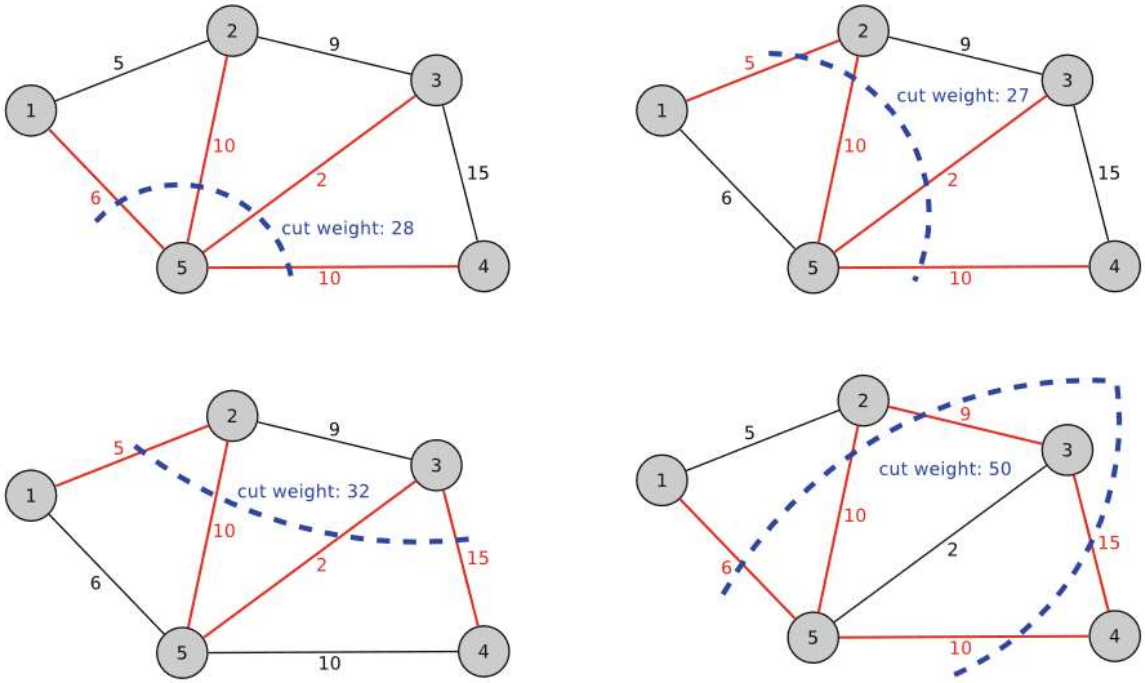
CSE 318 Assignment-03

Solving the Max-cut problem by GRASP

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. The pseudo-code is shown below:

```
procedure GRASP(MaxIterations)
1     for i = 1, ... , MaxIterations do
2           Build a greedy randomized solution x;
3           x ← LocalSearch(x);
4           if i = 1 then x* ← x;
5           else if w(x) > w(x*) then x* ← x;
6     end;
7     return (x*);
end GRASP;
```

Pseudo-code of a generic GRASP

Greedy heuristic for the maximum cut problem

A simple greedy heuristic for the maximum cut problem is described below:

The algorithm starts by placing one vertex to each partition *X* and *Y* (initially both are empty) such that each contains an endpoint of a largest-weight edge. The remaining $|V|$ - 2 vertices are examined one by one to find out the placement of which vertex to either one of the two partitions maximizes the weight of the partial cut constructed so far. At each iteration, one vertex is placed to either set *X* or set *Y*.

Semi-greedy heuristic for the maximum cut problem

For a semi-greedy heuristic, for each candidate element *x*, we apply a greedy function to *x*. In this way, we obtain a ranking of all elements according to their greedy function values. Next, well-ranked elements are placed in a restricted candidate list (RCL), and we may select one element randomly selected from the RCL to add to (partially constructed) solution.

Two methods may be considered for building the RCL. Cardinality-based methods place k best candidates. For example, topmost k = 5 or topmost k = 10 elements may be chosen. Value-based methods place all candidates having greedy values better than $\alpha \times$ best_value in RCL, where $\alpha \in [0,1]$.

