# Max Cut Problem Using GRASP

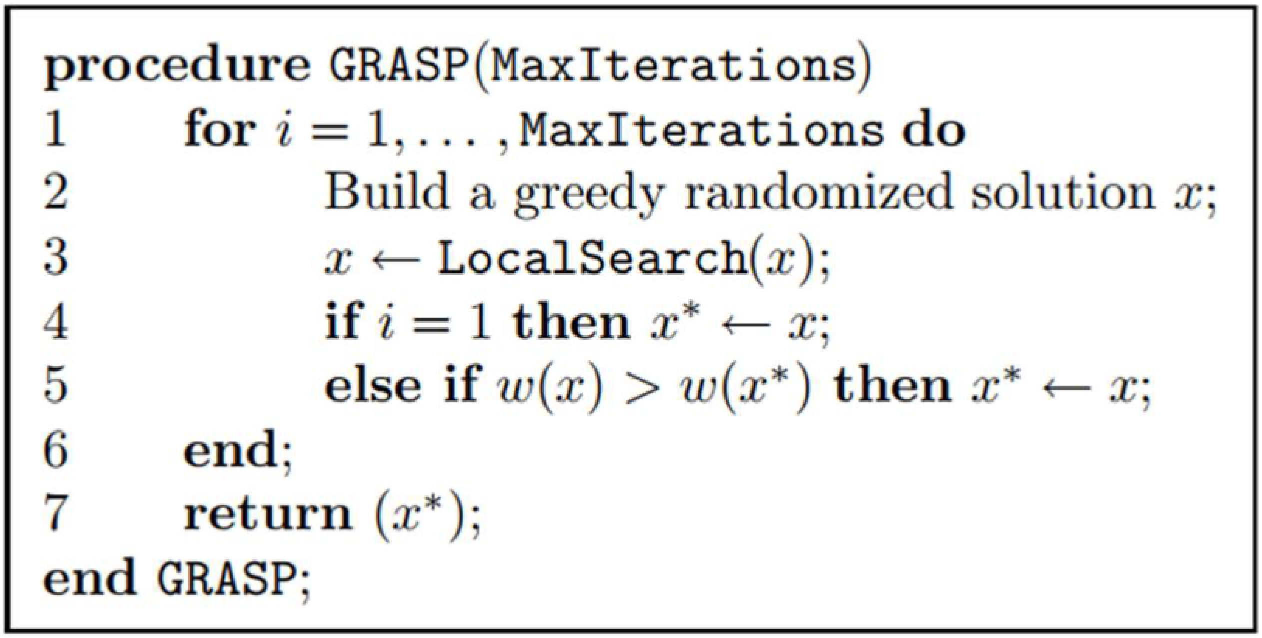
In the assignment, a set of inputs were given. Each input contained an undirected weighted graph in the form of a text file whose first line consisted of two integers: n, m. Here n is the number of vertices (order) and m is the number of edges (size) of the input graph. Each of the next m lines contains three integers describing an edge: V1 V2 Wt.

Here V1 and V2 are the endpoints of the edge which has the weight Wt

The task was to implement the GRASP (Greedy Randomized Adaptive Search Procedure) algorithm on this graph and try to find the max cut of the graph. As the max cut problem is NP hard so it is tried to analyze the results found using GRASP algorithm with respect to results found using other algorithms such as Semi Greedy, Greedy and Randomized. Local Search was also used to make the initial solution better and all of these results were compared with the best known solution for the given input.

## 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)
    for $i = 1, \ldots ,$ MaxIterations do
        Build a greedy randomized solution $x$;
        $x \leftarrow$ LocalSearch($x$);
        if $i = 1$ then $x^* \leftarrow x$;
        else if $w(x) > w(x^*)$ then $x^* \leftarrow x$;
    end;
    return $(x^*)$;
end GRASP;
```

**Pseudo-code of a generic GRASP**

The code implements three constructive algorithms:

**Greedy – 1:**

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.

**Randomized – 1:**

This algorithm works like a coin toss algorithm. We toss a coin, choose a vertex, and assign it to a set X or Y (X if outcome of toss is Heads otherwise Y). The work is done randomly, and the vertex has the same probability to go to any of the sets. After assigning all vertices to a set, the max cut is calculated.

**Semi Greedy – 1:**

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 the (partially constructed) solution.

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

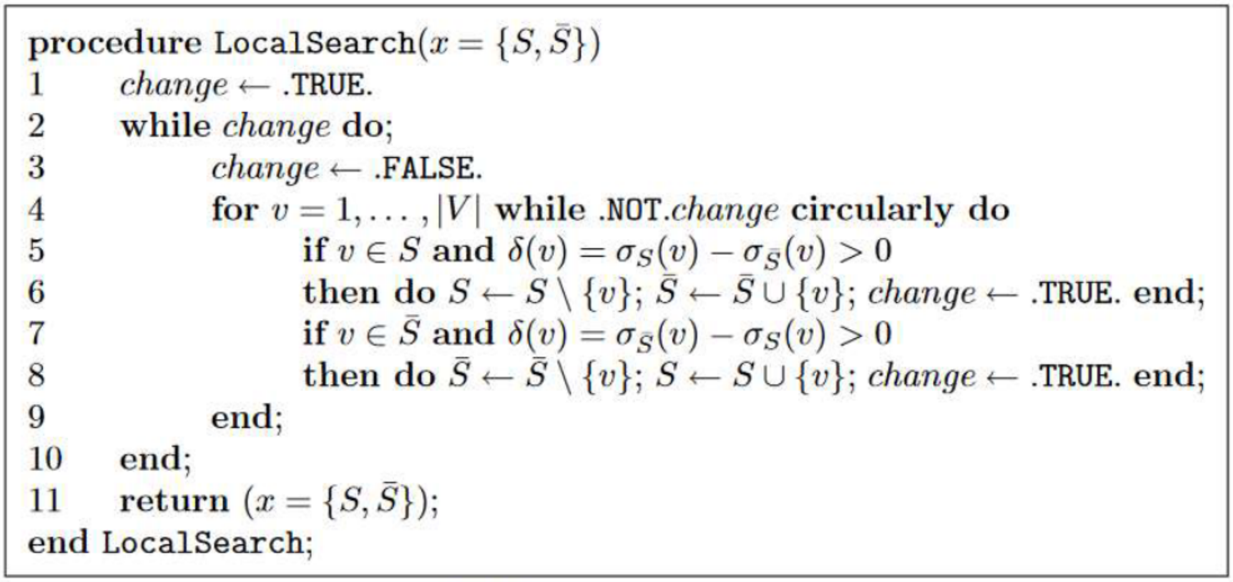
$$w_{min} = \min\{min_{v \in V'} \sigma_X(v), min_{v \in V'} \sigma_Y(v)\}$$

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

The code implements Local Search and GRASP algorithm to improve the performance.

**Local Search:**

This algorithm is used after getting a cut on the above mentioned three construction algorithms in the code. This algorithm chooses a vertex and checks if the cut can be increased when it is shifted to the other set. If the cut increases, the vertex is shifted to the other set. This process runs until we find a configuration where the current solution cannot be improved. Then we calculate the max cut. In the code, we have applied Local Search 20, 50, 100, 500, 1000 times in GRASP and taken the average value for local search result. The pseudocode for local search algorithm that is used is shown below:



```
procedure LocalSearch($x = \{S, \bar{S}\}$)
    $change \leftarrow$ .TRUE.
    while $change$ do;
        $change \leftarrow$ .FALSE.
        for $v = 1, \ldots, |V|$ while .NOT.$change$ circularly do
            if $v \in S$ and $\delta(v) = \sigma_S(v) - \sigma_{\bar{S}}(v) > 0$
            then do $S \leftarrow S \setminus \{v\}$; $\bar{S} \leftarrow \bar{S} \cup \{v\}$; $change \leftarrow$ .TRUE. end;
            if $v \in \bar{S}$ and $\delta(v) = \sigma_{\bar{S}}(v) - \sigma_S(v) > 0$
            then do $\bar{S} \leftarrow \bar{S} \setminus \{v\}$; $S \leftarrow S \cup \{v\}$; $change \leftarrow$ .TRUE. end;
        end;
    end;
    return ($x = \{S, \bar{S}\}$);
end LocalSearch;
```

**Pseudo-code of the local search phase**

## GRASP:

GRASP stands for "Greedy Randomized Adaptive Search Procedure," and it is a metaheuristic approach used to solve combinatorial optimization problems. Combinatorial optimization problems involve finding the best solution from a finite set of possible solutions, where the search space is often huge and discrete. The GRASP algorithm is designed to find high-quality solutions by combining greedy construction and local search techniques. It runs the Semi Greedy and Random algorithm for construction phase and Local Search algorithm to get better solutions for a number of iterations controlled by the user. The code runs the algorithm for 20, 50, 100, 500, 1000 times and takes the best value for max cut from it. The pseudocode for the GRASP algorithm that is used is already shown in the first page.

