Uma Abordagem Exata para o Problema de Roteamento de Veículos Capacitados com Restrições Bidimensionais de Carregamento

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Bruno Luis Pires de Azevedo e aprovada pela Banca Examinadora.

Campinas, 21 de dezembro de 2009.

Prof. Dr. Flávio Keidi Miyazawa -
IC-UNICAMP (Orientador)

Prof. Dr. Eduardo Cândido Xavier -
IC-UNICAMP (Co-orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Azevedo, Bruno Luis Pires de
Az25a Uma abordagem exata para o problema de roteamento de veículos capacitados com restrições bidimensionais de carregamento/Bruno Luis Pires de Azevedo-- Campinas, [S.P. : s.n.], 2009.

Orientador : Flávio Keidi Miyazawa.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Otimização combinatória. 2.Problema de roteamento de veículos. 3. Algoritmos. I. Miyazawa, Flávio Keidi. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: An exact approach for the capacitated vehicle routing problem with two-dimensional loading constraints

Palavras-chave em inglês (Keywords): 1. Combinatorial Optimization. 2. Vehicle routing problem. 3. Algorithms.

Área de concentração: Teoria da Computação

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Flávio Keidi Miyazawa (IC-UNICAMP)
Prof. Dr. Humberto José Longo (INF-UFG)
Prof. Dr. Orlando Lee (IC-UNICAMP)

Data da defesa: 21/12/2009

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 21 de dezembro de 2009, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Humberto José Longo
INF / UFG

Prof. Dr. Orlando Lee
IC / UNICAMP

Prof. Dr. Flávio Keidi Miyazawa
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Uma Abordagem Exata para o Problema de Roteamento de Veículos Capacitados com Restrições Bidimensionais de Carregamento

Bruno Luis Pires de Azevedo[1]

Dezembro de 2009

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa
  Instituto de Computação - UNICAMP (Orientador)
- Prof. Dr. Humberto José Longo
  Instituto de Informática - UFG
- Prof. Dr. Orlando Lee
  Instituto de Computação - UNICAMP
- Prof. Dr. Glauber Ferreira Cintra
  Universidade Estadual do Ceará (Suplente)
- Prof. Dr. Luis Augusto Angelotti Meira
  Universidade Federal de São Paulo (Suplente)

[1]Suporte financeiro de: Bolsa da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

Resumo

Nesta dissertação apresentamos um algoritmo exato para o Problema de Roteamento de Veículos Capacitados com Restrições Bidimensionais. Este combina o problema de carregar um conjunto de itens bidimensionais em veículos com o problema de minimizar o custo total de transporte. Existem várias aplicações práticas para este problema, dado que em muitas situações os itens não podem ser empilhados por diversas razões.

Propomos um algoritmo exato baseado em uma abordagem *branch-and-cut*. Sete desigualdades válidas para o Problema de Roteamento de Veículos Capacitados foram adaptadas e utilizadas. As restrições de empacotamento são garantidas através de um algoritmo exato. Também apresentamos uma nova heurística de empacotamento bidimensional. Exploramos duas variantes do problema, as versões sequencial e irrestrita. Para ambos os casos, consideramos os itens possuirem orientação fixa.

Efetuamos testes computacionais e comparamos os resultados obtidos com a abordagem exata, para o caso sequencial, apresentada por Iori, Salazar-González e Vigo. Observamos resultados satisfatórios e nove instâncias da literatura foram resolvidas à otimalidade pela primeira vez. Como o caso irrestrito ainda não havia sido abordado de modo exato, apresentamos também as soluções de cinquenta instâncias nunca resolvidas à otimalidade.

Abstract

We present an exact algorithm for the Vehicle Routing Problem with Two-dimensional Loading Constraints. This problem combines the problems of loading vehicles with two-dimensional items and minimizing transportation costs. It has many practical applications, since in many cases items can not be stacked on top of each other.

We propose an exact algorithm based on a branch-and-cut approach. Seven valid inequalities for the the Capacitated Vehicle Routing Problems were modified and used. The packing constraints are imposed by an exact algorithm. We also present a new heuristic for two-dimensional packing. We explored two variants of the problem, the sequential and the unrestricted cases. For both variants we assume the items to have fixed orientation.

We performed computational tests and compared the results with the exact approach by Iori, Salazar-González and Vigo for the sequential case. We found the results to be satisfactory and nine instances from the literature were solved to optimality for the first time. Since the unrestricted case haven't been tested so far by an exact algorithm, we also present the solutions for fifty instances never solved to optimality before.

Agradecimentos

À minha família, por todo apoio e incentivo.

Ao Flávio, por sua orientação sem igual, paciência e compreensão.

À Gau, por seu amor, carinho e companhia.

Ao Manoel e ao James, pelo empurrão inicial.

A todos os amigos pelos ótimos momentos, em particular ao Rodrigo e ao Bruno.

Ao meu co-orientador Eduardo, pela ajuda e sugestões.

Ao pessoal da secretaria do IC, por toda a ajuda oferecida.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pelo apoio financeiro prestado.

E por fim, a todos do IC que me proporcionaram uma experiência de mestrado mais rica.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O Problema de Roteamento de Veículos Capacitados com Restrições Bidimensionais de Carregamento (*Vehicle Routing Problem with Two-Dimensional Loading Constraints* – 2L-CVRP) é um caso mais geral do Problema de Roteamento de Veículos Capacitados, onde cada cliente demanda uma certa quantidade de itens retangulares. Dado um conjunto de veículos e um depósito, objetiva-se suprir as demandas de todos os clientes considerando que cada veículo possui uma capacidade máxima de peso e um contêiner de dimensões bidimensionais. Os itens devem ser empacotados de modo ortogonal e não podem se sobrepor ou ultrapassar os limites dos contêineres onde foram empacotados. O 2L-CVRP é fortemente NP-difícil, já que consiste em resolver o Problema de Roteamento de Veículos Capacitados e um problema de carregamento análogo a vários problemas de empacotamento conhecidos, particularmente o Problema de Empacotamento em Faixa e o Problema de Empacotamento em Contêineres Bidimensional.

Existem várias aplicações práticas para este problema, dado que em muitas situações os itens não podem ser empilhados por diversas razões. Os itens podem ser frágeis ou possuir uma forma irregular em seu topo. Podem também ser altos, de modo que ocupem toda, ou quase toda, a altura do contêiner.

Este é um problema ainda pouco estudado, sendo que em 2007 apareceu o primeiro trabalho abordando o 2L-CVRP [22]. Nos dois anos seguintes mais três trabalhos foram propostos [28] [26] [7], adicionando novas instâncias e variantes para o problema. No total foram apresentadas três abordagens heurísticas e uma abordagem exata.

Nesta dissertação apresentamos um algoritmo exato para o 2L-CVRP baseado em uma abordagem *branch-and-cut*. Exploramos duas variantes do problema, o 2L-CVRP sequencial e irrestrito. Na versão sequencial o padrão de empacotamento deve obedecer uma certa ordenação dos itens. Ao descarregar os itens de um cliente, estes não podem estar bloqueados por itens de clientes que serão visitados posteriormente. A versão irrestrita relaxa esta restrição de ordem. Para ambas as versões, consideramos os itens possuirem orientação fixa, ou seja, não foram permitidas rotações.

Esta dissertação encontra-se organizada do seguinte modo: no segundo capítulo apresentamos os conceitos básicos que serão usados ao longo desta dissertação, além de alguns problemas relacionados ao 2L-CVRP. No terceiro capítulo descrevemos o 2L-CVRP e apresentamos uma

revisão bibliográfica. No quarto capítulo apresentamos as desigualdades utilizadas no algoritmo *branch-and-cut* proposto. Cada seção deste capítulo descreve uma desigualdade diferente e o algoritmo utilizado para efetuar a sua separação. No quinto capítulo apresentamos os algoritmos utilizados por nossa abordagem na resolução do problema de empacotamento bidimensional do 2L-CVRP. No sexto capítulo detalhamos o algoritmo exato proposto. Descrevemos a formulação linear inteira e as estratégias desenvolvidas para tratar as restrições do problema. No sétimo capítulo apresentamos os resultados computacionais obtidos e analisamos o desempenho do algoritmo proposto quando comparado a literatura atual. Finalmente, no oitavo capítulo fazemos algumas considerações finais.

Capítulo 2

Preliminares

2.1 Introdução

Neste capítulo apresentaremos alguns conceitos básicos que serão usados ao longo desta dissertação. Definiremos também alguns problemas que relacionam-se diretamente com o problema abordado.

Este capítulo encontra-se organizado do seguinte modo: na seção 2.2 explicamos o funcionamento de algoritmos *branch-and-bound*. Planos de corte são explicados na seção 2.3. O funcionamento de algoritmos *branch-and-cut* é detalhado na seção 2.4. Na seção 2.5 descrevemos os problemas de Empacotamento em Faixa Bidimensional e de Empacotamento em Contêineres Bidimensional. Finalmente, na seção 2.6 descrevemos o Problema de Roteamento de veículos. Esta encontra-se dividida em duas sub-seções, a primeira descrevendo brevemente o problema e a segunda apresentando uma rápida revisão bibliográfica.

2.2 Branch-and-Bound

Branch-and-bound é um algoritmo usado para encontrar soluções de vários problemas, principalmente de otimização, consistindo de uma enumeração sistemática de todas soluções candidatas. O termo *branch* vem do fato que o método efetua partições no espaço das soluções e o termo *bound* refere-se ao funcionamento do algoritmo, onde um subconjunto de soluções é descartado através da utilização de limitantes superiores e inferiores. Este método foi inicialmente proposto por Land e Doig em 1960 [6] para problemas de programação linear inteira.

Assumiremos que o objetivo consiste em encontrar o valor máximo de uma função $f(x)$, sem perda de generalidade. Um método *branch-and-bound* necessita de dois procedimentos: uma rotina para particionamento que dado um conjunto de S soluções candidatas, devolve k subconjuntos menores S_i onde $k \geq 2$. Um procedimento para calcular os limitantes para o valor máximo de $f(x)$, dado um subconjunto S_i. Pode-se observar que a aplicação recursiva da rotina de ramificação define uma estrutura de árvore, onde cada nó é um subconjunto S_i, onde $S_i \in S$. Soluções candidatas são descartadas por três razões, que serão descritas a seguir.

Dadas uma rotina de ramificação e uma rotina de verificação dos limitantes, considere a decomposição de um nó S em dois nós S_1 e S_2 na figura 2.1:

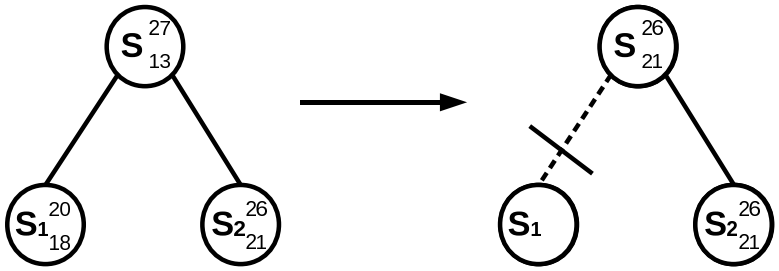

Figura 2.1: Poda por limitante.

Observando-se os nós gerados S_1 e S_2 temos o limitante superior global $LS_g = max(20, 26) = 26$ e o limitante inferior global $LI_g = max(18, 21) = 21$. Como o limitante superior em S_1 é 20 e o LI_g é 21, não existe solução ótima no subconjunto de soluções S_1. O conjunto de soluções S_1 pode então ser descartado por razão de limitante.

Considere agora uma mesma decomposição do nó S na figura 2.2:

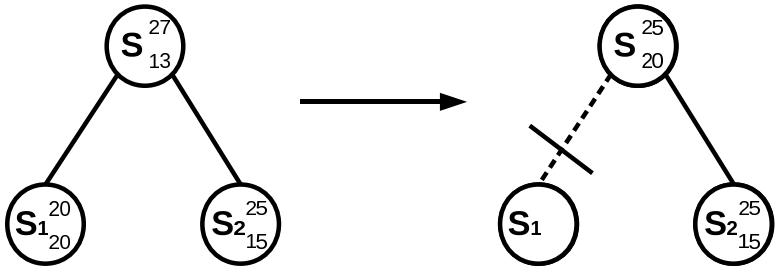

Figura 2.2: Poda por otimalidade.

Note que o nó S_1 possui ambos os limitantes superior e inferior iguais a 20, não existindo a possibilidade de se encontrar uma solução melhor. Deste modo o ramo S_1 pode ser descartado por razão de otimalidade.

Considere o seguinte trecho de uma árvore de *branch-and-bound* representando a resolução de um programa linear inteiro que possui as seguinte restrições: $x_1 + x_2 \geq 3$ e $3x_1 + 7x_2 \geq 26$.

Considere a solução S_2 na figura 2.3: a variável x_2 é selecionada para ser ramificada, gerando a restrição $x_2 \leq 2$ para as soluções de seu ramo esquerdo. Observando a restrição $x_1 \leq 2$ gerada no ramo esquerdo da solução S_1, claramente não é necessário examinar o ramo de soluções S_4, já que qualquer solução a partir deste ponto é inviável. Efetua-se então o descarte do ramo por inviabilidade.

Note que a qualidade do algoritmo *branch-and-bound* depende diretamente dos limites gerados em cada nó, deste modo, a estratégia de desdobramento da árvore tem uma influência direta no tempo utilizado para a resolução do problema. Uma estratégia ineficiente pode causar uma enumeração excessiva, reduzindo as podas de soluções. Diversas estratégias podem ser utilizadas, devendo-se procurar a mais eficiente para ser aplicada ao problema considerado.

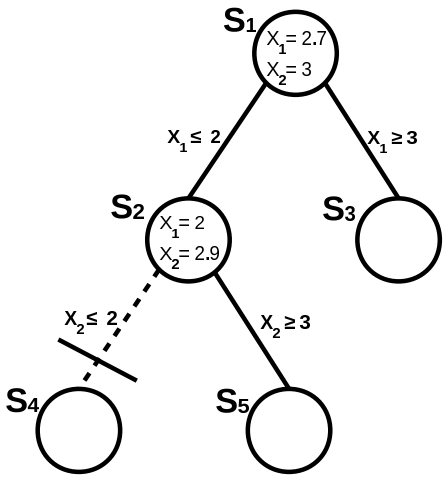

Figura 2.3: Poda por inviabilidade.

Um algoritmo *branch-and-bound* não possui o seu uso exclusivo à programas lineares, podendo ser utilizado em diversos problemas de otimização combinatória ou discreta. Por exemplo, considere o problema de se verificar a viabilidade do empacotamento de um conjunto de retângulos em um contêiner bidimensional. Um método *branch-and-bound* poderia ser usado para enumerar todas as possíveis configurações e as podas podem ser feitas através do cálculo de um limitante por área. Quando a área restante for menor que a área dos itens a serem empacotados, este ramo é podado.

2.3 Planos de Corte

Planos de corte são desigualdades válidas para um programa linear que permitem a remoção de uma solução não desejada. Encontrar um plano de corte consiste em resolver um problema considerando um programa linear inteiro e a solução que queremos remover.

Apesar de planos de corte não se aplicarem unicamente a problemas lineares inteiros, considere como exemplo o seguinte programa linear inteiro P^1:

$$\begin{aligned} \text{minimizar} \quad & cx \\ \text{s. a.} \quad & \\ & Ax \geq b \qquad (1) \\ & x \in \mathbb{Z}^+, \qquad (2) \end{aligned} \tag{2.1}$$

e sua relaxação linear P^1_r:

$$\begin{aligned} \text{minimizar} \quad & cx \\ \text{s. a.} \quad & \\ & Ax \geq b \qquad (1) \\ & x \geq 0. \qquad (2) \end{aligned} \tag{2.2}$$

Inicialmente resolvemos a relaxação P_r^1 encontrando uma solução x_r^*. Se x_r^* é inteira, então é uma solução ótima para P^1. Caso contrário, queremos encontrar uma desigualdade que satisfaça todas as soluções inteiras de P^1, mas que seja violada por x_r^*. Ou seja, desejamos uma desigualdade que quando adicionada a relaxação P_r^1, não seja satisfeita por x_r^*. Denotamos esta desigualdade como plano de corte ou apenas corte.

O problema de encontrar esta desigualdade violada é conhecido como problema de separação, já que corresponde a encontrarmos um hiperplano que separa x_r^* do politopo descrito por P_r^1. Consequentemente, com a adição deste hiperplano geramos outro programa linear que descreve um novo politopo que não possui x_r^*.

No exemplo anterior utilizamos um corte para remover uma solução fracionária, mas os cortes podem também ser utilizados para cortar soluções inteiras não desejadas. Por exemplo, considere o seguinte programa linear inteiro P^2:

$$\begin{aligned} \text{minimizar} \quad & cx \\ \text{s. a.} \quad & \\ & Ax \geq b \qquad (1) \\ & Dx \geq e \qquad (2) \\ & x \in \mathbb{Z}^+, \qquad (3) \end{aligned} \tag{2.3}$$

e sua relaxação P_r^2:

$$\begin{aligned} \text{minimizar} \quad & cx \\ \text{s. a.} \quad & \\ & Ax \geq b \qquad (1) \\ & x \geq 0. \qquad (3) \end{aligned} \tag{2.4}$$

Nesta relaxação removemos um conjunto de desigualdades representado por (2) além de relaxarmos as restrições de integralidade das variáveis. Suponha que com a resolução da relaxação P_r^2 obtemos uma solução inteira x_r^*. Apesar desta solução ser inteira, x_r^* só é ótima para P^2 se satisfazer todas as restrições de (2). Podemos verificar se a solução x_r^* viola alguma restrição em (2) resolvendo um problema de separação relacionado. Após resolvermos este problema, caso tenhamos encontrado alguma desigualdade violada, sabemos que x_r^* não é válida para P^2 e adicionamos os cortes encontrados em P_r^2. Caso contrário, x_r^* é ótima para P^2.

2.4 Branch-and-Cut

Um algoritmo *branch-and-cut* consiste de um *branch-and-bound* em que planos de corte são gerados nos nós da árvore de busca. Ao contrário de algoritmos *branch-and-bound* que podem ser aplicados em diversos problemas de otimização, o *branch-and-cut* é um método de otimização combinatória utilizado exclusivamente para resolver problemas lineares inteiros ou inteiros mistos.

Considere um problema linear inteiro que possua uma quantidade exponencial de restrições, ou então, a existência de desigualdades válidas de quantidade exponencial que podem ser adicionadas ao modelo fortalecendo a relaxação linear. Em ambas as situações, a alimentação de todas as restrições em um resolvedor de programação inteira é inviável. Neste caso, uma técnica de planos de corte pode ser usada para resolver o programa linear inteiro.

Seja P um programa linear inteiro com um conjunto de restrições de tamanho exponencial.

$$\begin{array}{rll} \text{minimizar} & cx & \\ \text{s. a.} & & \\ & Ax \geq b & (1) \\ & Dx \geq e & (2) \\ & x \in \mathbb{Z}^{+}. & (3) \end{array} \tag{2.5}$$

Considere que existe uma quantidade polinomial das restrições (1), podendo ser adicionadas ao modelo. Considere que as restrições (2) são muitas, com quantidade exponencial, para serem adicionadas a priori. Seja P_r a relaxação de P, removendo-se as restrições de integralidade e o conjunto de restrições representado por (2). Esta relaxação pode então ser resolvida, gerando uma solução ótima x_r^*. Se esta é viável para P, esta solução é ótima. Caso contrário, suponha a existência de um algoritmo (denominado de algoritmo de separação) que, dada a solução x_r^*, gera pelo menos uma desigualdade violada por x_r^*, mas que satisfaz todos os outros pontos de P. A desigualdade é adicionada ao modelo atual e o programa linear é resolvido novamente.

Caso uma solução ótima não tenha sido encontrada para P, o problema é decomposto em dois problemas. Neste caso, temos a etapa de ramificação do *branch-and-bound* e os novos problemas são resolvidos do mesmo modo. A etapa de verificação de limitantes também ocorre normalmente, podando ramos quando possível através de limitantes.

Existe uma relação de benefício mútuo entre a enumeração e os algoritmos de separação. Os cortes geralmente ajudam a melhorar os limitantes para a etapa de verificação destes, e o algoritmo de separação se beneficia da perturbação gerada pela ramificação da solução fracionária.

2.5 Problemas de Empacotamento

Nesta seção apresentamos dois problemas de empacotamento que possuem uma forte relação com o Problema de Roteamento de Veículos Capacitados com Restrições Bidimensionais de Carregamento. O Problema de Empacotamento em Faixa Bidimensional e o Problema de Empacotamento em Contêineres Bidimensional.

Denotaremos de empacotamento de um item em um contêiner, como o posicionamento ortogonal deste dentro do contêiner de mesma dimensão.

2.5.1 Problema de Empacotamento em Faixa Bidimensional

No Problema de Empacotamento em Faixa Bidimensional deseja-se empacotar um conjunto de retângulos em um retângulo de largura definida e de comprimento infinito. Denotaremos este retângulo de faixa. O objetivo é minimizar o comprimento do empacotamento na faixa. Os retângulos não podem se sobrepor, ultrapassar os limites da faixa e devem ser empacotados de forma ortogonal, ou seja, as arestas dos retângulos a serem empacotados devem estar paralelas ou perpendiculares as arestas da faixa.

Podemos então definir formalmente o Problema de Empacotamento em Faixa Bidimensional: dada uma lista de retângulos $R = (r_1, \ldots, r_n)$, onde cada retângulo r_i tem dimensões (w_i, l_i) e uma faixa $\mathcal{F}$ com dimensões (W, ∞), o objetivo é empacotar os retângulos R em $\mathcal{F}$ de forma a minimizar o tamanho do empacotamento na direção ilimitada da faixa $\mathcal{F}$.

O Problema de Empacotamento em Faixa Bidimensional é NP-difícil. Isto pode ser observado com a redução do Problema de Empacotamento em Contêineres Unidimensional, que é um conhecido problema NP-difícil, para o Problema de Empacotamento em Faixa Bidimensional.

2.5.2 Problema de Empacotamento em Contêineres Bidimensional

No Problema de Empacotamento em Contêineres Bidimensional deseja-se empacotar um conjunto de retângulos em um número ilimitado de contêineres retangulares. Os contêineres são idênticos e os retângulos a serem empacotados não podem se sobrepor, ultrapassar os limites do contêiner e devem ser empacotados ortogonalmente. O objetivo é minimizar o número de contêineres utilizados.

Definiremos formalmente o Problema de Empacotamento em Contêineres Bidimensional: dada uma lista de retângulos $R = (r_1, \ldots, r_n)$, onde cada retângulo r_i tem dimensões (w_i, l_i) e um conjunto de contêineres $\mathcal{B}$, cada um com dimensões (W, L), o objetivo é empacotar os retângulos R em $\mathcal{B}$ de forma a minimizar o número de contêineres utilizado.

Pode-se observar que o Problema de Empacotamento em Contêineres Bidimensional é NP-difícil por ser uma generalização do Problema de Empacotamento em Contêineres Unidimensional.

2.6 Problema de Roteamento de Veículos

2.6.1 Definição do Problema

O Problema de Roteamento de Veículos (*Vehicle Routing Problem* – VRP) consiste em atender um conjunto de clientes utilizando-se de uma frota de veículos que partem de um único depósito. Os veículos devem sair do depósito, atender clientes e voltar ao depósito. Todos os clientes devem ser visitados e cada cliente só pode ser atendido por um único veículo. O caminho percorrido pelo veículo é habitualmente denominado de rota. O objetivo é minimizar o custo total de deslocamento dos veículos.

Formalmente o Problema de Roteamento de Veículos pode ser descrito do seguinte modo: dado um grafo completo não dirigido $G = (V, E)$, V é um conjunto de $n + 1$ vértices correspondendo ao depósito (vértice 0) e n clientes (vértices $1, \ldots, n$). E representa um conjunto de arestas, onde para cada aresta $e \in E$ existe um custo associado $c_e \in \mathbb{Q}^+$. Existe uma frota homogênea de K veículos.

O problema consiste em encontrar um conjunto de circuitos $\mathcal{C} = \{C_1, \ldots, C_K\}$, também chamados de rotas, tal que:
(i) Toda rota C_i contém exatamente uma vez o vértice 0.
(ii) Todo vértice $i \in V_+ = V \setminus \{0\}$ consta exatamente uma vez em alguma rota de $\mathcal{C}$;
(iii) O custo do roteamento $\mathcal{C}$, dado por $c(\mathcal{C}) = \sum_{i=1}^{k} \sum_{e \in C_i} c_e$, é mínimo.

