Uma Reformulação Para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas

Armando H. Pereira, Sebastián A. Urrutia, Afonso H. Sampaio
Departamento de Ciência da Computação - Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627, Pampulha, Belo Horizonte, MG, CEP 31270-901, Brasil
{ahp,surrutia,afonsohs}@dcc.ufmg.br

RESUMO

Nesse trabalho consideramos o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. Neste problema tem-se um veículo que deve atender a um conjunto de requisições definidas por um par de coleta e entrega, que especifica o ponto onde o item deve ser coletado e o ponto onde o item deve ser entregue, respectivamente. Os itens são armazenados em pilhas, que têm capacidade limitada, e cuja operação deve seguir uma política LIFO. O objetivo do problema é atender todas as requisições com custo mínimo. Propomos uma reformulação de um modelo da literatura e um algoritmo *branch-and-cut* baseado na reformulação proposta. Resultados mostram que o algoritmo baseado na reformulação é mais rápido que o algoritmo baseado na formulação anterior. Além disso, duas instâncias não resolvidas previamente são solucionadas e vários gaps de otimalidade reduzidos.

PALAVRAS CHAVE. Caixeiro Viajante, Restrições de Carregamento, Planos de Corte, Coleta e Entrega.

ABSTRACT

In this paper we consider the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks. In this problem there is one vehicle which must fulfill a set of requests defined by a pickup and delivery pair, which specifies the point where the item is loaded and the point where the item is delivered, respectively. The items are stored in stacks, which have limited capacity, and whose operation must follow a LIFO policy. The objective of the problem is to fulfill all the requests with minimum cost. We propose a reformulation of a model in the literature and branch-and-cut algorithm based on the proposed reformulation. Results show that the algorithm based on the reformulation is faster than the algorithm based on the previous formulation. In addition, two previously unsolved instances are now solved and several optimality gaps are reduced.

KEYWORDS. Travelling Salesman, Loading Constraints, Branch-and-Cut, Pickup and Delivery.

1. Introdução

Devido a sua importância econômica e prática, um dos problemas mais estudados em Otimização Combinatória é o Problema de Roteamento de Veículos. O problema arquétipo na área de roteamento de veículos é o Problema do Caixeiro Viajante (PCV). O objetivo é encontrar um trajeto que visite todas as cidades e retorne a cidade inicial com menor custo possível. O PCV pode ser encontrado na literatura com diversas restrições adicionais, tais como janelas de tempo [Da Silva e Urrutia, 2010] e restrições de coleta e entrega [Dumitrescu et al., 2010].

Dois problemas centrais na logística de transportes são roteamento de veículos e o carregamento de objetos neles. Grande parte dos problemas nessas duas áreas são NP-difícil e consequentemente custosos de se resolver em prática. Por esse motivo, tradicionalmente, essas duas áreas de pesquisas tem sido tratadas separadamente [Iori e Martello, 2010]. Recentemente algoritmos combinando esses dois problemas tem sido propostos na literatura.

No Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas (Pickup and Delivery Traveling Salesman Problem with Multiple Stacks - PDTSPMS) um único veículo deve atender um conjunto de requisições. Cada requisição é definida por um ponto de coleta onde o item deve ser carregado em uma pilha, e um ponto de entrega onde ele deve ser descarregado. No veículo, os itens são armazenados em pilhas de capacidade limitada. A restrição LIFO impõe que a coleta seja feita no topo de uma pilha, enquanto a entrega só poderá ser realizada se o item associado também estiver. Iniciando e terminando em dados depósitos, o objetivo é encontrar uma rota com custo mínimo que atenda todas as requisições dos clientes. O problema encontra aplicações onde a coleta e a entrega dos itens só podem ser realizadas pela parte traseira do veículo (e.g. carro furgão, caminhão cegonha), já que somente o último item carregado em uma pilha está disponível para entrega [Côté et al., 2012a].

Segundo Iori e Martello [2010], a primeira contribuição para o Problema do Caixeiro Viajante com Coleta e Entrega seguindo a política LIFO foi feita por Ladany e Mehrez [1984]. Ladany e Mehrez [1984] exploram uma aplicação real de entrega de contêineres de leite em Israel. Apesar de fornecerem uma descrição do problema, nenhuma formulação matemática é fornecida. Levitin e Abezgaouz [2003] também tratam uma aplicação real do problema, eles consideram o caso onde os carregamentos são colocados em estrados e cada novo estrado é carregado em cima do lote anterior pelo veículo guiado automatizado. Com o intuito de evitar o uso excessivo de espaço e tempo para reorganizar os estrados no lote ao mover itens entre diferentes locais em um armazém, a coleta e a entrega são realizadas de acordo com a política LIFO.

O PDTSPMS é uma generalização do Problema do Caixeiro Viajante com Coleta e Entrega sob Carregamento LIFO (Pickup and Delivery Traveling Salesman Problem with LIFO loading constraints - PDTSPL). O PDTSPL é uma versão mais simples onde o veículo contém uma única pilha LIFO de capacidade ilimitada. Um algoritmo *branch-and-bound* exato foi introduzido por Carrabs et al. [2007a] para o problema. Cordeau et al. [2010b] propuseram três formulações inteiras para o PDTSPL, duas para o caso capacitado e uma para o não-capacitado, também são propostas desigualdades válidas, posteriormente usadas em um algoritmo *branch-and-cut*. O TSPPDL também foi estudado por Carrabs et al. [2007b], que propõe um heurística *Variable Neighborhood Search* e novos operadores de busca local. Um estudo poliédrico e um algoritmo *branch-and-cut* são apresentados em Dumitrescu et al. [2010].