## Explanation of Output:

For the assignment, a max cut of 37 graphs are found using above mentioned algorithms and all of their data is collected and written to a csv file. For each graph the max iteration of GRASP algorithm was varied in the range [20, 50, 100, 500, 1000] and for each of them 11 different alpha values were used starting with alpha = 0.0, 0.1, 0.2, 0.3, … which continued up to 1.0 .

The whole data can be found in a table in the below google sheets link:

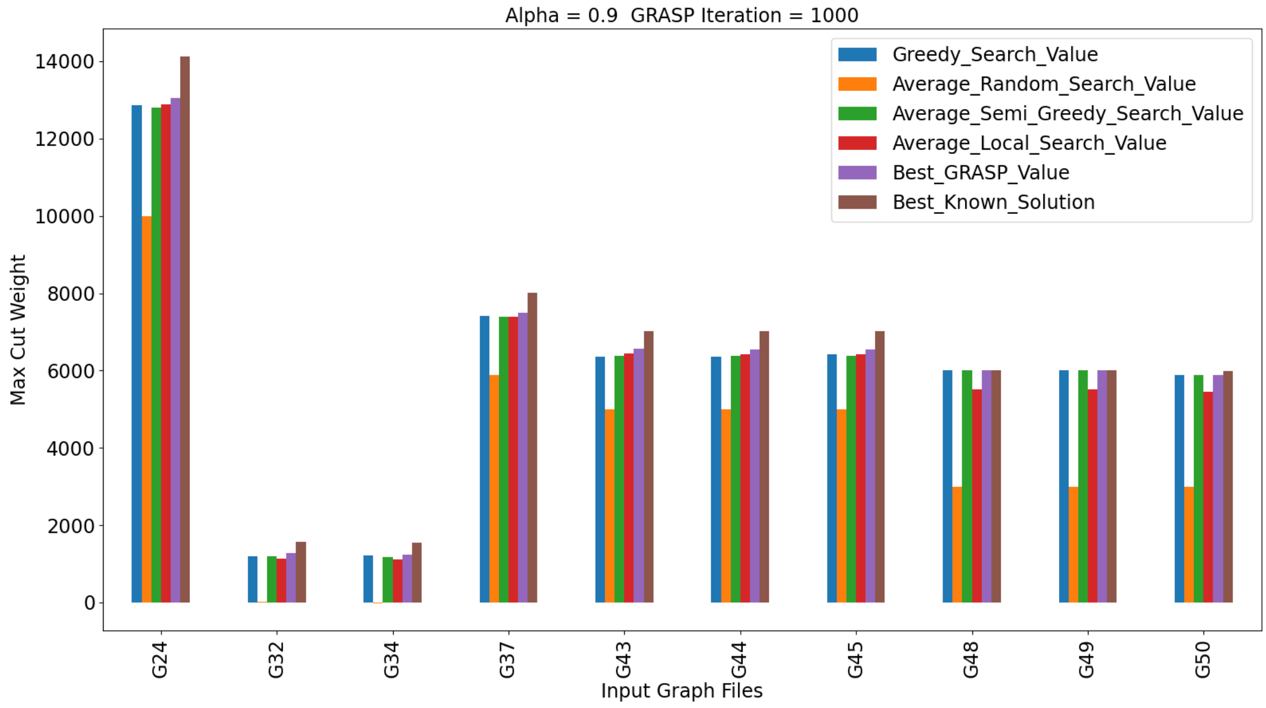
1905118_CSE318_Offline3_Report_Data

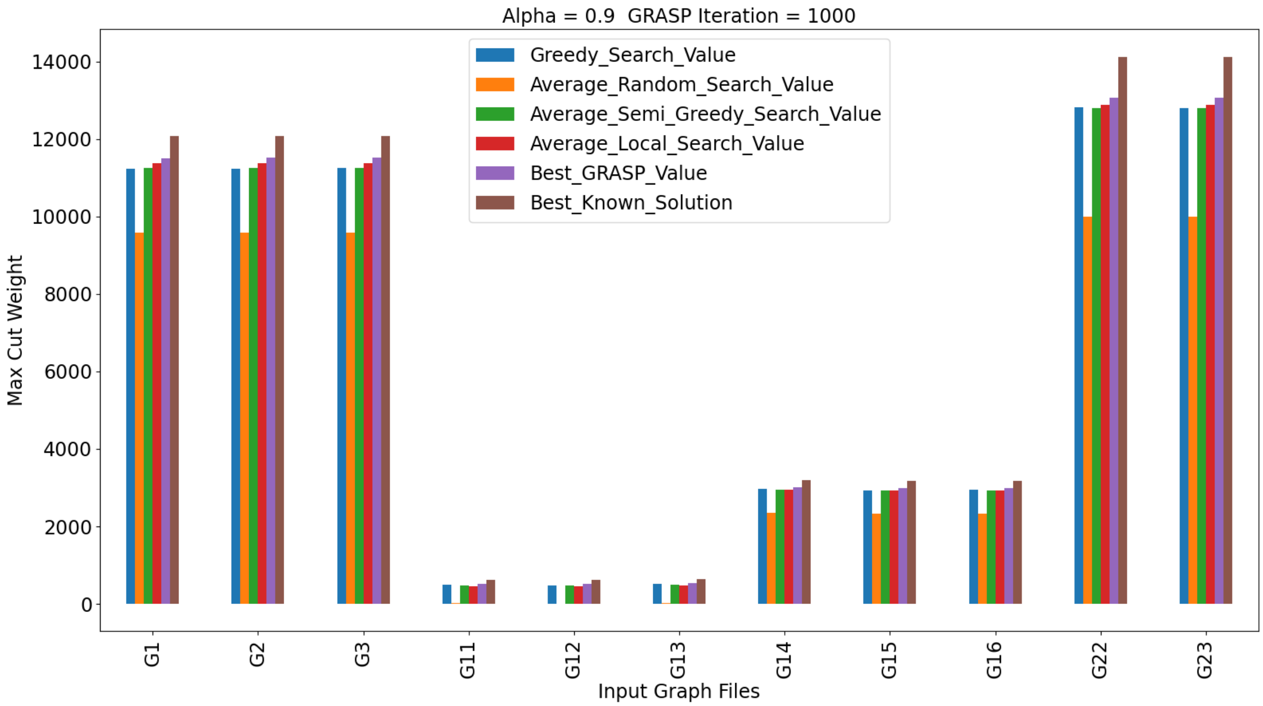
The csv file contains the average value of 20, 50, 100, 500, 1000 iterations for different alpha values from Randomized -1, the max cut value from Greedy – 1, average value of Semi Greedy max cut, average Local Search iterations and average Local Search max cut for 20, 50, 100, 500, 1000 GRASP iterations. The best value of GRASP for these iterations and the known best solution is also provided.

