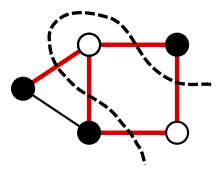
**Name:** Mohammad Sadat Hossain **Student ID:** 1905001

**Max Cut Problem:**

The Maximum Cut (Max-Cut) problem is a well-known combinatorial optimization problem that belongs to the class of NP-hard problems, specifically categorized as an NP-complete problem. It's a graph-based problem that involves partitioning the vertices of an undirected graph into two disjoint sets, such that the sum of the weights of the edges crossing the partition is maximized. In other words, the goal is to find a way to split the vertices into two groups in such a way that the total weight of the edges between these groups is as large as possible.



**Figure:** An example of a maximum cut

Formally, the Max-Cut problem can be defined as follows:

Given an undirected graph where is the set of vertices and is the set of edges, each edge has a weight, the task is to find a partition of into two disjoint subsets and such that the sum of the weights of the edges crossing the partition is maximized. Basically, the following has to be maximized:

The Max-Cut problem has practical applications in fields such as computer science, operations research, physics, and network analysis. Due to its NP-complete nature, finding an optimal solution for Max-Cut is computationally challenging and often infeasible for large instances.

**GRASP:**

The Greedy Randomized Adaptive Search Procedure (GRASP) is an algorithmic approach used to solve combinatorial optimization problems. It combines elements of greedy construction with randomization and iterative improvement.

In summary, the GRASP approach involves the following key characteristics:

**1.** **Greedy Construction Phase:** During each iteration of GRASP, a greedy construction algorithm is applied to build a candidate solution. The construction algorithm makes locally optimal choices based on some heuristic criteria. This produces a feasible solution that might not be of the highest quality.

**2.** **Randomized Component:** A certain level of randomness is introduced. The algorithm randomly perturbs or diversifies the constructed solution to explore different areas of the solution space. This randomization helps the algorithm avoid getting stuck in suboptimal solutions.

**3.** **Local Search and Iterative Improvement:** Following the greedy and randomization step, a local search or iterative improvement procedure is executed. The local search explores the neighborhood of the current solution, making small adjustments to obtain the local optima.

The GRASP approach can potentially play a very important role in solving this problem of our consideration, the max cut problem. At each iteration of the GRASP algorithm, at first a decent solution is prepared by the usage of any constructive algorithm. Since GRASP demands inclusion of some randomness to the solution, the semi greedy and randomized approaches were useful in this regard. The solution benefited from random restarts introduced by this randomness, and thus consistently worked on different search regions across different iterations. In the next step, a local search technique was used to try to reach the local optima, which is obviously an improved version of the constructed solution.

**Summary of Work:**

The assignment specification asked for some approaches to attain an approximate solution to the problem. In this regard, five constructive approaches (one randomized, two greedy and two semi-greedy) were adopted. In each of these cases, to improve the solution, a local search technique was used. The results of all the constructive algorithms, the subsequent applications of local search technique on them and the GRASP approaches have been presented in a spreadsheet for more vivid comparison. As benchmark test set, 54 graphs of different nature (with different size, node or edge count, edge weight variations) were used. To run the implemented solutions on 54 graphs simultaneously, a shell script was written. A python script was also introduced to automate filling up the spreadsheet with test results.

**Constructive Algorithms:**

Two greedy approaches were used for this problem, both of which also saw their semi-greedy versions used. Basically, instead of choosing the best vertex (or edge depending on the approach), a certain number of best solutions were taken to construct an RCL (Restricted Candidate List). Then, a candidate was randomly selected from this list and used to extend the solution. To build the RCL, we used a value-based approach. The threshold was set to be (max – min) \* ɑ + min, where ɑ is a randomly generated number. Candidates having the designated value above or equal to this threshold were added to the RCL. Needless to say, ɑ = 1 can mimic the greedy version of the semi-greedy approaches.

**1. Greedy-1:** This is basically implementation of the method mentioned in the problem specification and respective reference book. We iterate over the vertices one by one. For each vertex, we calculate the cumulative contribution it can make to the cut as a result of being added to set X or Y. For a certain node, the greedy function value is the maximum of the possible contributions to either of the sets.

