FORMULAÇÕES E ALGORITMOS EXATOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA E ENTREGA SOB MÚLTIPLAS PILHAS

ARMANDO HONORIO PEREIRA

FORMULAÇÕES E ALGORITMOS EXATOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM COLETA E ENTREGA SOB MÚLTIPLAS PILHAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais — Departamento de Ciência da Computação. como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Sebastián Alberto Urrutia

Belo Horizonte

Março de 2017

Pereira, Armando Honorio

P436f Formulações e algoritmos exatos para o problema do caixeiro viajante com coleta e entrega sob múltiplas pilhas / Armando Honorio Pereira. — Belo Horizonte, 2017

xx, 68 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais — Departamento de Ciência da Computação.

Orientador: Sebastián Alberto Urrutia

1. Computação — Teses. 2. Problema do caixeiro viajante. 3. Programação inteira. 4. Branch and Cut. I. Orientador. II. Título.

CDU 519.6*61(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Formulações e algoritmos exatos para o problema do caixeiro viajante com coleta e entrega sob múltiplas pilhas

ARMANDO HONORIO PEREIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. SEBASTIÁN ALBERTO URRUTIA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. DILSON LUCAS PEREIRA
Departamento de Ciência da Computação - UFLA

PROF. GERALDO ROBSON MATEUS
Departamento de Ciência da Computação - UFMG

PROF. RAFAEL AUGUSTO DE MELO
Instituto de Matemática - UFBA

Belo Horizonte, 31 de março de 2017.

Agradecimentos

Agradeço aos meus pais, Dilson e Catarina, pelo apoio durante meus estudos. Agradeço ao meu irmão, Lucas, pelas discussões sobre computação. Agradeço ao professor Sebastián, que além de orientador, foi um amigo. Finalmente, sou grato à CAPES pela bolsa.

"Não importa que apenas alguns em cada geração entendam e alcancem a realidade total da estatura apropriada ao Homem — e que o resto a traia. São esses poucos que movem o mundo e dão à vida seu significado — e é a esses poucos que eu sempre procuro me dirigir. O restante não me diz respeito; não é a mim ou a A Nascente *que eles traem: é às suas próprias almas."*

(Ayn Rand, A Nascente)

Resumo

No Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas, um único veículo deve atender a um conjunto de requisições de coleta e entrega. Cada requisição é definida por duas localizações, a localização de coleta, onde um item deve ser carregado no veículo, e a localização de entrega, onde o item previamente carregado deve ser entregue. Enquanto transportados, os items são armazenados em pilhas de capacidade limitada. Cada pilha deve seguir a política *Last-In-First-Out*. O que significa que somente o último item carregado em cada pilha está disponível para entrega. O objetivo do problema é encontrar uma rota para o veículo que atenda todas as requisições e que minimize a distância percorrida.

Nesta dissertação, propomos três novas formulações de programação inteira para o problema, junto com desigualdades válidas. A primeira formulação é uma reformulação de um modelo da literatura. A segunda formulação é um misto de dois modelos da literatura. A terceira formulação, até onde sabemos, é a primeira formulação compacta para o problema.

Propomos algoritmos *branch-and-cut* para essas novas formulações. Os algoritmos são testados em instâncias de *benchmark* e os resultados computacionais são comparados com a literatura. Instâncias para as quais uma solução ótima era conhecida previamente na literatura são resolvidas mais eficientemente nesse trabalho. Também, novos certificados de otimalidade são fornecidos para várias instâncias.

Palavras-chave: Roteamento de Veículos, Caixeiro Viajante, Restrições de Carregamento, Coleta e Entrega, Branch-and-Cut, Programação Inteira.

Abstract

In the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks, a single vehicle must fulfill a set of pickup and delivery requests. Each request is defined by two locations, the pickup location, where one item must be loaded in the vehicle, and the delivery location, where the previously loaded item must be delivered. While being transported, the items are stored in stacks with limited capacity. Each stack must follow the Last-In-First-Out policy, meaning that only the last item loaded in each stack is available for delivery. The objective of the problem is to find a vehicle route fulfilling all requests and minimizing the traveled distance.

In this dissertation, we propose three new integer programming formulations for the problem, along with valid inequalities. The first formulation is a reformulation of a model in the literature. The second formulation is a combination of two models in the literature. The third formulation, to the best of our knowledge, is the first compact formulation for the problem.

We propose branch-and-cut algorithms based on these new formulations. The algorithms are tested on benchmark instances and the computational results are compared with the literature. Instances for which an optimal solution was previously known in the literature are solved more efficiently in this work. Also, new optimality certificates are provided for various previously unsolved instances.

Keywords: Vehicle Routing Problem, Traveling Salesman, Loading constraints, Pickup and Delivery, Branch-and-cut, Integer Programming.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Empresas distribuidoras lidam diariamente com problemas de planejamento de suas rotas, tais como empacotamento de caixas no veículo, ordem de visitação dos clientes, e capacidade de armazenamento. Além disso, certos aspectos das rotas não podem ser negligenciados, por exemplo, distância ou duração máxima da rota, horário de funcionamento dos estabelecimentos e turnos dos funcionários. Ignorar essas restrições pode resultar em mudanças imprevistas, incorrendo em custos adicionais. Consequentemente, resolver o problema de roteamento e o de carregamento separados pode não ser suficiente. Grande parte dos problemas nessas duas áreas pertencem à NP-difícil e consequentemente difíceis de se resolver na prática. Por esse motivo, tradicionalmente, apesar de serem centrais na área de logística, essas duas áreas de pesquisas tem sido tratadas separadamente [Iori & Martello, 2010]. Recentemente, algoritmos combinando esses dois problemas tem sido propostos na literatura, segundo Pollaris et al. [2015], 60% das contribuições na literatura combinando essas duas áreas foram publicadas após 2009.

Devido à sua importância econômica e prática, um dos problemas mais estudados em Otimização Combinatória é o Problema de Roteamento de Veículos (*Vehicle Routing Problem* - VRP). No VRP, deve ser desenvolvido um conjunto de rotas para atender um conjunto de clientes com custo mínimo, geralmente dado pela distância total percorrida. Quando somente um veículo está disponível, o VRP se reduz a um problema arquétipo na área de roteamento de veículos, o Problema do Caixeiro Viajante (*Traveling Salesman Problem* - TSP), onde, dado um conjunto de cidades, junto com o custo de viajar entre cada par, o objetivo é encontrar uma rota de custo mínimo que visite todas elas e retorne à cidade inicial. O Problema de Roteamento de Veículos pode ser encontrado na literatura com janelas de tempo [Azi et al., 2007], restrições de distância [Kek et al., 2008], restrições de coleta e entrega [Dumitrescu et al., 2010],

entre outras.

Este trabalho trata do Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas (*Pickup and Delivery Traveling Salesman Problem with Multiple Stacks* - PDTSPMS). Nesse problema, um único veículo deve atender a um conjunto de requisições de clientes. Cada requisição é definida por duas localizações, a localização de coleta, onde um item deve ser carregado no veículo, e a localização de entrega, onde o item previamente carregado deve ser entregue. Enquanto transportados, os items são armazenados em pilhas de capacidade limitada. Cada pilha deve seguir a política *Last-In-First-Out* (LIFO). O que significa que somente o último item carregado em cada pilha está disponível para entrega. O objetivo do problema é encontrar uma rota para o veículo que atenda todas as requisições e que minimize a distância percorrida.

1.1 Justificativa

O Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas encontra aplicações onde a coleta e a entrega dos itens só podem ser realizadas pela parte traseira do veículo (e.g. carro furgão, caminhão cegonheira), dessa forma somente o último item carregado em uma pilha está disponível para entrega, e deve ser entregue antes de todos os outros. O fato de somente o último item estar disponível para entrega evita o problema de que, ao chegar em uma localização para realizar uma entrega, os itens que estão bloqueando o caminho da entrega correspondente sejam realocados dentro do veículo, uma vez que isso incorre tempo e gastos. Além disso, reposicionar os itens fora do depósito é oneroso, uma vez que os itens podem ser grandes, pesados, frágeis ou perigosos, por exemplo, eletrodomésticos, alimentos, objetos de vidro e porcelana. E, no caso onde é possível realocar os itens, estes devem ser pacotes pequenos, pois, caso sejam grandes, uma vez que o veículo está fora do depósito, talvez não haja equipamento ou local para mover os itens. Dentro desse contexto, é fundamental que o itinerário do veículo seja elaborado para evitar contratempos.

Os custos associados à operação dos veículos e à armazenagem dos itens formam uma importante componente dos gastos em sistemas de distribuição, dessa forma, uma pequena diminuição de custos pode resultar em uma economia significativa ao final do processo logístico. Consequentemente, o problema fornece uma fórmula para uma empresa transportadora reduzir os custos com sua frota e mão-de-obra.

Uma situação onde um único caminhão que coleta cargas similares de diferentes clientes espalhadas em uma região, e as entrega para clientes em uma outra região, é abordada por Ladany & Mehrez [1984]. O problema considerado pela empresa de fretes

é planejar a rota de um veículo que deve coletar cargas em Tel Aviv e entregá-las em Haifa. O caminhão utilizado só pode ser carregado pela porta traseira, e a área interna é estreita, dessa forma eles consideram o descarregamento baseado na sequência inversa do carregamento. Apesar de considerarem a política LIFO, eles também permitem o rearranjo dos itens. Assim, ao definir a função objetivo do problema, eles levam em conta o tempo de viagem, o tempo de carregamento e descarregamento, e o tempo gasto para rearranjar os itens.

O tempo para reposicionar os itens dentro do veículo depende do método de carregamento e rearranjo. De acordo com Ladany & Mehrez [1984], as três soluções mais comuns dentro desse contexto são: carregar as coletas inversamente à sequência de entregas, nesse caso nenhum rearranjo é necessário; planejar a rota desconsiderando o descarregamento, caso seja necessário, a carga é rearranjada na primeira localização da rota onde as caixas no caminhão não correspondem entrega a ser feita, nesse momento, o reposicionamento é feito de forma a satisfazer a sequência inteira de entregas; planejar a rota desconsiderando o descarregamento, e em cada ponto de entrega é feito um rearranjo parcial com as caixas que bloqueiam a que deve ser entregue. Ladany & Mehrez [1984] consideram somente os dois primeiros casos, uma vez que o terceiro, segundo os autores, é pior que o segundo método, já que aumenta a quantidade de rearranjos, e existe um custo de tempo associado com cada rearranjo adicional na rota.

Levitin & Abezgaouz [2003] também estudam uma aplicação real do problema. Eles consideram o caso onde as cargas em um armazém são colocadas em paletes, e cada novo palete coletado é colocado no topo do lote anterior carregado por um veículo guiado automatizado (VGA). Veículos guiados automatizados são amplamente usados em indústrias automatizadas para mover lotes entre diferentes locais. Para evitar o desperdício de tempo e espaço, a rota do VGA é planejada para previnir rearranjo dos lotes. Tais rotas seguem a política LIFO na qual o último palete carregado é entregue primeiro. Isto é feito com o intuito de evitar o uso excessivo de espaço e tempo para reorganizar os paletes em um lote.

Caso a carga destinada para uma estação de trabalho seja bloqueada por outras cargas, são necessários procedimentos de carga e descarga para rearranjá-las. Descarregar um palete arbitrário só é possível depois de descarregar todos os paletes acima dele, após isso, todos esses paletes devem ser carregados novamente no VGA. Além disso, esse procedimento exige um espaço para armazenar os paletes temporariamente descarregados, nessas operações também é dispendido tempo, o que poderia ser evitado caso a entrega estivesse no topo do veículo.

1.2 Contribuições

Nessa dissertação, resolvemos o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas de maneira exata através de algoritmos do tipo *branch-and-cut*. Propomos três novas formulações de programação inteira para o problema. A primeira formulação é uma reformulação de um modelo na literatura, em que um conjunto de varíaveis do modelo é eliminado, o que resulta também na eliminação de um conjunto de restrições. Com isso, obtemos mais rapidamente a solução da relaxação linear. A segunda formulação é uma combinação de dois modelos na literatura. E a terceira formulação, até onde sabemos, é a primeira formulação compacta para o problema. Também propomos desigualdades válidas para lidar com aspectos específicos do problema, como precedência, política LIFO, e capacidade. Finalmente, avaliamos os algoritmos propostos com instâncias de *benchmark* da literatura. Instâncias para as quais uma solução ótima era conhecida previamente são resolvidas mais eficientemente. Também, novos certificados de otimalidade são fornecidos para várias instâncias que estavam em aberto.

1.3 Organização

No desenvolvimento dos próximos capítulos, iremos citar o nome do problema em português e seu nome original em inglês, a sigla usada será correspondente ao nome em inglês, a escolha de não utilizar a sigla para o problema em português se deve ao fato de que as siglas para os problemas em inglês já são termos consolidados na literatura.

A dissertação é organizada da seguinte forma. No Capítulo 2, descrevemos o PDTSPMS, abordado nessa dissertação, no mesmo capítulo também apresentamos a literatura relacionada ao problema. Em seguida, no Capítulo 3, discutimos as formulações para o problema, começamos discutindo uma formulação para uma versão mais simples do problema, e ao final do capítulo, apresentamos nossas novas formulações para o problema. No Capítulo 4, descrevemos os algoritmos implementados nessa dissertação, e em sequência, no Capítulo 5, apresentamos os experimentos realizados e comparamos os algoritmos propostos com outros da literatura. Finalmente, no Capítulo 6, apresentamos as discussões finais e possíveis trabalhos futuros.

Capítulo 2

Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas

Neste capítulo, descrevemos o problema abordado nesta dissertação e fazemos uma revisão da literatura relacionada a ele. Inicialmente, definimos formalmente o problema e explicamos os aspectos relevantes em uma solução do mesmo, isto é, a precedência, política LIFO e a capacidade do veículo. Ainda, na seção de descrição do problema, analisamos características que podem ser utilizadas em novos algoritmos. Em seguida, apresentamos os problemas relacionados ao PDTSPMS na literatura e as estratégias utilizadas para resolvê-los.

2.1 Descrição do Problema

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 $V = P \cup D \cup \{0, 2n+1\}$ é o conjunto de vértices e $A = \{\, (i,j) \mid i,j \in V, i \neq j \,\}$ é o conjunto de arcos. Um conjunto de demandas com n requisições de clientes devem ser atendidas. Em cada requisição, uma coleta $i \in P$ está associada com uma entrega $n+i \in D$, e implica em coletar o item na localização i e entregá-lo na localização $n+i$. A imposição de visitar a localização i antes da localização $n+i$ recebe o nome de precedência. Cada arco $(i,j) \in A$ possui um custo $c_{ij} \geq 0$ associado, incorrido ao percorrer o arco. A cada ponto de coleta $i \in P$ está associado um item de tamanho $d_i \geq 0$, que deve ser

entregue no ponto $n+i \in D$. Considera-se que o ítem a ser entregue em $n+i$ tem tamanho $-d_i$. Itens de tamanho $d_0 = d_{2n+1} = 0$ são associados aos depósitos. Um único veículo, com um conjunto $M = \{1, \ldots, t\}$ de pilhas com capacidade $Q \geq 0$, onde cada item é carregado e descarregado seguindo a política LIFO, é utilizado para atender as requisições. A política LIFO impõe que cada item seja carregado no topo de uma pilha, e só pode ser descarregado se também estiver no topo. O objetivo do Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas é atender todas as requisições a custo mínimo, começando e terminando, respectivamente, nos depósitos 0 e $2n+1$, satisfazendo as restrições de precedência, capacidade e a política LIFO.

Definimos um caminho em G como uma sequência finita $W = v_0 e_1 v_1 e_2 v_2 \ldots e_k v_k$, onde os termos são alternadamente vértices e arestas, tal que, os nós terminais de uma aresta e_i, $0 < i \leq k$, são os vértices v_{i-1} e v_i, e todos os vértices v_i, $0 \leq i \leq k$, são distintos. Ao longo do texto, quando nos referirmos à uma rota, nos referimos a um caminho que se inicia no depósito 0, visita todos os vértices de $P \cup D$ e, termina no depósito final $2n+1$. Nesse caso, não fazemos nenhuma imposição da rota atender às restrições, portanto, quando utilizarmos somente o termo rota, essa rota pode violar a precedência, ou a política LIFO, ou a capacidade. Contudo, quando nos referirmos à uma rota **viável** para o PDTSPMS, essa rota atende todas as restrições do problema. Por clareza, iremos nos referir ao item coletado na localização i como item i.

Para formularmos o problema como um problema de Programação Linear Inteira, definimos $x_{ij} \in \{0,1\}$, $(i,j) \in A$, como uma variável binária indicando se j é visitado ou não imeditamente após i na rota. Temos que $x_{ij} = 1$ se a visita for realizada, 0, senão. Seja $x \in \mathbb{B}^{|A|}$, onde $\mathbb{B}^{|A|}$ é um conjunto de dimensão $|A|$ de todos os vetores binários, o vetor de incidência indicando a presença ou ausência de uma aresta na rota, ou seja $x_{ij} = 1$ se a aresta (i,j) pertence a rota, e $x_{i,j} = 0$, caso contrário.

Por úlitmo, seja $\mathcal{R} = \{\, x \in \mathbb{B}^{|A|} \mid x$ satisfaz a conectividade, precedência e, é possível atender a política LIFO e a capacidade $\}$ o conjunto dos vetores de incidência das rotas viáveis para o PDTSPMS. Por conectividade incluimos os fatos de que cada nó é visitado somente uma vez na rota, e que a rota não possui nenhuma sub-rota. A precedência implica que toda coleta i é visitada antes da sua entrega correspondente $n+i$. A política LIFO impõe que cada item deve ser carregado no topo de uma pilha e qualquer item entregue deve estar no topo de uma pilha. A capacidade implica que, em nenhuma localização da rota, a capacidade Q do veículo é excedida em cada uma de suas pilhas.

Denote por $\mathcal{P}$ o politopo do PDTSPMS determinado pelo fecho convexo dos vetores de incidência das viáveis rotas em $\mathcal{R}$. Formulamos o PDTSPMS como um

problema de Programação Linear Inteira da seguinte forma:

$$\left\{ \min \sum_{(i,j)\in A} c_{ij}x_{ij} \mid x \in \mathcal{P} \right\} \tag{2.1}$$

No Capítulo 3 mostramos como modelar o conjunto dos pontos inteiros em $\mathcal{P}$, ou seja, as restrições de conectividade, precedência, política LIFO, e capacidade.

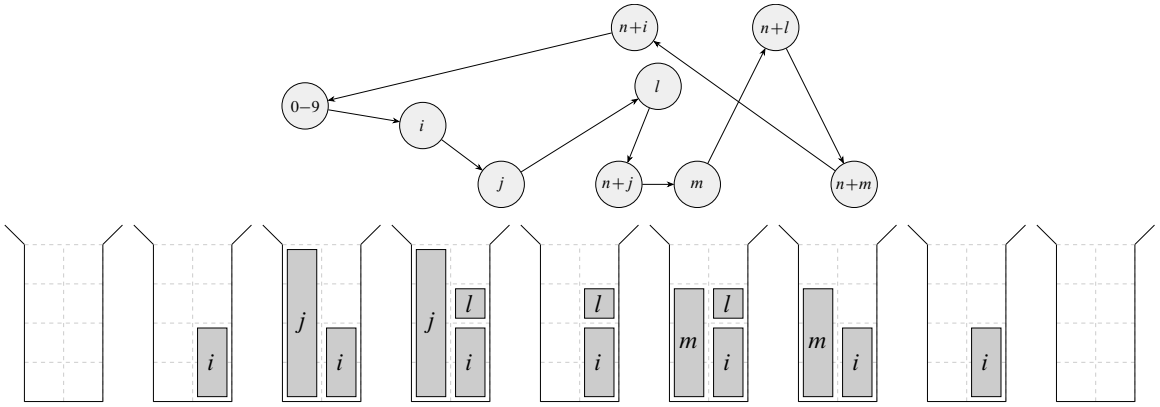

Figura 2.1: Diagrama descrevendo uma rota viável para o problema.

A Figura 2.1 exibe um exemplo de uma rota viável para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. Na figura, é exibida a traseira de um veículo vista verticalmente, as pilhas são delineadas por linhas tracejadas, junto com a rota percorrida pelo veículo. Na parte inferior, da esquerda para a direita, cada diagrama representa o estado da carroceria do veículo ao visitar cada uma das localizações da rota na parte superior. O veículo possui duas pilhas, cada uma de capacidade $Q = 4$, denominamos a primeira pilha, da esquerda para a direta, como pilha 1, e a segunda como pilha 2. A instância do problema possui $n = 4$ requisições, $P = \{i, j, l, m\}$, $D = \{n+i, n+j, n+l, n+m\}$, e depósitos inicial e final, respectivamente, 0 e 9, que são iguais nesse caso. Como descrito anteriormente, o item $i \in P$ é associado ao item $n+i \in D$, e.g., a coleta m é associada com a entrega $n+m$. Com relação ao tamanho dos itens, $d_i = 2$, $d_j = 4$, $d_l = 1$ e $d_m = 3$.

O veículo inicia a rota saindo do depósito 0, e visita, imediatamente, a coleta i, carregando o item na pilha 2. Em seguida, visita a coleta j, carregando seu item correspondente na pilha 1, após isto, o veículo coleta o item l e carrega na pilha 2, nesse momento, a pilha 1 do veículo está cheia e a pilha 2 possui somente 1 de capacidade livre, consequentemente, o veículo não pode coletar o item m restante, e deve realizar uma entrega. O item j é então entregue, e, logo após, o item m é carregado na pilha 1. Nesse instante, todas as coletas já foram visitadas pelo veículo, restando apenas

entregar os itens que ainda estão nas pilhas. O item l é entregue primeiro, depois o item m, e, finalmente, o item i. Nesse momento, o veículo desloca-se para o depósito final, que, nesse caso, é igual ao depósito inicial. Vale observar que, no PDTSPMS, o aspecto de carregamento e capacidade só considera uma dimensão.

