|  |
| --- |
| **CSE 318 Assignment-03: Solving the Max-cut problem by GRASP** |
| **Student ID: 1905003, Section: A1** |
| A brief report which summarizes and explains the output of the implemented solution. In this report, it is briefly explained which greedy or semi-greedy techniques or local search operators were implemented. |

**8/18/2023**

**Program/Implementation:**

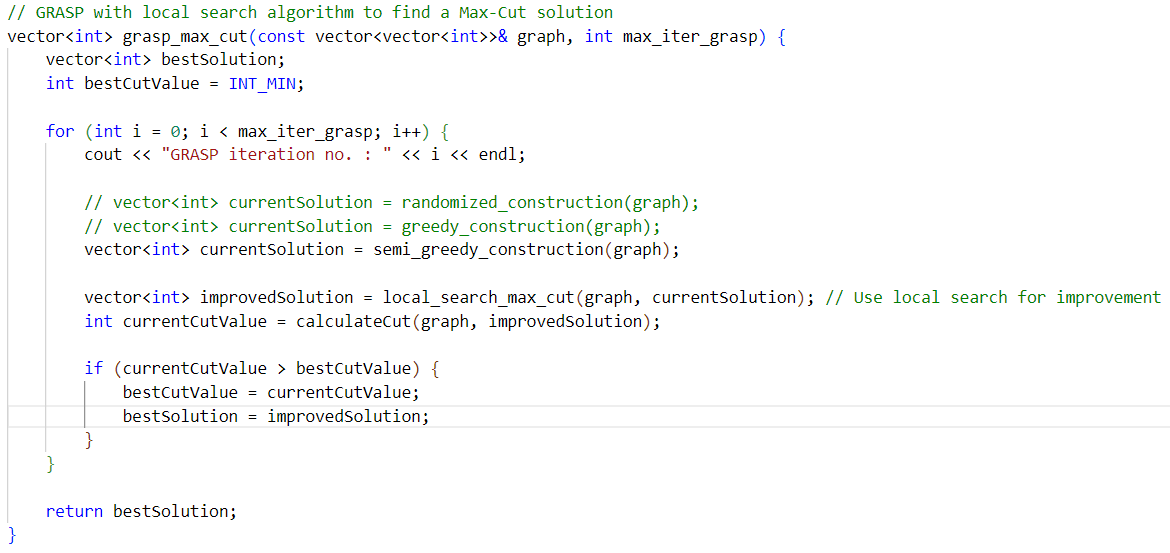


Figure GRASP implementation

In the implementation for GRASP, the construction phase can be implemented with three heuristic – randomized, greedy and semi-greedy. The number of iterations is obtained via the parameter *max\_iter\_grasp*. After getting currentSolution from the construction phase heuristic, the solution is improved by calling the method-

vector<int> local\_search\_max\_cut(const vector<vector<int>>& graph, vector<int> initialPartition)

This function takes the graph and an initialSolution as arguments from the method in Figure 1. The initalSolution is actually a vector describing which partition each vertex is currently included in [0 or 1]. It loops over all the vertices and decides which partition to place them depending on how much cut-weight they contribute until no further improvement of any vertex is possible. The cut-weight contribution of a vertex v is stored in local variables sigma\_0 and sigma\_1. The integer i keeps count of the number of iterations of this method. k and cutValue are two global variables for counting total number of local search iterations and total cut-weight for all GRASP iterations respectively.