Here,

**2. Semi Greedy-1:** This is same as the greedy-1 approach except for the fact that ɑ can be any random value here, not just 1 like greedy-1. Because of this random value, randomness was introduced which enabled us to perform the search across different search regions. If was 1, then every time we would construct the same initial solution, and thus potentially we could get stuck in some local optima. We will see later how this randomness essentially improved the solution from the purely greedy version.

**3. Greedy-2:** In this method, we iterated over the edges in non-increasing order of weights. For each edge , we considered its two end points and . There were some cases to consider here:

* If both the vertices are already added to any of the sets, do nothing.
* Otherwise, at least one vertex is yet to be added. For ease of decision making, we compute for the vertices and . The following cases arise then:
* Both vertices are yet to be added. We then consider the four cases of adding both of them to or ; one to , the other to and vice versa. The addition that contributes the highest is implemented and both and get added to the sets.
* is added already. is left for consideration. We again consider whether it is better to add this vertex to the set of or the opposite one. It is important to note here that adding to the opposite set allows us to include the current edge in the cut. However, the comparison of the values gives us a deterministic decision regarding where to add the vertex
* has been added, left. This case is handled similarly like the previous one.

Interestingly, this approach performed almost as good as greedy-1 approach and, in some cases, even outperformed that. We will later see the comparison.

**4. Semi Greedy-2:** Again, this was same as greedy-2 approach with some random value as ɑ. So, instead of considering the endpoints of the maximum weighted edge, we considered edges with weight above a threshold value and chose one of them randomly. The cases were then similarly considered as stated earlier.