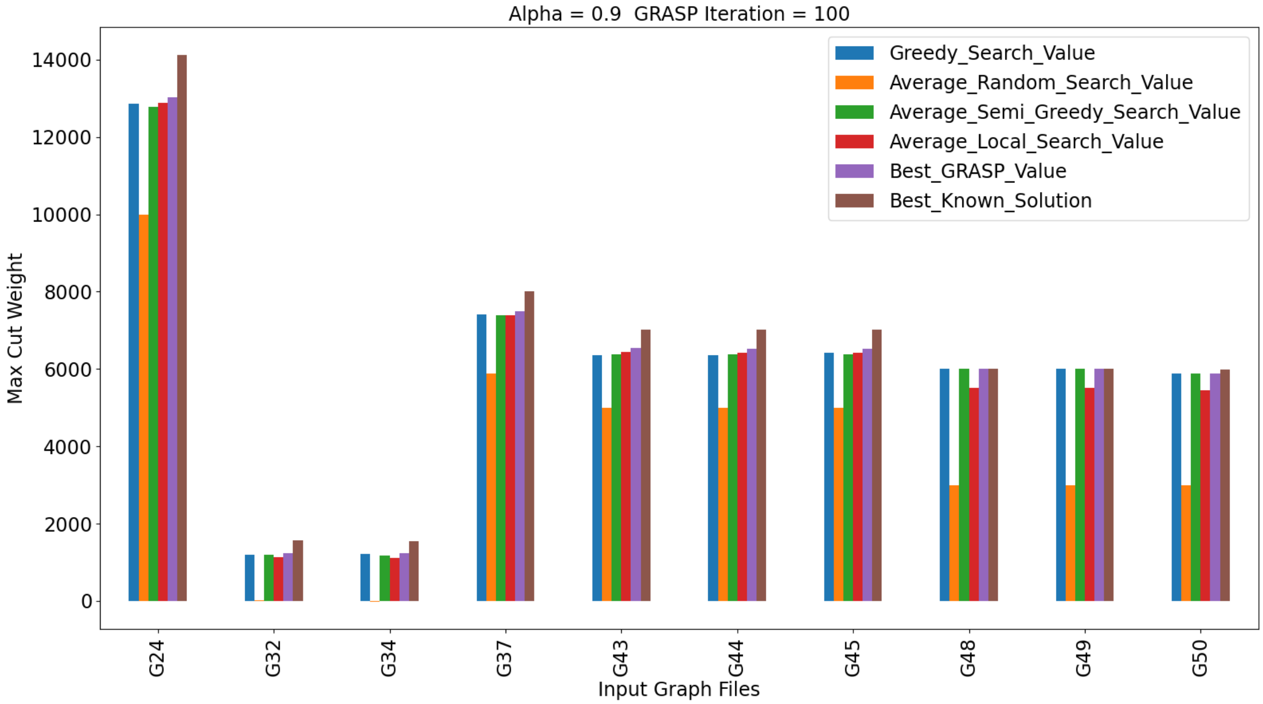
From the csv, it is evident that the Randomized – 1 algorithm provides the worst solution of max cut as it chooses the vertices randomly without any prior calculation. Then there is the Semi Greedy algorithm, as only the average of maximum iterations is taken, most of the time it gives a worse result than the Greedy algorithm. Local Search improves the Semi Greedy approach and gives a better solution than Greedy most of the time. Then there is GRASP algorithm, as the best value is taken from iterations of Semi Greedy and Local Search, it provides the best result among all of them. The dataset proves that

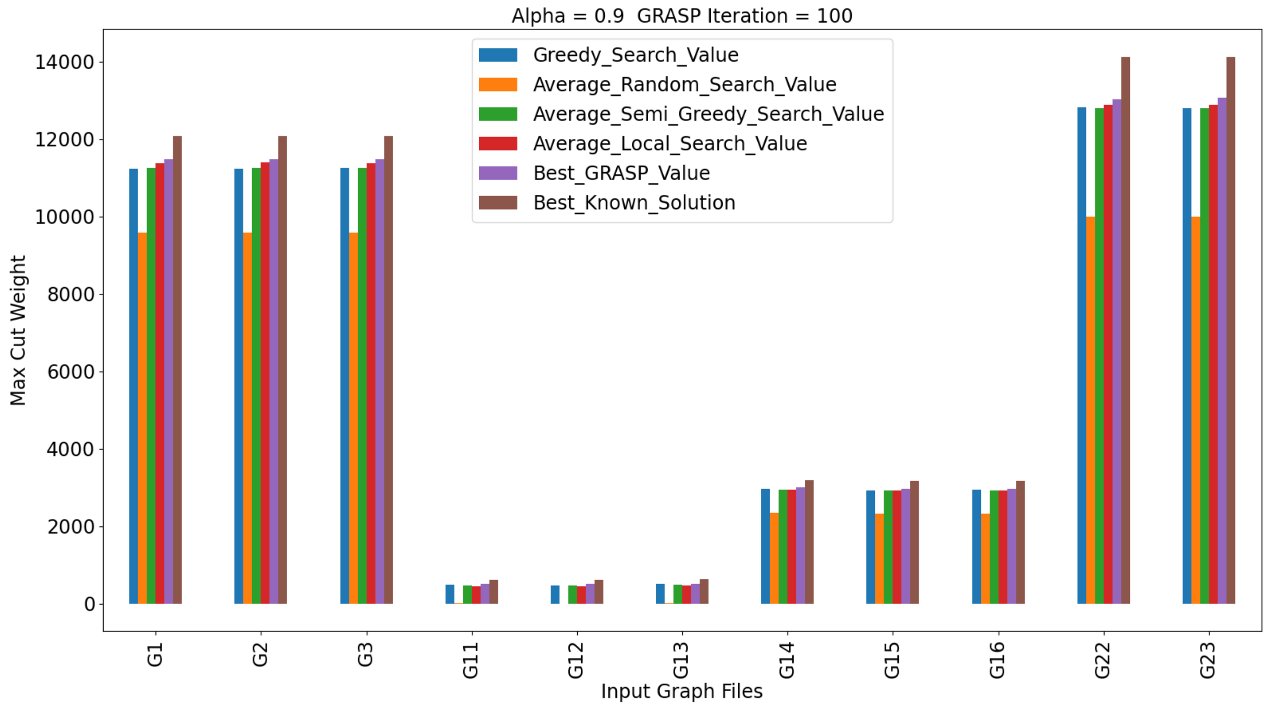
The GRASP algorithm is better than the Greedy algorithm. Besides from the csv, we can see that if we increase the iterations of GRASP, most of the time the max cut improves, though the improvement may be very low after some iterations. The GRASP algorithm gives quite a close value with comparison to the known best result of the inputs. Example: it provides greater than 90% of the best known result for almost all of the executed inputs. So, we can conclude that the GRASP algorithm works better than all other algorithms for the max cut problem.

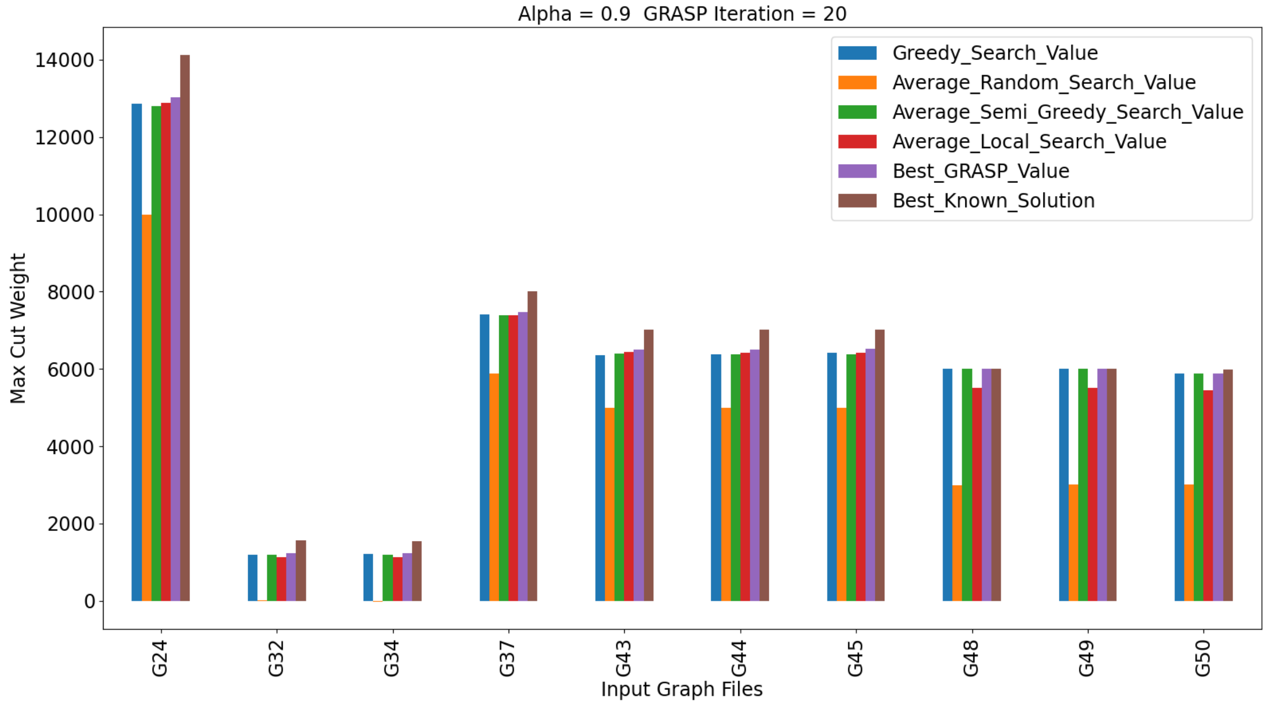
Below some graphs are being attached which show a comparison for the results obtained by various approaches for different alpha and GRASP iteration values. The graph also agrees with what is said above. Also for some cases the Randomized algorithm gives us negative weights. It is due to the fact that most of its edges which are crossing the two partition sets are of negative weights. As almost 50% of nodes fall into one set in the Randomized algorithm that is used here so it is likely possible to see such results.

