Problema do Caixeiro Viajante com Coleta e Entrega de Objetos com Base Retangular

Fabrício C. Machado
Instituto de Matemática e Estatística - USP, 05508-090, São Paulo-SP, Brasil
fabcm1@gmail.com

Thiago A. de Queiroz
Unidade de Matemática e Tecnologia - UFG/Regional Catalão, 75704-020, Catalão-GO, Brasil
taq@ufg.br

Maurício G. C. Resende
Amazon.com, Mathematical Optimization and Planning (MOP), Seattle, WA 98109 USA
resendem@amazon.com

Reinaldo Morabito
Departamento de Engenharia de Produção - UFSCar, 13565-905, São Carlos-SP, Brasil
morabito@ufscar.br

Flávio K. Miyazawa
Instituto de Computação - UNICAMP, 13083-852, Campinas-SP, Brasil
fkm@ic.unicamp.br

RESUMO

Este trabalho propõe duas abordagens de resolução para o Problema do Caixeiro Viajante com Coleta e Entrega, em que os objetos têm forma retangular. Uma solução para este problema consiste em um Ciclo Hamiltoniano de custo mínimo, em que um vértice de coleta deve ser visitado antes do seu respectivo vértice de entrega e os itens são empacotados ortogonalmente e sem sobreposição na superfície retangular do veículo, além de respeitar a sua capacidade máxima de carga. Propusemos um modelo de programação inteira, resolvido por um algoritmo do tipo *branch-and-cut*, com desigualdades de eliminação de subciclos, satisfazer a precedência e restrições de empacotamento. O problema de empacotamento é resolvido por um algoritmo de programação por restrições. A outra abordagem é baseada no algoritmo genético de chaves aleatórias viciadas, que usa um vetor de chaves para indicar um ciclo. Experimentos computacionais realizados em instâncias da literatura validam as abordagens propostas.

PALAVRAS CHAVE. Problema do Caixeiro Viajante com Coleta e Entrega, Programação Inteira, Algoritmo Genético de Chaves Aleatórias Viciadas.

Área Principal: Otimização Combinatória.

ABSTRACT

This work proposes two approaches to solve the Traveling Salesman Problem with Pickup and Delivery, and with two-dimensional rectangular shaped items This problem requests for a minimum Hamiltonian tour where a pickup vertex must be visited prior to its delivery vertex, and the rectangular items are arranged without overlapping, respecting the vehicle surface dimensions and the vehicle weight capacity. We propose an integer formulation solved by a branch-and-cut algorithm with inequalities on subtour elimination, precedence, and loading constraints. The two-dimensional packing problem is solved by a constraint programming algorithm, which uses a vector of keys to indicate tour sequence. Computational tests over instances from the literature indicate for a fast convergence of the approaches proposed.

KEYWORDS. Traveling Salesman Problem with Pickup and Delivery, Integer Programming, Biased Random-Key Genetic Algorithm.

Main Area: Combinatorial Optimization.

1. Introdução

Problemas que envolvem a roteirização de veículos aparecem frequentemente no ramo logístico. Um desses é o clássico problema do Caixeiro Viajante, em que é preciso visitar uma sequência de clientes a custo mínimo, em que cada cliente é visitado exatamente uma vez. Este problema requer que o veículo retorne ao cliente de origem, resultando em um ciclo, conhecido como ciclo Hamiltoniano (Johnson e McGeoch, 1997).

Uma parte comum a problemas de roteamento é a que envolve a consideração de restrições práticas. Existem diferentes restrições com o propósito de agregar valor operacional, por outro lado, impondo dificuldades durante a fase de modelagem e, até mesmo, durante a resolução do problema como um todo. Bischoff e Ratcliff (1995) elencaram um conjunto de restrições práticas quando o propósito é observar o problema de empacotamento, incluindo restrições de estabilidade de carga, empilhamento e fragilidade dos itens, organização dos itens em classes, ordem para a entrega, etc. Por outro lado, Lahyani et al. (2015) prioriza as restrições para problemas de roteamento de veículos, com destaque para as questões de janela de tempo, coleta e entrega, frota de veículos heterogênea, entre outras.

A proposta deste trabalho é investigar o problema do Problema do Caixeiro Viajante na presença da restrição de Coleta e Entrega, porém os objetos a serem transportados são itens de base retangular, como é o caso da coleta e distribuição de mercadorias organizadas em paletes, ou peças frágeis de museu. A restrição de Coleta e Entrega impõe que a cada ponto de coleta há um respectivo de entrega, onde itens são recolhidos do ponto de coleta e organizados sem sobreposição na superfície retangular do veículo para, depois, serem entregues no respectivo ponto de entrega. Da forma como mencionado, não é obrigatório que todos os pontos de coleta sejam visitados primeiramente para, em seguida, fazer a visita aos pontos de entrega. Por outro lado, é requerido que os itens daqueles pontos de coleta visitados sejam empacotados de forma viável no veículo, ao mesmo tempo que respeitam a sua capacidade máxima de carga. O objetivo deste problema é obter um ciclo Hamiltoniano de custo mínimo que satisfaça a restrição de coleta e entrega, considerando o empacotamento de itens bidimensionais.

O Problema do Caixeiro Viajante, do inglês *Traveling Salesman Problem - TSP*, é um dos problemas bastante investigado na área de otimização combinatória, com aplicações reais. Ele é aparentemente simples, que tem várias aplicações reais, porém está na classe NP-Difícil (Garey e Johnson, 1979). Por isso, a literatura tem buscado por diferentes estratégias de resolução, tanto exatas como heurísticas.