2.2 Revisão da Literatura

De acordo com Iori & Martello [2010] e Pollaris et al. [2015], Ladany & Mehrez [1984] fizeram a primeira contribuição para o Problema do Caixeiro Viajante com Coleta e Entrega onde a coleta e a entrega seguem a política LIFO. Os autores investigam uma aplicação real do problema, que envolve uma empresa de fretes em Israel. Embora seja fornecida uma descrição do problema, nenhuma formulação matemática é dada. Os autores também são os primeiros a lidar com o problema de rearranjar os itens durante a rota, eles propõem um algoritmo de enumeração e resolvem instâncias pequenas, de até 3 requisições e 7 nós. Levitin & Abezgaouz [2003] também abordam uma aplicação real do problema, na qual os itens são colocados em paletes, e, cada novo palete é carregado no topo do lote anterior por um veículo guiado automatizado. Para evitar o uso excessivo de espaço e tempo ao rearranjar os paletes, os itens são transportados dentro do armazém de acordo com a política LIFO. No problema, eles consideram que o veículo possui capacidade ilimitada e desenvolvem um algoritmo exato capaz de resolver instâncias com até 100 nós.

O PDTSPMS é uma generalização do Problema do Caixeiro Viajante com restrições de carregamento LIFO (Pickup and Delivery Traveling Salesman Problem with LIFO loading constraints - PDTSPL). O PDTSPL é uma versão mais simples, onde o veículo contém somente uma pilha, de capacidade ilimitada. Um algoritmo exato *branch-and-bound* foi proposto por Carrabs et al. [2007a]. Cordeau et al. [2010b] introduziram três formulações de programação inteira para o PDTSPL, duas para o caso capacitado e uma para o não capacitado, apesar de fornecerem formulações para o caso capacitado, o problema principal estudado pelos autores é o não capacitado. Desigualdades válidas também são propostas e subsequentemente usadas em um algoritmo exato *branch-and-cut*. O algoritmo é capaz de resolver instâncias de até 43 nós. O PDTSPL também foi estudado por Carrabs et al. [2007b], os autores desenvolvem novos operadores de busca local e uma heurística *Variable Neighborhood Search* (VNS). Um estudo poliedral e um algoritmo *branch-and-cut* são apresentados em Dumitrescu et al. [2010]. Arbib et al. [2009] fornecem um modelo compacto para o PDTSPL, eles utilizam variáveis de torneio para modelar a relação de quem foi visitado primeiro en-

tre dois pares de vértices, eles avaliam a formulação com um conjunto de experimentos bem limitado, uma vez que usam um conjunto pequeno de instâncias. Li et al. [2011] propõem representar soluções do PDTSPL como árvores ao invés de listas. Eles desenvolvem uma heurística VNS para o problema e, além de proporem novos operadores, adaptam os operadores propostos por Carrabs et al. [2007b]. Utilizando a representação de árvore eles obtém resultados melhores do que os obtidos por Carrabs et al. [2007b].

Outra variante do PDTSPMS é o Problema do Caixeiro Viajante Duplo com Múltiplas Pilhas (Double Traveling Salesman Problem with Multiple Stacks - DTSPMS) introduzido por Petersen & Madsen [2009]. No DTSPMS todas as coletas devem ser executadas antes que qualquer entrega possa acontecer. Petersen & Madsen [2009] propõem um modelo matemático e metaheurísticas para o problema. Petersen et al. [2010] e Lusby et al. [2010] propõem abordagens exatas para o problema. Uma heurística denominada *Large Neighborhood Search* desenvolvida para o PDTSPMS é aplicada para o DTSPMS por Côté et al. [2012b]. Ángel Felipe et al. [2009] propõem para o DTSPMS quatro estruturas de vizinhança que são usadas por uma heurística VNS. A heurística anterior é melhorada por Ángel Felipe et al. [2011].

Øvstebø et al. [2011] abordaram um problema em navios roll-on/roll-off (RoRo) no transporte de veículos. Os navios RoRo considerados no problema possuem múltiplos conveses, e cada convés pode ser divido em faixas onde as cargas podem ser armazenadas seguindo a política LIFO, nesse caso, as faixas podem ser vistas como pilhas em um veículo. O aspecto de roteamento considerado é similar ao DTSPMS, o navio deve navegar entre duas regiões geográficas, carregando as cargas em um região e entregando em outra. Além do roteamento e o carregamento, janelas de tempo e restrições de estabilidade do navio são acopladas ao problema. No problema, também é permitido a realocação dos itens carregados no navio. Os autores fornecem um modelo de programação inteira mista, nele, a política LIFO é imposta através de restrições de fluxo. Øvstebø et al. [2011] propõem um método heurístico baseado em busca tabu e comparam os resultados àqueles obtidos pelo solver Xpress executando tal modelo.

Outras políticas de carregamento, além da política LIFO, podem ser encontradas na literatura, Carrabs et al. [2007a] consideram uma variante com carregamento *first-in-first-out* (FIFO) para o Problema do Caixeiro Viajante com Coleta e Entrega. Os autores propõem um algoritmo *branch-and-bound* para o problema, Erdoğan et al. [2009] apresentam duas heurísticas para o problema e Cordeau et al. [2010a] propõem um algoritmo *branch-and-cut*. Com relação a outros aspectos de carregamento, Iori et al. [2007] e Martínez & Amaya [2013] lidam com um problema de roteamanto onde o carregamento leva em conta duas dimensões, enquanto Junqueira et al. [2013] lidam

com um problema de carregamento tridimensional.

Côté et al. [2012a] propuseram o primeiro algoritmo exato para resolver o PDTSPMS. Eles forneceram três formulações de programação inteira e desigualdades válidas que são incorporadas no algoritmo *branch-and-cut* apresentado. Os melhores resultados foram obtidos com a formulação que estende para o caso em que há múltiplas pilhas as desigualdades propostas por Cordeau et al. [2010b] no contexto do PDTSPL. O algoritmo exato proposto também foi aplicado à instâncias do DTSPMS. Sampaio & Urrutia [2017] introduziram uma nova formulação de programação inteira para o PDTSPMS junto com um algoritmo exato *branch-and-cut* para resolver o problema. Eles também fornecem desigualdades válidas para o problema, junto com seus respectivos métodos de separação. Ambas as abordagens propostas por Côté et al. [2012a] e Sampaio & Urrutia [2017] tem resultados computacionais similares, resolvendo na otimalidade quase o mesmo subconjunto das instancias de *benchmark* da literatura.

Capítulo 3

Formulações para o PDTSPMS

Nesta seção apresentamos as formulações da literatura para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. Primeiramente formulamos um modelo clássico para o PDTSP, uma vez que essa formulação é estendida por todas as outras da literatura com um conjunto de restrições para englobar a política LIFO e a capacidade das pilhas dentro do veículo. Em seguida, apresentamos uma formulação para o PDTSPL, uma versão restrita do PDTSPMS onde o veículo possui apenas uma pilha. Finalmente, apresentamos as formulações de programação inteira para o PDTSPMS propostas por Côté et al. [2012a] e depois a formulação proposta por Sampaio & Urrutia [2017].

Após expor as formulações existentes para o problema, três novas formulações de programação inteira são apresentadas. A primeira, reformula o modelo proposto por Sampaio & Urrutia [2017] através da remoção de um conjunto de variáveis, a segunda, é uma combinação das formulações propostas por Côté et al. [2012a] e Sampaio & Urrutia [2017], a terceira, é uma formulação compacta para o problema, que, até onde sabemos, é a primeira formulação compacta para o problema.

3.1 Problema do Caixeiro Viajante com Coleta e Entrega

No Problema do Caixeiro Viajante com Coleta e Entrega (Pickup and Delivery Traveling Salesman Problem - PDTSP) um único veículo deve atender um conjunto de requisições de clientes, cada uma definida por uma localização de origem onde uma carga deve ser coletada, e um localização de destino onde a carga deve ser entregue. O problema consiste em determinar uma rota passando por todas as localizações garan-

tindo que toda coleta seja visitada antes de sua respectiva entrega. Segundo Cordeau et al. [2010b], na literatura, o problema pode ser encontrado tanto no caso capacitado quanto no não capacitado. Quando o problema é capacitado, existe um valor estabelecido que o veículo não pode violar em nenhum local da rota. Se for não capacitado, não é imposto nenhum limite à quantidade de carga que o veículo pode carregar. O problema encontra aplicações, por exemplo, em serviços de correios e transportes de idosos e deficientes [Cordeau et al., 2007].

Primeiramente, apresentamos uma formulação para o PDTSP, que é uma versão mais simples do nosso problema. Em seguida, mostramos como a formulação foi estendida na literatura para o PDTSPMS. Para a definição do modelo, nesta seção e nas seções seguintes, considere a seguinte notação matemática:

- $\overline{S} = V \setminus S$, onde $S \subseteq V$.
- $x(S) = \sum_{i,j \in S} x_{ij}$.
- $x(S,T) = \sum_{i \in S, j \in T} x_{ij}$.
- Denote por $\mathcal{S}$ o conjunto 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 $n+i \in S$.
- Por simplicidade de notação, iremos usar i quando nos referirmos ao conjunto $\{i\}$, onde $i \in V$.

O PDTSP é formulado como:

$$\min \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{3.1}$$

$$s.a. \quad x(i,V) = 1, \qquad i \in P \cup D \cup \{0\} \tag{3.2}$$

$$x(V,i) = 1, \qquad i \in P \cup D \cup \{2n+1\} \tag{3.3}$$

$$x(S) \leq |S| - 1, \qquad S \subseteq P \cup D, |S| \geq 2 \tag{3.4}$$

$$x(S) \leq |S| - 2, \qquad S \in \mathcal{S} \tag{3.5}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in A \tag{3.6}$$

A função objetivo (3.1) minimiza o custo total da rota. As restrições (3.2) e (3.3) garantem que cada vértice é visitado somente uma vez. As restrições (3.4) asseguram a conectividade da rota, eliminando sub-rotas. As restrições (3.5), propostas por Balas et al. [1995], no contexto do TSP com precedências, e por Ruland & Rodin [1997] no

contexto do PDTSP, 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. Elas proíbem qualquer caminho pelo conjunto $S \in \mathcal{S}$ tal que o veículo visita $n+i$ antes de sair de S, e visita i, após sair de S, consequentemente violando a precêdencia do problema.

3.2 Problema do Caixeiro Viajante com Coleta e Entrega sob Política LIFO

O PDTSPMS apareceu pela primeira vez na literatura em uma versão mais fraca, o PDTSPL, nessa versão, o veículo possui apenas uma pilha, que pode ter capacidade limitada ou não. Cordeau et al. [2010b] propõem três formulações de programação inteira para o problema, apesar dos autores focarem no problema não capacitado, duas das formulações também englobam o caso onde as pilhas possuem capacidade limitada. Todas as três formulações estendem a formulação (3.1)-(3.6) para englobar a política LIFO. Duas delas adicionam um número polinomial de variáveis e restrições à formulação anterior do PDTSP. A terceira exige um número exponencial de desigualdades, mas não necessita de nenhuma nova variável.

A primeira formulação proposta por Cordeau et al. [2010b] é descrita a seguir. Associe a cada nó $i \in V$, uma variável contínua Q_i representando a carga do veículo ao sair de i. O PDTSPL pode ser formulado adicionando o seguinte conjunto de restrições à formulação (3.1)-(3.6) do PDTSP:

$$Q_j \geq (Q_i + d_j)x_{ij}, \qquad (i,j) \in A \tag{3.7}$$

$$Q_{n+i} = Q_i - d_i, \qquad i \in P \tag{3.8}$$

$$\max(0, d_i) \leq Q_i \leq \min(Q, Q + d_i), \qquad i \in V \tag{3.9}$$

A consistência das variáveis Q_i é garantida com as restrições (3.7), a política LIFO é garantida através das restrições (3.8), e a capacidade com as restrições (3.9). Sobre as restrições (3.9) notamos que $d_j = -d_i$ se $j = n+i$. Caso o problema seja não capacitado, basta definir $d_i = 1$, $\forall i \in P$, $d_{n+i} = -1$, $\forall n+i \in D$, e $Q = n$. Apesar da formulação ser não linear devido as restrições (3.7), Cordeau et al. [2010b] mostram como linearizá-las. Eles criam uma constante $W_j = \min\{Q, Q + q_j\}$, obtendo:

$$Q_j \geq Q_i + q_j - W_j(1 - x_{ij}) \qquad i \in V, j \in V \tag{3.10}$$

O problema também pode ser modelado utilizando restrições de fluxo. ParaAssocie cada arco $(i, j) \in A$, com uma variável contínua f_{ij}, que representa a carga do veículo no arco (i, j). Um novo conjunto de restrições para o problema é:

$$\sum_{j \in V} f_{ij} - \sum_{j \in V} f_{ji} = d_i, \qquad i \in P \cup D \tag{3.11}$$

$$\sum_{j \in V} f_{ji} - \sum_{j \in V} f_{n+i,j} = 0, \qquad i \in P \tag{3.12}$$

$$\max(0, d_i, -d_j)x_{ij} \leq f_{ij} \leq \min(Q, Q + d_i, Q - d_j)x_{ij}, \qquad (i, j) \in A \tag{3.13}$$

As restrições (3.11) mantém a integridade da capacidade das pilhas, uma operação de coleta aumenta o fluxo e uma operação de entrega o diminui. A política LIFO é imposta através das restrições (3.12). De acordo com as restrições (3.13) o fluxo passando por um arco não pode, em nenhum momento, ser maior do que a capacidade do veículo. Como comentado pelos autores, essa formulação é inspirada na formulação de fluxo de um produto para o VRP, proposta por Gavish & Graves [1978].

As duas formulações anteriores adicionam um número polinomial de novas variáveis e restrições à formulação (3.1)-(3.6). O problema pode ser formulado sem a adição de novas variáveis, embora, para isto, seja preciso adicionar um número exponencial de desigualdades à formulação anterior. Seja Ω o conjunto de todos os subconjuntos $S \subset P \cup D$ tal que existe pelo menos uma localização $j \in P$, tal que $j \in S$ e $n + j \notin S$, ou uma localização $n + j \in D$, tal que $n + j \in S$ e $j \notin S$. A política LIFO é garantida pela seguinte desigualdade:

$$x(i, S) + x(S) + x(S, n + i) \leq |S|, \qquad S \in \Omega, i, n + i \notin S, i \in P \tag{3.14}$$

Para verificar que as restrições são válidas, considere inicialmente que S foi definido de forma que existe $j \in S$ e $n + j \notin S$. A restrição diz que não pode existir um caminho que sai de i, passa por S, e vai para $n + i$. Por outro lado, se existir $n + k \in S$ e $k \notin S$, a restrição também diz que não pode existir um caminho de i para $n + i$ passando por S. No primeiro caso, o veículo tentará entregar i antes de j, no segundo caso, entregar k antes de i, em ambas as situações, a política LIFO é violada. A Figura 3.1 exibe um exemplo dessa situação.

Cordeau et al. [2010b] demonstram que as desigualdes (3.14) são suficientes para impor a política LIFO. Côté et al. [2012a] adaptam as desigualdades (3.14) para o

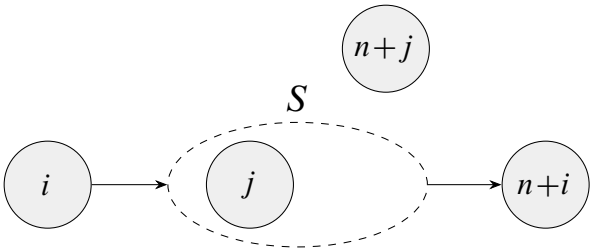

Figura 3.1: Exemplo de uma desigualdade (3.14) violada.

PDTSPMS, estendendo-as para o caso onde mais de uma pilha está disponível no veículo.

3.3 Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas

Nesta seção descrevemos as formulações e os algoritmos na literatura propostos para o PDTSPMS. Com o intuito de situar o leitor, seguimos uma ordem cronológica nas descrições. Começamos descrevendo a primeira abordagem exata para o problema e, em seguida, descrevemos a abordagem mais recente encontrada na literatura.

3.3.1 Primeiro Algoritmo Exato

O primeiro algoritmo exato para o PDTSPMS foi proposto por Côté et al. [2012a], os autores propõem três formulações diferentes para o problema. Eles também adaptam para o PDTSPMS desigualdades do VRP relacionadas à capacidade. Finalmente, propõem algoritmos *branch-and-cut* baseados nestas formulações, que incorporam as desigualdades válidas propostas para o problema. Antes de apresentarmos as formulações, utilizaremos a seguinte notação nas próximas exposições:

- $x(\delta^+(S)) = \sum_{i \in S, j \notin S} x_{ij}$, $S \subset V$, a soma das variáveis x_{ij} deixando S.
- $x(\delta^-(S)) = \sum_{i \notin S, j \in S} x_{ij}$, $S \subset V$, a soma das variáveis x_{ij} entrando em S.
- $x(\delta(S)) = x(\delta^+(S)) + x(\delta^-(S))$, $S \subset V$, a soma das variáveis no corte de S.
- $\pi(j) = j$, $j \in P$, a função π retorna o próprio nó se ele for uma coleta.
- $\pi(n + j) = j$, $n + j \in D$, a função π retorna a coleta correspondente se ele for uma entrega.

- $\pi(S) = \{\, i \in P \mid n+i \in S, S \subset V \,\}$, o conjunto de nós de coleta tal que seus nós correspondentes de entrega estão em S.
- $\sigma(S) = \{\, n+i \in D \mid i \in S, S \subset V \,\}$, o conjunto de nós de entrega tal que seus nós correspodentes de coleta estão em S.
- $S_j = \{\, S' \subset P \cup D \mid j \in S' \text{ e } n+j \notin S' \,\}$.
- $S_{n+j} = \{\, S' \subset P \cup D \mid n+j \in S' \text{ e } j \notin S' \,\}$.
- $i \prec j$, o veículo visita o nó i antes do nó j.

Para modelar o problema, considere a variável x_{ij} definida anteriormente e as seguintes variáveis:

- y_{ik}, $i \in P$, $k \in M$, que é igual a 1 se o item i é carregado na pilha k, 0, caso contrário.
- $0 \leq s_{ik} \leq Q$, $i \in V$, $k \in M$, é a carga da pilha k ao sair do nó i, onde $s_{0k} = 0$, $k \in M$.

Côté et al. [2012a] formulam o PDTSPMS adicionando o seguinte conjunto de restrições à formulação do PDTSP:

$$\sum_{k \in M} y_{ik} = 1, \qquad i \in P \tag{3.15}$$

$$s_{jk} \geq s_{ik} + d_j y_{\pi(j)k} - Q(1 - x_{ij}), \qquad i \in V, j \in P \cup D, k \in M \tag{3.16}$$

$$s_{jk} \leq s_{ik} + d_j y_{\pi(j)k} + Q(1 - x_{ij}), \qquad i \in V, j \in P \cup D, k \in M \tag{3.17}$$

$$s_{(n+j)k} \geq s_{jk} - d_j y_{jk} - Q(1 - y_{jk}), \qquad j \in P, k \in M \tag{3.18}$$

$$s_{(n+j)k} \leq s_{jk} - d_j y_{jk} + Q(1 - y_{jk}), \qquad j \in P, k \in M \tag{3.19}$$

$$s_{0k} = 0, \qquad k \in M \tag{3.20}$$

$$0 \leq s_{ik} \leq Q, \qquad i \in V, k \in M \tag{3.21}$$

$$y_{ik} \in \{0, 1\}, \qquad i \in P, k \in M \tag{3.22}$$

As restrições (3.15) indicam que cada item deve ser carregado em uma pilha. As restrições (3.16)-(3.17) definem a situação de cada pilha após uma coleta ou entrega. A política LIFO é imposta através das restrições (3.18)-(3.19). A capacidade é estabelecida pelas restrições (3.20)-(3.21). A seguir, duas formulações alternativas

são fornecidas, pois, como notado pelos autores, o uso das restrições (3.15)-(3.19) gera relaxações lineares fracas.

Côté et al. [2012a] estendem para o PDTSPMS a formulação (3.1)-(3.6), (3.11)-(3.13), e (3.15), proposta por Cordeau et al. [2010b] no contexto do PDTSPL, adicionando à cada variável de fluxo, um índice indicador da pilha. Portanto, seja f_{ij}^k o fluxo no arco $(i,j) \in A$ para a pilha k. A formulação fluxo é então dada por:

$$\sum_{j \in V} f_{ji}^k - \sum_{j \in V} f_{ij}^k = d_i y_{\pi(i)k}, \qquad i \in P \cup D \tag{3.23}$$

$$\sum_{j \in V} f_{ji}^k - \sum_{j \in V} f_{(n+i)j}^k \leq Q(1 - y_{ik}), \qquad i \in P, k \in M \tag{3.24}$$

$$\sum_{j \in V} f_{ji}^k - \sum_{j \in V} f_{(n+i)j}^k \geq -Q(1 - y_{ik}), \qquad i \in P, k \in M \tag{3.25}$$

$$0 \leq f_{ij}^k \leq Q, \qquad (i,j) \in A, k \in M \tag{3.26}$$

As restrições (3.23) lidam com a consistência de cada pilha, uma operação de coleta aumenta o fluxo, enquanto uma operação de entrega o diminui. As restrições (3.24)-(3.25) garantem a política LIFO, apontando que, a carga antes de atender o nó i deve ser igual a carga após servir o nó $n+i$. Por fim, a capacidade é garantida com as restrições (3.26). A formulação fluxo é definida por (3.1)-(3.6) e (3.23)-(3.26).

A terceira formulação proposta pelos autores não exige nenhuma variável adicional com relação à formulação fluxo, são adicionadas apenas novas restrições. Chamada formulação de caminho inviável, a formulação é inspirada no trabalho de Petersen et al. [2010] e elimina todos os caminhos que são inviáveis com relação à política LIFO ou à capacidade. A formulação é dada pelas restrições (3.1)-(3.6) do PDTSP e as restrições (3.27):

$$x(p) \leq |p| - 1, \qquad p \in \Phi \tag{3.27}$$

onde p é um caminho para o qual não existe nenhuma atribuição viável de pilhas que evite a violação da capacidade ou da política LIFO, $x(p)$ é a soma das variáveis x_{ij} associadas com os arcos do caminho, $|p|$ é a quantidade de arcos do caminho, e Φ denota o conjunto de todos os caminhos inviáveis que não possuem nenhuma atribuição viável de pilhas tal que satisfaça a capaciadade ou a politica LIFO.

Como o número de caminhos inviáveis com relação à capacidade e a política LIFO pode ser enorme, os autores relaxam essa restrição e checam por caminhos inviáveis

resolvendo um problema de empacotamento sempre que uma rota viável para o PDTSP for encontrada. Se o problema de empacotamento for inviável, eles adicionam uma restrição (3.27) para proibir o caminho. Note que, se o problema for viável, uma solução viável para o PDTSPMS foi obtida.

