Bangladesh University of Engineering & Technology

Course No: CSE 318

Course Title: Artificial Intelligence

Offline-3:Solving Max-Cut Problem by GRASP

Submitted By:

Name: Fatema Tuj Johora

Department: CSE

Section: A1

Roll: 1905022

Date of submission:08.18.2023

Title: Solving Max-Cut problem using Local Search

Abstract: Max-cut problem is a NP Hard problem; so no known polynomial-algorithm can give the optimized result. It involves partitioning the vertices of an undirected graph into two disjoint sets such that the number of edges between two sets(known as the “cuts”), is maximized.

In our assignment, we solved the Max-cut problem using different algorithms; full randomized approach, semi-greedy ,full greedy and GRASP(Greedy Randomized Adaptive Search Procedure) and compared them.

1. Introduction:

As no polynomial-algorithm can give the optimized result for this problem, we implemented different algorithms and compared them to check which one gives the better result.

In this assignment we implemented four different types of algorithms including GRASP which use local search to improve its result.

2. Algorithm:

In our algorithm we maintain an RCL(Restricted Candidate List) from which we select an edge and add its vertices to two partitions. Our optimized result varies with this RCL. The less the size of RCL, the more optimized the result will be. And the RCL is selected by a threshold value, which is determined by alpha. So, alpha controls the randomness.

- **Simple - Randomized:**

  This algorithm randomly partitioned vertices into two disjoint sets and then evaluated the cut value. This is basically a semi-greedy approach with alpha =0. So, it will consider all the vertices into RCL, which will give the least optimized result.

- **Semi-Greedy :**

  In this algo, a vertex is chosen randomly and in a greedy way more vertices are added to the partition to maximize the cut value. It keeps alternating vertices between two sets until no further improvement is possible. In this algo, to maintain the randomness , we use an “alpha”, by varying this randomness is controlled and so is RCL. Randomness decreases with the value of alpha. The more value of alpha is, the less size will be of RCL and the more optimized the result will be.

- **Full-Greedy:**

  It is a semi-greedy algo with alpha=1. If alpha becomes one, then the most optimized result will be found; which is a greedy approach. So, it always keeps adding the max-weighted edge to the partition.

- **GRASP:**

  In a greedy approach, it starts from the same vertex.

But this algorithm is a multistart algorithm, so if we run this algorithm multiple times, it will start from multiple vertices and we will get more optimized result. So, in our algorithm , we set an iteration count, in each iteration it will apply semi-greedy and then local search on that result to make it more optimized.

- **Local Search:**

  In GRASP, we used local search in each iteration to get the local maxima value. In local search, we keep alternating vertices of max-weighted edges between two partitions until we get stuck at local maxima.

3. Analysis:

a. Part One:

Problem			Constructive Algorithm		Local Search		Grasp	
FileName	Vertex	Edge	Simple Randomized	Simple Greedy	Simple Local Search		Grasp-1	
					No of Iterations	Best Value	No of Iterations	Best Value
G1	800	19176	11036	11289	64	11461	50	11461
G2	800	19176	11036	11248	74	11449	50	11449
G3	800	19176	11074	11257	66	11473	50	11473
G4	800	19176	11128	11326	72	11453	50	11453
G5	800	19176	11009	11275	75	11476	50	11476

