MDVSP - Otimização Combinatória

Lucas Hagen, Leonardo Bombardelli

December 2018

1 Introdução

Este trabalho tem como objetivo a implementação de uma meta-heurística para a resolução do problema de Multiple Depot Vehicle Scheduling (MDVSP). A meta-heurística implementada foi a Busca Tabu, cujos resultados serão apresentados nas seções seguintes.

Este problema consiste em uma lista de garagens, uma lista de trajetos a serem atendidos, um máximo de ônibus por garagem e uma matriz de custos das transições entre locais (garagens e trajetos).

O MDVSP foi também resolvido utilizando o solver GLPK. Devido à complexidade do problema, nem todas as instâncias puderam ser resolvidas com o solver. Algumas instâncias não possuem um valor ótimo conhecido.

2 Problema

Instância: é composta por um conjunto K = 1, . . . , k de garagens e um conjunto T = 1, . . . , t de viagens. Considerando P = 1, . . . , k, k+1, . . . , k+t, a matriz de adjacências M_{ij} indica a viabilidade de execução dos pares de viagens $i, j \in P$. Cada posição da matriz apresenta um valor de custo associado, ou -1 denotando uma sequência de viagens infactível. Além disso, cada garagem $k \in K$ apresenta um número máximo de veículos disponíveis $v_k \in N$

Solução: diversas sequências de viagens (caminhos). Cada veículo sai de uma das garagens do problema, executa uma sequência de viagens compatíveis, e retorna para a sua respectiva garagem. Objetivo: Dada uma matriz de conexões Mi,j e a capacidade das garagens, encontre a solução de menor custo em que cada viagem de T é executada por exatamente um dos veículos da solução. O número de veículos utilizados não deve ultrapassar a capacidade das garagens do problema

Objetivo: Dada uma matriz de conexões M_{ij} e a capacidade das garagens, encontre a solução de menor custo em que cada viagem de T é executada por exatamente um dos veículos da solução. O número de veículos utilizados não deve ultrapassar a capacidade das garagens do problema.

3 Formulação em Programação Inteira

Parâmetros:

$K = 1, ..., k$ garagens

$T = 1, ..., t$ viagens

$V_k = k \cup T$ nodos com apenas o depot selecionado

$A_k \in (i, j)$, onde $i, j \in V_k \wedge M_{ij} \geq 0$ arcos factíveis para a garagem k

$M_{ij}, i, j \in (K \cup T)$, onde:

$$M_{ij} = \begin{cases} -1 & \text{Transição infactível} \\ \geq 0 & \text{Custo da transição} \end{cases}$$

Variáveis:

$X_{k,(i,j)} \in \{0, 1\} \quad \forall k \in K, \forall (i, j) \in A_k$ onde:

$$X_{k,(i,j)} = \begin{cases} 1 & \text{Caso a existe a transição } i, j \in S \\ 0 & \text{Caso contrário.} \end{cases}$$

Função Objetivo:

$$\min \sum_{k \in K, (i,j) \in A_k} M_{ij} X_{k,(i,j)}$$

Restrições:

$$\sum_{k \in K, (i,j) \in A_k} X_{k,(i,j)} = 1 \qquad \forall j \in T \tag{1}$$

$$\sum_{(k,j) \in A_k} X_{k,(k,j)} \leq v_k \qquad \forall k \in K \tag{2}$$

$$\sum_{(i,n) \in A_k} X_{k,(i,)} = \sum_{(n,j) \in A_k} X_{k,(n,j)} \qquad \forall k \in K, \forall n \in V_k \tag{3}$$

$$X_{k,(i,j)} \in 0, 1 \qquad \forall k \in K, \forall (i, j) \in (T \cup K) \tag{4}$$

A restrição (1) garante que cada viagem será atendida exatamente uma vez, seja por um ônibus saindo de uma garagem, ou por um ônibus que terminou outra viagem.

A restrição (2) garante que os limites de ônibus por garagem serão respeitados.

A restrição (3) garante que o fluxo será preservado, ou seja, sempre que um ônibus atender a uma viagem, ele deverá sair dessa viagem e então atender a outra viagem, ou retornar à garagem. Essa restrução garante também que um ônibus retorne apenas para a sua respectiva garage.