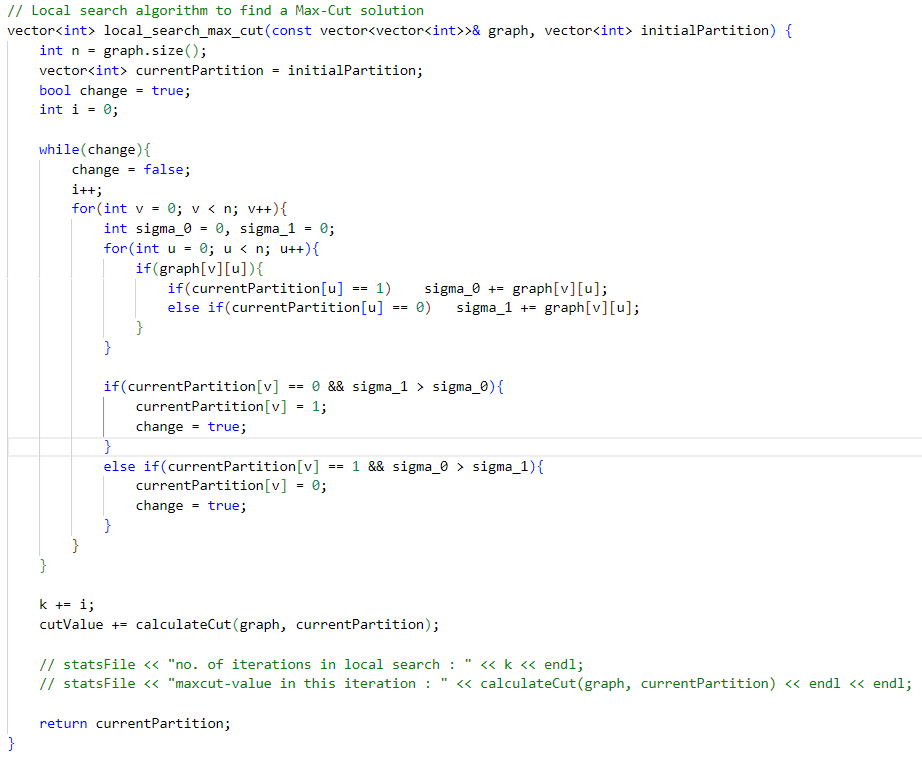


Figure Local Search Implementation

Now, the three heuristic, **greedy**, **random** and **semi\_greedy**, used in the program is described one by one.