3.3.1.1 Problema de Empacotamento

Dado um caminho p que satisfaz as restrições de precedência de coleta e entrega, mas que não necessariamente satisfaz as restrições de capacidade e LIFO, esta seção apresenta o problema de empacotamento que pode ser resolvido para determinar se existe uma atribuição de itens às pilhas que faça com que as restrições de capacidade e LIFO sejam atendidas.

Sejam dois nós, i e j, dizemos que eles se cruzam se eles forem visitados de modo que a seguinte relação é satisfeita $i \prec j \prec n+i \prec n+j$, o que significa que i e j devem ser coletados em pilhas diferentes, uma vez que, se forem carregados na mesma pilha, a política LIFO será violada, pois j foi coletado depois de i, mas i está sendo entregue primeiro. Se essa ordem de visitação acontece para mais de dois itens, também dizemos que todos esses itens se cruzam, por exemplo, $i \prec j \prec k \prec n+i \prec n+j \prec n+k$, dizemos que os itens i, j, e k se cruzam. Observe que um item deve cruzar com todos os itens visitados depois dele, caso contrário, não dizemos que todos os itens se cruzam, na ordem de visitação $i \prec j \prec k \prec n+j \prec n+i \prec n+k$, os itens i e j não se cruzam.

Seja I o conjunto de pares de itens que se cruzam no caminho p, e seja O a lista de nós no caminho ordenada pela ordem de visitação dos mesmos. Defina o parâmetro binário a_{ij}, que assume valor 1 se o item i está no veículo quando o nó j é visitado, e 0, caso contrário. Também, defina a variável binária z_{ik}, que assume valor 1 se o item i é coletado na pilha k, e 0, caso contrário. O problema de empacotamento é definido como:

$$\sum_{k \in M} z_{ik} = 1, \qquad i \in P \tag{3.28}$$

$$z_{ik} + z_{jk} \leq 1, \qquad (i,j) \in I, k \in M \tag{3.29}$$

$$\sum_{i \in P} a_{io} d_i z_{ik} \leq Q, \qquad o \in O, k \in M \tag{3.30}$$

$$z_{ik} \in \{0,1\}, i \in P, k \in M \tag{3.31}$$

Assim como nas duas primeiras formulações, as restrições (3.28) associam cada item a uma única pilha. As restrições (3.29) impedem que itens na mesma pilha se

cruzem. Finalmente, as restrições (3.30) garantem a capacidade, assegurando que, a soma dos tamanhos dos itens que estão ao mesmo tempo na mesma pilha não deve ultrapassar a capacidade da mesma.

Toulouse & Wolfler Calvo [2009] e Casazza et al. [2012] mostram que o problema de empacotamento no DTSPMS, onde todas as coletas devem ser executadas antes de fazer qualquer entrega, é NP-difícil. Em vista disso, Côté et al. [2012a] observam que é improvável que um algoritmo polinimial para o problema exista.

3.3.1.2 Sobre as Desigualdades LIFO

As desigualdades (3.14) propostas em Cordeau et al. [2010b] são válidas para o PDTSPMS somente para o caso onde o veículo possui apenas uma pilha, tornando-se inválidas caso haja mais de uma pilha. As desigualdades (3.14) impedem que duas coletas i e j se cruzem, entretanto no PDTSPMS, pode existir cruzamentos válidos, visto que existe mais de uma pilha. Por exemplo, se o veículo possui duas pilhas, duas coletas podem se cruzar, uma vez que podem ser carregadas em pilhas diferentes, entretanto, não pode acontecer um cenário onde três itens se cruzam, uma vez que seria necessário três pilhas. Consequentemente, em um veículo com k pilhas, não pode existir solução viável em que $k+1$ itens se cruzam. A partir desse fato, Côté et al. [2012b] adaptaram as desiguldades (3.14) para o PDTSPMS.

Com o intuito de estender as desigualdades (3.14), considere um caminho $i_1, i_2, \ldots, i_{M+1}, n+i_1, n+i_2, \ldots, n+i_{M+1}$ em que todas as coletas se cruzam. Seja $S_{i_h} = \{i_h, i_{h+1}, \ldots, i_{M+1}, n+i_1, \ldots, n+i_{h-2}\}$ e, $2 \leq h \leq M+1$, onde, caso $h=2$, assume-se que $n+i_0$ é definido como i_{M+1} e a sequência de $n+i_1$ até $n+i_0 = i_{M+1}$ é vazia. Observe que, i_1 até i_{h-1} e $n+i_{h-1}$ até $n+i_{M+1}$ não pertecem ao conjunto S_{i_h}. Para todo $2 \leq h \leq M+1$, S_{i_h} satisfaz as condições do conjunto S sobre a qual (3.14) é definida. Considere a seguinte desigualdade, claramente válida, que adiciona 1 unidade ao lado direito de (3.14):

$$x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h) \leq \left|S_{i_{h+1}}\right| + 1 \tag{3.32}$$

Some esta desigualdade para todo S_{i_h}, $2 \leq h \leq M+1$:

$$\sum_{h=1}^{M} \left[x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n+i_h)\right] \leq \sum_{h=1}^{M} \left|S_{i_{h+1}}\right| + M \tag{3.33}$$

A única possibilidade de igualdade é se o caminho $i_1, i_2, \ldots i_{M+1}, n + i_1, n + i_2, \ldots, n + i_{M+1}$, tiver sido usado, como esse caso é inviável, (3.34) segue:

$$\sum_{h=1}^{M} \left[x(i_h, S_{i_{h+1}}) + x(S_{i_{h+1}}) + x(S_{i_{h+1}}, n + i_h)\right] \leq \sum_{h=1}^{M} \left|S_{i_{h+1}}\right| + M - 1 \tag{3.34}$$

As desigualdades (3.34) proíbem todos os caminhos de i_h para $n + i_h$ passando pelos nós i_{h+1} até i_{M+1} e $n + i_1$ até $n + i_{h-1}$, para $h = 1, \ldots, M + 1$. Um exemplo para $M = 2$ é exibido na Figura 3.2. As desigualdades (3.27) poderiam ser usadas para eliminar esse caminho, entretanto a abordagem é fraca pois elas proibiriam somente esse caminho. A seguir demonstramos a validade das desigualdades.

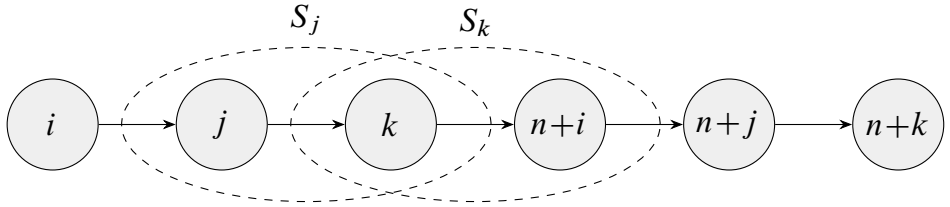

Figura 3.2: A desigualdade (3.34) gerada para o diagrama da figura é $x(i, S_j) + x(S_j) + x(S_j, n + i) + x(j, S_k) + x(S_k) + x(S_k, n + j) \leq |S_j| + |S_k| + 1$. Ela impede todos os caminhos de i para $n + i$ por j e k, e ao mesmo tempo, proíbe todos os caminhos de j para $n + j$ através de k e $n + i$.

Nas desigualdades (3.34) cada termo no somatório serve o propósito de garantir que os itens i_h e i_{h+1} se cruzam, ou seja, existe um caminho $i_h \rightsquigarrow S_{i_{h+1}} \rightsquigarrow n + i_h$. Consequentemente, em uma desigualdade violada, com todos os termos ativos, garante-se que $M+1$ itens se cruzam, uma vez que temos i_1 e i_M, respectivamente, no primeiro e último somatórios, e $(2M - 2)/2$ coletas nos somatórios intermediários (cada coleta aparece duas vezes no somatório), somando todas coletas anteriores, temos $2+M-1 = M + 1$ coletas cruzando-se entre si.

Em mais detalhes, $x(i_h, S_{i_{n+1}})$, da coleta visitada anteriormente, o veículo visita o conjunto que contém a próxima coleta, $x(S_{i_{h+1}})$, todos em $S_{i_{h+1}}$ são visitados antes de sair do conjunto, formando um caminho, $x(S_{i_{h+1}}, n + i_h)$, o veículo sai de $S_{i_{h+1}}$ diretamente para $n + i_h$, garantido que os itens i_h e i_{h+1} se cruzam. Os nós i_1 até i_h e $n + i_h$ até $n + i_{M+1}$ não pertecem ao conjunto $S_{i_{h+1}}$ exatamente para garantir o padrão de visitação em que os itens se cruzam, uma vez que i_1 até i_h tem que ter sido visitados anteriormente, e $n + i_h$ até $n + i_{M+1}$ não podem ser visitados entre i_h e i_{h+1}, especialmente $n + i_h$ e $n + i_{h+1}$, uma vez que se eles pertencessem a $S_{i_{h+1}}$, os itens i_h e i_{h+1} poderiam ser entregues.

Finalmente, $i_h, \ldots, i_{M+1} \in S_{i_h}$, para garantir que todas as coletas que se cruzam sejam visitadas antes de qualquer entrega. Depois de visitar i_h, o veículo ainda tem que visitar de i_{h+1} até i_{M+1} antes de fazer qualquer entrega. Além disso, somente $n+i_1, \ldots, n+i_{h-2} \in S_{i_h}$, para garantir que todos os itens se cruzam, pois se $n+i_{h-1}, n+i_h \in S_{i_h}$, poderíamos entregar os itens i_{h-1} e i_h, quebrando o fato de se cruzarem, o mesmo argumento é usado com relação à $n + i_{h-1}, \ldots, n + i_{M+1} \notin S_{i_h}$, uma vez que $i_h, \ldots, i_{M+1} \in S_{i_h}$, e novamente, o fato de cruzarem seria invalidado. Finalmente, se qualquer entrega $n + i_k$, $1 \leq k \leq M + 1$ for visitada entre $i_1, \ldots, i_{M+1}$, a desigualdade não será violada, caso $n + i_k$ seja visitado entre duas coletas, ele será visitado dentro de um conjunto $S_{i_{h \geq k+2}}$, consequentemente, $x(S_{i_{k+1}}, n + i_k) = 0$, e a desigualdade não será violada. Por exemplo, se $n + i_1$ é visitado em S_{i_3}, teremos $x(S_{i_2}, n + i_1) = 0$. Se uma coleta i_k, $2 \leq k \leq M + 1$, for visitada antes de uma coleta i_l, $l < k$, temos que $S_{i_k} < |S_{i_k}| - 1$, uma vez que o veículo não formará um caminho por S_{i_k}, já que i_k foi visitado antes de entrar no conjunto, e, novamente, a desigualdade não será violada. Caso uma entrega $n + i_k$, $2 \leq k \leq M + 1$, seja visitada antes de uma entrega $n + i_l$, $l < k$, o veículo o faz de um conjunto $S_{i_{h \geq k+2}}$, consequentemente, $x(S_{i_{k+1}}, n + i_k) = 0$, uma vez que $k + 1 < k + 2$, e novamente a desigualdade não será violada, visto que os itens deixam de se cruzar.

3.3.2 Segundo Algoritmo Exato

Sampaio & Urrutia [2017] propuseram o segundo algoritmo exato para o PDTSPMS. O algoritmo exato proposto pelos autores utiliza uma abordagem *branch-and-cut*, e uma formulação que evita a solução de um problema de empacotamento sempre que uma rota satisfazendo à precedência for encontrada, diferente do algoritmo *branch-and-cut* proposto por Côté et al. [2012a]. Sampaio & Urrutia [2017] utilizam como base a formulação (3.1)-(3.6) para o PDTSP, e introduzem novas variáveis binárias para indicar a pilha usada pelo veículo em um dado arco da rota. Com o uso das novas variáveis, eles propõem um conjunto de restrições para modelar a política LIFO e a capacidade do veículo.

Seja y_{ij}^k uma variável binária tal que, y_{ij}^k é igual a 1 se, após visitar a localização $i \in V$, o veículo coleta ou entrega o item j da pilha k, e 0, caso contrário. O objetivo das variáveis é ter a informação de qual pilha foi manipulada em um dado arco. As varíaveis y_{ij}^k são ligadas às variáveis x_{ij} da seguinte forma:

$$x_{ij} = \sum_{k \in M} y_{ij}^k, \qquad (i, j) \in A \tag{3.35}$$

Como as variáveis y_{ij}^k modelam tanto a operação de coleta quanto a de entrega, a pilha em que um item i for coletado deve ser a mesma utilizada para entregá-lo:

$$\sum_{i \in V \setminus \{2n+1\}} y_{ij}^k = \sum_{i \in V \setminus \{0,2n+1\}} y_{i,n+j}^k, \qquad j \in P, k \in M \tag{3.36}$$

A desigualdade utilizada para impor a política LIFO é similar à desigualdade (3.14) proposta no contexto do PDTSPL. Se o veículo coleta um item i na pilha k, visita um conjunto S, sem visitar vértices externos, coletando um item j, imediatamente após entrar em S, também na pilha k, tal que a entrega $n+j$ não está em S, e deixa o conjunto S diretamente pelo nó $n+i$, a política LIFO é violada, uma vez que j foi carregado após i, entretanto i foi entregue primeiro. Baseado na notação descrita previamente, seja $y^k(S,T) = \sum_{i \in S, j \in T} y_{ij}^k$, onde $k \in M$, e $S, T \subset V$. A política LIFO é imposta com as restrições:

$$y^k(\overline{S}, j) + x(S) + y^k(S, n+i) \leq |S|, \qquad S \subset P \cup D, i, n+i, n+j \notin S, j \in S, k \in M \tag{3.37}$$

As restrições (3.37) modelam a política LIFO dentro do cenário descrito no parágrafo anterior. Note que $y^k(S, n+i)$ é usado para indicar que o item i foi coletado na pilha k, uma vez que a pilha usada na coleta é idêntica à pilha usada na entrega.

Para impor a restrição de capacidade, Sampaio & Urrutia [2017] utilizam uma desigualdade similar às desigualdades (3.27), com a adição das variáveis y_{ij}^k para indicar as pilhas manipuladas pelo veículo no caminho. Seja w^k um caminho que viola a capacidade Q do veículo na pilha k, essa violação é proibida com:

$$y^k(w_1^k) + x(w_0^k) \leq |w| - 1, \qquad k \in M, (w^k, w_1^k) \in \Psi \tag{3.38}$$

onde $y^k(w_1^k)$ é a soma das variáveis y_{ij}^k para os arcos que usam a pilha k, $x(w_0^k)$ a soma das variáveis x_{ij} relacionadas aos outros arcos, $|w|$ o tamanho do caminho em arcos, e Ψ o conjunto de todos os pares de caminhos e subconjuntos de operações de coletas no caminho tal que se todas as coletas em w_1^k forem realizadas na mesma pilha, a capacidade do veículo é excedida.

O PDTSPMS é formulado pelo modelo consistindo das restrições (3.1)-(3.6) e (3.35)-(3.38).

3.4 Novas Formulações para o PDTSPMS

Nesta seção, apresentamos as novas formulações para o PDTSPMS propostas neste trabalho. A primeira formulação modifica o modelo de programação inteira proposto por Sampaio & Urrutia [2017], devido à uma observação que será descrita a seguir. Com o objetivo de obter um modelo cuja relaxação linear pode ser mais rapidamente resolvida, um conjunto de variávies é eliminado, consequentemente um conjunto de restrições do modelo também é eliminado. A segunda formulação utiliza as idéias das formulações propostas por Côté et al. [2012b] e Sampaio & Urrutia [2017], as restrições propostas por Sampaio & Urrutia [2017] para modelar a política LIFO e a capacidade, são adaptadas utilizando as variáveis de pilha apresentadas por Côté et al. [2012b]. A terceira formulação é um modelo de fluxo compacto para o problema.

3.4.1 Formulação Arco

Nesta seção, apresentamos uma formulação que reduz o número de variáveis da formulação proposta por Sampaio & Urrutia [2017]. Como descrito na seção anterior, naquele artigo, novas variáveis binárias $y_{ij}^k, (i,j) \in A$ e $k \in M$, foram introduzidas, indicando a pilha $k \in M$ usada quando o veículo visitou j vindo de i. Neste caso, temos $y_{ij}^k = 1$ se, após visitar a localização $i \in V$, o veículo usou a pilha k para carregar ($j \in P$) ou entregar ($j \in D$) o item j, e $y_{ij}^k = 0$, caso contrário.

Na formulação de Sampaio & Urrutia [2017], as variáveis y_{ij}^k existem para cada $j \in P \cup D$. Entretanto, uma vez que a pilha usada para carregar e descarregar cada par de coleta e entrega é a mesma, de posse da informação que o item é carregado na pilha k, podemos inferir que ele será entregue da mesma pilha. Consequentemente, nós podemos eliminar as variáveis y_{ij}^k com $j \in D$. Neste trabalho, usamos esse fato para reformular o modelo proposto por Sampaio & Urrutia [2017] eliminando as variáveis y_{ij}^k onde $j \in D$. Com a eliminação, temos que o valor de $y_{i,n+j}^k$ é dado por:

$$y_{i,n+j}^k = \left\lfloor \frac{x_{i,n+j} + \sum_{u \in V \setminus \{2n+1\}} y_{uj}^k}{2} \right\rfloor, \qquad i \in P \cup D, j \in P, k \in M \tag{3.39}$$

A função piso é usada porque o item j pode ter sido carregado usando a pilha k enquanto o arco $(i, n+j)$ não é usado, portanto, o numerador na Equação (3.39) será 1, resultando em um valor fracionário, embora $y_{i,n+j}^k = 0$. O mesmo argumento se aplica para $x_{i,n+j}$ quando o arco é usado ao passo que a pilha k não é.

Na formulação proposta por Sampaio & Urrutia [2017], o número total de variáveis

y_{ij}^k é $(4n^2-n)t-2n(t-1)$, que pelas observações anteriores, é reduzido para $(2n^2-n)t$ neste trabalho, onde n é o número total de requisições e t o número de pilhas. A seguir exibimos em detalhes o cálculo do número de variáveis no modelo de programação inteira proposto por Sampaio & Urrutia [2017]. Considerando os conjuntos P e D temos $2n(2n-1) = 4n^2 - 2n$ arcos direcionados. Somando os arcos de 0 para P e de D para $2n+1$ temos $2n$ arcos. Removendo os arcos $(n+i, i)$ temos $-n$ arcos. Somando esses valores temos $4n^2 - n$, como existe uma variável de pilha para cada arco, a multiplicação resulta em um total de $(4n^2 - n)t$. Entretanto, para eliminar a simetria relacionada às pilhas do problema, Sampaio & Urrutia [2017] usam somente uma pilha para arcos saindo de 0 e arcos entrando em $2n+1$, como temos $2n$ arcos no total saindo e entrando nos dois, temos de subtrair $2n(t-1)$, o que resulta em um total de $(4n^2-n)t-2n(t-1)$. Um cálculo similar é usado para a nossa reformulação do problema, mas considerando somente arcos que vão para P. Por exemplo, para um problema com 21 coletas e 3 pilhas, temos 5145 variáveis y_{ij}^k no modelo sem a reformulação, e, 2583 variáveis y_{ij}^k no modelo reformulado.

As variáveis y_{ij}^k são ligadas às variáveis x_{ij}, que indicam se o arco $(i,j) \in A$ é usado ou não, através da restrição:

$$x_{ij} = \sum_{k \in M} y_{ij}^k, \qquad i \in \{0\} \cup P \cup D, j \in P \tag{3.40}$$

Além da redução do número de variáveis y_{ij}^k, o número de restrições de ligação (3.35) entre as variáveis x_{ij} e y_{ij}^k também é reduzido com as novas restrições (3.39). Nas próximas seções exibimos as novas restrições para lidar com a política LIFO e a capacidade da pilha na nova formulação, visto que as restrições (3.37) e (3.38) não se aplicam mais.

3.4.1.1 Política LIFO

Como o intuito de mostrar como a política LIFO pode ser violada em uma rota, considere duas localizações 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 visitar i, percorrer o conjunto S sem deixá-lo e imediatamente depois visitar $n+i$, então a política LIFO é violada, uma vez que j é carregado depois de i, mas $n+i$ é visitado antes de $n+j$. Uma representação gráfica dessa situação é mostrada na Figura 3.3. Entretanto, se o veículo tivesse mais de uma pilha, o mesmo caminho poderia ser viável, uma vez que i e j podem ser carregados em pilhas diferentes. Com esse contexto em mente, as variáveis y são introduzidas para especificar a pilha usada

para carregar cada item.

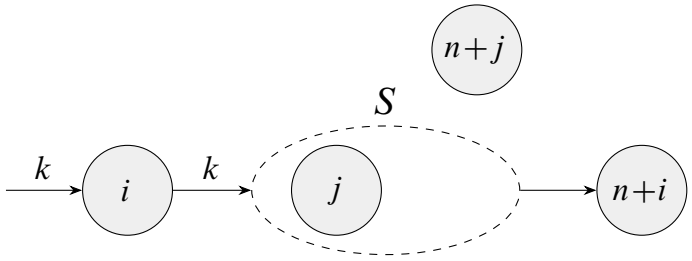

Figura 3.3: Caminho inviável considerando somente uma pilha.

A política LIFO é imposta através da desigualdade (3.41).

$$\begin{aligned} y^k(\overline{S}, i) + y^k(\overline{S}, j) + x(S) + x(S, n+i) \leq |S| + 1, \\ S \subset P \cup D, j \in S, i, n+i, n+j \notin S, k \in M \end{aligned} \tag{3.41}$$

A desigualdade proíbe cada caminho de i para $n+i$ tal que o item na localização 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 localizações externas ($x(S)$), e imediatamente depois de visitar S, a localização $n+i$ é visitada ($x(S, \{n+i\})$). Observe que somente caminhos de i para $n+i$ através de S com operações inviáveis de pilha são removidos do espaço de solução. Note também que consideramos j como a primeira localização visitada em S. Embora a desigualdade seja válida para o conjunto S considerando todas as localizações entre i e j nós optamos por considerar j como a primeira localização para fortalecer as desigualdades. Sampaio & Urrutia [2017] explicam o procedimento usado para fortalecer as desigualdades reduzindo o tamanho de S.

