Problema do Caixeiro Viajante Tridimensional com restrição de Ordem*

Pedro Hokama[1], Bruno L. P. de Azevedo[1], and Flávio K. Miyazawa[1]

Instituto de Computação
Universidade Estadual de Campinas
13084-971, Campinas, SP, Brasil
pedro.hokama@students.ic.unicamp.br, bruno.pires@students.ic.unicamp.br,
fkm@ic.unicamp.br

Abstract In this paper, we present a problem consisting of a combination of the Travelling Salesman Problem (TSP) and the Three-dimensional Strip Packing Problem, denoted by 3L-TSP. In this problem, a vehicle loaded with boxes must depart from a certain point and deliver these to a set of customers. Each customer demands a set of boxes and the goal is to minimize the total travelling cost. Unloading is done through a single side of the container and items from an unloading customer must not be blocked by items to be delivered later.
We present an exact algorithm for this problem and then we apply its structure to obtain heuristic solutions that offer a compromise between the travelling cost and the length of the container. The presented approach showed to be suitable to solve small and medium sized instances.

1 Introdução

Consideramos um problema que combina os problemas do Caixeiro Viajante (TSP) e do Carregamento em Contêiner, que chamamos por Problema do Caixeiro Viajante Tridimensional (3L-TSP). Neste problema, um veículo deve partir carregado de uma origem e entregar caixas em pontos pré-definidos para seus clientes. Cada cliente tem um conjunto de caixas que deve receber e o objetivo é minimizar o custo de deslocamento do veículo. As caixas devem ser retiradas a partir do fundo do contêiner do veículo e a remoção das caixas de um cliente não podem ser obstruidas pelas caixas a serem descarregadas posteriormente. Assim, diferentes rotas exigem a geração de diferentes configurações de empacotamento.

Apresentamos um algoritmo *branch-and-cut* que minimiza o comprimento total da rota utilizada pelo veículo e usamos a estrutura deste algoritmo para obter soluções heurísticas que apresentam soluções com compromisso entre o custo do deslocamento e o tamanho do contêiner obtido pela solução heurística.

Este problema generaliza o problema do Caixeiro Viajante (*Traveling Salesman Problem* TSP) e o problema de Empacotamento Tridimensional (*Three-dimensional Strip Packing Problem* 3SP), problemas conhecidos serem NP-difíceis.

* Esta pesquisa teve o apoio financeiro do CNPq e da CAPES.

Para mais informações sobre estes problema, veja [2] para o problema TSP e [14,15] para o problema 3SP.

A investigação de problemas que combinam problemas de rotas com restrições de empacotamento é bastante recente. No caso onde pode se usar K veículos, cada um com contêiner de mesma capacidade, e o objetivo é minimizar o comprimento total das rotas, temos o chamado 3L-CVRP (*Three-dimensional Loading Capacitated Vehicle Routing Problem*) e sua versão bidimensional por 2L-CVRP. O problema 2L-CVRP foi investigado sob a abordagem exata por Iori et al. [10] e na abordagem heurística por Gendreau et al. [6], Zachariadis et al.[16] e Fuellerer et al. [3]. O problema 3L-CVRP foi investigado pela primeira vez em 2006 por Gendreal et al. [5], que apresentam uma heurística de busca tabu. Recentemente, Fuellerer et al. [4] apresentam metaheurísticas baseadas em colônia de formigas.

Para o nosso conhecimento, não existe na literatura um algoritmo exato para o problema 3L-TSP nem para o problema 3L-CVRP. Além disso, exploramos este problema sob dois critérios, pela minimização de rotas e pela minimização do contêiner utilizado. A abordagem que apresentamos é particularmente interessante nas aplicações onde o conjunto de localidades a serem visitadas pelo veículo não é grande.

Apresentamos duas abordagens, uma exata onde são obtidos rotas de custo mínimo, e outra heurística que apesar de não necessariamente obter rotas de custo mínimo, pode tratar instâncias com maior número de caixas e clientes.