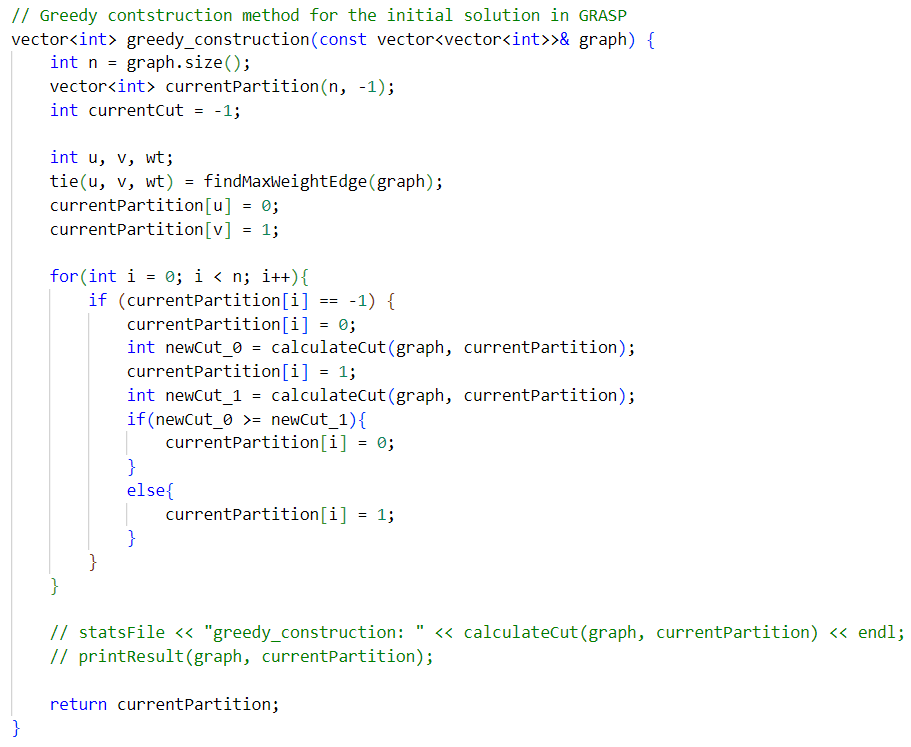


Figure Greedy Heuristic

The greedy heuristic first find the maximum-weighted edge by calling the function findMaxWeightEdge(graph) and puts the two endpoints u and v in two partitions, u in partition-0 and v in partition-1. curretnPartition was initialized to -1. Then it loops through all the vertices that doesn’t have their partition assigned yet and checks to see if the cut of the graph can be increased by including the vertex in partition-0 or partition-1. Finally, it returns resultPartition.

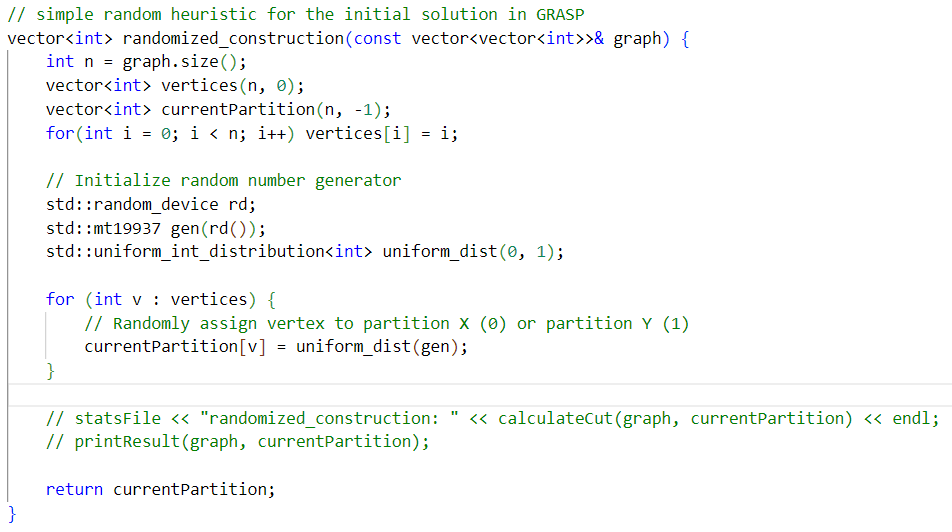


Figure Random Heuristic Implementation

currentPartition is initialized to -1 to indicate that none of the vertices have been assigned a partition yet. Then the sequence number of all the vertices is store in the vector vertices. Randomness is introduced by instantiating **Mersenne Twister 19937 Engine** using **uniform distribution**. For each vertex in vertices, its partition is randomly assigned through this object and the final currentPartition is returned.

