Maximum Cut Problem

1905072-Mahir Labib Dihan

Introduction:

1. **Construction:** Used 3 types of construction algorithm as mentioned in the specification. For Randomized Algorithm used uniform distribution(Discrete) of 0 and 1 to ensure each node has same probability of going to one of the partitions. For SemiGreedy algorithm, used value based mechanism for building RCL. Used uniform distribution (Continuous) to randomly generate α between 0 to 1.
2. **Local Search:** Implemented two types of local search, one from the specification and one improved version with best choice strategy and sideway moves, which is explained in details below.
3. **Grasp:** Used the pseudo code of grasp from specification.

Improvements:

I used the pseudocode from the specification as a reference. Below are the improvements I added.

1. **Generate neighbors in random sequence:**

The *LocalSearch* function chooses the first improving neighbor (First Choice Hill Climbing). Which means the order of generating neighbors' matters. At line 4 of *LocalSearch* function the nodes are iterated from 1 to |V|. So, if the same initial solution ($x = \{S, S'\}$) is passed to this function it will return the same improved solution. So, we won't gain anything from it. That's why in my code **I have iterated the nodes randomly which generated neighbors in random sequence every time.**

```
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;
```

2. **Go to Best improving neighbor:**

As I mentioned earlier the pseudocode uses First Choice Hillclimbing. But there is a Best choice version of hill climbing, which performs better in some cases. So, I combined both in my grasp. Like **for 50% of grasp iteration I have used First Choice and the other 50% Best choice hill climbing.**

3. **Allowed Sideway moves:**

This is the only improvement which is guaranteed to improve the solution in all cases.

If we look at the pseudocode again, it doesn't allow sideway(flat) moves. Which was added in my code. The first challenge I faced was, my local search fell into a cycle. To avoid cycles, I have tracked my previous sideway moves. Which prevents from switching the same vertex from one partition to the other if it's a sideway move.

a. Kept a $visited$ array of size V x
b. When I find a sideway move, which means if node v is switched from S to T no improvement is achieved. In that case we set $visited[v] \leftarrow true$.
c. When I find an uphill move, which means improvement is possible. In that case $visited$ is cleared. Because we will encounter cycle only for consecutive sideway moves.
d. Again, when a sideway move is found for node v, we will allow it only if $visited[v] = false$. Because if we had allowed this move it is highly probable that we will go back to a previous solution, thus we will fall into a cycle.

After these improvements, i**n G48 and G49 I have reached the best-known max cut.** And the rest of the test cases also improved significantly.

Summary Table:

Used constant seed (1905072) for my random number generator so that I could reproduce the results.

Problem			Constructive Algorithm			Local Search - 1		Local Search - 2		GRASP		Upper bound
Name	|V| or n	|E| or m	Randomized-1	Greedy-1	Semi Greedy-1	Iterations	Best Value	Iterations	Best Value	Iterations	Best Value	
G1	800	19176	9584.53	10955	11117.1	140.71	11358.6	294.12	11440.5	809	11557	12078
G2	800	19176	9585.08	11029	11115	138.38	11353.5	305.74	11445	308	11566	12084
G3	800	19176	9572.63	11011	11118.9	135.62	11342.9	299.96	11434.8	375	11573	12077
G4	800	19176	9578.71	11030	11117.5	136.48	11368.1	303.91	11460.5	50	11569	
G5	800	19176	9581.12	10989	11122.4	139.51	11350	319.49	11449	678	11572	
G6	800	19176	67.54	1514	1618.39	144.57	1893.11	319.29	1992.75	89	2117	
G7	800	19176	-76.66	1361	1452.29	144.92	1730.44	310.79	1824.17	430	1948	
G8	800	19176	-101.76	1330	1461.5	141.54	1734.53	314.88	1834.61	219	1957	
G9	800	19176	-34.91	1506	1498.5	144.36	1780.38	304.01	1868.51	556	1974	
G10	800	19176	-74	1349	1450.73	139.06	1724.82	324.37	1825.43	96	1929	
G11	800	1600	15.86	433	428.94	15.09	442.42	986.78	531.66	760	552	627
G12	800	1600	-0.32	417	413.62	16.11	429.6	976.27	524.32	776	546	621
G13	800	1600	15.64	431	436.96	18.25	454.58	1030.49	550.52	500	576	645