Na Seção 2, definimos formalmente o problema e apresentamos a formulação e desigualdes usadas pelo algoritmo *branch-and-cut* proposto. Na Seção 3, apresentamos a adaptação do algoritmo exato e das heurísticas do problema de empacotamento tridimensional para que respeitem a ordem do empacotamento. Na Seção 4, apresentamos os resultados computacionais obtidos. Por fim, na Seção 5, apresentamos as conclusões e trabalhos futuros.

2 Definição e formulação do problema

Para definir este problema, usaremos o espaço $\mathbb{R}^3$ com coordenadas xyz. Denotamos uma caixa b_i como uma tripla $b_i = (x_i, y_i, z_i)$, onde x_i, y_i e z_i são respectivamente *largura*, *altura* e *comprimento* (profundidade). O mesmo pode ser feito para o contêiner $B = (W, H, D)$. Primeiro, definimos o empacotamento de caixas dentro de um contêiner. Um empacotamento de uma lista de caixas $L = (b_1, \ldots, b_n)$ no contêiner $B = (W, H, D)$ é dado por uma função $\mathcal{P} : L \to [0, W) \times [0, H) \times [0, D)$, tal que

1. Nenhum item passa dos limites do contêiner. Isto é,

$$\mathcal{P}^x(b_i) + x(b_i) \leq W, \quad \mathcal{P}^y(b_i) + y(b_i) \leq H \quad \text{e} \quad \mathcal{P}^z(b_i) + z(b_i) \leq D,$$

   onde $\mathcal{P}(b_i) = (\mathcal{P}^x(b_i), \mathcal{P}^y(b_i), \mathcal{P}^z(b_i))$, $i = 1, \ldots, n$.
2. Dois itens não podem se sobrepor. Assim, se $\mathcal{R}(b_i)$ é definido como

$$\mathcal{R}(b_i) = [\mathcal{P}^x(b_i), \mathcal{P}^x(b_i) + x_i) \times [\mathcal{P}^y(b_i), \mathcal{P}^y(b_i) + y_i) \times [\mathcal{P}^z(b_i), \mathcal{P}^z(b_i) + z_i),$$

então

$$\mathcal{R}(b_i) \cap \mathcal{R}(b_j) = \emptyset \quad \forall i, j,\ 1 \leq i \neq j \leq n.$$

Para representar empacotamentos que respeitam a ordem de entrega das caixas em uma certa rota, estendemos a definição de empacotamento tridimensional de [14] para considerar também restrições de ordem: Dado uma sequência de listas $\mathcal{L} = (L_1, L_2, \ldots, L_k)$ dizemos que um empacotamento $\mathcal{P}$ de $L = L_1 \cup \ldots \cup L_k$ respeita a sequência de $\mathcal{L}$ na direção z se para toda caixa $b \in L_i$, temos que

$$\mathcal{R}^z(b) \cap \mathcal{R}(c) = \emptyset \quad \text{para toda caixa} \quad c \in L_j \quad \text{e} \quad j > i,$$

onde $\mathcal{R}^z(b) = [\mathcal{P}^x(b_i), \mathcal{P}^x(b_i) + x_i) \times [\mathcal{P}^y(b_i), \mathcal{P}^y(b_i) + y_i) \times [\mathcal{P}^z(b_i), D)$. Isto é, se $b \in L_i$, não há nenhuma caixa c na frente de b que pertence a um conjunto L_j que será descarregado depois.

Uma instância do problema 3L-TSP é dada por: *(i)* um contêiner de dimensões $B = (W, H, D)$, *(ii)* um grafo $G = (V, E, c)$ com conjunto de vértices $V = \{0, 1, \ldots, n\}$, arestas E, custo não negativo c_e para cada aresta $e \in E$ e *(iii)* conjuntos de caixas L_v para cada vértice $v \in V$. O objetivo é obter um circuito hamiltoniano $C = (0, v_1, \ldots, v_n)$ de custo total mínimo e um empacotamento $\mathcal{P}$ de todas as caixas em B que respeite a ordem de $\mathcal{L} = (L_{v_1}, \ldots, L_{v_n})$. A Figura 1 exemplifica, para o caso bidimensional, um empacotamento que respeita ordem para uma determinada rota.