Uma variação para o PDTSPMS é o Problema do Caixeiro Viajante Duplo com Múltiplas Pilhas (Double Traveling Salesman Problem with Multiple Stacks - DTSPMS) introduzido por Petersen e Madsen [2009]. No DTSPMS todas as coletas devem ser feitas antes que qualquer entrega possa ser realizada. Um modelo matemático é proposto por Petersen e Madsen [2009], além de heurísticas baseadas em metaheurísticas para resolver o problema. Abordagens exatas para o DTSPMS são encontradas em Petersen et al. [2010] e Lusby et al. [2010]. Uma heurística de *Large Neighborhood Search* desenvolvida para o PDTSPMS é aplicada para o DTSPMS em Côté et al. [2012b]. Outras variantes do Problema do Caixeiro Viajante com Coleta e Entrega com

políticas diferentes de coleta como FIFO, são encontradas em [Carrabs et al., 2007a; Cordeau et al., 2010a; Erdoğan et al., 2009].

O primeiro algoritmo exato para resolver o PDTSPMS foi proposto por Côté et al. [2012a]. Eles fornecem três formulações e desigualdades válidas, que são incorporadas em um algoritmo *branch-and-cut*. Os melhores resultados foram obtidos com a formulação que possui um número exponencial de desigualdades, que estendem para o caso em que mais de uma pilha está disponível as restrições de política LIFO propostas por Cordeau et al. [2010b]. Duas classes de instâncias são consideradas, o tamanho das instâncias resolvidas pelo algoritmo depende do tipo de instância considerada. Os autores também aplicam o algoritmo em instâncias do DTSPMS.

Sampaio e Urrutia [2016] introduziram uma nova formulação de Programação Inteira para o PDTSPMS junto com um algoritmo *branch-and-cut* exato para resolver o problema. Eles também fornecem desigualdades válidas para o problema, com os respectivos métodos de separação. Nessa nova formulação são usadas variáveis para indicar a pilha utilizada na coleta ou entrega de um item ao se percorrer um arco. Os resultados obtidos são competitivos com o algoritmo *branch-and-cut* proposto por Côté et al. [2012a]. Eles solucionaram duas instâncias não resolvidas na literatura previamente e também melhoraram o *gap* de otimalidade de várias instâncias.

O objetivo deste trabalho é reformular o algoritmo e o modelo de programação inteira propostos por Sampaio e Urrutia [2016]. Uma observação é utilizada para mostrar que diversas variáveis do modelo podem ser eliminadas. A quantidade de variáveis e número de restrições no modelo de programação matemática é reduzida. Com a redução do número de variáveis espera-se resultados melhores para as instâncias de *benchmark* propostas por Côté et al. [2012a], principalmente a solução de instâncias na literatura que ainda não foram resolvidas na otimalidade.

O problema é descrito formalmente na Seção 2. Nela o problema é reformulado e o processo utilizado descrito. Na Seção 3 são descritas desigualdades válidas para o problema. O algoritmo *branch-and-cut* proposto para resolver o problema é explicado na Seção 4 e na Seção 5 os experimentos e os resultados do algoritmo são relatados. Na Seção 6 são expostas as conclusões sobre a reformulação.

2. Formulação Matemática

Seja $P = \{1, \ldots, n\}$ um conjunto de pontos de coleta, seja $D = \{n+1, \ldots, 2n\}$ um conjunto de pontos de entrega, e sejam 0 e $2n+1$, respectivamente, os depósitos inicial e final. Seja $G = (V, A)$ um grafo direcionado e completo em que $V = P \cup D \cup \{0, 2n+1\}$ e A é o conjunto de arcos. Cada arco $(i, j) \in A$ possui um custo c_{ij} associado. À cada ponto de coleta $i \in P$ está associado um item de tamanho d_i, que deve ser entregue no ponto $n + i \in D$, associado com um item de tamanho $-d_i$. Um item de tamanho $d_0 = d_{2n+1} = 0$ é associado com os depósitos. Um único veículo, com um conjunto $K = \{1, \ldots, t\}$ de pilhas com capacidade Q, onde cada item é carregado e descarregado seguindo a política LIFO, é utilizado para atender os pedidos. O objetivo é atender todos os pedidos a custo mínimo, começando e terminando, respectivamente, nos depósitos 0 e $2n + 1$, satisfazendo as restrições de precedência, capacidade e a política LIFO.

O PDTSPMS é uma variante do Problema do Caixeiro Viajante com Coleta e Entrega (Traveling Salesman Problem With Pickup and Delivery - PDTSP). O problema consiste em determinar um *tour* de custo mínimo tal que cada vértice de coleta é visitado antes do seu vértice correspondente de entrega. Para formular o PDTSP associamos com cada arco $(i, j) \in A$ uma variável binária x_{ij}, que assume valor 1 se j é visitado a partir do nó i, e 0, caso contrário.

Para a definição do modelo considere a seguinte notação matemática: $\overline{S} = V \setminus S$ onde $S \subseteq V$, $x(S) = \sum_{i,j \in S} x_{ij}$, $x(S, T) = \sum_{i \in S, j \in T} x_{ij}$, $x(i, S) = x(\{i\}, S)$ e $x(S, i) = x(S, \{i\})$. Denote por $\mathcal{S}$ a coleção de todos os subconjuntos $S \subset V$, tal que $0 \in S, 2n + 1 \notin S$ e existe um $i \in P$ tal que $i \notin S$ e $n + i \in S$.

O PDTSP é formulado como:

$$min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$s.a. \quad x(i,V) = 1, \qquad i \in P \cup D \cup \{0\} \tag{2}$$

$$x(V,i) = 1, \qquad i \in P \cup D \cup \{2n+1\} \tag{3}$$

$$x(S) \leq |S| - 1, \qquad S \subseteq P \cup D, |S| \geq 2 \tag{4}$$

$$x(S) \leq |S| - 2, \qquad S \in \mathcal{S} \tag{5}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in A \tag{6}$$

A função objetivo (1) minimiza o custo total da rota. As restrições (2) e (3) garantem que cada vértice é visitado somente uma vez. As restrições (4) asseguram a conectividade na rota, eliminando sub-rotas. As restrições (5), propostas por Balas et al. [1995], estabelecem a relação de precedência entre nós de coleta e entrega, garantido que a entrega $n+i$ não será visitada antes da coleta correspondente i. O PDTSPMS pode ser formulado inserindo-se restrições adicionais ao modelo (1)-(6). Um conjunto de tais restrições é apresentado a seguir.