O Problema de Roteamento de Veículos Capacitados (Capacitated Vehicle Routing Problem - CVRP) é uma variante do Problema de Roteamento de Veículos onde cada vértice $i \in V_+ = V \setminus \{0\}$ possui uma demanda unidimensional d_i, os veículos possuem uma capacidade D e temos a seguinte restrição:

(iv) A soma das demandas de todos os clientes em C não pode exceder a capacidade do veículo para cada $C \in \mathcal{C}$. Ou seja, se V_C é o conjunto de clientes em C então $\sum_{i \in V_C} d_i \leq D$.

Por ser uma generalização do Problema do Caixeiro Viajante, um conhecido problema NP-difícil, o Problema de Roteamento de Veículos, e consequentemente sua variante, o Problema de Roteamento de Veículos Capacitados, são problemas NP-difíceis.

2.6.2 Revisão Bibliográfica

O trabalho precursor do Problema de Roteamento de Veículos surgiu em 1954, com um artigo de Dantzig, Fulkerson e Johnson [37]. Neste trabalho, estudou-se uma instância relativamente grande do Problema do Caixeiro Viajante (TSP). A incorporação de mais de um veículo foi feita em 1959, em um trabalho nomeado *The truck dispatching problem* [20], onde Dantzig e Ramser apresentaram um método baseado em programação linear para encontrar soluções próximas do ótimo. Esta abordagem foi melhorada cinco anos depois, por Clarke e Wright em [25], onde uma heurística gulosa foi proposta para o Problema de Roteamento de Veículos.

A designação conhecida atualmente, Roteamento de Veículos, surgiu em 1972, no artigo *Implementing Vehicle Routing Algorithms* de Golden et al. [41]. Nos anos 70, diversas variantes do Problema de Roteamento de Veículos foram investigadas, alguns trabalhos relevantes deste período foram de O'Connor e Wald [8], Marks e Stricker [38], Eilon et al. [9], Wilson e Sussman [23], Levin [48] e Liebman [49].

A incorporação de janelas de tempo ao Problema de Roteamento de Veículos foi feita por Solomon em 1983 [55]. Solomon também contribuiu com as primeiras instâncias para *benchmark* da literatura.

Nos anos 80, Laporte e Nobert investigaram algoritmos exatos para o Problema de Roteamento de Veículos [47]. Na década seguinte surgiram as primeiras meta-heurísticas aplicadas ao Problema de Roteamento de Veículos. Gendreau, Laporte e Potvin apresentaram em 1998 um livro sobre meta-heurísticas aplicadas ao problema [17]. Um estudo de abordagens exatas e

aproximadas foi apresentado em 1992 por Laporte [46].

Quando considerando apenas o Problema de Roteamento de Veículos Capacitados, mais especificamente abordagens exatas, alguns trabalhos relevantes podem ser listados. Em 1981, Christofides, Mingozzi e Toth apresentaram em [4] um algoritmo *branch-and-bound* resolvendo instâncias com até 25 clientes.

Posteriormente apareceram vários algoritmos melhorando este resultado, como: [44], [52] [5], [34], [14] e [16]. Mas Augertat et al. [21], Baldacci, Hadjiconstantinou e Mingozzi [12], Lysgaard, Letchford, e Eglese [3] e Baldacci, Christofides e Mingozzi [33] apresentaram trabalhos com os resultados mais fortes, resolvendo instâncias de até 135 clientes.

O algoritmo exato mais eficiente para o Problema de Roteamento de Veículos Capacitados na literatura atual foi apresentado por Fukasawa et al. em [19]. Este é um algoritmo *branch-and-cut-and-price* que consegue resolver instâncias de até 135 clientes consistentemente. Neste trabalho 18 instâncias da literatura foram resolvidas de forma exata pela primeira vez.

Capítulo 3

Problema de Roteamento de Veículos Capacitados com Restrições Bidimensionais de Carregamento

3.1 Introdução

O problema de roteamento de veículos capacitados com restrições bidimensionais de carregamento (*Two-dimensional Capacitated Vehicle Routing Problem with Loading Constraints* – 2L-CVRP) é um caso mais geral do CVRP, onde as demandas dos clientes são compostas de itens retangulares.

Dado um depósito, um conjunto de clientes e um conjunto de veículos, objetiva-se suprir as demandas de todos os clientes minimizando o custo total de deslocamento. Cada cliente demanda um conjunto de itens retangulares, cada item possuindo um peso associado. Para entregar os itens requeridos pelos clientes temos um conjunto de veículos idênticos. Cada veículo possui um contêiner bidimensional e uma capacidade de peso máxima. Os veículos devem iniciar e finalizar seu percurso no depósito, ou seja, eles devem sair do depósito com os itens, visitar os clientes e voltar para o depósito. Algumas restrições devem ser consideradas para que o veículo possa visitar um conjunto de clientes: quando colocando os itens a serem entregues dentro do contêiner, estes não podem se sobrepor, ultrapassar os limites do contêiner e o seu peso total não pode superar a capacidade do veículo. Observe-se também que cada cliente deve ser visitado por exatamente um veículo, ou seja, não são permitidas entregas repartidas. Definimos formalmente o 2L-CVRP na seção 3.2.

O 2L-CVRP é fortemente NP-difícil [22], já que consiste em resolver o Problema de Roteamento de Veículos Capacitados e um problema de carregamento análogo a vários problemas de empacotamento conhecidos, particularmente o Problema de Empacotamento em Contêineres Bidimensional e o Problema de Empacotamento em Faixa. Mais especificamente, dados os itens de um conjunto de clientes e um contêiner de um veículo, podemos saber a viabilidade do carregamento destes itens no contêiner resolvendo a versão de decisão do Problema

de Empacotamento em Contêineres Bidimensional. Caso não seja possível empacotar tudo em exatamente um contêiner, o carregamento é inviável. O mesmo é válido para a versão de decisão do Problema de Empacotamento em Faixa. O carregamento é inviável caso não seja possível empacotar todos os itens em uma faixa de comprimento igual ou menor ao do contêiner. Além da versão de decisão, a versão de otimização do Problema de Empacotamento em Contêineres Bidimensional também possui uma relação direta com o 2L-CVRP. Dados todos os itens de uma instância do 2L-CVRP e as dimensões do contêiner do veículo, a versão de otimização do Problema de Empacotamento em Contêineres Bidimensional nos dá o número mínimo de veículos necessário para servir todos os clientes.

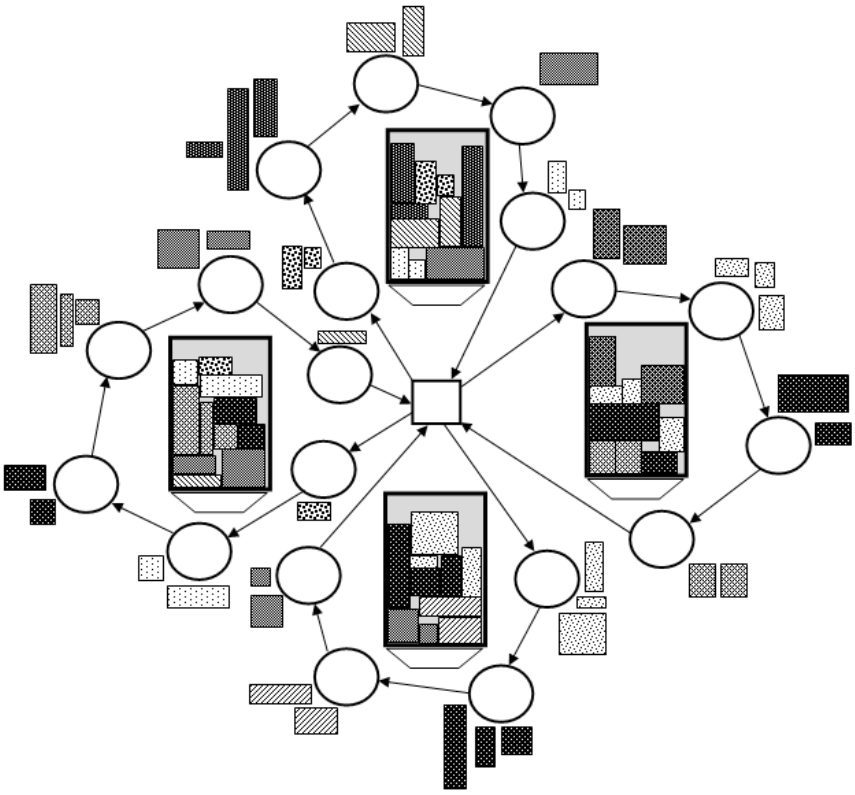

Figura 3.1: Exemplo de uma instância resolvida do 2L-CVRP.

Duas variantes do 2L-CVRP podem ser encontradas na literatura, o caso irrestrito e o caso sequencial. O 2L-CVRP irrestrito permite o re-arranjo de itens durante a fase de descarregamento, deste modo, não é necessária a organização dos itens em nenhum padrão especial, apenas o empacotamento destes no contêiner do veículo. No caso sequencial o padrão de empacotamento deve obedecer uma certa ordenação dos itens, ou seja, quando no ato de descarregamento dos itens de um cliente, nenhum destes podem estar bloqueados por itens que pertençam a outros clientes. Na prática, este caso ocorre devido a características dos itens. Podem existir itens muito frágeis ou muito pesados, de modo que efetuar a re-organização dos itens durante o descarregamento se torna difícil ou proibitivamente demorado. A figura 3.2 mostra o empacotamento de um mesma rota no caso sequencial e irrestrito. Nesta rota o veículo deve visitar

primeiro o cliente 3, seguido do 7 e por último o cliente 9. Considere que os itens são descarregados pelo lado superior do contêiner. Podemos ver que na versão irrestrita, exemplificada na rota à direita na figura, não existe nenhuma ordenação dentro do contêiner. Se não fosse permitido re-organizar os itens no ato de descarregamento, ambos os itens do cliente 7 bloqueariam a saida do item do cliente 9. Já no caso sequencial, exemplificado pela rota à esquerda na figura, existe um padrão no empacotamento. Os itens do cliente 7 não bloqueiam nenhum dos itens dos clientes 3 e 9 e o item do cliente 9 não bloqueia nenhum item do cliente 3.

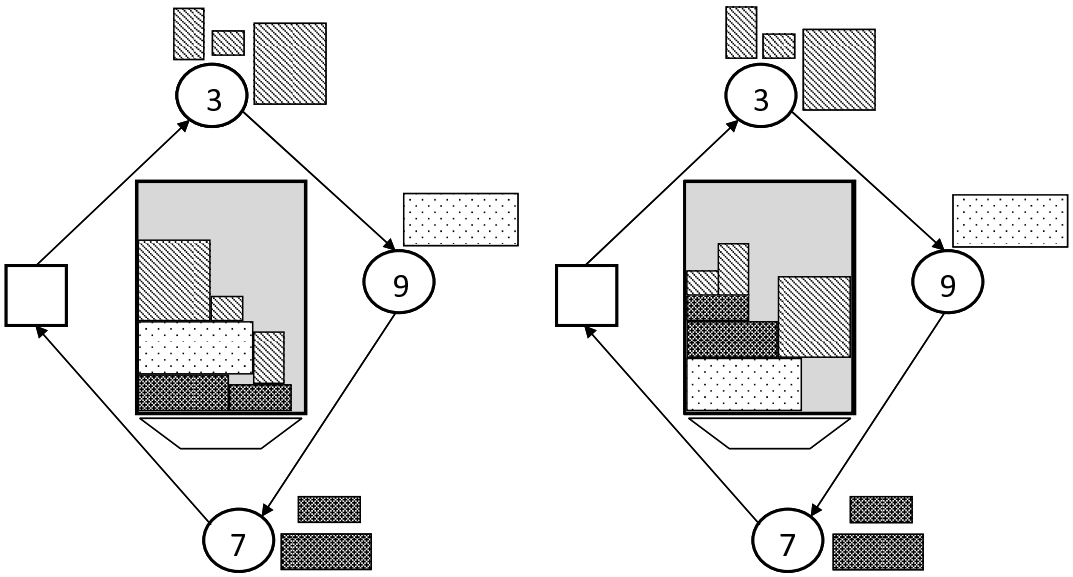

Figura 3.2: Carregamento Sequencial e Irrestrito, respectivamente

Pode-se considerar também mais duas variantes, permitindo rotações dos itens para ambos os casos sequencial e irrestrito. Observe que nesta dissertação consideramos que os itens possuem orientação fixa.

Este capítulo encontra-se organizado do seguinte modo: na seção 3.2 definimos formalmente o 2L-CVRP, na seção 3.3 apresentamos uma formulação linear inteira para o problema e na seção 3.4 apresentamos uma revisão bibliográfica.

3.2 Definição do Problema

Formalmente o 2L-CVRP pode ser descrito do seguinte modo: Dado um grafo completo não dirigido $G = (V, E)$, V é um conjunto de $n + 1$ vértices correspondendo ao depósito (vértice 0) e n clientes (vértices $1, \ldots, n$). E representa um conjunto de arestas, onde para cada aresta $e \in E$ existe um custo associado $c_e \in \mathbb{Q}^+$. Existe uma frota homogênea de K veículos, cada um com uma capacidade de carga D e um contêiner bidimensional de largura W e comprimento L para carregar os itens dos clientes. Seja $V_+ = V \setminus \{0\}$, cada cliente $i \in V_+$ demanda um conjunto de itens de peso total d_i. Seja B o conjunto completo de itens onde cada $b \in B$ é uma tupla $b = (w_b, l_b)$, sendo w_b e l_b a largura e comprimento do item b. Seja $B_i \subseteq B$ o

conjunto de itens do cliente i. O descarregamento do veículo é efetuado de modo ortogonal ao comprimento do contêiner e os pontos $(0, 0)$ e $(W, 0)$ representam o canto inferior esquerdo e direito do contêiner, respectivamente. Estes pontos encontram-se no lado oposto ao lado em que o descarregamento é feito. Os pontos $(0, L)$ e (W, L) representam os cantos superior esquerdo e direito do contêiner, sendo o lado por onde a operação de descarregamento é feita. A figura 3.2 mostra um exemplo desta operação.

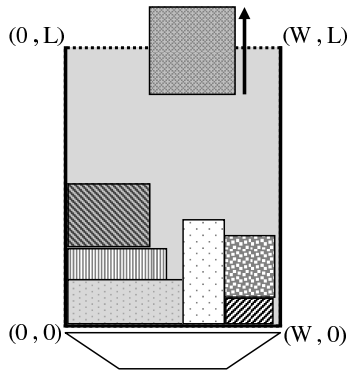

Figura 3.3: Exemplo da operação de descarregamento.

Um empacotamento $\mathcal{P}$ de uma lista de itens $T \subseteq B$ em um contêiner (W, L) é uma função $\mathcal{P} : T \to [0, W) \times [0, L)$ tal que:

(i) Os itens em T devem ter orientação fixa, ou seja, não podem ser rotacionados;

(ii) O empacotamento deve ser ortogonal, ou seja, as arestas dos itens devem estar paralelas ou perpendiculares às bordas do contêiner;

(iii) Os itens em T devem ser empacotados dentro dos limites do contêiner. Ou seja, dadas as coordenadas $\mathcal{P}(b) = (\mathcal{P}^w(b), \mathcal{P}^l(b))$, para cada $b \in T$, então

$$\mathcal{P}^w(b) + w_b \leq W \text{ e } \mathcal{P}^l(b) + l_b \leq L; \tag{3.1}$$

(iv) Os itens não podem se sobrepor. Ou seja, se $\mathcal{R}(b)$ é definido como $\mathcal{R}(b) = [\mathcal{P}^w(b), \mathcal{P}^w(b) + w_b) \times [\mathcal{P}^l(b), \mathcal{P}^l(b) + l_b)$ então

$$\mathcal{R}(b') \cap \mathcal{R}(b'') = \emptyset \text{ para todo } b', b'' \in T. \tag{3.2}$$

Uma rota viável C é um ciclo em G que contém o depósito e satisfaz as seguintes condições:

(v) O peso de todos os itens dos clientes em C não pode exceder a capacidade de carga do veículo. Ou seja, se V_C é o conjunto de clientes em C então $\sum_{i \in V_C} d_i \leq D$;

(vi) Existe um empacotamento $\mathcal{P}_C$ dos itens dos clientes em C;

(vii) Todos os itens de um dado cliente devem ser carregados no mesmo veículo. Ou seja, um cliente deve ser servido por uma única rota.

O 2L-CVRP consiste em encontrar um conjunto de rotas viáveis $\mathcal{C} = \{C_1, \ldots, C_K\}$, tal que minimize-se o custo de roteamento $c(\mathcal{C}) = \sum_{i=1}^{K} \sum_{e \in C_i} c_e$.

A operação de descarregamento dos itens de um cliente é feita através de um único lado do contêiner. Deste modo, a existência de itens muito pesados, grandes ou frágeis pode tornar a re-organização destes extremamente difícil ou proibitivamente demorado. Assim, pode ser desejável ter a seguinte condição adicional:

(viii) Itens que pertençam a um cliente que esteja descarregando não podem estar bloqueados por itens de outros clientes. Mais precisamente, dada uma rota C com a sequência de vértices $(0, i_1, \ldots, i_t, 0)$, dizemos que um empacotamento $\mathcal{P}_C$ de $T = B_{i_1} \cup \ldots \cup B_{i_t}$ respeita a sequência de C na direção l se satisfaz:

$$\mathcal{R}^l(b') \cap \mathcal{R}^l(b'') = \emptyset \quad \forall b' \in B_{i_r} \text{ e } \forall b'' \in B_{i_s} \quad 1 \leq r < s \leq t, \tag{3.3}$$

onde $\mathcal{R}^l(b) = [\mathcal{P}^w(b), \mathcal{P}^w(b) + w_b) \times [\mathcal{P}^l(b), L)$.

O 2L-CVRP Sequencial é um caso especial do 2L-CVRP onde todas as rotas devem satisfazer a condição (viii).

3.3 Formulação em Programação Linear Inteira

Nesta seção apresentaremos uma formulação em programação linear inteira para o 2L-CVRP. Esta formulação é baseada na formulação conhecida como *two-index* para o CVRP.

Seja x_e uma variável de fluxo igual a um se um veículo viaja usando a aresta $e \in E$ e zero caso contrário. Dado um subconjunto de clientes $S \subseteq V_+$, $d(S)$ é o peso total dos itens dos clientes em S, ou seja, $d(S) = \sum_{i \in S} d_i$. Seja B_i o conjunto de itens do cliente i, D a capacidade de carga de cada veículo e A a área do contêiner. Seja $a(S)$ a área total dos itens dos clientes S, ou seja, $a(S) = \sum_{i \in S} \sum_{b \in B_i} w_b l_b$. Seja $\delta(S)$ o conjunto de arestas em G que incidem em exatamente um vértice em S, ou seja, $\delta(S) = \{\{i, j\} \in E : i \in S, j \notin S\}$. Seja K a quantidade de veículos disponíveis para servir os clientes. Seja $r(S)$ o número mínimo de veículos necessário para suprir as demandas em S. Temos então a seguinte formulação em

programação inteira para o 2L-CVRP:

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{e \in E} c_e x_e & & \\
\text{s. a.} \quad & & & \\
& \sum_{e \in \delta(\{i\})} x_e = 2 & \forall i \in V_+ \qquad & (1) \\
& \sum_{e \in \delta(\{0\})} x_e = 2K & & (2) \\
& \sum_{e \in \delta(S)} x_e \geq 2r(S) & \forall S \subseteq V_+, |S| \geq 2 \qquad & (3) \\
& x_e \in \{0,1\} & \forall e \in E. \qquad & (4)
\end{aligned}
\tag{3.4}
$$

As restrições (1), as restrições de grau, impõem que cada cliente é visitado exatamente uma vez. As restrições (2) garantem que são utilizados exatamente K veículos para suprir as demandas dos clientes. As restrições (3), as Desigualdades de Capacidade e Conectividade, garantem a conectividade da solução e as restrições de peso e de empacotamento bidimensional. Para este modelo utilizamos as Desigualdades de Capacidade Fracas (ver capítulo 4). A tarefa de computar estas desigualdades mostra-se inviável computacionalmente, e resolver o problema de separação correspondente é NP-difícil (ver seção 4.2). Finalmente, as restrições (4) definem a característica binária das variáveis.

3.4 Revisão Bibliográfica

O 2L-CVRP foi inicialmente estudado em 2007 por Iori, Salazar-González e Vigo [22]. Este trabalho apresentou as primeiras instâncias para o 2L-CVRP, estendendo instâncias do CVRP e propôs um algoritmo exato usando uma abordagem *branch-and-cut*. Instâncias de até 35 clientes e mais de 100 itens foram resolvidas. Alguns padrões que seriam seguidos por trabalhos posteriores foram definidos nesta proposta, como a restrição sequencial e o descarregamento por um único lado do veículo.

No ano seguinte, Gendreau, Iori, Laporte e Martello [28] apresentaram a primeira abordagem heurística para o 2L-CVRP através de um algoritmo baseado em Busca Tabu. Além de utilizar as instâncias apresentadas em [22], outras instâncias foram criadas. O conjunto atual de 360 instâncias do 2L-CVRP na literatura consiste das instâncias criadas nestes dois primeiros trabalhos. Esta abordagem apresentou um comparativo com o trabalho de Iori, Salazar-González e Vigo [22] e conseguiu encontrar soluções viáveis para todas as instâncias. A variante irrestrita foi introduzida neste trabalho.

Em 2009, mais duas abordagens heurísticas foram propostas. Fuellerer, Doernera, Hartla e Iori apresentaram um algoritmo de Otimização por Colônia de Formigas em [26]. Este trabalho superou os resultados da heurística apresentada em [28] e apresentou uma variação das versões sequencial e irrestrita, permitindo rotação dos itens. O segundo trabalho abordando o 2L-CVRP

neste ano foi apresentado por Zachariadis, Tarantilis e Kiranoudis [7]. Neste, propuseram uma abordagem heurística com um algoritmo baseado em Busca Tabu Guiada. Foram consideradas ambas as variantes sequencial e irrestrita, mas não foi permitida rotação dos itens. Este trabalho apresentou resultados competitivos, conseguindo superar a abordagem de Gendreau et al. [28], mas não conseguiram melhorar os resultados obtidos por Fuellerer, Doernera, Hartla e Iori [26].

Capítulo 4

Desigualdades Válidas e Rotinas de Separação para o 2L-CVRP

4.1 Introdução

Neste capítulo apresentaremos as desigualdades utilizadas em nossa abordagem *branch-and-cut*. Até este momento não existem estudos poliedrais para o 2L-CVRP na literatura. Efetuamos então pequenas adaptações em desigualdades originalmente criadas para o CVRP para aplicá-las ao 2L-CVRP.

O politopo do 2L-CVRP possui uma descrição linear dado um número finito de desigualdades lineares, mas infelizmente não sabemos a sua descrição completa. Como o 2L-CVRP é uma generalização do CVRP, sabemos que as desigualdades válidas para o CVRP também mostram-se válidas para o 2L-CVRP e que algumas observações sobre o CVRP podem ser estendidas para o problema aqui estudado.

Naddef e Rinaldi [18] observaram que o politopo do CVRP não possui dimensão cheia, e esta não é determinada simplesmente por uma função do número de vértices no problema. A dimensão do politopo depende de uma complexa combinação de todos os termos que compõem o problema: o número de veículos, o número de clientes, a capacidade de carga dos veículos e o vetor de demandas dos clientes. Pode-se generalizar estas observações para o 2L-CVRP, enfatizando o aumento significativo de complexidade no problema, já que os clientes demandam itens bidimensionais além das demandas unidimensionais originais do CVRP.

A abordagem ideal para verificarmos o quanto forte é uma desigualdade consiste em provar que esta define uma faceta do politopo do problema estudado. Mas como o politopo do CVRP não possui dimensão cheia, esta é uma tarefa particularmente difícil. Para reduzir esta dificuldade, Naddef et al. em [18] sugerem relaxar o problema e provar que a desigualdade define faceta para o politopo da relaxação. Se verdadeiro, esta desigualdade seria então considerada forte. Entretanto, uma desigualdade considerada forte por esta definição não necessariamente define faceta para o politopo do problema não relaxado. Alternativamente, podemos utilizar desigualdades válidas e analisar resultados obtidos de forma empírica, através de testes com-

putacionais. Infelizmente esta abordagem pode gerar resultados variados e levar a conclusões erradas, já que algumas desigualdades podem se mostrar extremamente úteis para certas instâncias e inúteis para outras.