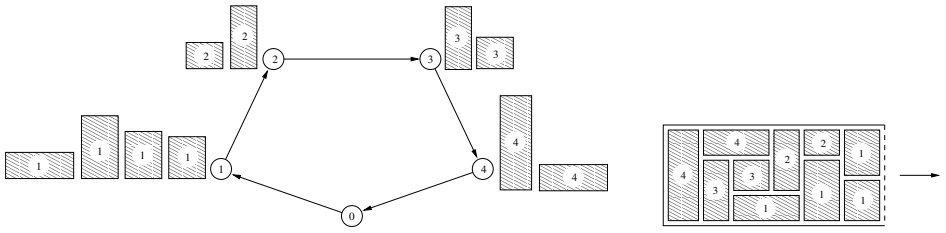


Figura 1. Exemplo de rota com os itens de cada cliente e um empacotamento que respeita a ordem da rota.

Usando a notação acima, denotaremos o conjunto de instâncias $\mathcal{I}$ do problema 3L-TSP como o conjunto das tuplas (V, E, c, W, H, D, L), onde $G = (V, E, c)$ é o grafo de entrada, $B = (W, H, D)$ são as dimensões do contêiner do veículo, L_v é uma lista de caixas para cada vértice $v \in V$ e o depósito será sempre dado pelo vértice 0. Dado uma instância $I \in \mathcal{I}$, denotaremos por $\mathcal{R}_I$ o conjunto das rotas de I que tem empacotamentos que respeitam ordem.

2.1 Formulação do 3L-TSP e abordagem Branch-and-Cut

A formulação em programação linear inteira do problema 3L-TSP, que utilizamos na abordagem *branch-and-cut*, é uma adaptação da formulação do problema

TSP, acrescido de restrições que satisfaçam empacotamentos que respeitam ordem. Dada instância $I = (V, E, c, W, H, D, L) \in \mathcal{I}$ o problema 3L-TSP pode ser formulado como:

$$\text{minimize} \sum_{e \in E} c_e x_e$$

$$\text{sujeito a}$$

$$\sum_{e \in \delta(v)} x_e = 2 \qquad \forall v \in V\,, \tag{1}$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V, \quad S \neq \emptyset\,, \tag{2}$$

$$\sum_{e \in R} x_e \leq n - 1 \qquad \forall R \notin \mathcal{R}_I\,, \tag{3}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E\,. \tag{4}$$

As restrições (4) fazem as variáveis x_e para cada $e \in E$ serem binárias e indicam a pertinencia de uma aresta na solução. As restrições (1) e (2) são as restrições básicas da formulação do TSP e garantem que a solução será um circuito hamiltoniano do grafo de entrada. As restrições em (3) restringem o conjunto de rotas apenas para as que podem ter empacotamentos com ordem. Apesar do número de restrições em (2) e (3) ser exponencial, o método *branch-and-cut* insere apenas aquelas necessárias para a resolução. Para isso, alimentamos o algoritmo de *branch-and-bound* com rotinas de separação, que inserem uma restrição violada sempre que detectam que um ponto viola alguma das restrições da formulação. Além disso, outras rotinas de separação podem ser utilizadas, desde que as novas restrições não removam nenhuma solução viável do problema.