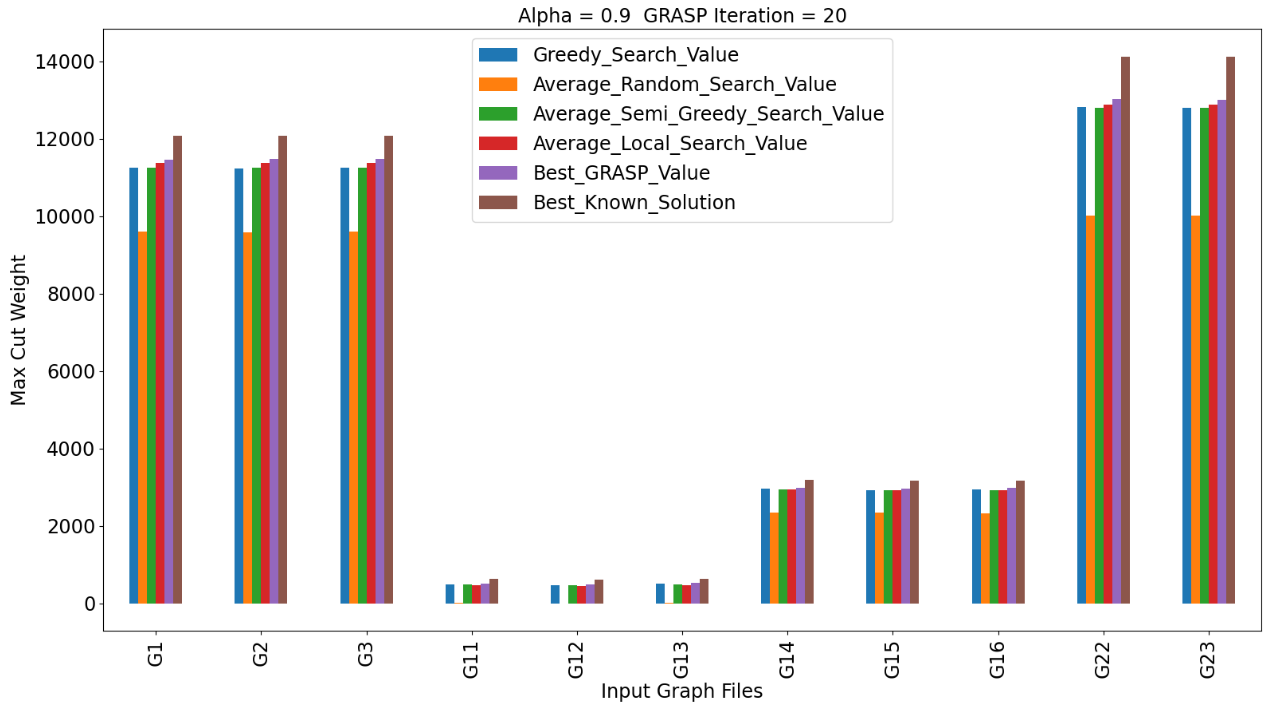


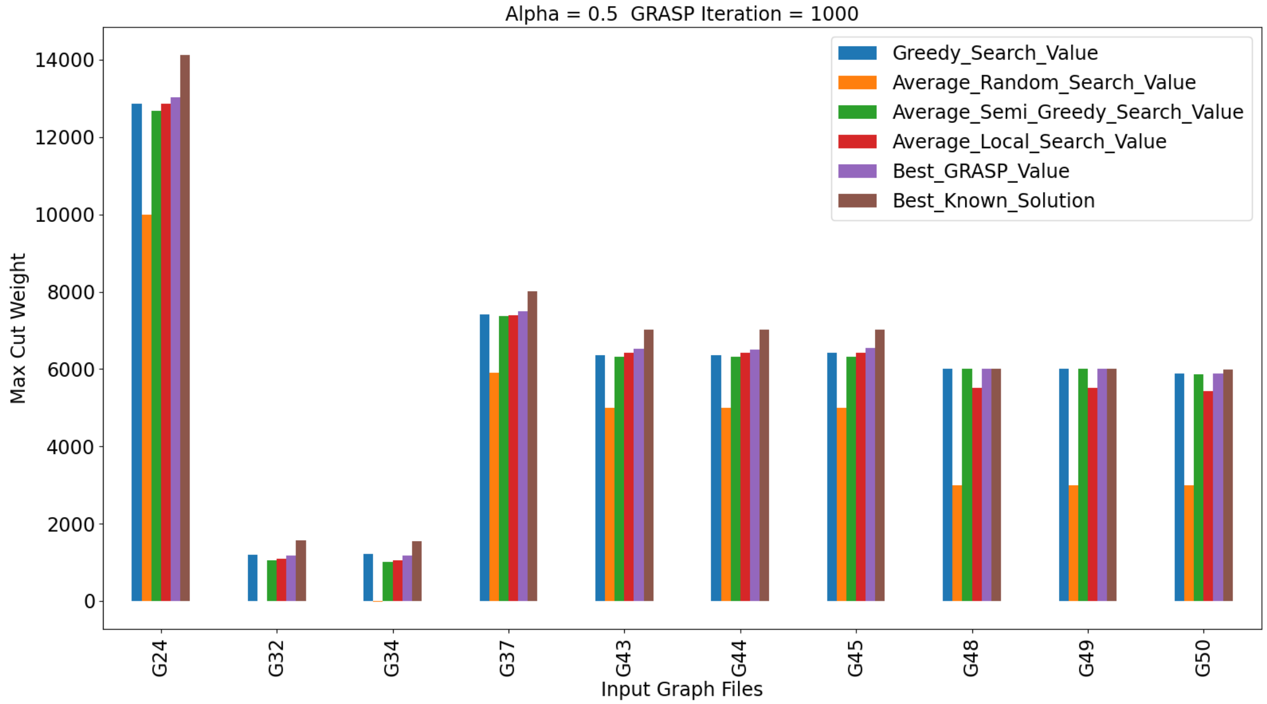

Alpha = 0.9  GRASP Iteration = 1000
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files

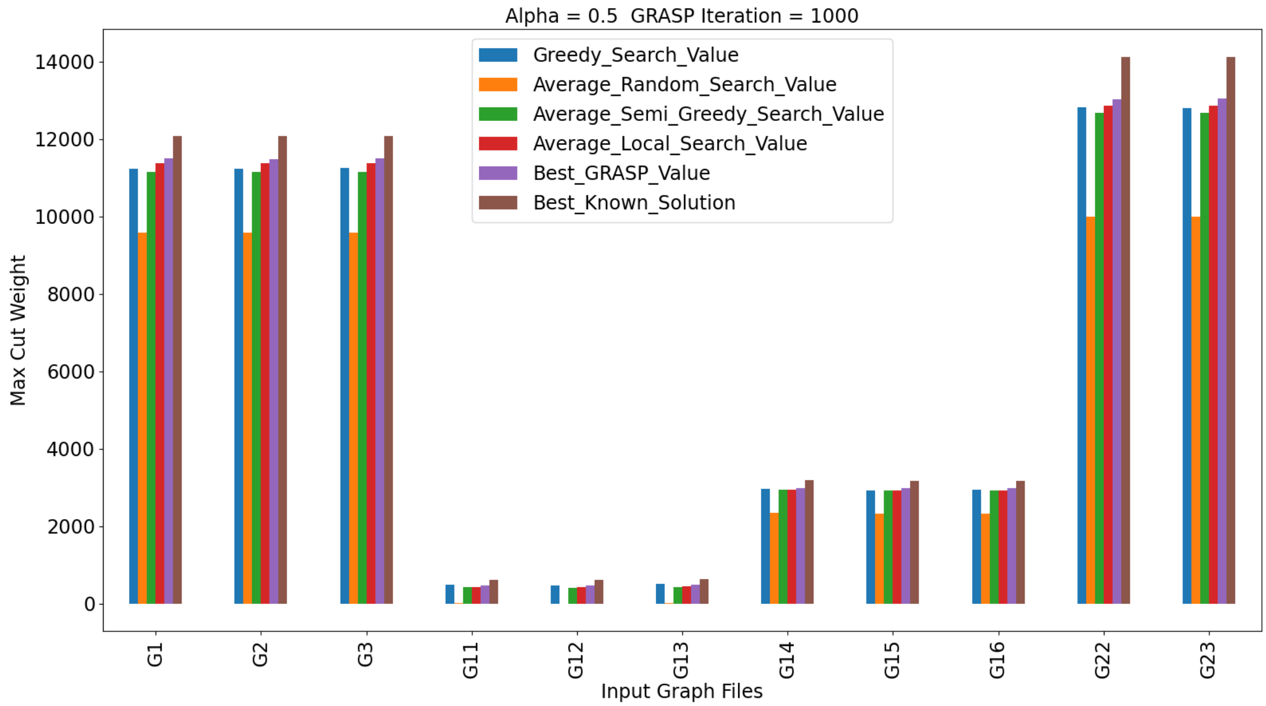

Alpha = 0.9  GRASP Iteration = 100
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files

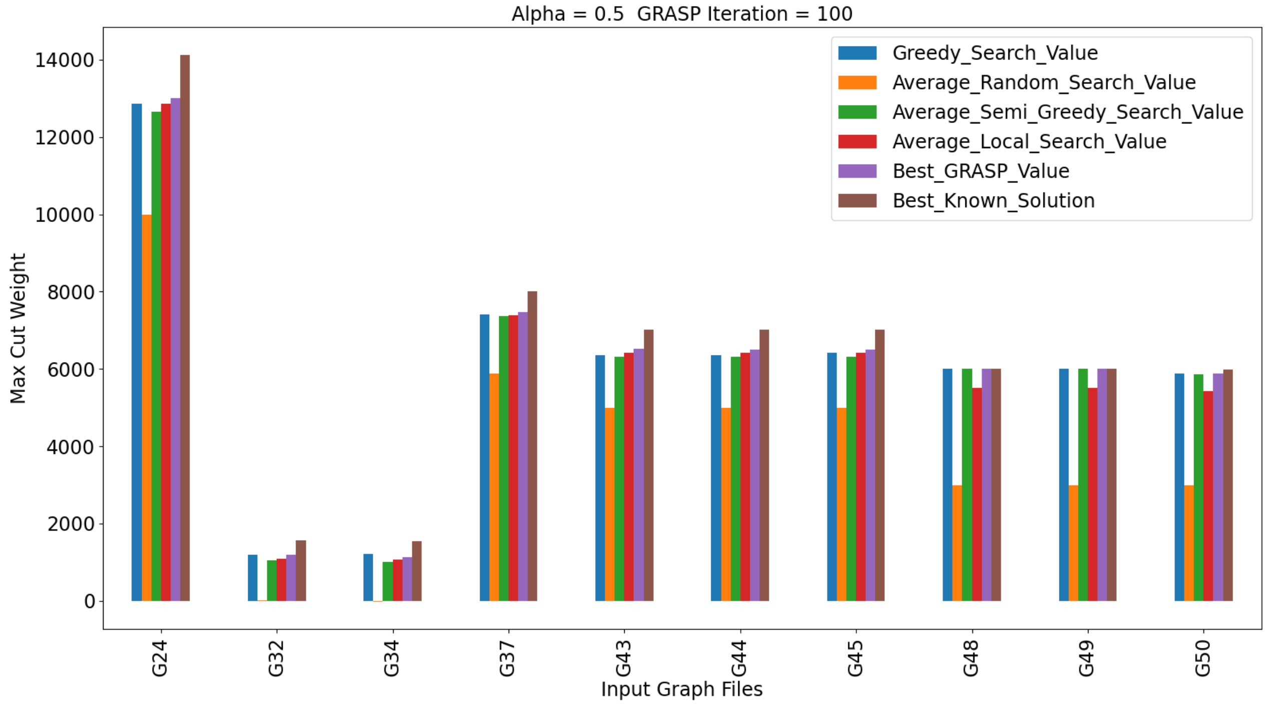

Alpha = 0.9  GRASP Iteration = 100
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files

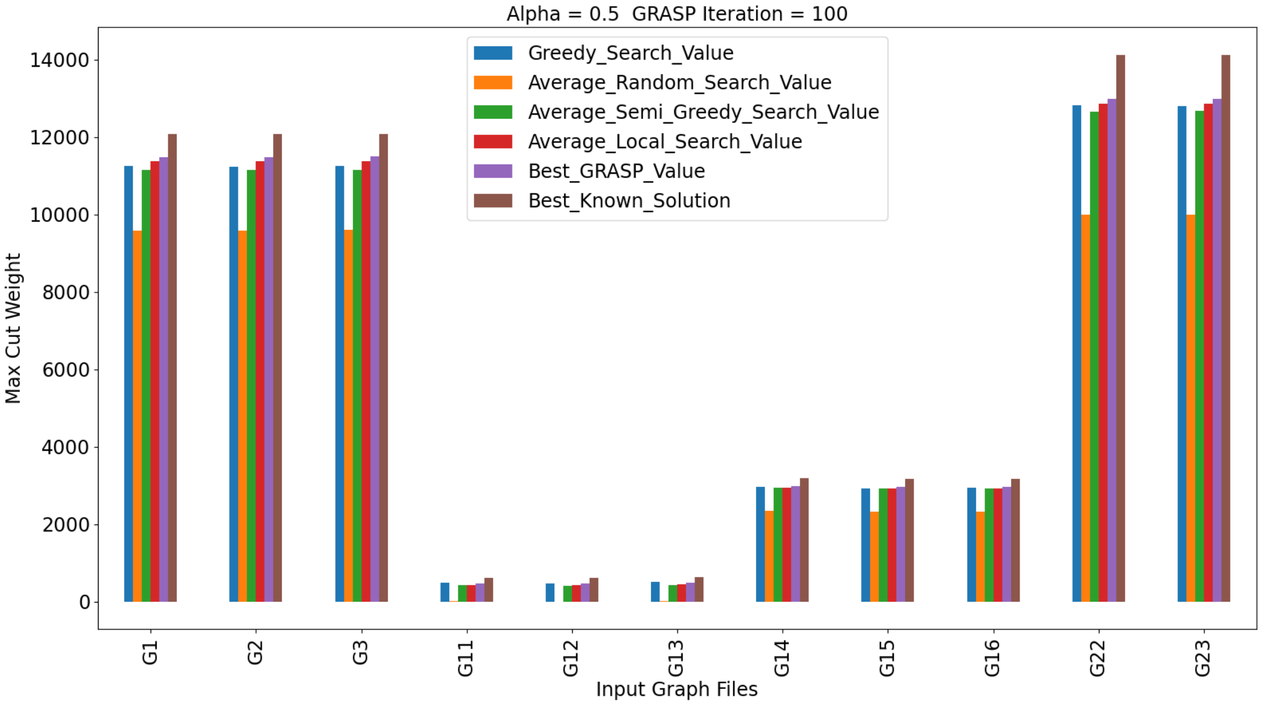