**The Semi-greedy heuristic is implemented identically to the pseudo code.**

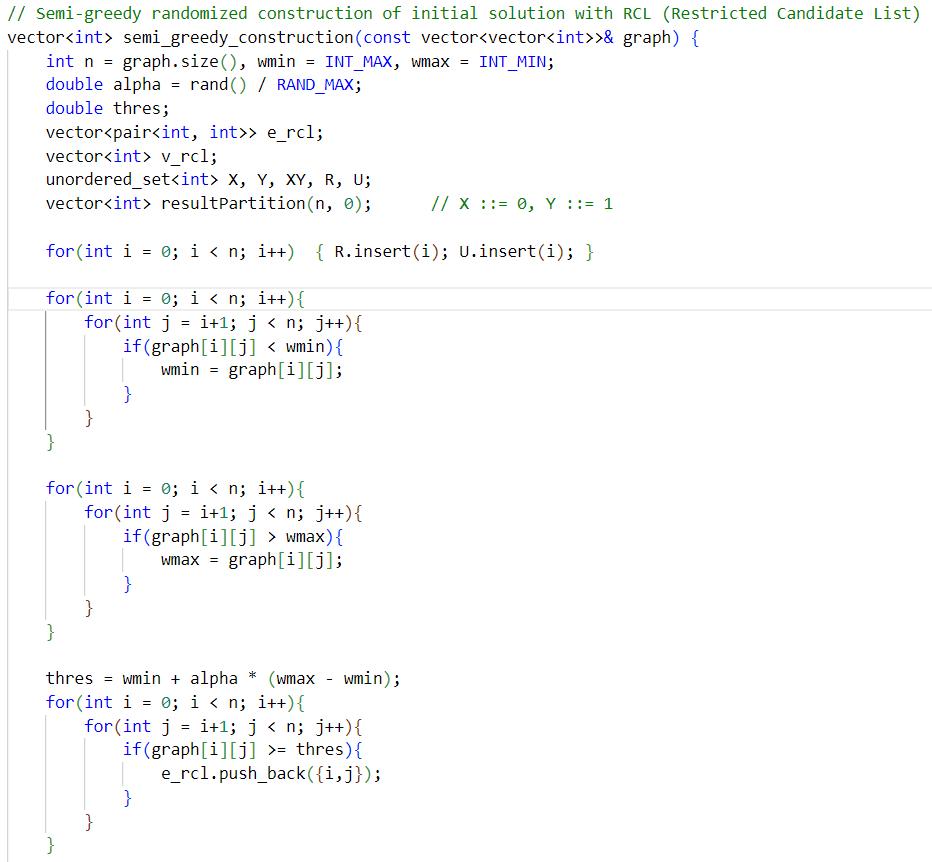


Figure Semi-greedy Heuristic Implementation part-1

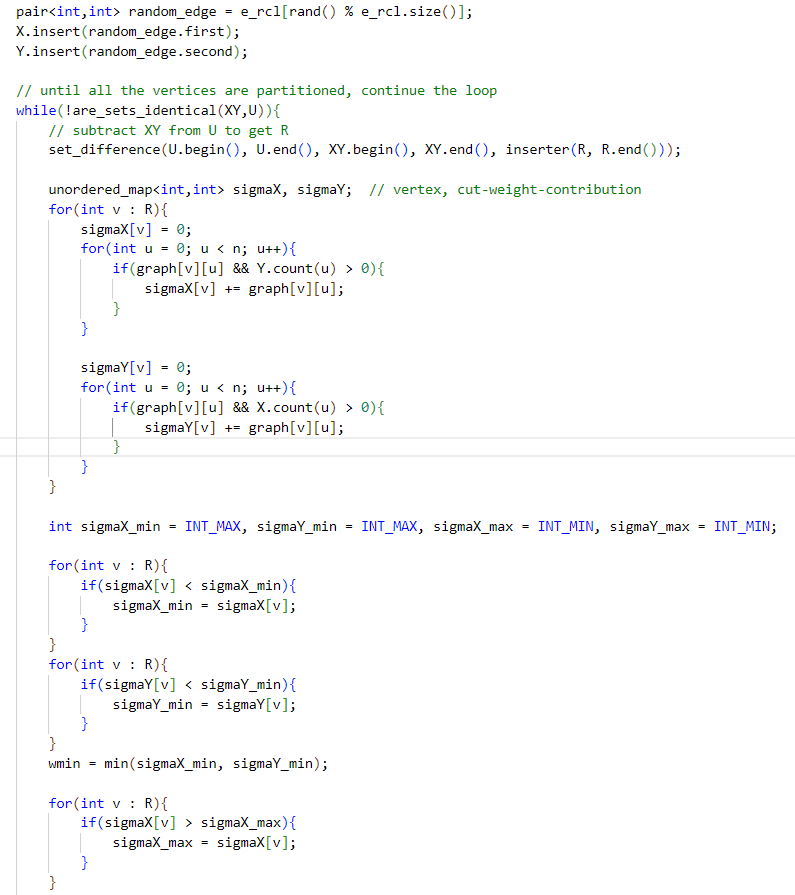


Figure Semi-greedy Heuristic Imeplementation part-2

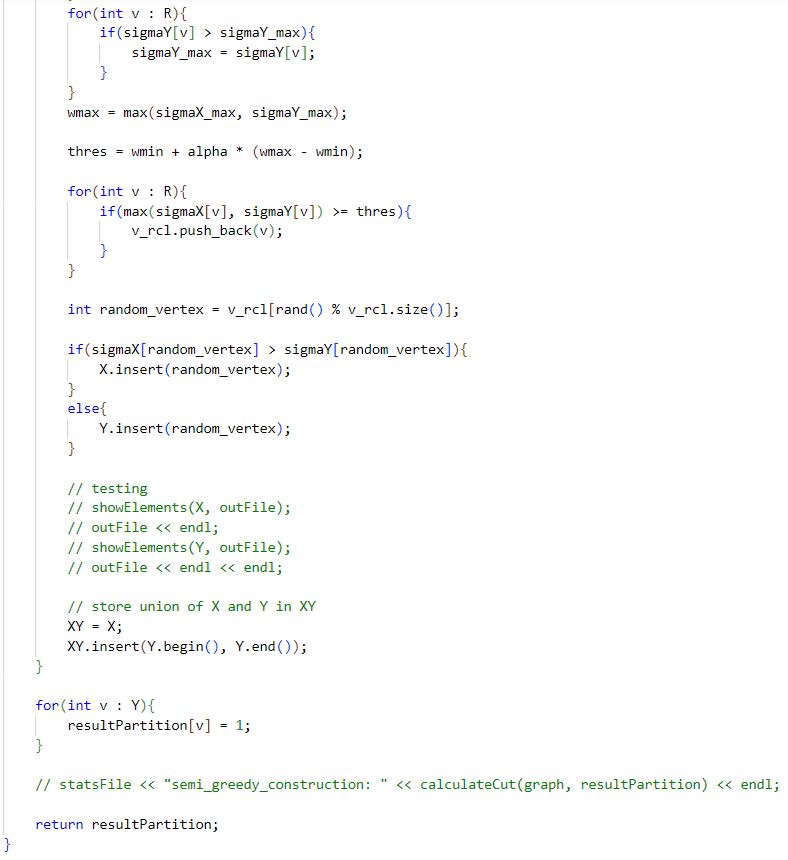


Figure Semi-greedy Heuristic Imeplementation part-3

Local variables:

**alpha** => calculated randomly by rand()/RAND\_MAX, RAND\_MAX being the maximum random number that can be generated by rand().

**thres** => corresponds to µ in the pseudo code.

**e\_rcl** => RCL for edges. Each element of e\_rcl is a pair<int,int> of vertices.

**v\_rcl** => RCL for vertices.

**X, Y** => the set of vertices included in partition-0 and partition-1 respectively. These two sets are updated at each iteration of the while loop.

**XY** => union of X and Y.

**U** => universal set of all the vertices. There is no change to this set throughout the code.

**R** => set of remaining vertices, that is, the vertices that haven’t been assigned a partition yet. This set is obtained by subtracting XY from U at the start of each iteration of the while loop.

**resultPartition** => final partition of the vertices, initially, all the vertices are placed in partion-0. At the end of the while loop, all the vertices of the graph will be processed. So, resultpartition of the vertices that belongs to set Y is changed to 1.