2.1. Reformulação

Nesta seção apresentamos como são reduzidas as variáveis introduzidas por Sampaio e Urrutia [2016]. Eles propuseram uma nova formulação de Programação Inteira para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas, nela são introduzidas as variáveis binárias y_{ij}^k, que indicam a pilha $k \in K$ que foi manipulada quando o veículo passou por j vindo de i. No caso temos $y_{ij}^k = 1$ se após visitar a localização $i \in V$, o veículo usa a pilha k para coletar ($j \in P$) ou entregar ($j \in D$) o item j, e $y_{ij}^k = 0$, caso contrário.

Na formulação de Sampaio e Urrutia [2016], dada uma variável y_{ij}^k, temos que $j \in P \cup D$. Entretanto, em virtude da pilha para a coleta e a entrega de um item ser a mesma, de posse da informação de que o item é coletado na pilha k, temos por equivalência que ele também será entregue na mesma pilha, dessa forma podemos eliminar as variáveis y_{ij}^k com $j \in D$. Neste trabalho, usamos esse fato para reformular o modelo proposto por Sampaio e Urrutia [2016] eliminando essas variáveis. Em vista disso, temos que o valor de $y_{i,n+j}^k$ com $j \in P$ é dado por:

$$y_{i,n+j}^k = \left\lfloor \frac{x_{i,n+j} + \sum_{u \in V \setminus \{2n+1\}} y_{uj}^k}{2} \right\rfloor, \qquad i \in P \cup D, j \in P, k \in K \tag{7}$$

A função chão é utilizada pois o item j pode ter sido carregado usando a pilha k com o arco $(i, n+j)$ não sendo utilizado, consequentemente o numerador da Equação (7) será 1, resultando em um valor fracionário, apesar de $y_{i,n+j}^k = 0$. O mesmo tipo de argumento se aplica quando o arco $x_{i,n+j}$ é usado embora a pilha k não tenha sido.

Na formulação de Sampaio e Urrutia [2016] o número total de variáveis y_{ij}^k é $(4n^2-n)t + 2n(1-t)$, que pelas observações anteriores é reduzido neste trabalho para $(2n^2-n)t$, onde n é o número de coletas e t o número de pilhas. As variáveis y_{ij}^k são ligadas as varíaveis x_{ij} que indicam se o arco $(i,j) \in A$ é usado ou não através da restrição:

$$x_{ij} = \sum_{k \in K} y_{ij}^k, \qquad i \in \{0\} \cup P \cup D, j \in P \tag{8}$$

Nas próximas seções mostramos as novas restrições para lidar com a política LIFO e a capacidade na reformulação.

2.2. Política LIFO

Para mostrar como a política LIFO pode ser violada, considere dois vértices de coleta, i e j, e um veículo com uma única pilha de capacidade ilimitada. Seja $S \subset P \cup D$ um conjunto tal que $j \in S$ e $i, n+i, n+j \notin S$. Se o veículo visita i, percorre o conjunto S sem deixá-lo e visita imediatamente $n+i$, temos que a política LIFO é violada, dado que j é carregado na mesma pilha em que i, mas $n+i$ é visitado antes de $n+j$. Uma representação gráfica é exibida na Figura 1. Entretanto, se o veículo possuir mais de uma pilha, o mesmo caminho pode ser viável, já que i e j podem ser carregados em pilhas diferentes. Dessa forma, as variáveis **y** são usadas para especificar a pilha utilizada na coleta.

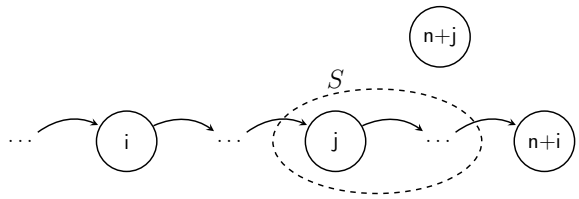

Figura 1: Caminho inviável para política LIFO com uma pilha.

Observe que essa violação não é alterada caso existam outras inviabilidades no subcaminho de i até j. Baseado na notação descrita anteriormente, considere $y^k(S,T) = \sum_{i \in S, j \in T} y^k_{ij}$ onde $k \in K$. A política LIFO é imposta através da desigualdade 9.

$$y^k(\overline{S}, \{i\}) + y^k(\overline{S}, \{j\}) + x(S) + x(S, \{n+i\}) \leq |S| + 1, \\ S \subset P \cup D, j \in S, i, n+i, n+j \notin S, k \in K \tag{9}$$

Ela proíbe todo caminho de i até $n+i$ tal que o nó i é coletado na pilha k ($y^k(\overline{S}, \{i\})$), j é coletado na pilha k ($y^k(\overline{S}, \{j\})$), o conjunto S é percorrido sem visitar vértices externos ($x(S)$) e imediatamente após atravessar S o nó $n+i$ é visitado ($x(S, \{n+i\})$). Repare que são removidos do espaço de solução apenas os caminhos de i até $n+i$ através de S, entrando e saindo por i e $n+i$, respectivamente, com uma estrutura de pilha que leva a inviabilidade da política LIFO.