Por apenas considerar as demandas unidimensionais, as desigualdades válidas para o CVRP tendem a definir faces de dimensão baixa do politopo do 2L-CVRP, mas adaptações podem ser feitas levando-se em conta a característisca bidimensional de seu problema de empacotamento. Nas seções deste capítulo descreveremos as desigualdades utilizadas em nosso algoritmo *branch-and-cut* e os algoritmos de separação utilizados. Utilizamos o código disponibilizado em [51] na obtenção dos cortes.

Na seção 4.2, apresentamos a hierarquia de Desigualdades de Capacidade e Conectividade. As Desigualdades de Capacidade Construídas são detalhadas na seção 4.3. Nas seções 4.4 e 4.5 detalhamos as Desigualdades Multi-Estrela Homogêneas e Parciais e as Desigualdades de Pente Fortalecidas. Na seção 4.6, a última do capítulo, descrevemos as Desigualdades Hipo-Ciclo Estendidas de 2-arestas. As duas primeiras desigualdades foram utilizadas para garantir a conectividade e as restrições de peso do problema. As três desigualdades restantes foram adicionadas na tentativa de reduzirmos a região viável do politopo durante a execução do algoritmo *branch-and-cut* proposto.

4.2 Desigualdades de Capacidade e Conectividade

Nesta seção focaremos no problema de separar um ponto arbitrário fracionário do politopo do 2L-CVRP usando as Desigualdades de Capacidade e Conectividade. Estas apresentam-se como uma ligação entre a estrutura de empacotamento e roteamento, sendo particularmente importantes para descrever a estrutura poliedral do politopo do 2L-CVRP. Existe uma hierarquia de Desigualdades de Capacidade e Conectividade [18], quase todas na forma

$$\sum_{e \in \delta(S)} x_e \geq \beta, \tag{4.1}$$

possuindo o mesmo lado esquerdo e variando o lado direito (representado por β). Quanto maior β, mais forte é a desigualdade. Consequentemente, o problema de separação correspondente também torna-se mais difícil. A figura 4.1 ilustra a hierarquia das Desigualdades de Capacidade e Conectividade.

As desigualdades aqui apresentadas foram originalmente propostas para o CVRP, mas foram adaptadas para o 2L-CVRP.

Iniciaremos com as Desigualdades de Capacidade, que possuem valor $2R(S)$ como seu lado direito e encontram-se em terceiro na hierarquia em termos de força. Seja $\mathcal{P}$ o conjunto de todas as l-partições viáveis de V_+. Para toda l-partição $P = \{S_1, \ldots, S_l\}$ e para todo subconjunto não vazio $S \subseteq V_+$, a função $R : 2^{V_+} \rightarrow Z_+$ é definida por:

$$R(S) = \min_{P \in \mathcal{P}} \sigma(P, S), \tag{4.2}$$

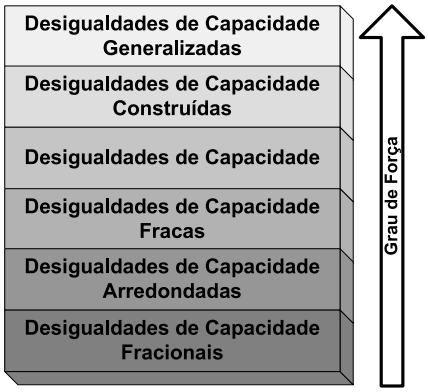

Figura 4.1: Hierarquia das Desigualdades de Capacidade e Conectividade

com σ definida por:

$$\sigma(P, S) = |\{i : S_i \cap S \neq \emptyset\}|. \tag{4.3}$$

Ou seja, se P é uma partição de clientes em veículos, a função σ devolve o número de veículos necessário para suprir as demandas de itens (com pesos associados) de todos os clientes em S. A função R devolve o número mínimo de veículos necessário para satisfazer as demandas de itens em S em uma l-partição viável.

Para o CVRP, as Desigualdades de Capacidade não necessariamente definem faceta, como demonstrado por Cornejuéls e Harche em [14]. O problema de separação das Desigualdades de Capacidade do CVRP é NP-difícil e consequentemente a separação para o 2L-CVRP também é NP-difícil, sendo um caso mais geral.

As Desigualdades de Capacidade Generalizadas são as desigualdades mais fortes da hierarquia, possuindo valor $2R(\mathcal{S})$ como lado direito e uma ligeira modificação no lado esquerdo. Para encontrarmos as Desigualdades de Capacidade Generalizadas precisamos compreender qual a deficiência apresentada pelas Desigualdades de Capacidade. Considere então um conjunto $\mathcal{S} = \{S_1, \ldots, S_t\}$ de $t > 1$ conjuntos disjuntos de V_+. Observe que pode não existir uma l-partição não viável justa à S_1 (ambos lados da desigualdade iguais), que também seja justa para todos os $t - 1$ subconjuntos restantes. Para fortalecer a desigualdade, deve-se então aumentar o máximo possível o seu lado direito. Deste modo, o maior valor possível para o lado direito será duas vezes $R(\mathcal{S})$, com a função $R(\mathcal{S})$ definida por:

$$R(\mathcal{S}) = min \left\{ \sum_{i=1}^{t} \sigma(P, S_i) \ : \ P \in \mathcal{P} \right\}. \tag{4.4}$$

As Desigualdades de Capacidade Generalizadas são definidas por:

$$\sum_{i=1}^{t} \sum_{e \in \delta(S_i)} x_e \geq 2R(\mathcal{S}) \tag{4.5}$$

DeVitis et al. [15] demonstraram que as Desigualdades de Capacidade Generalizadas definem faceta para o politopo do CVRP. Como as Desigualdades de Capacidade Generalizadas são

uma generalização das Desigualdades de Capacidade, o problema de separação correspondente é NP-difícil.

As Desigualdades de Capacidade Construídas encontram-se em segundo na hierarquia. Estas possuem uma forma diferente que todas as outras Desigualdades de Capacidade e Conectividade. Considere um conjunto $S \subseteq V_+$ e seja $P = \{S_1, \ldots, S_l\}$ uma l-partição de S. Seja $\tau(S, P)$, o número mínimo de veículos necessário para servir todos os vértices em S dado que a Desigualdade de Capacidade para cada S_i é justa. A Desigualdade de Capacidade Construída é definida por:

$$\sum_{e \in \delta(S)} x_e + \sum_{i=1}^{l} \sum_{e \in \delta(S_i)} x_e \geq 2\tau(S, P) + 2 \sum_{i=1}^{l} r(S_i). \tag{4.6}$$

O problema de separação das Desigualdades de Capacidade Construídas é NP-difícil (ver [18]).

As Desigualdades de Capacidade Fracas possuem valor $2r(S)$ como seu lado direito, onde $r(S)$ possui o significado já apresentado na descrição das Desigualdades de Capacidade Generalizadas: $r(S)$ é o número mínimo de contêineres necessário para suprir as demandas de todos os clientes em S. Para o 2L-CVRP, encontrar $r(S)$ é claramente NP-difícil, já que equivale a encontrar a solução ótima do Problema de Empacotamento em Contêineres Bidimensional dado o contêiner do veículo e os itens dos clientes. Cornuéjols e Harche [14] demonstraram que, para o caso unidimensional (CVRP), o problema de separação das Desigualdades de Capacidade Fracas é NP-difícil.

As Desigualdades de Capacidade Arredondadas estão em quarto na hierarquia e possuem valor $2k(S)$ como lado direito. Seja D a capacidade de peso do veículo e $d(S)$ o peso total de todos os itens dos clientes em S. A função k é definida por:

$$k(S) = \left\lceil \frac{d(S)}{D} \right\rceil. \tag{4.7}$$

Para fortalecer esta desigualdade para o 2L-CVRP podemos fazer $k(S)$ considerar a característica bidimensional de seu problema de empacotamento. Seja então A a área do contêiner e $a(S)$ área total de todos os itens dos clientes em S. A função k' é definida por:

$$k'(S) = max \left\{ \left\lceil \frac{d(S)}{D} \right\rceil, \left\lceil \frac{a(S)}{A} \right\rceil \right\}. \tag{4.8}$$

O problema de separação das Desigualdades de Capacidade Arredondadas do CVRP foi demonstrado ser NP-difícil por Cornuéjols e Harche em [14], quando utilizando $k(S)$. Logicamente isto mantém-se com a nossa adaptação.

Finalmente, temos as Desigualdades de Capacidade Fracionais, as mais fracas da hierarquia. Estas possuem

$$l(S) = \frac{d(S)}{D} \tag{4.9}$$

como lado direito. A separação destas desigualdades pode ser feita em tempo polinomial (ver [24], [18] e [30]). Como no caso das Desigualdades de Capacidade Arredondadas, podemos

efetuar uma simples modificação para fortalecer a desigualdade para o 2L-CVRP. $l'(S)$ é então definido por:

$$l'(S) = max \left\{ \frac{d(S)}{D}, \frac{a(S)}{A} \right\}. \tag{4.10}$$

Claramente a resolução do problema de separação continua sendo feita em tempo polinomial.

Para escolher qual das Desigualdades de Capacidade e Conectividade usar em uma formulação linear inteira para o 2L-CVRP, a hierarquia completa deve ser considerada. O difícil problema de separação das Desigualdades de Capacidade Fracas as tornam proibitivas em uma abordagem exata. Como $r(S)$ é um caso especial de $R(S)$ e de $R(\mathcal{S})$, podemos concluir o mesmo para as Desigualdades de Capacidade e as Desigualdades de Capacidade Generalizadas. As Desigualdades de Capacidade Fracionais podem ser separadas em tempo polinomial, mas são relativamente fracas, devido ao fato de seu lado esquerdo sempre possuir valor inteiro. Deste modo, uma combinação entre as Desigualdades de Capacidade Arredondadas, que apesar de possuírem um problema de separação NP-difícil, e as Desigualdades de Capacidade Fracionais apresenta-se como uma escolha interessante. Utilizamos esta combinação em nossa abordagem.

Como o problema de separação das Desigualdades de Capacidade e Conectividade é difícil, quatro algoritmos são utilizados para este propósito na busca de uma solução heurística. Estes algoritmos foram apresentados por Lysgaard et al. em [3]. Modificamos minimamente estes algoritmos, utilizando $k'(S)$ e $l'(S)$ em vez de $k(S)$ e $l(S)$. Dada uma solução x^* da relaxação do 2L-CVRP obtida com a remoção das Desigualdades de Capacidade e das restrições de integralidade, a seguinte notação será usada: seja o grafo de suporte $G^* = (V, E^*)$ onde E^* corresponde ao conjunto de arestas não nulas em x^*. Removendo-se o vértice origem (depósito) de G^*, temos $G^*_+ = (V_+, E^*_+)$. Um grafo reduzido obtido de G^*_+ será denominado $Gr^*_+ = (Vr^*_+, E^*_+)$.

Descreveremos as rotinas que compõe o algoritmo de separação das Desigualdades de Conectividade e Capacidade: a primeira rotina encontra as componentes conexas $S_1, \ldots, S_n$ de G^*_+ e, para cada componente, checa se temos alguma Desigualdade de Capacidade Arredondada violada. Em seguida verifica se há violações das Desigualdades de Capacidade Arredondadas para o conjunto de vértices formado por $V_+ \backslash S_i$ e para a união das componentes não conectadas ao depósito. Cortes são gerados para cada violação encontrada. Esta heurística sempre encontra Desigualdades de Capacidade Arredondadas violadas, se existirem, se a solução x^* for inteira. Para compreender isto, existem duas situações a serem consideradas. Suponha que, dada uma solução inteira, esta solução não é conexa. Note que uma solução contendo esta configuração pode ser viável para todas as desigualdades da relaxação. Nesta situação, a violação é encontrada com o uso da primeira rotina. A segunda situação leva em conta a capacidade de carga do veículo. Suponha uma solução inteira, onde a primeira situação não ocorre, mas que possua uma rota onde a soma total dos pesos dos itens é maior que a capacidade máxima de um veículo. Novamente a violação é encontrada com o uso da primeira rotina, já que a função k nos devolverá a necessidade de mais de um veículo para suprir a demanda destes clientes, mas teremos apenas duas arestas incidentes (é uma rota).

Procurando reduzir o tempo computacional, as próximas três rotinas utilizam um grafo re-

duzido Gr^*_+ como entrada. A rotina de redução funciona do seguinte modo: dada uma solução x^* e uma aresta $e = \{i, j\}$ não adjacente ao depósito tal que $x_e = 1$, efetuamos uma contração, formando um supervértice composto dos vértices i e j. A demanda deste supervértice consiste da soma dos componentes de i e j, ou seja, o conjunto de items e pesos associados de ambos. Tal rotina é repetidamente aplicada até que não haja mais arestas que possam ser contraídas. Note que a junção de um vértice com um supervértice ou de dois supervértices forma um novo supervértice composto de todos os vértices e componentes dos elementos da contração. Esta contração é segura (ver [24]), ou seja, se existe uma Desigualdade de Capacidade violada em G^*_+ então existe um conjunto de supervértices em Gr^*_+ onde a união destes define a Desigualdade de Capacidade com a mesma violação. A contração também é efetuada para conjuntos de cardinalidade 2 ou 3, conjuntos S com $k'(S) = 1$ e conjuntos S que possuam exatamente duas arestas incidentes, ou seja $\sum_{e\in\delta(S)} x_e = 2$. É seguro contrair obedecendo estas condições, como demonstrado em [3].

As rotinas seguintes são chamadas apenas se a primeira não encontrar nenhum corte. A segunda rotina efetua a separação das Desigualdades de Capacidade Fracionais. Como as Desigualdades de Capacidade Fracionais são desigualdades mais fracas que as Desigualdades de Capacidade Arredondadas, se as Desigualdades de Capacidade Fracionais estiverem violadas, consequentemente as Desigualdades de Capacidade Arredondadas também estarão.

As Desigualdades de Capacidade Fracionais podem ser separadas em tempo polinomial (ver [18] e [24]) resolvendo um problema de fluxo máximo. Entretanto, a abordagem aqui apresentada procura encontrar múltiplos cortes. Sejam T_{in} e T_{out} dois subconjuntos disjuntos de Vr^*_+, onde $T_{in} \subset S$ e $T_{out} \cap S = \emptyset$. Para cada vértice v_i de Vr^*_+, um problema de fluxo máximo é definido inicializando $T_{in} = v_i$ e T_{out} é determinado de modo que todos os cortes gerados previamente sejam proibidos. Ou seja, a cada iteração, T_{out} recebe todos os v_i utilizados até o momento que não resultaram na geração de um corte. O conjunto Vr^*_+ é iterado três vezes, e em cada iteração os cortes das iterações anteriores também são proibidos.

A terceira rotina itera entre os vértices de Gr^*_+ para construir S. Dado um conjunto S inicial vazio, a cada iteração a rotina adiciona um vértice que minimize a folga da Desigualdade de Capacidade Arredondada nesta iteração. Ou seja, um vértice que minimize a folga em $\sum_{e\in\delta(S)} x_e = 2k'(S)$ entre todos os restantes. Esta rotina executa $|Vr^*_+|$ vezes, cada vez escolhendo um vértice inicial diferente. Para evitar a criação de conjuntos iguais, um vértice i escolhido na iteração é adicionado apenas se não gerar um conjunto $S \cup \{i\}$ criado previamente. Caso este não seja escolhido, testa-se o próximo vértice em $Vr^*_+ \setminus S$ que minimiza a folga da Desigualdade de Capacidade Arredondada.

A quarta e última rotina utiliza-se dos cortes encontrados pelas rotinas anteriores para gerar novas desigualdades. Para cada Desigualdade de Capacidade Arredondada encontrada, a rotina substitui o conjunto S associado pelo menor conjunto de vértices em Vr^*_+ que contém S. Os vértices são então ordenados em ordem não decrescente de peso total dos itens e cada vértice é considerado para remoção, caso isto diminua a folga da Desigualdade de Capacidade Arredondada. Em seguida, os vértices restantes são adicionados, removidos ou substituidos iterativamente de modo a diminuir a folga.

4.3 Desigualdades de Capacidade Construídas

Nesta seção abordaremos as Desigualdades de Capacidade Construídas que são uma modificação das Desigualdades de Capacidade Generalizadas. As Desigualdades de Capacidade Construídas foram propostas por Augerat [2] e Pochet [53] e são Desigualdades de Capacidade e Conectividade, encontrando-se entre as Desigualdades de Capacidade Generalizadas e as Desigualdades de Capacidade na hierarquia, mas possuem uma seção própria por terem um papel importante na abordagem utilizada.

Considere um conjunto $S \subseteq V_+$ e seja $P = \{S_1, \ldots, S_l\}$ uma l-partição de S. Seja $\tau(S, P)$, o número mínimo de veículos necessário para suprir a demanda de todos os vértices em S dado que a Desigualdade de Capacidade para cada S_i é justa (ver a seção 4.2). Seja $r(S_i)$ o número mínimo de veículos necessário para servir todos os clientes em S_i. A Desigualdade de Capacidade Construída é definida por:

$$\sum_{e \in \delta(S)} x_e + \sum_{i=1}^{l} \sum_{e \in \delta(S_i)} x_e \geq 2\tau(S, P) + 2 \sum_{i=1}^{l} r(S_i). \tag{4.11}$$

Note que se S for substituido por todos os clientes, V_+, temos as Desigualdades de Capacidade Generalizadas.

Para efetuar a separação das Desigualdades de Capacidade Construídas utilizamos o algoritmo apresentado por Lysgaard, Letchford e Eglese em [3]. Nossa adaptação na rotina original de Lysgaard et al. para o 2L-CVRP consiste em utilizar $k'(S)$ em vez de $k(S)$.

Devido a dificuldade de se calcular $\tau(S, P)$, este é substituido por um limitante inferior obtido através da resolução do Problema de Empacotamento de Contêineres. Do mesmo modo, para evitar o cálculo de r para todo S_i, $r(S_i)$ é substituido pelo limitante $k'(S_i)$ (ver a descrição das Desigualdades de Capacidade Arredondadas na seção 4.2).

Uma busca em profundidade efetua a separação das Desigualdades de Capacidade Construídas do seguinte modo: a partir do nó l com a partição P^l, gera-se um nó filho para cada uma das s arestas do grafo reduzido no nó l. Estes nós filhos representam as partições $P^l_1, \ldots, P^l_s$, cada i-ésima partição sendo obtida comprimindo a i-ésima aresta de maior peso em P^l e proibindo a compressão das outras $i-1$ arestas de peso maior. Arestas recém formadas de peso um também são comprimidas.

Durante a busca, o Problema de Empacotamento em Contêineres Unidimensional é resolvido de forma exata para cada partição e, se uma violação é encontrada, este corte é adicionado. *Backtracking* é efetuado quando a violação ocorre, se apresenta uma folga maior ou igual a dois ou se o procedimento tenta a redução de uma aresta proibida.

4.4 Desigualdades Multi-Estrela Homogêneas e Parciais

Nesta seção descreveremos as Desigualdades Multi-Estrela Homogêneas e as Desigualdades Multi-Estrela Homogêneas Parciais e os algoritmos de separação correspondentes. As Desi-

gualdes Multi-Estrela Homogêneas e Parciais foram originalmente definidas por Araque, Hall e Magnanti em [27] para o Problema de Roteamento de Veículos com Demandas Unitárias.

Para separarmos as Desigualdades Multi-Estrela Homogêneas e as Desigualdades Multi-Estrela Homogêneas Parciais utilizamos os algoritmos apresentados por Letchford et al. em [39].

Dados dois conjuntos disjuntos de vértices R e S, denotamos por $E(R : S)$ o conjunto de arestas com um vértice em R e outro em S e por $E(R)$ o conjunto de arestas que incidem apenas em vértices de R. Seja um conjunto de arestas T, $x(T)$ denota $\sum_{e \in T} x_e$. O conjunto de clientes é denotado por V_+, ou seja, $V_+ = V \backslash \{0\}$. Denotaremos como núcleo, o conjunto $N \subset V_+$. Os vértices do conjunto $S \subseteq (V_+ \backslash N)$ serão denotados de satélites. Os conectores são representados por $C \subset N$. As Desigualdades Multi-Estrela Homogêneas são definidas por:

$$\alpha \sum_{e \in E(N)} x_e + \beta \sum_{e \in E(N:S)} x_e \leq \gamma \tag{4.12}$$

e as Desigualdades Multi-Estrela Homogêneas Parciais por:

$$\alpha \sum_{e \in E(N)} x_e + \beta \sum_{e \in E(C:S)} x_e \leq \gamma. \tag{4.13}$$

Onde α, β e γ são constantes que dependem de $|N|$ e $|S|$ (ver [39]).

Seja x^* a solução para qual desejamos verificar se existem desigualdades violadas. Para encontrarmos N usamos o Algoritmo 1. Encontrado o núcleo N, para cada conjunto $N_c \in N$ geramos os satélites do seguinte modo: S é definido inicialmente como $\{i \in (V_+ \backslash N_c : x^*(E(\{i\} : N_c)) > 0\}$. Em seguida, removemos iterativamente, um por vez, o satélite $s \in S$ com o menor valor $x^*(E(\{i\} : N_c))$. Em posse de N e S podemos verificar se existem Desigualdades Multi-Estrela Homogêneas violadas.

Se não encontrarmos Desigualdades Multi-Estrela Homogêneas violadas, tentamos separar as Desigualdades Multi-Estrela Homogêneas Parciais. Dado o núcleo N encontrado pelo Algoritmo 1, chamamos o Algoritmo 2 para cada $N_c \in N$. Encontrados cortes, estes são adicionados ao sistema.

4.5 Desigualdades de Pente Fortalecidas

Nesta seção descreveremos as Desigualdades de Pente Fortalecidas e o algoritmo utilizado para efetuar sua separação. As Desigualdades de Pente Fortalecidas são uma adaptação das Desigualdades de Pente (ver [31] e [32]) para o Problema do Caixeiro Viajante. Como as Desigualdades de Pente mostram-se altamente eficientes [3] para o Problema do Caixeiro Viajante, diversas adaptações destas desigualdades para o CVRP podem ser encontradas na literatura [43, 1, 21, 42].

Para efetuar a separação das Desigualdades de Pente Fortalecidas utilizamos o algoritmo apresentado por Lysgaard et al. em [3]. Este algoritmo não apresenta uma nova adaptação, mas utiliza a adaptação de Araque [1].

Algoritmo 1: Algoritmo para encontrar o núcleo das Desigualdades Multi-Estrela Homogêneas

Entrada: Conjunto de todos os clientes V_+
Saída: Núcleo N
$N \leftarrow \emptyset$
$L1 \leftarrow False$
para $i \leftarrow 1$ **até** $|V_+|$ **faça**
  se $L1 = Falso$ **então**
    $N_c \leftarrow i$
  senão
    $L1 \leftarrow Falso$
  $W \leftarrow \{j \in (V_+ \backslash N_c) : x^*(E(N_c : j)) > 0, N_c \cup \{j\} \notin N\}$
  se $W \neq \emptyset$ **então**
    Escolha $j \in W$ tal que maximize $x^*(E(N_c : \{j\})) - x^*_{0j}$
    $N_c \leftarrow N_c \cup \{j\}$
  se $N_c \neq V_+$ **então**
    $N \leftarrow N \cup \{N_c\}$
    $L1 \leftarrow Verdadeiro$
Devolva N

Algoritmo 2: Algoritmo para encontrar o núcleo das Desigualdades Multi-Estrela Homogêneas Parciais

Entrada: Todo subconjunto $N_c \in N$
para $p \leftarrow min(4, |N_c| - 1)$**até**2 **faça**
  Defina C como os p vértices em N_c com os maiores valores de $x^*(E(\{i\} : (V_+ \backslash N_c)))$
  Defina S como os clientes em $V_+ \backslash N_c$ que estão x^*-conectados a C
  enquanto $|S| \geq 2$ **faça**
    Verifique se existem Desigualdades Multi-Estrela Homogêneas Parciais violadas com estes N_c, S e C
    Remova o cliente com o menor $x^*(E(\{i\} : C))$ de S

Seja um pente formado por um conjunto de vértices $H \subset V_+$ (cabo) e $t > 2$ conjuntos $T_1, \ldots, T_t$ de vértices restantes (dentes). Seja $H \cap T_j \neq \emptyset$ e $T_j \backslash H \neq \emptyset$ para $j = 1, \ldots, t$. Seja $T_i \cap T_j \cap H = \emptyset$ para todos os pares $\{i, j\} \subset \{1, \ldots, t\}$.

