Otimização do transporte heterogêneo de cargas: modelagem e meta-heurística

Davi J. F. Alves
Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo
Avenida Cesare Mansueto Giulio Lattes, n. 1201. Eugênio de Mello, São José dos Campos, São Paulo, Brasil. CEP: 12247-014
davi.juliano@unifesp.br

Roberval G. M. Filho
Instituto Tecnológico de Aeronautica
Praça Marechal Eduardo Gomes, n. 50. São José dos Campos, SãoPaulo, Brasil. CEP 12228-900
roberval1994@live.com

Tiago R. Macedo
Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo
Avenida Cesare Mansueto Giulio Lattes, n. 1201. Eugênio de Mello, São José dos Campos, São Paulo, Brasil. CEP: 12247-014
tmacedo@unifesp.br

Luiz Leduíno de S. Neto
Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo
Avenida Cesare Mansueto Giulio Lattes, n. 1201. Eugênio de Mello, São José dos Campos, São Paulo, Brasil. CEP: 12247-014
luiz.leduino@unifesp.br

Horacio H. Yanasse
Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo
Avenida Cesare Mansueto Giulio Lattes, n. 1201. Eugênio de Mello, São José dos Campos, São Paulo, Brasil. CEP: 12247-014
horacio.yanasse@unifesp.br

Mateus de A. C. Lima
Optime Rotas
Av. Floriano Peixoto, n. 1615. Uberlândia, Minas Gerais, Brasil. CEP: 38400-700
mateusdeassislima@gmail.com

RESUMO

Neste trabalho, os autores comparam três meta-heurísticas, *Variable Neighborhood Search*, *Greedy Randomized Adaptive Search Procedure* e *Simulated Annealing*, com um modelo criado para resolver um problema de roteamento de veículos heterogêneos. O problema analisado é uma variante do problema dos caixeiros viajantes múltiplo, proposto por uma empresa e visando aplicações práticas. Nessa variante, além da determinação de quais caixeiros (veículos) visitam quais cidades (pontos de entrega de produtos), também é necessário distribuir cargas de produtos

https://proceedings.science/p/157593?lang=pt-br

dentre os veículos levando em conta restrições de volume e peso, uma vez que os veículos são heterogêneos. Os algoritmos usados foram implementados usando a linguagem Python, o modelo foi executado no solver comercial Gurobi, utilizando da biblioteca `gurobipy`, e o resultados apresentados foram baseados em 16 instâncias criadas pelos autores.

PALAVRAS CHAVE. Modelagem matemática, Meta-heurísticas, Roteamento de veículos

ABSTRACT

In this work, the authors compare three meta-heuristics, *Variable Neighborhood Search*, *Greedy Randomized Adaptive Search Procedure* and *Simulated Annealing*, with an exact model created to resolve a heterogeneous vehicle routing problem. This problem is a variant of the multiple traveling salesman problems, proposed by a company and aimed towards practical applications. In this variant, not only the assignment of which salesman (vehicle) visits which city (products delivery point), but to assign which product goes to which vehicle is also necessary. The latter assignment takes into account the volume and weight restrictions that the vehicles have since these vehicles are heterogeneous. The authors implemented the algorithms in Python, executed the model using the commercial solver Gurobi, using the library `gurobipy`, and the results were based on 16 instances created by them.

KEYWORDS. Mathematical modeling, Optimization, Vehicle routing, Multiple traveling salesman problem, Heuristics

https://proceedings.science/p/157593?lang=pt-br

1. Introdução e descrição do problema

O objetivo principal desse artigo é descrever um método para otimizar entregas de produtos desde pontos de armazenamento e distribuição até seus consumidores. Para isso, nós apresentaremos métodos que otimizam conjuntamente a distribuição de carga dentro de veículos e as rotas de cada um desses veículos de entrega. Assim, a função objetivo a ser minimizada será dada pelo custo total de deslocamento de todos os veículos necessários para realizar as entregas de todos os produtos armazenados.

O problema abordado nesse artigo é interessante, desde o ponto de vista da Pesquisa Operacional, devido a suas aplicações na diminuição de custos de empresas de distribuição e entrega de produtos. Esse interesse se reflete na literatura científica. Por exemplo, em um artigo recente, Rajaei, Moslehi e Reisi-Nafchi trataram de um problema similar, no qual são consideradas diferentes restrições de carregamento e roteamento de veículos (ver o artigo Rajaei et al. [2022] e a excelente revisão bibliográfica contida nele). Além dos diversos artigos citados por Rajaei, Moslehi e Reisi-Nafchi, outros artigos recentes que tratam desse problema são Meliani et al. [2021]; Song et al. [2020]; Koch et al. [2018]. Com o recente crescimento do *e-commerce* e os recentes acontecimentos em escala global, é natural esperar que o interesse por esse problema venha a aumentar nos próximos anos.