2.3. Capacidade

Além da política LIFO o veículo também deve respeitar a capacidade de cada uma de suas pilhas. Seja $q(S) = \sum_{i \in S} d_i$ o somatório do tamanho dos itens em S. Considere um caminho de arestas c^k onde a capacidade da pilha $k \in K$ é violada, isto é, existe um conjunto de itens coletados na pilha cuja soma dos tamanhos excede a capacidade dela. Denote $c^k_y = \{j : (i,j) \in c^k, y^k_{ij} = 1\}$ como o conjunto de itens no caminho coletados na pilha k e $c^k_x = \{j : (i,j) \in c^k, y^k_{ij} = 0\}$ como o conjunto de itens não coletados na pilha k. Seja $y(c^k_y) = \sum_{j \in c^k_y} y^k_{ij}$ a soma dos arcos que usam a pilha k e $x(c^k_x) = \sum_{j \in c^k_x} x_{ij}$ a soma dos arcos que não usam a pilha k. A desigualdade de caminho inviável é definida como:

$$x(c^k_x) + y(c^k_y) \leq \left|c^k\right| - 1, \qquad c^k \in \mathcal{C}, k \in K \tag{10}$$

onde $\mathcal{C}$ denota o conjunto de todos os caminhos inviáveis em capacidade tal que $q(c^k_y) > Q$.

Um exemplo é mostrado na Figura 2. Suponha um veículo com capacidade $Q = 2$ e $d_i = 1$ para todo item i, os arcos (i,j) com o rótulo k indicam que o item j foi coletado na pilha k. Considerando que o veículo possui a pilha k vazia quando visita o nó i, ao final do caminho a capacidade da pilha k é excedida, já que $q(c^k_y) = 3$.

A reformulação do PDTSPMS resulta no modelo definido por (1)-(6) e (8)-(10).

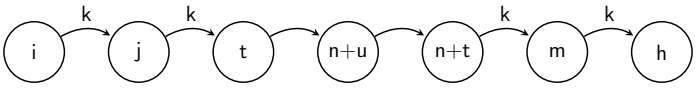

Figura 2: Caminho c^k que viola a capacidade da pilha k. Temos que $c^y_x = \emptyset$ e $c^k_y = \{j, t, m, h\}$. A desigualdade (10) gerada é $x_{t,n+u} + x_{n+u,n+t} + y^k_{ij} + y^k_{jt} + y^k_{n+t,m} + y^k_{mh} \leq 5$.

3. Desigualdades Válidas

Sendo uma versão restringida do TSPPD, todas as desigualdades validas para este problema também são válidas para o TSPPDMS. Seja $\pi(S) = \{i \in P : n + i \in S \subset V\}$ o conjunto de vértices de coleta tais que os respectivos vértices de entrega estão em S, e $\sigma(S) = \{n + i \in D : i \in S \subset V\}$ o conjunto de vértices de entrega tais que os respectivos vértices de coleta estão em S. Três classes de desigualdades introduzidas por Balas et al. [1995] são consideradas:

$$x(n+i, S) + x(S) + x(S, i) \leq |S|, \qquad \forall S \subseteq V \setminus \{0, 2n+1, i, n+i\} \tag{11}$$

$$x(S) + x(S, \overline{S} \cap \pi(S)) + x(S \cap \pi(S), \overline{S} \setminus \pi(S)) \leq |S| - 1, \qquad S \subset P \cup D \tag{12}$$

$$x(S) + x(\overline{S} \cap \sigma(S), S) + x(\overline{S} \setminus \sigma(S), S \cap \sigma(S)) \leq |S| - 1, \qquad S \subset P \cup D \tag{13}$$

A desigualdade (11) impede o caminho de $n + i$ para i através de S, proibindo um caminho onde a entrega é visitada antes da coleta. As desigualdades (12) e (13) são referidas como desigualdades de predecessor e sucessor, respectivamente, e dominam a desigualdade (11). Além disso, também são adicionadas as desigualdades de quebra de ciclo de precedência introduzidas por Balas et al. [1995] para o Problema do Caixeiro Viajante com restrições de precedência:

$$\sum_{u=1}^{m} x(S_u) \leq \sum_{u=1}^{m} |S_u| - m - 1 \tag{14}$$

onde $S_1, \ldots, S_m \subset P \cup D$ são subconjuntos mutuamente disjuntos tal que $u_1, \ldots, u_m \in P$ são coletas onde $u_v, n + u_{v+1} \in S_v$ para $v = 1, \ldots, m$, e $u_{m+1} = u_1$. Sejam i e j duas coletas com suas respectivas entregas $n + i$ e $n + j$, para $m = 2$ e $S_1 = \{i, n + j\}$ e $S_2 = \{j, n + i\}$, em uma desigualdade (14) violada, ou um ciclo ocorre em S_1 ou S_2, ou uma das precedências é desrespeitada.

Côté et al. [2012a] adaptaram para o PDTSPMS desigualdades clássicas do Problema de Roteamento de Veículos Capacitado. Ao invés de considerar cada pilha individualmente, eles levam em conta a capacidade total $K \times Q$, produzindo as desigualdades de capacidade arredondadas:

$$x(S, \overline{S}) + x(\overline{S}, S) \geq 2 \left\lceil \frac{|q(S)|}{KQ} \right\rceil, \qquad \forall S \subset P \cup D \tag{15}$$

Também são utilizadas as desigualdades de conflito de capacidade propostas por Côté et al. [2012a]:

$$x(i, S) + x(S) + x(S, n+i) \leq |S|, \qquad \forall S \subset P \cup D, i, n+i \notin S, z(S) > Q(K-1) \tag{16}$$

onde $z(S) = \max(q(\pi(S) \setminus S), -q(\sigma(S) \setminus S))$. A desigualdade proíbe itens que cruzam com i de serem carregados na mesma pilha, consequentemente eles devem ser carregados no espaço remanescente $(K - 1)Q$ disponível.