3.4.1.2 Restrições de Capacidade

Em adição à política LIFO, o veículo também deve respeitar a capacidade de cada uma de suas pilhas. Antes de formalmente definir as desigualdades responsáveis por impor essa restrição, iremos descrever os conjuntos usados em sua definição. Seja $a = \{ a_i^k \mid i \in P, k \in M \}$ uma atribuição de pilhas para os nós de coleta, tal que, a_i^k é um parâmetro que indica se o item na localização $i \in P$ é carregado na pilha $k \in M$, $a_i^k = 1$, ou não, $a_i^k = 0$. Seja $h = v_0 e_1 v_1 \dots v_{m-1} e_m v_m$ um caminho visitado pelo veículo seguindo à atribuição a, tal que, quando v_0 é visitado, a pilha k está vazia, e quando o veículo visita v_m, a capacidade da pilha k é execedida. Seja $\mathcal{H}(k)$ o conjunto de todos os possíveis pares (h, a). Defina $A(h)$ como o conjunto de arcos de h, e seja $\mathcal{Y}(h, k) = \{ (i, j) \mid (i, j) \in A(h), a_j^k = 1 \}$ o conjunto de arcos do caminho h, tal que, o nó j é carregado na pilha k.

As desigualdades responsáveis por prevenir à violação de capacidade são definidas como:

$$\sum_{(i,j)\in A(h)\setminus\mathcal{Y}(h,k)} x_{ij} + \sum_{(i,j)\in\mathcal{Y}(h,k)} y_{ij}^{k} \leq |A(h)| - 1, \qquad k \in M, (h,a) \in \mathcal{H}(k) \tag{3.42}$$

As desigualdades (3.42) afirmam que, se o veículo visitar o caminho h realizando as operações de pilha em a, a capacidade da pilha k será violada.

Um exemplo de um caminho inviável é mostrado na Figura 3.4. Suponha que $Q = 2$ e $d_i = 1$ para cada item i, o rótulo k em um arco (i, j) indica que o item j foi carregado na pilha k, ou seja $a_j^k = 1$. Considerando que o veículo tem a pilha $k = 2$ vazia quando ele visita o nó u, no fim do caminho a capacidade da pilha k é excedida. No caminho da Figura 3.4 temos $A(h) \setminus \mathcal{Y}(h,k) = \{(t, n+u), (n+u, n+t)\}$ e $\mathcal{Y}(h,k) = \{(u,j), (j,t), (n+t,m), (m,i)\}$. A desigualdade do tipo (3.42) resultante é $y_{uj}^k + y_{jt}^k + x_{t,n+u} + x_{n+u,n+t} + y_{n+t,m}^k + y_{mi}^k \leq 5$.

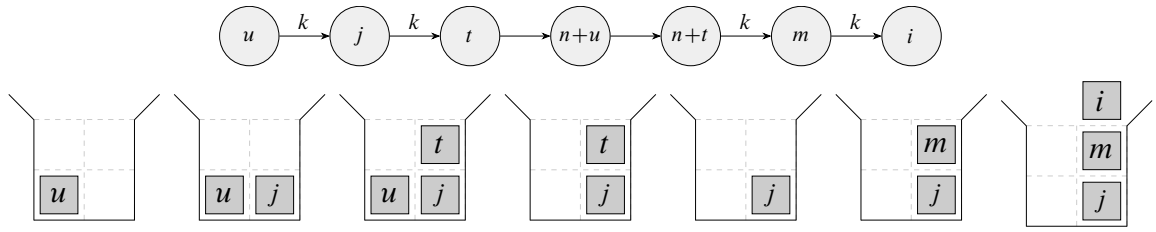

Figura 3.4: Caminho que excede a capacidade da pilha $k = 1$, onde $Q = 2$ e $d_i = 1$ todo item i.

A nova formulação para o PDTSPMS é dada pelo modelo definido pelas restrições (3.1)-(3.6), (3.40)-(3.42) e (3.43).

$$y_{ij}^{k} \in \{0,1\}, \qquad i \in \{0\} \cup P \cup D, j \in P, k \in M \tag{3.43}$$

3.4.1.3 Desigualdades Válidas

Nessa seção, apresentamos desigualdades válidas para a formulação descrita anteriormente. Sampaio & Urrutia [2017] propuseram duas famílias de desigualdades para o problema, Desigualdades de Sucessor (*Sucessor Inequalities*) e Restrições de Eliminação de Sub-rota de Capacidade (*Capacity Subtour Elimination Constraints*). Entretanto, a versão das Restrições de Eliminação de Sub-rota de Capacidade baseada em entregas e as Desigualdades de Sucessor utilizam variáveis y_{ij}^k com $j \in D$. A versão das Restrições de Eliminação de Sub-rota de Capacidade baseada em coletas não necessita de

adaptação nenhuma e é herdada diretamente, uma vez que ela só utiliza variáveis y_{ij}^k com $j \in P$. As Restrições de Eliminação de Sub-rota de Capacidade baseadas em coletas foram testadas dentro do algoritmo *branch-and-cut* mas não trouxeram nenhuma melhoria nos limites obtidos na raiz e nem quando empregadas no *branch-and-bound* como um todo, por isso, decidimos não utilizá-las, e consequentemente não às expomos.

3.4.1.4 Desigualdades de Sucessor

Cordeau et al. [2010b] propuseram as desigualdades de sucessor para o PDTSPL e Sampaio & Urrutia [2017] adaptaram-nas para o PDTSPMS. Devido ao uso das variáveis que foram eliminadas, propomos uma variação das Desigualdades de Sucessor, ela é um caso especial das desigualdades (3.41) quando S contém somente j, nesse caso, ao fixarmos a pilha $k \in M$, o número de desigualdades é quadrático em V.

Considere uma rota onde o veículo carrega o item i na pilha k e mais tarde visita a localização j, também carregando-o na pilha k, e imediatamente após o veículo visita $n+i$ para entregar o item i. Entretanto, sabemos que isso é impossível, uma vez que i foi carregado antes na pilha k. Não é viável visitar uma localização de coleta, carregar seu item em uma pilha, e tentar entregar logo em seguida um item que foi armazenado anteriormente na mesma pilha, uma vez que esse procedimento viola a política LIFO. Essa classe de violações pode ser previnida na nossa formulação através das seguintes desigualdades:

$$\sum_{v \in \{0\} \cup P \cup D} y_{vi}^k + \sum_{v \in \{0\} \cup P \cup D} y_{vj}^k + x_{i,n+j} \leq 2, \qquad i \neq j \in P, k \in M \tag{3.44}$$

3.4.2 Formulação Nó

Na formulação seguinte, ao invés de associar a pilha usada em uma dada localização com os arcos incidindo no nó correspondente, nós associamos a pilha usada com o próprio nó. Para esse fim, definimos as variáveis de decisão z_i^k, igual a 1 se o item i é carregado na pilha k, e 0, caso contrário. Note que Côté et al. [2012a] define o mesmo conjunto de variáveis. Nossa formulação difere da deles na forma que lidamos com as restrições de capacidade e LIFO. Côté et al. [2012a] não usam as variáveis z_i^k ao definir desigualdades para caminhos inviáveis em LIFO ou capacidade.

Com o novo conjunto de variáveis, impomos que cada item seja carregado em

uma pilha através das restrições (3.45):

$$\sum_{k \in M} z_i^k = 1, \qquad i \in P \tag{3.45}$$

Como descrito anteriormente, se o veículo carrega primeiro o item i na pilha k, percorre um conjunto S sem visitar vértices externos e também coleta o item j na pilha k, e visita a localização $n+i$ antes da localização $n+j$ ao sair de S, então a política LIFO é violada. Portanto, podemos impor a política LIFO com o novo conjunto de variáveis de uma maneira similar ao que é feito nas desigualdades (3.41), obtendo as desigualdades (3.46):

$$\begin{aligned} z_i^k + z_j^k + x(\overline{S}, j) + x(S) + x(S, n+i) \leq |S| + 2, \\ S \subset P \cup D, j \in S, i, n+i, n+j \notin S, k \in M \end{aligned} \tag{3.46}$$

A desigualdade (3.46) proíbe todas as soluções que coletam ambos i e j na pilha k, visita j, percorre S sem deixá-lo e visita a localização $n+i$ imediatamente depois.

O veículo também deve satisfazer as restrições de capacidade. Na nossa formulação, eliminamos todos os caminhos que violam esse aspecto. Usando a notação descrita anteriormente na Seção 3.4.1, considere o par $(h, a) \in \mathcal{H}(k)$, onde a capacidade da pilha k é violada se o caminho h for visitado realizando as operações em a. Defina $N(h)$ como o conjunto de nós no caminho h, e seja $\mathcal{Z}(h,k) = \{\, i \mid i \in N(h), a_i^k = 1 \,\}$ o conjunto de coletas no caminho h, cujos itens são carregados na pilha k de acordo com as atribuições em a. A restrição de capacidade é imposta através das desigualdades (3.47).

$$\sum_{(i,j) \in A(h)} x_{ij} + \sum_{v \in \mathcal{Z}(h,k)} z_v^k \leq |A(h)| + |\mathcal{Z}(h,k)| - 1, \qquad k \in M, (h,a) \in \mathcal{H}(k) \tag{3.47}$$

As desigualdades (3.47) proíbem todas as soluções que visitam o caminho h e carregam na pilha k todos os itens em $\mathcal{Z}(h,k)$. A Formulação Nó é definida pelas restrições (3.1)-(3.6), (3.45)-(3.47) e (3.48).

$$z_i^k \in \{0, 1\}, \qquad i \in P, k \in M \tag{3.48}$$

3.4.2.1 Desigualdades Válidas

Nessa seção apresentamos famílias de desigualdades válidas para a formulação. A primeira desigualdade é uma adaptação das desigualdades de sucessor apresentadas na

formulação anterior. A segunda família de desigualdades é o resultado de um fortalecimento feito na restrições que impõem a política LIFO. Nesse caso outros itens em S são levados em conta na violação da política LIFO. A terceira desigualdade é uma versão fortalecida da restrição de caminho utilizada para impor a capacidade na rota, ao invés de proibir somente um caminho, todos os caminhos por S que utilizam uma atribuição de pilhas inviável para as coletas é proibido.

3.4.2.2 Desigualdades de Sucessor

Como descrito na Seção 3.4.1.4, o veículo não pode visitar uma localização de coleta, carregar seu item em uma pilha, e logo em seguida, tentar entregar um item que foi armazenado anteriormente na mesma pilha, uma vez que esse procedimento viola a política LIFO. Utilizando as variáveis z_i^k essa classe de violações pode ser proibida através da seguinte desigualdade:

$$z_i^k + z_j^k + x_{i,n+j} \leq 2, \qquad i \neq j \in P, k \in M \tag{3.49}$$

3.4.2.3 Desigualdades LIFO fortalecidas

As restrições (3.46) impõe a política LIFO entre cada par de vértices de coletas i e j em um caminho de j para $n+i$ através de um conjunto $S \subset P \cup D$, onde i, $n+i$, $n+j \notin S$, e $j \in S$. Mas, note que, se carregarmos na pilha k qualquer item de coleta u dentro de S, cuja entrega correspondente $n+u$ está fora de S, a política LIFO é violada entre os nós i e u. Portanto, através de uma versão fortalecida da desigualdade (3.46), podemos também eliminar outras violações da política LIFO, entre o nó i e outros nós de S, além de j. Seja $\mathcal{A} = \{\, u \mid u \in S, n+u \notin S \,\}$ um conjunto de coletas cujas entregas correspondentes estão fora de S, as desigualdades LIFO fortalecidas são definidas da seguinte forma:

$$z_i^k + \frac{\sum_{u \in \mathcal{A}} z_u^k}{|\mathcal{A}|} + x(\overline{S}, j) + x(S) + x(S, n+i) \leq |S| + 2, \qquad S \subset P \cup D, i, n+i \notin S, k \in M \tag{3.50}$$

Se o item i é carregado na pilha k, o veículo percorre o conjunto S sem deixá-lo, e logo em seguida visita $n+i$, então o item de qualquer localização de coleta em $\mathcal{A}$ não pode ser carregado na pilha k. Como a contribuição do segundo termo na desigualdade original (3.46) é 1, normalizamos o segundo termo na versão fortalecida para que ele faça, no máximo, uma contribuição unitária na violação da desigualdade.

3.4.2.4 Desigualdades de Caminho Fortalecidas

Quando a rota viola a capacidade de uma pilha, a desigualdade (3.47) é usada para proibir o caminho que leva a essa solução inviável. Tais desigualdades eliminam somente um caminho, aquele que violou a capacidade da pilha. Note que qualquer outro caminho usando esses mesmos nós em ordem diferente, e carregando os mesmos itens na pilha com capacidade excedida, seria ainda um caminho inviável. Portanto, podemos eliminar não somente o caminho inviável, mas todos os caminhos que usam esses mesmos nós e carregam os itens na pilha violada. Considerando a notação exposta nas seções 3.4.1.2 e 3.4.2, a seguinte desigualdade é válida para o PDTSPMS:

$$x(N(h)) + \sum_{j \in \mathcal{Z}} z_j^k \leq |N(h)| + |\mathcal{Z}| - 2, \qquad k \in M, (h,a) \in \mathcal{H} \tag{3.51}$$

3.4.3 Formulação Fluxo

Até onde sabemos, nenhuma formulação de programação inteira de tamanho polinomial foi proposta para o PDTSPMS. Nessa seção, apresentamos uma formulação de fluxo com um número polinomial de variáveis e restrições. Côté et al. [2012a] também propuseram uma formulação de fluxo, entretanto, a formulação deles ainda usa um número exponencial de restrições de eliminação de sub-rotas (3.4) e desigualdades de precedência (3.5). Nessa formulação usamos as varíaveis x_{ij} e z_{ij} descritas nas seções anteriores. Antes de apresentarmos a formulação, definimos as seguintes novas variáveis:

- $\{f_{ij}^k \in \mathbb{R}_+ : (i,j) \in A, k \in M\}$, f_{ij}^k é o valor da carga do veículo na pilha k ao passar pelo arco (i,j).
- $\{r_{ij} \in \{0,1\} : i \neq j \in V\}$, $z_{ij} = 1$ se, e somente se, o veículo visita i antes de j, 0, caso contrário.

O PDTSPMS pode ser formulado da seguinte maneira:

$$\min \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{3.52}$$

$$s.t. \quad x(i,V) = 1, \qquad i \in P \cup D \cup \{0\} \tag{3.53}$$

$$x(V,i) = 1, \qquad i \in P \cup D \cup \{2n+1\} \tag{3.54}$$

$$\sum_{k \in M} z_i^k = 1, \qquad i \in P \tag{3.55}$$

$$\sum_{(i,j) \in A} f_{ij}^k - \sum_{(j,i) \in A} f_{ji}^k = d_i z_{\pi(i)}^k, \qquad i \in P \cup D, k \in M \tag{3.56}$$

$$f_{ij}^k \leq Q x_{ij}, \qquad (i,j) \in A, k \in M \tag{3.57}$$

$$r_{ij} + r_{ji} = 1, \qquad i,j \in P \cup D \tag{3.58}$$

$$r_{ij} \geq x_{ij}, \qquad i,j \in P \cup D \tag{3.59}$$

$$r_{ij} \geq r_{iv} + x_{vj} - 1, \qquad i,j,v \in P \cup D \tag{3.60}$$

$$r_{n+i,i} = 0, \qquad i \in P \tag{3.61}$$

$$z_i^k + z_j^k + r_{ij} + r_{j,n+i} + r_{n+i,n+j} \leq 4, \qquad i,j \in P, k \in M \tag{3.62}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in A \tag{3.63}$$

$$r_{ij} \in \{0,1\}, \qquad i,j \in P \cup D, i \neq j \tag{3.64}$$

$$z_i^k \in \{0,1\}, \qquad i \in P, k \in M \tag{3.65}$$

$$f_{ij}^k \in \mathbb{R}_+, \qquad (i,j) \in A, k \in M \tag{3.66}$$

A função objetivo (3.52) minimiza o custo total da rota. Cada nó é visitado somente uma vez com as restrições (3.53)-(3.54). As restrições (3.55) indicam que cada item deve ser carregado em uma única pilha. As restrições (3.56) são as restrições de balanço de fluxo, cada coleta aumenta o fluxo e cada entrega diminui o fluxo. A carga em cada pilha, em qualquer momento, não deve exceder a capacidade (3.57). As restrições (3.58)-(3.60) estebelecem a ordem relativa de visitação entre as localizações na rota. As restrições (3.61) impõem a precedência entre as localizações de coleta e entrega. Finalmente, as restrições (3.62) fixam a política LIFO na rota.

A necessidade de desigualdades de eliminação de sub-rotas é eliminada através das restrições (3.58)-(3.60). Considere um ciclo e dois vértices i e j dentro dele, como consequência de ser um ciclo, haverá um caminho de i para j e um caminho de j para i, devido as restrições (3.59) e (3.60), teremos $r_{ij} = 1$ e $r_{ji} = 1$, o que é impossível devido as restrições (3.58).

Também notamos que as restrições (3.58)-(3.60) permitem a relaxação das restrições de integralidade das variáveis r_{ij}. Em linhas gerais a prova é feita por indução a partir das variáveis x_{ij}, que são inteiras, consequentemente através das restrições (3.59) as variáveis r_{ij} correspondentes os arcos da rota também terão valores inteiros, e como as variáveis são definidas para todos os pares de nós através das restrições (3.58) e (3.60), as outras variáveis r_{ij} também terão valores inteiros.

Capítulo 4

Algoritmos *Branch-and-Cut*

Nessa seção, apresentamos os algoritmos *branch-and-cut* implementados para as formulações apresentadas no Capítulo 3. Inicialmente, damos uma breve explicação informal de um algoritmo *branch-and-cut*. Em seguida, apresentamos desigualdades válidas para o problema incorporadas nos algoritmos. Após a apresentação das desigualdades, expomos os algoritmos de separação utilizados para encontrar desigualdades violadas. Finalmente, descrevemos detalhadamente o funcionamento dos algoritmos.

4.1 *Branch-and-Cut*

A maioria dos problemas abordados em otimização combinatória pertencem à classe NP-difícil, possuindo um espaço de solução gigantesco. Explorar tal espaço de solução por completo é uma abordagem ineficiente devido à natureza exponencial do problema. Consequentemente se faz necessário uma forma inteligente de explorar tal espaço para chegar à solução ótima, sem ter de checá-lo todo. Uma dessas abordagens é o que chamamos de *branch-and-cut*, uma exploração da árvore de possibilidades do problema através de um algoritmo *branch-and-bound* com a geração de cortes para desigualdades violadas. É uma tentativa de melhorar os limites inferiores e superiores do problema, com o objetivo de que a melhoria de tais limites permita eliminar a necessidade de exploração de certos nós da árvore, realizando, dessa maneira, uma exploração implícita dos mesmos.

Em um algoritmo *branch-and-cut* começamos com um politopo $\mathcal{A}$ com o objetivo de aproximar o fecho convexo $\mathcal{P}$ do problema. O politopo $\mathcal{A}$ pode ser definido por um conjunto pequeno de restrições do modelo do problema. Tendo definido $\mathcal{A}$, relaxamos a integralidade das variáveis do problema e usamos $\mathcal{A}$ nos nós da árvore de *branch-and-bound*. Em cada nó resolvemos a relaxação de $\mathcal{A}$ e procuramos por restrições originais de

$\mathcal{P}$ que não pertencem a $\mathcal{A}$ e que são violadas pela solução da relaxação. Dessa forma, adicionamos iterativamente ao politopo $\mathcal{A}$ os cortes das restrições violadas. Com a geração de cortes o mesmo se aproxima de $\mathcal{P}$, ao mesmo tempo que aproximamos $\mathcal{A}$ de $\mathcal{P}$ na fase de exploração da árvore de *branching*, melhoramos os limites inferior e superior do problema, o que permite evitar explorar certos nós. Esse processo é executado até que o limite inferior seja igual ao superior, momento este que garante que exploramos implicitamente toda a árvore de possibilidades do problema.

4.2 Desigualdades Válidas

Primeiro, expomos duas famílias de desigualdades válidas para o problema desenvolvido nessa dissertação. A primeira família diz respeito às Restrições de Eliminação de Sub-rota com Precedência, onde incorporamos nas desigualdades de eliminação de sub-rota o aspecto de precedência. A segunda diz respeito às Desigualdades de Início e Fim de Rota, onde aproveitamos a estrutura especial de caminhos que contêm somente coletas a partir do depósito inicial, e caminhos que contêm somente entregas para o depósito final. Em seguida, exibimos as desigualdades válidas para o PDTSPMS anteriormente na literatura.

4.2.1 Restrições de Eliminação de Sub-rota com Precedência

Dado uma rota para o PDTSPMS, um arco $(i, n+j)$ ou $(n+j, i)$ significa que a coleta j deve ser visitada antes da coleta i na rota, caso contrário, a precedência de j não é satisfeita. Consequentemente, por exemplo, a rota não pode utilizar dois arcos $(i, n+j)$ e $(j, n+i)$, uma vez que ambas as precedências não podem ser simultaneamente satisfeitas. Usando esse fato, considere um nó de coleta i e uma entrega $n+j$ dentro de $S \subset P \cup D$, onde suas respectivas entrega $n+i$ e coleta j não pertencem a S. Isto é, $i, n+j \in S$ e $n+i, j \notin S$. A seguinte desigualdade é válida para o PDTSP:

$$x(S) + x_{j,n+i} \leq |S| - 1, \qquad S \subset P \cup D, i, n+j \in S, j, n+i \notin S \tag{4.1}$$

Se o veículo formou um caminho em S sem sair do conjunto, temos ou um caminho de i para $n+j$, ou de $n+j$ para i, em ambos os casos o nó $j \notin S$ tem que ser visitado antes de i. Dessa forma, o arco $x_{j,n+i}$ não pode estar ativo, uma vez que ele implica que i deve ser visitado antes de j, o que faz com que seja impossível satisfazer ambas as precedências dos nós i e j ao mesmo tempo.