Para todo conjunto $S \subset V$, seja $r_c(S) = r(V \backslash S)$ se o depósito se encontra em S e $r_c(S) = r(S)$ caso contrário. A quantidade $S(H, T_1, \ldots, T_t)$ é definida como:

$$S(H, T_1, \ldots, T_t) = \sum_{j=1}^{t} (r_c(T_j \cap H) + r_c(T_j \backslash H) + r_c(T_j)). \tag{4.14}$$

Quando $S(H, T_1, \ldots, T_t)$ é ímpar temos então as Desigualdades de Pente Fortalecidas para o CVRP:

$$\sum_{e \in \delta(H)} x_e + \sum_{j=1}^{t} \sum_{e \in \delta(T_j)} x_e \geq S(H, T_1, \ldots, T_t) + 1. \tag{4.15}$$

A heurística de separação apresentada por Lysgaard et al. [3] é descrita a seguir: primeiramente, para garantir a tratabilidade computacional, a desigualdade é modificada substituindo todas as instâncias de $r(S)$ por $k'(S)$ (ver a seção 4.2). Um grafo reduzido é gerado obedecendo aos seguintes critérios: conjuntos de tamanho dois ou três e conjuntos que possuam exatamente duas arestas incidentes são comprimidos iterativamente se satisfazerem uma das duas condições: sejam dois conjuntos disjuntos de vértices S e T, $E(S : T)$ representa o conjunto de arestas que incidem simultaneamente em S e T.

$$\sum_{e \in E(S:i)} x^*_e = 1 \qquad i \in V_+ \backslash S$$

ou (4.16)

$$\sum_{e \in E(S:0)} x^*_e = 1 \qquad \textbf{e} \qquad 2k(V_+ \backslash S) = 2k(V_+) = \sum_{e \in \delta(0)} x^*_e.$$

Após a execução do procedimento de redução, as arestas são todas removidas. Em seguida, as arestas removidas no passo anterior são inseridas em ordem não decrescente de $|x^*_e - 0.5|$ e a cada passo verificando se uma componente conexa foi criada. Caso contrário a aresta não é adicionada. Para cada componente conexa, a desigualdade de 2-emparelhamento com folga mínima é encontrada. Isto gera uma desigualdade de pente tradicional. Para cada desigualdade encontrada, seja violada ou não, faz-se uma expansão de modo guloso dos vértices do conjunto associado. Caso uma desigualdade violada ainda não tenha sido encontrada, é usado o algoritmo exato de separação para desigualdades 2-emparelhamento apresentado por Padberg e Rao em [29].

4.6 Desigualdades Hipo-Ciclo Estendidas de 2-arestas

Nesta seção abordaremos as Desigualdades Hipo-Ciclo Estendidas de 2-arestas. Estas desigualdades são uma variação das Desigualdades Hipo-Ciclo para o CVRP, ver [21]. Sejam duas arestas distintas $e_1 = (u_1, v_1)$ e $e_2 = (u_2, v_2)$. Seja um dado conjunto $W \subset V_+$ onde $e_1, e_2 \in \delta(W)$

e $v_1, v_2 \notin W$. Denotaremos estes vértices v_1 e v_2 como terminais. Dado um $F \subset E$, as Desigualdades Hipo-Ciclo Estendidas de 2-arestas definem que qualquer rota viável que sirva todos os clientes em $W \cup \{v_1, v_2\}$, começando e terminando em v_1 e v_2, deve usar pelo menos uma aresta de F. Consequentemente, F deve ser um caminho hamiltoniano através de $W \cup \{v_1, v_2\}$, sendo estendido por dois caminhos disjuntos em vértices até o depósito. Além disso, para este ciclo gerado, o peso total dos itens dos clientes visitados não pode exceder a capacidade do veículo. As Desigualdades Hipo-Ciclo Estendidas de 2-arestas são definidas como:

$$\sum_{e \in \delta(W)} x_e + 2 \sum_{e \in F} x_e \geq 2x_{e_1} + 2x_{e_2}. \tag{4.17}$$

Descreveremos a rotina de separação utilizada para separar Desigualdades Hipo-Ciclo Estendidas de 2-arestas, apresentada por Lysgaard et al. em [3]: dada uma solução x^* gera-se um grafo reduzido utilizando-se da mesma rotina de redução usada na separação das Desigualdades de Capacidade. Em seguida, uma busca gulosa é aplicada para encontrar os conjuntos candidatos $S = W \cup \{v_1, v_2\}$. Detalhes desta heurística podem ser encontrados no código disponibilizado em [51]. Após obter os conjuntos iniciais, efetua-se a remoção de conjuntos utilizando-se de um critério de dominância. Sejam dois conjuntos S_1 e S_2, diz-se que S_1 domina S_2 se $\sum_{e \in \delta(S_1)} x_e \leq \sum_{e \in \delta(S_2)} x_e$ com $S_2 \subset S_1$. Para cada conjunto restante S_i, geram-se os conjuntos $W \cup \{v_1, v_2\}$ do seguinte modo: para cada S_i, encontram-se todos os pares $v_1, v_2 \in S_i$ onde $2x^*_{e_1} + 2x^*_{e_2} - \sum_{e \in \delta(S \setminus \{v_1, v_2\})} x^*_e > 0$. Note que todas as combinações são consideradas para gerar os conjuntos $W \cup \{v1, v2\}$ que serão usados na etapa seguinte. Seja G_{ind} o subgrafo induzido de G^* pelos vértices em $V \backslash W$ e E_{ind} seu conjunto de arestas. Para cada aresta $\{i, j\} \in E_{ind}$ é associado o custo $c_{ij} = (w_i + w_j)/2$, onde $w_i = 0$ para $i \in \{0, v_1, v_2\}$ e $w_i = d_i$ caso contrário.

Um elemento chave para a separação consiste em encontrar dois caminhos disjuntos em vértices, $\{0, ..., v_1\}$ e $\{0, ..., v_2\}$, de peso mínimo total. O problema de encontrar este par de caminhos de peso total mínimo pode ser modelado como um problema de alocação de tarefas. Se o peso total exceder $D - \sum_{i \in S} d_i$, uma Desigualdade Hipo-Ciclo Estendida de 2-arestas violada foi encontrada, e pode-se partir para o próximo conjunto S_i. Se uma desigualdade violada não for encontrada, arestas são removidas procurando a violação. Denotamos L_{rs} como o menor caminho entre dois vértices quaisquer r e s. Sejam os dois vértices terminais v_i e v_j da Desigualdade Hipo-Ciclo Estendida de 2-arestas não violada atual, definimos $L_{max} = D - (\sum_{i \in S} d_i) - L_{v_j 0}$. Seja $P = (v_i, 0) \cup \{p \in V_+ \backslash S : L_{v_i p} + L_{p0} \leq L_{max}\}$, observe que qualquer caminho $(v_i, ..., 0)$ de comprimento menor ou igual a L_{max} pode apenas visitar vértices em P. Considere então G_{ind_p} o subgrafo de G_{ind} induzido por P. A rotina encontra os componentes biconexos de G_{ind_p} e caso $L_{v_i 0}$ visite mais de um componente, todas as arestas incidentes sobre os vértices que se encontram na articulação dos componentes são removidas. Note que a cada articulação entre dois componentes biconexos, duas desigualdades podem ser geradas. Calculamos removendo todas as arestas adjacentes a um dos componentes e em seguida removendo as arestas adjacentes ao do outro componente. Se uma violação ainda não foi encontrada, a rotina iterativamente remove a aresta de menor valor x^* do subgrafo G_{ind_p}. A

cada aresta removida, verificamos se uma violação foi encontrada. Isto é feito até que o valor x^* da aresta atual seja grande demais para que ocorra uma violação.

Capítulo 5

Algoritmos para Tratar o Empacotamento Bidimensional do 2L-CVRP

5.1 Introdução

Neste capítulo descreveremos os algoritmos utilizados na resolução do problema de empacotamento do 2L-CVRP na abordagem proposta. Detalhes do uso destes algoritmos podem ser encontrados no capítulo 6.

O 2L-CVRP é um caso mais geral do CVRP, onde cada cliente possui uma demanda de um conjunto de itens retangulares com um peso associado. O 2L-CVRP apresenta-se muito mais difícil que o CVRP justamente por possuir um conjunto de demandas bidimensionais para cada cliente, além das demandas unidimensionais originais do CVRP. Logicamente, se resolvermos o 2L-CVRP ignorando as restrições de empacotamento bidimensional estaremos resolvendo o CVRP. Os algoritmos deste capítulo abordam a característica bidimensional do empacotamento, não considerando as demandas unidimensionais. As demandas unidimensionais são tratadas por outros algoritmos em nossa abordagem (ver capítulo 4).

Uma rota do 2L-CVRP é considerada viável se obedece a um conjunto de restrições. Uma dessas restrições é a necessidade da existência de um empacotamento viável dos itens no contêiner do veículo: os itens não podem se sobrepor, ultrapassar os limites do contêiner e suas arestas devem estar ortogonais às bordas do contêiner. Quando considerando a variante sequencial do 2L-CVRP, mais uma restrição é colocada: itens pertencentes a um cliente que esteja descarregando não podem estar bloqueados por itens de outros clientes.

Resolver o problema de empacotamento de apenas uma rota do 2L-CVRP corresponde a resolver o Problema de Empacotamento em Faixa Bidimensional. Um empacotamento é inviável se o seu comprimento na faixa ultrapassar o comprimento do contêiner do veículo. Deste modo, o problema de empacotamento associado ao 2L-CVRP é claramente NP-completo.

Neste capítulo apresentamos algoritmos que tratam o problema de empacotamento do 2L-CVRP. Ambas as variantes sequencial e irrestrita são consideradas. O capítulo encontra-se organizado do seguinte modo: na seção 5.2 descrevemos o limitante inferior utilizado na tenta-

tiva de se eliminar rapidamente empacotamentos inviáveis. A tabela de dispersão implementada para armazenar a viabilidade dos empacotamentos de rotas conhecidas é descrita na seção 5.3. Na seção 5.4 detalhamos uma abordagem heurística e na seção 5.5 descrevemos uma abordagem exata para descobrirmos a viabilidade de um empacotamento.

5.2 Limitantes Inferiores

Nesta seção detalharemos o limitante inferior utilizado na abordagem proposta. Este limitante foi apresentado por Martello, Pisinger e Vigo [11] e o denotaremos por $MPVL_2$. Como o $MPVL_2$ foi desenvolvido para o problema de empacotar um conjunto de caixas em um contêiner tridimensional, apresentaremos nossa adaptação para o caso bidimensional.

Descreveremos inicialmente um limitante, que denotaremos por $MPVL_1$, obtido por redução para o caso unidimensional que é utilizado no cálculo do limitante inferior $MPVL_2$.

Seja um contêiner bidimensional de dimensões W e L e seja J o conjunto completo dos itens a serem empacotados. Para cada item $j \in J$ temos as dimensões w_j e l_j associadas. Considere o conjunto β definido por

$$\beta = \left\{ j \in J : l_j > \frac{L}{2} \right\} \tag{5.1}$$

e o conjunto ξ definido por

$$\xi = \left\{ j \in \beta : w_j > \frac{W}{2} \right\}. \tag{5.2}$$

Sejam $J_s(p)$ e $J_l(p)$ definidos por

$$J_s(p) = \left\{ j \in \beta : \frac{W}{2} \geq w_j \geq p \right\} \tag{5.3}$$

$$J_l(p) = \left\{ j \in \beta : W - p \geq w_j > \frac{W}{2} \right\}. \tag{5.4}$$

Considere Γ e Υ definidos por

$$\Gamma(p) = \left\lceil \frac{\sum_{j \in J_s(p)} w_j - (|J_l(p)|W - \sum_{j \in J_l(p)} w_j)}{W} \right\rceil \tag{5.5}$$

$$\Upsilon(p) = \left\lceil \frac{|J_s(p)| - \sum_{j \in J_l(p)} \lfloor \frac{W - w_j}{p} \rfloor}{\lfloor \frac{W}{p} \rfloor} \right\rceil. \tag{5.6}$$

Podemos então obter o limitante inferior $MPVL_1^W$:

$$MPVL_1^W = |\xi| + \max_{1 \leq p \leq \frac{W}{2}} \{\Gamma(p), \Upsilon(p)\} \tag{5.7}$$

Considere também o limitante inferior $MPVL_1^L$ obtido de (5.1)-(5.7), substituindo-se W por L e w_j por l_j. $MPVL_1^W$ e $MPVL_1^L$ são combinações dos limitantes originalmente apresentados por Dell'Amico et al. em [40] e por Martello et al. em [36].

As observações a seguir consideram o limitante $MPVL_1^W$, mas aplicam-se de forma análoga ao limitante $MPVL_1^L$. O primeiro termo da soma ($|\beta|$) representa o conjunto de itens suficientemente grandes de forma que dado qualquer par deste conjunto, são necessários dois contêineres para empacotá-los. Consequentemente precisamos de um contêiner para cada elemento deste conjunto.

O segundo termo da soma é composto por dois elementos distintos na função *max*. Em ambos variamos p, procurando o melhor resultado das divisões entre os conjuntos de itens. Consideraremos primeiramente Γ. Descreveremos os itens em $J_s(p)$ e $J_l(p)$ como pequenos e grandes, respectivamente. O primeiro termo de Γ nos dá a largura total dos itens pequenos. O segundo termo calcula a largura total de todos os contêineres necessários para se colocar os itens grandes. Deste, subtraimos a largura realmente ocupada por estes itens, descobrindo a largura da folga existente. Ou seja, no numerador procuramos encaixar os itens pequenos nesta folga de espaço do empacotamento dos itens grandes. Consequentemente a divisão da sobra desta subtração pela largura do contêiner nos dá o número mínimo de contêineres necessário para colocarmos os itens pequenos que não puderam ser encaixados de forma fracionária nesta folga.

Abordaremos o segundo elemento da função *max*, representado por Υ. O primeiro termo de Υ representa o número de itens pequenos. O segundo termo nos dá o número de itens de tamanho p que cabem na folga do empacotamento dos itens grandes. Como p é a menor largura que um item pequeno pode possuir, a subtração dos dois termos representa a tentativa de se encaixar os itens pequenos na folga. O denominador da fração nos dá o número de itens de tamanho p que cabem em um contêiner. Deste modo, a divisão do que sobrou na subtração pelo denominador nos dá o número mínimo de contêineres necessário para encaixar os itens pequenos que não puderam ser encaixados na folga dos itens grandes.

Finalmente, definimos o limitante inferior $MPVL_1$ como o máximo entre os valores obtidos por $MPVL_1^W$ e $MPVL_1^L$:

$$MPVL_1 = max\{MPVL_1^W, MPVL_1^L\}. \tag{5.8}$$

Definiremos então uma adaptação para o caso bidimensional do limitante $MPVL_2$ originalmente apresentado por Martello, Pisinger e Vigo em [11]. Considere a notação utilizada na descrição do limitante $MPVL_1$ e seja a_j a área do item j e A a área do contêiner. Sejam $K_v(p)$, $K_l(p)$ e $K_s(p)$ definidos como

$$K_v(p) = \{j \in J : w_j > W - p\}, \tag{5.9}$$

$$K_l(p) = \{j \in J \backslash K_v(p) : w_j > \frac{W}{2}\}, \tag{5.10}$$

$$K_s(p) = \{j \in J \backslash (K_v(p) \cup K_l(p)) : w_j \geq p\}. \tag{5.11}$$

Seja então

$$MPVL_2^W = \max_{1 \leq p \leq \frac{W}{2}} \left\{ MPVL_1^W + max \left\{ 0, \left\lceil \frac{\sum_{j \in (K_l(p) \cup K_s(p))} a_j - (W\ MPVL_1^W - \sum_{j \in K_v(p)} w_j)L}{A} \right\rceil \right\} \right\} \tag{5.12}$$

Considere também o limitante inferior $MPVL_2^L$ obtido de (5.9)-(5.12), substituindo-se W por L e w_j por l_j.

Detalharemos o cálculo do limitante $MPVL_2^W$ a seguir. Descreveremos os itens em $K_v(p)$, $K_l(p)$ e $K_s(p)$ como grandes, médios e pequenos, respectivamente. Inicialmente varia-se o valor de p na função *max*, de forma a buscar o maior resultado levando-se em conta todas as divisões possíveis entre os itens. Esta função é composta por uma soma com dois termos. No primeiro termo temos o número de contêineres obtido pelo limitante $MPVL_1^W$. Uma função *max* é aplicada ao segundo termo para evitarmos o decréscimo deste valor que já sabemos ser válido. Analisaremos então a fração encontrada dentro da segunda função *max*. O primeiro termo do numerador nos dá a soma de todas as áreas dos itens médios e pequenos. O segundo termo nos dá a área total dos contêineres encontrados por $MPVL_1^W$ menos a área ocupada pelos itens grandes, ou seja, a área da folga existente no empacotamento dos itens grandes nos contêineres encontrados por $MPVL_1^W$. O numerador representa então a tentativa de encaixarmos os itens pequenos e médios na folga de espaço do empacotamento dos itens grandes. A divisão do que sobra pela área do contêiner nos dá o número de contêineres necessários para colocarmos os itens que não puderam ser encaixados. Observe que esta análise aplica-se de forma análoga ao limitante $MPVL_2^L$.

Definimos então o limitante inferior $MPVL_2$ como o máximo entre os valores obtidos por $MPVL_2^W$ e $MPVL_2^L$:

$$MPVL_2 = max\{MPVL_2^W, MPVL_2^L\} \tag{5.13}$$

Podemos então tentar descobrir a inviabilidade de uma rota com o cálculo deste limitante. Dada uma rota do 2L-CVRP, se o número de contêiners encontrado calculando $MPVL_2$ for maior que um significa que a rota é inviável. A utilização deste limitante na abordagem proposta está detalhada no capítulo 6.

5.3 Tabela de Dispersão para Armazenar Rotas

Uma vez descoberta a viabilidade do empacotamento de uma rota, verificações subsequentes tornam-se redundantes. Deste modo, buscando reduzir o esforço computacional, armazenamos rotas conhecidas em uma tabela de dispersão. Observe que armazenamos uma rota seja esta viável ou não. Descreveremos a seguir o algoritmo utilizado para a geração, armazenamento e verificação da chave.

O algoritmo proposto possui comportamentos diferentes dependendo da variante do 2L-CVRP abordada. Este algoritmo recebe como entrada um vetor composto pelos clientes da rota ordenado por ordem crescente de visita.

A ordem dos vértices é relevante para o caso sequencial, deste modo procuramos obter uma chave diferente para rotas distintas que possuam o mesmo conjunto de vértices. Dada uma rota com n vértices, considere um vértice qualquer de número k em uma posição arbitrária i no vetor de entrada: inicialmente calculamos *vertexHash* = $k \times (n - i)$, ou seja, cada vértice é multiplicado pelo tamanho da rota menos o seu índice no vetor (quando este cliente será visitado). O valor obtido é então ligeiramente deslocado por uma constante (utilizamos 3 nos testes computacionais) e multiplicado por seu valor anterior. Em seguida encontramos o resto de sua divisão pelo tamanho total da tabela de dispersão (que denotaremos como *sizeHash*). Ou seja, calculamos *vertexHash* = *(vertexHash* $\times$ *(vertexHash* + 3)) $\%$ *sizeHash*. Todas as chaves individuais de cada vértice são então somadas para obtermos uma única chave inicial para a rota. Esta chave é então multiplicada pela constante $(0.5 \times (\sqrt{5} - 1))$. Em seguida, extraimos a parte fracionária do valor resultante. Obtemos a chave final da rota truncando o resultado do produto desta parte fracionária por *sizeHash*.

Para o caso irrestrito, o procedimento é similar: como devemos obter uma única chave para um conjunto de vértices, não importando sua ordem de visita, somamos os números de todos os clientes da rota. Em posse deste valor, o algoritmo procede como no caso sequencial: multiplica o valor obtido por $(0.5 \times (\sqrt{5} - 1))$, remove a parte inteira e trunca o resultado do produto da parte fracionária pelo tamanho da tabela de dispersão.

Além da etapa de geração, o armazenamento e a busca também dependem de qual variante está sendo abordada. Para a variante sequencial apenas armazenamos o vetor original da rota. A busca consiste em verificar se existe uma discrepância entre o tamanho das rotas (a atual e a armazenada), e se não houver, o algoritmo verifica se os vetores são idênticos.

Para o caso irrestrito, duas rotas são iguais se possuirem os mesmos elementos, independente da ordem em que estes se encontram. Por exemplo, a rota $A = (5, 9, 3, 4)$ é idêntica a rota $B = (3, 4, 9, 5)$. Deste modo, o procedimento de armazenamento precisa considerar esta particularidade de forma a agilizar a comparação dos elementos. Para isto, alocamos um vetor com o número total de clientes, e definimos como 1 a posição equivalente ao cliente. Por exemplo, para a rota B, temos na tabela de dispersão um vetor com 1 nos índices $3, 4, 5$ e 9 e zero no resto. A busca consiste então em acessar diretamente as posições representadas pelos clientes e verificar se esta encontra-se com 1. Como no caso sequencial, o algoritmo também verifica se existe uma discrepância de tamanho entre as rotas antes de efetuar a comparação de elementos.

Caso ocorra colisão durante a operação de armazenamento, esta é resolvida através de encadeamento.

5.4 Heurística Com e Sem Restrição de Ordem

Descreveremos nesta seção a heurística de empacotamento utilizada para testar a viabilidade de rotas encontradas pelo algoritmo *branch-and-cut*. Esta heurística procura um empacotamento válido de todos os itens dos clientes em uma rota considerando a restrição sequencial. Ou seja, como o veículo possui apenas uma saída para efetuar o descarregamento, os itens são empacotados de modo que não sejam bloqueados por itens que serão servidos posteriormente.

O problema de empacotamento do 2L-CVRP é abordado de modo heurístico utilizando-se do procedimento $2DLPga$ (*Two-dimensional loading problem general algorithm*). Este procedimento consiste em chamar a heurística de empacotamento $2DLPh$ (*Two-dimensional loading problem heuristic*) de quatro modos diferentes, até encontrar o empacotamento dos itens da rota ou não conseguir verificar a sua viabilidade. Caso não consiga encontrar um empacotamento viável, uma abordagem exata é utilizada, descrita na seção 5.5.

O algoritmo $2DLPh$ recebe como parâmetros a lista de itens $T = (t_1, t_2, \ldots, t_n)$, cada item t_i com dimensões (w_i, l_i), um contêiner F de dimensões (W, L), um vetor C com clientes associados aos itens em T, uma variável booleana *ordenaPorComprimento* para definir o modo primário de ordenação e os fatores de tolerância α, β e γ. O algoritmo devolve um Empacotamento $\mathcal{P}$ de T em F obedecendo a ordem dos clientes em C ou que o empacotamento é inviável.

O $2DLPh$ inicialmente ordena a lista T em ordem crescente de cliente. Em seguida os itens são ordenados novamente, mas mantendo a ordem de clientes. Ou seja, um item só troca de lugar com itens do mesmo cliente. Nesta ordenação o parâmetro *ordenaPorComprimento* é considerado: caso este seja definido como verdadeiro, a ordenação é feita em ordem não crescente de comprimento. Caso contrário, a ordenação é feita em ordem não crescente de largura. Mais detalhadamente, sejam dois itens r e s de dimensões (w_r, l_r) e (w_s, l_s), respectivamente. O item r é posicionado antes do item s caso a seguinte condição seja satisfeita:

$$l_r - (\gamma \times L) < l_s \ \textbf{e} \ l_r + (\gamma \times L) > l_s. \tag{5.14}$$

Em caso de empate, o desempate é feito pela largura do item. Caso o parâmetro *ordenaPorComprimento* seja igual a falso, troca-se comprimento por largura e consequentemente o desempate é feito pelo comprimento.