Seja S $\subset P \cup D$ um conjunto tal que $-q(\sigma(S) \setminus S) > Q$, ou seja, um conjunto no qual o tamanho total de itens de entrega fora de S com coleta correspondente em S excede a capacidade Q. Denote por $\Pi = \{i \in S : n+i \in \overline{S}\}$ as coletas em S com entrega correspondente fora de S e v o vértice visitado imediatamente antes de entrar em S. Se o veículo realizar a coleta dos itens em Π na pilha k, um caminho através de S deve entrar e sair do conjunto mais de uma vez. Caso contrário, se o veículo entrar e sair de S somente uma vez, quando o veículo sair de S, todos os itens em Π estarão carregados na pilha k, estourando sua capacidade. Portanto, se o veículo entra e sai de S somente uma vez, nem todas as coletas em Π podem ser feitas na pilha k. Dessa forma é adicionada a seguinte desigualdade introduzida por Sampaio e Urrutia [2016]:

$$\begin{aligned} x(S \cup \{v\}, S \setminus \Pi(S)) + y^k(S \cup \{v\}, \Pi(S)) \leq |S| - 1, \\ S \subset P \cup D, v \in \overline{S} \setminus \{2n+1\}, k \in K, -q(\sigma(S) \setminus S) > Q \end{aligned} \tag{17}$$

4. Algoritmo Branch-and-Cut

O algoritmo implementado utiliza o CPLEX 12.5 como *framework* B&C através da API Concert. Pré-processamento, heurísticas e cortes automáticos do solver são desligados. O mecanismo de *callbacks* é usado para separação de *user cuts* e *lazy constraints*. Durante o *branching* as variáveis x_{ij} recebem preferência sobre as variáveis y^k_{ij}, com o objetivo de obter um *tour* e então separar as desigualdades LIFO e de capacidade. Para seleção de nós, variáveis e direção de *branching* são utilizados os valores padrões do CPLEX. O tempo limite de execução para o algoritmo proposto é definido como 3600 segundos. É usado como limite superior (UB) o valor fornecido pela heurística *Large Neighborhood Search* para o PDTSPMS em Carrabs et al. [2007b].

Antes de inicializar o algoritmo arcos que não aparecem em nenhuma solução viável são eliminados do grafo. Arcos da forma $(0, j)$, com $j \in D$, e $(i, 2n+1)$, com $i \in P$, são eliminados, pois o depósito 0 não pode ser o predecessor de nenhuma entrega e o depósito $2n+1$ não pode ser o sucessor de nenhuma coleta, respectivamente. Similarmente, arcos da forma $(n+i, i)$ tal que $i \in P$ são removidos, dado que o predecessor de uma coleta não pode ser sua entrega correspondente.

As variáveis **y** introduzem um grau de simetria na formulação, qualquer solução viável tem um conjunto equivalente de soluções, consistindo na mesma rota permutando as atribuições de itens de duas ou mais pilhas. A primeira visita feita à partir do depósito 0 é um vértice $i \in P$, e este pode ser coletado em qualquer pilha. Em vista disso, para quebrar parte da simetria, a restrição (18) é incluída no modelo, indicando que a primeira coleta deve ser feita usando a pilha 0.

$$\sum_{i \in P} y^0_{0,j} = 1 \tag{18}$$

O modelo inicial é composto pela função objetivo (1), pelas restrições de grau (2) e (3), as restrições de acoplamento (8) entre as variáveis **x** e **y**, e a restrição de simetria (18). São incluídas as restrições de eliminação de sub-rota (4) com $|S| = 2$ e as desigualdades de quebra de ciclo de precedência (13) para $m = 2$, $S_1 = \{i, n+j\}$ e $S_2 = \{j, n+i\}$, onde $i, j \in P$.

4.1. Separação

Dado um vetor ($\mathbf{x}^*$,$\mathbf{y}^*$) $\in \mathbb{R}^{|A|} \times \mathbb{R}^{K|A|}$ não negativo, o grafo de suporte associado G^* é o grafo com vértices $V^* = P \cup D \cup \{0, 2n+1\}$ e com arcos $A^* = \{(i,j) : x^*_{ij} > 0\}$. Os algoritmos de separação consistem essencialmente da resolução de problemas de fluxo máximo em G^* dados uma fonte e um sumidouro, onde o peso do arco $(i,j) \in A^*$ é $w(i,j) = x^*_{ij}$. A partir do fluxo, um conjunto $S \in V$ violando uma desigualdade é computado usando o teorema do corte mínimo.

Para uma solução fracionária ($\mathbf{x}^*$,$\mathbf{y}^*$), os procedimentos de separação para as desigualdades (4), (5) e (11) são descritos em Cordeau et al. [2010b]. Para a separação das desigualdades (12) e (13), quando uma desigualdade (11) é encontrada, o conjunto S é estendido com $\{i\}$ e $\{n+i\}$ para definir desigualdades violadas (12) e (13), respectivamente, já que ambas desigualdades dominam (11). Se nenhuma violação for encontrada, o conjunto S encontrado pelos procedimentos de

separação anteriores é usado para checar se o conjunto define uma desigualdade violada (15), (16) ou (17). As desigualdades (9) não são separadas quando uma solução fracionária é obtida.

Quando a solução é inteira, ao invés de resolver problemas de fluxo para encontrar desigualdades violadas, são usados algoritmos polinomiais para resolver os problemas de separação com menos esforço. Em particular, procedimentos de separação exatos são usados para as restrições LIFO (9) e as restrições de caminho (10). Dessa maneira, uma rota encontrada é determinada viável para o PDTSPMS através desses procedimentos, ao invés de resolver um problema de empacotamento NP-difícil como feito por Côté et al. [2012b]. Os procedimentos polinomiais são descritos abaixo.