Entretanto a desigualdade (4.1) anterior pode ser fortalecida usando todas as coletas $u \in S$ tal que $n + u \notin S$, ao invés de somente o nó i. Isso leva à desigualdade:

$$x(S) + \sum_{u \in S, n+u \notin S} x_{j,n+u} \leq |S| - 1, \qquad S \subset P \cup D \tag{4.2}$$

4.2.2 Desigualdades de Início e Fim de Rota

O aspecto de precedência do problema impõe um grande número de restrições na rota. Entre cada par de localizações de coleta e entrega, o nó de coleta tem de ser visitado primeiro, como resultado não é possível visitar o nó de entrega a não ser que sua coleta correspondente tenha sido efetuada. Por exemplo, usando a fórmula proposta por Ruland & Rodin [1997] para calcular o número de rotas viáveis para o PDTSP, considerando um problema com $n = 5$ requisições, enquanto temos 3.628.800 rotas viáveis para o TSP, para o PDTSP temos 113.400 rotas viáveis. Por esses números é possível ter uma noção de como a precedência restringe as rotas. Para ver a quantidade de rotas para outros valores de n, recomendamos a leitura do trabalho de Cordeau et al. [2010b]. Cordeau et al. [2010b] apresentam uma fórmula de recorrência para calcular o número de rotas viáveis para o PDTSPL, ainda usando $n = 5$, para o PDTSPL temos 5.040 rotas viáveis.

Devido à precedência, depois que um conjunto S, somente de coletas, é visitado no início de uma rota, as únicas entregas que podem ser visitadas são aquelas cujas coletas estão em S. O mesmo raciocínio se aplica ao fim de uma rota. Se um conjunto S somente de entregas é visitado imediatamente antes do depósito final, então somente coletas correspondentes a estas entregas podem ser visitadas imeditamente antes de S. Considere um caminho começando no depósito inicial consistindo somente de coletas, depois de visitar todas as coletas nesse caminho, você não pode visitar um nó de entrega cuja coleta correspondente não está no caminho. Se qualquer nó de entrega além daqueles cuja coleta correspondente está no caminho for visitado, então a precedência é violada. A primeira dessas observações leva à seguinte desigualdade válida:

$$x(0, S) + x(S) + x(S, D \setminus \sigma(S)) \leq |S|, \qquad S \in P \tag{4.3}$$

A desigualdade (4.3) elimina todos os caminhos que começam no depósito inicial, visitam S, e deixam o conjunto através de um nó de entrega cuja coleta correspondente não está dentro de S. Um exemplo é mostrado na Figura 4.1. A rota visita $S = \{ i, k \}$ a partir do deposito inicial e e imediatamente após visita $n + j$, violando a precedência do nó j.

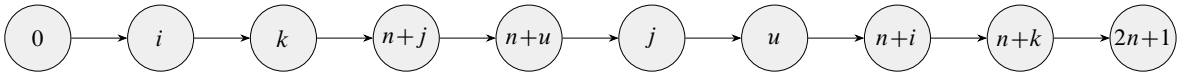

Figura 4.1: Caminho que viola ambas as desigualdades de início e fim de rota.

Um argumento similar é usado para derivar as desigualdades (4.4), nesse caso consideramos um caminho terminando no depósito final consistindo somente de nós de entrega, antes de visitar todas as entregas nesse caminho o veículo não pode visitar uma coleta cujo nó de entrega correspondente não está no caminho. Um representação gráfica também é exibida na Figura 4.1. Antes de terminar a rota no depósito final, o veículo visita o nó u e em seguida percorre um caminho por $S = \{ n + i, n + k \}$, violando a precedência do nó u.

$$x(P \setminus \pi(S), S) + x(S) + x(S, 2n + 1) \leq |S|, \qquad S \in D \tag{4.4}$$

Estas desigualdades exploram a vantagem da estrutura especial das invibialidades que acontecem ao visitar somente coletas no início da rota ou visitar somente entregas no fim da rota.

4.2.3 Desigualdades para o PDTSP

Sendo uma versão restrita do PDTSP, todas as desigualdades válidas para o problema anterior também são válidas para o PDTSPMS. Dado $S \subset V$, seja $\pi(S) = \{ i \in P \mid n + i \in S, \}$ o conjunto de nós de coleta tal que suas entregas correspondentes estão dentro de S, e $\sigma(S) = \{ n + i \in D \mid i \in S \subset V \}$ o conjunto de nós de entregas tal que seus respectivos nós de coleta estão em S. Três classes de desigualdades introduzidas por Balas et al. [1995] são consideradas:

$$x(n + i, S) + x(S) + x(S, i) \leq |S|, \qquad \forall i \in P, S \subseteq V \setminus \{0, 2n + 1, i, n + i\} \tag{4.5}$$

$$x(S) + x(S, \overline{S} \cap \pi(S)) + x(S \cap \pi(S), \overline{S} \setminus \pi(S)) \leq |S| - 1, \qquad S \subset P \cup D \tag{4.6}$$

$$x(S) + x(\overline{S} \cap \sigma(S), S) + x(\overline{S} \setminus \sigma(S), S \cap \sigma(S)) \leq |S| - 1, \qquad S \subset P \cup D \tag{4.7}$$

As desigualdades (4.5) previnem um caminho de $n + i$ para i através de S, proi-

bindo um caminho por S onde a entrega é visitada antes da coleta. As desigualdades (4.6) e (4.7) são referidas como desigualdades de predecessor e sucessor, respectivamente, e dominam a desigualdade (4.5). Além disso, as desigualdades de quebra de ciclo de precedência (*precedence cycle break inequalities*) introduzidas por Balas et al. [1995] para o Problema do Caixeiro Viajante com Precedências também são válidas:

$$\sum_{u=1}^{m} x(S_u) \leq \sum_{u=1}^{m} |S_u| - m - 1 \tag{4.8}$$

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 $u_{m+1} = u_1$.

Para exemplificar a desigualdade anterior, considere duas coletas i e j e suas respectivas entregas $n + i$ e $n + j$, sejam $S_1 = \{i, n + j\}$ e $S_2 = \{j, n + i\}$ os conjuntos usados na definição da desigualdade, em uma desigualdade (4.8) violada, ou um ciclo ocorre em S_1 ou S_2, ou uma das precedências é violada.

4.2.4 Desigualdades para o PDTSPMS

No Capítulo 3, apresentamos desigualdades válidas para o PDTSPMS dentro das formulações arco e nó. Nesta seção, apresentamos desigualdades válidas simultâneamente para ambas as formulações. As desigualdades lidam com o aspecto de capacidade do problema e foram propostas por Côté et al. [2012a].

Côté et al. [2012a] adaptaram desigualdades clássicas do VRP para o PDTSPMS. Eles adaptam as Desigualdades de Capacidade Arredondada (*Rounded Capacity Inequalities*, referimos ao trabalho de Naddef & Rinaldi [2001] para mais informações) para o problema. Seja $r(S)$ a quantidade mínima de veículos necessários para atender todos os clientes em $S \subset P \cup D$, a seguinte desigualdade é válida para o VRP:

$$x(\delta(S)) \geq 2r(S), \qquad S \subset P \cup D \tag{4.9}$$

Como o cálculo de $r(S)$ envolve resolver um problema de *bin packing*, que é NP-difícil, é utilizado o limite inferior (LB) $\lceil |\sum_{i \in S} d_i|/Q \rceil$ no lugar de se resolver o problema, obtendo então:

$$x(\delta(S)) \geq 2 \left\lceil \frac{|\sum_{i \in S} d_i|}{Q} \right\rceil, \qquad S \subset P \cup D \tag{4.10}$$

Como múltiplas pilhas estão disponíveis no PDTSPMS , as desigualdades (4.10)

são adaptadas para levar em conta a capacidade total MQ do veículo:

$$x(\delta(S)) \geq 2 \left\lceil \frac{\left|\sum_{i \in S} d_i\right|}{MQ} \right\rceil, \qquad S \subset P \cup D \tag{4.11}$$

Como notado por Côté et al. [2012a] as desigualdades (4.11) não são suficientes para eliminar todas as soluções que violam a capacidade. Côté et al. [2012a] e Sampaio & Urrutia [2017] incorporam as desigualdades (4.11) aos algoritmos *branch-and-cut* propostos, entretanto, como notado por Ropke & Cordeau [2009], o LB $\lceil |\sum_{i \in S} d_i|/Q \rceil$ pode ser melhorado para $k(S) = \max\{1, \lceil q(\pi(S) \setminus S)/Q \rceil, \lceil -q(\sigma(S) \setminus S)/Q \rceil\}$, uma vez que $\lceil q(\pi(S) \setminus S)/Q \rceil$ é um LB na carga total do veículo ao entrar em S e $\lceil -q(\sigma(S) \setminus S)/Q \rceil$ é um LB na carga total do veículo ao deixar S. O novo LB é mais forte que o anterior porque:

$$\begin{aligned}
|q(S)| &= |q(S \cap D) + q(S \cap P)| \\
&= |q(\pi(S)) + q(\sigma(S))| && \leftarrow q(S \cap D) = -q(\pi(S)), q(S \cap P) = -q(\sigma(S)) \\
&= |q(\pi(S) \setminus S) + q(\sigma(S) \setminus S)| && \leftarrow q(\{\, i, n+i \,\}) = 0 \\
&\leq \max(q(\pi(S) \setminus S), -q(\sigma(S) \setminus S) && \leftarrow q(\pi(S) \setminus S) \geq 0, q(\sigma(S) \setminus S) \leq 0
\end{aligned}$$

Assim, obtemos a seguinte desigualdade válida para o PDTSPMS:

$$x(\delta(S)) \geq 2k(S), \qquad S \subset P \cup D \tag{4.12}$$

As desigualdades de conflito de capacidade (*conflict capacity inequalities*) propostas por Côté et al. [2012a] também são usadas:

$$x(i, S) + x(S) + x(S, n+i) \leq |S|, \qquad S \subset P \cup D, i, n+i \notin S, z(S) > Q(M-1) \tag{4.13}$$

onde $z(S) = \max(q(\pi(S) \setminus S), -q(\sigma(S) \setminus S))$, como descrito anteriormente, $q(\pi(S) \setminus S)$ é um LB na capacidade total do veículo ao entrar em S, $\pi(S) \setminus S$ são as coletas fora de S com entrega correspondente em S, ou seja, os itens que serão entregues dentro de S, e $-q(\sigma(S) \setminus S)$ é um LB na capacidade total do veículo ao deixar S, $\sigma(S) \setminus S$ são as entregas fora de S com coleta correspondente em S, ou seja, os itens que serão entregues fora de S. A desigualdade proíbe itens que cruzam com i de serem carregados na mesma pilha, consequentemente eles devem ser carregados na capacidade restante $(M-1)Q$ disponível. Se $z(S) > Q(M-1)$, então as desigualdades (4.13) proíbem

todos os caminhos de i para $n+i$ através de S, uma vez que os itens que cruzam com i devem ser carregados no espaço remanescente, entretanto a carga total desses itens é maior do que a capacidade disponível, consequentemente tornando o caminho inviável.

4.3 Algoritmos de Separação

Dada a projeção de uma solução fracionária $\mathrm{x}^* \in \mathbb{R}^{|A|}$, o grafo de suporte associado G^* é o grafo com vértices V e arcos $A^* = \{\, (i,j) \mid x^*_{ij} > 0 \,\}$. Dado o grafo de suporte G^*, o adaptamos de acordo com cada desigualdade sendo separada, e então executamos algoritmos de fluxo máximo para obter o conjunto S do corte mínimo com o objetivo de obter, caso o fluxo máximo seja menor que um dado valor, uma desigualdade violada.

Quando a solução é inteira, ao invés de resolver problemas de fluxo máximo para encontrar desigualdades violadas, algoritmos mais simples são usados para resolver o problema de separação com menos esforço. Em particular, procedimentos de separação exatos são usados para as desigualdades LIFO (3.41) e (3.46), e de caminho (3.42) e (3.47). Soluções viáveis são determinadas para o PDTSPMS através desses procedimentos, ao invés de resolver um problema de empacotamento NP-difícil como é feito por Côté et al. [2012b]. Em cada solução inteira existe um caminho do nó 0 para o nó $2n+1$, contendo ou não todos os outros vértices. Somente esse caminho é considerado ao separar as desigualdades nesse estágio. As desigualdades violadas são identificadas simulando a visitação do caminho pelo veículo.

Procedimentos de separação para as desigualdades (3.4), (3.5) e (4.5) usando o grafo de suporte G^* são descritos em Cordeau et al. [2010b]. Observe que essas desigualdades são independentes do aspecto de carga do problema. Nas seções seguintes descreveremos esses procedimentos em mais detalhes. Para separação das nossas desigualdades LIFO propostas modificamos o grafo de suporte com a informação dada pelas variáveis **y** ou **z** dependendo da formulação sendo resolvida.

4.3.1 Restrições de Eliminação de Sub-rota

Seja $S \subset P \cup D$ tal que $S \neq \emptyset$, para todo $i \in S$, devido às restrições de grau, temos $x(\delta^+(\{i\})) = 1$, consequentemente, $\sum_{i \in S} x(\delta^+(\{i\})) = |S|$, a decomposição da soma anterior em arestas em S e arestas no corte resulta em, $x(S) + x(\delta^+(S)) = |S|$, ou seja, a soma das arestas com ambos os nós em S e das arestas com somente o nó terminal de partida em S. Pelas restrições de eliminação de sub-rotas (3.4) temos $x(S) \leq |S| - 1$, substituindo na inequação anterior resultante da decomposição, obtemos, $x(\delta^+(S)) \geq$

$|S| - (|S| - 1) = 1$, ou seja, $x(\delta^+(S)) \geq 1$ se e somente se $x(S) \leq |S| - 1$. Dessa forma, caso uma desigualdade (3.4) seja violada, isto é, $x(S) > |S| - 1$, temos $x(\delta^+(S)) < 1$.

Para cada aresta $(i, j) \in G^*$, definimos sua capacidade como x^*_{ij}. Então, o valor de $x^*(\delta^+(S))$ para qualquer conjunto S de nós é precisamente o mesmo que a capacidade do corte $\delta^+(S)$ em G^*. Consequentemente, podemos aplicar o teorema do corte mínimo em problemas de fluxo máximo. Existe um conjunto S de nós que viola (3.4) se e somente se existe um corte em G com capacidade menor do que um.

Para separação exata das desigualdades (3.4), resolvemos um problema de fluxo máximo em G^* de 0 para cada coleta $i \in P$, e de cada entrega $n+i \in D$ para $2n+1$. Se o fluxo for menor do que 1, uma desigualdade violada (3.4) foi encontrada. No primeiro caso, devido às restrições de grau teremos que 0 e $2n + 1$ pertencerão ao conjunto S, já que todo fluxo que sai de 0 deve chegar em $2n + 1$ em G^*, dessa forma ao invés de usar S para definir a desigualdade, usamos o conjunto $\overline{S}$. Caso um nó de coleta esteja incluído em um conjunto S previamente gerado, não executamos o fluxo de 0 para ele, da mesma forma, caso um nó de entrega esteja incluído em um conjunto S previamente gerado, não executamos o fluxo a partir dele para $2n + 1$. Isso é feito para não gerar um corte previamente gerado, evitando a redundância.

Caso a solução seja inteira, não é necessário resolver problemas de fluxo máximo. Se G^* viola qualquer desigualdade (3.4), teremos uma componente definida pelo caminho de 0 para $2n + 1$, e outras componentes conectadas que definem desigualdades (3.4) violadas, ou seja, definem ciclos. Inicialmente, percorremos o caminho de 0 para $2n + 1$ identificando os vértices visitados. Em seguida, escolhemos um vértice u não visitado e percorremos um caminho, novamente identificando os vértices visitados, até que u seja visitado novamente. Nesse momento um ciclo foi encontrado, geramos uma desigualdade (3.4) para o conjunto S definido por essa componente. Continuamos esse procedimento até que todos os vértices tenham sido visitados, e consequentemente todos os ciclos.

4.3.2 Restrições de Precedência

As desigualdades (3.5) definem que para cada $S \in \mathcal{S}$, $S \leq |S| - 2$, ou seja, não podemos formar um caminho por S, pois se formado, uma precedência foi violada. Isso significa que para cada $S \in \mathcal{S}$, devemos sair do conjunto pelo menos duas vezes, consequentemente, $S \leq |S| - 2$ pode ser reescrito como $x(S, \overline{S}) \geq 2$, pois caso contrário uma desigualdade (3.5) é violada. Uma prova formal é similar à exibida na Seção 4.3.1.

Para separação exata das desigualdades (3.5) o grafo de suporte G^* deve ser modificado. Para cada $i \in P$ conectamos 0 ao nó $n + i$ e o nó i ao nó $2n + 1$ com

capacidade 2, em outras palavras, $x^*_{0,n+i} = 2$ e $x^*_{i,2n+1} = 2$. Em seguida resolvemos o problema de fluxo máximo do nó 0 ao $2n + 1$ e obtemos um conjunto S^*, que por construção, contém 0 e $n + i$, e não contém i. Se o fluxo for menor do que 2, então o conjunto S^* define uma desigualdade (3.5) violada.

Caso a solução seja inteira, simulamos a visitação do caminho de 0 a $2n + 1$ pelo veículo. Começando no depósito 0, visitamos o caminho checando a ordem de visitação dos vértices, caso um vértice de entrega $n + i$ seja visitado antes de sua coleta correspondente i, temos que uma desigualdade (3.5) foi violada. Definimos uma desigualdade (3.5) para os vértices contidos no caminho de 0 a $n+i$. Note que, dentro de um ciclo, não é possível definir uma ordem de visitação, consequentemente não separamos as desigualdades (3.5) dentro de ciclos em uma solução inteira.

Separamos as desigualdades (4.5) de maneira exata para cada $i \in P$ da seguinte maneira. Criamos um vértice artificial $2n + 2$ e conectamos $2n + 2$ a cada nó $k \in V \setminus \{n + i, i\}$ com um arco de capacidade $w_{2n+2,k} = x^*_{n+i,k} + x^*_{k,i}$. Aumentamos a capacidade dos arcos $(2n + 1, i)$, $(0, i)$, e $(n + i, i)$ para 2. Resolvemos o problema de fluxo máximo de $2n + 2$ para i, e obtemos um conjunto S^* tal que, $2n + 2 \in S^*$ e, por construção, i, $n + i$, 0, $2n + 1 \notin S^*$. As desigualdades podem ser reescritas como $x(\delta^+(S)) + x(n + i, S) + x(\overline{S}, i) \geq 2$, assim, se o fluxo for menor do que 2, então o conjunto $S^* \setminus \{2n + 2\}$ define uma desigualdade (4.5) violada. Ao invés de incluir a desigualdade (4.5) violada diretamente no modelo, o conjunto S^* correspondente a restrição violada é estendido com $\{i\}$ e $\{n + i\}$ para definir desigualdades violadas do tipo (4.6) e (4.7), respectivamente, uma vez que ambas as desigualdades dominam as desigualdades (4.5). Referimos o leitor ao trabalho de Balas et al. [1995] para mais detalhes sobre as desigualdades.

Para as desigualdades (4.8) adotamos a mesma abordagem de Sampaio & Urrutia [2017]. Cordeau et al. [2010b] desenvolveram uma heurística para separá-las, mas como avaliado por Sampaio & Urrutia [2017] os limites inferiores fornecidos na raíz usando as desigualdades não apresentaram melhoria significativa. Dessa forma seguimos a abordagem de Sampaio & Urrutia [2017] e adicionamos as desigualdades (4.8) somente no modelo inicial, uma vez que essa abordagem forneceu melhores resultados do que não adicioná-las de nenhuma forma.

4.3.3 Desigualdades de Capacidade

Para a separação das desigualdades (4.12) seguimos a abordagem utilizada por Côté et al. [2012a], que implementa a heurística proposta por Ropke & Cordeau [2009]. O algoritmo escolhe um vértice inicial para S aleatoriamente e iterativamente escolhe

o nó que maximiza a função (4.14) abaixo. A heurística é executada cinco vezes e é interrompida quando uma desigualdade (4.12) violada é encontrada, ou quando é improvável que tal violação vá acontecer, ou seja, o conjunto S permanece o mesmo durante uma iteração.

A função $f(S)$ contém três parâmetros λ_1, λ_2, e λ_3. No início de cada execução da heurística, λ_1 e λ_2 recebem aleatoriamente valores no intervalo $[1, 5]$, e λ_3 recebe um valor aleatório no intervalo $[0, 1]$. A primeira expressão reflete as mudanças das demandas dos nós em S. Quando a segunda expressão é maior do que 0, uma desigualdade (4.12) violada foi descoberta. A terceira expressão foca em maximizar a expressão que não está ativa nos outros dois termos.

$$\begin{aligned} f(S) =& \lambda_1 \left(\max\left\{q(\pi(S) \setminus S), -q(\sigma(S) \setminus S)\right\} - Qx(\delta^+(S))\right) \\ &+\lambda_2 Q \left(\max\left\{\left\lceil \frac{q(\pi(S) \setminus S)}{Q} \right\rceil, \left\lceil \frac{-q(\sigma(S) \setminus S)}{Q} \right\rceil\right\} - x(\delta^+(S)\right) \\ &+\lambda_3 \left(\min\left\{q(\pi(S) \setminus S), -q(\sigma(S) \setminus S)\right\} - Qx(\delta^+(S))\right) \end{aligned} \tag{4.14}$$

Observamos que, na implementação da heurística, para computar a demanda total do conjunto, não é necessário percorrer, em cada iteração, o conjunto S, o que na verdade seria ineficiente. Ao adicionar um item ao conjunto S, checa-se inicialmente se ele é uma entrega ou coleta, em seguida é checado se a coleta ou entrega correspondente ao item adicionado já está em S, após as duas checagens anteriores, adiciona-se o item em S e atualiza-se o valor da demanda total, $\pi(S) \setminus S$ ou $\sigma(S) \setminus S$. Por exemplo, ao adicionar uma coleta i em S, onde sua entrega $n + i$ já pertence ao conjunto, devemos subtrair a demanda d_i de $q(\pi(S) \setminus S)$.

Além disso, os conjuntos S gerados ao separar as desigualdades (3.4),(4.6) e (4.7) também são avaliadaos para checar se definem desigualdades violadas do tipo (4.12).

Separamos as desigualdades (4.13) somente para soluções inteiras. Se o conjunto S no caminho $i \rightsquigarrow S \rightsquigarrow n+i$ não definir uma desigualdade LIFO violada, checamos se $z(S) > Q(M-1)$. Se isto for verdadeiro, o conjunto define uma desigualdade (4.13) violada. Separamos as desigualdades somente para soluções inteira, pois, nessa fase, temos em mãos os itens que se cruzam com i no caminho de i para $n + i$.