Como rotinas de separação das restrições (2), utilizamos cortes de conexidade, obtidas a partir de cortes mínimos. Inicialmente, executamos o procedimento da árvore de Gomory-Hu [7,9], que nos dá todos os cortes mínimos entre vértices através de $n - 1$ chamadas para o algoritmo de corte mínimo (em vez de se chamar para cada um dos $\binom{n}{2}$ pares possíveis de vértices). Com isso, inserimos todos os cortes violados da árvore. Este processo é combinado com buscas de corte mínimo simples e repetido até que todas as restrições em (2) estejam satisfeitas. Posteriormente, utilizamos uma heurística que procura por restrições conhecidas como *comb inequalities* [8,2], desenvolvidas para o TSP. Se a solução ainda é fracionária e não foram encontradas desigualdades de corte ou do tipo *comb* violadas, o algoritmo faz uma enumeração padrão nas variáveis (*branch*) e repete o processo em cada um dos problemas resultantes. Tal enumeração nos dá uma árvore (de *branch-and-cut*). Sempre que um nó inteiro (rota) for encontrado na árvore, executamos as rotinas de separação das restrições (4). Para isso, aplicamos inicialmente as heurísticas e caso nenhuma delas consiga obter um empacotamento que satisfaça a ordem da rota, executamos um algoritmo exato. Caso não haja empacotamento viável que respeite a ordem da rota, a correspondente desigualdade em (4) é inserida e o processo continua.

As rotinas de separação para encontrar cortes mínimos e cortes de Gomory Hu, foram adaptadas a partir do código disponibilizado pelo ZIB [17]. As heurísticas de separação para encontrar as *comb inequalities* foram adaptadas do código disponibilizado por Lysgaard [12]. Como rotinas de separação para encontrar empacotamentos com ordem, implementamos a heurística de George e Robinson [11] e uma heurística híbrida por níveis. A rotina para obter empacotamentos exatos que respeitam ordem, foi adaptado a partir do código disponibilizado por Martello et al. [13]. A adaptação destas heurísticas e o algoritmo exato estão descritos na próxima seção.

3 Problema do Empacotamento Tridimensional com Ordem

Nesta seção apresentamos o algoritmo exato e as heurísticas utilizadas para obter empacotamentos que respeitam a ordem dada por uma rota.

3.1 Algoritmo Exato

O algoritmo apresentado por Martello et al. [13], gera um empacotamento com características para ser realizado por robôs. Neste caso, uma caixa só pode ser colocada a direita, acima ou na frente de outra caixa, e nunca a esquerda, abaixo ou atrás de outra. O algoritmo usa a estratégia *branch-and-bound* e enumera o espaço de busca através das posições onde uma caixa pode ser colocada. Dado um empacotamento parcial $\mathcal{Q}$, as posições de pontos $\rho(\mathcal{Q})$, que são candidatas a receber uma nova caixa são dadas por uma rotina, denominada *3D-Corners*. Tais pontos foram denominados de *pontos de canto*. A complexidade computacional deste algoritmo é $O(m^2)$, onde m é o número de caixas em $\mathcal{Q}$.

O algoritmo exato, denominado por *OneBin* aprensentado em [13], usa a abordagem *branch-and-bound* explorando os pontos de canto do empacotamento. As caixas são consideradas em uma ordem específica, definida pela entrada e a busca é feita de maneira recursiva. No início, nenhuma caixa está empacotada e portanto $\rho(\emptyset) = (0, 0, 0)$. A cada chamada da rotina, são computados os pontos de canto e o volume do empacotamento parcial. Cada item não empacotado é atribuido para cada ponto de canto encontrado por *3D-Corners* e *OneBin* é chamado recursivamente, desde que o empacotamento de um item não viole a restrição de ordem. Quando nenhum dos itens puder ser empacotado, então é feito um *backtracking*. Adaptamos esse algoritmo de forma que sempre que uma caixa atribuida a um ponto bloqueia outra já empacotada, também é feito um *backtracking*. Pelos experimentos realizados, optamos pela ordenação das caixas de maneira decrescente de cliente, desempatando com prioridade para caixas de maior volume, uma vez que obteve um melhor desempenho do algoritmo *OneBin*.