Alpha = 0.9  GRASP Iteration = 20
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files

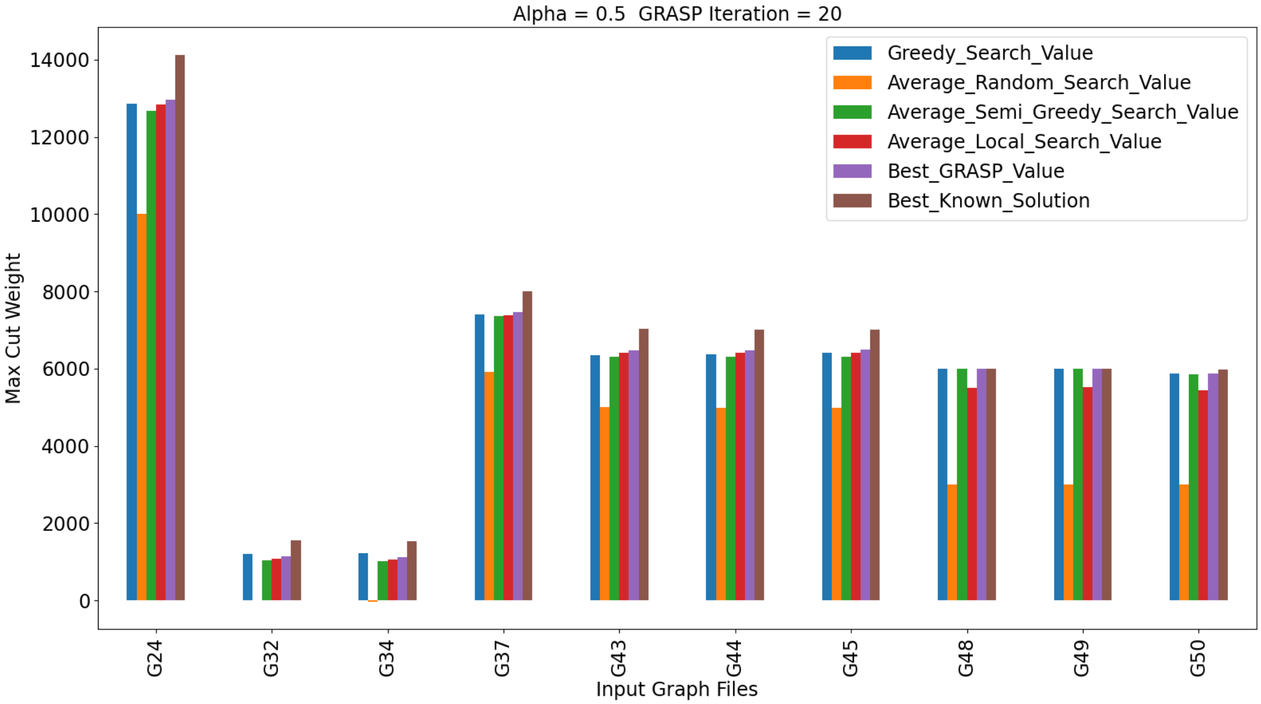

Alpha = 0.9  GRASP Iteration = 20
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files

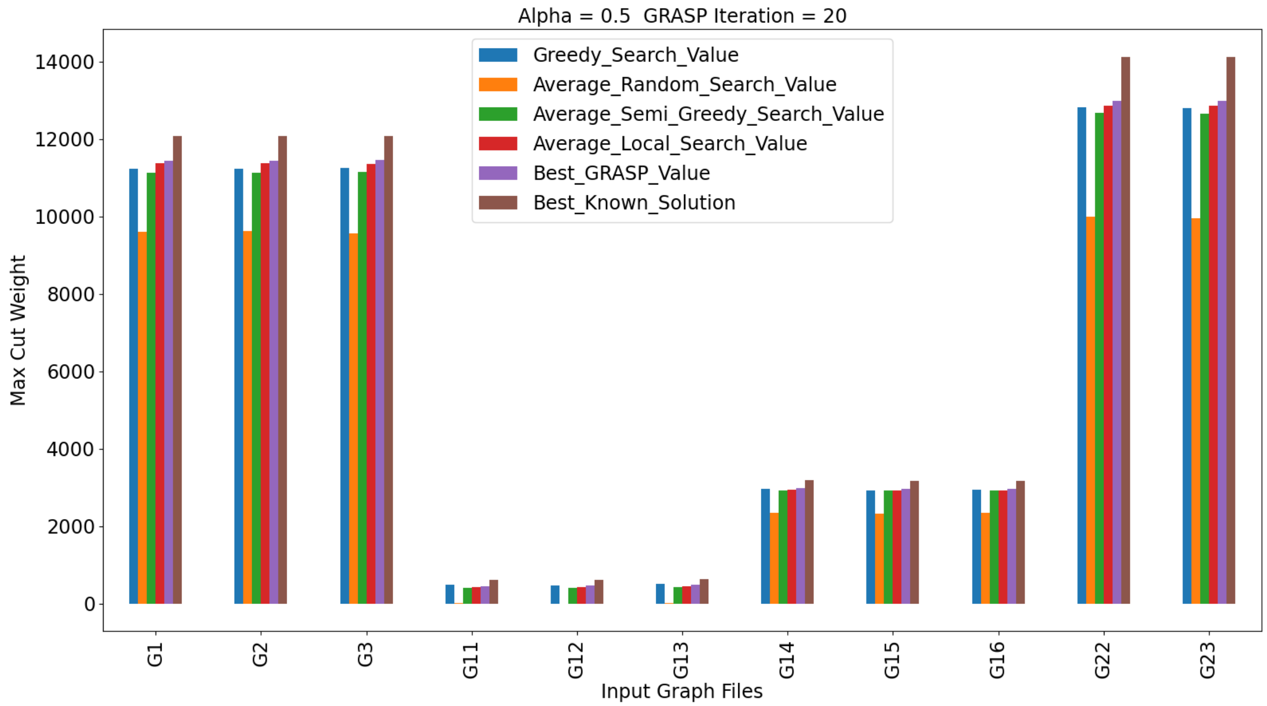

Alpha = 0.5  GRASP Iteration = 1000
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files

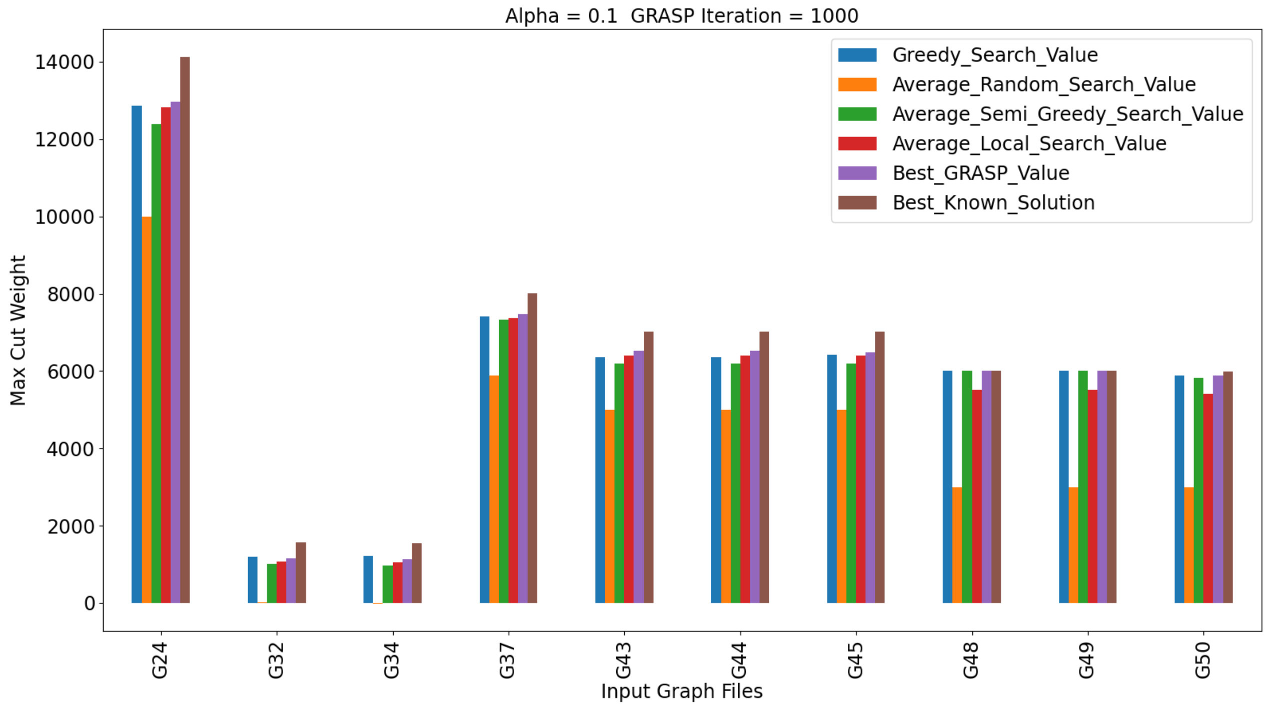

Alpha = 0.5  GRASP Iteration = 1000