As desigualdades (3.42) e (3.47) são separadas da mesma maneira. As desigualdades só são separadas para soluções inteiras, além disso, o procedimento de separação só é executado caso não tenha sido encontrada nenhuma violação de precedência ou LIFO. O caminho do depósito 0 ao depósito $2n+1$ é percorrido mantendo a capacidade residual de cada pilha. Ao coletar um item i em uma pilha k a capacidade da pilha é

incrementada e ao entregar um item $n+i$ da pilha k a mesma é decrementada. Dessa maneira, o caminho é visitado mantendo a capacidade corrente das pilhas, quando um item i é carregado em uma pilha k e a capacidade dela excede a capacidade Q, temos que o caminho do último nó antes de visitar i em que a pilha k estava vazia até o nó i viola as desigualdades (3.42) e (3.47). Dessa forma, geramos uma desigualdade de caminho, correspondente a formulação sendo resolvida, para o caminho da última posição em que a pilha estava vazia até o nó que causou a violação.

4.3.4 Desigualdades LIFO

Para separar as desigualdades LIFO (3.41), dada uma solução fracionária, propomos um procedimento de separação heurístico baseado em fluxo máximo. Precisamos checar, para cada par $i, j \in P$, se existe um conjunto $S \subset P \cup D$ onde $j \in S$, $i, n+j, n+i \notin S$, tal que, o veículo carrega o item i na pilha k, visita os clientes em S, também carregando j na pilha k, e deixa S através do nó $n+i$, consequentemente violando a política LIFO.

Relembrando as desigualdades (3.41) :

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

e note que elas são equivalentes a:

$$\sum_{l \neq k} y^l(\overline{S}, i) + \sum_{l \neq k} y^l(\overline{S}, j) + x(S, j) + x(S, \overline{S}) + x(\overline{S}, n+i) \geq 2, \qquad (4.15)$$
$$S \subset P \cup D, j \in S, i, n+i, n+j \notin S, k \in M$$

i.e. , se (4.15) é satisfeita então o veículo deixa o conjunto S pelo menos duas vezes, ou carrega o item i em uma pilha $l \neq k$, ou carrega j em uma pilha $l \neq k$, ou visita j dentro de S, ou não visita $n+i$ imediatamente depois de S.

Dado o grafo de suporte G^* prosseguimos da seguinte forma:

1. Aumente a capacidade do arco (j, u), $u \in \{0\} \cup P \cup D$, pelo valor $\sum_{l \neq k} y^l_{uj}$.
2. Aumente a capacidade do arco (u, i), $u \in \{0\} \cup P \cup D$, pelo valor $\sum_{l \neq k} y^l_{ui}$.
3. Aumente a capacidade do arco (j, u), $u \in P \cup D$, pelo valo $x_{u,n+i}$.
4. Defina a capacidade dos arcos $(n+j, i)$, $(n+i, i)$ e $(0, i)$ como ∞.

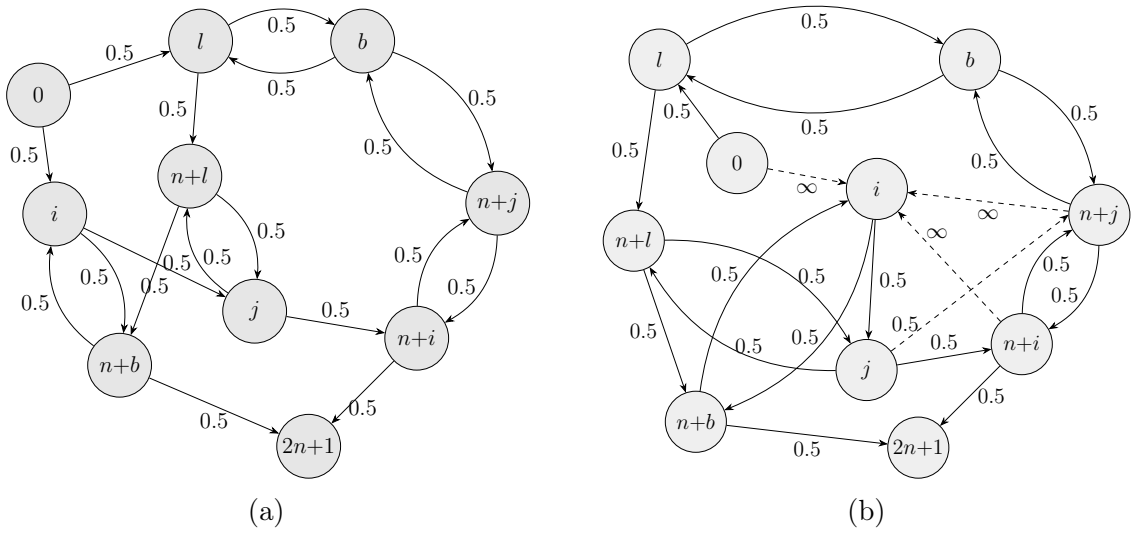

Figura 4.2: Procedimento para separação das desigualdades LIFO na formulação arco.

5. Execute o fluxo máximo de j para i. Se o valor do fluxo é menor do que 2, então pelo teorema do corte mínimo, podemos obter um conjunto S tal que $j \in S$ e $i, n+i, n+j \notin S$, e S viola a desigualdade (3.41).

Quando calculamos o fluxo máximo, estamos calculando $x(S, \overline{S})$, ao agregar os valores $\sum_{l \neq k} y^l_{ui}$ nos arcos (u, i) e $\sum_{l \neq k} y^l_{uj}$ nos arcos (j, u) estamos também calculando a soma $\sum_{l \neq k} y^l(\overline{S}, i) + \sum_{l \neq k} y^l(\overline{S}, j)$. Com a agregação dos valores $x_{u,n+i}$ nos arcos (i, u) também levamos em conta a soma $x(\overline{S}, \{n+i\})$. Então, se o fluxo é menor do que 2 o conjunto é candidato a uma uma desigualdade (3.41) violada. Por fim, checamos se $x(S, j)$ faz a desigualdade ser satisfeita.

A Figura 4.2 exibe a execução do procedimento em um grafo de suporte para um par de nós i e j. A Figura 4.2a exibe o grafo de suporte e a Figura 4.2b exibe o grafo após a execução do procedimento. Considere que temos 2 pilhas e seja $y^0_{0i} = 0.5$, $y^0_{bi} = 0.5$, $y^0_{ij} = 0.5$ e $y^0_{lj} = 0.5$. Na Figura 4.2b as linhas tracejadas representam os arcos resultantes do procedimento. Uma vez que os valores das variáveis $\mathbf{y}$ para i e j para a pilha 1 são zero, os passos 1 e 2 não modificam nenhum arco. O passo 3 irá adicionar o arco $x_{j,n+j} = 0.5$ e o Passo 4 irá adicionar os arcos com capacidade ∞. O fluxo máximo de j para i será 1.5 e o conjunto do corte mínimo $S = \{4\}$ definirá uma desigualdade (3.41) violada.

As desigualdades (3.46):

$$z^k_i + z^k_j + x(\overline{S}, j) + x(S) + x(S, n+i) \leq |S| + 2,$$

$$S \subset P \cup D, j \in S, i, n+i, n+j \notin S, k \in M$$

impõem a política LIFO na formulação nó, para separá-las exatamente usamos um procedimento que funciona da seguinte maneira. Primeiro reescrevemos (3.46) como:

$$\sum_{l \neq k} z_i^l + \sum_{l \neq k} z_j^l + x(S, \overline{S}) + x(S, j) + x(\overline{S}, n+i) \geq 2, \quad S \subset P \cup D, i, n+i, n+j \notin S, j \in S, k \in M \tag{4.16}$$

Se a desigualdade (4.16) é satisfeita, então o veículo deixa o conjunto S pelo menos duas vezes, ou visita a coleta j a partir de S, ou o veículo não deixa o conjunto S através do nó de entrega $n+i$, ou o item i não é carregado na pilha k, ou o item j não é carregado na pilha k. Separamos as desigualdades (4.16) da seguinte forma:

1. Crie um nó extra $2n+2$ em G^* e conecte $2n+2$ a cada nó $v \in P \cup D \setminus \{n+i\}$ com um arco de capacidade de $x^*_{v,n+i}$.

2. Crie um nó extra $2n+3$ em G^* e conecte a $2n+3$ cada nó $v \in P \cup D \setminus \{j\}$ com um arco de capacidade de x^*_{vj} .

3. Defina a capacidade dos arcos $(2n+2, j)$, $(2n+1, n+i)$, $(0, n+i)$, $(j, n+i)$, $(n+j, n+i)$, $(i, n+i)$ e $(2n+3, n+i)$ para 2

4. Execute o fluxo máximo de $2n+2$ para $n+i$. Se o fluxo máximo for menor do que 2, então o conjunto S é um candidato para uma desigualdade (4.16) violada. Para garantir que a violação ocorre, a soma $\sum_{l \neq k} z_i^l + \sum_{l \neq k} z_j^l$, não levada em conta na modificação do grafo de suporte, é checada. Se o fluxo mais a soma $\sum_{l \neq k} z_i^l + \sum_{l \neq k} z_j^l$ ainda for menor do que 2, o conjunto $S \setminus \{2n+2\}$ define uma desigualdade (3.46) violada.

A Figura 4.3 exibe a execução do procedimento em um grafo de suporte para um par de nós i e j. A Figura 4.3a mostra o grafo de suporte e a Figura 4.3b mostra o grafo após a execução do procedimento. Considere que temos 2 pilhas e seja $z_i^1 = 1$ e $z_j^1 = 1$. Na Figura 4.3b as linhas tracejadas representam arcos resultantes do procedimento. O passo 1 adiciona somente um arco, de $2n+2$ para j. O passo 2 adiciona o arco de k para $2n+3$, e o passo 3 adiciona os arcos restantes. O fluxo máximo de $2n+2$ para $n+i$ é 1, e uma vez que $z_i^0 + z_j^0 = 0$, o corte mínimo $S = \{j, l\}$ define uma desigualdade (3.46) violada.

Para uma solução inteira, separamos as desigualdades LIFO de maneira mais simples. Percorremos a rota e checamos para cada coleta i que tem a precedência

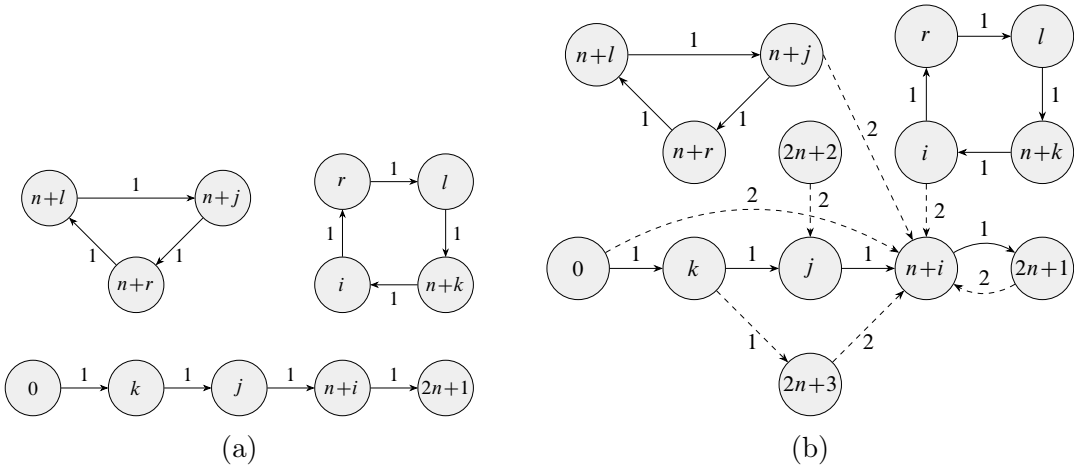

Figura 4.3: Procedimento para separação das desigualdades LIFO na formulação nó.

satisfeita, se existe um nó j entre o caminho de i até $n+i$ tal que $n+j$ não pertence a esse caminho, e ambos i e j foram carregados na mesma pilha, caso isso aconteça, o conjunto S contendo os nós de j até o último vértice visitado antes de $n+i$ define desigualdades violadas (3.41) e (3.46) para a pilha em que i e j foram carregados.

4.4 Pré-processamento

Antes de inicializar, 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, uma vez que 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 $i \in P$ e $n+i \in D$ são removidos, uma vez que o predecessor de uma coleta não pode ser sua entrega correspondente. No contexto geral do TSP com precedências, Balas et al. [1995] fornece uma demonstração de todos os arcos inválidos em uma rota viável.

4.5 Simetria

As variáveis propostas para modelar as operações de pilha introduzem um grau de simetria em suas formulações. Portanto, uma solução viável tem um conjunto equivalente de soluções consistindo da mesma rota permutando as atribuições de pilhas de dois ou mais itens. Em vista disso, adotamos a abordagem utilizada por Sampaio & Urrutia [2017]. As formulações permitem que, a primeira localização de coleta visitada, pode ser carregada em qualquer pilha dentro do veículo, consequentemente, para reduzir a

simetria em nossos modelos, as restrições (4.17) e (4.18) são incluídas, respectivamente, nas formulações arco e nó, indicando que a primeira coleta é feita usando a pilha 0.

$$\sum_{j \in P} y^0_{0,j} = 1 \tag{4.17}$$

$$x_{0,j} \leq y^0_j, \qquad j \in P \tag{4.18}$$

4.6 Modelo Inicial

O modelo inicial para a formulação arco é composto da função objetivo (3.1), pelas restrições de grau (3.2) e (3.3), as restrições (3.40) acoplando as variáveis x e y, e as restrições de simetria (4.17). As restrições de eliminação de sub-rotas (3.4) são incluídas para $|S| = 2$ e as restrições de quebra de ciclo de precedência (4.7) para $m = 2$, $S_1 = \{i, n+j\}$ e $S_2 = \{j, n+i\}$, onde $i, j \in P$. Observe que, as restrições (4.7) dominam as restrições (3.4) quando $S = \{i, n+j\}$, $i \in P$ e $n+j \in D$, nesse caso, não inserimos as desigualdades (3.4) para evitar a redundância.

O modelo inicial para a formulação nó é composto, da mesma maneira, da função objetivo (3.1), pelas restrições de grau (3.2) e (3.3), as restrições (3.45) indicando que cada item deve ser carregado em uma pilha, e as restrições de simetria (4.18). Também são incluídas as mesmas desigualdades (3.4) e (4.7) descritas anteriormente.

4.7 Detalhes da Implementação

Os algoritmos implementados usam o solver CPLEX 12.6 como *framework branch-and-cut* através da API Concert. Pré-processamento, heurísticas e todos os cortes automáticos são desligados. O mecanismo de *callback* é usado como *framework* para implementar os algoritmos *branch-and-cut*. Os parâmetros do CPLEX para seleção de nós, variáveis e direção de branching foram deixados em seus valores padrão. Outras opções de *branching* como *strong branching* foram testadas para os algoritmos, apesar de diminuir o número de nós, a estratégia não resultou em uma melhoria do tempo de execução e na quantidade de instâncias resolvidas. O tempo de execução limite para os algoritmos propostos foi definido como 3600 segundos, uma vez que esse é o valor utilizado pelos outros algoritmos da literatura. Com o objetivo de inicialmente obter uma rota, as variáveis x_{ij} recebem prioridade sobre as outras variáveis durante o *branching*. Um limite superior (UB) é fornecido para o CPLEX no início do algoritmo.

O UB fornecido é o valor obtido pela heurística *Large Neighboorhood Search* para o PDTSPMS proposta por Carrabs et al. [2007b].

Capítulo 5

Experimentos Computacionais

Neste capítulo, apresentamos os experimentos utilizados para avaliar os algoritmos implementados. Os algoritmos *branch-and-cut* implementados foram executados no conjunto de instâncias de *benchmark* da literatura. Inicialmente, descrevemos os conjuntos de instâncias utilizados. Em seguida, apresentamos os resultados computacionais. Inicialmente, avaliamos os modelos propostos nessa dissertação junto com as desigualdades apresentadas. Posteriormente, avaliamos em específico cada instância do conjunto de *benchmark* para comparar os resultados com os algoritmos da literatura. Finalmente, estendemos o tempo limite de uma hora para três horas, e executamos o algoritmo para as instâncias que permaneceram não resolvidas dentro do tempo de execução proposto. Os experimentos computacionais foram realizados em duas máquinas, uma com um processador atual e, para comparar com os outros dois algoritmos da literatura, uma máquina semelhante ao primeiro algoritmo proposto na literatura, e idêntica à máquina utilizada pelo segundo algoritmo proposto.

5.1 Ambiente Computacional

Todos os algoritmos desse trabalho foram implementados em C++ e usados em conjunto com o pacote de otimização CPLEX [1]. Devido ao fato de já existirem processadores mais eficientes do que os processadores em Côté et al. [2012a] e Sampaio & Urrutia [2017], este trabalho avalia os experimentos com uma máquina mais atual e veloz, pois, caso contrário, os experimentos ficarão ultrapassados e presos aos experimentos dos dois trabalhos anteriores. Os experimentos desse trabalho foram realizados em um Intel Core i7-4790K 4.00GHz. Executamos os algoritmos com um tempo limite de 3600 segundos, já que é o tempo limite utilizado pelos outros algoritmos da literatura.

[1] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Uma vez que os experimentos feitos em Sampaio & Urrutia [2017] e Côté et al. [2012a] foram realizados em um processor diferente, para podermos comparar o tempo de execução e determinar qual algoritmo é mais rápido nas instâncias de *benchmark*, nós também realizamos experimentos em um processador Intel Xeon E5520 2.2 GHz, igual ao usado em Sampaio & Urrutia [2017] e similar ao de Côté et al. [2012a]. Esses experimentos são apresentados nas Seções 5.4 e 5.5. Quando nos referirmos aos experimentos que realizamos por essa máquina deixaremos explícito, em todas as outras referências de experimentos nos referimos à máquina Intel Core i7-4790K 4.00GHz.

5.2 Instâncias de *Benchmark*

Para realizar os experimentos dos algoritmos implementados é utilizado o conjunto de instâncias 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]. Essas instâncias são baseadas na biblioteca de instâncias TSPLIB [2] para o Problema do Caixeiro Viajante. Foram geradas duas classes de instâncias. Na primeira classe, C1, a demanda de cada nó é 1, o número de pilhas é um número aleatório entre 2 e 4 e a capacidade de cada pilha é um número aleatório entre 1 e 10. Na segunda classe, C2, a demanda 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. Os valores apertados para a capacidade tem o objetivo de gerar instâncias difíceis.

5.3 Comparação dos Modelos

Para avaliar a efetividade dos modelos propostos, utilizamos o conjunto de instâncias de *benchmark* propostas por Côté et al. [2012a]. Os resultados são exibidos nas Tabelas 5.1, 5.2, 5.3 e 5.4. O modelo M1 refere-se à formulação arco definida por (3.1)-(3.6) e (3.40)-(3.43). M2 refere-se à formulação nó definida por (3.1)-(3.6) e (3.45)-(3.48). M4 é dado por (3.52)-(3.66). Reportamos os resultados nos modelos comuns e em cada modelo, exceto M4, adicionamos as desigualdades propostas (3.44) (SUCC), (4.3) e (4.4) (SERI) e (4.2) (LPRS). O modelo M2 também possui as desigualdades (3.50) e (3.51), consequentemente, para avaliá-las, substituimos em M2 as desigualdades (3.46) por (3.50) e (3.47) por (3.51), o que resulta no modelo M3.

[2] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Nas Tabelas (5.1) e (5.2), a coluna Resolvidas indica a quantidade de instâncias resolvidas do total de 54, a coluna Raiz indica o *gap* médio no nó raiz, calculado como $100 * (UB - LB)/UB$, a coluna *Gap* é o gap final médio de todas as instâncias, a coluna Tempo indica o tempo médio do algoritmo em todas as instâncias, a coluna Nós indica a média de nós em todas as instâncias, a coluna SEC representa a quantidade média de cortes do tipo (3.4), a coluna PREC representa a quantidade média de cortes do tipo (3.5), (4.6) e (4.7), a coluna LIFO a quantidade média de cortes LIFO, em M1 se refere às desigualdades (3.41), em M2 se refere às desigualdades (3.46), e em M3 às desigualdades (3.50). A coluna ICAP indica a quantidade média de cortes do tipo (4.13), a coluna CCAP a quantidade média de cortes de caminho de capacidade inviável, em M1 se refere às desigualdades (3.42), em M2 se refere às desigualdades (3.47), e em M3 às desigualdades (3.51). Finalmente, a coluna RCAP se refere a quantidade média de cortes do tipo (4.12).

Nas tabelas (5.3) e (5.4) exibimos as mesmas informações, mas dessa vez para as intâncias resolvidas. A coluna Tempo é o tempo médio gasto nas instâncias que são resolvidas por todos os algoritmos da literatura. No modelo M4 não reportamos o tempo pois o mesmo resolveu um conjunto menor de instâncias do que as instâncias resolvidas por todos os algoritmos.

Como observado nas Tabelas 5.1 e 5.2, o modelo M4 obteve o pior comportamento entre todos os modelos, por isso não executamos os outros testes adicionando as nossas desigualdades propostas no modelo. Vale ressaltar que a vantagem do modelo compacto é sua facilidade de implementação, pode ser fornecido para qualquer solver.

Os modelos M2 e M3 exploram em média, mais nós do que M1, uma vez que sua relaxação possui menos variáveis e restrições no modelo inicial do que M1, ela pode ser resolvida mais rapidamente. O que também resulta em gaps finais melhores, uma vez que é possível explorar mais nós, isso pode ser percebido ao comparar os gaps de M1, M2 e M3, sem levar em conta as desigualdades, já que resolvem o mesmo número de instâncias. Apesar disso, M2 e M3 não foram capazes de resolver mais instâncias do que M1 + ALL. Supomos que isso aconteça por M1 utilizar as variáveis y_{ij}^k, que possuem mais chance de modificar uma rota, uma vez que estão conectadas com as variáveis x_{ij}, enquanto o mesmo não acontece com as variáveis z_i^k, que não possuem nenhuma conexão com as variáveis x_{ij}.

Não é notado nenhuma melhoria significativa no nó raiz com as nossas desigualdades, uma vez que as desigualdades (4.6), (4.7), (3.42), (3.47), (3.51), (4.12) e (4.13) são separadas de forma heurística, dessa forma, ficando difícil de quantificar os ganhos. Entretanto, é possível perceber o efeito no aumento da quantidade de instâncias resolvidas.