Em particular, o problema do Caixeiro Viajante com a restrição de Coleta e Entrega foi resolvido em: Kalantari et al. (1985), que propuseram um algoritmo do tipo *branch-and-bound*; Renaud et al. (2000), por uma heurística de duas fases, uma de construção e outra de melhoria; Renaud et al. (2002), que desenvolveram heurísticas de perturbação sobre buscas locais; Hernández-Pérez e Salazar-González (2004), que apresentaram um modelo de programação inteira, resolvido por um algoritmo *branch-and-cut*; e, Dumitrescu et al. (2009), que resolveram as maiores instâncias do problema de forma exata, considerando até 35 vértices de coleta e entrega. Tais autores também fizeram um estudo do poliedro do problema, para o qual conseguiram obter e provar várias desigualdades que são facetas.

A versão do TSP aqui resolvida é diferente das versões tratadas na literatura, uma vez que considera os itens com forma retangular, gerando um subproblema de empacotamento ortogonal de itens bidimensionais. Este subproblema é NP-Difícil (Fekete et al., 2007) e tem sido resolvido de forma independente pela literatura. Fekete et al. (2007) propuseram uma abordagem baseada na resolução de um grafo de intervalos; Clautiaux et al. (2007) propuseram um algoritmo do tipo *branch-and-bound*; Clautiaux et al. (2008) desenvolveram um algoritmo do tipo *branch-and-bound* combinado com o uso da técnica de programação por restrição. Estes últimos resultados foram melhorados por Mesyagutov et al. (2012).

Um descrição formal do problema em estudo é dada na Seção 2, com a apresentação de

um modelo de programação inteira baseado em uma formulação por arestas. Também, é feito o estudo de desigualdades válidas que podem ser consideradas durante a resolução do modelo inteiro, com destaque para as desigualdades de eliminação de subciclos e precedência entre clientes de coleta e entrega. Visando comparar os resultados com uma heurística, a Seção 3 traz o Algoritmo Genético de Chaves Aleatórias Viciadas, que tem sido aplicado com sucesso na resolução de vários problemas de otimização, mas que ainda não tinha sido aplicado a esta variante do problema do Caixeiro Viajante. Os experimentos computacionais, realizados sobre instâncias da literatura, são apresentados na Seção 4, enquanto que conclusões e direções para trabalhos futuros estão na Seção 5.

2. Formulação Inteira

O Problema do Caixeiro Viajante com Coleta e Entrega, denotado por TSPPD, é definido da seguinte forma. Considere um conjunto de n pedidos, cada um composto por um ponto de coleta e um de entrega. Seja $P = \{s_1, \ldots, s_n\}$ o conjunto dos pontos de coleta e $D = \{t_1, \ldots, t_n\}$ o conjunto dos pontos de entrega. Vértices Oa e Ob representam a saída e a entrada para o depósito, respectivamente. O TSPPD é definido em um grafo completo e não direcionado $G = (V, E, c)$, em que $V = P \cup D \cup \{Oa, Ob\}$ é o conjunto de vértices, $E = \{(x,y) \mid x,y \in V, x \neq y\}$ é o conjunto de arestas e $c(e)$ denota o custo (não negativo) das arestas $e \in E$. O objetivo do TSPPD é encontrar um ciclo Hamiltoniano de custo total mínimo, começando no vértice Oa e terminando em Ob, sujeito à restrição de precedência de que cada vértice de coleta deve ser visitado antes de seu respectivo vértice de entrega.

Todavia, a versão do TSPPD estudada neste artigo considera ainda que cada pedido é composto por uma lista de itens bidimensionais. Assim, no ponto de coleta s_i, para $i = 1, \ldots, n$, existe uma lista R_i, onde cada item $r \in R_i$ tem dimensões retangulares (l_{ir}, c_{ir}), área a_{ir} e peso d_{ir}. Além disso, o veículo tem superfície retangular (L, C) e capacidade máxima de carga (peso) Q. Denotaremos este problema por 2L-TSPPD.

O objetivo do 2L-TSPPD é o mesmo do TSPPD, com a restrição adicional de que para cada caminho no ciclo envolvendo uma sequência de pontos de coleta, todos os itens coletados devem ser organizados ortogonalmente e sem sobreposição na superfície do veículo, além de respeitar a sua capacidade máxima de carga. Consequentemente, surge um subproblema de empacotamento ortogonal de itens bidimensionais.

Um modelo de programação inteira para o 2L-TSPPD é definido na formulação (1-7). Uma solução x corresponde ao vetor de incidência das arestas usadas no ciclo, de modo que serão usadas $|E|$ variáveis binárias, em que cada x_e corresponde à uma aresta $e \in E$. Dado $S \subseteq V$, seja $\delta(S) = \{(i,j) \in E \mid i \in S, j \notin S\}$. Se $S = \{i\}$, escreveremos $\delta(i)$ em vez de $\delta(\{i\})$ e dado $E' \subseteq E$, usaremos ainda a notação $x(E') = \sum_{e \in E'} x_e$ e $E[E']$ sendo o conjunto de arestas com ambas as extremidades em E'.

A função objetivo (1) visa obter um ciclo de custo total mínimo. A restrição (2) faz com que os vértices Oa e Ob apareçam consecutivamente no ciclo. Assim, o ciclo começa em Oa e termina em Ob. Observe que como essa aresta sempre é usada, ela não precisa ser considerada como uma variável real do modelo.