A seguir, vamos descrever o problema abordado nesse artigo de maneira mais detalhada. Como mencionado acima, nosso objetivo principal é descrever um método para otimizar entregas de produtos desde pontos de armazenamento e distribuição até seus consumidores. Assim, nossos dados iniciais são listas de: produtos, contendo seus respectivos códigos, locais de origem e pesos; pontos de armazenamento e distribuição (que são os locais de origem dos produtos), contendo seus respectivos códigos, latitudes e longitudes; veículos de entrega, contendo suas respectivas placas, lotações, latitude, longitude e custo; e pedidos, contendo seus respectivos códigos, produtos, locais de origem e destinatários. A partir desses dados iniciais, nós queremos obter uma lista de veículos a serem utilizados, juntamente com uma lista de produtos carregados em cada um desses veículos e com a ordem de entrega de cada um dos produtos, de forma que o custo total dessas entregas seja o menor possível.

Observe que esse problema pode ser dividido em dois: um problema de roteamento de veículos e um problema de carregamento de produtos. O primeiro desses problemas está relacionado com o famoso problema do caixeiro viajante. Já o segundo desses problemas está relacionado com problemas de empacotamento de volumes. Como ambos os problemas são NP-difíceis, algumas das restrições consideradas nesse processo de resolução foram a escalabilidade e a factibilidade computacional. Ou seja, nós esperamos que o método proposto seja realmente capaz de ser implementado em larga escala por empresas. Por isso, nós comparamos os resultados de três métodos heurísticos para resolver esse problema. Para comparar e validar os resultados desses métodos heurísticos, nós desenvolvemos um método exato, que é utilizado como *benchmark*.

Os resultados obtidos dão informações interessantes sobre os métodos analisados, principalmente para as suas aplicações práticas desse problema complexo (de roteamento de veículos heterogêneos). Nas devidas circunstâncias, para instâncias menores, o modelo exato foi eficiente e retornou alguns resultado ótimo. Entretanto, para instâncias maiores, em um tempo extremamente reduzido, os métodos heurísticos obtiveram soluções com rapidez muito maior que o modelo exato.

2. Metodologia

Como mencionado na Seção 1, o problema tratado nesse artigo pode ser dividido em dois: um problema de roteamento de veículos e um problema de carregamento de produtos. Como ambos

os problemas são NP-difíceis e nosso objetivo é obter um método com boa escalabilidade e factibilidade computacional, nós comparamos os resultados de três métodos heurísticos para resolver esse problema. Além disso, nós desenvolvemos um modelo exato para comparar e validar os resultados desses métodos heurísticos.

A seguir, vamos descrever tanto os métodos heurísticos como o modelo exato utilizados para resolver e validar os resultados.

2.1. Métodos heurísticos

Nesse artigo, vamos comparar o desempenho de três meta-herísticas na resolução do problema de carregamento e roteamento de veículos: *Simulated Annealing*, *Greedy Randomized Adaptive Search Procedure* e *Variable Neighborhood Search*. Em cada uma dessas meta-heurísticas são utilizadas ainda heurísticas construtivas, para construir soluções iniciais, e métodos de busca local, para melhorar as soluções encontradas.

Nessa seção, vamos começar descrevendo as três meta-heurísticas que serão comparadas. Em seguida, vamos descrever as heurísticas construtivas e, por fim, os métodos de busca local utilizados em cada uma dessas três meta-heurísticas.

2.1.1. Meta-heurísticas

Meta-heurísticas são métodos utilizados para acelerar a obtenção de resultados cuja solução exata demanda muito tempo ou capacidade computacional. Em contra-partida, as meta-heurísticas entregam resultados que não são necessariamente ótimos. As três meta-heurísticas comparadas nesse artigo são: *Simulated Annealing*, *Greedy Randomized Adaptive Search Procedure* e *Variable Neighborhood Search*. Vamos descrever cada uma delas em detalhes.

***Simulated annealing* (SA):** O algoritmo de SA é inspirado no processo de térmico de recozimento de materiais (ver Kirkpatrick et al. [1987]). O principal objetivo desse processo é alterar a rigidez dos materiais. Analogamente, os algoritmos de *simulated annealing* tem como objetivo flexibilizar a procura por mínimos, admitindo possíveis pioras na função objetivo. A principal motivação para essa flexibilização é permitir escapar de ótimos locais que os algoritmos de busca local não permitem (ver Subsubseção 2.1.3).

