CSE 318 Assignment-02

Solving the Max-cut problem by GRASP

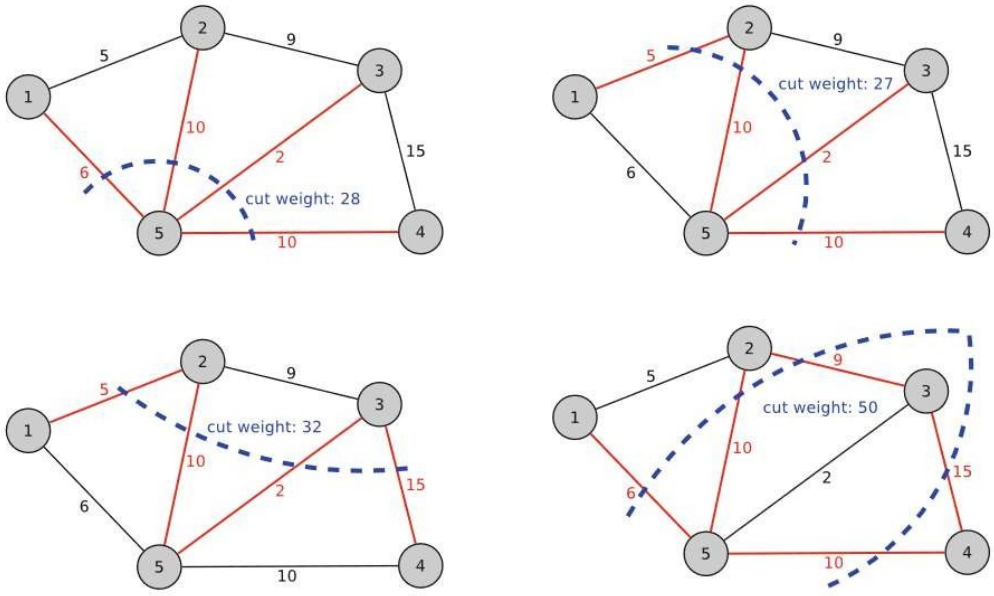
April 28, 2025

The maximum cut (MAX-CUT) problem

Given an undirected graph $G = (V, U)$, where V is the set of vertices and U is the set of edges, and weights w_{uv} associated with each edge $(u, v) \in U$. The **Maximum Cut (MAX-CUT)** problem consists in finding a nonempty proper subset of vertices $S \subset V$ $(S \neq \emptyset)$, such that the weight of the cut $(S, \bar{S})$, given by

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

is maximized.



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 $w(S, \bar{S}) = 50$.

The above figure shows four cuts having different weights on a graph with five nodes and seven edges. The maximum cut has $S = \{1, 2, 4\}$ and $\bar{S} = \{3, 5\}$, with weight $w(S, \bar{S}) = 50$.

Solving a combinatorial optimization problem by GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) is a randomized multistart iterative method for computing good quality solutions of combinatorial optimization problems. Each GRASP iteration is usually made up of:

- a *construction phase*, where a feasible solution is constructed, and
- a *local search phase*, which starts at the constructed solution and applies iterative improvement until a locally optimal solution is found.

Typically, the construction phase of GRASP is a semi-greedy or randomized greedy algorithm.

Algorithm 1 GRASP (Greedy Randomized Adaptive Search Procedure)

```
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
```

1: Start GRASP with a maximum number of tries (MaxIterations)

2–3: For each try, build a smart-random solution (greedy + randomness)

5–6: If this is the first try, save this as your best solution

7–8: If this new solution is better than the best so far, replace it

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.

Algorithm 2 Randomized-1 Heuristic for MAX-CUT

```
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} w_uv
13:        totalCutWeight ← totalCutWeight + cutWeight
14:    end for
15:    averageCutWeight ← totalCutWeight / n
16:    return averageCutWeight
17: end procedure
```

Greedy Heuristic for the Maximum Cut Problem

The algorithm starts by placing one vertex to 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.

Semi-greedy Heuristic for the Maximum Cut Problem

A semi-greedy heuristic builds on a greedy function by introducing randomness in 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.

