Multiple Traveling Thieves Problem

Maria Eduarda Oliveira Miranda Rafael Victor Costa Braz

Outubro de 2019

1 Introdução

O problema do mochileiro viajante (TTP) é a combinação de dois problemas NP-difíceis muito conhecidos: o problema da mochila e o problema do caixeiro viajante.

Para tal, os dados de entrada consistem em um conjunto de N cidades e suas respectivas coordenadas (considera-se que todas as cidades possuem ligações entre si) e, para cada uma, há um conjunto de M itens com seus respectivos valores e pesos. Além disso, há a capacidade máxima da mochila W, e as velocidades máxima e mínima dos ladrões. No TTP, o objetivo é maximizar o lucro respeitando as limitações da mochila. Nele, um mochileiro pode passar por cada cidade uma única vez e paga uma taxa de aluguel por cada unidade de seu percurso. Para cada item pego, o peso de sua mochila aumenta e sua velocidade diminui linearmente, respeitando o limite inferior dado nas especificações. Assim, o objetivo é maximizar a função:

$$Z(\pi, P) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R(\frac{d_{x_n x_1}}{v_{max} - \upsilon W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_{max} - \upsilon W_{x_i}})$$

com π sendo a rota do ladrão e P o plano de coleta de itens.

Proposto em 2016, o problema dos múltiplos mochileiros viajantes (MTTP) visa deixar a ideia do TTP mais próxima da realidade. No MTTP, em que há um conjunto de pessoas que devem trabalhar em conjunto para maximizar o lucro, sendo que a velocidade de cada viajante depende de sua mochila individual, porém a capacidade máxima é dada para todos. Isso pode ser explicado como se cada um coletasse um subconjunto de itens para, posteriormente, juntar todos em um depósito de capacidade W. Nesta versão, assim como no TTP não há a necessidade de visitar todas as cidades, porém uma cidade pode ser visitada por vários ladrões em momentos diferentes (CHAND; WAGNER, 2016).

2 Métodos

2.1 Construção da solução inicial

Para a solução inicial, foi proposto um algoritmo guloso construtivo (1), que foi criado com o objetivo de distribuir os itens entre os ladrões de forma que cada um não viaje muito para obter um item se não houver um benefício alto, ou seja, ele

pega o item que não foi pego ainda com o maior valor de custo benefício, dado por $\frac{valor\ do\ item}{A*distância\ entre\ a\ cidade\ atual\ e\ a\ cidade\ do\ item\ *\ aluguel\ +\ B*peso\ do\ item}$.

Algoritmo 1: Solução Inicial

```
Solução_Inicial ():
enquanto há itens que cabem na mochila faça
    para ladrao em ladroes faça
        para item em items faça
            custo_beneficio ← valor do item / (A*distância entre a cidade atual e a cidade do item * aluguel + B*peso do item);
        fim
        item ← item disponível com maior custo_beneficio;
        π de ladrão ← cidade do item;
        P de ladrão ← item;
        Mochila de ladrão += peso do item;
    fim
fim
retorna solução_inicial
```

Para o algoritmo temos uma lista de itens P associada à mochila de cada ladrão, e uma lista global que mostra se o item já foi pego ou não.

2.2 Vizinhanças

Trade_items: Nessa vizinhança o ladrão escolhido troca um item de sua mochila por um item ainda não pego.

Move_cities: Nessa vizinhança o ladrão escolhido tem uma das cidades, de sua rota, trocada a ordem em que a mesma será visitada para uma nova posição.

Shuffle_items: Nessa vizinhança um número(>1) de ladrões trocam entre si um item de suas mochilas.

Swap_cities: Nessa vizinhança o ladrão escolhido tem duas das cidades, de sua rota, trocadas em relação a ordem de visitação.

Remove_items: Nessa vizinhança o ladrão escolhido tem um de seus itens removido de sua mochila.

Add_new_items: Nessa vizinhança o ladrão escolhido adiciona a sua mochila um novo item ainda não pego.

2.3 Heurísticas propostas

Nesta seção, serão descritas as heurísticas propostas em mais detalhes e a função de avaliação utilizada em todas as heurísticas apresentadas a seguir foi a própria função objetivo.

Foi utilizado o Iterated Race for Automatic Algorithm Configuration (IRACE[1]), um pacote que busca encontrar a melhor configuração de parâmetros para algoritmos de otimização, para encontrar os melhores parâmetros. Porém, por ser um problema

[1] http://iridia.ulb.ac.be/irace/

complexo, o IRACE foi executado apenas para 1 ladrão e para as instâncias dos tipos: Berlin52_n51, Berlin52_n153, Berlin52_n255, Berlin52_n510, Rat195_n194 e Rat195_n582. Os parâmetros escolhidos foram replicados para o restante, devido à limitações de tempo.