Observe, no entanto, que a SA também usa uma heurística de busca local. No nosso caso, nós escolhemos a heurística conhecida como *variable neighborhood descent*. Além disso, a SA também depende de uma solução inicial, que nós obtivemos através da heurística construtiva do vizinho mais próximo. Por fim, a SA depende da escolha de quatro parâmetros, denotados por T_0, $T_{\min}$, $i_{\max}$ e α. Os parâmetros escolhidos foram os seguintes : $T_0 = 100.000$, $T_{\min} = 0,01$, $i_{\max} = 4.000$ e $\alpha = 0,9$, justificados pela recorrência na literatura.

Uma descrição explícita do algoritmo da SA utilizado para minimizar uma certa função f está contida no Algoritmo 1.

***Greedy randomized adaptive search procedure* (GRASP):** O algoritmo da GRASP consiste, basicamente, de duas fases: a fase de construção de uma solução e a fase de otimização da solução encontrada (ver Feo e Resende [1995]; Resende e Ribeiro [2019]). Na fase de construção, uma solução inicial viável é obtida através de um método iterativo, guloso e adaptativo. Já na fase de otimização, uma busca local é empregada para minimizar a função objetivo. Essa busca local é baseada na solução inicial obtida na fase de construção. Observe que, como uma nova solução inicial é obtida em cada iteração, essa é uma meta-heurística *multistart*.

O método utilizado para construir as sucessivas soluções iniciais foi a heurística aleatória gulosa. Já o método de busca local utilizado na fase de otimização foi o *variable neighborhood*

Algoritmo 1: Algoritmo da *simulated annealing*

Entrada: x_0 é uma solução inicial.
$x_{\min} = x_0$;
$T = T_0$;
enquanto $T > T_{min}$ **faça**
  enquanto $i < i_{\max}$ **faça**
    $x' = \text{VND}(x_{\min})$;
    $\Delta = f(x') - f(x_{\min})$;
    se $\Delta \leq 0$ **então**
      se $f(x') < f(x_{\min})$ **então**
        $x_{\min} = x'$;
      fim
    fim
    senão
      escolha $r \in [0, 1]$ aleatoriamente;
      se $r < e^{\frac{-\Delta}{T}}$ **então**
        $x_{\min} = x'$;
      fim
    fim
    $i = i + 1$;
  fim
  $T = T \cdot \alpha$
fim
Saída : $f(x_{\min})$ é o valor mínimo da função f.

descent. Além disso, o algoritmo da GRASP depende da escolha de dois parâmetros iniciais: $i_{\max}$ e α. Nós escolhemos $i_{\max} = 5000$ e $\alpha = 0,45$.

Uma descrição explícita do algoritmo da GRASP utilizado para minimizar uma certa função f está contida no Algoritmo 2.

Algoritmo 2: Algoritmo da *greedy randomized adaptive search procedure*

$x_{\min} =$ Construção Randômica Gulosa $(\vec{x} = (), \alpha)$;
$\vec{x} = (x_{\min})$;
enquanto $i < i_{\max}$ **faça**
  $x_0 =$ Construção Randômica Gulosa $(\vec{x}, \alpha)$;
  $x' = \text{VND}(x_0)$;
  se $f(x') < f(x_{\min})$ **então**
    $x_{\min} = x'$;
  fim
  $\vec{x} = (\vec{x}, x')$;
  $i = i + 1$;
fim
Saída: $f(x_{\min})$ é o valor mínimo da função f.

***Variable neighborhood Search* (VNS):** Essa é uma família de meta-heurísticas que busca repetidas vezes por soluções ótimas em vizinhanças de uma certa solução e, conforme essa solução melhora, a busca por soluções ótimas se reinicia nas vizinhanças da solução melhorada (Mladenović e Hansen [1997]). As escolhas de vizinhanças podem ser determinísticas, estocásticas ou ambas. Nesse artigo, nós fizemos a escolha determinística. Nesse caso, a VNS é chamada de *variable neighborhood descent* (VND).

A VND que nós implementamos utiliza quatro heurísticas de busca local e também é a utilizada nas meta-heurísticas SA e GRASP. Ela depende da construção de uma solução inicial e de um parâmetro. Nós construímos a solução inicial com a heurística construtiva do vizinho mais próximo e escolhemos o parâmetro i_{max} como sendo igual a 12.

Uma descrição explícita do algoritmo dessa meta-heurística para encontrar o ponto de mínimo de uma certa função f está contida no Algoritmo 3. As heurísticas de busca local utilizadas na meta-heurística do VND foram: *swap* inter-rotas, *swap* intra-rotas, 2-opt e 3-opt.

2.1.2. Heurísticas construtivas

Como descrito na Subsubseção 2.1.1, em cada uma das meta-heurísticas utilizadas para a busca do menor custo para a distribuição de produtos, nós empregamos heurísticas construtivas para obter soluções iniciais do nosso problema. Nessa seção, vamos descrever as duas heurísticas construtivas empregadas por nós: a do vizinho mais próximo e a aleatória gulosa.