No algoritmo para separar as desigualdades (9), dado uma solução inteira, se a solução contém ciclos, eles não são pesquisados para tentar encontrar desigualdades LIFO, já que não existe uma ordem de visita dos vértices dentro do ciclo. Desse modo, só é considerado o caminho de 0 a $2n+1$. O caminho é percorrido e para cada i, carregado em uma pilha k tal que $n+i$ também está no caminho e a precedência é satisfeita, um caminho $i \rightsquigarrow S \rightsquigarrow n+i$ é obtido, então é pesquisado em S uma coleta j carregada na pilha k onde $n+j \notin S$, ou uma entrega $n+j \in S$ tal que $j \notin S$ e j é carregado na pilha k. No primeiro caso, uma desigualdade (9) é obtida para o subcaminho de j até $n+i$, e no segundo caso, para o subcaminho de i até $n+j$.

A separação das desigualdades (10) em uma solução inteira é realizada se o caminho de 0 a $2n+1$ não contém nenhuma precedência e política LIFO violadas. O caminho é percorrido mantendo a capacidade residual de cada pilha. Quando o veículo carrega um item em uma pilha k e a capacidade total excede Q, é adicionada uma desigualdade de caminho da última posição em que pilha k estava vazia até esta última coleta realizada. Também é adicionada uma desigualdade (10) para esse caminho para cada pilha diferente de k.

5. Experimentos e Resultados

O algoritmo foi implementado em C++ e os experimentos computacionais executados em um Intel Xeon E5520 2.2 GHz. O conjunto de instâncias utilizado nos experimentos é o introduzido por Côté et al. [2012a]. Os autores geraram as instâncias baseando-se nas instâncias do PDTSPL em Carrabs et al. [2007a,b] e Cordeau et al. [2010b]. Foram geradas duas classes de instâncias. Na primeira classe C1 o tamanho de cada nó é 1, o número de pilhas é um número aleatório entre 2 e 4 e a capacidade da pilha é um número aleatório entre 1 e 10. Na segunda classe C2, o tamanho de cada nó de coleta é um número aleatório entre 1 e 10, o número de pilhas é um número aleatório entre 2 e 4 e a capacidade é um número aleatório entre 10 e 15.

Para os experimentos computacionais Côté et al. [2012a] usam um AMD Opteron 275 2.2 Ghz e Sampaio e Urrutia [2016] utilizam um Intel Xeon E5520 2.2 GHz. É utilizado em ambos o mesmo conjunto de instâncias usadas neste trabalho.

Nas tabelas 1 e 2 são reportados para cada instância o tempo de CPU em segundos (Tempo) e o *gap* de otimalidade (*Gap*) em instâncias não resolvidas, calculado como $100 * (UB - LB)/UB$, onde LB é o limite inferior. Nas Tabelas 3 e 4 são reportados o número de instâncias resolvidas (Resolvidas), a média do *gap* no nó raiz (Raiz), a média do *gap* de otimalidade (*Gap*), a média de tempo de CPU em segundos (Tempo), nessa ordem, para o conjunto de instâncias resolvidas por todos algoritmos e para todas as instâncias, e em $Caminho$, é reportado o número de desigualdades de caminho inviáveis geradas.

Os resultados para a classe C1 estão sumarizados na Tabela 3. O algoritmo foi capaz de resolver 34 das 54 instâncias, enquanto 34 e 33 são resolvidas por Côté et al. [2012a] e Sampaio e Urrutia [2016], respectivamente. O algoritmo resolveu todas as instâncias C1 resolvidas por Sampaio e Urrutia [2016], além disso um novo certificado de otimalidade é fornecido para a instância a280 com $n = 21$, não resolvida previamente na literatura. O algoritmo também solucionou todas as instâncias resolvidas por Côté et al. [2012a], exceto para d18512 com $n = 15$ e pr1002 com $n = 21$, como o algoritmo proposto resolve duas instâncias que eles não resolvem, o número de instâncias resolvidas é igual.

Instância			B&C Proposto		Sampaio e Urrutia B&C		Côté et al. B&C	
Nome	n	Opt	Gap	Tempo	Gap	Tempo	Gap	Tempo
a280	11	449		5.21		5.9		2.0
	13	477		14.21		19.4		6.9
	15	542		267.01		607.6		105.1
	17	624		83.75		93.3		181.3
	19	669		213.26		470.9		2202.1
	21	739		2837.22	6.19	3600	10.35	3600
att532	11	4177		0.05		0.1		0.2
	13	4937		0.07		0.2		0.7
	15	5151		0.36		0.7		2.9
	17	5294		0.36		0.8		2.8
	19	5587		0.97		2.6		13.8
	21	9266		77.31		214.0		2034.2
brd14051	11	4396		1.88		2.5		8.7
	13	4439		8.47		22.7		30.4
	15	n.a.	4.49	3600	4.79	3600	3.41	3600
	17	n.a.	8.88	3600	9.57	3600	4.65	3600
	19	n.a.	32.2	3600	32.56	3600	4.72	3600
	21	n.a.	15.88	3600	16.15	3600	4.40	3600
d15112	11	74,603		0.49		0.9		2.9
	13	80,690		8.22		14.8		37.0
	15	89,754		6.42		13.7		20.6
	17	96,804		1148.25		852.6		3278.0
	19	n.a.	5.35	3600	5.67	3600	5.39	3600
	21	n.a.	11.07	3600	12.03	3600	9.68	3600
d18512	11	4280		0.29		0.4		1.4
	13	4301		0.9		1.7		5.7
	15	4638	1.86	3600	1.94	3600		2574.1
	17	n.a.	1.85	3600	2.32	3600	3.56	3600
	19	n.a.	3.77	3600	4.48	3600	5.78	3600
	21	n.a.	8.67	3600	9.16	3600	9.02	3600
fnl4461	11	1889		0.1		0.2		0.6
	13	2088		2.47		7.2		1.6
	15	2356		1135.08		3005.5		16.8
	17	2517	3.37	3600	4.29	3600		102.6
	19	n.a.	14.63	3600	15.27	3600	3.02	3600
	21	n.a.	27.17	3600	27.14	3600	3.55	3600
nrw1379	11	2690		0.21		0.5		0.9
	13	n.a.	5.04	3600	5.00	3600	3.68	3600
	15	n.a.	5.56	3600	5.95	3600	5.63	3600
	17	n.a.	5.36	3600	5.72	3600	6.21	3600
	19	n.a.	12.69	3600	13.2	3600	12.11	3600
	21	n.a.	16.03	3600	16.17	3600	14.69	3600
pr1002	11	13,718		0.11		0.2		0.4
	13	15,436		0.63		1.4		5.1
	15	16,268		49.1		38.6		146.6
	17	17,601		114.62		164.5		384.2
	19	18,673		222.42		343.7		1761.7
	21	20,150		531.84		578.7	2.31	3600
ts225	11	22,000		0.04		0.3		1.5
	13	29,395		0.4		0.1		2.4
	15	32,541		103.1		286.3		4.0
	17	36,405		52.53		262.2		44.4
	19	n.a.	1.18	3600	0.63	3600	2.18	3600
	21	n.a.	6.67	3600	6.07	3600	5.65	3600