As restrições (3) são restrições de grau, de modo que todo vértice tenha grau 2. As restrições (4) são as restrições de eliminação de subciclos, que fazem o ciclo ser conexo.

$$\textit{minimizar} \sum_{e \in E} c(e) x_e \tag{1}$$

sujeito a:

$$x_{Oa,Ob} = 1 \tag{2}$$
$$x(\delta(i)) = 2 \qquad \forall i \in V \tag{3}$$
$$x(\delta(S)) \geq 2 \qquad \forall S \subseteq V,\ 3 \leq |S| \leq |V|/2 \tag{4}$$
$$x(\delta(S)) \geq 4 \qquad \forall S \in \mathcal{U} \tag{5}$$
$$\sum_{e \in E[T]} x_e \leq |E[T]| - 1 \qquad \forall T \in \mathcal{T} : |T| \geq 2 \tag{6}$$
$$x_e \in \{0,1\} \qquad \forall e \in E \tag{7}$$

As restrições (5) são as restrições de precedência, que garantem que o vértice s_i seja visitado antes do vértice t_i para todo $s_i \in P$, uma vez que o conjunto $\mathcal{U}$ é a coleção dos subconjuntos $S \subseteq V$ satisfazendo $3 \leq |S| \leq |V| - 2$, com $Oa \in S$, $Ob \notin S$ e tal que existe $s_i \in P$ com $s_i \notin S$ e $t_i \in S$.

Com relação ao subproblema de empacotamento, $\mathcal{T} \subset V$ contém todos os caminhos inviáveis com relação ao empacotamento, que iniciam em um ponto de coleta s_i, para $i \in P$, passam por nenhum, um ou mais pontos de coletas e entrega, e finalizam em um ponto de coleta t_j, tal que após t_j, o próximo ponto é de entrega. Note que cada caminho inviável $T \in \mathcal{T}$ precisa ter pelo menos dois pontos de coleta, uma vez que um caminho com apenas um ponto de coleta sempre é viável, ou seja, o conjunto de itens de qualquer ponto de coleta pode ser empacotado no veículo. Segue que as restrições (6) eliminam os caminhos que podem aparecer na solução que não possuem um empacotamento viável. As restrições (7) são as restrições de integralidade.

2.1. Rotinas de Separação

Observe que existe uma restrição (4) de subciclo para cada subconjunto de vértices (na desigualdade está escrito $|S| \leq |V|/2$, porque se $|S| > |V|/2$ violar a desigualdade, temos que $\bar{S} = V \backslash S$ também viola, já que $\delta(S) = \delta(\bar{S})$). Existe, portanto, um número exponencial de restrições, o que torna inviável que todas elas sejam consideradas logo no início da otimização.

Entretanto, o problema de separação destas desigualdades pode ser resolvido em tempo polinomial. Dada uma solução x, para saber se esta satisfaz todas as desigualdades (4), basta testar se existem pares de vértices u e v separados por um corte $\delta(S)$ conforme (8). Podemos verificar isto por meio de um algoritmo de fluxos máximos que encontre um uv-corte mínimo. Na realidade, é possível melhorar isto computando a árvore de cortes de Gomory e Hu. (1961).

$$x(\delta(S)) < 2, \qquad u \in S,\ v \in V \backslash S \tag{8}$$

A rotina de separação para a restrições de precedência (5) é obtida de forma análoga. É suficiente considerar um par de vértices (s_i, t_i), tal que para o conjunto S satisfazer $\mathcal{U}$, é preciso aplicar o algoritmo de fluxo máximo no grafo representado pela solução x', obtida de x. Então, os valores de $x_{(Oa,t_i)}$ e $x_{(Ob,s_i)}$ são substituídos por valores altos, pois eles não podem aparecer no corte mínimo.

Outras desigualdades utilizadas para resolver o 2L-TSPPD são discutidas adiante. A primeira delas é a desigualdade de Blossom, comumente utilizada na resolução do TSP (Letchford et al., 2004). Esta desigualdade é usada para evitar subciclos que podem aparecer na solução fracionária. Seja $H, T_1, ..., T_k$ subconjuntos de V, tal que:

- $|T_i| = 2$, $T_i \cap H \neq \emptyset$ e $T_i \cap (V \backslash H) \neq \emptyset$, $\forall i \in \{1, ..., k\}$.
- $T_1, ..., T_k$ são dois a dois disjuntos.
- k é ímpar e $k \geq 3$.

As desigualdades de Blossom são dadas em (9) e a rotina de separação é baseada em uma busca realizada no grafo de suporte criado a partir da solução fracionária e um pequeno valor de ε. Primeiro, a rotina computa as componentes do grafo aplicando uma busca em profundidade. Cada componente é candidata a ser um elemento do conjunto H e, então, a rotina procura pelos vértices de arestas em $\delta(H)$ satisfazendo $x_e \geq 1-\varepsilon$. Estas arestas vão formar os conjuntos T. Por fim, a desigualdade (9) é verificada somente se existe uma quantidade ímpar de conjuntos do tipo T.

$$x(\delta(H)) + \sum_{i=1}^{k} x(\delta(T_i)) \geq 3k+1 \tag{9}$$

As desigualdades π e σ a seguir foram utilizadas na resolução do TSPPD por Dumitrescu et al. (2009), de forma que consideramos a mesma abordagem gulosa e aleatória, proposta por tais autores, como rotina de separação. Seja $S \subseteq V\backslash\{Ob\}$, com $E_S^{\pi} = \{\{i,j\} \mid i \in S\backslash\pi(S)\ ,\ j \notin S \cup \pi(S) \cup \{Oa\}\}$. As desigualdades π são dadas em (10), tal que o último vértice de S visitado por um ciclo válido deve pertencer a $S\backslash\pi(S)$ e seu sucessor deve pertencer a $V\backslash(S \cup \pi(S) \cup \{Oa\})$.

$$x(E_S^{\pi}) \geq 1 \tag{10}$$

As desigualdades σ são definidas de forma análoga em (11), mas agora observando o primeiro vértice em S visitado por um ciclo válido. Este vértice deve pertencer a $S\backslash\sigma(S)$ e seu predecessor pertencer a $V\backslash(S \cup \sigma(S) \cup \{Ob\})$, em que $E_S^{\sigma} = \{\{i,j\} \mid i \in S\backslash\sigma(S)\ ,\ j \notin S \cup \sigma(S) \cup \{Ob\}\}$.