Os dados iniciais de ambas essas heurísticas são os seguintes. Um conjunto C formado pelos pontos de armazenamento (ou distribuição) e entrega de produtos. O ponto de armazenamento, que é de onde os veículos saem e para onde eles voltam, será denotado por 0. Depois, um conjunto V formado pelos veículos que podem ser usados nas entregas dos produtos. Para cada par de pontos $i, j \in C$, a distância $d_{i,j}$ entre os pontos i e j (em quilômetros, por exemplo). Além disso, para cada $v \in V$, o custo c_v do veículo v (em reais por quilômetros, por exemplo). Para cada $i \in C$, o peso p_i (em quilos, por exemplo) e o volume w_i (em metros cúbicos, por exemplo) da entrega a

Algoritmo 3: Algoritmo da *variable neighborhood descent*

Entrada: x_0 é uma solução inicial.
$x_{\min} = x_0$
para $BL =$ *cada uma das heurísticas de buscas locais* **faça**
  enquanto $i \leq i_{\max}$ **faça**
    $x' = BL(x_{\min})$;
    $i = i + 1$;
    se $f(x') \leq f(x_{\min})$ **então**
      $x_{\min} = x'$;
      $i = 1$;
    fim
  fim
fim
Saída : $x_{\min}$ é um ponto de mínimo da função f.

ser realizada no ponto i. Por fim, para cada $v \in V$, o peso máximo P_v (em quilos, por exemplo) e o volume máximo W_v (em metros cúbicos, por exemplo) que o veículo v consegue carregar.

Heurística construtiva do vizinho mais próximo: Nós usamos essa heurística nas meta-heurísticas SA e VNS. Ela é uma heurística gulosa que procura sempre pelo veículo com menor custo para realizar a entrega mais próxima. Uma descrição explícita do algoritmo dessa heurística está contida no Algoritmo 4.

Heurística construtiva aleatória gulosa Essa heurística construtiva, que é usada no GRASP, busca pelo veículo de menor custo para realizar a próxima entrega (assim como a heurística construtiva do vizinho mais próximo). No entanto (ao contrário da heurística construtiva do vizinho mais próximo), o ponto escolhido para a próxima entrega é um ponto aleatório dentro dos pontos que satisfazem a seguinte desigualdade:

$$d_{i,j} \leq \min\{d_j \mid j \in C'\} + \alpha \left(\max\{d_j \mid j \in C'\} - \min\{d_j \mid j \in C'\}\right),$$

onde C' é formado pelos pontos de C que ainda não receberam entregas, i é o último ponto onde o veículo realizou sua entrega, e α é um parâmetro. Baseados na literatura e visando diversificação, nós escolhemos $\alpha = 0,45$. Uma descrição explícita do algoritmo dessa heurística está contida no Algoritmo 5.

2.1.3. Heurísticas de busca local

As heurísticas de busca local são utilizadas em todas as meta-heurísticas para otimizar (ou intensificar) as soluções encontradas (ver Subsubseção 2.1.1). Para buscar o menor custo para a distribuição de produtos, nós empregamos quatro heurísticas de busca local: *swap* inter-rotas, *swap* intra-rotas (ver Figura 1), 2-opt (ver Croes [1958]) e 3-opt (ver Lin [1965]).

2.2. Modelo exato

Nessa seção, vamos descrever a formulação de um modelo usado para resolver o problema de forma exata, a fim de estabelecer um *benchmark* para comparar e validar os resultados dos métodos heurísticos descritos acima. Esse modelo é uma adaptação do modelo do caixeiro viajante

Algoritmo 4: Algoritmo da heurística construtiva do vizinho mais próximo

para $v \in V$ **faça**
  $\vec{x}_v = (0)$;
fim
$C = C \setminus \{0\}$;
enquanto $C \neq \emptyset$ **faça**
  v é o veículo em V cuja função objetivo é mínima;
  $C' = \{j \in C \mid p_j < P_v \text{ e } w_j < W_v\}$;
  enquanto $C' \neq \emptyset$ **faça**
    i é a última entrada de $\vec{x}_v$;
    j é um ponto em C' tal que a distância $d_{i,j}$ é mínima;
    $\vec{x}_v = (\vec{x}_v, j)$;
    $P_v = P_v - p_j$;
    $W_v = W_v - w_j$;
    $C = C \setminus \{j\}$;
    $C' = \{j \in C \mid p_j < P_v \text{ e } w_j < W_v\}$;
  fim
fim
para $v \in V$ **faça**
  $\vec{x}_v = (\vec{x}_v, 0)$;
fim
Saída: $\{\vec{x}_v\}_{v \in V}$.