Com relação as desigualdades (3.46) e (3.50), nota-se que as desigualdades (3.50) reduzem o número de cortes LIFO gerados, isso pode ser observado nas Tabelas 5.1, 5.2, 5.3 e 5.4. Note que a formulação M3 possui uma quantidade média de cortes LIFO menor. Apesar das desigualdades (3.50) serem mais fortes ao impor a política LIFO, não é percebido uma melhoria significativa nos limites fornecidos, pois como notam Côté et al. [2012a] e Sampaio & Urrutia [2017], as desigualdades LIFO dizem respeito ao aspecto de carregamento do problema, uma vez que para satisfazê-las não é necessário mudanças drásticas na rota, mas sim nas atribuições de pilhas para cada item.

Apesar de ser possível notar nas tabelas 5.2 e 5.4 uma redução na quantidade média de CCAP do modelo M2 para o modelo M3, como as desigualdades (3.47) e (3.51) são separadas heuristicamente, não podemos afirmar com certeza que isso se deve as desigualdades (3.51) serem mais fortes.

Analisando as Tabelas 5.1 e 5.2, com relação ao modelo M1, temos que os melhores resultados foram obtidos por M1 + LPRS e M1 + ALL, apesar dos resultados terem sido similares, escolhemos reportar os resultados de M1 + ALL em mais detalhes nas próximas seções pois obteve os menores tempos nas instâncias resolvidas da literatura. Iremos nos referir ao algoritmo *branch-and-cut* dessa formulação como B&C Arco. Com relação aos modelos M2 e M3, escolhemos reportar os resultados de M3 + ALL pois foi o modelo que resolveu mais instâncias em C2. Apesar de esse não ter sido o modelo que obteve os melhores tempos nas instâncias resolvidas da literatura, iremos nos referir ao algoritmo *branch-and-cut* dessa formulação como B&C Nó.

Tabela 5.1: Resultado para todas as instâncias na família C1.

Classe C1											
Modelo	Resolvidas	Raiz	Gap	Tempo	Nós	SEC	PREC	LIFO	ICAP	PCAP	RCAP
M1	37	8.75	3.21	1209.49	24879.06	176.46	5297.72	335.98	79.56	533.72	197.33
M1 + SERI	36	8.38	3.16	1227.61	26155.63	181.07	5347.19	366.72	86.11	608.15	197.96
M1 + LPRS	38	8.64	3.19	1178.04	23836.72	173.74	5237.44	340.04	76.26	515.56	206.30
M1 + SUCC	36	8.74	3.23	1230.54	24473.57	184.04	5072.31	326.48	90.70	540.57	190.94
M1 + ALL	38	8.66	3.14	1188.16	24095.39	178.33	5119.72	314.52	86.94	492.48	202.67
M2	37	8.79	2.44	1216.78	31429.81	156.59	4903.83	263.13	50.87	762.61	264.50
M2 + SERI	37	8.98	2.43	1169.64	26860.33	158.35	4827.98	263.81	55.52	760.28	238.43
M2 + LPRS	37	8.69	2.44	1205.71	30157.98	152.22	4749.28	258.41	54.17	767.61	254.11
M2 + SUCC	37	8.79	2.47	1224.66	31236.20	151.89	4800.85	245.39	50.43	764.02	237.19
M2 + ALL	37	8.76	2.41	1179.59	28921.31	154.33	4763.65	245.07	55.00	753.61	244.09
M3	37	8.82	2.52	1189.10	27715.98	158.89	4849.13	175.93	35.72	788.76	254.96
M3 + SERI	37	8.82	2.40	1217.27	28763.96	155.24	4834.02	178.98	39.44	765.87	262.44
M3 + LPRS	37	8.71	2.55	1221.98	29582.74	153.54	4768.85	176.78	35.67	770.78	246.19
M3 + SUCC	37	8.80	2.52	1230.72	30608.02	151.28	4865.04	167.04	34.17	773.30	244.78
M3 + ALL	37	8.67	2.76	1222.84	31841.50	153.00	4764.78	161.91	34.54	802.31	245.13
M4	26	10.53	9.65	2102.95	2115.17	61.72	703.20	19.06	0.07	6.39	53.67

Tabela 5.2: Resultado para todas as instâncias na família C2.

Classe C2											
Modelo	Resolvidas	Raiz	Gap	Tempo	Nós	SEC	PREC	LIFO	ICAP	PCAP	RCAP
M1	29	10.13	3.80	1785.85	46047.15	256.06	6767.07	697.11	126.35	1284.94	313.63
M1 + SERI	30	10.05	3.76	1789.36	48123.02	235.43	6759.85	695.74	122.65	1479.91	321.50
M1 + LPRS	29	10.10	3.70	1801.10	47726.39	247.22	6725.52	712.50	133.85	1303.59	322.22
M1 + SUCC	29	10.18	3.80	1805.39	44445.70	252.28	6727.96	656.63	134.41	1341.44	306.57
M1 + ALL	29	9.99	3.70	1785.15	46292.11	247.15	6490.17	643.46	129.57	1377.59	312.28
M2	27	10.25	3.54	1876.26	59110.69	241.70	6656.96	454.98	74.67	1782.22	333.94
M2 + SERI	28	10.26	3.52	1859.99	58591.93	238.61	6782.33	459.11	71.11	1837.46	343.31
M2 + LPRS	28	10.18	3.49	1852.94	59590.04	232.39	6517.81	464.22	74.52	1798.59	350.06
M2 + SUCC	27	10.39	3.66	1876.87	60301.56	241.39	6740.56	434.87	67.46	1870.57	328.30
M2 + ALL	29	10.15	3.47	1817.90	56371.80	237.41	6674.39	439.74	75.35	2002.63	337.72
M3	27	10.23	3.62	1874.66	55782.24	236.02	6626.41	261.70	39.78	1126.04	339.54
M3 + SERI	28	10.29	3.21	1847.15	54435.54	227.57	6589.04	252.98	37.24	1193.70	329.83
M3 + LPRS	29	10.16	3.54	1846.24	55922.80	227.24	6554.78	261.87	39.19	1195.72	340.63
M3 + SUCC	28	10.31	3.65	1868.39	54267.80	229.43	6696.72	245.61	37.41	1153.19	336.41
M3 + ALL	30	10.16	3.49	1849.07	58179.85	230.06	6416.76	243.00	41.46	1144.59	335.78
M4	17	9.99	10.73	2674.89	5985.15	83.28	933.83	30.43	0.00	103.39	63.81

Tabela 5.3: Resultado para as instâncias na classe C1 resolvidas simultaneamente pelos algoritmos da literatura.

Class C2								
Modelo	Tempo	Nós	SEC	PREC	LIFO	ICAP	PCAP	RCAP
M1	30.95	3249.94	52.66	454.59	127.34	24.22	271.53	51.56
M1 + SERI	31.22	3521.50	49.34	413.25	142.06	27.31	346.00	50.34
M1 + LPRS	24.28	2820.66	50.81	424.28	129.38	22.69	268.16	52.81
M1 + SUCC	39.80	3964.31	49.59	430.81	127.56	26.16	296.34	49.59
M1 + ALL	21.65	2267.66	46.88	391.13	105.56	23.16	237.72	49.38
M2	62.05	8078.17	50.76	500.21	133.48	19.86	466.83	43.52
M2 + SERI	26.44	3852.34	47.59	441.53	136.78	23.06	374.69	38.78
M2 + LPRS	31.39	4642.84	49.53	462.31	126.41	21.81	389.69	45.47
M2 + SUCC	47.30	6140.78	50.84	473.97	137.34	23.22	424.19	34.78
M2 + ALL	36.58	5193.25	48.38	448.56	123.88	22.28	397.78	43.25
M3	41.32	4366.09	48.28	481.22	101.63	18.56	397.09	38.75
M3 + SERI	55.08	6262.84	45.59	438.47	100.69	18.72	409.19	35.91
M3 + LPRS	65.24	6891.38	49.63	473.78	103.47	19.41	400.78	38.38
M3 + SUCC	74.66	7678.28	49.19	482.66	99.69	17.66	452.09	39.75
M3 + ALL	75.82	9011.13	48.72	490.47	94.34	17.81	426.41	44.06

Tabela 5.4: Resultado para as instâncias na classe C2 resolvidas simultaneamente pelos algoritmos da literatura.

Class C2								
Modelo	Tempo	Nós	SEC	PREC	LIFO	ICAP	PCAP	RCAP
M1	37.14	4991.91	57.26	463.48	209.48	19.70	1287.96	34.26
M1 + SERI	42.09	5668.04	56.13	466.39	209.83	18.52	1441.35	31.04
M1 + LPRS	32.81	4888.30	56.65	466.87	217.91	20.30	1316.48	33.83
M1 + SUCC	43.87	6053.74	56.96	476.65	199.70	19.00	1416.26	30.74
M1 + ALL	34.17	4650.65	56.52	465.30	198.87	21.61	1404.22	36.22
M2	34.08	5665.52	54.61	455.87	183.70	15.96	1372.00	21.96
M2 + SERI	52.05	8051.48	58.09	510.57	185.74	16.91	1360.00	44.65
M2 + LPRS	38.81	5990.52	56.04	510.78	178.17	16.43	1376.48	30.87
M2 + SUCC	53.31	8212.70	57.74	540.43	181.70	16.22	1422.78	30.91
M2 + ALL	49.15	7953.61	53.96	480.00	171.65	16.83	1437.26	28.35
M3	31.29	5517.30	54.22	478.04	125.35	13.04	542.78	23.17
M3 + SERI	29.13	4963.65	55.35	475.87	119.83	10.96	588.61	26.57
M3 + LPRS	33.79	5740.78	53.87	488.35	121.17	11.91	531.26	24.78
M3 + SUCC	22.87	4596.26	49.57	451.00	115.09	10.22	531.30	23.96
M3 + ALL	39.77	6497.65	52.61	430.91	114.57	11.57	506.04	19.48

5.4 Conjunto de Instâncias C1

Nas Tabelas 5.5, 5.6 e 5.7, exibimos os resultados para o conjunto de instâncias C1. Na tabela 5.5 exibimos em detalhes o resultado dos algoritmos propostos para cada instância no conjunto. Nela exibimos o gap final obtido nas instâncias não resolvidas e o tempo gasto em cada instância. A coluna n indica a quantidade de requisições, portanto uma instância possui $2n + 2$ nós. A coluna Opt exibe a solução ótima da instância, caso ela seja conhecida. Para avaliar nossos algoritmos, executamos experimentos em duas máquinas, um Intel Core i7-4790K 4.00GHz, cujos resultados são exibidos nas colunas Gap1 e Tempo1, e em um Intel Xeon E5520 2.2 GHz, idêntico ao usado em Sampaio & Urrutia [2017] e similar ao utilizado em Côté et al. [2012a], cujos resultados são exibidos nas colunas Gap2 e Tempo2. Para fins de comparação, nas Tabelas 5.6 e 5.7 exibimos nossos resultados somente para o Intel Xeon E5520 2.2 GHz.

Na Tabela 5.6, apresentamos o resultado para cada classe de instância dentro de C1, a coluna Sol mostra quantas instâncias foram resolvidas de um total de 6, a coluna Raiz mostra a média do gap do algoritmo no nó raiz, a coluna Gap mostra a média do gap final em todas as instâncias, e a coluna Tempo mostra o tempo médio gasto para todas as instâncias na classe. Na Tabela 5.7 exibimos em Sol o total de instâncias resolvidas, em Raiz a média do gap no nó raiz, em TempoR exibimos o tempo médio gasto considerando o mesmo conjunto de instâncias resolvidas para todos os algoritmos

da literatura, na coluna TempT exibimos o tempo médio gasto levando em conta todas as instâncias.

No conjunto de instâncias C1, nossos algoritmos foram capazes de resolver 37 instâncias, enquanto Côté et al. [2012a] resolvem 34 e Sampaio & Urrutia [2017] resolvem 33. Além disso, obtivemos um tempo de execução menor nas classes de instâncias, isso pode ser notado nas classes a280 e fnl4461. Em a280 fomos capazes de resolver uma instância a mais que os outros algoritmos, em d18512 resolvemos uma instância a mais que Côté et al. [2012a] e duas a mais que Sampaio & Urrutia [2017], e na classe ts225 resolvemos uma a instância a mais que os outros algoritmos. Na Tabela 5.6 temos que o tempo médio das instâncias resolvidas em d18512 é maior que os outros dois algoritmos da literatura, isso é devido a considerarmos mais instâncias resolvidas, considerando somente o mesmo conjunto nosso algoritmo obtém um tempo superior, a mesma análise pode ser feita para outras classes de instâncias onde isso aconteceu, como ts225.

Considerando somente instâncias individuais, nosso algoritmos obtém em quase todas as instâncias resolvidas os menores tempos, em algumas instâncias fomos capazes de diminuir drasticamente o tempo necessário para resolvê-las, por exemplo, para a280 com $n = 19$, somos capazes de resolver em 101 e 47 segundos com nossos algoritmos, enquanto antes ela era resolvida em 470 e 2202 segundos pelos outros dois algoritmos da literatura, o mesmo pode ser dito para $n = 21$ na mesma classe, que antes não era resolvida. Essa mesma análise também pode ser feita individualmente para as classes d18512, pr1002 e ts225.

A instância resolvida em que Côté et al. [2012a] gastam mais tempo é d15112 com $n = 17$, necessitando 3278 segundos, enquanto ela é resolvida por B&C Arco em 578 segundos e por B&C Nó em 817 segundos, vale ressaltar que essa também é a instância resolvida por ambos algoritmos que levamos mais tempo. A instância resolvida por ambos os algoritmos que Sampaio & Urrutia [2017] leva mais tempo é fnl4461 com $n = 15$, necessitando 3005 segundos, ela é resolvida por B&C Arco em 394 segundos e por B&C Nó em 1901 segundos, levando em conta somente B&C Nó, essa é a instância resolvida pelo algoritmo que gasta mais tempo.

Além de resolver instâncias ainda não resolvidas da literatura, os algoritmos propostos foram capazes de resolver todas as instâncias que antes eram resolvidas por Sampaio & Urrutia [2017], com relação a Côté et al. [2012a], o algoritmo resolve todas as instâncias resolvidas por eles, exceto fnl4461 com $n = 17$.

Tabela 5.5: Resultados detalhados para o conjunto de instâncias C1.

Instância			B&C Arco				B&C Nó				Sampaio & Urrutia [2017]		Côté et al. [2012a]	
Nome	n	Opt	Gap1	Tempo1	Gap2	Tempo2	Gap1	Tempo1	Gap2	Tempo2	Gap	Tempo	Gap	Tempo
a280	11	449		0.67		1.52		0.41		1.74		5.9		2.0
	13	477		2.63		12.48		3.04		4.42		19.4		6.9
	15	542		5.59		12.31		37.08		23.74		607.6		105.1
	17	624		7.04		15.39		39.66		87.86		93.3		181.3
	19	669		47.93		102.07		13.44		47.61		470.9		2202.1
	21	739		292.99		602.53		715.83		761.67	6.19	3600	10.35	3600
att532	11	4177		0.04		0.12		0.06		0.12		0.1		0.2
	13	4937		0.07		0.17		0.08		0.18		0.2		0.7
	15	5151		0.45		0.97		0.41		0.95		0.7		2.9
	17	5294		0.32		0.61		0.76		1.7		0.8		2.8
	19	5587		1.8		3.67		0.94		1.99		2.6		13.8
	21	9266		103.28		188.57		509.49		1088.64		214.0		2034.2
brd14051	11	4396		1.08		2.5		1.05		2.45		2.5		8.7
	13	4439		6.76		23.16		3.09		7.38		22.7		30.4
	15	n.a.	4.63	3600	5.31	3600	3.16	3600	3.5	3600	4.79	3600	3.41	3600
	17	n.a.	8.21	3600	8.59	3600	5.32	3600	5.79	3600	9.57	3600	4.65	3600
	19	n.a.	32.15	3600	32.3	3600	20.63	3600	6.82	3600	32.56	3600	4.72	3600
	21	n.a.	13.97	3600	13.97	3600	10.29	3600	11.46	3600	16.15	3600	4.40	3600
d15112	11	74,603		0.31		0.69		0.37		0.86		0.9		2.9
	13	80,690		3.95		8.28		4.07		9.06		14.8		37.0
	15	89,754		5.87		12.77		7.78		17.96		13.7		20.6
	17	96,804		260.68		580.11		372.34		815.63		852.6		3278.0
	19	n.a.	4.98	3600	5.57	3600	4.35	3600	5.01	3600	5.67	3600	5.39	3600
	21	n.a.	9.48	3600	9.75	3600	9.89	3600	10.44	3600	12.03	3600	9.68	3600
d18512	11	4280		0.12		0.28		0.1		0.22		0.4		1.4
	13	4301		0.4		1.09		0.21		0.54		1.7		5.7
	15	4638		150.91		333.09		156.08		289.5	1.94	3600		2574.1
	17	4741		1575.48		3400.08		1033.82		2290.95	2.32	3600	3.56	3600
	19	n.a.	2.69	3600	3.22	3600	1.74	3600	1.97	3600	4.48	3600	5.78	3600
	21	n.a.	8.57	3600	8.71	3600	6.16	3600	6.67	3600	9.16	3600	9.02	3600
fnl4461	11	1889		0.11		0.23		0.11		0.24		0.2		0.6
	13	2088		1.37		2.92		3.74		8.13		7.2		1.6
	15	2356		183.51		395.09		1305.09		1903.75		3005.5		16.8
	17	2517	2.77	3600	3.5	3600	3.68	3600	4.57	3600	4.29	3600		102.6
	19	n.a.	14.18	3600	14.59	3600	14.54	3600	14.92	3600	15.27	3600	3.02	3600
	21	n.a.	26.13	3600	26.88	3600	27.23	3600	27.22	3600	27.14	3600	3.55	3600
nrw1379	11	2690		0.21		0.48		0.12		0.25		0.5		0.9
	13	n.a.	5.08	3600	5.48	3600	3.5	3600	4.24	3600	5.00	3600	3.68	3600
	15	n.a.	4.76	3600	5.39	3600	4.44	3600	4.96	3600	5.95	3600	5.63	3600
	17	n.a.	4.63	3600	5.22	3600	4.22	3600	4.65	3600	5.72	3600	6.21	3600
	19	n.a.	11.84	3600	12.12	3600	11.74	3600	12.06	3600	13.2	3600	12.11	3600
	21	n.a.	15.34	3600	15.57	3600	14.92	3600	15.25	3600	16.17	3600	14.69	3600
pr1002	11	13,718		0.05		0.1		0.05		0.1		0.2		0.4
	13	15,436		0.48		1.57		0.49		0.96		1.4		5.1
	15	16,268		6.07		13.29		12.86		17.59		38.6		146.6
	17	17,601		18.55		40.51		84.77		186.19		164.5		384.2
	19	18,673		28.68		60.4		21.92		60.76		343.7		1761.7
	21	20,150		66.56		872.68		294.24		585.58		578.7	2.31	3600
ts225	11	22,000		0.04		0.08		0.05		0.09		0.3		1.5
	13	29,395		0.23		0.46		0.14		0.3		0.1		2.4
	15	32,541		0.72		1.59		0.33		0.93		286.3		4.0
	17	36,405		3.82		8.09		2.33		5.4		262.2		44.4
	19	40395		462.96		969.15		409.17		587.8	0.63	3600	2.18	3600
	21	42878		3584.45	3.69	3600	3.32	3600	3.91	3600	6.07	3600	5.65	3600

Tabela 5.6: Resultados sumarizados para as famílias de instâncias em C1.

	B&C Arco				B&C Nó				Côté et al. [2012a]				Sampaio & Urrutia [2017]			
Classe	Sol	Raiz	Gap	Tempo	Sol	Raiz	Gap	Tempo	Sol	Raiz	Gap	Tempo	Sol	Raiz	Gap	Tempo
a280	6	11.31	0.00	28.75	6	10.63	0.00	33.07	5	6.20	1.75	499.48	5	11.91	1.03	239.42
att532	6	1.42	0.00	32.35	6	1.87	0.00	182.26	6	2.00	0.00	342.43	6	2.16	0.00	36.40
brd14051	2	16.67	10.03	12.83	2	16.65	4.60	4.92	2	6.15	2.86	19.55	2	16.80	10.51	12.60
d15112	4	8.76	2.56	150.46	4	8.87	2.58	210.88	4	7.73	2.51	834.63	4	9.16	2.95	220.50
d18512	4	6.50	1.99	0.69	4	6.50	1.44	0.38	3	5.71	3.06	3.55	2	6.65	2.98	1.05
fnl4461	3	11.27	7.50	132.75	3	11.22	7.79	637.37	4	6.22	1.10	6.33	3	11.28	7.78	1004.30
nrw1379	1	11.17	7.30	0.48	1	11.09	6.86	0.25	1	10.92	7.05	0.90	1	11.52	7.67	0.50
pr1002	6	3.81	0.00	23.17	6	3.72	0.00	53.12	5	3.99	0.39	459.60	6	5.19	0.00	109.68
ts225	5	7.16	0.62	2.56	5	7.50	0.65	1.68	4	5.73	1.31	13.08	4	7.98	1.12	137.23

Tabela 5.7: Resultados sumarizados de todos os algoritmos para a classe de instâncias C1.

Algoritmo	Resolvidas	Raiz	Gap	TempoT	TempoR
Côté et al. [2012a]	34	6.07	2.22	1573.77	322.09
Sampaio & Urrutia [2017]	33	9.18	3.78	1529.89	201.11
B&C Arco	37	8.67	3.33	1270.42	46.61
B&C Nó	37	8.67	2.66	1292.04	134.30

5.5 Conjunto de Instâncias C2

Nesta seção, apresentamos os resultados dos algoritmos *branch-and-cut* propostos para o conjunto de instâncias C2. As Tabelas 5.8, 5.9 e 5.10 seguem as mesmas convenções descritas na Seção 5.4 anterior.

No conjunto de instâncias C2 nossos algoritmos B&C Arco e B&C Nó foram capazes de resolver, considerando a máquina similar, respectivamente, 27 e 25 instâncias, de um total de 54, enquanto Côté et al. [2012a] resolvem 24 Sampaio & Urrutia [2017] e resolvem 25 instâncias. Considerando a máquina com taxa de clock maior o número de instâncias resolvidas por B&C Arco passa de 27 para 29, e por B&C Nó, passa de 25 para 30 instâncias. Observando as Tabelas 5.9 e 5.10, nossos algoritmos obtiveram tempos melhores do que os outros dois algoritmos da literatura. Reduzimos drasticamente o tempo necessário para resolver as instâncias nas classes a280, brd14051, d15112 e ts225. Para a classe ts225, B&C Arco, obteve uma média de tempo de execução superior a Côté et al. [2012a], entretanto nesse caso resolvemos a instância não resolvida ts225 com $n = 15$, desconsiderando essa instância na análise de tempo, a média de tempo é inferior a dos outros dois algoritmos.