$$x(E_S^{\sigma}) \geq 1 \tag{11}$$

Outras duas famílias de desigualdades válidas são as *Lifted Subtour Elimination Constraints* descritas em (12), e as *Generalized Order Constraints* definidas em (13). Para cada uma destas famílias foram implementadas as rotinas de separação propostas em Dumitrescu et al. (2009).

$$x(S) + \sum_{s_j \in S \cap P, t_j \notin S} x_{\{s_i,t_j\}} \leq |S| - 1, \tag{12}$$

sendo $S \subseteq P \cup D$, tal que existe $s_i \in P \cap S$ e $t_i \in S$.

$$\sum_{i=1}^{m} x(S_i) \leq \sum_{i=1}^{m} |S_i| - m - 1, \tag{13}$$

em que $S_1, ..., S_m \subset P \cup D$ são conjuntos dois a dois disjuntos, tal que $m \geq 2$ e $S_i \cap \pi(S_{i+1}) \neq \emptyset, \forall i \in \{1, ..., m-1\}$ e $S_m \cap \pi(S_1) \neq \emptyset$. Para estas restrições foi considerado apenas o caso em que $m = 2$.

2.2. Subproblema de Empacotamento

Aqui é apresentado somente o algoritmo utilizado para verificar se uma dada instância pode ser empacotada. A próxima seção apresenta como é uma instância do subproblema de empacotamento. O algoritmo utilizado é um *branch-and-bound* proposto em Clautiaux et al. (2008), porém combinamos ele com o uso de pontos chamados de *reduced raster points*.

Clautiaux et al. (2008) consideraram, para cada item i, variáveis X_i e Y_i denotando um ponto viável $p = (X_i, Y_i)$ onde o item i pode ser empacotado. O domínio destas variáveis é $X_i \in [0, \ldots, L - l_i]$ e $Y_i \in [0, \ldots, C - c_i]$. Assumindo que os dados de entrada são números inteiros, segue que $X_i \in \{0, 1, \ldots, L - l_i\}$ e $Y_i \in \{0, 1, \ldots, C - c_i\}$. Um modelo de programação por restrições simples requer que cada par $\{i, j\}$ não tenha sobreposição, ou seja:

$$[X_i + l_i \leq X_j] \; or \; [X_j + l_j \leq X_i] \; or \; [Y_i + c_i \leq Y_j] \; or \; [Y_j + c_j \leq Y_i] \tag{14}$$

Em seguida, seja o problema de *scheduling* contínuo e não-preemptivo para o qual $A^L = \{A_1^L, A_2^L, \ldots, A_n^L\}$ é o conjunto de atividades (todos os itens em V) relacionadas com a dimensão

L do recipiente. Cada atividade A_i^L têm duração l_i, consome c_i da capacidade C e pode ocorrer no intervalo $[s_i^L,\ e_i^L) = [0, L - l_i]$, em que s_i^L e e_i^L representam, respectivamente, o tempo mais cedo para iniciar e o maior prazo para terminar tal atividade. Uma solução do problema consiste em determinar o tempo de início $start_i^L$ para cada atividade A_i^L, enquanto satisfaz as restrições de capacidade, ou seja, em qualquer tempo $t \in [0, \ldots, L]$, a soma do recurso consumido pelas atividades com $start_i^L \leq t < start_i^L + l_i$ tem que ser menor ou igual a capacidade C.

De forma similar, é definido um segundo problema de *scheduling* contínuo e não-preemptivo para o conjunto $A^C = \{A_1^C, A_2^C, \ldots, A_n^C\}$ com relação a dimensão C do recipiente. Os dois problemas, cada um para uma das dimensões, pode ser conectado por meio das restrições $[start_i^L = X_i]$ e $[start_i^C = Y_i]$, permitindo resolver o problema de Empacotamento Ortogonal Bidimensional (Clautiaux et al., 2008).

No algoritmo *branch-and-bound*, a ramificação em cada vértice u ocorre nas variáveis não fixadas. Primeiro, entre todas as variáveis $start_i^L$ não fixadas, aquela de menor tempo de início é escolhida. Dois nós são criados na árvore: u_1 para o qual $start_{i_{min}}^L$ é fixo no seu limitante inferior, e u_2 onde o limitante inferior do domínio de $start_{i_{min}}^L$ é acrescido de um fator. Limitantes inferiores baseados em funções duais válidas, limitantes de problemas de *scheduling* e soluções de problemas da mochila são utilizados por (Clautiaux et al., 2008) para apertar os limitantes do algoritmo *branch-and-bound*.

Apesar de considerar o domínio discreto, sabemos que não é preciso assumir todos as posições inteiras para obter uma solução ótima. Sem perda de generalidade, é possível utilizar uma malha de pontos sobre os pontos de discretização de Herz (1972). Um ponto de discretização da largura (respectivamente, comprimento) é um inteiro não-negativo $d \leq L$ (respectivamente, $e \leq C$), ou seja, uma combinação binária não-negativa do tamanho dos itens $l = (l_1, \ldots, l_n)$ (respectivamente, $c = (c_1, \ldots, c_n)$). O conjunto de pontos de discretização da largura (respectivamente, comprimento) é denotado por HP (respectivamente, HQ).

Todavia, foi utilizado um conjunto de pontos mais específicos para a malha, que são os *reduced raster point* de Scheithauer e Terno (1996), definidos como:

$$\begin{array}{ll} \tilde{RP} := \{\langle L - r\rangle_{HP} \mid r \in HP\}, & \text{onde } \langle s\rangle_{HP} = \max\{t \in HP \mid t \leq s\}; \\ \tilde{RQ} := \{\langle C - u\rangle_{HQ} \mid u \in HQ\}, & \text{onde } \langle a\rangle_{HQ} = \max\{b \in HQ \mid b \leq a\}, \end{array} \tag{15}$$

em que $\tilde{RP}$ (de HP) e $\tilde{RQ}$ (de HQ) são os *reduced raster points* para a largura e comprimento, respectivamente.