4 Algoritmo - Tabu Search

A metaheurística implementada foi a *Tabu Search* (Busca Tabu). Esse é um algoritmo de busca local que tenta explorar o espaço de soluções do problema, mas se difere dos demais pois visa reduzir o número de vizinhos não-aprimorantes a partir da restrição de vizinhanças que já foram visitadas recentemente. O algoritmo foi implementado utilizando a linguagem de programação Python 3, e foram utilizadas diversas técnicas e ferramentas dispostas pela linguagem para permitir uma eficiência maior do mesmo.

4.1 Representação do Problema

Abstraímos o problema para que as seguintes informações descrevessem uma instância do mesmo: *nDepots* é a quantidade de *depots* (garagens); *nTrips* é a quantidade de *trips* (viagens); *numBus* é a quantidade de veículos disposta em cada garagem; *depotToTrip*, *tripToTrip* e *tripToDepot* são matrizes que representam, respectivamente, custo de caminhos de *depot* para *trip*, *trip* para *trip* e *trip* para *depot*; *value* define o valor da solução atual; por fim, *listOfTripsPerBus* é uma lista com todas as viagens que cada veículo irá fazer, além de sua respectiva garagem. Com essa abstração das instâncias, o problema então se torna encontrar o conjunto de caminhos *listOfTripsPerBus* que minimize o valor *value*.

4.2 Principais Estruturas de Dados

As principais estruturas utilizadas foram matrizes, utilizadas para armazenar as informações de custo vas variáveis *depotToTrip*, *tripToTrip* e *tripToDepot*. Representamos cada viagem dos veículos como uma tupla (*depot*, *path*), onde *depot* contém a informação de qual garagem o mesmo sai e *path* uma lista que armazena o caminho a ser feito. *listOfTripsPerBus*, portanto, é uma lista que contém uma destas tuplas para cada veículo da solução.

4.3 Geração da Solução Inicial

Utilizamos uma técnica construtiva para a geração da solução inicial, visto que as técnicas de geração com soluções aleatórias se mostraram extremamente ineficientes para gerar uma solução inicial factível. Para isso, armazenamos todas as *trips* em uma lista de candidatos, selecionamos uma delas aleatoriamente, percorremos todos os nodos adjacentes a ele (removendo-os da lista de candidatos) e continuamos percorrendo este grafo linearmente até não termos nenhum nodo da lista de candidatos alcançável. Repetimos o processo até que todos os nodos estejam e um destes grafos direcionados, e selecionamos depots para cada um desses grafos. Como a escolha dos nodos a serem percorridos é aleatória, não temos como garantir que estaremos selecionando o grafo com o maior número de nodos neste algoritmo, e, portanto, não temos como garantir que teremos

veículos o suficiente para todos os caminhos gerados. Por tal motivo, esse algoritmo ou gera uma solução inicial factível para a instância do problema, ou não consegue gerar solução inicial e precisa reiniciar o processo com grafos diferentes.

4.4 Vizinhança e a Estratégia para Seleção dos Vizinhos

Definimos uma vizinhança como uma solução onde, se um nodo i de um caminho k é alcançado (tem valor diferente de -1) pelo nodo final de outro caminho j, o nodo i deixará de pertencer ao caminho k e pertencerá ao caminho j. Após essa troca, verificamos se esse nodo, agora no final do caminho j, alcança o primeiro nodo de algum outro dos caminhos. Se sim, significa que podemos concatenar estes dois caminhos, reduzindo o número de veículos sendo utilizados. Este algoritmo para geração de vizinhanças gera apenas vizinhanças que irão satisfazer todas as restrições do problema. Junto com o fato de que a solução inicial nunca gera instâncias que não satisfazem as restrições, temos certeza que toda solução desenvolvida no decorrer do algoritmo é uma solução válida para o problema. Na implementação do *Tabu Search*, ao modificarmos a posição de um nodo i, inserimos o mesmo na *Lista Tabu*, que impedirá o mesmo de ser selecionado para ser alterado por algumas próximas iterações. Com essa definição de vizinhança definida, selecionamos os vizinhos gerando n vizinhos à solução *best* e verificamos se algum deles oferece um resultado melhor que a mesma. Se algum deles oferece, substituímos *best* pelo atual melhor valor.