Para agilizar o processo de enumeração, o ramo de uma chamada com empacotamento $\mathcal{Q}$, contendo uma lista de caixas Q, pode ser podado, caso não seja considerado promissor. Denote por V^* o maior volume obtido por um empacotamento parcial da enumeração, V_B o volume do contêiner, V_Q o volume total

das caixas em Q, $V_{\mathcal{Q}}$ o volume do envelope de $\mathcal{Q}$. Se a desigualdade

$$V_Q + (V_B - V_{\mathcal{Q}})) \leq V^*$$

é válida, então é feito um *backtracking* pois mesmo que todo o volume restante do contêiner seja preenchido nós não melhoramos o valor de V^*.

Uma outra poda pode ser feita da seguinte maneira: Seja $\mathcal{Q}$ um empacotamento e $S_1, \ldots, S_k$ uma discretização da superfície do envelope de $\mathcal{Q}$ em pedaços retangulares minimais, de maneira que cada superfície S_i esteja contida na face de uma caixa a ser entregue em um único cliente, digamos da posição $\sigma(S_i)$. Com isso, a região definida entre a superfície e a saída do contêiner só pode receber caixas que estão nos clientes de posição $j \leq \sigma(S_i)$. Com isso, se não pudermos preencher estas regiões com o volumes das caixas restantes, podemos também fazer um *backtracking*. Este limitante inferior não foi utilizado na obtenção dos resultados computacionais.

O algoritmo exato também pode funcionar como uma heurística, restringindo-se o tempo de sua execução. Assim, denominamos por MPV o algoritmo exato e por MPV T, o algoritmo exato limitado a uma execução de T segundos.

3.2 Heurísticas

Uma das heurísticas utilizadas, que denominamos por GR, é uma adaptação da heurística desenvolvida por George e Robinson [11] que utiliza uma abordagem de *Wall-Building* (construção de parede). Esta heurística gera um empacotamento construindo camadas (paredes) ao longo do contêiner e espaços não utilizados em uma camada são guardados em uma pilha de espaços para possível uso nas camadas seguintes. Em cada camada, o algoritmo gera uma coluna de caixas iguais empacotadas a partir do início da camada. Tal coluna gera três novos espaços (a frente, do lado e acima), que são colocados a disposição na pilha. Se um determinado espaço não pode receber nenhuma caixa, este é adicionado no conjunto de espaços rejeitados.

A cada novo espaço S que retiramos da pilha, busca-se no conjunto de espaços rejeitados, espaços que sejam adjacentes na parte posterior e que possam ser fundidos a S, desde que não haja perdas em nenhuma das dimensões do espaço atual. Cada caixa recebe uma prioridade definida de modo decrescente em $\min\{x_b, y_b, z_b\}$ onde (x_b, y_b, z_b) são as dimensões da caixa. A caixa de maior prioridade define a profundidade da camada. Para cada espaço a ser preenchido, escolhemos a caixa de maior prioridade que pode ocupar aquele espaço. Geramos então uma coluna com o maior numero de caixas iguais possível. Para mais detalhes sobre a heurística GR, veja [11].

Para a obtenção de empacotamentos com ordem, restringimos a ordem das caixas de maneira que todas as caixas de um cliente são empacotadas após as caixas dos clientes anteriores. Com isso, nenhuma caixa empacotada será bloqueada por outra colocada posteriormente.

A segunda heurística, é baseada no algoritmo híbrido HFF (*Hibrid First Fit*) para o empacotamento de placas retangulares. A heurística, que denominamos