O próximo passo do algoritmo consiste em iniciar o empacotamento dos itens. Para cada item $t_i \in T$ é chamada a função *calculaPontosFronteira*, que encontra as possíveis posições onde este item pode ser empacotado. Esta função possui duas etapas: na primeira etapa encontra todos os pontos superiores direitos dos itens que se encontram na fronteira do empacotamento e os projeta o mais à esquerda quanto for possível. Ou seja, até encontrar a aresta direita de um item já posicionado ou a aresta do contêiner. Na segunda etapa a função efetua a remoção dos pontos nos quais não é possível posicionar o item devido a largura ou comprimento disponível e pontos que não possuam contato com a fronteira do empacotamento após a sua projeção. Note que o fundo do contêiner F é considerado como se fosse um único item de largura W. A partir deste ponto exemplificaremos considerando o parâmetro *ordenaPorComprimento* como

verdadeiro, mas o raciocínio é análogo para o caso que este seja definido como falso. A figura 5.1 exemplifica o cálculo dos pontos de fronteira:

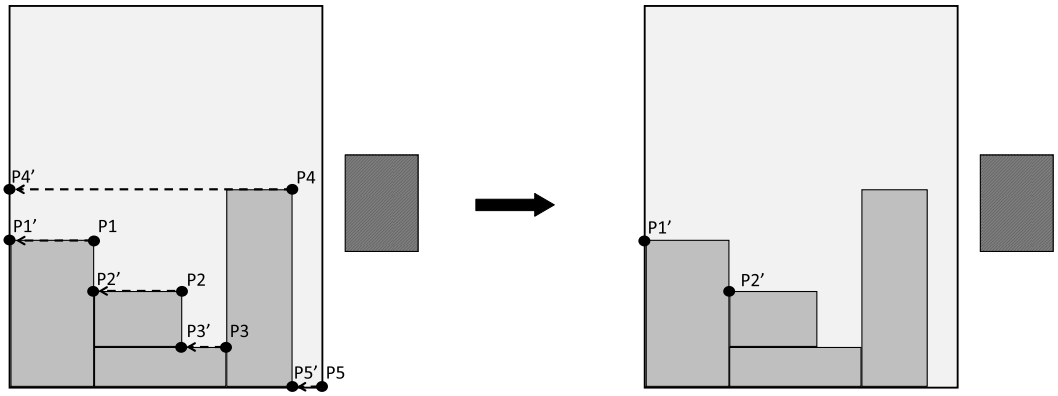

Figura 5.1: Exemplo do cálculo dos pontos de fronteira.

Considere o item localizado fora do contêiner. Após encontrar os pontos de fronteira para este item, os pontos são projetados o mais a esquerda possível, gerando P'_1, P'_2, P'_3, P'_4 e P'_5. Em seguida, o algoritmo percorre estes pontos verificando a viabilidade da colocação do item em cada um deles. Os pontos P'_1 e P'_2 são mantidos, sendo viáveis para o posicionamento do item. Os pontos P'_3 e P'_5 são removidos porque as larguras disponíveis são menores que a largura do item. O ponto P'_4 é removido por não possuir contato com a fronteira do empacotamento atual e por não ser largo o suficiente para se apoiar no item relativo ao ponto P'_4.

O algoritmo devolve um indicador de inviabilidade se não for encontrada nenhuma posição onde o item t_i pode ser colocado. Se forem encontrados pontos viáveis, o algoritmo inicia uma nova fase. Dado um ponto selecionado, este é comparado aos pontos à sua direita. A cada comparação um novo ponto pode substituir, ou não, o ponto temporariamente selecionado para a colocação do item. Isto ocorre em duas etapas: na primeira etapa um ponto é selecionado se o valor de seu eixo y for menor que o do ponto previamente escolhido. Esta comparação é efetuada aplicando-se uma tolerância α passada como parâmetro, de modo que o ponto precisa ter um valor de y tão significativamente menor quanto do ponto atualmente escolhido. Utilizaremos a figura 5.2 para exemplificar a varredura dos pontos. O algoritmo marca inicialmente o ponto projetado P'_1 como escolhido. A comparação é efetuada entre o ponto selecionado P'_1 e o ponto seguinte, o ponto projetado P'_2. Como P'_1 é o ponto escolhido, P'_2 só será levado em conta para a próxima etapa se sua coordenada y, que denotaremos por $P'_2.y$, for significativamente menor que a coordenada $P'_1.y$, dada uma tolerância α. Mais especificamente, se $P'_2.y < P'_1.y$ - $\alpha \times L$, o algoritmo avança para a próxima etapa. Caso contrário, P'_1 continua como o ponto escolhido e um novo ponto é selecionado para a comparação. Independente da escolha de P'_2, o ponto P'_3 será descartado por possuir $P'_3.y$ maior que ambos $P'_1.y$ e $P'_2.y$. Sabemos que se $P'_1.y - P'_2.y$ for relevante, então P'_4 será considerado para a comparação.

Encontrado um ponto com um valor de y significativamente menor que do ponto atualmente selecionado, o algoritmo inicia a segunda etapa da comparação, utilizando a função *areaPerdida*. Esta função calcula a área da região desperdiçada com a colocação do item neste ponto.

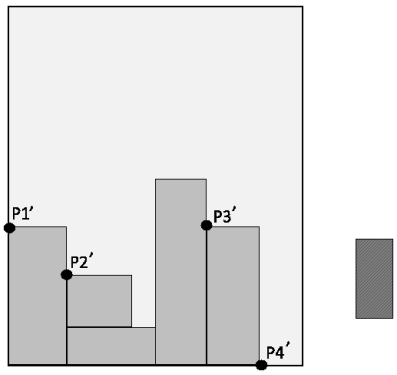

Figura 5.2: Exemplo da varredura dos pontos de fronteira.

Ou seja, a função encontra a área entre o item posicionado em um ponto e a fronteira do empacotamento atual. A figura 5.3 demonstra o cálculo da função *areaPerdida*. Seja P'_1 o ponto atualmente selecionado e P'_2 o ponto a ser comparado. Neste passo sabemos que há uma redução significativa do valor de y entre os dois pontos, já que P'_2 está sendo considerado. Para que P'_2 substitua P'_1 como o ponto selecionado, a área perdida com a colocação do item em P'_2 pode ser até $\beta \times W \times L$ maior que a área perdida com a colocação em P'_1. Ou seja, o algoritmo $2DLPh$ procura baixar o valor de y do ponto selecionado, e mesmo que a área desperdiçada pelo novo ponto seja um pouco maior, tanto quanto o cálculo de tolerância permitir, este ponto é escolhido.

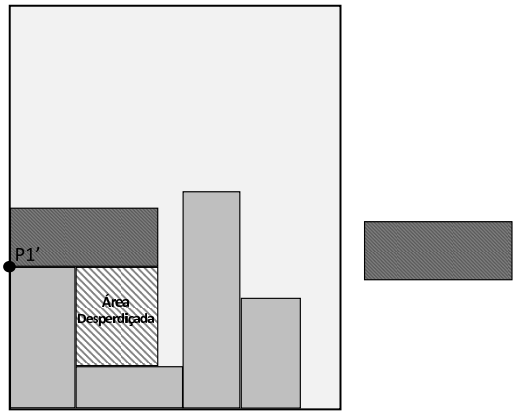

Figura 5.3: Exemplo do cálculo da área desperdiçada

Após percorrer todos os pontos, o item é colocado no ponto selecionado ao fim de todas as comparações. Quando todos os itens foram colocados no contêiner, o empacotamento $\mathcal{P}$ é devolvido.

Também apresentamos uma versão da heurística $2DLPh$ sem restrição de ordem para efetuarmos testes para o caso irrestrito do 2L-CVRP. É facil notar que a heurística $2DLPh$ pode ser modificada para não considerar a restrição de ordem, apenas mudando a sua etapa de ordenação. Denotamos essa modificação da heurística $2DLPh$ como $2DLPh_i$. O algoritmo $2DLPh_i$ consiste do $2DLPh$ tendo o passo da ordenação por cliente removido.

Algoritmo 3: Heurística de empacotamento bidimensional $2DLPh$

Entrada: Contêiner F de dimensões (W, L), lista T de itens, cada com dimensões (w_i, l_i), um vetor de clientes C associados aos itens em T, uma constante booleana *ordenaPorComprimento* e constantes reais α, β e γ
Saída: Empacotamento $\mathcal{P}$ de T em F
Ordena a lista de itens de acordo com o seguinte critério:
Dados dois itens r e s da lista T.
se *os items são do mesmo cliente, ou seja,* $C_r = C_s$ **então**
    se *ordenaPorComprimento = Verdadeiro* **então**
        se (l_r - ($\gamma \times L$) $<$ l_s) ***e*** (l_r + ($\gamma \times L$) $>$ l_s) **então**
            Coloca r e s em ordem não crescente de largura.
        senão
            Coloca r e s em ordem não crescente de comprimento.
    senão
        se (w_r - ($\gamma \times W$) $<$ w_s) ***e*** (w_r + ($\gamma \times W$) $>$ w_s) **então**
            Coloca r e s em ordem não crescente de comprimento.
        senão
            Coloca r e s em ordem não crescente de largura.
senão
    Coloca r e s em ordem crescente de cliente.
$\mathcal{P} \leftarrow \emptyset$
para $i \leftarrow 1$ **até** $|T|$ **faça**
    $\psi \leftarrow$ *calculaPontosFronteira*(F, $\mathcal{P}$, t_i)
    se $|\psi| = 0$ **então**
        Devolva *Empacotamento inviável*
    compMin $\leftarrow L + (\alpha \times L)$
    desperdicioMin $\leftarrow W \times L$
    para $j \leftarrow 1$ **até** $|\psi|$ **faça**
        comp $\leftarrow$ coordenada y de ψ_j
        se *comp* $<$ *compMin* - ($\alpha \times L$) **então**
            desperdicio $\leftarrow$ *areaPerdida*(ψ_j, t_i)
            se *desperdicio* $<$ *desperdicioMin* + ($\beta \times W \times L$) **então**
                desperdicioMin $\leftarrow$ *desperdicio*
                compMin $\leftarrow$ *comp*
                pontoEscolhido $\leftarrow \psi_j$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{$*pontoEscolhido*$\}$
Devolva $\mathcal{P}$

Algoritmo 4: Heurística de empacotamento bidimensional $2DLPh_i$

Entrada: Contêiner F de dimensões (W, L), lista T de itens, cada com dimensões (w_i, l_i), uma constante booleana *ordenaPorComprimento* e constantes reais α, β e γ
Saída: Empacotamento $\mathcal{P}$ de T em F
Ordena a lista de itens de acordo com o seguinte critério:
Dados dois itens r e s da lista T.
se *ordenaPorComprimento = Verdadeiro* **então**
    se *(l_r - ($\gamma \times L$) < l_s)* ***e*** *(l_r + ($\gamma \times L$) > l_s)* **então**
        Coloca r e s em ordem não crescente de largura.
    senão
        Coloca r e s em ordem não crescente de comprimento.
senão
    se *(w_r - ($\gamma \times W$) < w_s)* ***e*** *(w_r + ($\gamma \times W$) > w_s)* **então**
        Coloca r e s em ordem não crescente de comprimento.
    senão
        Coloca r e s em ordem não crescente de largura.
$\mathcal{P} \leftarrow \emptyset$
para $i \leftarrow$ *1* **até** $|T|$ **faça**
    $\psi \leftarrow$ *calculaPontosFronteira(F, $\mathcal{P}$, t_i)*
    se $|\psi| = 0$ **então**
        Devolva *Empacotamento inviável*
    compMin $\leftarrow L + (\alpha \times L)$
    desperdicioMin $\leftarrow W \times L$
    para $j \leftarrow$ *1* **até** $|\psi|$ **faça**
        comp $\leftarrow$ coordenada y de ψ_j
        se *comp* < *compMin - ($\alpha \times L$)* **então**
            desperdicio $\leftarrow$ *areaPerdida(ψ_j, t_i)*
            se *desperdicio* < *desperdicioMin* + $(\beta \times W \times L)$ **então**
                desperdicioMin $\leftarrow$ *desperdicio*
                compMin $\leftarrow$ *comp*
                pontoEscolhido $\leftarrow \psi_j$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{$*pontoEscolhido*$\}$
Devolva $\mathcal{P}$

Descreveremos agora o algoritmo $2DLPga$. Este algoritmo consiste em chamar a heurística de empacotamento $2DLPh$ (ou $2DLPh_i$) de quatro modos diferentes, até encontrar o empacotamento da rota ou não conseguir verificar a sua viabilidade:

(i) Ordenação por comprimento e o sentido original dos clientes

(ii) Ordenação por comprimento e o sentido inverso dos clientes

(iii) Ordenação por largura e o sentido inverso dos clientes

(iv) Ordenação por largura e o sentido original dos clientes

Inicialmente o $2DLPga$ chama a heurística $2DLPh$ com o parâmetro booleano *ordenaPorComprimento* definido como verdadeiro. Caso não encontre um empacotamento viável, a função *inverteListaDeClientes* é chamada. Esta função recebe o vetor de clientes C como parâmetro e o inverte, de modo que o primeiro cliente torna-se o último a ser visitado. Para compreender o benefício desta inversão, observe a figura 5.4.

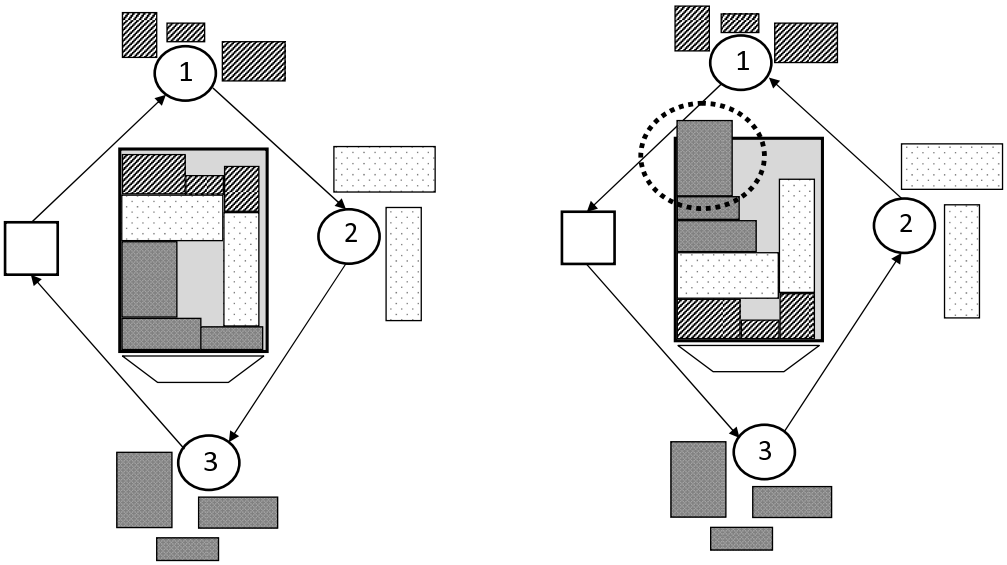

Figura 5.4: Exemplo do empacotamento de uma rota em ambos sentidos.

Nesta figura temos a tentativa do empacotamento dos itens de uma rota através de uma heurística *Bottom Left Decreasing Width* (ver [13]) modificada para considerar a restrição de ordem. A imagem à direita mostra que a heurística não foi capaz de encontrar um empacotamento viável dos itens. Invertendo a ordem de visita dos clientes temos a imagem a esquerda. Note que com esta inversão obtemos um empacotamento viável para a rota utilizando-se da heurística mencionada. Após a inversão do vetor de clientes C, o algoritmo chama novamente a heurística $2DLPh$. Caso não encontre um empacotamento viável com a rota invertida, o $2DLPga$ tenta mais duas estratégias: a heuristica $2DLPh$ é chamada com ordenação por largura – parâmetro *ordenaPorComprimento* definido como falso – para ambas a rota original e a invertida.

Algoritmo 5: Algoritmo geral para o empacotamento bidimensional $2DLPga$

Entrada: Contêiner F de dimensões (W, L), lista T de itens, cada um com dimensões (w_i, l_i), um vetor de clientes C associados aos itens em T e constantes reais α, β e γ
Saída: Valor booleano *empacotamentoViavel*

```
ordenaPorComprimento ← Verdadeiro
se 2DLPh(F, T, C, ordenaPorComprimento, α, β, γ) ≠ Empacotamento inviável então
    empacotamentoViavel ← Verdadeiro
senão
    inverteListaDeClientes(C)
    se 2DLPh(F, T, C, ordenaPorComprimento, α, β, γ) ≠ Empacotamento inviável
    então
        empacotamentoViavel ← Verdadeiro
    senão
        ordenaPorComprimento ← Falso
        se 2DLPh(F, T, C, ordenaPorComprimento, α, β, γ) ≠ Empacotamento inviável
        então
            empacotamentoViavel ← Verdadeiro
        senão
            inverteListaDeClientes(C)
            se 2DLPh(F, T, C, ordenaPorComprimento, α, β, γ) ≠ Empacotamento
            inviável então
                empacotamentoViavel ← Verdadeiro
            senão
                empacotamentoViavel ← Falso
Devolva empacotamentoViavel
```

Para os testes computacionais, ambas as versões com e sem ordem são consideradas. Efetuamos experimentos computacionais para determinarmos os fatores de tolerância α, β e γ. Para o parâmetro α e β observamos que valores abaixo de 0.1 e 0.05, respectivamente, geraram resultados melhores para as instâncias testadas do 2L-CVRP. Para o parâmetro γ encontramos resultados melhores utilizando uma relação entre o parâmetro α e o comprimento do contêiner. Definimos os valores dos parâmetros α, β e γ para os testes computacionais como 0.07, 0.005 e $\alpha \times \left(\frac{L}{2}\right)$, respectivamente.

5.5 Algoritmo Exato Com e Sem Restrição de Ordem

Nesta seção apresentaremos o algoritmo exato utilizado para verificar se todos os itens em uma rota podem ser empacotados no contêiner do veículo.

Martello, Pisinger e Vigo [11] propuseram uma abordagem para efetuar o empacotamento de caixas em contêineres. Utilizamos o algoritmo *branch-and-bound* desta abordagem para testar o empacotamento em um único contêiner e o modificamos para considerar a ordem sequencial de descarregamento dos clientes. Modificamos também a ordenação inicial dos itens, levando-se em conta esta nova restrição, de forma a melhorar sua eficiência. Observe que apesar deste algoritmo considerar o caso tridimensional, o seu funcionamento para o caso de duas dimensões é consequência direta, sendo este um caso mais específico.

A rotina $2D\text{-}Corners$ é utilizada para encontrar os pontos para o posicionamento de um conjunto de itens. Inicialmente os itens são ordenados em ordem não crescente de comprimento, com o desempate feito por largura. A rotina possui três fases: a primeira fase encontra os itens extremos, ou seja, itens que tenham seu ponto superior direito coincidindo com um ponto onde o envelope (a definição formal de envelope pode ser encontrada em [54]) muda de horizontal para vertical. A segunda fase consiste em encontrar os pontos de canto. Estes são definidos como as interseções entre as linhas saindo dos pontos superiores direitos dos itens encontrados na fase anterior. A terceira e última fase remove os pontos onde nenhum item de T pode ser posicionado devido ao espaço disponível.

O Algoritmo 6 detalha a rotina $2D\text{-}Corners$. Dado um conjunto de itens T, cada item $t_i \in T$ com dimensões (w_i, l_i) e um empacotamento existente $\mathcal{T}$, a rotina encontra o conjunto de pontos de canto $C(\mathcal{T})$. Note que e_m representa o item m do vetor de itens extremos e.

Em posse desta rotina, podemos então descrever o algoritmo para verificar a viabilidade do empacotamento de uma rota. O algoritmo *branch-and-bound* apresentado por Martello et al. [11] funciona do seguinte modo: seja $\mathcal{T}$ o conjunto de itens já empacotados no contêiner e $C(\mathcal{T})$ o conjunto de pontos de canto encontrados pela rotina $2D\text{-}Corners$. Denotaremos por $A(\mathcal{T})$ como a área do envelope definido pelos pontos de canto e A como a área total do contêiner. Seja T' o conjunto de itens ainda não empacotados onde cada item $i \in T'$ possui uma área a_j. Seja $A^*(\mathcal{T})$ a área do melhor empacotamento encontrado até o momento.

Inicialmente os itens são ordenados por área em ordem não crescente. A cada iteração $C(\mathcal{T})$ e $A(\mathcal{T})$ são calculados. Cada item $i \in T'$ é então colocado em cada ponto de canto $c \in C(\mathcal{T})$

Algoritmo 6: $2D\text{-}Corners$

Entrada: Empacotamento existente $\mathcal{T}$ e conjunto de itens a serem empacotados T
Saída: Conjunto de pontos de canto $C(\mathcal{T})$
se $\mathcal{T} = \emptyset$ **então**
  $C(\mathcal{T}) = \{(0, 0)\}$
senão
  $\bar{x} = m = 0$
  para $i = 1$ **até** $|\mathcal{T}|$ **faça**
    se $x_i + w_i > \bar{x}$ **então**
      $m = m + 1$
      $e_m = i$
      $\bar{x} = x_i + w_i$
  $C(\mathcal{T}) = \{(0, y_{e_1} + l_{e_1})\}$;
  para $i = 2$ **até** m **faça**
    $C(\mathcal{T}) = C(\mathcal{T}) \cup \left\{(x_{e_{i-1}} + w_{e_{i-1}}, y_{e_j} + l_{e_j})\right\}$
    $C(\mathcal{T}) = C(\mathcal{T}) \cup \left\{(x_{e_m} + w_{e_m}, 0)\right\}$
  para cada $(x'_i, y'_j) \in C(\mathcal{T})$ **faça**
    se $x'_j + min_{i \in T \setminus \mathcal{T}} \{w_i\} > W$ ***ou*** $y'_j + min_{i \in T \setminus \mathcal{T}} \{l_i\} > L$ **então**
      $C(\mathcal{T}) = C(\mathcal{T}) \setminus \left\{(x_j, y'_j)\right\}$
Devolva $C(\mathcal{T})$

e o algoritmo é chamado recursivamente. Se necessário $A^*(\mathcal{T})$ é atualizado. *Backtracking* ocorre quando mais nenhum item pode ser colocado no contêiner ou quando $\sum_{i \in \mathcal{T}} a_i + (A - A(\mathcal{T})) \leq A^*(\mathcal{T})$, já que se a área remanescente foi completamente preenchida não teriamos como melhorar $A^*(\mathcal{T})$.

Para lidar com a restrição de ordem, efetuamos a seguinte modificação no algoritmo: no momento que tentamos posicionar um item no contêiner, verificamos se este viola a restrição sequencial, podando o nó se isto ocorre. Logicamente esta modificação não é considerada pelo algoritmo proposto quando abordando o 2L-CVRP irrestrito. Modificamos também a ordenação inicial dos itens, sendo agora ordenados por cliente e o desempate sendo efetuado por área. Claramente isto tende a reduzir o número de violações, já que o algoritmo tenta colocar os últimos itens a serem descarregados no fundo do contêiner do veículo.

O algoritmo do trabalho completo de Martello et al., *The Three-Dimensional Bin Packing Problem*, [11] possui um código disponibilizado em [10]. Adaptamos este código para considerar apenas um contêiner e em seguida efetuamos as modificações propostas nesta seção. Os resultados computacionais foram obtidos utilizando-se desta nova versão.

Capítulo 6

Algoritmo Exato Branch-and-Cut para o 2L-CVRP

6.1 Introdução

Existem diversas aplicações práticas para o 2L-CVRP, dado que itens não podem ser empilhados uns em cima dos outros em várias situações do mundo real. Isto pode ocorrer por motivo de fragilidade, altura ou forma dos itens.

Duas variantes do 2L-CVRP foram exploradas, o caso sequencial e o caso irrestrito. Para o caso sequencial, o padrão de empacotamento deve obedecer uma certa ordenação dos itens. Ou seja, dado que o descarregamento é efetuado por um único lado do veículo, quando descarregando itens de um cliente estes não podem estar bloqueados por itens que pertençam a outros clientes. Esta restrição ocorre na prática devido a características dos itens, estes podendo ser muito frágeis ou muito pesados, fazendo com que a re-organização do empacotamento torne-se muito difícil ou demorada. O 2L-CVRP irrestrito relaxa esta restrição de ordem, permitindo que os itens encontrem-se em qualquer posição relativa a outros dentro do contêiner.

A única metodologia exata para o 2L-CVRP foi apresentada por Iori, Salazar-González e Vigo em [22], considerando apenas a variante sequencial. Observando que esta abordagem gastava uma parcela considerável do tempo apenas na resolução no problema de roteamento do 2L-CVRP, consideramos criar uma nova abordagem utilizando-se de várias desigualdades válidas para o CVRP.

Neste capítulo apresentamos um novo algoritmo exato para o 2L-CVRP baseado em uma abordagem *branch-and-cut*. Propomos duas estratégias cíclicas para a inserção de cortes utilizando cinco famílias de desigualdades válidas para o CVRP. As restrições de empacotamento são garantidas através da utilização de um algoritmo exato. Usamos também limitantes, uma heurística e uma tabela de dispersão buscando reduzir o esforço computacional. Ambas as variantes do problema são consideradas em nossa abordagem.