Analisando as instâncias individualmente através da Tabela 5.8, na classe de instâncias a280 obtivemos tempos menores para todas as instâncias com ambos algoritmos propostos, além disso resolvemos a instância não resolvida a280 com $n = 21$, embora, nesse caso o resultado tenha sido obtido pela máquina de taxa de clock superior. Na classe de instâncias brd14051 acontece o mesmo, quando $n = 13$ obtemos um tempo menor por várias ordens de magnitude, apesar de nesse caso não resolvermos nenhuma instância inédita, conseguimos melhorar o gap final para $n = 15$ e $n = 17$.

Como pode ser observado, a classe C2 possui a maior quantidade de instâncias não resolvidas, o que significa que a classe C1 possui um número maior de instâncias mais fáceis. Nessa classe, as instâncias possuem a capacidade de pilha mais apertada que C1, o resultado disso é visto nos problemas de empacotamentos de Côté et al. [2012a] onde ele gera mais cortes violados do que em C1. O conjunto de instâncias

nr1379 é bem difícil de se resolver, somente uma instância da classe é resolvida, como notado por Côté et al. [2012a] a versão do PDTSP destas instâncias também é bem difícil de se resolver. Apesar disso, notamos que fomos capazes de obter gaps finais menores em duas instâncias, no caso $n = 13$ e $n = 17$, analisando o resultado da máquina de clock maior a instância com $n = 13$ quase foi resolvida dentro de uma hora, atigindo um gap final de 0.88%.

A instância resolvida por ambos os algoritmos que Côté et al. [2012a] gasta mais tempo é d15112 com $n = 15$, necessitando de 1402 segundos, enquanto os algoritmos propostos B&C Arco e B&C Nó, gastam, respectivamente, 419 segundos e 1281 segundos. Para B&C Nó, essa também é a instância resolvida por ambos que ele gasta mais tempo. Para B&C Arco, a instância resolvida mais difícil foi pr1002 com $n = 21$, levando 891 segundos, enquanto Côté et al. [2012a] gasta 1322 segundos, para essa instância B&C Nó e Sampaio & Urrutia [2017] obtém tempos inferiores. A instância resolvida por ambos algoritmos em que Sampaio & Urrutia [2017] gasta mais tempo é d15112 com $n = 15$,com 748 segundos, B&C Arco também resolve em menos tempo, levando 419 segundos, nesse caso B&C Nó teve um superior, levou 1281 segundos.

Além de resolver 6 instâncias ainda não resolvidas da literatura, nesse caso considerando os resultados da máquina de clock maior, os algoritmos propostos foram capazes de resolver todas as instâncias que antes eram resolvidas por Sampaio & Urrutia [2017] e Côté et al. [2012a] na classe C2. Considerando a máquina idêntica a de Sampaio & Urrutia [2017] e similar a de Côté et al. [2012a] o algoritmo B&C Arco resolveu duas instâncias a mais que os outros algoritmos.

5.6 Novas Instâncias Resolvidas

Nas Seções 5.4 e 5.5, exibimos os resultados detalhados dos algoritmos para as instâncias de *benchmark*. Dentro do tempo limite de uma hora, para o conjunto de instâncias C1, o algoritmo B&C Arco resolveu as seguintes instâncias não resolvidas, a280, com $n = 21$, ts225, com $n = 19$, d18512, com $n = 15$, d18512, com $n = 17$, e ts225, com $n = 21$. Além disso, para o mesmo conjunto de instâncias, o algoritmo B&C Nó foi capaz de resolver as mesmas instâncias, exceto a ts225, com $n = 21$. No conjunto de instâncias C2, B&C Arco resolveu fnl4461, com $n = 17$, ts225, com $n = 13$, ts225, com $n = 15$, a280, com $n = 21$, e att532, com $n = 15$, enquanto B&C Nó resolveu d18512, com $n = 13$, d18512, com $n = 15$, ts225, com $n = 13$, ts225, com $n = 15$, a280, com $n = 21$, e att532, com $n = 15$. Na classe de instâncias C1 foram resolvidas 5 instâncias não resolvidas previamente, e na classe C2 foram resolvidas 7 instâncias para quais o

Tabela 5.8: Resultados detalhados para o conjunto de instâncias C2.

Instância			B&C Arco				B&C Nó				Sampaio & Urrutia [2017]		Côté et al. [2012a]	
Nome	n	Opt	Gap1	Tempo1	Gap2	Tempo2	Gap1	Tempo1	Gap2	Tempo2	Gap	Tempo	Gap	Tempo
a280	11	455		0.99		2.19		0.54		1.19		4.0		8.6
	13	479		4.29		9.32		2.79		6.19		5.6		15.3
	15	553		20.88		45.14		5.78		12.74		255.5		497.5
	17	635		30.61		65.36		11.12		24.57		234.3		1082.3
	19	677		35.42		75.18		36.98		80.19		475.7	6.94	3600
	21	753		838.28		1776.5		2943.1	5.51	3600	4.63	3600	12.68	3600
att532	11	4190		0.07		0.12		0.07		0.18		0.1		0.3
	13	5033		0.47		1.02		0.54		1.25		0.9		4.7
	15	5665		1106.64		2393.34		2890.07	2.27	3600	3.72	3600	5.23	3600
	17	n.a.	4.72	3600	5.24	3600	5.74	3600	6.25	3600	6.40	3600	6.98	3600
	19	n.a.	4.07	3600	4.64	3600	5	3600	5.45	3600	6.22	3600	7.68	3600
	21	n.a.	6.38	3600	6.78	3600	5.73	3600	6.08	3600	8.31	3600	8.12	3600
brd14051	11	4386		0.63		1.44		0.46		0.99		1.9		4.0
	13	4458		60.27		130.28		13.91		29.97		168.3		960.4
	15	n.a.	3.12	3600	4.05	3600	3.18	3600	3.89	3600	6.13	3600	3.99	3600
	17	n.a.	7.8	3600	8.11	3600	7.23	3600	7.69	3600	8.43	3600	8.21	3600
	19	n.a.	27.42	3600	27.56	3600	27.39	3600	27.6	3600	27.72	3600	2.81	3600
	21	n.a.	27.49	3600	27.69	3600	27.46	3600	27.59	3600	28.48	3600	3.37	3600
d15112	11	73,872		0.47		1.06		0.61		1.38		0.8		5.1
	13	81,657		102.85		221.95		128.22		282.14		224.1		452.7
	15	91,799		194.35		421.3		579.89		1273.69		748.0		1402.2
	17	n.a.	1.67	3600	2.26	3600	2.58	3600	3.21	3600	2.56	3600	3.22	3600
	19	n.a.	1.19	3600	1.76	3600	2.28	3600	2.72	3600	1.85	3600	3.48	3600
	21	n.a.	6.42	3600	6.95	3600	6.45	3600	6.96	3600	7.48	3600	8.50	3600
d18512	11	4341		12.36		27.92		6.99		16.53		27.6		70.6
	13	4572	0.45	3600	1.24	3600		1850.88	0.35	3600	3.47	3600	2.35	3600
	15	4893	4.01	3600	4.28	3600		2550.72	0.79	3600	4.33	3600	2.82	3600
	17	n.a.	6.89	3600	7.08	3600	2.93	3600	3.34	3600	8.00	3600	4.15	3600
	19	n.a.	10.52	3600	10.9	3600	5.48	3600	5.83	3600	12.77	3600	6.48	3600
	21	n.a.	18.72	3600	18.95	3600	13.99	3600	14.87	3600	20.13	3600	12.00	3600
fnl4461	11	1889		0.06		0.13		0.07		0.16		0.1		0.5
	13	2088		5.34		11.86		12.2		26.9		23.5		14.7
	15	2262		29		62.73		28.05		61.99		73.4		36.8
	17	2428		2618.34	0.98	3600	1.02	3600	1.83	3600	2.75	3600	3.09	3600
	19	n.a.	5.3	3600	5.71	3600	6.3	3600	6.64	3600	6.60	3600	8.35	3600
	21	n.a.	4.76	3600	5.24	3600	6.33	3600	6.79	3600	7.46	3600	8.79	3600
nrw1379	11	2690		0.19		0.42		0.11		0.23		0.5		1.3
	13	n.a.	0.93	3600	2.06	3600	1.91	3600	2.8	3600	3.19	3600	2.82	3600
	15	n.a.	5.07	3600	5.68	3600	5.22	3600	5.64	3600	6.19	3600	5.62	3600
	17	n.a.	4.83	3600	5.24	3600	4.9	3600	5.33	3600	6.15	3600	6.01	3600
	19	n.a.	10.41	3600	10.69	3600	10.44	3600	10.72	3600	11.66	3600	9.47	3600
	21	n.a.	14.64	3600	14.87	3600	14.63	3600	14.85	3600	15.76	3600	11.33	3600
pr1002	11	13,527		0.4		0.86		0.24		0.52		0.4		2.0
	13	15,221		6.13		13.32		2.56		5.69		17.8		14.2
	15	15,676		3.52		7.43		3.15		6.85		14.4		30.9
	17	17,009		23.68		50.84		6.12		12.98		20.8		75.7
	19	18,136		150.38		313.19		50.92		107.44		172.5		504.8
	21	19,613		138.87		287.03		60.21		128.62		135.9		1322.2
ts225	11	22,000		0.05		0.12		0.05		0.09		0.2		0.9
	13	34,000		245.92		543.05		205.06		442.96	1.45	3600		985.3
	15	37,703		1130.28		2428.68		2340.27	2.21	3600	5.2	3600	3.44	3600
	17	n.a.	4.08	3600	5.12	3600	3.28	3600	4.3	3600	7.09	3600	5.63	3600
	19	n.a.	7.61	3600	8.43	3600	7.68	3600	8.33	3600	11.09	3600	11.29	3600
	21	n.a.	10.98	3600	11.67	3600	11.1	3600	11.85	3600	13.23	3600	14.18	3600

Tabela 5.9: Resultados sumarizados para as famílias de instâncias em C2.

	B&C Arco				B&C Nó				Côté et al. [2012a]				Sampaio & Urrutia [2017]			
Classe	Sol	Raiz	Gap	Tempo	Sol	Raiz	Gap	Tempo	Sol	Raiz	Gap	Tempo	Sol	Raiz	Gap	Tempo
a280	6	12.84	0.00	30.50	5	12.74	0.92	11.17	4	8.32	3.27	400.93	5	13.12	0.77	124.85
att532	3	7.81	2.78	0.57	2	8.30	3.34	0.72	2	7.93	4.65	2.50	2	8.55	4.11	0.50
brd14051	2	14.72	11.24	65.86	2	14.73	11.13	15.48	2	6.64	3.06	482.20	2	14.85	11.79	85.10
d15112	3	8.08	1.83	214.77	3	8.19	2.15	519.07	3	8.23	2.53	620.00	3	8.49	1.98	324.30
d18512	1	12.51	7.08	27.92	1	12.58	4.20	16.53	1	7.26	5.13	70.60	1	12.59	8.12	27.60
fnl4461	3	5.18	1.99	24.91	3	5.10	2.54	29.68	3	5.87	3.37	17.33	3	5.17	2.80	32.33
nrw1379	1	10.58	6.42	0.42	1	10.58	6.56	0.23	1	9.78	5.86	1.30	1	10.98	7.16	0.50
pr1002	6	2.55	0.00	112.11	6	2.69	0.00	43.68	6	3.17	0.00	324.97	6	2.98	0.00	60.30
ts225	3	15.66	4.21	0.12	2	17.16	4.45	0.09	2	12.77	5.76	0.90	2	17.60	6.34	0.20

Tabela 5.10: Resultado sumarizado de todos os algoritmos para a classe de instâncias C2.

Algoritmo	Resolvidas	Raiz	Gap	TempoT	TempoR
Côté et al. [2012a]	24	7.78	3.74	2138.76	282.94
Sampaio & Urrutia [2017]	25	10.48	4.79	2048.26	92.63
B&C Arc	27	9.99	3.95	1891.39	72.83
B&C Node	25	10.23	3.92	1974.23	87.06

valor ótimo não era conhecido, totalizando um total de 12 instâncias não resolvidas, agora solucionadas.

Tabela 5.11: Instâncias anteriormente em aberto agora solucionadas.

C1		
Name	n	Opt
a280	**21**	**739**
d18512	17	4741
ts225	19	40395
ts225	**21**	**42878**

C2		
Name	n	Opt
a280	**21**	**753**
att532	15	5665
d15112	**19**	**99660**
d18512	13	4572
d18512	15	4893
fnl4461	17	2428
nrw1379	13	3055
ts225	15	37703

Para avaliar os algoritmos e procurar resolver mais instâncias, o tempo limite de 1 hora utilizado nos experimentos anteriores foi estendido para 3 horas, e os algoritmos B&C Arco e B&C Nó foram reexecutados para as instâncias não resolvidas que restavam. Com 3 horas, no conjunto de instâncias C1, B&C Nó foi capaz de resolver ts225, com $n = 21$ em 10172.85 segundos, com um ótimo de 42878, exceto essa, nenhuma outra instância em $C1$ foi resolvida. No conjunto de instâncias C2, B&C Arco foi capaz de resolver d15112, com $n = 19$, em 8376.61 segundos, com um ótimo de 99660, nrw1379, com $n = 13$ em 5214.69 segundos, com um ótimo de 3055, d18512, com $n = 13$ em 4336.59 segundos, com um ótimo de 4572. No mesmo conjunto de instâncias, B&C Nó resolveu fnl4461, com $n = 17$, em 5737.16 segundos, com um ótimo de 2428, e nrw1379, com $n = 13$, em 9012.52 segundos, com um ótimo de 3055. Na Tabela 5.11 exibimos todas as instâncias não resolvidas previamente na literatura que nossos algoritmos resolveram, nas instâncias em negrito, a melhor solução conhecida foi melhorada.

Capítulo 6

Conclusão

Nosso trabalho abordou formulações e algoritmos exatos para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. Ao longo do trabalho discutimos o problema e apresentamos os trabalhos da literatura relacionados. Em seguida apresentamos nossas formulações e os algoritmos *branch-and-cut* baseados nas mesmas. Por fim, avaliamos nossos algoritmos com as instâncias de *benchmark* da literatura e comparamos os resultados com os outros algoritmos exatos existentes.

Nessa seção, discutimos as contribuições do nosso trabalho e indicamos trabalhos futuros que podem contribuir para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas.

6.1 Contribuições

A principal contribuição do nosso trabalho foi a introdução de três novas formulações para o Problema do Caixeiro Viajante com Coleta e Entrega sob Múltiplas Pilhas. Na primeira mostramos como adaptar a formulação proposta por Sampaio & Urrutia [2017], nessa reformulação eliminamos um conjunto de variáveis do modelo e consequentemente eliminamos também um conjunto de restrições do modelo inicial. A segunda formulação é inspirada nas formulações de programação inteira de Côté et al. [2012a] e Sampaio & Urrutia [2017], nela, utilizamos as variáveis introduzidas pelo primeiro trabalho para modelar a atribuição de pilhas para os itens, e usamos as desigualdades propostas pelo segundo trabalho para modelar a política LIFO e a capacidade dos veículos. A terceira formulação é uma formulação compacta para o PDTSPMS, que no melhor do nosso conhecimento, ainda não existia nenhuma.

Além de propor novas formulações de programação inteira para o PDTSPMS, propusemos novas desigualdades válidas para o problema dentro das nossas formula-

ções, e também propusemos desigualdades válidas para o PDTSP, uma versão mais abrangente do nosso problema.

Após a apresentação das novas formulações e desigualdades, propomos algoritmos *branch-and-cut* baseados nas formulações e que incorporam as desigualdades válidas propostas. Executamos experimentos com os algoritmos propostos nas instâncias de *benchmark* da literatura e comparamos os resultados com os outros dois algoritmos exatos da literatura, propostos por Côté et al. [2012a] e Sampaio & Urrutia [2017]. Nossos algoritmos propostos conseguem resolver as instâncias que antes já eram resolvidas na literatura de forma mais eficiente e, além disso, conseguem resolver novas instâncias que antes possuíam o estado de não resolvidas.

6.2 Trabalhos Futuros

Dentro do conjunto de instãncias de *benchmark* para o PDTSPMS ainda existem várias instâncias não resolvidas, o que destaca a dificuldade do problema e a necessidade de novas abordagens e técnicas para lidar com o problema. Nos experimentos computacionais realizados com nossos algoritmos, notamos que os limites obtidos no nó raiz não são fortes. Além disso, apesar de propormos um conjunto de desigualdades válidas para o problema, não utilizamos nenhum algoritmo exato para separá-las, utilizando somente uma quantidade linear ou quadrática dessas desigualdades no nó raiz. Consequentemente, a investigação de métodos de separação específicos para essas desigualdades pode resultar em limites melhores para o problema. Também, novas desigualdades, além daqueles aqui propostas, podem contribuir nesse objetivo.

Outra direção de pesquisa com relação ao PDTSPMS é a aplicação dos nossos algoritmos propostos para o DTSPMS. O DTSPMS é uma versão mais restrita do nosso problema onde todas as coletas devem ser realizadas antes que qualquer entrega possa ser feita. Côté et al. [2012a] adaptaram o algoritmo para o PDTSPMS proposto por eles para o DTSPMS. Planejamos no futuro adaptar nossos algoritmos propostos para o DTSPMS e comparar os resultados com os outros algoritmos da literatura.

Ainda com relação ao PDTSPMS, um tipo de trabalho futuro para o problema inclui a aplicação de outras abordagens de otimização combinatória para o problema. No momento todos os algoritmos que existem para o problema são baseados em técnicas de *branch-and-cut*. Dessa forma, uma vez que não existem algoritmos que se baseiam nessas técnicas, uma nova linha de pesquisa é investigar algoritmos *branch-and-price* para o problema e algoritmos baseados em relaxação lagrangeana.

Finalmente, outra possível investigação, são variações do PDTSPMS. A política

LIFO imposta no problema é bem restritiva, uma vez que somente itens no topo de cada pilha podem ser entregues, desse modo, uma variação para o problema seria permitir a realocação de itens, nesse caso incorrendo um custo caso a mesma seja feita. Outra variação para o problema inclui a adição de uma frota de veículos ao invés de um único veículo ou, então, a adição de janelas de tempo para a coleta e a entrega.

Referências Bibliográficas

Ángel Felipe; Ortuño, M. T. & Tirado, G. (2009). The double traveling salesman problem with multiple stacks: A variable neighborhood search approach. *Computers & Operations Research*, 36(11):2983 – 2993. ISSN 0305-0548.

Ángel Felipe; Ortuño, M. T. & Tirado, G. (2011). Using intermediate infeasible solutions to approach vehicle routing problems with precedence and loading constraints. *European Journal of Operational Research*, 211(1):66 – 75. ISSN 0377-2217.

Arbib, C.; Marinelli, F. & Servillio, M. (2009). On the pickup and delivery travelling salesman problem with lifo loading. Em *Proceedings of the International Network Optimisation Conference*.

Azi, N.; Gendreau, M. & Potvin, J.-Y. (2007). An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research*, 178(3):755--766.

Balas, E.; Fischetti, M. & Pulleyblank, W. R. (1995). The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68(3):241--265.

Carrabs, F.; Cerulli, R. & 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. & 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.

Casazza, M.; Ceselli, A. & Nunkesser, M. (2012). Efficient algorithms for the double traveling salesman problem with multiple stacks. *Computers & Operations Research*, 39(5):1044--1053. ISSN 0305-0548.

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

Cordeau, J.-F.; Iori, M.; Laporte, G. & 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.

Cordeau, J.-F.; Laporte, G.; Potvin, J.-Y. & Savelsbergh, M. W. P. (2007). *Transportation on Demand*, capítulo 7, pp. 429--466. Volume 14 edição.

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

Dumitrescu, I.; Ropke, S.; Cordeau, J.-F. & 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. & 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.

Gavish, B. & Graves, S. C. (1978). *The Travelling Salesman Problem and Related Problems*. Working Paper. Massachusetts Institute of Technology, Operations Research Center.

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

Iori, M.; Salazar-González, J.-J. & Vigo, D. (2007). An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41(2):253--264. ISSN 1526-5447.

Junqueira, L.; Oliveira, J. F.; Carravilla, M. A. & Morabito, R. (2013). An optimization model for the vehicle routing problem with practical three-dimensional loading constraints. *International Transactions in Operational Research*, 20(5):645--666. ISSN 1475-3995.

Kek, A. G.; Cheu, R. L. & Meng, Q. (2008). Distance-constrained capacitated vehicle routing problems with flexible assignment of start and end depots. *Mathematical and Computer Modelling*, 47(1–2):140 – 152.

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

Levitin, G. & Abezgaouz, R. (2003). Optimal routing of multiple-load agv subject to lifo loading constraints. *Computers & Operations Research*, 30(3):397--410.

Li, Y.; Lim, A.; Oon, W.-C.; Qin, H. & Tu, D. (2011). The tree representation for the pickup and delivery traveling salesman problem with lifo loading. *European Journal of Operational Research*, 212(3):482 – 496. ISSN 0377-2217.

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

Martínez, L. & Amaya, C.-A. (2013). A vehicle routing problem with multi-trips and time windows for circular items. *Journal of the Operational Research Society*, 64(11):1630--1643. ISSN 1476-9360.

Naddef, D. & Rinaldi, G. (2001). The vehicle routing problem. capítulo Branch-and-cut Algorithms for the Capacitated VRP, pp. 53--84. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

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

Petersen, H. L. & 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.

Pollaris, H.; Braekers, K.; Caris, A.; Janssens, G. K. & Limbourg, S. (2015). Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum*, 37(2):297--330. ISSN 1436-6304.

Ropke, S. & Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267--286.

Ruland, K. & Rodin, E. (1997). The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1 – 13. ISSN 0898-1221.

Sampaio, A. H. & Urrutia, S. (2017). New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 24(1-2):77--98. ISSN 1475-3995.

Toulouse, S. & Wolfler Calvo, R. (2009). *On the Complexity of the Multiple Stack TSP, kSTSP*, pp. 360--369. Springer Berlin Heidelberg, Berlin, Heidelberg.

Øvstebø, B. O.; Hvattum, L. M. & Fagerholt, K. (2011). Routing and scheduling of roro ships with stowage constraints. *Transportation Research Part C: Emerging Technologies*, 19(6):1225 – 1242. ISSN 0968-090X.