HFF3, também empacota as caixas por níveis, porém cada nível só recebe caixas de um cliente e todas as caixas de um nível estão empacotadas na mesma profundidade. Para empacotar as caixas de um cliente, o algoritmo HFF3 ordena as caixas em ordem decrescente de profundidade e gera o primeiro nível com a mesma profundidade do comprimento da primeira caixa. Em seguida, empacota o maior número de caixas neste nível, seguindo a ordenação obtida, através do algoritmo HFF. I.e., dando preferência para as caixas mais profundas. Para gerar os próximos níveis, este processo é repetido para as caixas restantes, até que todas as caixas tenham sido empacotadas. Para a obtenção de empacotamentos com ordem, o empacotamento das caixas de um cliente são realizadas após o empacotamento das caixas dos clientes anteriores. Como os níveis de cada cliente da rota são dispostos em sequência dentro do contêiner, o último nível do $(i+1)$-ésimo cliente a ser atendido na rota também podem receber caixas do i-ésimo cliente. Para isto, as caixas são inseridas neste último nível, sempre seguindo a ordem decrescente de altura, sem violar a altura do nível.

4 Experimentos computacionais

Inicialmente apresentamos o desempenho do algoritmo *branch-and-cut* usando as heurísticas e o algoritmo exato como geradores de planos de corte e obtenção dos empacotamentos viáveis. Em seguida, analisamos o desempenho do algoritmo quando contrapomos o tamanho do contêiner com o comprimento da rota.

Realizamos testes em várias instâncias geradas computacionalmente na comparação dos algoritmos. O número de clientes variou entre 7 e 20 e o número de caixas consideradas foi de 9 a 25. O contêiner considerado foi o padrão *Dry Box* de 20 pés de dimensões $(8, 8, 20)$ (em pés). Cada dimensão das caixas varia entre $[2, 5]$. Geramos instâncias razoavelmente fáceis para se empacotar, onde o volume dos itens não passa de 60% do volume do contêiner, denominada de Classe E, e instâncias difíceis, onde o volume das caixas é de pelo menos 95% do volume do contêiner, denominada de Classe H. Os experimentos foram executados em um único *Core* em um Intel Quad Core 2.40 GHz. O resolvedor de programação linear inteira escolhido foi o Xpress-Optimizer ® v19.00.17. Foi definido um tempo limite de 3600s para a execução de cada instância.

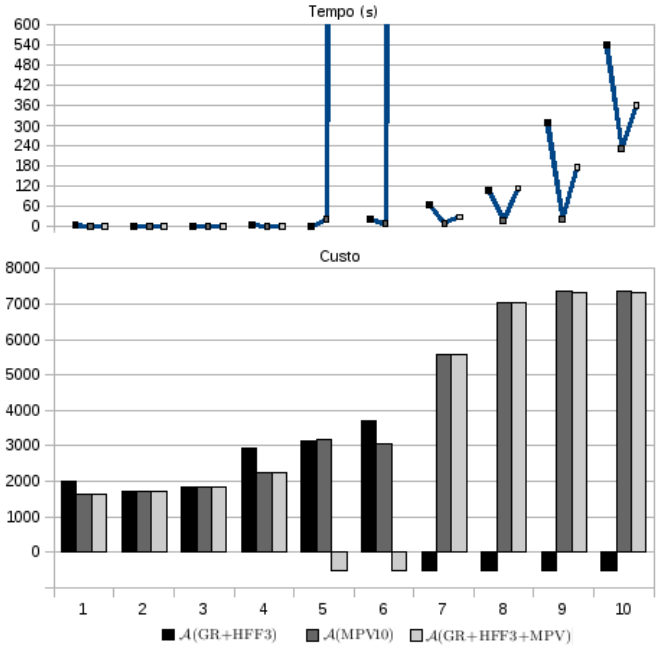
Cada uma das heurísticas foi executada tanto para a rota original encontrada quanto para a respectiva rota inversa. Dependendo das rotinas usadas para a obtenção dos cortes de empacotamento, o algoritmo *branch-and-cut* obtém resultados com tempo e qualidade distintos. Assim, denotaremos por $\mathcal{A}(\mathcal{R})$ o algoritmo *branch-and-cut* usando a(s) rotina(s) de empacotamento $\mathcal{R}$ para gerar cortes de empacotamento.