Este capítulo encontra-se organizado do seguinte modo: na seção 6.2 detalhamos a formulação em programação inteira utilizada para modelar o problema. O algoritmo branch-and-cut

proposto é descrito em detalhes na seção 6.3. As seções 6.4 e 6.5 descrevem as estratégias utilizadas para garantir as restrições de conectividade e peso e as restrições de empacotamento, respectivamente.

6.2 Formulação em Programação Linear Inteira

Seja x_e uma variável de fluxo igual a um se um veículo viaja usando a aresta $e \in E$ e zero caso contrário. Dado um subconjunto de clientes $S \subseteq V_+$, $d(S)$ é o peso total dos itens dos clientes em S, ou seja, $d(S) = \sum_{i \in S} d_i$. Seja B_i o conjunto de itens do cliente i, D a capacidade de peso do veículo e A a área do contêiner. Seja $a(S)$ a área total dos itens dos clientes S, ou seja, $a(S) = \sum_{i \in S} \sum_{b \in B_i} w_b l_b$. Seja $\delta(S)$ o conjunto de arestas em G que incidem em exatamente um vértice em S, ou seja, $\delta(S) = \{\{i, j\} \in E : i \in S, j \notin S\}$. Seja K a quantidade de veículos disponíveis para servir os clientes. Seja um conjunto $\mathcal{S} = \{S_1, \ldots, S_t\}$ de $t > 1$ conjuntos disjuntos de V_+. Seja $\mathcal{P}$ o conjunto de todas as l-partições viáveis V_+ e $P = \{V_1, \ldots, V_l\}$ uma destas l-partições. Seja σ definida por

$$\sigma(P, S) = |\{i : S_i \cap S \neq \emptyset\}| \tag{6.1}$$

e a função $\mathcal{R}$ definida por

$$R(\mathcal{S}) = min \left\{ \sum_{i=1}^{t} \sigma(P, S_i) \ : \ P \in \mathcal{P} \right\}, \tag{6.2}$$

temos então a seguinte formulação em programação inteira para o 2L-CVRP:

$$\begin{aligned}
\text{minimizar} \quad & \sum_{e \in E} c_e x_e & & \\
\text{s. a.} \quad & & & \\
& \sum_{e \in \delta(\{i\})} x_e = 2 & \forall i \in V_+ \quad & (1) \\
& \sum_{e \in \delta(\{0\})} x_e = 2K & & (2) \\
& \sum_{i=1}^{t} \sum_{e \in \delta(S)} x_e \geq 2\mathcal{R}(\mathcal{S}) & & (3) \\
& x_e \in \{0, 1\} & \forall e \in E. \quad & (4)
\end{aligned} \tag{6.3}$$

As restrições (1), as restrições de grau, impõem que cada cliente é visitado exatamente uma vez. As restrições (2) garantem que são utilizados exatamente K veículos para suprir as demandas dos clientes. As Restrições (3), as Desigualdades de Capacidade e Conectividade, garantem a conectividade da solução e as restrições de peso e de empacotamento bidimensional. Para este modelo utilizamos as Desigualdades de Capacidade Generalizadas, as mais fortes na hierarquia das Desigualdades de Capacidade e Conectividade. A tarefa de computar estas desigualdades

mostra-se inviável computacionalmente, e resolver o problema de separação correspondente é NP-difícil (ver seção 4.2). Finalmente, as restrições (4) definem a característica binária das variáveis.

A intratabilidade computacional das Desigualdades de Capacidade e Conectividade utilizadas, somado ao difícil problema de separação sugere a escolha de outra Desigualdade de Capacidade e Conectividade.

Como não se observa muita vantagem na escolha das Desigualdades de Capacidade ou das Desigualdades de Capacidade Fracas, pode-se então argumentar a favor do uso das Desigualdades de Capacidade Arredondadas. Apesar de serem significativamente mais fracas na hierarquia, estas podem ser tratadas computacionalmente. Entretanto, o problema de separação correspondente continua NP-difícil [14]. Substituindo as Desigualdades de Capacidade Generalizadas pelas Desigualdades de Capacidade Arredondadas temos então:

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{e \in E} c_e x_e & & \\
\text{s. a.} \quad & & & \\
& \sum_{e \in \delta(\{i\})} x_e = 2 & \forall i \in V_+ \quad & (1) \\
& \sum_{e \in \delta(\{0\})} x_e = 2K & & (2) \\
& \sum_{e \in \delta(S)} x_e \geq 2k'(S) & \forall S \subseteq V_+, |S| \geq 2 \quad & (3) \\
& x_e \in \{0,1\} & \forall e \in E. \quad & (4)
\end{aligned}
\tag{6.4}
$$

A função k' é tratável computacionalmente, mas como precisamos considerar todos os subconjuntos $S \subseteq V_+$, onde $|S| \geq 2$, temos um número exponencial de desigualdades. Removendo-se as restrições (3) e relaxando a restrição de integralidade das variáveis, temos uma relaxação que pode ser facilmente resolvida. A formulação correspondente é definida como:

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{e \in E} c_e x_e & & \\
\text{s. a.} \quad & & & \\
& \sum_{e \in \delta(\{i\})} x_e = 2 & \forall i \in V_+ \quad & (1) \\
& \sum_{e \in \delta(\{0\})} x_e = 2K & & (2) \\
& 0 \leq x_e \leq 1 & \forall e \in E. \quad & (3)
\end{aligned}
\tag{6.5}
$$

Esta relaxação gera um limitante inferior válido que pode então ser fortalecido através da adição de cortes. Logicamente não temos garantias quando considerando a conectividade da solução, as restrições de peso dos itens ou a viabilidade dos empacotamentos bidimensionais

das rotas. Deste modo cortes devem ser inseridos para garantirmos estas restrições e obtermos soluções válidas para o 2L-CVRP.

Note que as formulações apresentadas não permitem rotas com apenas um cliente. Caso queira-se permitir rotas que possuam apenas um cliente, deve-se substituir as restrições (4) das formulações por:

$$\begin{array}{ll} x_e \in \{0,1,2\} & \forall e \in \delta(0) \\ x_e \in \{0,1\} & \forall e \in E \setminus \delta(0). \end{array} \tag{6.6}$$

Consequentemente, para a relaxação (6.5), as restrições (3) são substituidas por:

$$\begin{array}{ll} 0 \leq x_e \leq 2 & \forall e \in \delta(0) \\ 0 \leq x_e \leq 1 & \forall e \in E \setminus \delta(0). \end{array} \tag{6.7}$$

Para os testes computacionais não consideramos rotas com um único cliente. Isto foi definido para possibilitar a comparação com os resultados da literatura. Este mesmo motivo também definiu a escolha do número de veículos utilizado em cada instância como o menor possível.

Descreveremos de forma sucinta como este valor foi encontrado por Iori et al. em [22]: as instâncias para o 2L-CVRP foram construídas em cima de instâncias existentes para o CVRP, já possuindo um número mínimo de veículos associado que denotaremos como K_{cvrp}. Para cada instância, Iori et al. [22] resolveram de forma heurística o Problema de Empacotamento em Contêineres Bidimensional (2BPP) associado. Em [22] não são oferecidos detalhes sobre a heurística utilizada, mas é mencionado que esta é uma adaptação para o caso bidimensional da heurística descrita em [11]. O valor de K para a instância do 2L-CVRP é então determinado como o valor máximo entre K_{cvrp} e o valor encontrado pela heurística. Iori et al. [22] observam que como as restrições de ordem e de clusterização (todos os itens de um cliente devem estar em um mesmo veículo) não são consideradas pela abordagem heurística para o 2BPP, o número de veículos resultante pode ser menor que o número mínimo realmente necessário. Isto logicamente levaria a uma instância inviável, mas de acordo com Iori et al. [22], esta situação não ocorreu com nenhuma das instâncias propostas.

6.3 Algoritmo Branch-and-Cut

Nesta seção descreveremos o algoritmo exato proposto para o 2L-CVRP utilizando uma abordagem *branch-and-cut*. Como detalhado na seção 2.4, um método *branch-and-cut* é um *branch-and-bound* em que planos de cortes são gerados dinamicamente nos nós da árvore de busca.

A existência de um número exponencial de restrições de capacidade e conectividade inviabiliza a utilização da formulação (6.4), deste modo obtemos um limitante inferior válido através da resolução da relaxação (6.5). Como a solução obtida pode violar várias restrições do 2L-CVRP, utilizamos planos de corte para remover as soluções indesejadas.

Apresentamos duas estratégias distintas para garantir as restrições necessárias. As restrições de peso dos itens e de conectividade do grafo são garantidas através da separação das Desigualdades de Capacidade Arredondadas e das Desigualdades de Capacidade Fracionais. Além destas, separamos também as seguintes desigualdades: Desigualdades de Capacidade Construídas, Desigualdades Multi-Estrela Homogêneas e Parciais, Desigualdades de Pente Fortalecidas e as Desigualdades Hipo-Ciclo Estendidas de 2-arestas. Detalhes sobre todas estas desigualdades e seus algoritmos de separação podem ser encontrados no capítulo 4. A estratégia de inserção de cortes utilizando-se destas desigualdades é descrita na seção 6.4. Para garantirmos as restrições de empacotamento, cortes são gerados através do uso do limitante inferior descrito na seção 5.2 e do algoritmo exato detalhado na seção 5.5. A estratégia de geração de cortes para o empacotamento encontra-se na seção 6.5.

Detalharemos então a estratégia geral para cada nó da árvore de enumeração. Tendo obtido um limitante inferior a partir da resolução da relaxação (6.5), efetuamos a separação das Desigualdades de Capacidade Arredondadas e das Desigualdades de Capacidade Fracionais. O algoritmo detalhado na seção 4.2 é então utilizado. Este algoritmo é composto de quatro rotinas. A primeira rotina tenta separar as Desigualdades de Capacidade Arredondadas. A partir do resultado desta primeira rotina podemos ter uma idéia da viabilidade da solução, já que se a solução for inteira e nenhum corte for encontrado temos uma solução conexa em que todas as rotas respeitam a restrição de peso do veículo.

Considere então que temos uma solução inteira e nenhum corte foi encontrado pela primeira rotina. Precisamos então verificar se todas as rotas tem um empacotamento viável. Ou seja, para cada rota da solução encontrada, precisamos checar se todos os itens podem ser empacotados no contêiner (com ou sem a restrição de ordem, dependendo da variante considerada). O algoritmo para verificar a viabilidade do empacotamento é utilizado. Detalhes deste algoritmo podem ser encontrados na seção 6.5, mas basicamente ele gera cortes de acordo com a viabilidade de cada rota da solução. Logicamente, caso nenhum corte seja encontrado, temos uma solução viável para o 2L-CVRP e esta pode ser utilizada para tentarmos atualizar a melhor solução encontrada até o momento durante a busca da árvore de *branch-and-bound*. Isto feito, como estamos considerando uma solução inteira, mudamos de nó. Caso cortes tenham sido encontrados, estes são adicionados ao sistema linear e re-otimizamos, já que esta solução não é viável para o 2L-CVRP.

Considere que temos uma solução fracionária e o algoritmo de separação das Desigualdades de Capacidade e Conectividade não conseguiu encontrar nenhum corte. Neste ponto chamamos o algoritmo para verificar a viabilidade do empacotamento (seção 6.5). O algoritmo identifica apenas as rotas inteiras (rotas sem arestas fracionárias) da solução fracionária, em seguida testando a viabilidade de seus empacotamentos. Cortes são então adicionados para cada rota inteira inviável encontrada e então re-otimizamos continuando neste nó. Caso cortes não tenham sido encontrados, seja porque todas as rotas são fracionárias, ou caso as rotas inteiras encontradas sejam viáveis, podemos então efetuar a etapa de ramificação. Utilizamos a estratégia de ramificação padrão do resolvedor linear Xpress-Optimizer v19.00.00 (ver seção 4.4 de [50]), utilizado nos testes computacionais (ver capítulo 7).

Considere que temos uma solução inteira ou fracionária e cortes foram encontrados. Neste ponto, estes cortes podem ter sido encontrados por qualquer uma das três primeiras rotinas do algoritmo de separação das Desigualdades de Capacidade e Conectividade. Lembramos que a quarta rotina apenas trabalha em cima de cortes existentes buscando gerar mais desigualdades violadas. Utilizamos então duas estratégias de separação, uma para o nó raiz e outra para os nós restantes da árvore. Estas estratégias procuram separar quatro famílias de desigualdades válidas para o CVRP. Mais especificamente, tentamos separar outra família de Desigualdades de Capacidade e Conectividade além de três outras famílias, de forma a tentar reduzir a região viável do politopo. Os detalhes destas estratégias de separação encontram-se na seção 6.4.

Nosso algoritmo também utiliza a geração de cortes de Gomory do resolvedor Xpress-Optimizer. Após experimentos computacionais definimos o número de cortes de Gomory a serem inseridos na raiz como vinte. Para os outros nós da árvore de busca, determinamos a inserção de quinze cortes de Gomory para cada novo nó gerado.

6.4 Estratégia para Separar as Restrições de Conectividade e Peso

Nesta seção propomos duas estratégias para efetuar a separação das restrições de conectividade e peso. Estas estratégias são utilizadas quando encontramos cortes com as rotinas de separação das Desigualdades de Capacidade Arredondadas e das Desigualdades de Capacidade Fracionais.

As seguintes desigualdades são utilizadas nas estratégias aqui apresentadas: Desigualdades de Capacidade Construídas, Desigualdades Multi-Estrela Homogêneas e Parciais, Desigualdades de Pente Fortalecidas e as Desigualdades Hipo-Ciclo Estendidas de 2-arestas. Detalhes das desigualdades listadas e das rotinas de separação correspondentes encontram-se no capítulo 4.

Propomos duas estratégias distintas, já que experimentos extensivos demonstraram ser mais efetivo tratar os nós não raiz de um modo diferente do nó raiz. Apresentaremos inicialmente a estratégia proposta para o nó raiz. Dado que esta estratégia é utilizada apenas se encontrarmos cortes de Capacidade e Conectividade, verificamos se a maior violação encontrada pelo algoritmo de separação descrito em 4.2 é menor ou igual a 0.2. Caso a maior violação seja suficientemente grande, excedendo 0.2, não consideramos necessário tentar separar outras desigualdades válidas nesta iteração e re-otimizamos. Se a maior violação encontrada não excede 0.2, tentamos separar as Desigualdades de Capacidade Construídas. Determinamos como 20000 o número máximo de nós as serem explorados na árvore de busca usada para a separação das Desigualdades de Capacidade Construídas. Caso encontremos cortes com esta separação, re-otimizamos. Se não encontrarmos Desigualdades de Capacidade Construídas violadas utilizamos então duas estratégias cíclicas procurando aumentar a probabilidade de encontrarmos mais planos de corte a cada re-otimização do nó raiz.

Uma das estratégias ocorre a cada quinta re-otimização do nó raiz. Ou seja, caso a iteração atual seja múltipla de cinco tentamos separar as Desigualdades Multi-Estrela Homogêneas e Parciais e as Desigualdades de Pente Fortalecidas. Observe que os algoritmos de separação

correspondentes são chamados incondicionalmente e que o número de iterações começa em zero, deste modo esta estratégia é utilizada na primeira iteração do nó raiz. Após tentarmos as separações re-otimizamos, já que no mínimo teremos desigualdades violadas encontradas pelo algoritmo de separação para as Desigualdades de Capacidade e Conectividade.

A outra estratégia é utilizada para todas as outras iterações do nó raiz. Utilizamos uma estratégia circular onde tentamos separar três famílias de desigualdades válidas, as Desigualdades Multi-Estrela Homogêneas e Parciais, as Desigualdades de Pente Fortalecidas e as Desigualdades Hipo-Ciclo Estendidas de 2-arestas. Detalharemos a seguir os três ciclos utilizados.

No primeiro ciclo tentamos separar as Desigualdades Multi-Estrela Homogêneas e Parciais se a maior violação encontrada para as Desigualdades de Capacidade e Conectividade é menor ou igual a 0.05. Se estiver neste limite e cortes forem encontrados, re-otimizamos. Se a maior violação for maior que 0.05 ou se cortes não forem encontrados, verificamos se a maior violação é menor ou igual a 0.1, e então tentamos separar as Desigualdades de Pente Fortalecidas. Se cortes forem encontrados fazemos a re-otimização. Caso não tenhamos encontrados Desigualdades de Pente Fortalecidas violadas ou caso a maior violação das Desigualdades de Capacidade e Conectividade for maior que 0.1, tentamos separar as Desigualdades Hipo-Ciclo Estendidas de 2-arestas. A separação é efetuada se a maior violação das Desigualdades de Capacidade e Conectividade for menor ou igual a 0.1. Caso não encontremos desigualdades violadas ou o algoritmo de separação nem tenha sido chamado devido ao valor máximo da violação das Desigualdades de Capacidade e Conectividade, então re-otimizamos apenas com os cortes encontrados das Desigualdades de Capacidade e Conectividade. Do mesmo modo, se cortes forem encontrados com a separação das Desigualdades Hipo-Ciclo Estendidas de 2-arestas, a re-otimização também é efetuada após adicionar estes cortes aos já encontrados com a separação das Desigualdades de Capacidade e Conectividade.

Existem outros dois ciclos, ambos utilizando-se da estratégia acima, mas mudando a ordem em que tentamos separar as três desigualdades. No segundo ciclo a ordem é: Desigualdades de Pente Fortalecidas, Desigualdades Hipo-Ciclo Estendidas de 2-arestas e Desigualdades Multi-Estrela Homogêneas e Parciais. No terceiro, temos as Desigualdades Hipo-Ciclo Estendidas de 2-arestas, seguidas das Desigualdades Multi-Estrela Homogêneas e Parciais e das Desigualdades de Pente Fortalecidas.

A cada iteração, o ciclo a ser utilizado depende de qual família de Desigualdades tentamos separar, com ou sem sucesso por último na iteração anterior. A idéia é forçar uma mudança de família para a próxima iteração. Se encontramos desigualdades violadas de uma família, na próxima iteração tentamos outra família procurando gerar uma certa perturbação. Caso não tenhamos encontrado cortes, mudamos então o ciclo para tentarmos achar cortes de outras famílias de desigualdades. Por exemplo, suponha que estamos no primeiro ciclo, e tentamos separar as Desigualdades de Pente Fortalecidas e conseguimos encontrar cortes. Na próxima iteração o terceiro ciclo será usado. Suponha então que estamos no segundo ciclo e tentamos separar as Desigualdades Hipo-Ciclo Estendidas de 2-arestas e não encontramos cortes, mas a maior violação encontrada entre os cortes das Desigualdades de Capacidade e Conectividade é 0.1. Seguindo a ordem deste ciclo o algoritmo tenta separar as Desigualdades Multi-Estrela

Homogêneas e Parciais se a maior violação for menor ou igual a 0.05. Como esta é 0.1, então não tentaremos efetuar esta separação e re-otimizamos. Na próxima iteração, obedecendo todas as condições até este ponto, o algoritmo usará o primeiro ciclo, já que a próxima família de desigualdades a ser considerada é a das Desigualdades Multi-Estrela Homogêneas e Parciais. Ou seja, tentamos separar as desigualdades de acordo com uma ordem circular, obedecendo a sequência: Desigualdades Multi-Estrela Homogêneas e Parciais, Desigualdades de Pente Fortalecidas e as Desigualdades Hipo-Ciclo Estendidas de 2-arestas.

Em nossos experimentos observamos ser mais efetivo tratar os outros nós da árvore de forma diferente do nó raiz. Como no caso do nó raiz, esta estratégia é utilizada quando encontramos cortes com as rotinas de separação das Desigualdades de Capacidade Arredondadas e das Desigualdades de Capacidade Fracionais. Utilizamos a seguinte estratégia: na primeira iteração de cada nó tentamos separar as Desigualdades de Capacidade Construídas, as Desigualdades Multi-Estrela Homogêneas e Parciais e as Desigualdades de Pente Fortalecidas. Para as Desigualdades de Capacidade Construídas, definimos o número máximo de nós a serem explorados na árvore de busca usada em sua separação como 100. Todas estas rotinas de separação são chamadas incondicionalmente. Nas iterações subsequentes esta estratégia não é utilizada, ou seja, tentamos separar apenas as Desigualdades de Capacidade e Conectividade.

6.5 Estratégia para Separar as Restrições de Empacotamento

Nesta seção detalharemos a estratégia utilizada para garantirmos as restrições de empacotamento de uma solução. Dado que definimos o número de veículos a serem utilizados para um valor fixo K, o algoritmo determina a viabilidade de uma solução dependendo da viabilidade de todas as K rotas existentes. Se todas as rotas possuirem empacotamentos viáveis, a solução é viável para o empacotamento. Este algoritmo é utilizado quando temos uma solução viável para as restrições de peso e conectividade, sendo esta solução inteira ou fracionária. Denotamos de rotas inteiras como rotas compostas apenas de arestas inteiras.

Geramos um corte para uma rota inviável adicionando a restrição equivalente ao conjunto das arestas desta rota. Por exemplo, suponha uma rota inviável composta pelas arestas $5, 9, 20$ e 52. O corte equivalente a esta rota seria $x_5 + x_9 + x_{20} + x_{52} \leq 3$. Ou seja, não podemos ter todas estas arestas juntas em uma rota de uma solução, já que sabemos ser inviável.

Considere então uma solução inteira ou fracionária: inicialmente encontramos as rotas inteiras da solução, excluindo as rotas que possuam arestas fracionárias. Para cada rota inteira encontrada, verificamos a sua existência na tabela de dispersão. Detalhes sobre a busca na tabela de dispersão estão na seção 5.3. Considere uma rota inteira encontrada pelo algoritmo: se encontrarmos a rota na tabela de dispersão significa que já sabemos a sua viabilidade. Deste modo, caso seja inviável, o corte equivalente já encontra-se inserido no sistema.

Caso a rota não tenha sido encontrada na tabela de dispersão utilizamos o limitante inferior apresentado em 5.2. Se o cálculo do limitante devolver a necessidade de mais de um contêiner para carregar todos os itens da rota sabemos que esta é inviável. Deste modo, geramos o

corte equivalente a rota e esta é adicionada a tabela de dispersão (ver seção 5.3) com a sinalização desta ser inviável. Caso contrário, se o limitante não nos mostrar que a rota é inviável, logicamente nenhuma conclusão pode ser tirada.

Se a rota não foi considerada inviável pelo limitante inferior então utilizamos uma abordagem heurística. Esta consiste em utilizar o procedimento $2DLPga$ (ver seção 5.4), que chama a heurística de empacotamento $2DLPh$ de quatro modos diferentes, até encontrar um empacotamento dos itens da rota em um contêiner ou não conseguir verificar a sua viabilidade. No primeiro modo colocamos os itens em ordem não crescente de comprimento e chamamos a heurística $2DLPh$ (detalhes desta heurística em ver 5.4). Caso esta não tenha encontrado um empacotamento viável, mantemos a ordenação, mas invertemos a sequência dos clientes e chamamos $2DLPh$ novamente. Se ainda não conseguimos efetuar o empacotamento dos itens, utilizamos a sequência dos clientes da primeira tentativa, mas desta vez ordenamos os itens por ordem não crescente de largura. Se esta terceira tentativa não conseguir gerar um empacotamento válido, mantemos a ordenação por largura, mas invertemos a sequência dos clientes. Os detalhes desta abordagem e dos algoritmos utilizados podem ser encontrados na seção 5.4.

Caso encontremos um empacotamento viável com a abordagem heurística adicionamos a rota à tabela de dispersão, com a sinalização de ser viável. Caso contrário, utilizamos o algoritmo exato de empacotamento bidimensional descrito na seção 5.5. Adicionamos então a rota na tabela de dispersão, sinalizando se esta é viável ou não. Caso não seja viável inserimos o corte equivalente no sistema.

Esta estratégia é aplicada para todas as rotas inteiras da solução. Claramente, se a solução for inteira, viável para as restrições de conectividade e peso e todas as rotas possuirem empacotamentos viáveis nos contêineres, esta solução é viável para o 2L-CVRP.

Capítulo 7

Resultados Computacionais

7.1 Introdução

O algoritmo proposto foi codificado em C e testado com instâncias disponibilizadas em [45]. Para resolver o modelo de programação linear inteira, utilizamos o resolvedor Xpress-Optimizer v19.00.00. Os experimentos foram executados em um único core de 2.40GHz de um Intel Core Q6600.

Efetuamos testes com todas as instâncias já resolvidas de forma exata na literatura, além de um conjunto de instâncias que ainda não possuiam o ótimo conhecido. Ambas as variantes do 2L-CVRP foram consideradas para os testes computacionais.

