Problem			Constructive algorithm			Local search		GRASP		Known best solution or upper bound
Name	|V| or n	|E| or m	Simple Greedy or Greedy-1	Semigreedy-1	Simple Randomized or Randomised-1	Simple local or local-1		GRASP-1		
						No. of iterations	Best value	No. of iterations	Best value	
G1	800	19176	11259	11257	11242	70	11400	50	11416	12078
G2	800	19176	11284	11244	11240	71	11405	50	11434	12084
G3	800	19176	11229	11276	11270	71	11402	50	11434	12077
G4	800	19176	11335	11261	11261	68	11410	50	11514	
G5	800	19176	11278	11298	11259	76	11413	50	11510	
G6	800	19176	1744	1795	1738	84	1948	50	1998	
G7	800	19176	1589	1563	1590	81	1779	50	1831	
G8	800	19176	1696	1596	1592	93	1794	50	1884	
G9	800	19176	1664	1655	1668	93	1813	50	1905	
G10	800	19176	1633	1598	1593	84	1764	50	1850	
G11	800	1600	482	481	486	5	487	50	506	627
G12	800	1600	486	472	470	5	481	50	496	621
G13	800	1600	510	494	494	5	503	50	534	645
G14	800	4694	2943	2949	2942	21	2972	50	2994	3187
G15	800	4661	2948	2927	2926	21	2949	50	2979	3169
G16	800	4672	2925	2923	2925	23	2958	50	2985	3172
G17	800	4667	2928	2920	2925	22	2951	50	2971	
G18	800	4694	844	839	826	29	883	50	924	
G19	800	4661	764	751	755	31	800	50	836	

G20	800	4672	787	802	791	31	832	50	864	
G21	800	4667	717	761	774	34	816	50	835	
G22	2000	19990	12780	12789	12782	95	12958	50	13032	14123
G23	2000	19990	12811	12794	12806	98	12957	50	13023	14129
G24	2000	19990	12862	12797	12785	95	12954	50	13014	14131
G25	2000	19990	12806	12812	12791	94	12960	50	13009	
G26	2000	19990	12790	12798	12778	90	12953	50	13022	
G27	2000	19990	2777	2677	2681	127	2905	50	3003	
G28	2000	19990	2679	2628	2641	129	2873	50	2958	
G29	2000	19990	2704	2751	2749	137	2968	50	3065	
G30	2000	19990	2723	2762	2748	131	2970	50	3029	
G31	2000	19990	2636	2682	2679	133	2896	50	2979	
G32	2000	4000	1206	1193	1186	12	1218	50	1244	1560
G33	2000	4000	1190	1183	1177	10	1205	50	1228	1537
G34	2000	4000	1194	1173	1179	11	1204	50	1226	1541
G35	2000	11778	7383	7368	7374	51	7444	50	7471	8000
G36	2000	11766	7376	7369	7367	53	7440	50	7468	7996
G37	2000	11785	7372	7380	7377	52	7450	50	7488	8009
G38	2000	11779	7374	7373	7375	54	7448	50	7486	
G39	2000	11778	1974	2000	2025	77	2129	50	2225	
G40	2000	11776	2025	1992	2012	77	2122	50	2166	
G41	2000	11785	2049	2013	2018	71	2122	50	2172	
G42	2000	11779	2115	2086	2102	72	2118	50	2164	
G43	1000	9990	6348	6370	6390	48	6464	50	6508	7027
G44	1000	9990	6402	6381	6389	48	6460	50	6501	7022
G45	1000	9990	6401	6381	6374	49	6467	50	6522	7020
G48	3000	6000	6000	6000	6000	1	6000	50	6000	6000

G49	3000	6000	6000	6000	6000	1	6000	50	6000	6000
G50	3000	6000	5880	5880	5880	1	5880	50	5880	5988
G51	1000	5909	3719	3703	3701	28	3729	50	3759	
G52	1000	5916	3699	3708	3697	26	3736	50	3759	
G53	1000	5914	3686	3699	3703	26	3732	50	3762	
G54	1000	5916	3679	3696	3697	27	3729	50	3749	

Implemented Construction Algorithms:

Greedy Algorithm: I started by selecting the heaviest edge in the graph and used one of its vertices to initialize sets X and Y. Then, I entered a loop for the remaining vertices. For each vertex v, I computed its potential contribution to the maximum cut if it belonged to set X (σ_x) and if it were part of set Y (σ_Y). If σ_x was greater than or equal to σ_Y, I included vertex v in set X. Conversely, if σ_x was less than σ_Y, I placed vertex v in set Y. Once the loop was finished, all vertices were assigned to their respective sets. Finally, I calculated the weight of the cut based on this partitioning.

Semi Greedy Algorithm: This algorithm cleverly combines randomness with a general greedy approach to compute the maximum cut in a graph. To achieve this, two Restricted Candidate List (RCL) sets are maintained, one for edges and the other for vertices. Initially, an RCL of edges is formed using a partially greedy randomized weight criterion. Subsequently, an edge is selected from the created RCL, and both set X and set Y are initialized using this edge. The process then enters a while loop to handle the remaining vertices. During each iteration of the loop, an RCL of vertices is computed, and a vertex is randomly chosen from this RCL. The chosen vertex is then inserted into either set X or set Y based on a greedy evaluation.

Randomized Algorithm: This algorithm bears resemblance to the Semi-Greedy algorithm, yet it distinguishes itself by diminishing its strict adherence to greediness and infusing an element of randomness. By adjusting the criteria for selecting vertices and edges (with α set to 0), the Restricted Candidate List (RCL) sets are populated with choices that are purely random. Opting for selections from these RCLs ensures that the final outcome consistently demonstrates randomized characteristics.

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;
```

Local Search:

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 . 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