G14	800	4694	2344.12	2889	2929.19	44.87	2944.33	706.43	3006.86	821	3031	3187
G15	800	4661	2336.5	2872	2906.4	43.67	2925.73	710.1	2984.67	103	3013	3169
G16	800	4672	2334.78	2877	2908.67	44.8	2928.61	682.61	2988.27	958	3017	3172
G17	800	4667	2336.72	2863	2905.44	45.96	2925.47	679.3	2981.84	883	3013	
G18	800	4694	29.84	779	750.09	71.79	836.83	608.02	912.07	95	962	
G19	800	4661	-59.74	680	662.11	72.41	747.41	577.29	818.38	316	875	
G20	800	4672	-18.48	697	691.34	70.33	774.77	596.02	845.73	607	909	
G21	800	4667	-30.84	703	680.95	74.92	770.51	553.65	841.78	367	899	
G22	2000	19990	9996.02	12345	12562.8	240.75	12810.4	1081.37	13041.6	234	13186	14123
G23	2000	19990	10000.5	12290	12561.3	239.33	12809.5	1103.19	13051.3	316	13223	14129
G24	2000	19990	9989	12366	12566.5	241.73	12814.4	1026.93	13049.7	968	13172	14131
G25	2000	19990	9988.46	12407	12556.1	237.82	12811	1180.78	13053.5	839	13203	
G26	2000	19990	9987.96	12354	12561.8	241.75	12802.9	1125.97	13044.4	143	13172	
G27	2000	19990	-15.95	2315	2429.28	247.52	2795.82	1033.61	3033.75	542	3189	
G28	2000	19990	-50.16	2328	2407.99	243.26	2767.16	1132.25	3007.61	572	3132	
G29	2000	19990	39.19	2366	2499.08	252.45	2855.4	1125.98	3096.67	567	3222	
G30	2000	19990	33.84	2400	2499.95	250.88	2866.62	1125.77	3110.1	683	3267	
G31	2000	19990	-51.44	2289	2410.4	256.36	2779.6	1149.65	3024.05	37	3139	
G32	2000	4000	7.17	1057	1059.68	41.2	1100.5	2104.95	1308.12	131	1358	1560
G33	2000	4000	-16.72	999	1030.74	42.83	1066.5	2157.86	1286.54	379	1334	1537
G34	2000	4000	-21.51	987	1028.14	44.4	1066.94	2093.26	1291.2	831	1336	1541
G35	2000	11778	5901.46	7250.48	7340.51	95.37	7387.88	1829.15	7519.97	831	7575	8000
G36	2000	11766	5887.28	7223.59	7329.32	96.68	7382.59	1850.83	7516.64	62	7566	7996
G37	2000	11785	5884.84	7262.26	7342.42	93.99	7395.55	1916.93	7527.47	457	7586	8009
G38	2000	11779	5897.62	7302.9	7338.51	93.71	7391.45	1735.19	7520.63	282	7585	
G39	2000	11778	17.75	1858.3	1758.91	184.3	2006.47	1837.48	2193.01	428	2299	
G40	2000	11766	-55.55	1831.13	1752.4	188.1	1975.8	1720.17	2165.53	294	2258	
G41	2000	11785	2.69	1872.82	1756.2	184.53	1987.06	1797.76	2173.25	64	2283	
G42	2000	11779	62.49	1985.77	1838.76	190.81	2073.86	1720.44	2252.83	722	2365	
G43	1000	9990	4991.58	6186.57	6266.67	118.94	6390.95	493.73	6510.31	959	6591	7027
G44	1000	9990	4995.28	6156.3	6263.53	118.02	6392.86	496.59	6508.61	480	6609	7022
G45	1000	9990	4992.1	6177.5	6258.19	117.58	6386.93	487.08	6502.95	950	6591	7020
G46	1000	9990	4995.65	6149.6	6266.03	115.92	6391.82	473.4	6506.68	621	6600	

G47	1000	9990	4983.51	6218.57	6271.19	123.83	6404.28	467.43	6515.06	777	6592	
G48	3000	6000	2996.56	5997.78	5372.5	73.87	5078.14	4418.25	5789.32	1	6000	6000
G49	3000	6000	3000.68	5998.3	5379.54	74.19	5072.3	4636.95	5801.82	25	6000	6000
G50	3000	6000	2998.36	5877.46	5387.06	74.92	5073.14	4348.45	5752.82	104	5876	5988
G51	1000	5909	2955.37	3624.76	3674.32	54.91	3693.76	903.97	3767.07	107	3806	
G52	1000	5916	2958.75	3648.48	3681.33	53.84	3701.07	886.88	3773.89	159	3806	
G53	1000	5914	2953.09	3630.65	3674.48	52.21	3695.82	987.05	3775.39	729	3805	
G54	1000	5916	2965.49	3631.67	3674.77	56.71	3698.13	931.02	3773.71	116	3807	

Output Explanation:

Executed 3 types of construction algorithm for 100 times and took the average of the max cut of them. As expected, Randomized algorithm gives very bad results. There is a battle between Greedy and Semigreedy. As semigreedy uses randomization, it can give better results than greedy sometimes.

I have executed two types of local search (As described above) with 100 iterations. Tracked the count of local moves and max cut and took their average. As we can see Local Search-2 needs significantly more local moves as sideway moves is allowed here. We can control this by limiting the maximum consecutive sideway moves.In this experiment I have set the limit to 100. If we look at the best value column localsearch-2 made the solution significantly better.Which justifies the improvements I made.

For GRASP, semigreedy and localsearch-2 is used. And iterated 1000 times. Maximum cut and number of iterations to achieve that is mentioned in summary table.