Trabalho de Implementação 03

Thiago Martin Poppe

16/12/2021

1 Metaheurística - GRASP

Para o terceiro trabalho de implementação foi solicitado que construíssemos uma metaheurística para o problema do caixeiro viajante. Optei por utilizar a metaheurística GRASP (ou *Greedy Randomized Adaptive Search Procedure*), por ser uma metaheurística relativamente simples mas que ainda assim produz resultados muito interessantes. Para esse trabalho eu **não** desenvolvi um GRASP Reativo, sendo assim, o parâmetro α permanece fixo durante toda a execução do algoritmo.

Para a etapa de **construção** presente na metaheurística, foi utilizado uma variação da heurística construtiva *Nearest Neighbors* (NN), desenvolvida no trabalho de implementação 1, onde, ao invés de escolhermos sempre o vizinho mais próximo, nós escolhemos aleatoriamente um vizinho que pertença a nossa lista restrita de candidatos, construída da mesma forma que foi visto durante as aulas. Ou seja, definindo um limiar de forma que o vértice i fará parte da solução se $c_i \leq c^{\min} + \alpha(c^{\max} - c^{\min})$, onde com $\alpha = 0$ teremos na prática um NN e com $\alpha = 1$ um algoritmo completamente aleatorizado. Algo interessante de se notar é que o resultado obtido com $\alpha = 0$ não necessariamente será igual ao resultado de aplicarmos um NN + VND, já que, se tivermos mais de uma aresta com peso mínimo, a nossa heurística construtiva aleatorizada nem sempre dará a mesma resposta.

Para a etapa de **busca local** presente na metaheurística, foi utilizado o mesmo VND desenvolvido no trabalho de implementação 2 apenas por praticidade, uma vez que já tinhamos implementado um algoritmo de busca local relativamente interessante.

Uma outra coisa que devemos mencionar é que esse código foi escrito em C, já que iria demorar muito para executar a metaheurística em Python mesmo utilizando bibliotecas eficientes.

2 Resultados

A seguir podemos observar 3 tabelas que sumarizam os resultados obtidos com o GRASP para as instâncias fornecidas. Na primeira, podemos perceber que o tempo de execução foi consideravelmente maior do que o que gastamos no trabalho de implementação 2 (VND). A segunda tabela apenas traz uma comparação entre o que foi desenvolvido em cada um dos trabalhos de implementação. Já a terceira tabela traz uma comparação entre os diferentes GRASPs para valores de $\alpha = [0, 0.25, 0.5, 0.75, 1]$, fixando o número de iterações em 1000.

	instância	número de cidades	tempo de execução	custo da solução (GRASP)	distância utilizada
0	att48	48	0.66	10638	ATT
1	berlin52	52	0.76	7801	EUC_2D
2	kroA100	100	7.59	21456	EUC_2D
3	kroA150	150	27.92	27363	EUC_2D
4	kroA200	200	81.74	30410	EUC_2D
5	kroB100	100	7.51	22540	EUC_2D
6	kroB150	150	31.93	26602	EUC_2D
7	kroB200	200	77.08	30457	EUC_2D
8	kroC100	100	7.67	20852	EUC_2D
9	kroD100	100	7.71	21690	EUC_2D
10	kroE100	100	7.79	22612	EUC_2D
11	lin105	105	9.39	14412	EUC_2D
12	pr107	107	10.45	44881	EUC_2D
13	pr124	124	15.78	59159	EUC_2D
14	pr136	136	21.46	98322	EUC_2D
15	pr144	144	23.53	58603	EUC_2D
16	pr152	152	26.60	74844	EUC_2D
17	pr76	76	3.31	109335	EUC_2D
18	rat195	195	68.46	2415	EUC_2D
19	rat99	99	7.15	1230	EUC_2D
20	st70	70	2.25	682	EUC_2D

Figure 1: Resultado do GRASP com 1000 iterações e $\alpha = 0.25$ para cada instância

	instância	número de cidades	custo da solução (NN)	custo da solução (2-OPT)	custo da solução (VND)	custo da solução (GRASP)
0	att48	48	12861	10782	10782	10638
1	berlin52	52	8980	8385	8043	7801
2	kroA100	100	27807	23737	22489	21456
3	kroA150	150	33633	28612	28612	27363
4	kroA200	200	35859	31749	31518	30410
5	kroB100	100	29158	25393	25393	22540
6	kroB150	150	34499	28435	27442	26602
7	kroB200	200	36980	33375	33375	30457
8	kroC100	100	26227	23107	22968	20852
9	kroD100	100	26947	24721	24721	21690
10	kroE100	100	27460	24459	24268	22612
11	lin105	105	20356	18281	18230	14412
12	pr107	107	46680	45340	44921	44881
13	pr124	124	69297	61910	61823	59159
14	pr136	136	120769	110103	110103	98322
15	pr144	144	61652	61400	61400	58603
16	pr152	152	85699	79996	79996	74844
17	pr76	76	153462	121315	121128	109335
18	rat195	195	2752	2497	2493	2415
19	rat99	99	1554	1418	1399	1230
20	st70	70	830	770	709	682

Figure 2: Comparação da solução dos algoritmos para cada instância

	instância	GRASP (alpha=0.00)	GRASP (alpha=0.25)	GRASP (alpha=0.50)	GRASP (alpha=0.75)	GRASP (alpha=1.00)
0	att48	10782	10638	10653	10663	10653
1	berlin52	8043	7801	7718	7727	7682
2	kroA100	22489	21456	21470	21346	21369
3	kroA150	28612	27363	27335	27399	27012
4	kroA200	31518	30410	29950	30141	30099
5	kroB100	25393	22540	22503	22664	22484
6	kroB150	26933	26602	26571	26560	26698
7	kroB200	32060	30457	30371	30643	30345
8	kroC100	22968	20852	21086	21048	20884
9	kroD100	24721	21690	21704	21703	21713
10	kroE100	24169	22612	22564	22483	22328
11	lin105	18230	14412	14529	14536	14605
12	pr107	44641	44881	45024	44967	45008
13	pr124	61823	59159	59227	59777	59246
14	pr136	107170	98322	98952	99603	99805
15	pr144	61400	58603	59070	58636	59005
16	pr152	79996	74844	74570	75451	74572
17	pr76	121128	109335	109150	108590	109225
18	rat195	2477	2415	2407	2398	2424
19	rat99	1298	1230	1231	1241	1234
20	st70	709	682	679	677	676

Figure 3: Comparação dos GRASPs com 1000 iterações