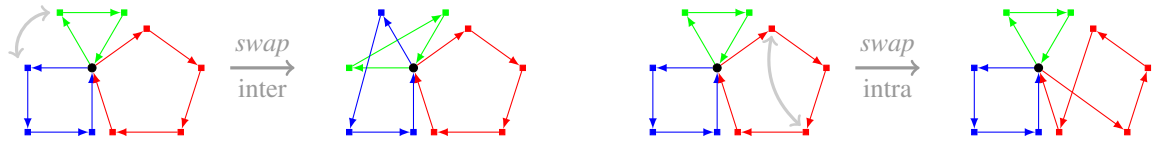


Figura 1: Exemplos de iterações da *swap* inter-rotas e *swap* intra-rotas

múltiplo contido em Christofides et al. [1981], com a adição das restrições de empacotamento (de peso e volume para cada veículo).

Vamos começar descrevendo os parâmetros do nosso modelo. Primeiro, considere um conjunto C formado pelos pontos de armazenamento (ou distribuição) e entrega de produtos. O ponto de armazenamento, que é de onde os veículos saem e para onde eles voltam, será denotado por 0. Depois, considere um conjunto V formado pelos veículos que podem ser usados nas entregas dos produtos. Para cada par de pontos $i, j \in C$, a distância entre os pontos i e j (em quilômetros, por exemplo) será denotada por $d_{i,j}$. Além disso, para cada $v \in V$, o custo do veículo v (em reais por quilômetros, por exemplo) será denotado por c_v. Para cada $i \in C$, o peso (em quilos, por exemplo) e o volume (em metros cúbicos, por exemplo) da entrega a ser realizada no ponto i serão denotados por p_i e w_i, respectivamente. Por fim, para cada $v \in V$, o peso máximo (em quilos, por exemplo) e o volume máximo (em metros cúbicos, por exemplo) que o veículo v consegue carregar serão denotados por P_v e W_v, respectivamente.

Agora vamos descrever as variáveis do nosso modelo. Para cada par de pontos $i, j \in C$ e cada veículo $v \in V$, considere uma variável de decisão $x_{i,j,v} \in \{0, 1\}$, onde: $x_{i,j,v} = 1$ quando o veículo v realizar uma entrega que passa sucessivamente pelos pontos i e j (nesta ordem, sem passar

Algoritmo 5: Algoritmo da heurística aleatória gulosa

para $v \in V$ **faça**
  $\vec{x}_v = (0)$;
fim
$C = C \setminus \{0\}$;
enquanto $C \neq \emptyset$ **faça**
  v é o veículo em V cujo custo c_v é mínimo;
  $C' = \{j \in C \mid p_j < P_v \text{ e } w_j < W_v\}$;
  enquanto $C' \neq \emptyset$ **faça**
    i é a última entrada de $\vec{x}_v$;
    j é um ponto aleatório em C' tal que a distância $d_{i,j}$ satisfaz:

$$d_{i,j} \leq \min\{d_j \mid j \in C'\} + \alpha \left(\max\{d_j \mid j \in C'\} - \min\{d_j \mid j \in C'\}\right).$$

    $\vec{x}_v = (\vec{x}_v, j)$;
    $P_v = P_v - p_j$;
    $W_v = W_v - w_j$;
    $C = C \setminus \{j\}$;
    $C' = \{j \in C \mid p_j < P_v \text{ e } w_j < W_v\}$;
  fim
fim
para $v \in V$ **faça**
  $\vec{x}_v = (\vec{x}_v, 0)$;
fim
Saída: $\{\vec{x}_v\}_{v \in V}$.

por nenhum outro ponto intermediário); e $x_{i,j,v} = 0$, caso contrário. (Em particular, $x_{i,i,v} = 0$ para todo $i \in C$ e $v \in V$.) Além disso, para cada ponto $i \in C \setminus \{0\}$, considere uma variável de decisão $u_i \in \{0, 1, 2, \ldots, |C| - 1\}$, onde: $u_i = k$ quando i for o k-ésimo ponto por onde o veículo irá passar depois de sair do ponto de armazenamento. (Em particular, $u_0 = 0$.)