Nas tabelas 7.2 e 7.3 demonstramos o desempenho do algoritmo proposto para o caso sequencial e o comparamos com a abordagem exata de Iori, Salazar-González e Vigo [22]. Nas tabelas 7.4 e 7.5 abordamos a variante irrestrita e efetuamos uma comparação com os resultados obtidos para o caso sequencial. Finalmente, na tabela 7.6 analisamos o impacto que a adição de uma heurística primal poderia ter no algoritmo apresentado.

Na variante sequencial do 2L-CVRP o empacotamento deve obedecer a uma certa ordenação dos itens. Como o descarregamento é efetuado por um único lado do veículo, os itens de um cliente que esteja descarregando não podem estar bloqueados por itens de outros clientes. Características dos itens fazem com que esta restrição ocorra na prática. Os itens podem ser muito frágeis ou pesados, dificultando a re-organização do empacotamento no contêiner.

O desempenho do algoritmo proposto para a variante sequencial do 2L-CVRP pode ser observado nas tabelas 7.2 e 7.3. Comparamos os resultados obtidos com os de Iori, Salazar-González e Vigo [22] por ser a única metodologia exata para o 2L-CVRP de que temos conhecimento. Para possibilitar a comparação, truncamos as distâncias euclidianas entre os vértices e definimos o número de veículos utilizado na formulação linear inteira como o menor possível. Veja a seção 6.2 para detalhes de como foi encontrado o número de veículos para cada instância.

Comparação com outros trabalhos é difícil devido a diferenças de *hardware*, detalhes de implementação, a escolha do resolvedor linear e até mesmo o fato que versões diferentes de um mesmo resolvedor podem gerar resultados bem distintos. Sugerimos então que o leitor

mantenha uma postura crítica quando estudando os resultados apresentados.

7.2 Resultados Obtidos para o 2L-CVRP Sequencial

Descreveremos as colunas das tabelas 7.2 e 7.3 a seguir. Detalhes sobre a instância são encontrados nas primeiras quatro colunas. Nome e classe identificam a instância. As colunas n e L mostram o número de vértices e o número total de itens, respectivamente. A próxima coluna, K, mostra o número de veículos utilizado para resolver a instância. Dados sobre o esforço de empacotamento são encontrados nas próximas quatro colunas. Em P_{he} temos o número de rotas que a heurística para empacotamento bidimensional (ver seção 5.4) conseguiu gerar um empacotamento viável. O número de rotas que a heurística não foi capaz de resolver, mas o algoritmo exato (ver seção 5.5) provou possuirem um empacotamento viável encontra-se na coluna P_{ex}. A coluna P_{in} representa o número de rotas encontradas durante a resolução da instância que o algoritmo exato ou o limitante inferior (seções 5.5 e 5.2) provaram ser inviáveis. O número de vezes em que rotas foram encontradas na tabela de dispersão é mostrado por P_{ha}. Este número representa quantas repetições de rotas foram encontradas durante a resolução da instância. As colunas C_{pck} e C_{rot} mostram o número de cortes de empacotamento e de conectividade e peso inseridos, respectivamente. Em C_{ior} temos o número total de cortes inseridos pela abordagem de Iori et al. [22]. Esta coluna foi adicionada para permitir uma comparação com o número total de cortes inserido pelo algoritmo proposto, representado pela próxima coluna, C_{tot}. Esta coluna é a soma de C_{pck} e C_{rot}. A coluna N_{bb} mostra o número total de nós da árvore de enumeração que foram explorados na busca da solução. Não sabemos o número de nós explorados pelo algoritmo de Iori et al. [22]. A coluna Z mostra o valor da solução ótima, quando encontrada. Finalmente, as colunas T_{ior} e T_s mostram o tempo computacional em segundos utilizado para resolver a instância com o algoritmo de Iori et al. e com o algoritmo proposto, respectivamente. Observe que Iori, Salazar-González e Vigo utilizaram um Intel Pentium IV 3.0GHz para obter T_{ior}.

Definimos um limite de 5400 segundos para o algoritmo resolver as instâncias com o ótimo já conhecido. Para as instâncias ainda não resolvidas de forma exata na literatura, definimos um limite de 86400 segundos. Todos os resultados apresentados por Iori et al. em [22] foram obtidos com este limite de tempo.

Nas tabelas 7.2 e 7.3 temos todas as instâncias da literatura que possuem uma solução exata conhecida e um conjunto de instâncias que anteriormente não possuiam o ótimo conhecido. As instâncias que possuem o nome e classe marcados em negrito foram resolvidas pela primeira vez de forma exata por nosso algoritmo. As instâncias em que o nosso algoritmo resolveu com um tempo computacional menor que a abordagem de Iori et al. possuem a coluna T_s marcada em negrito. As instâncias que não puderam ser resolvidas no tempo máximo permitido possuem o tempo limite (5400 segundos para o nosso algoritmo e 86400 segundos para o algoritmo de Iori et al.) na coluna do tempo e um marcador l em seu lado direito indicando a sua não resolução de forma exata.

Analisaremos então alguns resultados demonstrados na tabela 7.2 e 7.3 e apresentaremos alguns estudos mais detalhados de certas instâncias. As observações sobre os tempos de empacotamento e roteamento do algoritmo de Iori et al. baseiam-se nos dados colocados na tabela 7.1. Nesta tabela listamos apenas as instâncias que analisamos aqui, mas os resultados completos podem ser encontrados em [22]. As duas primeiras colunas da tabela 7.1 mostram o nome e a classe da instância. As colunas seguintes, T_{pck}^{ior} e T_{rot}^{ior}, mostram o tempo que o algoritmo de Iori et al. gastou em rotinas de empacotamento e no problema de roteamento, respectivamente.

Tabela 7.1: Tempo gasto em empacotamento e roteamento pelo algoritmo de Iori et al.

Nome	Classe	T_{pck}^{ior}	T_{rot}^{ior}
$E016\text{-}03m$	5	40.01	0.02
$E021\text{-}06m$	5	46.36	0.37
$E022\text{-}04g$	5	1867.22	0.16
$E030\text{-}03g$	1	0.49	54635.13

A instância $E016\text{-}03m\text{-}5$ apresenta um tempo grande de resolução quando consideramos que apenas dois nós da árvore de busca foram explorados para encontrarmos a solução ótima. Este tempo é explicado pela coluna P_{in}. Uma análise detalhada demonstrou que o algoritmo gasta cerca de 1329 segundos verificando a viabilidade de uma rota com o algoritmo exato de empacotamento, que a define como inviável. A abordagem de Iori et al. é mais rápida descobrindo a inviabilidade desta rota, mas como pode-se ver em [22], o algoritmo deles gasta 40.01 segundos dos 40.03 segundos do tempo total no problema de empacotamento. Esta é uma instância com uma situação particularmente difícil de empacotamento bidimensional. A heurística de empacotamento bidimensional proposta não conseguiu resolver um difícil empacotamento na instância $E016\text{-}03m\text{-}5$, deixando o empacotamento para ser resolvido pelo algoritmo exato. Isto ocasionou um grande gasto de tempo, fazendo com que a instância fosse resolvida em mais de trinta minutos. Logicamente desejamos que a heurística ou o limitante possam encontrar ou cortar rotas para evitar que estas tenham que ser resolvidas pelo algoritmo exato. Mas a heurística mostrou-se fundamental em várias outras situações, como por exemplo na resolução da instância $E021\text{-}06m\text{-}5$. Esta é outra instância que possui a característica de ter um problema de empacotamento bidimensional difícil. A abordagem de Iori et al. gasta 46.36 de 46.74 segundos no problema de empacotamento (ver tabela 7.1). Na tabela 7.2 pode-se observar nas colunas P_{he} e P_{ex} que seis rotas tiveram seu empacotamento resolvido pela heurística e o algoritmo não necessitou usar o algoritmo exato para verificar a viabilidade de nenhuma rota. Deste modo, a solução ótima foi encontrada extremamente rápido, em apenas 0.1 segundo.

Logicamente, são raras as situações onde ambas abordagens encontram exatamente a mesma dificuldade com o empacotamento. Na maioria dos casos, as restrições de conectividade e peso e de empacotamento mostram-se completamente dependentes. A inserção de certos cortes pode mudar o problema de modo que o empacotamento bidimensional fique mais fácil ou mais difícil. O mesmo ocorre na situação oposta, onde um corte de empacotamento pode mudar consideravelmente a dificuldade do problema de roteamento. Com o estudo do 2L-CVRP podemos

perceber que este combina o roteamento e empacotamento de tal forma que geralmente os dois subproblemas encontram-se fortemente ligados. Um exemplo disto é a instância $E022\text{-}04g\text{-}5$, outra instância que estudamos mais detalhadamente. Para resolver esta instância o algoritmo de Iori et al. precisou de 1867.38 segundos, gastando 1867.22 segundos com o problema de empacotamento (ver tabela 7.1). Inicialmente parecia ser uma instância com um problema de empacotamento difícil, mas o algoritmo proposto apresentou uma redução significativa de tempo, resolvendo a instância em apenas 0.12 segundos. Uma análise detalhada, observando as rotas geradas durante a resolução desta instância, revelou que os empacotamentos encontrados por nossa abordagem não foram particularmente difíceis. A inserção de certas desigualdades transformaram o problema de modo que rotas com problemas de empacotamento de demorada resolução nem chegaram a aparecer.

Além de $E022\text{-}04g\text{-}5$, podemos observar que, quando efetuando a comparação com a abordagem anterior, várias instâncias também apresentaram reduções significativas no tempo de resolução. Podemos destacar a instância $E030\text{-}03g\text{-}1$, para qual o método proposto obteve a solução exata em apenas 0.99 segundos após explorar 135 nós da árvore de busca. O algoritmo de Iori et al. teve dificuldades com o roteamento nesta instância (ver tabela 7.1), gastando 54635.61 segundos, no total, para encontrar o ótimo. O mesmo ocorre com a instância $E026\text{-}08m\text{-}3$, onde o algoritmo proposto precisou de 2.00 segundos para encontrar a solução ótima, contra 164.22 segundos da abordagem anterior. Ganhos relevantes de tempo também podem ser observados em várias outras instâncias, como na $E022\text{-}04g\text{-}4$, $E022\text{-}06m\text{-}2$, $E022\text{-}06m\text{-}3$, $E026\text{-}08m\text{-}1$, $E026\text{-}08m\text{-}2$, $E033\text{-}03n\text{-}1$, $E033\text{-}05s\text{-}1$, entre outras.

O algoritmo apresentado também obteve reduções substanciais de tempo quando resolvendo as instâncias $E036\text{-}11h$, que possuem o maior número de vértices de todo o conjunto de teste. Nestas, o número de cortes inseridos foi consideravelmente maior que da abordagem exata anterior, mas este esforço computacional adicional de geração e inserção dos cortes foi completamente superado por seus benefícios. Para todas as classes da instância $E036\text{-}11h$, nosso algoritmo inseriu em média de 87.62 vezes mais cortes que o algoritmo de Iori et al. e obteve uma redução percentual de $78,82\%$, na média, do tempo de resolução.

Observando-se as colunas C_{tot} e C_{ior} das tabelas 7.2 e 7.3 pode-se perceber que a a abordagem proposta apresentou a característica de inserir consideravelmente mais cortes que a abordagem anterior. Entre todas as instâncias resolvidas, as únicas exceções foram as instâncias $E023\text{-}03g\text{-}1$, $E023\text{-}05s\text{-}1$, $E030\text{-}03g\text{-}1$ e $E033\text{-}03n\text{-}1$.

O algoritmo não conseguiu resolver todas as instâncias testadas no tempo limite. Isto é esperado, já que o 2L-CVRP é um problema recente e ainda pouco compreendido. Estudos preliminares das razões que estas instâncias mostraram-se difíceis para a abordagem proposta ainda não geraram respostas. Observe que em alguns destes casos não podemos saber se a instância seria resolvida, inclusive com um tempo menor ou maior que o algoritmo de Iori et al., já que definimos um tempo limite de execução significativamente menor (5400 contra 86400 segundos). Isto ocorre com as instâncias $E030\text{-}03g\text{-}1$, $E033\text{-}03n\text{-}3$ e $E033\text{-}04g\text{-}4$, onde a abordagem anterior precisou de mais de 5400 segundos para encontrar a solução ótima. Foram necessários 23699.77, 14240.75 e 25542.6 segundos, respectivamente.

Além de testarmos as instâncias que já haviam sido resolvidas de forma exata na literatura, também testamos um pequeno conjunto de instâncias que ainda não possuiam o ótimo conhecido. Para estas instâncias definimos um limite de tempo de execução maior, de 86400 segundos. Estes experimentos resultaram com a nossa abordagem encontrando a solução ótima de nove instâncias pela primeira vez na literatura.

As instâncias que o nosso algoritmo resolveu até a otimalidade pela primeira vez são: $E031$-$09h$-1, $E031$-$09h$-2, $E031$-$09h$-3, $E031$-$09h$-5, $E033$-$04g$-1, $E045$-$04f$-1, $E051$-$05e$-1, $E072$-$04f$-1 e $E101$-$10c$-1. Estas encontram-se nas tabelas 7.2 e 7.3 com o nome e a classe em negrito. Dentre estas, a mais demorada foi a $E031$-$09h$-2 na qual o algoritmo precisou de 48680.60 segundos para encontrar a solução exata. Nosso algoritmo inseriu um número de cortes substancialmente maior para esta instância do que todas as outras resolvidas, inserindo 1861987 cortes espalhados por 24803 nós da árvore de enumeração. A heurística de empacotamento mostrou-se eficiente na instância $E031$-$09h$-3, descobrindo a viabilidade de 67 rotas testadas e necessitando do uso do algoritmo exato em apenas dois casos.

Tabela 7.2: Desempenho do algoritmo Branch-and-cut para o 2L-CVRP Sequencial

Nome	Classe	n	L	K	P_{he}	P_{ex}	P_{in}	P_{ha}	C_{pck}	C_{rot}	C_{ior}	C_{tot}	N_{bb}	Z	T_{ior}	T_s
$E016\text{-}03m$	1	16	15	3	7	0	0	7	0	529	209	529	51	273	0.94	**0.19**
	2	16	24	3	49	22	181	1224	452	15095	702	15547	1797	285	15.53	**11.60**
	3	16	31	3	44	21	126	880	315	6764	601	7079	1031	280	21.02	49.85
	4	16	37	4	11	3	11	93	22	101	28	123	25	288	3.09	5.19
	5	16	45	4	5	0	1	10	2	5	4	7	2	279	40.03	1329.17
$E016\text{-}05m$	1	16	15	5	5	0	0	5	0	284	145	284	11	329	0.36	**0.19**
	2	16	25	5	45	3	17	699	82	5609	397	5691	423	342	4.34	**2.30**
	3	16	31	5	44	8	24	931	125	6906	379	7031	527	347	3.86	**3.12**
	4	16	40	5	18	3	3	129	10	2693	212	2703	139	336	19.86	**7.18**
	5	16	48	5	5	0	0	5	0	284	111	284	11	329	0.27	**0.16**
$E021\text{-}04m$	1	21	20	4	11	0	0	13	0	2056	352	2056	79	351	5.08	**0.98**
	2	21	29	5	-	-	-	-	-	-	2937	-	-	396	496.00	5400^l
	3	21	46	5	-	-	-	-	-	-	133	-	-	387	13.86	5400^l
	4	21	44	5	34	19	26	507	91	1169	91	1260	235	374	58.69	174.05
	5	21	49	5	4	1	0	5	0	105	46	105	13	369	0.27	**0.08**
$E021\text{-}06m$	1	21	20	6	6	0	0	6	0	243	93	243	9	423	0.20	**0.09**
	2	21	32	6	25	3	21	456	52	7896	238	7948	387	434	2.03	3.98
	3	21	43	6	22	6	11	301	28	4170	587	4198	211	432	11.30	14.83
	4	21	50	6	32	14	9	399	48	20530	368	20578	907	438	6.14	15.98
	5	21	62	6	6	0	0	6	0	243	154	243	9	423	46.74	**0.10**
$E022\text{-}04g$	1	22	21	4	4	0	0	4	0	101	41	101	1	367	0.03	**0.02**
	2	22	31	4	17	7	13	145	45	1305	126	1350	105	380	2.30	**0.95**
	3	22	37	4	6	3	6	49	12	185	52	197	15	373	1.89	**0.18**
	4	22	41	4	6	3	3	27	6	391	383	397	13	377	35.42	**3.04**
	5	22	57	5	4	1	0	5	0	47	39	47	1	389	1867.38	**0.12**
$E022\text{-}06m$	1	22	21	6	21	0	0	46	0	1043	519	1043	45	488	5.78	**0.41**
	2	22	33	6	24	3	14	283	36	1928	1010	1964	123	491	26.52	**1.50**
	3	22	40	6	30	16	30	1028	109	7478	1081	7587	513	496	36.84	**5.91**
	4	22	57	6	13	5	3	47	6	1016	311	1022	37	489	3.02	**1.26**
	5	22	56	6	21	0	0	46	0	1043	509	1043	45	488	4.74	**0.42**
$E023\text{-}03g$	1	23	22	3	3	0	0	3	0	11	30	11	1	558	0.00	0.01
	2	23	32	5	-	-	-	-	-	-	787	-	-	724	42.28	5400^l
	3	23	41	5	-	-	-	-	-	-	47	-	-	698	3.27	5400^l

Tabela 7.3: Desempenho do algoritmo Branch-and-cut para o 2L-CVRP Sequencial

Nome	Classe	n	L	K	P_{he}	P_e	P_{in}	P_{ha}	C_{pck}	C_{rot}	C_{ior}	C_{tot}	N_{bb}	Z	T_{ior}	T_s
$E023\text{-}03g$	4	23	51	5	-	-	-	-	-	-	304	-	-	714	1959.63	5400^l
	5	23	55	6	17	6	19	453	52	55	20	107	65	742	787.33	939.43
$E023\text{-}05s$	1	23	22	5	5	0	0	7	0	2	6	2	3	657	0.03	**0.02**
	2	23	29	5	-	-	-	-	-	-	705	-	-	720	18.74	5400^l
	3	23	42	5	-	-	-	-	-	-	384	-	-	730	60.88	5400^l
	4	23	51	5	-	-	-	-	-	-	37	-	-	701	18.59	5400^l
	5	23	55	5	-	-	-	-	-	-	8	-	-	721	958.64	5400^l
$E026\text{-}08m$	1	26	25	8	8	0	0	8	0	901	727	901	9	609	19.80	**1.33**
	2	26	40	8	15	1	5	149	9	4847	1199	4856	123	612	57.55	**3.72**
	3	26	61	8	7	2	2	93	6	1765	1244	1771	35	615	164.22	**2.00**
	4	26	63	8	41	8	10	1654	72	42491	1440	42563	841	626	258.20	**68.80**
	5	26	91	8	7	1	0	8	0	901	527	901	9	609	43.17	**22.47**
$E030\text{-}03g$	1	30	29	3	17	0	0	45	0	1435	6207	1435	131	524	54635.61	**0.99**
	3	30	49	6	-	-	-	-	-	-	1068	-	-	638	433.23	5400^l
	4	30	72	7	-	-	-	-	-	-	1280	-	-	738	23699.77	5400^l
$E030\text{-}04s$	1	29	29	3	4	0	0	4	0	24	-	24	1	500	0.10	**0.02**
$\boldsymbol{E031\text{-}09h}$	**1**	31	30	9	80	0	0	1955	0	327412	-	327412	4293	596	$\mathbf{86400}^l$	**1297.32**
	2	31	50	9	179	41	140	15506	1036	1860951	-	1861987	24803	605	$\mathbf{86400}^l$	**48680.60**
	3	31	56	9	67	2	1	2099	3	376710	-	376713	4863	596	$\mathbf{86400}^l$	**1625.59**
	5	31	101	9	70	10	0	1955	0	327412	-	327412	4293	596	$\mathbf{86400}^l$	**27326.53**
$E033\text{-}03n$	1	33	32	3	6	0	0	6	0	30	319	30	5	1909	6.14	**0.05**
	3	33	56	7	-	-	-	-	-	-	2716	-	-	2754	14240.75	5400^l
$\boldsymbol{E033\text{-}04g}$	**1**	33	32	4	10	0	0	16	0	1020	-	1020	13	823	$\mathbf{86400}^l$	**0.82**
	4	32	65	7	-	-	-	-	-	-	-	-	-	1166	25542.6	5400^l
$E033\text{-}05s$	1	32	32	5	10	0	0	33	0	906	-	906	23	907	17.8	**0.45**
$E036\text{-}11h$	1	36	35	11	48	0	0	485	0	118124	1962	118124	975	682	2114.28	**163.31**
	2	36	56	11	63	7	17	1173	58	153477	1857	153535	1309	682	1426.05	**310.37**
	3	36	74	11	23	7	1	152	1	56633	1589	56634	431	682	993.34	**56.63**
	4	36	93	11	96	28	67	3049	291	639759	3413	640050	5543	691	7946.74	**4480.93**
	5	36	114	11	48	0	0	485	0	118124	1637	118124	975	682	1139.86	**163.39**
$\boldsymbol{E045\text{-}04f}$	**1**	45	44	4	4	0	0	4	0	236	-	236	3	709	$\mathbf{86400}^l$	**0.29**
$\boldsymbol{E051\text{-}05e}$	**1**	51	50	5	12	0	0	18	0	16645	-	16645	107	508	$\mathbf{86400}^l$	**37.99**
$\boldsymbol{E072\text{-}04f}$	**1**	72	71	4	4	0	0	7	0	63	-	63	2	216	$\mathbf{86400}^l$	**0.20**
$\boldsymbol{E101\text{-}10c}$	**1**	101	100	10	11	0	0	37	0	14271	-	14271	5	795	$\mathbf{86400}^l$	**175.65**

7.3 Resultados Obtidos para o 2L-CVRP Irrestrito

A versão irrestrita do 2L-CVRP não leva em conta a organização dos itens nos contêineres. Quando empacotados, estes não podem se sobrepor ou ultrapassar os limites do contêiner, mas não precisam obedecer nenhuma ordenação específica. Na prática, pode-se considerar que o re-arranjo dos itens é viável durante a operação de descarregamento.

Entre os algoritmos propostos para o 2L-CVRP encontrados na literatura, quase todos abordam ambas as versões sequencial e irrestrita do problema. A exceção é justamente a única abordagem exata conhecida, apresentada por Iori, Salazar-González e Vigo em [22]. Deste modo, apresentamos então os resultados obtidos com o nosso algoritmo exato para a variante irrestrita. Como no caso das tabelas 7.2 e 7.3, definimos um limite de 5400 segundos para o *branch-and-cut*.

As tabelas 7.4 e 7.5 apresentam os resultados encontrados com a relaxação da restrição sequencial. O marcador ir é utilizado para distinguir as colunas dos resultados obtidos para o 2L-CVRP irrestrito. As colunas P^{ir}_{he} e P^{ir}_{ex} representam o número de rotas verificadas como viáveis pela heurística e pelo algoritmo exato de empacotamento, respectivamente. Em P^{ir}_{in} temos o número de rotas descobertas inviáveis pelo limitante inferior ou pelo *branch-and-bound*. Na coluna P^{ir}_{ha} podemos observar o número de vezes que rotas viáveis ou inviáveis foram encontradas na tabela de dispersão, reduzindo o esforço computacional. Ou seja, o número de vezes que rotas, já com a viabilidade de seus empacotamentos verificada, apareceram novamente durante a resolução da instância. A coluna C^{ir}_{pck} mostra o número de cortes encontrados pelos algoritmos de empacotamento. Em C^{ir}_{tot} temos o número total de cortes inseridos pelo algoritmo, somando-se os cortes de conectividade e peso e os cortes de empacotamento. As coluna N^{ir}_{bb} mostra o número total de nós explorados na árvore de busca. A solução ótima, quando encontrada, é representada por Z^{ir}. A coluna T^{ir}_{s} mostra o tempo utilizado pelo algoritmo para encontrar a solução exata. As colunas restantes foram retiradas das tabelas 7.2 e 7.3 para fim de comparação com a versão sequencial.