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 $\mu = w_{min} + \alpha * (w_{max} - w_{min})$, where α is a parameter such that 0 ≤ α ≤ 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 $\sigma_X(v) = \sum_{u \in X} w_{vu}$ and $\sigma_Y(v) = \sum_{u \in Y} w_{vu}$

The greedy function value of $v = \max\{\sigma_X(v), \sigma_Y(v)\}$

If $\sigma_X(v) > \sigma_Y(v)$, then v is placed in Y. Otherwise v is placed in X

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

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

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

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

Represents the change in the objective function 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). The pseudo-code is shown below:

```
procedure LocalSearch(x = {S, S̄})
1     change ← .TRUE.
2     while change do;
3           change ← .FALSE.
4           for v = 1, ..., |V| while .NOT.change circularly do
5                 if v ∈ S and δ(v) = σ_S(v) − σ_S̄(v) > 0
6                 then do S ← S \ {v}; S̄ ← S̄ ∪ {v}; change ← .TRUE. end;
7                 if v ∈ S̄ and δ(v) = σ_S̄(v) − σ_S(v) > 0
8                 then do S̄ ← S̄ \ {v}; S ← S ∪ {v}; change ← .TRUE. end;
9           end;
10    end;
11    return (x = {S, S̄});
end LocalSearch;
```

Pseudo-code of the local search phase

The format of the Input

The input is a text file whose first line consists of two integers: *n*, *m*.

Here *n* is the number of vertices (order) and *m* is the number of edges (size) of the input graph.

Each of the next *m* lines contains three integers describing an edge: *V*1 *V*2 *Wt*.

Here *V*1 and *V*2 are the endpoints of the edge which has the weight *Wt*.

```
1000 9990
1 794 1
1 477 1
1 150 1
1 717 1
1 196 1
1 615 1
```

```
3375 10125
1 2 -89257
2 3 200640
3 4 282346
4 5 22701
5 6 57242
6 7 -254656
7 8 150118
8 9 12733
9 10 22475
```

Two examples of input files:

The first graph has 1000 vertices and 9990 edges.

The second graph has 3375 vertices and 10125 edges.

The format of the output

The output may contain numeric data (excel/csv/text) for the performance of your program implementing GRASP, but the submission of a brief report (text/pdf) which summarizes and explains the output is mandatory.

The format of the report depends on the design of the GRASP you have implemented. You should briefly mention or describe which greedy or semi-greedy techniques or local search operators were implemented. If it is your own innovation or improvisation, it needs to be described properly. If you have used a research paper or website, you should mention it and provide web links.

The summary table may have the following format:

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	Best value	No. of iterations	Best value	
G1	800	19176								12078
G22	2000	19990								14123
G43	1000	9990								7027

It is possible to make separate performance summarizing table for constructive phase, local search phase, and GRASP. If possible, you should include more information in your summary table. For example, it may be possible to include a column in the problem area which provides the weight information. The weights may be all +1, or (-1/+1), or integers between -100 to +100 (-100, +100). Similarly, for semi-greedy, you may include information regarding value of α or cardinality k.

For comparison with *Greedy-1* or the simple greedy algorithm described earlier, we may use a simple randomized algorithm (*Randomized-1*) for the max cut problem. It is described below:

Initially, both partitions *X* and *Y* are empty. For each vertex $v \in V$, place v in partition *X* or partition *Y* with uniform randomness, i.e., with probability $\frac{1}{2}$. The procedure terminates when all vertices are placed either in *X* or *Y*.

For putting value in the column of *Randomized-1* or *Semi-greedy-1*, you should generate *n* number of outputs and then take an average of them. For example, you may construct *n* = 10 (or, *n* = 100) runs of Randomized-1 and divide the sum total by 10 (or, 100).

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 or upper bound	Problem	Known best solution or upper bound	Problem	Known best solution or upper bound	Problem	Known best solution or upper bound
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

You should pick up a number of them to perform experimentation and present the obtained results in the summary table.

Words of advice

The importance of this assignment is not only on implementation, but on experimentation also. The purpose of the specification document is not to suggest the basic minimum requirements of your submission. Rather, it is provided to serve as guide rail or guard rail. You should make an effort to experiment/innovate/improvise with alternative greedy heuristics, variations of semi-greedy choices, local search operators, GRASP designs, etc.