O objetivo desse nosso modelo é minimizar a soma dos gastos necessários para os veículos disponíveis realizarem as entregas de todos os produtos. Ou seja, nosso objetivo é minimizar a função

$$\sum_{i \in C} \sum_{j \in C} \sum_{v \in V} c_v d_{i,j} x_{i,j,v}.$$

Já as restrições que devem ser satisfeitas são as seguintes:

- Restrições que garantem que cada ponto de $C \setminus \{0\}$ esteja conectado por um veículo a exatamente dois outros pontos, um deles representa de onde o veículo vem antes de chegar e o outro representa para onde o veículo vai depois de sair deste ponto:

$$\sum_{v \in V} \sum_{i \in C \setminus \{0\}} x_{i,j,v} = 1, \quad \text{para cada } j \in C \setminus \{0\}, \text{ e}$$

$$\sum_{i \in C \setminus \{0\}} (x_{i,j,v} - x_{j,i,v}) = 0, \quad \text{para cada } j \in C \setminus \{0\} \text{ e } v \in V.$$

  Observe que essas restrições excluem a possibilidade de um trajeto passar duas vezes por um mesmo ponto (exceto se esse ponto for a origem). No entanto, nos casos em que os parâmetros $\{d_{i,j}\}_{i,j \in C}$ representam distâncias (métricas), essas restrições não excluem possíveis mínimos da função objetivo, pois $d_{i,k} \leq d_{i,j} + d_{j,k}$ para todos $i, j, k \in C$; ou seja, ir de um ponto a outro passando por um terceiro ponto é mais caro que ir direto.

- Restrições que garantem que cada veículo de V saia e chegue no máximo uma vez do ponto de armazenamento:

$$\sum_{j \in C \setminus \{0\}} x_{0,j,v} \leq 1 \quad \text{e} \quad \sum_{i \in C \setminus \{0\}} x_{i,0,v} \leq 1, \quad \text{para cada } v \in V.$$

  Observe que essas restrições não excluem a possibilidade que um veículo deixe de ser utilizado.

- Restrições que garantem que trajeto de cada veículo seja formado por um único ciclo:

$$u_i - u_j + |C| x_{i,j,v} \leq |C| - 1 \qquad \text{para cada } i \in C,\ j \in C \setminus \{0\} \text{ e } v \in V.$$

  Observe que essa restrição pode ser dividida em dois casos: quando $x_{i,j,v} = 0$ e quando $x_{i,j,v} = 1$. No caso em que $x_{i,j,v} = 0$, ela se torna $u_i - u_j \leq |C| - 1$. Essas restrições são satisfeitas trivialmente pela hipótese de que $u_i \in \{0, 1, 2, \ldots, |C| - 1\}$ para todo $i \in C$. No caso em que $x_{i,j,v} = 1$, ela se torna $u_i - u_j \leq -1$. Isso significa que, se o ponto j sucede o ponto i imediatamente, então $u_i + 1 \leq u_j$.

- Restrições que garantem que cada veículo em V é capaz de carregar a soma dos pesos das entregas designadas a ele:

$$\sum_{i \in C} \sum_{j \in C \setminus \{0\}} p_j x_{i,j,v} \leq P_v \quad \text{e} \quad \sum_{i \in C \setminus \{0\}} \sum_{j \in C} p_i x_{i,j,v} \leq P_v \qquad \text{para cada } v \in V.$$

- Restrições que garantem que cada veículo em V é capaz de carregar a soma dos volumes das entregas designadas a ele:

$$\sum_{i \in C} \sum_{j \in C \setminus \{0\}} w_j x_{i,j,v} \leq W_v \quad \text{e} \quad \sum_{i \in C \setminus \{0\}} \sum_{j \in C} w_i x_{i,j,v} \leq W_v \qquad \text{para cada } v \in V.$$

O modelo descrito acima foi implementado em `Python`, usando a biblioteca `gurobipy`. Esse modelo foi utilizado para estabelecer um *benchmark* para validar os resultados obtidos pelos métodos heurísticos.

3. Resultados e conclusões

Nesta seção serão apresentados resultados alcançados através da execução do modelo exato e das meta-heurísticas em 16 instâncias sintéticas. Essas instâncias foram construídas combinando alguns subconjuntos de pontos de entrega de produtos (contendo 20, 30, 40 e 50 pontos, respectivamente) e subconjuntos de veículos (contendo 4, 5, 7 e 9 veículos, respectivamente). Lembre que cada ponto possui uma localização (determinada pela sua latitude e longitude) e uma lista de produtos a serem entregues (com a respectiva demanda de peso e volume). Além disso, cada veículo tem uma capacidade de transporte (peso e volume), um custo de deslocamento e uma origem.