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files

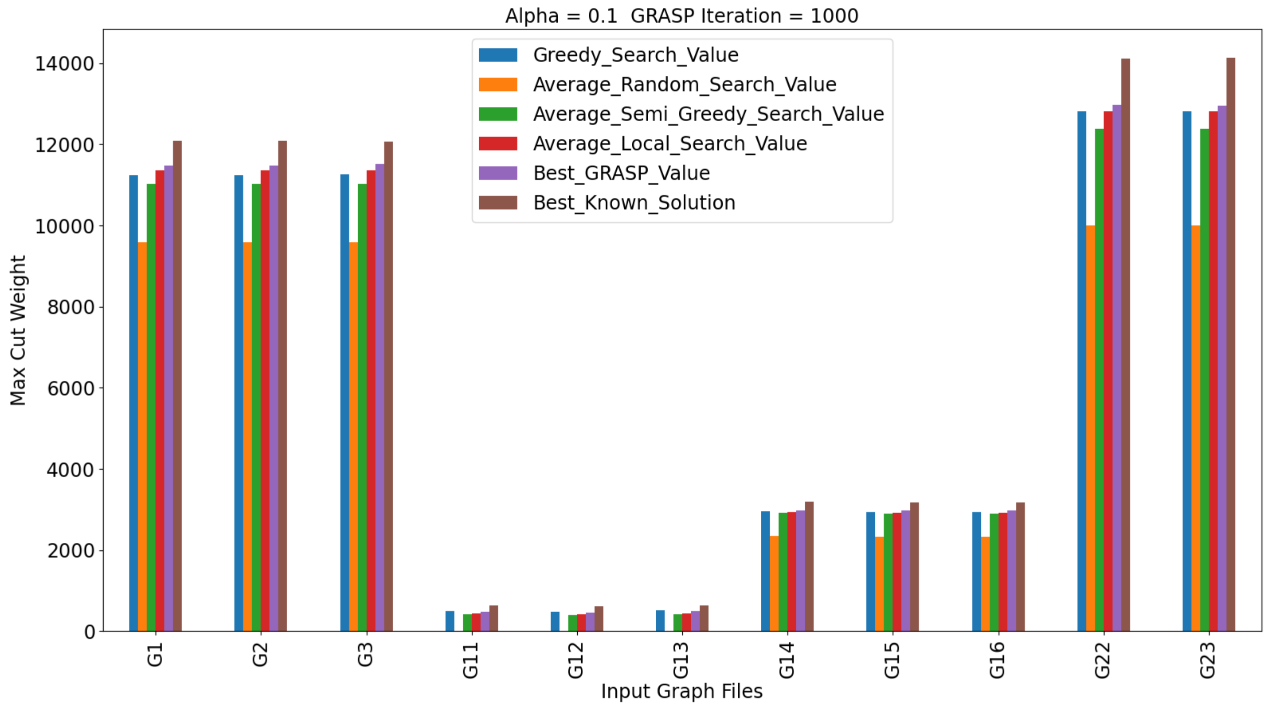

Alpha = 0.5  GRASP Iteration = 100
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files

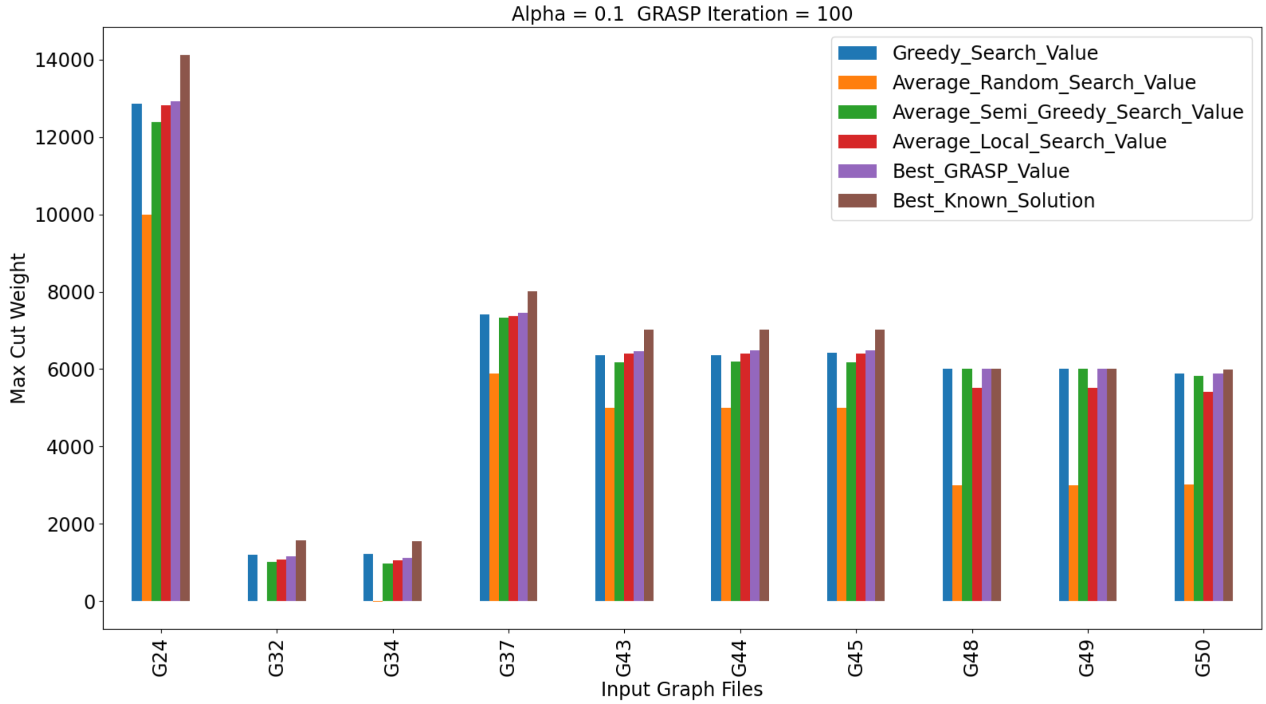

Alpha = 0.5  GRASP Iteration = 100
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files

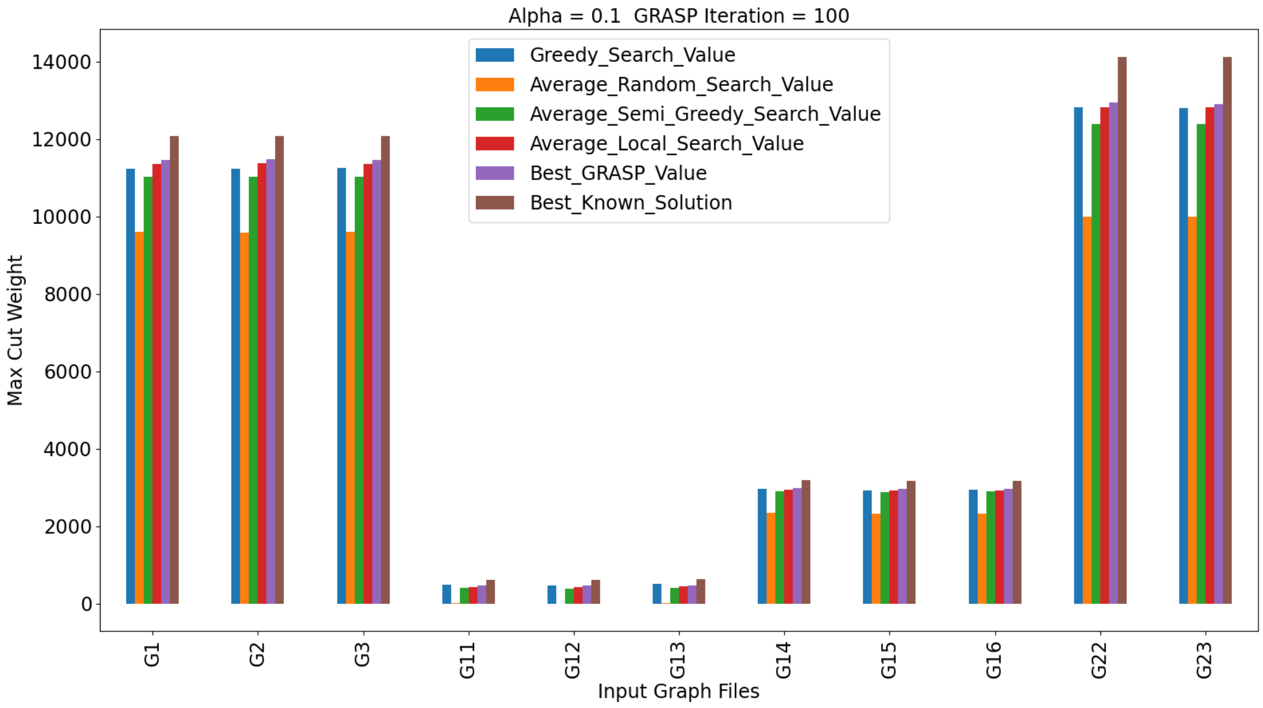

Alpha = 0.5  GRASP Iteration = 20
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files

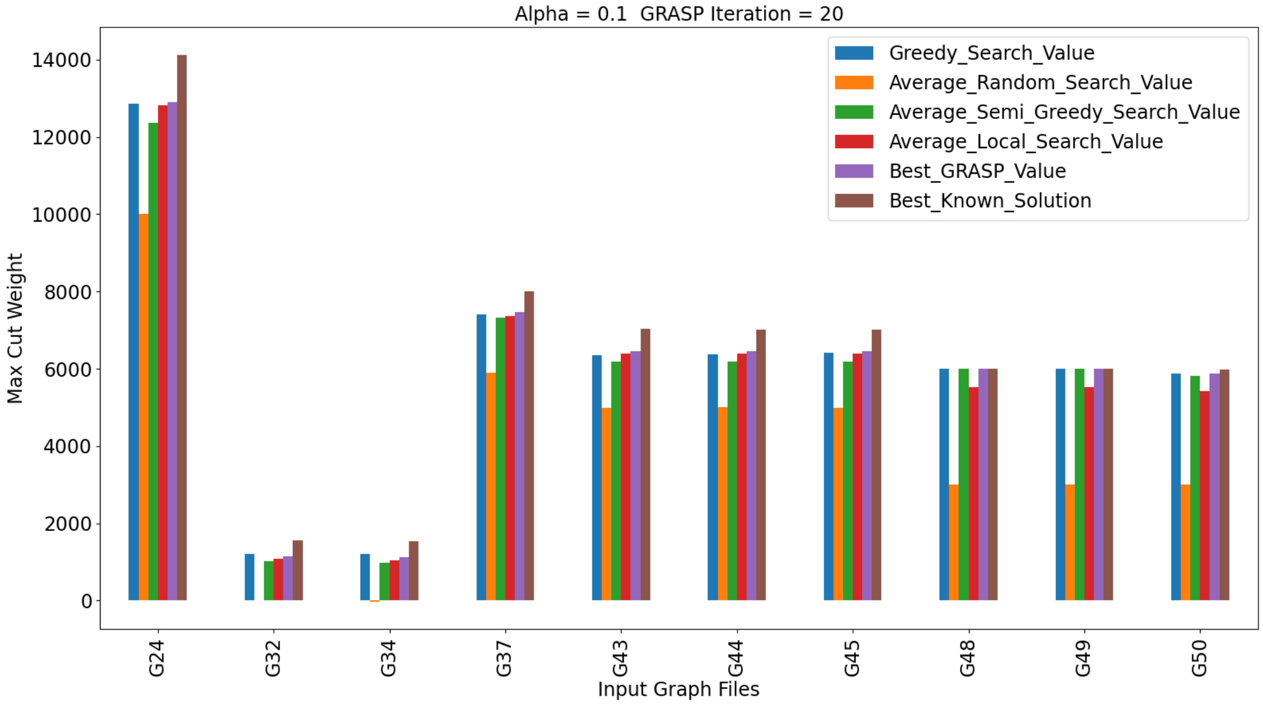