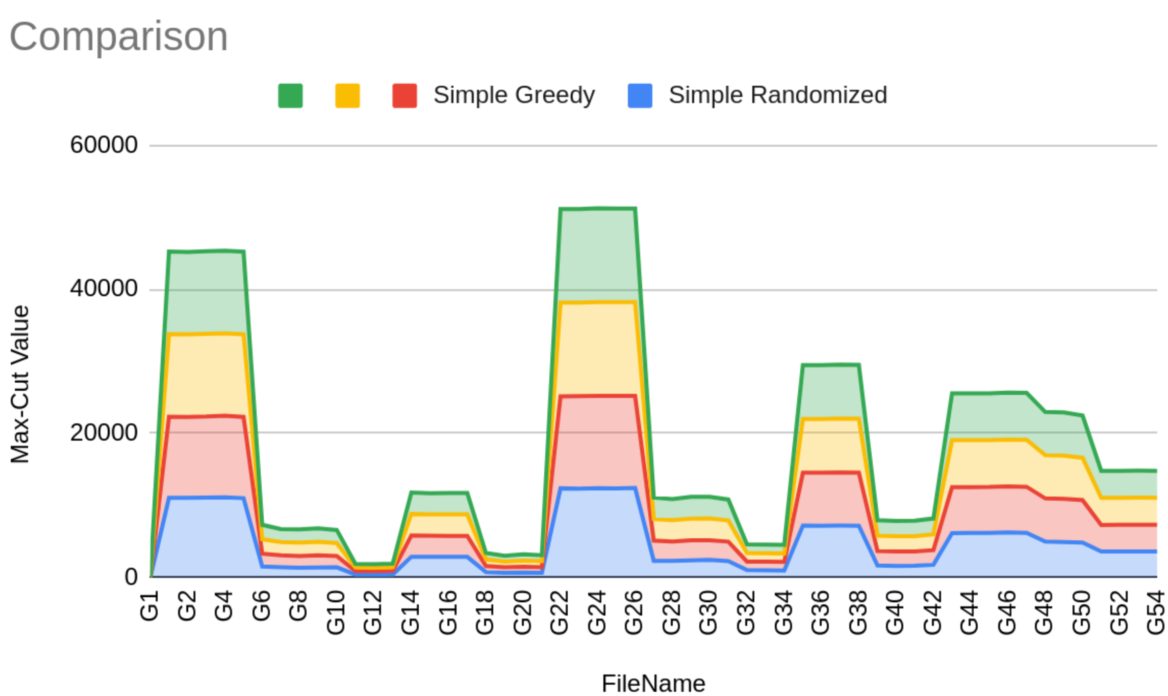
G6	800	19176	1520	1778	80	2003	50	2003
G7	800	19176	1417	1652	70	1829	50	1829
G8	800	19176	1340	1629	101	1858	50	1858
G9	800	19176	1396	1686	86	1861	50	1861
G10	800	19176	1401	1572	109	1806	50	1806
G11	800	1600	410	478	6	490	50	490
G12	800	1600	402	472	11	484	50	484
G13	800	1600	400	506	8	508	50	508
G14	800	4694	2876	2948	20	2984	50	2984
G15	800	4661	2854	2924	20	2964	50	2964
G16	800	4672	2856	2912	18	2978	50	2978
G17	800	4667	2851	2919	26	2973	50	2973
G18	800	4694	732	863	49	901	50	901
G19	800	4661	627	787	44	796	50	796
G20	800	4672	666	816	50	863	50	863
G21	800	4667	635	786	41	836	50	836
G22	2000	19990	12372	12781	97	13001	50	13001
G23	2000	19990	12314	12855	116	12990	50	12990
G24	2000	19990	12402	12804	94	13019	50	13019
G25	2000	19990	12359	12844	115	13014	50	13014
G26	2000	19990	12421	12790	112	13005	50	13005
G27	2000	19990	2295	2832	177	2983	50	2983
G28	2000	19990	2311	2689	174	2932	50	2932
G29	2000	19990	2358	2784	146	3022	50	3022

G30	2000	19990	2452	2715	153	3017	50	3017
G31	2000	19990	2280	2692	125	2922	50	2922
G32	2000	4000	1022	1182	17	1194	50	1194
G33	2000	4000	996	1180	21	1192	50	1192
G34	2000	4000	966	1182	22	1184	50	1184
G35	2000	11778	7193	7340	48	7479	50	7479
G36	2000	11766	7163	7374	53	7471	50	7471
G37	2000	11785	7193	7404	63	7473	50	7473
G38	2000	11779	7180	7383	37	7482	50	7482
G39	2000	11778	1667	1995	105	2140	50	2140
G40	2000	11766	1596	2016	112	2108	50	2108
G41	2000	11785	1620	1995	137	2123	50	2123
G42	2000	11779	1751	2047	157	2191	50	2191
G43	1000	9990	6150	6401	50	6508	50	6508
G44	1000	9990	6189	6352	63	6511	50	6511
G45	1000	9990	6172	6392	44	6499	50	6499
G46	1000	9990	6242	6403	43	6506	50	6506
G47	1000	9990	6194	6393	47	6515	50	6515
G48	3000	6000	4972	6000	0	6000	50	6000
G49	3000	6000	4910	6000	0	6000	50	6000
G50	3000	6000	4850	5880	0	5880	50	5880
G51	1000	5909	3597	3694	35	3753	50	3753
G52	1000	5916	3604	3699	35	3751	50	3751
G53	1000	5914	3602	3723	22	3751	50	3751

G54	**1000**	**5916**	**3596**	**3698**	**32**	**3753**	**50**	**3753**

In this table, we compare between fully randomized ,fully greedy and GRASP results. We also compared the required iteration number to reach the optimized result and total iteration number. In this assignment, I used 50 as an iteration number. If iteration number is increased, then more optimized result will be found. Now, from this table, it can be said that

- ❖ Greedy algorithm gives better results than randomized ones.
- ❖ GRASP (which uses Local search) gives better than full greedy algorithm.
- ❖ Iteration count for local search may be less, equal or more than total iteration count.