Assim, o domínio das variáveis X_i e Y_i, consequentemente, $start_i^L$ e $start_i^C$, para cada item i, correspondem aos conjuntos de *reduced raster points*. Chamamos de CP2 o algoritmo *branch-and-bound* de (Clautiaux et al., 2008) combinado com uso dos *reduced raster points*.

3. Algoritmo Genético de Chaves Aleatórias Viciadas - BRKGA

O BRKGA foi introduzido por Gonçalves e Almeida (2002) e Ericsson et al. (2002) e, desde então, tem sido utilizado para resolver eficientemente vários problemas em otimização, incluindo: problemas de cobertura (Resende et al., 2011), problemas de empacotamento (Gonçalves e Resende, 2011), e *scheduling* (Gonçalves et al., 2009).

Duas características fundamentais do BRKGA são: a codificação dos cromossomos em um vetor de chaves aleatórias (alelos) dentro do intervalo $[0, 1)$; e, um processo de evolução caracterizado por um operador de cruzamento uniforme parametrizado (Spears e DeJong, 1991).

Assim, o BRKGA realiza a operação de cruzamento sem se preocupar com o fato de que novos indivíduos podem não resultar em soluções viáveis para o problema. Na etapa de decodificação dos cromossomos em soluções para o problema, aqueles que não originarem uma solução viável têm um fator de penalidade adicionado a sua função de avaliação.

O Algoritmo 1 resume um típico *framework* para o BRKGA, como o proposto por Toso e Resende (2012), em que os parâmetros a serem definidos são: tamanho t dos cromossomos,

tamanho pop da população, tamanho pop_e do conjunto de cromossomos elite, número pop_μ de mutantes a introduzir em cada geração, e a probabilidade ρ_e de herança da chave elite.

Algoritmo 1: BRKGA.

1 Gere uma população inicial Pop.
2 **enquanto** *critério de parada não atendido* **faça**
3 **Decodificar** cada cromossomo de Pop e avaliá-lo.
4 Ordene os cromossomos de Pop em ordem não-crescente de suas aptidões. Os primeiros pop_e cromossomos formam o grupo elite Eli.
5 Copie Eli para a próxima geração Qn.
6 Adicione pop_μ novos cromossomos gerados de forma aleatória (os mutantes) em Qn.
7 Gere $pop - pop_e - pop_\mu$ descendentes aplicando o cruzamento parametrizado, selecionando um pai de Eli e outro de $Pop \backslash Eli$. Adicione os descendentes em Qn.
8 $Pop \leftarrow Qn$.
9 **retorna** *cromossomo de melhor aptidão*.

3.1. Decodificador para o 2L-TSPPD

O objetivo do decodificador é extrair do cromossomo uma solução para o 2L-TSPPD. Consideramos cromossomos de tamanho $|P| + |D|$, em que cada alelo é associado ao par $\{$chave aleatória, vértice de $P \cup D\}$.

O primeiro passo consiste em ordenar os alelos de forma crescente pelo valor da chave aleatória. Após a ordenação, a ordem dos vértices associada ao cromossomo sofreu modificação, assim originando um ciclo Tr. Claramente, isto pode resultar em um ciclo inviável por não atender as restrições de precedência entre os pontos de coleta e entrega.

As restrições de precedência são verificadas da seguinte forma: cada vértice s_i precisa aparecer antes do seu respectivo t_i no ciclo Tr. Ou seja, conta-se o número de vértices t_i que aparecem antes dos respectivos vértices de coleta s_i, armazenando a quantidade total dos que violam em $vPrec$. Se $vPrec$ é zero, então Tc é uma solução válida para o TSPPD, tal que ainda é preciso verificar para a viabilidade do empacotamento segundo solicita o 2L-TSPPD. Caso contrário, usamos a variável $areaP$ para armazenar a soma total da área dos itens que tiveram seus vértices violando a restrição de precedência. Similarmente, usamos a variável $weightP$ para armazenar a soma total do peso destes itens que violam a restrição.

A rotina para verificar se um empacotamento é viável, chamada de *checkPack*, utiliza o algoritmo CP2 descrito na Seção 2.2. A entrada de *checkPack* é um conjunto de caminhos obtido do ciclo Tr. O primeiro caminho T_1 inicia em Oa, em seguida vai para um ponto de coleta s_i, podendo passar por outros pontos de coleta/entrega, tendo seu término em um ponto de entrega t_j, tal que o próximo vértice após t_j é um ponto de coleta s_k (ou Ob).

Para todos os pontos de coleta em T_1, a rotina cria uma instância para o CP2 e restringe o seu tempo de execução em um dado valor limite. Note que a instância contém os itens de todos os pontos de coleta do referido caminho. Se CP2 retorna com um empacotamento viável, *checkPack* considera o próximo caminho de Tr, chamado de T_2, que inicia no ponto de coleta s_k, podendo passar por outros pontos de coleta/entrega, e finaliza em algum ponto de entrega (ou em Ob). Agora, a nova instância para o CP2 consiste de todos os itens dos pontos de coleta em T_2 mais os itens dos pontos de coleta de T_1 que foram atendidos em T_1 (são os itens remanescentes no ciclo). Esses passos são repetidos até que o último caminho T_{end} comece em Ob.