**5. Randomized-1:** This was a purely randomized approach. For each vertex, we just randomly added it to any of the sets with equal probability, without considering anything else at all.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Problem** | | | **Constructive Algorithm** | | | | |
| Test Set Name | |V| or n | |E| or m | **Simple Randomized  or  Randomized-1** | **Greedy-1** | **Semi  Greedy 1** | **Greedy-2** | **Semi Greedy 2** |
|
| G1 | 800 | 19176 | 9588 | 11225 | 11176 | 10914 | 10949 |
| G2 | 800 | 19176 | 9585 | 11250 | 11179 | 10919 | 10962 |
| G3 | 800 | 19176 | 9585 | 11222 | 11184 | 10907 | 10952 |
| G4 | 800 | 19176 | 9597 | 11287 | 11191 | 10922 | 10965 |
| G5 | 800 | 19176 | 9588 | 11248 | 11194 | 10969 | 10965 |
| G6 | 800 | 19176 | 85 | 1796 | 1646 | 1502 | 1495 |
| G7 | 800 | 19176 | -76 | 1552 | 1489 | 1356 | 1325 |
| G8 | 800 | 19176 | -90 | 1635 | 1505 | 1330 | 1334 |
| G9 | 800 | 19176 | -27 | 1629 | 1543 | 1357 | 1366 |
| G10 | 800 | 19176 | -89 | 1665 | 1482 | 1399 | 1321 |
| G11 | 800 | 1600 | 19 | 485 | 424 | 457 | 461 |
| G12 | 800 | 1600 | 0 | 479 | 414 | 443 | 456 |
| G13 | 800 | 1600 | 21 | 505 | 433 | 463 | 471 |
| G14 | 800 | 4694 | 2352 | 2927 | 2936 | 2799 | 2785 |
| G15 | 800 | 4661 | 2331 | 2890 | 2915 | 2754 | 2765 |
| G16 | 800 | 4672 | 2328 | 2886 | 2918 | 2751 | 2769 |
| G17 | 800 | 4667 | 2334 | 2904 | 2917 | 2753 | 2767 |
| G18 | 800 | 4694 | 29 | 839 | 762 | 631 | 665 |
| G19 | 800 | 4661 | -55 | 756 | 663 | 590 | 584 |
| G20 | 800 | 4672 | -24 | 777 | 703 | 579 | 606 |
| G21 | 800 | 4667 | -31 | 793 | 689 | 646 | 619 |
| G22 | 2000 | 19990 | 9999 | 12747 | 12687 | 12183 | 12143 |
| G23 | 2000 | 19990 | 9993 | 12785 | 12685 | 12179 | 12158 |
| G24 | 2000 | 19990 | 10007 | 12811 | 12681 | 12157 | 12143 |
| G25 | 2000 | 19990 | 9994 | 12831 | 12688 | 12119 | 12158 |
| G26 | 2000 | 19990 | 9993 | 12818 | 12685 | 12114 | 12148 |
| G27 | 2000 | 19990 | -12 | 2707 | 2513 | 2121 | 2150 |
| G28 | 2000 | 19990 | -50 | 2635 | 2475 | 2116 | 2108 |
| G29 | 2000 | 19990 | 37 | 2724 | 2559 | 2286 | 2208 |
| G30 | 2000 | 19990 | 64 | 2718 | 2580 | 2182 | 2221 |
| G31 | 2000 | 19990 | -47 | 2668 | 2475 | 2168 | 2115 |
| G32 | 2000 | 4000 | 9 | 1225 | 1052 | 1128 | 1141 |
| G33 | 2000 | 4000 | -15 | 1207 | 1028 | 1144 | 1132 |
| G34 | 2000 | 4000 | -24 | 1211 | 1020 | 1097 | 1127 |
| G35 | 2000 | 11778 | 5887 | 7334 | 7365 | 7003 | 6977 |
| G36 | 2000 | 11766 | 5888 | 7273 | 7353 | 6927 | 6975 |
| G37 | 2000 | 11785 | 5894 | 7315 | 7363 | 6986 | 6987 |
| G38 | 2000 | 11779 | 5882 | 7307 | 7362 | 6987 | 6979 |
| G39 | 2000 | 11778 | 22 | 2034 | 1779 | 1570 | 1597 |
| G40 | 2000 | 11766 | -38 | 2042 | 1803 | 1536 | 1551 |
| G41 | 2000 | 11785 | -9 | 2052 | 1811 | 1547 | 1570 |
| G42 | 2000 | 11779 | 62 | 2117 | 1871 | 1647 | 1661 |
| G43 | 1000 | 9990 | 4990 | 6384 | 6323 | 6061 | 6066 |
| G44 | 1000 | 9990 | 4981 | 6371 | 6320 | 6088 | 6058 |
| G45 | 1000 | 9990 | 5003 | 6423 | 6315 | 6108 | 6070 |
| G46 | 1000 | 9990 | 5004 | 6410 | 6324 | 6041 | 6071 |
| G47 | 1000 | 9990 | 4991 | 6330 | 6324 | 6126 | 6078 |
| G48 | 3000 | 6000 | 2998 | 5999 | 5706 | 4927 | 4896 |
| G49 | 3000 | 6000 | 3002 | 5999 | 5714 | 4899 | 4910 |
| G50 | 3000 | 6000 | 3002 | 5879 | 5679 | 4975 | 4911 |
| G51 | 1000 | 5909 | 2948 | 3659 | 3689 | 3476 | 3503 |
| G52 | 1000 | 5916 | 2953 | 3664 | 3688 | 3540 | 3503 |
| G53 | 1000 | 5914 | 2953 | 3634 | 3689 | 3523 | 3496 |
| G54 | 1000 | 5916 | 2952 | 3664 | 3684 | 3506 | 3499 |

**Table 1:** Constructive Algorithm Cut Values (Average)

The cut values here are the averages found over all the iterations. Quite expectedly, both the greedy (or semi-greedy) approaches outperformed the randomized approach by some huge margin. In almost all the cases, the first greedy approach was the best performing one. This is not unexpected since this approach goes through some very rigorous calculations before adding a vertex to the sets. But it is also important to note, no matter how bad a constructive algorithm performs, local search technique may well transform this to a very decent solution, as we will see later with the application of local search technique on randomized algorithm.

**Local Search Technique:**

We used a simple local search technique. The technique works like this: for each vertex, we try moving this vertex from its current one to the opposite one. If it results in a better cut, we keep the change. As soon as we find a better solution by moving a vertex, we resort to that. We keep iterating like this and stop when no improvement is found any more. This technique essentially finds us the local optima. As said earlier, even this simple local search technique can improve the construction cut to a great extent. Another point to note here is that the better the initial construction cut value, the less the subsequent local search iteration counts. Because it is more probable that a better algorithm will be closer to a local optimum than a worse one.