As meta-heurísticas descritas na Subsubseção 2.1.1 foram executadas 30 vezes para cada instância, em um *notebook Dell*, com o sistema operacional *Windows 10*, 16GB de memória *RAM*, processador *Intel core* i7 de 10^a geração, e no ambiente de desenvolvimento *Jupyter Notebook*. Já a implementação do modelo exato foi feita em um *notebook Dell* utilizando o sistema operacional *Windows 10*, com um processador *Intel core* i5 de 8^a geração, 8GB de memória *RAM*, e no ambiente de desenvolvimento *Jupyter Notebook*. A Tabela 1 e a Figura 2 apresentam os resultados obtidos pelas abordagens exata e aproximativa bem como o tempo demandado em cada solução:

	Simulated Annealing			GRASP			VNS			Resultados exatos		
	Melhor FO	Média FO	Tempo	Melhor FO	Média FO	Tempo	Melhor FO	Média FO	Tempo	FO	Tempo	GAP
1-1	825,29	996,81	9,71	674,54	718,37	16,11	654,90	685,47	27,30	433,40	46,56	00%
1-2	836,66	1146,10	10,33	769,01	797,12	21,09	708,57	753,99	32,27	433,40	265,19	00%
1-3	1115,28	1288,24	12,40	908,52	959,77	19,78	898,93	925,70	28,27	433,40	1202,21	08%
1-4	1093,65	1342,23	14,71	1004,19	1100,35	13,12	1018,44	1073,73	23,80	456,41	1202,23	17%
2-1	821,35	1128,93	21,56	754,03	789,54	39,85	753,21	790,50	67,00	561,38	1202,15	17%
2-2	1090,00	1267,23	22,04	860,10	898,33	36,02	804,38	872,83	68,59	549,08	1202,53	19%
2-3	1166,64	1497,49	20,25	1033,78	1080,29	33,30	1007,15	1059,95	62,94	580,13	1200,86	30%
2-4	1361,35	1678,58	21,83	1219,21	1272,03	31,97	1160,11	1258,03	58,29	633,47	1200,40	39%
3-1	1153,08	1294,92	45,72	850,17	936,84	66,09	857,22	907,19	113,99	714,11	1201,26	18%
3-2	1173,29	1424,28	35,60	980,29	1030,86	61,18	926,18	991,31	122,64	716,24	1202,07	21%
3-3	1496,32	1703,32	34,65	1126,11	1201,78	52,53	1151,47	1199,99	90,74	774,34	1203,81	32%
3-4	1689,54	1931,51	31,45	1327,82	1374,88	48,75	1357,79	1400,48	84,86	793,51	1207,49	37%
4-1	1164,31	1299,60	58,46	961,33	1027,85	112,06	913,22	962,46	210,22		Inviável	
4-2	1207,80	1478,67	65,97	1039,70	1115,38	109,50	971,51	1069,45	179,56	984,87	1201,95	40%
4-3	1456,17	1760,29	60,76	1220,03	1295,33	90,06	1176,10	1256,31	175,52	904,88	1205,45	39%
4-4	1685,91	2044,02	55,90	1402,53	1473,19	81,88	1411,06	1471,57	380,52	864,54	1201,92	39%

Tabela 1: Tabela de resultados das melhores soluções e das soluções médias

Observe que nas menores instâncias (1-1 e 1-2), o modelo exato obteve um resultado dentro do tempo estipulado (20 minutos). Isso indica que, nesses casos, ou seja, não mais de vinte pontos de entrega e não mais de cinco veículos, o método exato é a melhor alternativa para resolver esse problema. Em todas as outras instâncias, o modelo exato não foi capaz de atingir um resultado ótimo dentro do intervalo de tempo estipulado. De fato, nesses casos, as meta-heurísticas obtiveram resultados razoavelmente próximos aos resultados obtidos pelo modelo em um intervalo de tempo muito menor (sempre menor que 3 minutos e 30 segundos, cerca de um sétimo de 20 minutos).