Observe que todos os itens de um dado caminho não necessariamente podem ser empacotados juntos na superfície retangular do veículo, devido ao limite de suas dimensões ou da sua carga máxima permitida. Também, porque o CP2 pode ter falhado em encontrar uma solução viável, ou por ter atingido o tempo limite imposto. Em qualquer um destes casos, um fator de penalidade é considerado na avaliação do cromossomo, resultando em uma das três situações:

- Se a área total dos itens daquele caminho ultrapassa a área do veículo, ou o peso total destes itens ultrapassa o capacidade de carga do veículo, então considere *penal* recebendo a soma do valor de área com o do peso que ultrapassam as limitações do veículo;
- Se CP2 provar que não existe solução viável para o empacotamento, então considere *penal* recebendo a diferença entre a capacidade de área do veículo e a área total dos itens empacotados, mais a diferença entre a capacidade de carga do veículo e o peso total destes itens;
- Se CP2 falha em encontrar uma solução devido ao tempo limite imposto, então considere *penal* recebendo metade do valor descrito no item anterior.

O valor resultante da penalidade retornado pela rotina *checkPack* é armazenado na variável *penalPack* e ele corresponde a soma de *penal* para cada uma rotas extraídas do ciclo Tr. Após a chamada a *checkPack*, o cromossomo é avaliado, ou seja, se *vPrec* não é nula:

$$fitness = \sum_{e \in Tc} c(e) + vPrec \times \left(\frac{\sum_{e \in E} c(e)}{|E|} \right) + vPrec \times (areaP + weightP), \tag{16}$$

em que E é o conjunto de arestas no grafo da instância de entrada. Por outro lado, se alguma restrição de precedência foi violada:

$$fitness = \sum_{e \in Tc} c(e) + vPrec \times \left(\frac{\sum_{e \in E} c(e)}{|E|} \right) + penalPack. \tag{17}$$

4. Experimentos Computacionais

Todos os algoritmos foram codificados na linguagem C++ e os experimentos ocorreram em um computador com processador Intel Core i7-4790K de 4.0 GHz, 32 GB de memória RAM e sistema operacional Linux, rodando em uma única *thread*.

O algoritmo *branch-and-cut* padrão fornecido pelo Gurobi Optimizer 6.0 foi utilizado para resolver a formulação inteira do 2L-TSPPD. O algoritmo CP2 utiliza o *framework* do IBM ILOG CP Optimizer 12.6.1 e IBM ILOG CP 1.7. O tempo limite de 3600 segundos foi imposto para o Gurobi resolver cada uma das instâncias, incluindo neste tempo o tempo total do CP2, com cada chamada limitada a 300 segundos.

Os parâmetros utilizados pelo BRKGA foram obtidos a partir de testes de calibração, em que apenas uma população independente foi considerada. Nestes testes, o objetivo foi balancear a razão: valor da solução por tempo computacional requerido para alcançá-la. Assim, os parâmetros resultantes são: $pop = 3t$, em que t é o tamanho do cromossomo; $pop_e = 20\%$ da população; $pop_\mu = 10\%$ da população; e, $\rho_e = 60\%$. O critério de parada adotado consistiu em alcançar um número máximo de iterações (usamos 500) ou um tempo limite (usamos uma hora). Aqui, cada chamada do CP2 foi limitada em 20 segundos.

As rotinas para eliminação de subciclos, restrições de precedência e resolver o subproblema de empacotamento são aplicadas sempre que uma solução inteira é encontrada. Para o subproblema de empacotamento, usamos a rotina *checkPack* descrita na Seção 3.1. No caso de uma solução fracionária (relaxação de um nó da árvore), o subproblema de empacotamento não é considerado. Por outro lado, as outras rotinas de separação são todas aplicadas até atingir o 100º nó da árvore. Após isto, as rotinas passam a ser aplicadas de dez em dez nós.

Uma vez que a literatura não havia resolvido o 2L-TSPPD, adaptamos 35 instâncias do TSPPD usadas por Dumitrescu et al. (2009). O número de pedidos está em $\{5, 10, 15, 20, 25, 30, 35\}$ e o custo das arestas corresponde a distância Euclidiana entre as posições de seus vértices extremos. As coordenadas dos vértices foram geradas de forma aleatória considerando a malha $[0, 1000] \times [0, 1000]$.

O veículo transporta um contêiner de 40-pés padrão, em que a sua superfície retangular tem $A = L \times C = 2358 \times 12032$ mm^2 e a carga máxima suportada é de $Q = 26600$ kg. Além disso, os valores de L e C são divididos por um fator α. As seguintes dimensões de paletes padrões foram consideradas (todos os valores em *mm*): 1016×1219, 1165×1165, 1067×1067, 800×1200, 1219×1016, 1219×1219, 889×1156, 1219×1143, 914×914, 1219×914, 1219×508, 800×600, 600×400, 400×300, de forma a criar a demanda de cada ponto de coleta. Estas dimensões foram divididas por um fator α, enquanto o peso d de cada palete é igual a sua área vezes um fator β.

O tamanho do conjunto de itens R_i, de cada ponto de coleta $i \in P$, foi determinado de forma aleatória, atribuindo alguns dos paletes mencionados para cada s_i. O tamanho de R_i é limitado em até 10 paletes. Finalmente, assumimos $\alpha = 100$ e $\beta = 10$ por melhor representarem o problema e, também, consideramos apenas a parte inteira dos números para formar as instâncias.

4.1. Resultados

Apresentamos na Tabela 1 os resultados para todas as instâncias. Cada linha da tabela tem: nome da instância; número total de vértices; número total de itens; a saída para o BRKGA é o tempo total gasto em segundos por todas as chamadas do CP2, o tempo total (incluindo o do CP2) e o valor da solução; a saída do algoritmo *branch-and-cut* mostra o número de cortes de usuário inseridos, o tempo total gasto pelo CP2, o tempo total (incluindo o do CP2) e o valor da solução; e, a última coluna contém a razão entre o valor da solução do BRKGA pela do *branch-and-cut*.