Algorithm 3 Greedy Heuristic for MAX-CUT

procedure GREEDYMAXCUT($G = (V, E), w$)
  Initialize partitions: $X \leftarrow \emptyset$, $Y \leftarrow \emptyset$
  Find edge $(u, v) \in E$ with maximum weight w_{uv}
  $X \leftarrow X \cup \{u\}$, $Y \leftarrow Y \cup \{v\}$
  $U \leftarrow V \setminus \{u, v\}$ ▷ Unassigned vertices
  for all vertex $z \in U$ **do**
    Compute weight if $z \in X$: $w_X = \sum_{y \in Y} w_{zy}$
    Compute weight if $z \in Y$: $w_Y = \sum_{x \in X} w_{zx}$
    if $w_X > w_Y$ **then**
      $X \leftarrow X \cup \{z\}$
    else
      $Y \leftarrow Y \cup \{z\}$
    end if
  end for
  return partitions X, Y
end procedure

There are two primary strategies for building the RCL:

- **Cardinality-based method:** Selects the top-k candidates (e.g., $k = 5$ or $k = 10$).
- **Value-based method:** Selects all candidates v with greedy function value $\geq \mu$, where

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

  and $\alpha \in [0, 1]$ is a tunable parameter.

For the above greedy heuristic, the candidate elements are vertices which are not yet assigned to set X and set Y. Let $V' = V - \{X \cup Y\}$ be the set of all candidate elements. The cut-off value is denoted by

$$\mu = w_{\min} + \alpha \cdot (w_{\max} - w_{\min}) \tag{1}$$

where α is a parameter such that $0 \leq \alpha \leq 1$, and the restricted candidate list consists of all vertices whose value of the greedy function is greater than or equal to μ. A vertex is randomly selected from the restricted candidate list.

For each vertex v, we define

$$\sigma_X(v) = \sum_{u \in X} w_{vu} \quad \text{and} \quad \sigma_Y(v) = \sum_{u \in Y} w_{vu}.$$

The greedy function value of v is

$$\max\{\sigma_X(v), \sigma_Y(v)\}$$

However, $w_{\min}$ and $w_{\max}$ can be updated using the following naive rules:

$$w_{\min} = \min\left\{\min_{v \in V'} \sigma_X(v), \min_{v \in V'} \sigma_Y(v)\right\}$$

$$w_{\max} = \max\left\{\max_{v \in V'} \sigma_X(v), \max_{v \in V'} \sigma_Y(v)\right\}$$

Use equation (1) for determining the placement of the current candidate vertex in X or Y.

Local Search for the Maximum Cut Problem

A simple local search method for the maximum cut problem is described below:

Assume that we have obtained a feasible solution at the end of the construction phase and $S, \bar{S}$ are two partitions of the current solution $(S, \bar{S})$. The local search phase is based on the following

neighborhood structure. To each vertex $v \in V$, we associate either the neighbor $(S - \{v\}, \bar{S} \cup \{v\})$ if $v \in S$, or the neighbor $(S \cup \{v\}, \bar{S} - \{v\})$ if $v \in \bar{S}$.

The value

$$\delta(v) = \begin{cases} \sigma_S(v) - \sigma_{\bar{S}}(v), & \text{if } v \in S \\ \sigma_{\bar{S}}(v) - \sigma_S(v), & \text{if } v \in \bar{S} \end{cases} \tag{2}$$

represents the change in the objective function (MAX-CUT value) associated with moving vertex v from one subset of the cut to the other. All possible moves are investigated and the best improving neighbor replaces the current solution.

The local search stops when no improving neighbor is found after evaluation of all possible moves (we are stuck in a local optima).

Input Format

The input is provided as a text file with the following structure:

- The first line contains two integers: n and m, where:
  - n is the number of vertices
  - m is the number of edges (the size of the graph).
- Each of the next m lines describes an edge of the graph using three integers:

$$V_1 \quad V_2 \quad W_t$$

  - V_1 and V_2 are the endpoints of the edge,
  - W_t is the weight associated with the edge (V_1, V_2).

The graphs are undirected.

Benchmark Data Set

The benchmark data set contains 54 input graphs. For 24 of them, the known best solution or upper bound is provided in the table below:

Problem	**Known Best Solution**	**Problem**	**Known Best Solution**	**Problem**	**Known Best Solution**	**Problem**	**Known Best Solution**
G1	12078	G14	3187	G32	1560	G43	7027
G2	12084	G15	3169	G33	1537	G44	7022
G3	12077	G16	3172	G34	1541	G45	7020
G11	627	G22	14123	G35	8000	G48	6000
G12	621	G23	14129	G36	7996	G49	6000
G13	645	G24	14131	G37	8009	G50	5988

Table 1: Known best solutions or upper bounds for selected benchmark instances.