Alpha = 0.5  GRASP Iteration = 20
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files

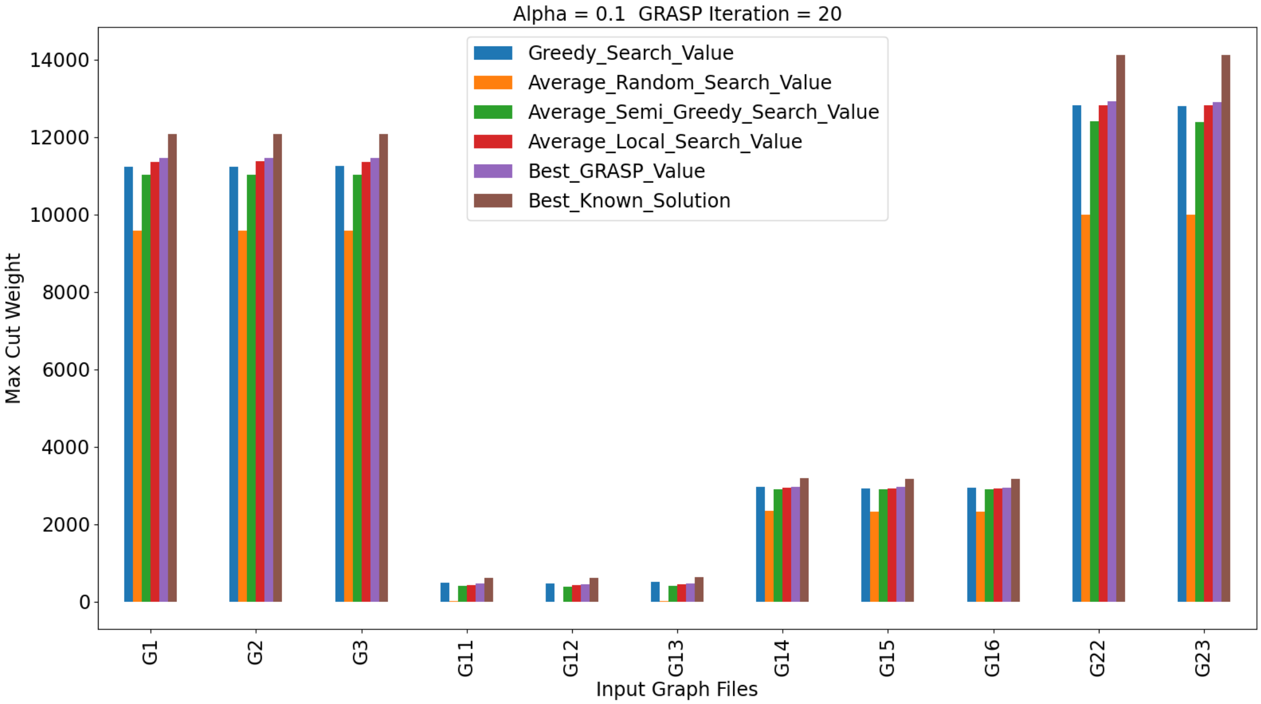

Alpha = 0.1  GRASP Iteration = 1000
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files


Alpha = 0.1  GRASP Iteration = 1000
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files


Alpha = 0.1  GRASP Iteration = 100
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files


Alpha = 0.1  GRASP Iteration = 100
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files


Alpha = 0.1  GRASP Iteration = 20
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
G24
G32
G34
G37
G43
G44
G45
G48
G49
G50
Input Graph Files


Alpha = 0.1  GRASP Iteration = 20
Greedy_Search_Value
Average_Random_Search_Value
Average_Semi_Greedy_Search_Value
Average_Local_Search_Value
Best_GRASP_Value
Best_Known_Solution
Max Cut Weight
0
2000
4000
6000
8000
10000
12000
14000
G1
G2
G3
G11
G12
G13
G14
G15
G16
G22
G23
Input Graph Files