É importante mencionar que a solução retornada pelo algoritmo *branch-and-cut* pode não ser a ótima, caso o tempo limite de uma hora tenha sido alcançado, ou se o tempo limite do CP2 foi alcançando antes de provar a viabilidade do empacotamento. Assim, marcamos a coluna com o tempo do CP2 com um "*" para indicar que alguma das chamadas do CP2 atingiu o tempo limite imposto. Além disso, para a solução do algoritmo *branch-and-cut* pode aparecer um "*" indicando que aquela solução é o melhor limitante (possivelmente fracionário) computado para a função objetivo, mesmo se nenhuma solução inteira foi encontrada.

De acordo com os resultados na Tabela 1, somente 8 delas foram resolvidas na otimalidade com o algoritmo *branch-and-cut*, enquanto que para as demais, este algoritmo alcançou o tempo limite imposto (ou o CP2 teve alguma chamada que alcançou seu tempo limite), de forma que o melhor limitante da função objetivo foi retornado. O tempo total deste algoritmo foi de $2.926,97$ segundos, na média. Para todas as instâncias, o número total de cortes inseridos que foram oriundos das rotinas de separação foi de 129.857, com valor médio de $3.710,2$ por instância.

O BRKGA obteve soluções piores comparados ao algoritmo *branch-and-cut* para quase todas as instâncias da Tabela 1. O maior problema foi no tempo limite imposto para cada chamada do CP2, que é bem menor do que aquele considerado para o algoritmo *branch-and-cut*. Assim, o BRKGA conseguiu gerar ciclos válidos bem rapidamente, porém o CP2 não foi bom o suficiente para checar a viabilidade dos empacotamentos.

Note que o BRKGA computou a solução ótima para duas das 35 instâncias (veja prob5a e prob5e), enquanto que a razão entre a sua solução e a do algoritmo *branch-and-cut* é de $1,57$, na média, significando que a solução do BRKGA teve seu valor 57% maior, na média, comparada com a do outro algoritmo. A pior razão aconteceu para a instância *prob*35*c*, com crescimento de 121% no valor da solução.

Comparando o tempo total das abordagens, destaque para o BRKGA que requereu $811,10$ segundos contra $2926,97$ segundos, na média, do algoritmo *branch-and-cut*. Claramente, esta diferença não justifica o comportamento fraco do BRKGA frente o problema a ser resolvido, mas indica que o CP2 não é a melhor escolha para resolver o subproblema de empacotamento.

Tabela 1: Resultados para o conjunto de 35 instâncias.

Instância	#vértices	#itens	BRKGA			branch-and-cut				Razão
			tempo do CP2	tempo total	solução	#cortes	tempo do CP2	tempo total	solução	
prob5a	11	24	42,53	42,57	3585	26	0,00	0,00	3585	1,00
prob5b	11	19	0,07	0,10	2758	47	0,00	0,02	2657	1,04
prob5c	11	28	1295,76*	1295,78	4738	38	20,1	20,11	3851	1,23
prob5d	11	19	0,20	0,23	3230	4	0,00	0,00	3128	1,03
prob5e	11	20	0,23	0,26	3140	49	0,00	0,01	3140	1,00
prob10a	21	54	2512,78*	2512,82	6521	879	3413,5*	3439,37	5167	1,26
prob10b	21	57	1629,95*	1629,99	5867	976	3507,81*	3580,71	5071	1,16
prob10c	21	57	3617,36*	3617,4	5037	649	3617,58*	3631,81	4145*	1,21
prob10d	21	48	3631,08*	3631,08	7437	631	801,54*	818,11	4783	1,55
prob10e	21	72	985,86*	985,86	6065	1253	3636,34*	3685,96	5070*	1,20
prob15a	31	96	703,83*	703,87	7368	1791	3317,9*	3624,68	5225*	1,41
prob15b	31	93	742,2*	742,24	8372	1559	3046,67*	3766,03	5538*	1,51
prob15c	31	85	852,42*	852,45	7415	4171	3235,61*	3620,02	5107*	1,45
prob15d	31	79	2941,48*	2941,5	7272	4312	3452,19*	3654,05	5466*	1,33
prob15e	31	71	1378,38*	1378,43	8480	4533	3436,66*	3708,94	5333*	1,59
prob20a	41	104	171,02*	171,09	8029	7559	2096,53*	3571,87	5719*	1,40
prob20b	41	92	795,17*	795,25	9565	6076	2602,5*	3741,56	6122*	1,56
prob20c	41	125	361,91*	362,01	9325	6657	2576,58*	3763,18	6104*	1,53
prob20d	41	114	1360,26*	1360,3	9664	2199	2397,83*	3582,56	6092*	1,58
prob20e	41	130	300,09*	300,16	8946	2186	336,81*	3588,32	6358*	1,40
prob25a	51	144	170,01*	170,25	10724	4664	2726,36*	3670,79	6653*	1,61
prob25b	51	136	254,66*	254,94	10695	5042	1801,38*	3834	6259*	1,70
prob25c	51	136	425,14*	425,45	12356	7695	2993,13*	3858,52	6784*	1,82
prob25d	51	142	138,83*	139,04	11628	4299	2097,21*	3570,88	6562*	1,77
prob25e	51	125	1146,03*	1146,29	11638	6014	1275,81*	3564,97	6515*	1,79
prob30a	61	158	43,72*	44,14	11466	3108	0,1	3566,03	6817*	1,68
prob30b	61	158	89,9*	90,34	12563	2710	599,63*	3582,65	6476*	1,94
prob30c	61	168	334,52*	334,98	13772	7150	299,49*	3560,31	7290*	1,89
prob30d	61	158	83,81*	84,22	12136	8095	905,1*	3555,73	7042*	1,72
prob30e	61	154	306,56*	306,92	14395	8997	320,94*	3561,8	6675*	2,15
prob35a	71	194	465,77*	466,36	14853	1422	0,00	5,19	6971	2,13
prob35b	71	191	87,71*	88,3	14466	2844	12,39	3574,64	7140*	2,03
prob35c	71	193	222,19*	222,69	16714	3763	300,48*	3574,91	7565*	2,21
prob35d	71	205	1055,92*	1056,44	15384	9404	599,62*	3585,08	7272*	2,11
prob35e	71	194	233,72*	234,22	16321	9055	901,4*	3581,48	7612*	2,14
Média	-	-	**810,88**	**811,08**	-	**3710,2**	**1609,41**	**2926,97**	-	**1,57**