In the following tables, the no. of iterations and best value are both averages of respective values found over all iterations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Set Name** | **Simple Randomized  or  Randomized-1** | **Simple Local Search for Randomized-1** | |
| No. of  Iterations | Best Value |
| G1 | 9588 | 630 | 11369 |
| G2 | 9585 | 629 | 11379 |
| G3 | 9585 | 631 | 11377 |
| G4 | 9597 | 629 | 11386 |
| G5 | 9588 | 621 | 11362 |
| G6 | 85 | 636 | 1927 |
| G7 | -76 | 629 | 1759 |
| G8 | -90 | 639 | 1753 |
| G9 | -27 | 626 | 1796 |
| G10 | -89 | 623 | 1745 |
| G11 | 19 | 182 | 426 |
| G12 | 0 | 185 | 417 |
| G13 | 21 | 186 | 441 |
| G14 | 2352 | 261 | 2920 |
| G15 | 2331 | 269 | 2904 |
| G16 | 2328 | 271 | 2906 |
| G17 | 2334 | 271 | 2904 |
| G18 | 29 | 327 | 834 |
| G19 | -55 | 329 | 743 |
| G20 | -24 | 326 | 772 |
| G21 | -31 | 331 | 774 |
| G22 | 9999 | 1139 | 12834 |
| G23 | 9993 | 1127 | 12823 |
| G24 | 10007 | 1136 | 12818 |
| G25 | 9994 | 1136 | 12824 |
| G26 | 9993 | 1138 | 12812 |
| G27 | -12 | 1149 | 2826 |
| G28 | -50 | 1126 | 2769 |
| G29 | 37 | 1141 | 2878 |
| G30 | 64 | 1128 | 2881 |
| G31 | -47 | 1143 | 2796 |
| G32 | 9 | 466 | 1059 |
| G33 | -15 | 465 | 1036 |
| G34 | -24 | 464 | 1027 |
| G35 | 5887 | 670 | 7327 |
| G36 | 5888 | 664 | 7320 |
| G37 | 5894 | 670 | 7327 |
| G38 | 5882 | 660 | 7323 |
| G39 | 22 | 811 | 2013 |
| G40 | -38 | 828 | 1990 |
| G41 | -9 | 818 | 1990 |
| G42 | 62 | 816 | 2074 |
| G43 | 4990 | 581 | 6415 |
| G44 | 4981 | 569 | 6397 |
| G45 | 5003 | 562 | 6400 |
| G46 | 5004 | 555 | 6400 |
| G47 | 4991 | 574 | 6402 |
| G48 | 2998 | 834 | 5002 |
| G49 | 3002 | 833 | 5011 |
| G50 | 3002 | 833 | 5012 |
| G51 | 2948 | 338 | 3673 |
| G52 | 2953 | 332 | 3672 |
| G53 | 2953 | 328 | 3668 |
| G54 | 2952 | 333 | 3670 |