Para a maioria das instâncias houve uma redução no tempo computacional, quando comparando com o caso sequencial. Além disto, algumas instâncias que o algoritmo não conseguiu resolver com a restrição de ordem no tempo limite acabaram sendo resolvidas na versão irrestrita. Isto ocorreu com as instâncias $E021$-04-3, $E023$-05-2 e $E033$-03-3. Como esta versão do 2L-CVRP possui um problema de empacotamento menos restrito, isto era esperado. O curioso foi o ocorrido com a instância $E016$-$03m$-5, que já sabemos ser uma instância problemática. Esta instância foi resolvida para o caso sequencial dentro do limite de tempo, mas o mesmo não ocorre com o caso irrestrito. Buscando uma análise mais detalhada, aumentamos o tempo limite e conseguimos resolve-la em 6697.70 segundos. O motivo desta demora vem do que já conhecemos sobre a instância: ela possui uma rota com um empacotamento difícil (para nossos algoritmos de empacotamento). Consequentemente, quase todo o tempo gasto pelo algoritmo *branch-and-cut* é utilizado pelo algoritmo exato *branch-and-bound* de empacotamento. Possivelmente, o motivo da diferença significativa de tempo entre ambas as variantes vem do fato que o algoritmo de empacotamento *branch-and-bound* consegue podar ramos mais facilmente

no caso sequencial, já que um ramo é podado quando coloca-se um item que viola a restrição de ordem (ver 5.5). No caso irrestrito isto não ocorre e o algoritmo precisa explorar um número maior de nós para verificar a viabilidade do empacotamento.

A comparação das colunas C^{ir}_{pck} e C_{pck} revela uma drástica redução de 55.74%, na média, no número de cortes de empacotamento inseridos quando considerando as instâncias resolvidas em ambas as versões. Isto não é surpreendente, já que temos menos restrições quando gerando empacotamentos. Podemos observar também uma redução de 23.55% na média do número total de cortes inseridos. Observe que isto não ocorre simplesmente devido ao menor número de cortes de empacotamento. Por exemplo, considere a instância $E016\text{-}03m\text{-}2$. A versão sequencial teve 452 cortes de empacotamento inseridos, enquanto que a versão irrestrita teve 78, uma redução de 82.74%. Mas quando considerando o número total de cortes inseridos, temos 15547 e 2049 cortes inseridos para a variante sequencial e irrestrita, respectivamente. Ou seja, a redução no número total de cortes não pode ser explicada apenas com a diferença entre os cortes de empacotamento. A redução média de 23.55% possivelmente ocorre devido a forte interligação entre os subproblemas de roteamento e empacotamento do 2L-CVRP.

A versão irrestrita mostrou uma redução de custo da rota de 0.61% em média para todas as instâncias testadas. Isto é relevante do ponto de vista prático, onde uma companhia pode considerar um aumento na dificuldade nas operações de descarregamento em troca de uma redução de custos.

Tabela 7.4: Desempenho do algoritmo Branch-and-cut para o 2L-CVRP Irrestrito

Nome	Classe	P_{he}	P_{he}^{ir}	P_{ex}	P_{ex}^{ir}	P_{in}	P_{in}^{ir}	P_{ha}	P_{ha}^{ir}	C_{pck}	C_{pck}^{ir}	C_{tot}	C_{tot}^{ir}	N_{bb}	N_{bb}^{ir}	Z	Z^{ir}	T_s	T_s^{ir}
$E016\text{-}03m$	1	7	7	0	0	0	0	7	7	0	0	529	529	51	51	273	273	0.19	0.63
	2	49	16	22	12	181	12	1224	324	452	78	15547	2049	1797	341	285	273	11.60	4.80
	3	44	16	21	11	126	14	880	268	315	75	7079	2943	1031	437	280	279	49.85	20.11
	4	11	3	3	1	11	0	93	4	22	0	123	1	25	1	288	277	5.19	0.08
	5	5	-	0	-	1	-	10	-	2	-	7	-	2	-	279	-	1329.17	5400^{l}
$E016\text{-}05m$	1	5	5	0	0	0	0	5	5	0	0	284	284	11	11	329	329	0.19	0.36
	2	45	5	3	0	17	0	699	5	82	0	5691	284	423	11	342	329	2.30	0.36
	3	44	36	8	6	24	8	931	1003	125	124	7031	6256	527	447	347	347	3.12	7.72
	4	18	4	3	1	3	0	129	5	10	0	2703	284	139	11	336	329	7.18	0.45
	5	5	4	0	1	0	0	5	5	0	0	284	284	11	11	329	329	0.16	16.33
$E021\text{-}04m$	1	11	11	0	0	0	0	13	13	0	0	2056	2056	79	79	351	351	0.98	1.76
	2	-	182	-	45	-	288	-	8871	-	10245	-	12227	-	2999	-	381	5400^{l}	42.59
	3	-	125	-	62	-	217	-	14152	-	3954	-	24543	-	4691	-	387	5400^{l}	223.45
	4	34	-	19	-	26	-	5-7	-	91	-	1260	-	235	-	374	-	174.05	5400^{l}
	5	4	5	1	0	0	0	5	5	0	0	105	105	13	13	369	369	0.08	0.22
$E021\text{-}06m$	1	6	6	0	0	0	0	6	6	0	0	243	243	9	9	423	423	0.09	0.26
	2	25	5	3	1	21	0	456	6	52	0	7948	243	387	9	434	423	3.98	0.24
	3	22	4	6	2	11	0	301	6	28	0	4198	243	211	9	432	423	14.83	0.32
	4	32	29	14	7	9	4	399	395	48	43	20578	16146	907	783	438	438	15.98	54.49
	5	6	5	0	1	0	0	6	6	0	0	243	243	9	9	423	423	0.10	1.13
$E022\text{-}04g$	1	4	4	0	0	0	0	4	4	0	0	101	101	1	1	367	367	0.02	0.14
	2	17	1	7	3	13	0	145	4	45	0	1350	101	105	1	380	367	0.95	0.12
	3	6	8	3	1	6	5	49	50	12	12	197	197	15	15	373	373	0.18	0.38
	4	6	7	3	2	3	3	27	27	6	6	397	397	13	13	377	377	3.04	3.32
	5	4	5	1	0	0	0	5	5	0	0	47	47	1	1	389	389	0.12	0.11
$E022\text{-}06m$	1	21	19	0	0	0	0	46	48	0	0	1043	1043	45	45	488	488	0.41	1.05
	2	24	17	3	2	14	2	283	75	36	4	1964	1385	123	59	491	488	1.50	1.35
	3	30	15	16	6	30	4	1028	93	109	11	7587	1408	513	61	496	489	5.91	3.74
	4	13	14	5	4	3	3	47	47	6	6	1022	1022	37	37	489	489	1.26	1.13
	5	21	19	0	0	0	0	46	48	0	0	1043	1043	45	45	488	488	0.42	1.00

Tabela 7.5: Desempenho do algoritmo Branch-and-cut para o 2L-CVRP Irrestrito

Nome	Classe	P_{he}	P_{he}^{ir}	P_{ex}	P_{ex}^{ir}	P_{in}	P_{in}^{ir}	P_{ha}	P_{ha}^{ir}	C_{pck}	C_{pck}^{ir}	C_{tot}	C_{tot}^{ir}	N_{bb}	N_{bb}^{ir}	Z	Z^{ir}	T_s	T_s^{ir}
$E023\text{-}03g$	1	3	3	0	0	0	0	3	3	0	0	11	11	1	1	558	558	0.01	0.09
	3	-	37	-	9	-	37	-	2593	-	372	-	959	-	491	-	685	5400^{l}	5.28
	5	17	17	6	3	19	7	453	468	52	52	107	107	65	65	742	742	939.43	514.71
$E023\text{-}05s$	1	5	5	0	0	0	0	7	7	0	0	2	2	3	3	657	657	0.02	0.10
	2	-	49	-	3	-	53	-	2583	-	392	-	1021	-	537	-	680	5400^{l}	4.55
$E026\text{-}08m$	1	8	8	0	0	0	0	8	8	0	0	901	901	9	9	609	609	1.33	1.54
	2	15	7	1	1	5	0	149	8	9	0	4856	901	123	9	612	609	3.72	1.58
	3	7	7	2	1	2	0	93	8	6	0	1771	901	35	9	615	609	2.00	1.55
	4	41	37	8	6	10	4	1654	1798	72	77	42563	45059	841	915	626	626	68.80	127.08
	5	7	7	1	1	0	0	8	8	0	0	901	901	9	9	609	609	22.47	1.59
$E030\text{-}03g$	1	17	15	0	0	0	0	45	47	0	0	1435	1435	131	131	524	524	0.99	2.33
$E030\text{-}04s$	1	4	4	0	0	0	0	4	4	0	0	24	24	1	1	500	500	0.02	0.08
$E031\text{-}09h$	1	-	70	-	0	-	0	-	1965	-	0	-	327412	-	4293	-	596	5400^{l}	1540.39
	2	-	70	-	18	-	18	-	2714	-	81	-	376886	-	4677	-	596	5400^{l}	2188.47
	3	-	54	-	4	-	1	-	2110	-	3	-	376713	-	4863	-	596	5400^{l}	1918.63
$E033\text{-}03n$	1	6	6	0	0	0	0	6	6	0	0	30	30	5	5	1909	1909	0.05	0.14
$E033\text{-}04g$	1	10	10	0	0	0	0	16	16	0	0	1020	1020	13	13	823	823	0.82	1.18
$E033\text{-}05s$	1	10	9	0	0	0	0	33	34	0	0	906	906	23	23	907	907	0.45	0.78
$E036\text{-}11h$	1	48	44	0	0	0	0	485	489	0	0	118124	118124	975	975	682	682	163.31	190.55
	2	63	70	7	3	17	12	1173	1313	58	56	153535	203979	1309	1657	682	682	310.37	500.33
	3	23	24	7	5	1	1	152	153	1	1	56634	56634	431	431	682	682	56.63	65.93
	4	96	48	28	17	67	8	3049	951	291	56	640050	226840	5543	1807	691	687	4480.93	993.19
	5	48	44	0	0	0	0	485	489	0	0	118124	118124	975	975	682	682	163.39	190.51

7.4 Resultados Obtidos Com o Ótimo como Limitante Superior

A utilização de um bom limitante superior obtido através de um método heurístico pode ajudar a podar ramos da árvore de enumeração, consequentemente reduzindo o tempo computacional. O algoritmo proposto não possui uma heurística primal implementada. Para explorarmos o possível benefício de termos tal heurística, decidimos por uma abordagem apresentada em [3]. Selecionamos todas as instâncias que não conseguimos melhorar o tempo computacional apresentado em [22] (ver tabelas 7.2 e 7.3, resultados obtidos para o 2L-CVRP Sequencial) e alimentamos a solução ótima conhecida no nó raiz como limitante superior e executamos o algoritmo novamente.

A tabela 7.6 sumariza os resultados encontrados. As colunas com o marcador pr sinalizam os resultados obtidos com a inserção do ótimo como limitante superior. Destes, C_{tot}^{pr}, N_{bb}^{pr} e T_s^{pr} representam o número total de cortes encontrados, o número de nós explorados na árvore de busca e o tempo computacional, respectivamente. As colunas restantes foram obtidas das tabelas 7.2 e 7.3 e mantém os mesmos significados, representando os resultados sem a utilização do limitante superior no nó raiz. C_{tot} é o número total de cortes inseridos, considerando o empacotamento e roteamento. Em N_{bb} temos o número total de nós explorados e T_s nos dá o tempo computacional gasto. A coluna $\%N_{var}$ nos mostra a variação percentual entre N_{bb}^{pr} e N_{bb}. A variação entre T_s^{pr} e T_s é representada por $\%T_{var}$. Um resultado negativo significa uma redução do valor anterior.

O algoritmo apresentou uma melhora de desempenho substancial, inclusive encontrando a solução de algumas instâncias não resolvidas previamente no tempo limite. Isto ocorreu com as instâncias $E021\text{-}04m\text{-}3$, $E023\text{-}05s\text{-}2$ e $E023\text{-}05s\text{-}4$. Como não sabemos o tempo que estas instâncias levariam para ser resolvidas, vamos limitar o possível ganho por baixo: suponha que cada uma fosse resolvida em um tempo muito próximo ao limite, por exemplo 5401 segundos. Ou seja, na análise mais conservadora possível temos uma redução de tempo de 94.62% na média.

Quando considerando apenas as instâncias em que o algoritmo proposto já havia encontrado uma solução exata dentro do tempo limite, a redução do tempo computacional foi de 30.12% na média. Um ganho considerável, principalmente considerando que o crescimento desta média foi restringido pela conhecida instância $E016\text{-}03m\text{-}5$ e pelas instâncias que já possuiam uma resolução muito rápida originalmente.

Este ganho não é inesperado, mas oferece uma idéia de quanto o desempenho do algoritmo apresentado pode ser melhorado com o uso de uma heurística primal.

Tabela 7.6: Desempenho com a solução ótima como limitante superior

Name	Class	C_{tot}	C_{tot}^{pr}	N_{bb}	N_{bb}^{pr}	$\%N_{var}$	T_s	T_s^{pr}	$\%T_{var}$
$E016\text{-}03m$	3	7079	3459	1031	273	-73.52	49.85	13.72	-72.48
	4	123	76	25	21	-16.00	5.19	4.95	-4.62
	5	7	4	2	1	-50.00	1329.17	1282.49	-3.51
$E021\text{-}04m$	2	-	-	-	-	-	5400^l	5400^l	-
	3	-	15393	-	1917	(-)	5400^l	49.16	(-)
	4	1260	719	235	51	-78.30	174.05	144.05	-17.24
$E021\text{-}06m$	2	7948	4902	387	157	-59.43	3.98	3.74	-6.03
	3	4198	2125	211	97	-54.03	14.83	2.91	-80.38
	4	20578	10410	907	351	-61.3	15.98	9.26	-42.05
$E023\text{-}03g$	2	-	-	-	-	-	5400^l	5400^l	-
	3	-	-	-	-	-	5400^l	5400^l	-
	4	-	-	-	-	-	5400^l1	5400^l	-
	5	107	123	65	53	-18.46	939.43	801.51	-14.68
$E023\text{-}05s$	2	-	56692	-	23799	(-)	5400^l	695.60	(-)
	3	-	-	-	-	-	5400^l	5400^l	-
	4	-	6138	-	2899	(-)	5400^l	126.93	(-)
	5	-	-	-	-	-	5400^l	5400^l	-
$E030\text{-}03g$	3	-	-	-	-	-	5400^l	5400^l	-
	4	-	-	-	-	-	5400^l	5400^l	-
$E033\text{-}03n$	3	-	-	-	-	-	5400^l	5400^l	-
$E033\text{-}04g$	4	-	-	-	-	-	5400^l	5400^l	-

Capítulo 8

Conclusão

Nesta dissertação de mestrado apresentamos um algoritmo exato para o Problema de Roteamento de Veículos Capacitados com Restrições Bidimensionais de Carregamento. Este é um problema recente na literatura, com o primeiro trabalho sendo publicado no ano de 2007.

Testamos o algoritmo proposto com todas as instâncias já resolvidas de forma exata na literatura, além de um pequeno conjunto de instâncias que ainda não possuiam o ótimo conhecido. Ambas variantes, sequencial e irrestrita, foram consideradas para os testes computacionais. Comparamos os resultados obtidos, para o caso sequencial, com a abordagem exata de Iori, Salazar-González e Vigo [22] e observamos resultados satisfatórios. Nove instâncias da literatura foram resolvidas à otimalidade pela primeira vez. A abordagem proposta teve a característica de inserir uma quantidade consideravelmente maior de cortes, mas este aumento no esforço computacional foi compensado pela redução do tempo na maior parte dos casos. Como o caso irrestrito ainda não havia sido abordado de modo exato, testamos as instâncias utilizadas na outra variante e apresentamos várias soluções ótimas. Esperamos que estas ajudem pesquisadores desenvolvendo algoritmos heurísticos para o 2L-CVRP irrestrito a verificar e melhorar a qualidade de suas soluções.

Percebemos algumas possiveis estratégias para melhorarmos o desempenho da abordagem proposta. Como observado no capítulo 7, a adição de uma heurística que encontre bons limitantes superiores pode reduzir significativamente o tempo computacional utilizado. A utilização de uma estratégia de ramificação customizada também poderia oferecer benefícios consideráveis, sendo ideal considerar características do 2L-CVRP, e não apenas do CVRP. A utilização de desigualdades para o problema de carregamento e de novas desigualdades para o CVRP também podem apresentar ganhos interessantes.

Como detalhado no capítulo 4, utilizamos desigualdades válidas para o CVRP no arcabouço do *branch-and-cut*. Procuramos fortalecer estas desigualdades para o 2L-CVRP com uma simples adaptação, mas esperamos que futuros estudos poliedrais encontrem desigualdades específicas para o 2L-CVRP que apresentem-se mais fortes do que as usadas.

Observamos que o 2L-CVRP combina os problemas de roteamento e empacotamento de forma que os dois subproblemas geralmente encontram-se fortemente ligados. A inserção de desigualdades de roteamento pode mudar a configuração do problema de modo que o empacota-

mento encontrado torne-se mais fácil ou mais difícil. O mesmo ocorre na situação oposta, onde um corte de empacotamento pode influenciar consideravelmente o problema de roteamento. Consequentemente, uma maior consistência poderia ser obtida com a utilização de cortes específicos para o 2L-CVRP.

Finalmente, a partir deste trabalho geramos o artigo de título *A Branch-and-Cut Approach for the Vehicle Routing Problem with Two-dimensional Loading Constraints* [35] que foi aceito e apresentado no XLI Simpósio Brasileiro de Pesquisa Operacional.

Referências Bibliográficas

[1] J.R Araque. Lots of combs of different sizes for vehicle routing. *Discussion paper, Center for Operations Research and Econometrics, Catholic University of Louvain, Belgium*, 1990.

[2] P. Augerat. Approche polyèdrale dy problème de tournées de véhicules. *Ph.D. thesis, Institut National Polytechnique de Grenoble, France*, 1995.

[3] J. Lysgaard e A. Lechtford e R. Eglese. A new branch-and-cut algorithm for the capacitated vrp. *Mathematical Programming*, 100(2):423–445, 2003. `http://www.hha.dk/~lys/`.

[4] N. Christofides e A. Mingozzi e P. Toth. Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxation. *Mathematical Programming*, 10:255–280, 1981.

[5] N. Christofides e A. Mingozzi e P. Toth. State space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[6] A.H. Land e A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[7] E. Zachariadis e C. Tarantilis e C. Kiranoudis. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195(3):729–743, 2009.

[8] A.D. O'Connor e C.A. De Wald. A sequential deletion algorithm for the design of optimal transportation networks. *Bulletin of the Operations Research Society of America*, 18(1), 1970.

[9] S. Eilon e C.D.T. Watson-Gandy e N. Christofides. Distribution management: Mathematical modelling and practical analysis. 1971.

[10] S. Martello e D. Pisinger e D. Vigo. The three-dimensional bin packing problem. `http://www.diku.dk/~pisinger/`.

[11] S. Martello e D. Pisinger e D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000. `http://www.diku.dk/~pisinger/`.

[12] R. Baldacci e E. Hadjiconstatinou e A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52:723–738, 2004.

[13] B.S. Baker e E.G. Coffman e R.L. Rivest. Orthogonal packings in two-dimensions. *SIAM Journal on Computing*, 9:846–855, 1980.

[14] G. Cornuéjols e F. Harche. Polyhedral study of the capacitated vehicle routing. *Mathematical Programming*, 60:21–52, 1993.

[15] A. De Vitis e F. Harche e G. Rinaldi. Generalized capacity inequalities for vehicle routing problems. *In preparation*.

[16] J.R. Araque e G. Kudva e T.L. Morin e J.F. Pekni. A branch-and-cut algorithm for vehicle routing problems. *Annals of Operations Research*, 50:37–59, 1994.

[17] M. Gendreau e G. Laporte e J.Y. Potvin. Metaheuristics for the vehicle routing problem. *Technical Report*, Les Cahiers du GERAD, Montreal, Canada, G-98-52, 1998.

[18] D. Naddef e G. Rinaldi. Branch-and-cut algorithms for the capacitated vrp. toth p. vigo d., eds. the vehicle routing problem. *SIAM Monographs on Discrete Mathematics and Applications*, 9:53–84, 2002.

[19] R. Fukasawa e H. Longo e J. Lysgaard e M. P. Aragao e M. Reis e E. Uchoa e R. F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106:491–511, 2006.

[20] G.B. Dantzig e J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[21] P. Augerat e J. M. Belenguer e E. Benavent e A. Corberán e D. Naddef e G. Rinaldi. Computational results with branch and cut code for the capacitated vehicle routing problem. *Technical Report 1 RR949-M, ARTEMIS-IMAG, Grenoble France*, 1995.

[22] M. Iori e J. Salazar-González e D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41(2):253–264, 2007.

[23] N. Wilson e J. Sussman. Implementation of computer algorithms for the dial-a-bus system. *Bulletin of the Operations Research Society of America*, 19(1), 1971.

[24] P. Augerat e J.M. Belenguer e E. Benavent e A. Corberán e D. Naddef. Separating capacity constraints in the cvrp using tabu search. *European Journal of Operations Research*, 106:546–557, 1998.

[25] G. Clarke e J.W. Wright. Scheduling of vehicle from a depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[26] G. Fuellerer e K. Doernera e R. Hartla e M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers and Operations Research*, 36(3):655–673, 2009.

[27] J.R. Araque e L.A. Hall e T.L. Magnanti. Capacitated trees, capacitated routing and associated polyhedra. *Discussion paper, Center for Operations Research and Econometrics, Catholic University of Louvain, Belgium*, 1990.

[28] M. Gendreau e M. Iori e G. Laporte e S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1):4–18, 2008.

[29] M.W. Padberg e M.R. Rao. Odd minimum cut-sets and b-matchings. *Mathematical Operations Research*, 7:67–80, 1982.

[30] T. McCormick e M.R. Rao e G. Rinaldi. Easy and difficult objective functions for max cut. *Mathematical Programming*, 94(2-3):459–466, 2000.

[31] M. Grötschel e M.W. Padberg. On the symmetric travelling salesman problem. *Mathematical Programming*, 16:265–280, 1979.

[32] M. Grötschel e M.W. Padberg. On the symmetric travelling salesman problem ii: lifting theorems and facets. *Mathematical Programming*, 94:281–302, 1979.

[33] R. Baldacci e N. Christofides e A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

[34] A. Mingozzi e N. Christofides e E. Hadjiconstantinou. An exact algorithm for the vehicle routing problem based on the set partitioning formulation. *Technical Report*, University of Bologna, 1994.

[35] B.L.P Azevedo e P. Hokama e F.K. Miyazawa e E.C. Xavier. A branch-and-cut approach for the vehicle routing problem with two-dimensional loading constraints. *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional*, pages 1–12, 2009.

[36] S. Martello e P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[37] G.B. Dantzig e R. Fulkerson e S. Johnson. Solutions of large-scale travelling salesman problem. *Operations Research*, 2:393–410, 1954.

[38] D.H. Marks e R. Stricker. Routing for public service vehicles. *ASCE Journal of the Urban Planning and Development Division*, 97:165–178, 1970.

[39] A. N. Letchford e R. W. Eglese e J. Lysgaard. Multistars, partial multistars and the capacitated vehicle routing problem. *Mathematical Programming*, 94(1):21–40, 2002.

[40] M.Dell'Amico e S. Martello. Optimum scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7(2):191–200, 1995.

[41] B.L. Golden e T.L. Magnanti e H.Q. Nguyan. Implementing vehicle routing algorithms. *Networks*, 7(2):113–148, 1972.

[42] U. Blasum e W. Hochstättler. Application of the branch-and-cut method to the vehicle routing problem. *Technical report, Universität zu Köln*, 2002.

[43] G. Laporte e Y. Nobert. Comb inequalities for the vehicle routing probleme. *Methods of Operations Research*, 51:271–276, 1984.

[44] M. L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42:626–642, 1994.

[45] M. Iori. Instâncias para o 2l-cvrp. `http://www.or.deis.unibo.it/`.

[46] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[47] G. Laporte e W. Nobert. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147–184, 1987.

[48] A. Levin. Scheduling and fleet routing models for transportation systems. *Transportation Science*, 5(3):232–256, 1971.

[49] J.C. Liebman. Mathematical models for solid waste collection and disposal. *Bulletin of the Operations Research Society of America*, 18(2), 1970.

[50] Dash Optimization Ltd. Xpress-optimizer reference manual. *Release 18*, 2007.

[51] J. Lysgaard. Cvrpsep. `http://www.hha.dk/~lys/`.

[52] D. L. Miller. A matching based exact algorithm for capacitated vehicle routing problems. *ORSA J. Comput.*, 7:1–9, 1995.

[53] Y. Pochet. New valid inequalities for the vehicle routing problem. *In preparation*.

[54] G. Scheithauer. Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operativa*, 27(83):3–34, 1998.

[55] M. Solomon. Vehicle routing and scheduling with time windows constraints: Models and algorithms. *Technical Report*, 1983.