5. Conclusões

Este trabalho apresentou uma heurística e um modelo de programação inteira, resolvido por um algoritmo do tipo *branch-and-cut*, para resolver uma variante do problema do Caixeiro Viajante em que há a inclusão da restrição de Coleta e Entrega e os itens possuem forma bidimensional.

A heurística consistiu no Algoritmo Genético de Chaves Aleatórias Viciadas, em que a etapa de decodificação opera sobre os cromossomos gerando um ciclo Hamiltoniano que satisfaça as restrições de precedência. Em seguida, uma rotina é usada para checar a viabilidade de caminhos desse ciclo quanto a um problema de empacotamento bidimensional. Embora a heurística apresentasse soluções para o problema, elas eram inferiores às computadas pelo algoritmo *branch-and-cut* e algumas vezes despendia o mesmo tempo computacional deste último.

O algoritmo *branch-and-cut* computou a solução ótima para 8 das 35 instâncias, porém sempre requisitou mais tempo computacional comparado ao algoritmo genético. Claramente, este resultado foi influenciado pelo algoritmo para resolver o modelo de programação por restrições, que várias vezes não conseguiu provar a viabilidade do empacotamento. Similarmente, temos a mesma conclusão para os resultados não satisfatórios da heurística.

Em trabalhos futuros, pretendemos considerar novas rotinas de separação e desigualdades válidas, incluindo adaptar aquelas que foram propostas em Dumitrescu et al. (2009) para o TSPPD. Para o algoritmo genético, desejamos considerar uma nova estratégia para realizar o empacotamento dos itens, de preferência deixar que ele próprio decida sobre o empacotamento.

Agradecimentos.

Os autores gostariam de agradecer o apoio financeiro recebido do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), FAPEG (Fundação de Amparo à Pesquisa do Estado de Goiás) e FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo).

Referências

E. E. Bischoff e M. S. W. Ratcliff. 1995, Issues in the development of approaches to container loading. *OMEGA*, 23(4):377–390.

F. Clautiaux, J. Carlier e A. Moukrim. 2007, A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183:1196–1211.

F. Clautiaux, A. Jouglet, J. Carlier e A. Moukrim. 2008, A new constraint programming approach for the orthogonal packing problem. *Computers & Operations Research*, 35:944–959.

I. Dumitrescu, S. Ropke, J.-F. Cordeau e G. Laporte. 2009, The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269–305.

M. Ericsson, M. G. C. Resende e P. M. Pardalos. 2002, A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333.

S. P. Fekete, J. Schepers e J. C. van der Veen. 2007, An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587.

M. R. Garey e D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, 1979.

R. E. Gomory e T. C. Hu. 1961, Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570.

J. F. Gonçalves, J. J. M. Mendes e M. G. C. Resende. 2009, A random key based genetic algorithm for the resource constrained project scheduling problems. *Computers and Operations Research*, 36:92–109.

J. F. Gonçalves e J. Almeida. 2002, A hybrid genetic algorithm for assembly line balancing. *J. of Heuristics*, 8:629–642.

J. F. Gonçalves e M. G. C. Resende. 2011, A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22:180–201.

H. Hernández-Pérez e J.-J. Salazar-González. 2004, A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139.

J. C. Herz. 1972, A recursive computational procedure for two-dimensional stock-cutting. *IBM Journal of Research Development*, páginas 462–469.

D. S. Johnson e L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In *Local Search in Combinatorial Optimization*, páginas 215–310. John Wiley and Sons, 1997.

B. Kalantari, A. V. Hill e S. R. Arora. 1985, An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386.

R. Lahyani, M. Khemakhem e F. Semet. 2015, Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, 241:1–14.

A. Letchford, G. Reinelt e D. Theis. A faster exact separation algorithm for blossom inequalities. In Daniel Bienstock e George Nemhauser, editors, *Integer Programming and Combinatorial Optimization*, volume 3064 of *Lecture Notes in Computer Science*, páginas 196–205. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22113-5.

M. Mesyagutov, G. Scheithauer e G. Belov. 2012, Lp bounds in various constraint programming approaches for orthogonal packing. *Computers & Operations Research*, 39(10):2425–2438.

J. Renaud, F. F. Boctor e J. Ouenniche. 2000, A heuristic for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 27(9):905–916.

J. Renaud, F. F. Boctor e G. Laporte. 2002, Perturbation heuristics for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 29(9):1129–1141.

M. G. C. Resende, R. F. Toso, J. F. Gonçalves e R. M. A. Silva. 2011, A biased random-key genetic algorithm for the Steiner triple covering problem. *Optimization Letters*, páginas 1–15.

G. Scheithauer e J. Terno. 1996, The G4-heuristic for the pallet loading problem. *Journal of the Operational Research Society*, 47:511–522.

W. M. Spears e K. A. DeJong. 1991, On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, páginas 230–236.

R. F. Toso e M. G. C. Resende. A c++ application programming interface for biased random-key genetic algorithms. Technical report, AT&T Labs Research, Florham Park, New Jersey, 2012.