Tabela 1: Resultados para o conjunto de instâncias C1.

Instância			B&C Proposto		Sampaio e Urrutia B&C		Côté et al. B&C	
Nome	n	Opt	Gap	Tempo	Gap	Tempo	Gap	Tempo
a280	11	455		2.93		4.0		8.6
	13	479		8.17		5.6		15.3
	15	553		194.35		255.5		497.5
	17	635		288.85		234.3		1082.3
	19	677		108.87		475.7	6.94	3600
	21	n.a.	8.04	3600	4.63	3600	12.68	3600
att532	11	4190		0.06		0.1		0.3
	13	5033		0.56		0.9		4.7
	15	n.a.	4.35	3600	3.72	3600	5.23	3600
	17	n.a.	5.86	3600	6.40	3600	6.98	3600
	19	n.a.	5.81	3600	6.22	3600	7.68	3600
	21	n.a.	7.58	3600	8.31	3600	8.12	3600
brd14051	11	4386		1.27		1.9		4.0
	13	4458		53.68		168.3		960.4
	15	n.a.	4.93	3600	6.13	3600	3.99	3600
	17	n.a.	7.62	3600	8.43	3600	8.21	3600
	19	n.a.	27.76	3600	27.72	3600	2.81	3600
	21	n.a.	27.84	3600	28.48	3600	3.37	3600
d15112	11	73,872		0.52		0.8		5.1
	13	81,657		193.02		224.1		452.7
	15	91,799		645.08		748.0		1402.2
	17	n.a.	2.74	3600	2.56	3600	3.22	3600
	19	n.a.	1.44	3600	1.85	3600	3.48	3600
	21	n.a.	7.12	3600	7.48	3600	8.50	3600
d18512	11	4341		27.15		27.6		70.6
	13	n.a.	2.65	3600	3.47	3600	2.35	3600
	15	n.a.	4.08	3600	4.33	3600	2.82	3600
	17	n.a.	7.28	3600	8.00	3600	4.15	3600
	19	n.a.	11.69	3600	12.77	3600	6.48	3600
	21	n.a.	19.48	3600	20.13	3600	12.00	3600
fnl4461	11	1889		0.05		0.1		0.5
	13	2088		9.36		23.5		14.7
	15	2262		51.75		73.4		36.8
	17	n.a.	1.74	3600	2.75	3600	3.09	3600
	19	n.a.	6.06	3600	6.60	3600	8.35	3600
	21	n.a.	5.94	3600	7.46	3600	8.79	3600
nrw1379	11	2690		0.21		0.5		1.3
	13	n.a.	1.83	3600	3.19	3600	2.82	3600
	15	n.a.	5.34	3600	6.19	3600	5.62	3600
	17	n.a.	5.69	3600	6.15	3600	6.01	3600
	19	n.a.	10.98	3600	11.66	3600	9.47	3600
	21	n.a.	15.52	3600	15.76	3600	11.33	3600
pr1002	11	13,527		0.43		0.4		2.0
	13	15,221		5.1		17.8		14.2
	15	15,676		5.86		14.4		30.9
	17	17,009		23.34		20.8		75.7
	19	18,136		195.29		172.5		504.8
	21	19,613		394.28		135.9		1322.2
ts225	11	22,000		0.07		0.2		0.9
	13	34,000		523.38	1.45	3600		985.3
	15	37,703		3427.23	5.2	3600	3.44	3600
	17	n.a.	4.95	3600	7.09	3600	5.63	3600
	19	n.a.	8.81	3600	11.09	3600	11.29	3600
	21	n.a.	12.76	3600	13.23	3600	14.18	3600

Tabela 2: Resultados para o conjunto de instâncias C2.

Algoritmo	Resolvidas	Raiz	Gap	Tempo	Caminho
B&C Proposto	34	9.17%	9.5%	109.9 - 1454.6	577.9
Côté et al. B&C	34	6.1%	6.00%	322.0 - 1573.8	8.3
Sampaio e Urrutia B&C	33	9.18%	11.2%	201.1 - 1529.9	506.7

Tabela 3: Média dos resultados para as instâncias C1.

Algoritmo	Resolvidas	Raiz	Gap	Tempo	Caminho
B&C Proposto	26	10.5%	8.4%	93.0 - 1973.4	931.2
Côté et al. B&C	24	7.8%	6.6%	282.9 - 2138.8	186.5
Sampaio e Urrutia B&C	24	9.8%	10.2%	92.6 - 2048.2	460.7

Tabela 4: Média dos resultados para as instâncias C2.