**Table 2:** Improvement from Construction to Local Search for Randomized-1

Since the initial solution for Randomized-1 was pretty bad, the local search technique had much to work on, and thus the iteration count is quite high. We can also see the local search technique hugely improved the results in maximum cases.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Set Name** | **Greedy-1** | **Simple Local Search for Greedy-1** | |
| No. of  Iterations | Best Value |
| G1 | 11225 | 92 | 11400 |
| G2 | 11250 | 69 | 11386 |
| G3 | 11222 | 105 | 11417 |
| G4 | 11287 | 79 | 11435 |
| G5 | 11248 | 58 | 11365 |
| G6 | 1796 | 71 | 1941 |
| G7 | 1552 | 106 | 1756 |
| G8 | 1635 | 109 | 1818 |
| G9 | 1629 | 65 | 1768 |
| G10 | 1665 | 47 | 1766 |
| G11 | 485 | 1 | 487 |
| G12 | 479 | 4 | 487 |
| G13 | 505 | 2 | 509 |
| G14 | 2927 | 33 | 2972 |
| G15 | 2890 | 46 | 2946 |
| G16 | 2886 | 43 | 2941 |
| G17 | 2904 | 38 | 2957 |
| G18 | 839 | 33 | 884 |
| G19 | 756 | 33 | 801 |
| G20 | 777 | 33 | 827 |
| G21 | 793 | 23 | 826 |
| G22 | 12747 | 105 | 12943 |
| G23 | 12785 | 81 | 12937 |
| G24 | 12811 | 77 | 12944 |
| G25 | 12831 | 70 | 12950 |
| G26 | 12818 | 81 | 12958 |
| G27 | 2707 | 129 | 2923 |
| G28 | 2635 | 119 | 2847 |
| G29 | 2724 | 119 | 2945 |
| G30 | 2718 | 126 | 2937 |
| G31 | 2668 | 158 | 2931 |
| G32 | 1225 | 13 | 1251 |
| G33 | 1207 | 8 | 1223 |
| G34 | 1211 | 11 | 1233 |
| G35 | 7334 | 77 | 7441 |
| G36 | 7273 | 97 | 7401 |
| G37 | 7315 | 88 | 7443 |
| G38 | 7307 | 89 | 7426 |
| G39 | 2034 | 75 | 2134 |
| G40 | 2042 | 53 | 2123 |
| G41 | 2052 | 47 | 2120 |
| G42 | 2117 | 52 | 2186 |
| G43 | 6384 | 33 | 6437 |
| G44 | 6371 | 55 | 6450 |
| G45 | 6423 | 32 | 6479 |
| G46 | 6410 | 42 | 6490 |
| G47 | 6330 | 52 | 6435 |
| G48 | 5999 | 0 | 5999 |
| G49 | 5999 | 0 | 5999 |
| G50 | 5879 | 0 | 5879 |
| G51 | 3659 | 58 | 3733 |
| G52 | 3664 | 61 | 3755 |
| G53 | 3634 | 55 | 3722 |
| G54 | 3664 | 46 | 3729 |

**Table 3:** Improvement from Construction to Local Search for Greedy-1

The construction values were good enough, resulting in low iteration count for local search. In some cases, the construction even found the local optimum, resulting in zero average iteration count for local search.

Similarly, the results for local search on rest of the constructive algorithms follow.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Set Name** | **Semi**  **Greedy-1** | **Simple Local Search for Semi Greedy-1** | |
| No. of  Iterations | Best Value |
| G1 | 11176 | 104 | 11378 |
| G2 | 11179 | 104 | 11384 |
| G3 | 11184 | 100 | 11382 |
| G4 | 11191 | 102 | 11400 |
| G5 | 11194 | 97 | 11386 |
| G6 | 1646 | 135 | 1922 |
| G7 | 1489 | 122 | 1743 |
| G8 | 1505 | 121 | 1754 |
| G9 | 1543 | 124 | 1796 |
| G10 | 1482 | 130 | 1747 |
| G11 | 424 | 13 | 450 |
| G12 | 414 | 13 | 440 |
| G13 | 433 | 14 | 462 |
| G14 | 2936 | 24 | 2968 |
| G15 | 2915 | 25 | 2946 |
| G16 | 2918 | 25 | 2950 |
| G17 | 2917 | 26 | 2949 |
| G18 | 762 | 56 | 852 |
| G19 | 663 | 59 | 760 |
| G20 | 703 | 56 | 794 |
| G21 | 689 | 61 | 787 |
| G22 | 12687 | 132 | 12912 |
| G23 | 12685 | 131 | 12908 |
| G24 | 12681 | 133 | 12907 |
| G25 | 12688 | 127 | 12906 |
| G26 | 12685 | 131 | 12910 |
| G27 | 2513 | 192 | 2847 |
| G28 | 2475 | 188 | 2809 |
| G29 | 2559 | 196 | 2909 |
| G30 | 2580 | 195 | 2925 |
| G31 | 2475 | 195 | 2822 |
| G32 | 1052 | 33 | 1118 |
| G33 | 1028 | 32 | 1092 |
| G34 | 1020 | 35 | 1091 |
| G35 | 7365 | 60 | 7442 |
| G36 | 7353 | 63 | 7433 |
| G37 | 7363 | 62 | 7442 |
| G38 | 7362 | 61 | 7439 |
| G39 | 1779 | 156 | 2039 |
| G40 | 1803 | 141 | 2038 |
| G41 | 1811 | 143 | 2044 |
| G42 | 1871 | 149 | 2116 |
| G43 | 6323 | 70 | 6439 |
| G44 | 6320 | 65 | 6432 |
| G45 | 6315 | 69 | 6433 |
| G46 | 6324 | 64 | 6437 |
| G47 | 6324 | 69 | 6441 |
| G48 | 5706 | 3 | 5711 |
| G49 | 5714 | 3 | 5720 |
| G50 | 5679 | 3 | 5685 |
| G51 | 3689 | 32 | 3729 |
| G52 | 3688 | 33 | 3730 |
| G53 | 3689 | 31 | 3729 |
| G54 | 3684 | 32 | 3726 |