Na Tabela 1, apresentamos um comparativo das heurísticas e do algoritmo exato na abordagem proposta. As colunas da tabela contém, da esquerda para a direita, as seguintes informações: Classe da instância (*Classe*), número de clientes (*#Clientes*), número total de caixas (*#Caixas*), custo da rota obtida por $\mathcal{A}$(GR+ HFF3) e seu tempo de processamento, custo da rota obtida por $\mathcal{A}$(MPV10) (MPV10 é igual ao algoritmo MPV, limitado a 10 segundos, por chamada) e

seu tempo computacional e custo da rota obtida pelo algoritmo exato $\mathcal{A}$(GR + HFF3 + MPV) e seu tempo computacional. No gráfico apresentado acima da tabela 1 custos negativos representam instâncias onde não foi encontrado solução viável.

Apesar de terem um bom desempenho na obtenção de empacotamentos da classe E, as heurísticas GR e HFF3 não obtiveram soluções viáveis para os empacotamentos mais complexos da classe H. Isto fez com que o algoritmo *branch-and-bound* percorresse toda a árvore de enumeração, levando a um grande tempo de processamento para se certificar que de fato não há nenhum ramo onde estas heurísticas conseguem obter empacotamento viável.

Já o algoritmo *branch-and-cut* usando a heurística MPV com limitação de tempo, MPV10, foi mais robusta para restrições de ordem, obtendo soluções viáveis para todas as instâncias, com custo pouco acima do ótimo.



Id	*Classe*	*#Clientes*	*#Caixas*	$\mathcal{A}$(GR+HFF3)		$\mathcal{A}$(MPV10)		$\mathcal{A}$(GR+HFF3+MPV)	
				Custo	Tempo	Custo	Tempo	Custo	Tempo
1	E	7	9	2013,51	6s	1644,36	0,16s	1644,36	0,11s
2	E	7	10	1709,65	0,10s	1709,65	0,10s	1709,65	0,11s
3	E	7	20	1827,16	0,14s	1827,16	0,10s	1827,16	0,11s
4	E	8	15	2919,95	7s	2226,04	0,12s	2226,04	0,10s
5	E	10	20	3157,46	0,14s	3159,01	21s	-	3600s
6	E	10	25	3704,99	21s	3065,97	10s	-	3600s
7	H	10	20	-	66s	5575,10	11s	5556,53	29s
8	H	15	20	-	108s	7048,80	19s	7030,24	115s
9	H	20	20	-	308s	7360,54	24s	7310,42	177s
10	H	20	25	-	540s	7366,18	231s	7310,42	360s

Tabela 1. Comparação das soluções heurísticas e exata.

Por ser uma heurística rápida com desempenho razoavelmente bom, analisamos o desempenho do algoritmo $\mathcal{A}$(GR) contrapondo duas funções objetivo: uma que minimiza o custo da rota e outra que minimiza o tamanho do contêiner. A Tabela 2, apresenta o comportamento deste algoritmo para contêiners com largura e altura fixos e o comprimento D variando de 107 a 200, para uma instância de 30 clientes e um total de 120 caixas.

D	$\mathcal{A}$(GR)	Tempo
200	14244,49	0,66s
150	14244,49	0,35s
114	14244,49	50s
113	14352,86	127s
111	14449,47	110s
110	14449,47	83s
109	14524,58	90s
108	24947,90	316s
107	-	406s

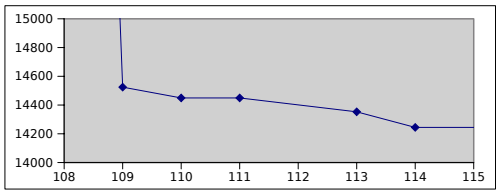


Tabela 2. Desempenho do algoritmo $\mathcal{A}$(GR) contrapondo o comprimento do contêiner com comprimento total das rotas obtidas.

A estratégia se mostrou viável para tratar instâncias grandes. Mesmo para instâncias com até 300 caixas no total, o programa obteve soluções em poucos segundos.