O algoritmo baseado na reformulação reduziu aproximadamente pela metade o tempo de execução para o conjunto de instâncias C1. Além disso o *gap* de otimalidade médio também foi reduzido. O *gap* de otimalidade também foi reduzido em várias instâncias não resolvidas pelos outros algoritmos. Diferente dos outros dois algoritmos, poucas desigualdades de caminho inviáveis são geradas pelos problemas de empacotamento de Côté et al. [2012a] para as instâncias em $C1$, ou seja, quando um *tour* satisfazendo a precedência é encontrado pelo algoritmo deles, é improvável que ele seja inviável. Note que o tempo de execução para todas as instâncias é ligeiramente menor na nossa abordagem.

Na Tabela 4 estão sumarizados os resultados para a classe C2. O algoritmo foi capaz de resolver a instância ts225 para $n = 15$ previamente não resolvida, fornecendo um novo certificado de otimalidade. A instância ts225 para $n = 13$ previamente não resolvida em Sampaio e Urrutia [2016] foi resolvida com o algoritmo proposto, o *gap* de otimalidade médio também melhorou em relação ao algoritmo anterior. Apesar do tempo de execução do algoritmo proposto ser menor na maioria das instâncias em C2 quando comparado com Sampaio e Urrutia [2016], o tempo médio para as instâncias resolvidas por todos os algoritmos foi semelhante devido a instância pr1002 com $n = 21$, em nosso algoritmo o tempo de execução foi de 434.05 segundos enquanto no deles foi de 135.9 segundos. Desconsiderando essa instância, a média de tempo do algoritmo proposto cai de 93 para 77.5 segundos, menor se comparado com a queda de 92.6 para 83 segundos obtido pelo algoritmo deles. No total o algoritmo foi capaz de resolver mais instâncias do que os outros dois algoritmos. A classe C2 é a classe com mais desigualdades de caminhos inviáveis geradas para todos os algoritmos, o que pode ser explicado pelo conjunto possuir as instâncias com a capacidade das pilhas mais apertadas.

6. Conclusão

Esse artigo abordou o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. Propusemos uma reformulação de um modelo da literatura e um algoritmo *branch-and-cut* para a solução exata do problema. Na reformulação as variáveis y_{ij}^k introduzidas para representar a operação de coleta e entrega em cada pilha são reduzidas pela metade, também reduzindo o número de restrições no modelo inicial. Os resultados computacionais são relatados e o algoritmo proposto foi capaz de obter dois novos certificados de otimalidade e melhorar o *gap* de otimalidade para várias instâncias não resolvidas. Além disso, executando no mesmo ambiente computacional, o tempo de execução do algoritmo proposto foi menor do que o algoritmo original e resolveu três instâncias que o mesmo não resolvia antes. Trabalhos futuros incluem a aplicação do algoritmo ao Problema do Caixeiro Viajante Duplo e o desenvolvimento de algoritmos de separação para as desigualdades LIFO.

Referências

Balas, E., Fischetti, M., e Pulleyblank, W. R. (1995). The precedence-constrained asymmetric traveling salesman polytope. *Math. Program.*, 68(3):241–265.

Carrabs, F., Cerulli, R., e Cordeau, J.-F. (2007a). An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with lifo or fifo loading. *INFOR: Information Systems and Operational Research*, 45(4):223–238.

Carrabs, F., Cordeau, J.-F., e Laporte, G. (2007b). Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4): 618–632.

Cordeau, J.-F., Dell'Amico, M., e Iori, M. (2010a). Branch-and-cut for the pickup and delivery traveling salesman problem with {FIFO} loading. *Computers & Operations Research*, 37(5): 970 – 980. Disruption Management.

Cordeau, J.-F., Iori, M., Laporte, G., e Salazar González, J. J. (2010b). A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading. *Networks*, 55(1):46–59.

Côté, J.-F., Archetti, C., Speranza, M. G., Gendreau, M., e Potvin, J.-Y. (2012a). A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(4):212–226. ISSN 1097-0037.

Côté, J.-F., Gendreau, M., e Potvin, J.-Y. (2012b). Large neighborhood search for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(1):19–30.

Da Silva, R. F. e Urrutia, S. (2010). A general vns heuristic for the traveling salesman problem with time windows. *Discret. Optim.*, 7(4):203–211. ISSN 1572-5286. URL `http://dx.doi.org/10.1016/j.disopt.2010.04.002`.

Dumitrescu, I., Ropke, S., Cordeau, J.-F., e Laporte, G. (2010). The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269–305.

Erdoğan, G., Cordeau, J.-F., e Laporte, G. (2009). The pickup and delivery traveling salesman problem with first-in-first-out loading. *Computers & Operations Research*, 36(6):1800 – 1808. ISSN 0305-0548.

Iori, M. e Martello, S. (2010). Routing problems with loading constraints. *TOP*, 18(1):4–27.

Ladany, S. P. e Mehrez, A. (1984). Optimal routing of a single vehicle with loading and unloading constraints. *Transportation Planning and Technology*, 8(4):301–306.

Levitin, G. e Abezgaouz, R. (2003). Optimal routing of multiple-load agv subject to lifo loading constraints. *Comput. Oper. Res.*, 30(3):397–410.

Lusby, R. M., Larsen, J., Ehrgott, M., e Ryan, D. (2010). An exact method for the double tsp with multiple stacks. *International Transactions in Operational Research*, 17(5):637–652.

Petersen, H. L., Archetti, C., e Speranza, M. G. (2010). Exact solutions to the double travelling salesman problem with multiple stacks. *Networks*, 56(4):229–243.

Petersen, H. L. e Madsen, O. B. (2009). The double travelling salesman problem with multiple stacks – formulation and heuristic solution approaches. *European Journal of Operational Research*, 198(1):139 – 147.

Sampaio, A. H. e Urrutia, S. (2016). New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research*.