**Table 4:** Improvement from Construction to Local Search for Semi Greedy-1

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Set Name** | **Greedy-2** | **Simple Local Search for Greedy-2** | |
| No. of  Iterations | Best Value |
| G1 | 10914 | 190 | 11364 |
| G2 | 10919 | 243 | 11434 |
| G3 | 10907 | 205 | 11300 |
| G4 | 10922 | 184 | 11343 |
| G5 | 10969 | 184 | 11369 |
| G6 | 1502 | 186 | 1899 |
| G7 | 1356 | 156 | 1715 |
| G8 | 1330 | 171 | 1720 |
| G9 | 1357 | 232 | 1845 |
| G10 | 1399 | 172 | 1775 |
| G11 | 457 | 62 | 497 |
| G12 | 443 | 68 | 498 |
| G13 | 463 | 58 | 503 |
| G14 | 2799 | 93 | 2939 |
| G15 | 2754 | 111 | 2921 |
| G16 | 2751 | 110 | 2909 |
| G17 | 2753 | 90 | 2891 |
| G18 | 631 | 126 | 834 |
| G19 | 590 | 109 | 785 |
| G20 | 579 | 117 | 801 |
| G21 | 646 | 85 | 776 |
| G22 | 12183 | 338 | 12817 |
| G23 | 12179 | 361 | 12839 |
| G24 | 12157 | 353 | 12850 |
| G25 | 12119 | 393 | 12836 |
| G26 | 12114 | 340 | 12788 |
| G27 | 2121 | 335 | 2793 |
| G28 | 2116 | 340 | 2773 |
| G29 | 2286 | 304 | 2855 |
| G30 | 2182 | 341 | 2826 |
| G31 | 2168 | 364 | 2860 |
| G32 | 1128 | 171 | 1234 |
| G33 | 1144 | 155 | 1251 |
| G34 | 1097 | 159 | 1220 |
| G35 | 7003 | 238 | 7347 |
| G36 | 6927 | 273 | 7328 |
| G37 | 6986 | 222 | 7327 |
| G38 | 6987 | 237 | 7362 |
| G39 | 1570 | 265 | 2021 |
| G40 | 1536 | 278 | 2039 |
| G41 | 1547 | 288 | 2026 |
| G42 | 1647 | 271 | 2117 |
| G43 | 6061 | 214 | 6442 |
| G44 | 6088 | 151 | 6386 |
| G45 | 6108 | 139 | 6388 |
| G46 | 6041 | 167 | 6377 |
| G47 | 6126 | 193 | 6442 |
| G48 | 4927 | 88 | 5107 |
| G49 | 4899 | 108 | 5121 |
| G50 | 4975 | 110 | 5201 |
| G51 | 3476 | 127 | 3671 |
| G52 | 3540 | 115 | 3713 |
| G53 | 3523 | 111 | 3709 |
| G54 | 3506 | 117 | 3688 |