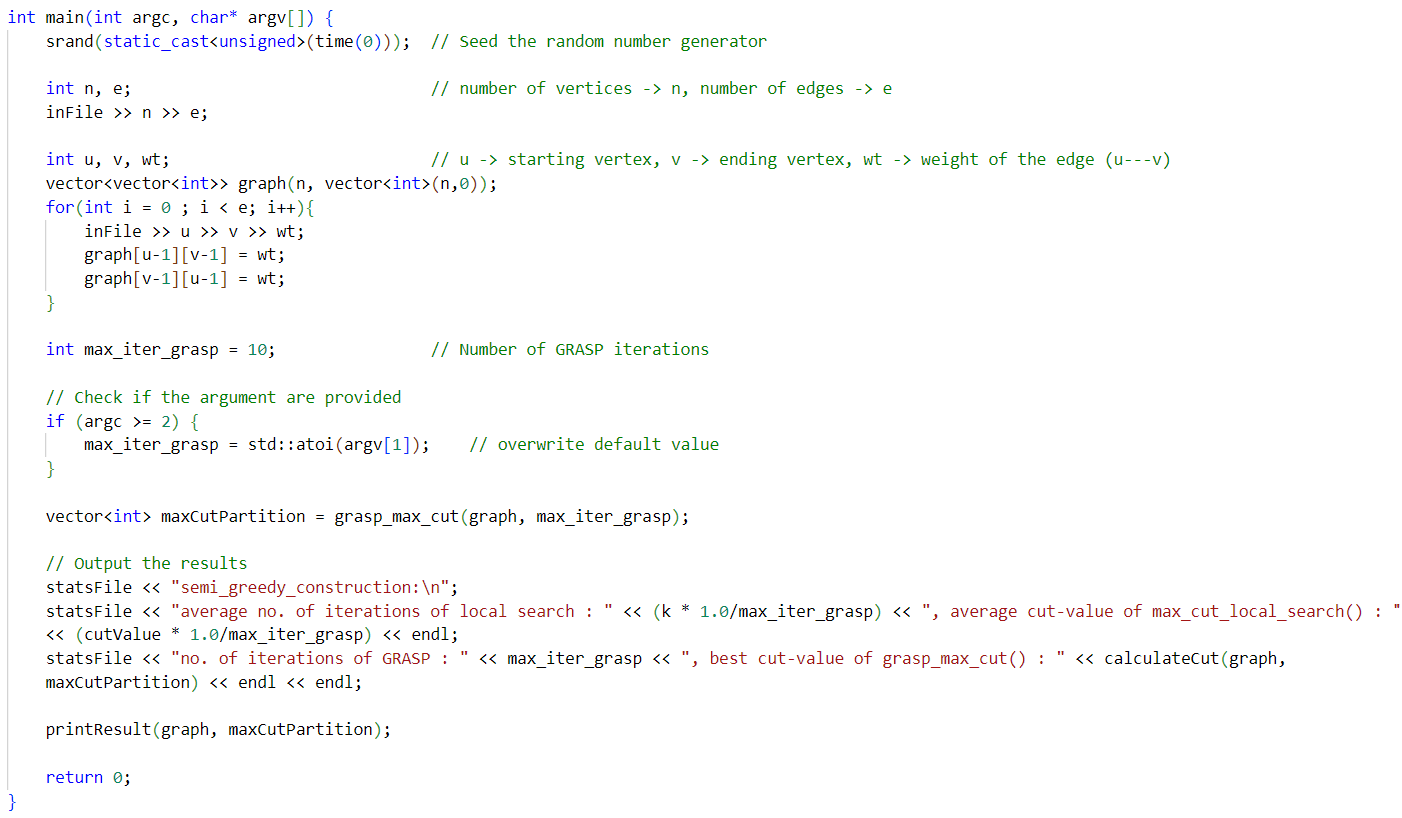


Figure main() Function

Finally, the main function takes input graph using ifstream object inFile. max\_iter\_grasp = 10 by default, if no command line argument is provided. Otherwise, the provided argument is used instead. This function calls grasp\_max\_cut(graph, max\_iter\_grasp) to obtain maxCutPartition. It also prints average number of local search iterations performed and average max-cut value for a given max\_iter\_grasp in an ofstream object statsFile. By calling the function printResult(graph, maxCutPartition), it prints the final partitions and max-cut in an output file.