Task Description

1. Algorithm Implementation

You need to implement the algorithms: **Greedy**, **Randomized**, **Semi-Greedy**, **Local Search** and **GRASP**. For each of these algorithms, you are required to record the performance on all 54 benchmark graphs. For the **Semi-Greedy** algorithm, you need to use the value-based selection method as described earlier.

2. CSV File Submission

A CSV file named 2105***.csv should be submitted summarizing the results obtained from running the different algorithms on the 54 benchmark graphs provided. The format of the CSV can be as follows:

Problem			Constructive algorithm			Local search		GRASP		Known best solution or upper bound
Name	|V| or n	|E| or m	Simple Randomized or Ranomized-1	Simple Greedy or Greedy-1	Semi-greedy-1	Simple local or local-1		GRASP-1		
						No. of iterations	Aver age value	No. of iterations	Best value	
G1	800	19176								12078
G22	2000	19990								14123
G43	1000	9990								7027

For semi-greedy, you may include information regarding value of α.

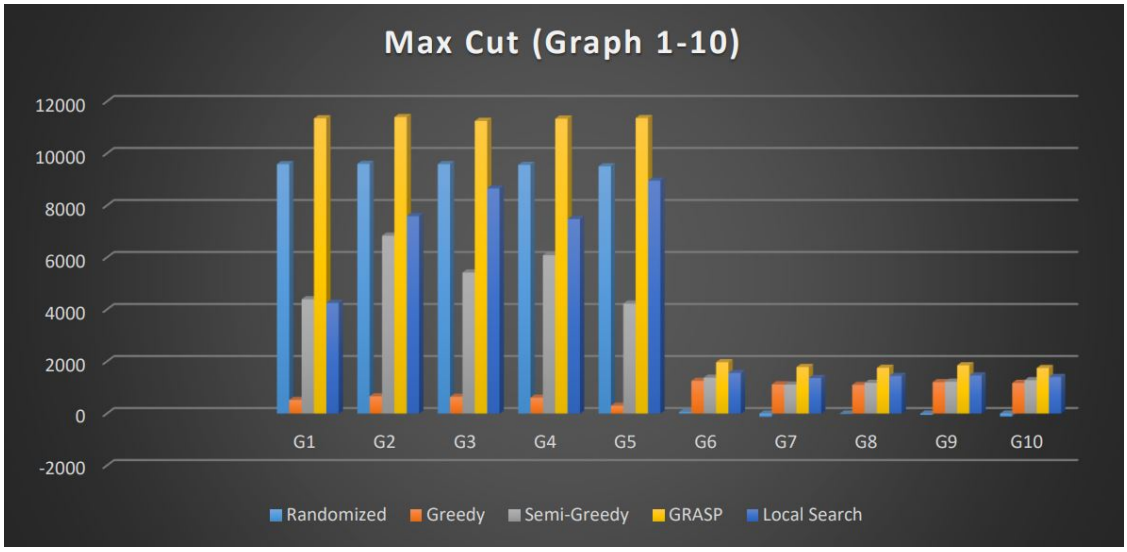
You should be prepared to regenerate the result for any one algorithm on any one graph during evaluation.

3. Report Submission

Along with the CSV file, you should submit a brief report in PDF format (renamed as 2105***.pdf). The report should include:

- A high-level description of each algorithm implemented (Randomized, Greedy, Semi-Greedy, Local Search, GRASP).
- Comparison of algorithms (e.g., which algorithm consistently performs better across the benchmarks).
- Plots or visualizations that demonstrate how the algorithms perform across different graphs.

An example plot is shown below for reference:



You do not need to match the exact values as shown in the sample plots—they are included for illustration only.

Mark Distribution

Task	Marks
Implementation of Randomized Algorithm	10
Implementation of Greedy Algorithm	10
Implementation of Semi-Greedy Algorithm	20
Implementation of GRASP	40
Report	10
Observation from Plots	5
Proper Submission	5
Total	**100**

Table 2: Mark Distribution for the Assignment

Submission Format

Create a folder named `2105***`. Place all source files (e.g., code files) inside the folder, and rename them as your student ID. Include the report as `2105***.pdf`, and the result CSV file as `2105***.csv`. Compress the folder into a ZIP file and submit the zip file (2105***.zip) in Moodle.

Submission Deadline

12 May, Monday, 11:55 pm