**Table 5:** Improvement from Construction to Local Search for Greedy-2

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Set Name** | **Semi**  **Greedy-2** | **Simple Local Search for Semi Greedy-2** | |
| No. of  Iterations | Best Value |
| G1 | 10949 | 197 | 11361 |
| G2 | 10962 | 194 | 11373 |
| G3 | 10952 | 185 | 11347 |
| G4 | 10965 | 194 | 11369 |
| G5 | 10965 | 192 | 11374 |
| G6 | 1495 | 196 | 1908 |
| G7 | 1325 | 194 | 1745 |
| G8 | 1334 | 187 | 1745 |
| G9 | 1366 | 194 | 1780 |
| G10 | 1321 | 191 | 1736 |
| G11 | 461 | 59 | 504 |
| G12 | 456 | 62 | 497 |
| G13 | 471 | 59 | 517 |
| G14 | 2785 | 96 | 2931 |
| G15 | 2765 | 100 | 2914 |
| G16 | 2769 | 100 | 2920 |
| G17 | 2767 | 102 | 2917 |
| G18 | 665 | 109 | 849 |
| G19 | 584 | 108 | 765 |
| G20 | 606 | 107 | 790 |
| G21 | 619 | 111 | 795 |
| G22 | 12143 | 356 | 12820 |
| G23 | 12158 | 354 | 12824 |
| G24 | 12143 | 356 | 12823 |
| G25 | 12158 | 357 | 12836 |
| G26 | 12148 | 357 | 12819 |
| G27 | 2150 | 360 | 2827 |
| G28 | 2108 | 359 | 2787 |
| G29 | 2208 | 356 | 2883 |
| G30 | 2221 | 351 | 2890 |
| G31 | 2115 | 367 | 2807 |
| G32 | 1141 | 151 | 1249 |
| G33 | 1132 | 153 | 1237 |
| G34 | 1127 | 142 | 1231 |
| G35 | 6977 | 248 | 7356 |
| G36 | 6975 | 249 | 7353 |
| G37 | 6987 | 246 | 7361 |
| G38 | 6979 | 247 | 7357 |
| G39 | 1597 | 270 | 2060 |
| G40 | 1551 | 272 | 2024 |
| G41 | 1570 | 270 | 2032 |
| G42 | 1661 | 269 | 2122 |
| G43 | 6066 | 178 | 6402 |
| G44 | 6058 | 184 | 6406 |
| G45 | 6070 | 174 | 6399 |
| G46 | 6071 | 178 | 6410 |
| G47 | 6078 | 180 | 6414 |
| G48 | 4896 | 114 | 5130 |
| G49 | 4910 | 111 | 5139 |
| G50 | 4911 | 111 | 5141 |
| G51 | 3503 | 122 | 3686 |
| G52 | 3503 | 124 | 3692 |
| G53 | 3496 | 124 | 3684 |
| G54 | 3499 | 123 | 3686 |