2.3.1 Variable Neighborhood Search with Iterated Local Search

A primeira heurística proposta foi baseada na junção de duas metaheurísticas, sendo elas a Variable Neighborhood Search (MLADENOVIĆ; HANSEN, 1997) e Iterated Local Search (LOURENÇO; MARTIN; STÜTZLE, 2003), em 3. Primeiramente é construída uma solução inicial viável e, partindo dela, é realizado o processo de busca local com o método first improvement até que a busca local chegue em um mínimo local, nesse momento a vizinhança do problema é então trocada e novamente se aplica a busca local(first improvement). Caso encontre algum vizinho que melhore a solução, a vizinhança volta, novamente, para a inicial. Esse processo é repetido até que não haja melhora em nenhuma vizinhança implementada. Nesse momento é aplicado o processo de perturbação, em 2, do ILS e então o VNS retorna a partir do resultado da perturbação. Esse processo é repetido até que haja n perturbações. O resultado é dado pela melhor solução encontrada durante todo o processo.

Algoritmo 2: Perturbação

disturbe(s, qt_disturbe):
$v \leftarrow vizinhanca\_aleatoria$;
$i \leftarrow 0$ **enquanto** $i! = qt\_disturbe$ **faça**
  $s \leftarrow vizinho\_aleatorio(v, s)$;
  $i \leftarrow i + 1$
fim
retorna s

Algoritmo 3: Variable Neighborhood Search + Iterated Local Search

```
VNS_ILS(qt_disturbe, max_disturbe):
s ← solucaoo_inicial();
v ← vizinhanca_inicial;
n_disturbe ← 0;
enquanto não alcançar o critério de parada faça
    s ← busca_local(s, v);
    se achou vizinho melhor então
        v ← vizinhanca_inicial;
    fim
    senão
        se v é a última vizinhança então
            se n_disturbe = max_disturbe então
                retorna Melhor solução encontrada
            fim
            s ← perturbacao(s, qt_disturbe);
            n_disturbe ← n_disturbe + 1;
        fim
        senão
            v ← proxima_vizinhanca;
        fim
    fim
fim
```

Após os testes feitos pelo IRACE, levando em consideração as restrições impostas, os paramêtros $qt\_disturbe$ e $max\_disturbe$ foram setados como 7 e 9, respectivamente.

2.3.2 Greedy Randomized Adaptive Search Procedure

A segunda heurística proposta foi baseada na metaheurística Greedy Randomized Adaptive Search Procedure (GRASP), em 4 , que consiste em uma metaheuristica que é aplicada várias vezes e contém duas fases: Construção e Busca Local. Primeiramente é construída uma solução inicial viável e, com ela é realizada uma busca local para que seja encontrado melhor vizinho. O resultado é dado pela melhor solução encontrada durante o processo (RESENDE; RIBEIRO, 2010).

Algoritmo 4: Greedy Randomized Adaptive Search Procedure

```
GRASP (I_max, SizeRCL, IR_max):
enquanto I < I_max faça
    s ← solução_inicial(SizeRCL, IR_max);
    s' ← busca_local(s);
    se melhor_solução < s então
        melhor_solução ← s;
    fim
fim
retorna melhor_solução
```

A proposta da solução inicial segue aquela explicada na seção 1, que analisa o custo benefício dos itens. A diferença basíca é que, para as IR_{max} primeiras iterações a solução inicial pega um item aleatório dentre os $SizeRCL$ melhores, e para as iterações restantes

ele pega o item com maior custo benefício.

Em relação aos paramêtros, após os testes feitos pelo IRACE, o mesmo retornou que a melhor combinação de paramêtros, dadas às restrições ditas anteriormente, foram $IR_{max} = 10$ e $SizeRCL = 0.70$.

3 Resultados

Os algoritmos propostos foram implementados utilizando C++ e os testes foram realizados em um computador com processador $Intel(R)\ Core^{TM} i7 - 77003.60GHz \times 8$ com 16GB de memória RAM e sistema operacional Ubuntu 16.04 LTS. Nas tabelas — a seguir, apresenta-se os resultados dos algorimos propostos na seção 2. Como o MTTP é um extensão do TTP, que têm muitas instâncias feitas para testes, escolhemos uma de cada tipo de instância do TTP e fizemos testes para com o número de ladrões 1(1), 2(2), 3(3), 4(4), 5(5).