4.5 Parâmetros do Método

Após algumas rodadas de testes, descobrimos que um valor de vizinhança $n = 5$ ofereceu resultados razoáveis. Utilizamos um critério de parada que consistiu em executar k iterações do algoritmo, e descobrimos que após $k = 1000$ iterações a função objetivo já havia convergido para seu máximo local e raramente conseguia sair do mesmo. O parâmetro mais complexo de modificar, contudo, foi o tamanho da *Lista Tabu*. Tamanhos grandes ofereciam uma grande variedade nos grafos que eram utilizados e o resultado da função objetivo convergia rapidamente, mas geralmente ficava travada em valores muito altos. Tamanhos menores faziam a função objetivo convergir mais lentamente, mas nas iterações mais tardias do algoritmo víamos que, em média, os valores eram menores. Acabamos optando por uma solução de tamanho 20 na *Lista Tabu*, ou seja, apenas após 20 iterações poderemos escolher o mesmo nodo novamente.

5 Resultados

Como é possível ver na tabela 1, apenas para as instancias m4n500s2 e m4n500s4 foram encontradas respostas inteiras otimas. Par as outras soluções, o GLPK não achou nem a solução para a relaxação linear.

Tabela 1: Resultados GLPK

Instância	BKS	Relaxacao Linear	Melhor Sol.	Tempo GLPK (s)
m4n500s2	1283811	1.283.782,89	1.283.812,00	3.468,00
m4n500s4	1317077	1.317.044,27	1.317.078,00	9.146,00
m4n1000s3	2490812	n.e.	n.e.	3.605,00
m4n1000s4	2519191	n.e.	n.e.	3.609,00
m4n1500s1	3559176	n.e.	n.e.	3.607,00
m8n500s0	1292411	n.e.	n.e.	3.602,00
m8n1000s0	2422112	n.e.	n.e.	3.614,00
m8n1000s2	2556313	n.e.	n.e.	3.612,00
m8n1500s1	3802650	n.e.	n.e.	3.601,00
m8n1500s4	3704953	n.e.	n.e.	3.611,00

Tabela 2: Resultados Tabu Search (* = Médio)

Instância	BKS	Valor Inicial*	Melhor Valor*	Desv. Padrão	Temp. Exec*	Desvio %
m4n500s2	1283811	2.720.739,90	2.449.668,50	36.502,40	15,54	90,81
m4n500s4	1317077	2.765.295,60	2.480.353,30	39.865,32	9,62	88,32
m4n1000s3	2490812	5.512.355,20	5.023.256,20	75.844,11	33,75	101,67
m4n1000s4	2519191	5.528.866,90	4.953.745,20	63.114,55	16,35	96,64
m4n1500s1	3559176	8.143.274,80	7.412.879,80	84.889,25	28,85	108,28
m8n500s0	1292411	2.720.615,50	2.451.556,50	64.812,66	20,80	89,69
m8n1000s0	2422112	5.390.634,70	4.922.055,60	59.255,27	38,34	103,21
m8n1000s2	2556313	5.467.679,80	4.973.255,30	95.404,95	24,92	94,55
m8n1500s1	3802650	8.116.350,70	7.449.507,80	93.145,80	37,48	95,90
m8n1500s4	3704953	8.029.853,00	7.362.423,30	114.944,49	29,26	98,72

6 Análise dos Resultados

O problema de escalonamento de veículos de múltiplas garagens infelizmente não pode ser resolvido através de metodos tradicionais e exatos pois sua complexidade é muito grande. Ao crescermos um número consideravel a quantidade de viagens, o *solver* demorará muito mais tempo, fazendo com que seja necessário o desenvolvimento de meta-heurísticas para resolver em um tempo muito menor os problemas, porém de uma maneira não ótima.

Por outro lado, a meta-heurística implementada não ofereceu resultados próximos do valor ótimo. Em contraste a isto, contudo, o tempo de execução da mesma é muito menor que o tempo de execução do *solver*, e o fato de que conseguimos soluções factíveis para todas as instâncias do problema mostram mais uma vantagem no uso da mesma.

7 Conclusões

Podemos ver que mesmo não achando a solução ótima, a meta-heurística é muito mais usada na prática, pois não temos tempo para deixar o *solver* processando toda a instância. Por outro lado, a meta-heurística implementada oferece resultados que, em média, são muito maiores que os valores mínimos conhecidos

para o problema, mas sempre ofereceu um resultado factível para o mesmo.

Referências

[1] Andrew Makhorin. Gnu linear programming kit, 01 2005.

[2] Nemanja Milovanovic. *Solving the multiple-depot vehicle scheduling problem.* PhD thesis, Erasmus University Rotterdam, 07 2015.

[3] Supachai Pathumnakul. Solving multi depot vehicle routing problem for iowa recycled paper by tabu search heuristic. Master's thesis, Iowa State University, 1996.