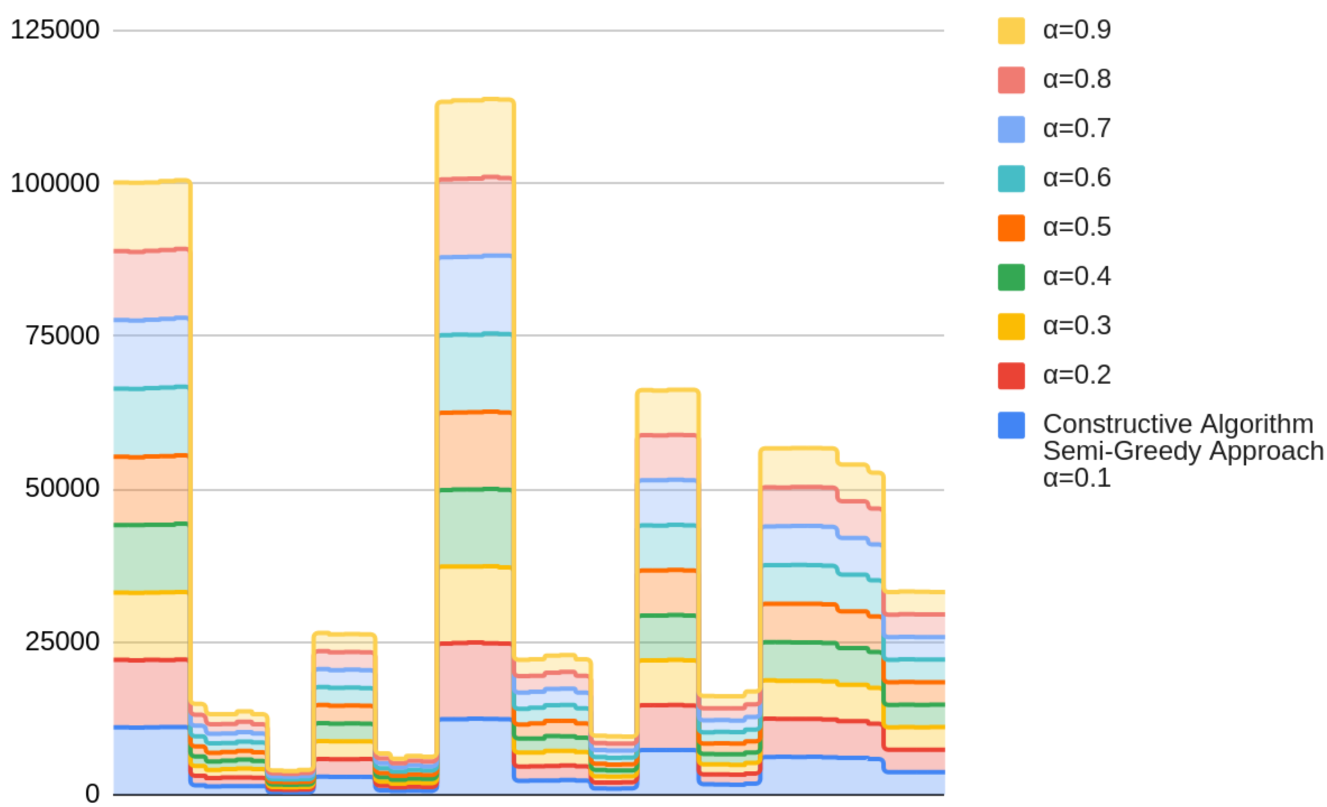
b. Part 2:

Problem			Constructive Algorithm								
FileName	Vertex	Edge	Semi-Greedy Approach								
			α=0.1	α=0.2	α=0.3	α=0.4	α=0.5	α=0.6	α=0.7	α=0.8	α=0.9
G1	800	19176	11000	11061	10990	11070	11172	11149	11249	11255	11245
G2	800	19176	10989	10997	11034	11082	11116	11167	11192	11226	11344
G3	800	19176	10996	11054	11028	11093	11189	11175	11166	11278	11227
G4	800	19176	11040	10974	11082	11084	11206	11229	11263	11246	11313
G5	800	19176	11040	11084	11031	11197	11181	11238	11262	11288	11234
G6	800	19176	1541	1555	1568	1579	1632	1663	1758	1786	1770
G7	800	19176	1323	1371	1362	1379	1403	1560	1549	1548	1674
G8	800	19176	1405	1405	1394	1363	1391	1473	1577	1557	1611
G9	800	19176	1398	1427	1442	1454	1437	1523	1569	1689	1669
G10	800	19176	1376	1380	1408	1365	1411	1578	1474	1535	1630
G11	800	1600	412	404	420	424	420	442	482	482	488
G12	800	1600	394	400	412	416	424	436	452	464	468
G13	800	1600	396	436	420	406	430	460	482	510	516
G14	800	4694	2924	2917	2942	2934	2945	2931	2960	2933	2956
G15	800	4661	2901	2902	2911	2910	2916	2919	2920	2918	2927
G16	800	4672	2895	2904	2934	2933	2923	2913	2938	2907	2921
G17	800	4667	2897	2918	2892	2904	2917	2914	2932	2928	2923
G18	800	4694	713	745	682	692	726	755	810	804	837
G19	800	4661	628	577	592	647	637	674	713	692	712
G20	800	4672	669	641	656	655	673	761	755	753	773
G21	800	4667	627	635	641	658	675	696	714	769	781
G22	2000	19990	12334	12394	12588	12529	12647	12719	12725	12754	12723
G23	2000	19990	12328	12472	12515	12614	12636	12734	12707	12773	12821
G24	2000	19990	12415	12468	12463	12579	12685	12658	12751	12802	12785

