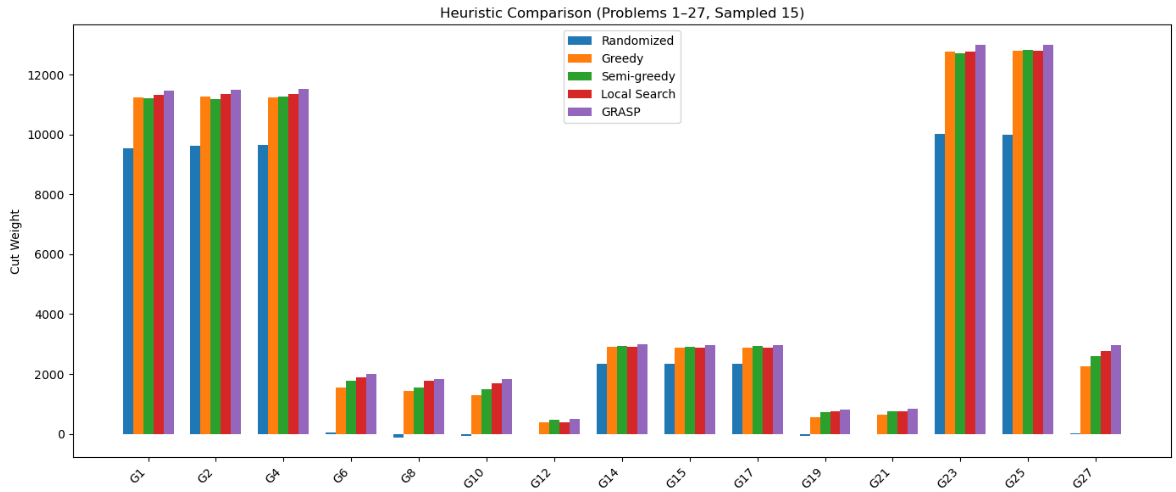

Figure 1: Cut values from 5 heuristics on 15 uniformly sampled problems from indices 1–27.

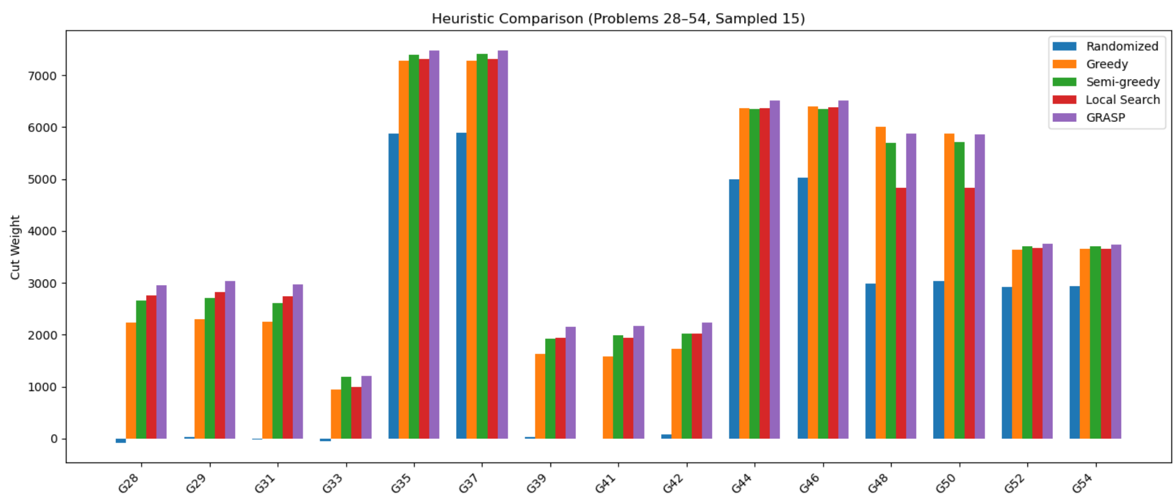

Figure 2: Cut values from 5 heuristics on 15 uniformly sampled problems from indices 28–54.

Conclusion

This report presents a comparative study of heuristic and metaheuristic approaches to the MAX-CUT problem. Based on both complexity analysis and experimental data from 54 benchmark graphs, the key findings are as follows:

(i) **Randomized** performs with the lowest complexity per trial $O(V + E)$, and is useful as a fast baseline. However, it shows the highest variability and lowest average cut value.

(ii) **Naive Greedy** achieves excellent efficiency at $O(V + E)$, processing each vertex once and making partition decisions based on immediate local gain. It is ideal for large-scale graphs when speed is critical. However, its fixed ordering and lack of global awareness can result in poor-quality cuts, especially in adversarial structures.

(iii) **Improved Greedy**, at $O((V + E) \log V)$, improves cut quality by always placing the vertex with the highest marginal gain. While slower than the naive variant, it achieves a better balance between performance and quality and remains practical for medium-to-large graphs.

(iv) **Semi-Greedy** introduces probabilistic decision-making via a Restricted Candidate List (RCL), allowing controlled diversification during construction. Its $O(V^2 + E)$ complexity stems from full gain evaluations and selection mechanisms. This method often outperforms greedy heuristics, especially when fine-tuned via the α parameter.

(v) **Local Search** improves an initial cut by flipping vertices to increase the cut value. Our **Naive (Best Improvement)** strategy recomputes gains for all vertices each round, leading to a complexity of $O(V(V+E))$. It explores deeper local optima but is costly for large graphs. In contrast, the **First Improvement** strategy has complexity $O(I \cdot E)$, making a move as soon as any improvement is found. This version is often faster, though it may terminate earlier in a suboptimal state.

(vi) **GRASP** integrates semi-greedy construction with local search over M iterations, yielding the best overall results. Its complexity is $O(M \cdot (V^2 + E + I \log V))$, but it consistently approaches known optimal values across diverse graphs. GRASP is especially suitable for applications where cut quality is more important than computation time.

In summary:

- When **speed is the priority**, use **Naive Greedy** or **Randomized**.
- When **moderate quality with good efficiency** is acceptable, choose **Improved Greedy** or **Local Search (First Improvement)**.
- When **solution quality is critical** and resources permit, **GRASP** is the most robust and effective strategy among those tested.