**Table 6:** Improvement from Construction to Local Search for Semi Greedy-2

**GRASP Results:**

Finally, we can analyze the GRASP results. We had two algorithms GRASP-1, GRASP-2, using Semi Greedy-1 and Semi Greedy-2 algorithms respectively. A comparative analysis of them against the available best-known values are given below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test**  **Set**  **Name** | **GRASP** | | | | **Known  Best Solution or Upper Bound** |
| **GRASP-1  (using Semi Greedy 1)** | | **GRASP-2 (using Semi Greedy 2)** | |
| No. of  Iterations | Best Value | No. of  Iterations | Best Value |
| G1 | 50 | 11437 | 50 | 11432 | 12078 |
| G2 | 50 | 11460 | 50 | 11471 | 12084 |
| G3 | 50 | 11496 | 50 | 11410 | 12077 |
| G4 | 50 | 11485 | 50 | 11442 |  |
| G5 | 50 | 11470 | 50 | 11474 |  |
| G6 | 50 | 2007 | 50 | 2011 |  |
| G7 | 50 | 1828 | 50 | 1849 |  |
| G8 | 50 | 1863 | 50 | 1856 |  |
| G9 | 50 | 1906 | 50 | 1867 |  |
| G10 | 50 | 1850 | 50 | 1871 |  |
| G11 | 50 | 472 | 50 | 527 | 627 |
| G12 | 50 | 472 | 50 | 513 | 621 |
| G13 | 50 | 486 | 50 | 547 | 645 |
| G14 | 50 | 2985 | 50 | 2956 | 3187 |
| G15 | 50 | 2980 | 50 | 2937 | 3169 |
| G16 | 50 | 2969 | 50 | 2947 | 3172 |
| G17 | 50 | 2976 | 50 | 2942 |  |
| G18 | 50 | 905 | 50 | 893 |  |
| G19 | 50 | 818 | 50 | 819 |  |
| G20 | 50 | 843 | 50 | 832 |  |
| G21 | 50 | 824 | 50 | 823 |  |
| G22 | 50 | 12992 | 50 | 12912 | 14123 |
| G23 | 50 | 13028 | 50 | 12953 | 14129 |
| G24 | 50 | 13013 | 50 | 12913 | 14131 |
| G25 | 50 | 13025 | 50 | 12944 |  |
| G26 | 50 | 12993 | 50 | 12904 |  |
| G27 | 50 | 2914 | 50 | 2923 |  |
| G28 | 50 | 2927 | 50 | 2871 |  |
| G29 | 50 | 2993 | 50 | 2955 |  |
| G30 | 50 | 3017 | 50 | 2994 |  |
| G31 | 50 | 2962 | 50 | 2896 |  |
| G32 | 50 | 1172 | 50 | 1283 | 1560 |
| G33 | 50 | 1144 | 50 | 1272 | 1537 |
| G34 | 50 | 1136 | 50 | 1265 | 1541 |
| G35 | 50 | 7474 | 50 | 7402 | 8000 |
| G36 | 50 | 7477 | 50 | 7386 | 7996 |
| G37 | 50 | 7472 | 50 | 7408 | 8009 |
| G38 | 50 | 7480 | 50 | 7395 |  |
| G39 | 50 | 2107 | 50 | 2134 |  |
| G40 | 50 | 2105 | 50 | 2089 |  |
| G41 | 50 | 2092 | 50 | 2093 |  |
| G42 | 50 | 2212 | 50 | 2185 |  |
| G43 | 50 | 6500 | 50 | 6454 | 7027 |
| G44 | 50 | 6474 | 50 | 6493 | 7022 |
| G45 | 50 | 6476 | 50 | 6466 | 7020 |
| G46 | 50 | 6490 | 50 | 6511 |  |
| G47 | 50 | 6493 | 50 | 6471 |  |
| G48 | 50 | 6000 | 50 | 5234 | 6000 |
| G49 | 50 | 5910 | 50 | 5214 | 6000 |
| G50 | 50 | 5810 | 50 | 5236 | 5988 |
| G51 | 50 | 3751 | 50 | 3729 |  |
| G52 | 50 | 3752 | 50 | 3729 |  |
| G53 | 50 | 3752 | 50 | 3711 |  |
| G54 | 50 | 3749 | 50 | 3711 |  |

**Table 7:** Comparison between GRASP-1 and GRASP-2

Out of the 54 test sets, GRASP-1 performed better in 37 of them whereas GRASP-2 did better in the rest 17. Some patterns can also be observed in test sets where GRASP-2 performed better. If we look at the edge weights of G6, G7, G10, G11, G12, G13, G32, G33, G34 where GRASP-2 was better, the edge weights were either 1 or -1. The fact that GRASP-2 dealt with edges of weight 1 first (since they are of higher weight) played a role here. In both of the algorithms, at each step we somehow consider the possible effect of adding a certain vertex to the sets built already. Hence, the edges added earlier play a larger role in all the later computations. Since GRASP-2 considers edges of higher weight first, these higher weight-edges are considered in all the subsequent iterations and possibly generate a higher cost cut.

However, for the maximum of the rest of the test sets, all edge weights were 1 (i.e., equal weight). GRASP-1 performed better in most of those cases. Since all edges are of same weight, GRASP-2’s edge ordering does not seem to play much role there. Probably, in a more generalized set with unequal edge weights, GRASP-2 would perform even better.

It is also worth mentioning that because of the intensive computation needed for running all 54 test cases (and 5 instances of each of them running 5 constructive algorithms) simultaneously, GRASP iteration limit was set to 50. If the iteration count was significantly higher, we would probably find results closer to the best-known ones.