G25	2000	19990	12374	12390	12614	12641	12670	12799	12753	12842	12784
G26	2000	19990	12340	12399	12458	12689	12663	12837	12824	12732	12810
G27	2000	19990	2273	2318	2312	2281	2272	2586	2671	2667	2683
G28	2000	19990	2291	2338	2276	2256	2476	2587	2591	2632	2687
G29	2000	19990	2316	2373	2453	2446	2497	2547	2666	2666	2786
G30	2000	19990	2375	2382	2421	2371	2520	2627	2630	2750	2753
G31	2000	19990	2286	2375	2310	2337	2363	2448	2587	2721	2713
G32	2000	4000	996	978	1030	1052	976	1098	1168	1162	1196
G33	2000	4000	994	1002	954	990	990	1106	1160	1164	1208
G34	2000	4000	1006	1000	978	952	1012	1100	1148	1146	1180
G35	2000	11778	7309	7318	7339	7353	7390	7369	7377	7384	7393
G36	2000	11766	7301	7331	7361	7334	7347	7364	7397	7357	7349
G37	2000	11785	7319	7350	7372	7365	7352	7394	7370	7366	7362
G38	2000	11779	7293	7313	7375	7333	7359	7398	7383	7400	7389
G39	2000	11778	1690	1638	1651	1678	1700	1833	1952	1987	2019
G40	2000	11766	1662	1652	1648	1653	1786	1887	1929	1935	1964
G41	2000	11785	1629	1625	1607	1733	1753	1897	1877	1996	1993
G42	2000	11779	1804	1666	1721	1713	1829	1907	2067	2063	2113
G43	1000	9990	6174	6251	6268	6265	6279	6312	6326	6413	6373
G44	1000	9990	6198	6194	6229	6308	6322	6307	6359	6356	6390
G45	1000	9990	6155	6219	6249	6295	6331	6365	6364	6352	6400
G46	1000	9990	6176	6221	6210	6306	6316	6352	6403	6376	6356
G47	1000	9990	6129	6162	6245	6294	6302	6352	6364	6405	6418
G48	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000	6000
G49	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000	6000
G50	3000	6000	5824	5804	5840	5866	5856	5880	5880	5880	5880
G51	1000	5909	3659	3677	3683	3680	3707	3653	3704	3691	3717
G52	1000	5916	3675	3670	3700	3702	3703	3690	3700	3712	3706
G53	1000	5914	3678	3664	3656	3703	3696	3701	3693	3705	3712
G54	1000	5916	3684	3672	3684	3674	3691	3684	3705	3692	3689

In this table, we showed semi-greedy results with all possible alpha.

We know, if alpha increases, randomness decreases. And so, result will be more optimized. But as randomness is included, this is not always guaranteed .



4. Discussion:

In semi-greedy approach, as randomness is always included, it does not guarantee that more alpha will always give more optimized result.So, here random result is found. In GRASP, we set alpha=0.6 and then applied GRASP.From our observation, we noticed GRASP gives the most optimized result , as it every time starts from different vertices and continues iterations , and whenever it gets the better result ,it updates its value. Here, alpha=0.6 is used in GRASP . If we increase alphas value, then more optimized result will be found. So, yes GRASP is a better algorithm to solve Max-cut Problem.

5. Conclusion:

In this assignment, we implemented randomized algo, semi-greedy , greedy and GRASP and compared their results. By comparing, it is found that GRASP always gives the most optimized result, though less than the upper bound. As Max-cut is an NP-Hard problem, so no polynomial algorithm can provide the most optimized result which is equal to the upper bound. So, the solution by GRASP algorithm is acceptable.