É possível notar que a medida que o tamanho do contêiner diminui, o número de empacotamentos possíveis de serem gerados pela heurística GR que satisfazem a restrição de ordem também diminui, levando muitas vezes a rotas de custo elevado. Por exemplo, na tabela 2, para se usar um contêiner de comprimento 108 (o menor possível), o melhor custo de deslocamento obtido pelo algoritmo $\mathcal{A}$(GR) foi de quase 25000. Isto representa um gasto de deslocamento de quase 72% a mais que a rota usada para um contêiner de tamanho 109.

5 Conclusões e Trabalhos Futuros

Neste artigo investigamos o problema 3L-TSP que combina o problema do caixeiro viajante com o problema de empacotamento tridimensional. Apresentamos um algoritmo exato e heurísticas baseadas na abordagem *branch-and-cut*. Este foi o primeiro algoritmo exato para este problema.

O desempenho das heurísticas também puderam ser analisadas contrapondo duas funções objetivo: uma que minimiza o custo da rota e outra que minimiza o tamanho do contêiner. Por ser rápida, as heurísticas de empacotamento puderem obter soluções viáveis em pouco tempo, e com isso, um usuário pode optar por usar um veículo menor, mas percorrendo uma distância maior, ou um veículo maior, percorrendo menor distância. Certamente uma importante informação na tomada de decisões.

Para melhorar o desempenho do algoritmo apresentado, iremos implementar outras rotinas de separação, como as usadas em [1] e desenvolver outras heurísticas. Neste trabalho, não utilizamos metaheurísticas nem heurísticas primais para a construção de soluções viáveis para o 3L-TSP. O desenvolvimento

de tais heurísticas também será foco de nossa atenção, que além de serem pontos interessantes de investigação, poderão agilizar bastante a busca por soluções ótimas. Nossa intenção é tratar outras condições práticas, como a geração de empacotamentos que respeitam condições de estabilidade e balanceamento; e variantes onde as caixas de um cliente i devem ser carregadas em um ponto de origem o_i e entregues em um ponto de destino d_i.

Referências

1. D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. Concorde tsp solver, 2004. `www.tsp.gatech.edu`.
2. D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A computational study.* Princeton Press, 2006.
3. G. Fuellerer, K. F. Doerner, R. F. Hartl, and M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers & Operations Research*, 36:655–673, 2009.
4. G. Fuellerer, K. F. Doerner, R. F. Hartl, and M. Iori. Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, To appear.
5. M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40:342–350, 2006.
6. M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1):4–18, 2008.
7. R. E. Gomory and T. C. Hu. Multi-terminal network flows. *SIAM Journal on Computing*, 9(4):551–570, 1961.
8. M. Grötschel and M. W. Padberg. On the symmetric travelling salesman problem i: Inequalities. *Mathematical Programming*, 16:265–280, 1979.
9. D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.
10. Salazar-González J. Iori M and Vigo D. An exact approach for the vehicle routing problem with two-dimensional loading constrains. *Transportation Science*, 41(2):253–264, 2007.
11. D. F. Robinson J. A. George. A heuristic for packing boxes into a container. *Computer and Operations Research 7*, 4:147–156, 1980.
12. Lechtford A. Lysgaard J. and Eglese R. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2003.
13. S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.
14. F. K. Miyazawa and Y. Wakabayashi. An algorithm for the three-dimensional packing problem with asymptotic performance analysis. *Algorithmica*, 18(1):122–144, 1997.
15. F. K. Miyazawa and Y. Wakabayashi. Three-dimensional packings with rotations. *Computers and Operations Research*, 36:2801–2815, 2009.
16. Tarantilis C. Zachariadis E. and Kiranoudis C. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195(3):729–743, 2009.
17. Zib-Berlin. Mathprog: A collection of codes for solving various mathematical programming problems. `http://elib.zib.de/pub/Packages/mathprog/index.html`.