**Output:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem | | | Constructive Algorithm | | | Local-1 | | GRASP-1 | |  |
| name | |V| or n | |E| or m | Randomized-1 | Greedy-1 | Semi-greedy-1 | No. of Iterations | Best Value | No. of Iterations | Best Value | Known Best Solution |
| G11 | 800 | 1600 | 460 | 464 | 492 | 2.47 | 464.4 | 30 | 492 | 627 |
| G13 | 800 | 1600 | 474 | 468 | 496 | 3 | 477.2 | 30 | 496 | 645 |
| G12 | 800 | 1600 | 452 | 450 | 476 | 3 | 450 | 30 | 476 | 621 |
| G15 | 800 | 4661 | 2922 | 2934 | 2958 | 3 | 2934 | 30 | 2958 | 3169 |
| G16 | 800 | 4672 | 2937 | 2949 | 2965 | 3 | 2949 | 50 | 2965 | 3172 |
| G14 | 800 | 4694 | 2955 | 2962 | 2971 | 3 | 2962 | 50 | 2971 | 3187 |
| G1 | 800 | 19176 | 11454 | 11397 | 11433 | 15 | 11397 | 80 | 11454 | 12078 |
| G43 | 1000 | 9990 | 6460 | 6409 |  | 8 | 6409 | 80 | 6460 | 7027 |

Figure Results from Different Test Graphs

The output results are computed from the statsFile which was modified from main() to include various information related to the simulation as given below. These results are obtained by changing the number of GRASP iterations from the command line and/or changing the heuristic in the function,

vector<int> grasp\_max\_cut(const vector<vector<int>>& graph, int max\_iter\_grasp)

For example, in case of G16, the statsFile contains the following info:

----------g16------------

randomized\_construction:

average no. of iterations of local search : 4.18, average cut-value of max\_cut\_local\_search() : 2903.38

no. of iterations of GRASP : 50, best cut-value of grasp\_max\_cut() : 2931

randomized\_construction:

average no. of iterations of local search : 4.1, average cut-value of max\_cut\_local\_search() : 2899.67

no. of iterations of GRASP : 60, best cut-value of grasp\_max\_cut() : 2933

randomized\_construction:

average no. of iterations of local search : 4.24286, average cut-value of max\_cut\_local\_search() : 2901.86

no. of iterations of GRASP : 70, best cut-value of grasp\_max\_cut() : 2925

randomized\_construction:

average no. of iterations of local search : 4.025, average cut-value of max\_cut\_local\_search() : 2902.47

no. of iterations of GRASP : 80, best cut-value of grasp\_max\_cut() : 2937

randomized\_construction:

average no. of iterations of local search : 4.29, average cut-value of max\_cut\_local\_search() : 2900.8

no. of iterations of GRASP : 100, best cut-value of grasp\_max\_cut() : 2928

greedy\_construction:

average no. of iterations of local search : 3, average cut-value of max\_cut\_local\_search() : 2949

no. of iterations of GRASP : 20, best cut-value of grasp\_max\_cut() : 2949

semi\_greedy\_construction:

average no. of iterations of local search : 3.32, average cut-value of max\_cut\_local\_search() : 2936.5

no. of iterations of GRASP : 50, best cut-value of grasp\_max\_cut() : 2965

Thus, in the row labeled G16 in the table, the columns are filled in the following ways:

* Randomized-1: max(best cut-value of grasp\_max\_cut() for the statistics under the heading randomized\_construction) = 2937
* Greedy-1: max(best cut-value of grasp\_max\_cut() for the statistics under the heading greedy\_construction) = 2949
* Semi-greedy-1: max(best cut-value of grasp\_max\_cut() for the statistics under the heading semi\_greedy\_construction) = 2965
* Local-1:
  + - Best Value: max(average cut-value of max\_cut\_local\_search()) = 2949
    - No. of Iterations: average no. of iterations of local corresponding to the Best Value = 3
* GRASP-1:
  + - Best value: max(best cut-value of grasp\_max\_cut() among all headings) = 2965
    - No. of Iterations: no. of iterations of GRASP corresponding to the Best Value = 2965