	1 Ladrão			
	GRASP		VNS	
	Valor	Tempo(s)	Valor	Tempo(s)
berlin52_n153_uncorr-similar-weights_03	11134,83	0,03	18678,83	5,19
berlin52_n510_uncorr_07	-14461,1	0,36	72054,56	603,51
rat195_n1940_bounded-strongly-corr_07	88312,32	6,19	194257,09	668,43
a280_n837_bounded-strongly-corr_03	105761,03	0,9	145944,23	627,05
rat783_n2346_bounded-strongly-corr_03	351207,74	8,64	354085,57	694,08
dsj1000_n2997_uncorr_07	67557,72	29,96	-456511,21	605,32
pcb3038_n9111_uncorr-similar-weights_07	559937,21	43,15	722188,88	622,24

Figura 1: Resultados para 1 ladrão

	2 Ladrões			
	GRASP		VNS	
	Valor	Tempo(s)	Valor	Tempo(s)
berlin52_n153_uncorr-similar-weights_03	2088,26	0,1	18895,48	2,19
berlin52_n510_uncorr_07	-89281,41	0,4	35548,92	273,71
rat195_n1940_bounded-strongly-corr_07	-338339	12,81	-95991,46	601,14
a280_n837_bounded-strongly-corr_03	54192,29	3,38	155803,08	610,91
rat783_n2346_bounded-strongly-corr_03	272277,17	32,44	361840,39	612,23
dsj1000_n2997_uncorr_07	-307608,55	101,44	-305317,06	656,58
pcb3038_n9111_uncorr-similar-weights_07	-226624,4	896,11	73338,61	621,61

Figura 2: Resultados para 2 ladrões

	3 Ladrões			
	GRASP		VNS	
	Valor	Tempo(s)	Valor	Tempo(s)
berlin52_n153_uncorr-similar-weights_03	-12128,32	0,1	16749,26	1,2
berlin52_n510_uncorr_07	-133438,27	0,45	6632,83	256,64
rat195_n1940_bounded-strongly-corr_07	-914004,22	21,28	-564955,62	601,4
a280_n837_bounded-strongly-corr_03	-39118,39	3,08	134273,69	607,94
rat783_n2346_bounded-strongly-corr_03	92635,71	23,11	249796,6	640,63
dsj1000_n2997_uncorr_07	-501613,96	51,13	-667048,37	606,19
pcb3038_n9111_uncorr-similar-weights_07	-1079869,24	434,03	-640424,22	652,01

Figura 3: Resultados para 3 ladrões

	4 Ladrões			
	GRASP		VNS	
	Valor	Tempo(s)	Valor	Tempo(s)
berlin52_n153_uncorr-similar-weights_03	-17900,61	0,08	16146,08	1,38
berlin52_n510_uncorr_07	-163899,41	0,65	-25064,76	223,66
rat195_n1940_bounded-strongly-corr_07	-1271705,27	19,93	-1071901,18	618,36
a280_n837_bounded-strongly-corr_03	-94288,78	3,49	123495,26	606,13
rat783_n2346_bounded-strongly-corr_03	-73176,13	46,78	179247,86	618,35
dsj1000_n2997_uncorr_07	-645346,17	17,66	-915334,53	603,49
pcb3038_n9111_uncorr-similar-weights_07	-1814100,07	805,68	-1136476,37	609

Figura 4: Resultados para 4 ladrões

	5 Ladrões			
	GRASP		VNS	
	Valor	Tempo(s)	Valor	Tempo(s)
berlin52_n153_uncorr-similar-weights_03	-20147,94	0,07	13743,12	0,51
berlin52_n510_uncorr_07	-249903,28	0,52	-45006,97	183,8
rat195_n1940_bounded-strongly-corr_07	-1499717,34	24,52	-1567812,03	667,72
a280_n837_bounded-strongly-corr_03	-177929,58	5,42	110102,89	600,43
rat783_n2346_bounded-strongly-corr_03	-190320,78	37,69	101795,6	668,65
dsj1000_n2997_uncorr_07	-840325,25	28,21	-1183364,83	648,21
pcb3038_n9111_uncorr-similar-weights_07	-1474013,77	581,18	-1643550,83	702,98

Figura 5: Resultados para 5 ladrões

4 Conclusões

Foi possível observar que o VNS+ILS apresentou melhores resultados como um todo, porém com um tempo de execução muito maior que o do GRASP, de modo que o mesmo teve que ser limitado a um tempo de 10min por instância. Nos dois algoritmos, é notável que os parâmetros influenciaram muito nos resultados, uma vez que ele apresenta resultados satisfatórios nas execuções com um ladrão, que foi a que teve a combinação de parâmetros escolhida metodicamente, enquanto que para outros ladrões essa combinação foi apenas replicada sem haver uma validação.

Uma possível forma de melhorar os dois algoritmos seria através da criação de uma solução inicial mais elaborada, que possa explorar mais características importantes para o problema e, também, a execução de testes de paramêtros de maneira mais justa, considerando todos os ladrões e um número maior de instâncias.

Referências

CHAND, S.; WAGNER, M. Fast heuristics for the multiple traveling thieves problem. In: ACM. *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. [S.l.], 2016. p. 293–300. Citado na página 1.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 320–353. Citado na página 3.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & operations research*, Elsevier, v. 24, n. 11, p. 1097–1100, 1997. Citado na página 3.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances and applications. *Handbook of metaheuristics*, Citeseer, v. 146, p. 281–317, 2010. Citado na página 4.