https://proceedings.science/p/157593?lang=pt-br

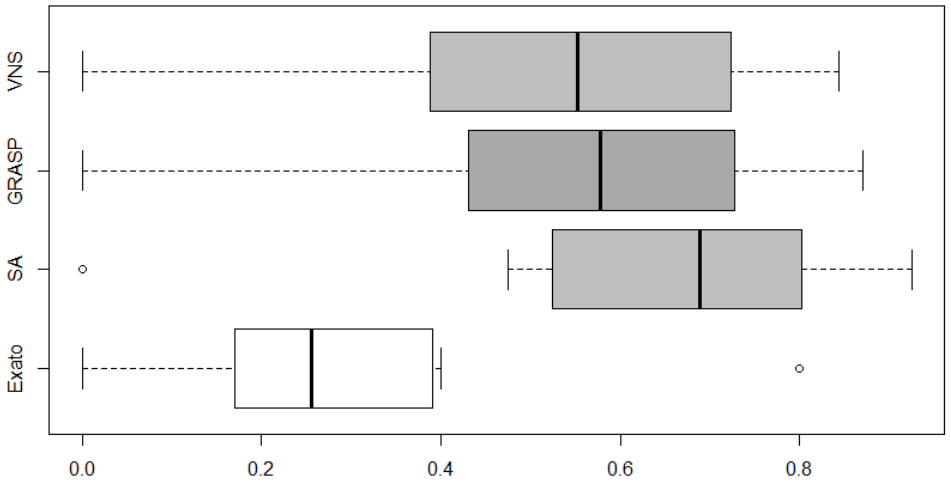


Figura 2: *Boxplot* do GAP percentual em cada meta-heurística e no modelo exato

Observe ainda que, pelo resultado do modelo exato, a instância 4-1 não é viável. De fato, nessa instância, a capacidade total dos veículos não é suficiente para carregar todos os produtos a serem entregues. No entanto, as meta-heurísticas ainda são capazes de obter soluções para essa instância. Isso ocorre por que a restrição de realizar todas as entregas não é incluída nas heurísticas construtivas (ver Subsubseção 2.1.2). Assim, os resultados das meta-heurísticas realizam entregas parciais dos produtos.

É interessante observar que, apesar da entrega parcial dos produtos não ser uma solução completa para nosso problema inicial, na prática, essa resposta é melhor do que a informação de que a instância em questão é inviável. De fato, para a empresa que propôs o problema e acompanhou o seu desenvolvimento, esse é um resultado adequado para ser colocado em prática.

Comparando o resultado obtido pelo modelo exato com os resultados obtidos pelas meta-heurísticas, vemos que os resultados obtidos pelo modelo são melhores, apesar da quantidade de tempo de processamento ser muito maior. Dentre as três meta-heurísticas utilizadas, aquelas que apresentam as melhores soluções são a VNS e a GRASP, enquanto a SA é aquela que obtém soluções com maior rapidez.

Alguns pontos que nós gostaríamos de melhorar em trabalhos futuros são: os resultados das meta-heurísticas desenvolvidas, testando e utilizando outros parâmetros; adicionar dimensões de altura, largura e profundidade aos produtos e suas respectivas restrições no modelo exato; adicionar janelas de tempo para cada veículo, limitando o trabalho de cada um; adicionar a possibilidade dos produtos advirem de diversas origens (pontos de armazenamento).

Agradecimentos

Os autores gostariam de agradecer a empresa OPTIME por ter proposto o problema, fornecido as instâncias e acompanhado todo o desenvolvimento desse projeto. Eles gostariam de agradecer também aos professores Luiz Leduíno Neto e Horácio Yanasse por diversas sugestões para melhorar o modelo e as heurísticas.

Referências

Christofides, N., Mingozzi, A., e Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Math. Programming*, 20(3):255–282. ISSN 0025-5610. URL https://doi.org/10.1007/BF01589353.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6): 791–812. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/167074.

Feo, T. A. e Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *J. Global Optim.*, 6(2):109–133. ISSN 0925-5001. URL https://doi.org/10.1007/BF01096763.

Kirkpatrick, S., Gelatt, C., e Vecchi, M. (1987). Optimization by simulated annealing. In Fischler, M. A. e Firschein, O., editors, *Readings in Computer Vision*, p. 606–615. Morgan Kaufmann, San Francisco (CA). ISBN 978-0-08-051581-6. URL https://www.sciencedirect.com/science/article/pii/B9780080515816500593.

Koch, H., Bortfeldt, A., e Wäscher, G. (2018). A hybrid algorithm for the vehicle routing problem with backhauls, time windows and three-dimensional loading constraints. *OR Spectrum*, 40(4):1029–1075. ISSN 0171-6468. URL https://doi.org/10.1007/s00291-018-0506-6.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Tech. J.*, 44:2245–2269. ISSN 0005-8580. URL https://doi.org/10.1002/j.1538-7305.1965.tb04146.x.

Meliani, Y., Hani, Y., Elhaq, S. L., e Mhamedi, A. E. (2021). Vehicle routing problem with three-dimensional loading constraints: Experimentations and evaluation. *IFAC-PapersOnLine*, 54 (1):104–109. ISSN 2405-8963. URL https://www.sciencedirect.com/science/article/pii/S2405896321008090. 17th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2021.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100. ISSN 0305-0548. URL https://www.sciencedirect.com/science/article/pii/S0305054897000312.

Rajaei, M., Moslehi, G., e Reisi-Nafchi, M. (2022). The split heterogeneous vehicle routing problem with three-dimensional loading constraints on a large scale. *European J. Oper. Res.*, 299(2):706–721. ISSN 0377-2217. URL https://doi.org/10.1016/j.ejor.2021.08.025.

Resende, M. G. e Ribeiro, C. C. (2019). Greedy randomized adaptive search procedures: advances and extensions. In *Handbook of metaheuristics*, p. 169–220. Springer.

Song, X., Jones, D., Asgari, N., e Pigden, T. (2020). Multi-objective vehicle routing and loading with time window constraints: a real-life application. *Ann. Oper. Res.*, 291(1-2):799–825. ISSN 0254-5330. URL https://doi.org/10.1007/s10479-019-03205-2.