Universidade Estadual de Campinas
Instituto de Computação

Miguel Angel Marfurt Alarcon

Algoritmos de aproximação para problemas de estoque e roteirização

CAMPINAS
2021

Miguel Angel Marfurt Alarcon

Algoritmos de aproximação para problemas de estoque e roteirização

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Lehilton Lelis Chaves Pedrosa

Este exemplar corresponde à versão final da Dissertação defendida por Miguel Angel Marfurt Alarcon e orientada pelo Prof. Dr. Lehilton Lelis Chaves Pedrosa.

CAMPINAS
2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Marfurt Alarcon, Miguel Angel, 1996-
M336a Algoritmos de aproximação para problemas de estoque e roteirização / Miguel Angel Marfurt Alarcon. – Campinas, SP : [s.n.], 2021.

Orientador: Lehilton Lelis Chaves Pedrosa.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Algoritmos de aproximação. 2. Problema de roteamento de veículos. 3. Cadeia de suprimentos. 4. Programação linear. 5. Otimização combinatória. I. Pedrosa, Lehilton Lelis Chaves, 1985-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Approximation algorithms for Inventory Routing Problems
Palavras-chave em inglês:
Approximation algorithms
Vehicle routing problem
Supply chains
Linear programming
Combinatorial optimization
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Lehilton Lelis Chaves Pedrosa [Orientador]
Mário César San Felice
Fábio Luiz Usberti
Data de defesa: 18-08-2021
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-0106-1588
- Currículo Lattes do autor: http://lattes.cnpq.br/5170377067741812

Universidade Estadual de Campinas
Instituto de Computação

Miguel Angel Marfurt Alarcon

Algoritmos de aproximação para problemas de estoque e roteirização

Banca Examinadora:

- Prof. Dr. Lehilton Lelis Chaves Pedrosa
  Instituto de Computação - UNICAMP
- Prof. Dr. Mário César San Felice
  Departamento de Computação - UFSCar
- Prof. Dr. Fábio Luiz Usberti
  Instituto de Computação - UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 18 de agosto de 2021

Agradecimentos

Agradeço aos meus pais, Jorge e Lucy, e a minha irmã Domenica, que sempre cuidaram de mim e me apoiaram nas minhas decisões profissionais, acadêmicas e pessoais.

Ao meu orientador, Prof. Doutor Lehilton, pelos ensinamentos desde o meu período na graduação, sendo grande influenciador na escolha da minha área de atuação.

A minha namorada, Ana, que me acompanhou durante todo o mestrado e por todos os momentos felizes que passamos juntos.

A todos os meus amigos, em especial, ao Renato, Anderson e Douglas, que por sua vez, tornaram a graduação e o trabalho mais divertidos.

A todos os alunos, funcionários, docentes e a Unicamp como um todo, por possibilitar diversas experiências que impulsionaram meu desenvolvimento ao decorrer destes anos.

Resumo

Algoritmos de aproximação são algoritmos polinomiais para problemas de otimização que produzem soluções com uma certa garantia de qualidade. Para muitos problemas relevantes, não se conhecem algoritmos exatos eficientes e muitos deles são NP-difíceis. Desse modo, algoritmos de aproximação são uma alternativa para encontrar soluções boas para esses problemas. Neste trabalho, investigamos problemas que integram questões de inventário e roteamento. Mais especificamente, estudamos o *Inventory Routing Problem* (Problema de Estoque e Roteirização, IRP). Dado um ou mais depósitos, uma frota de veículos, um conjunto de clientes e um plano de horizonte de demandas de T períodos para cada cliente, o objetivo é formar entregas de forma que os veículos entreguem todas as demandas dos clientes a tempo, minimizando a soma dos custos de transporte e dos possíveis custos de armazenamento.

Nossa contribuição é dividida em duas partes. Primeiro, estudamos o *Inventory Access Problem* (IAP), uma versão particular do IRP em que um único cliente enfrenta demandas em um horizonte de planejamento discreto e o objetivo é encontrar uma política de abastecimento que minimize a soma dos custos de transporte e armazenamento. Embora a versão não capacitada seja polinomial, apenas uma 3-aproximação é conhecida para o caso capacitado. Nós melhoramos esse fator para 2,619 e, como resultado direto, também diminuímos o melhor fator conhecido para o problema chamado *Star Inventory Routing Problem with Facility Location* (SIRPFL), que é uma extensão do IAP com vários depósitos e clientes e cujas soluções correspondem a rotas em formato de estrelas. Além disso, estudamos os casos capacitados do IAP e do SIRPFL em que todos os itens de uma mesma demanda devem ser entregues na mesma viagem e diminuímos os melhores fatores conhecidos também para essas versões.

Em seguida, estudamos o *Tree Joint Replenishment Problem* (*Tree* JRP), outra versão particular do IRP em que um conjunto de clientes enfrenta demandas diárias por itens em um horizonte de planejamento discreto. As demandas são satisfeitas por itens já mantidos em estoque, que é reabastecido a partir de pedidos para um depósito que servem um subconjunto de clientes simultaneamente. No *Tree* JRP, a cadeia de abastecimento é representada por uma árvore cujas folhas são os clientes e o custo de um pedido para um subconjunto de clientes é determinado pelo custo de uma subárvore mínima que os conecta à raiz. O objetivo é decidir quando fazer os pedidos e quais clientes farão parte de cada pedido, de modo que os custos totais de transporte e armazenamento sejam minimizados. Neste trabalho, começamos o estudo da variante cujos pedidos têm capacidade limitada e fornecemos uma 5-aproximação baseada em arredondamento de PL. Destacamos que essa é a primeira aproximação tanto para o caso em que as demandas podem ser divididas em várias viagens, quanto para o caso em que a demanda de um período não pode ser dividida.

Abstract

Approximation algorithms are polynomial algorithms for optimization problems that produce solutions with a certain quality assurance. For many interesting optimization problems, no exact efficient algorithms are known and many of them are NP-hard. Therefore, approximation algorithms are an alternative to find good solutions for these problems. In this work, we investigate problems that integrate inventory and routing decisions. More specifically, we study the Inventory Routing Problem (IRP). Given one or more depots, a fleet of vehicles, a set of customers, and a demand planning horizon of T periods for each customer, the objective is to make trips in a way that the vehicles deliver all customer demands on time, minimizing the sum of transportation and storage costs.

Our contribution is divided into two parts. First, we study the Inventory Access Problem (IAP), a particular version of the IRP where a single client faces demands in a discrete planning horizon and the goal is to find a supply policy that minimizes the sum of transportation and storage costs. Although the uncapacitated version is polynomial, only a 3-approximation is known for the capacitated case. We improved this factor to 2.619 and, as a direct result, we also improved the best factor for the problem called Star Inventory Routing Problem with Facility Location (SIRPFL), which is an extension of the IAP with many depots and customers and whose solutions have the format of a star. Moreover, we study the capacitated cases of IAP and SIRPFL in which all items of a single demand must be delivered by the same trip, and we improved the best-known factors for these versions as well.

Next, we study the Tree Joint Replenishment Problem (Tree JRP), another particular version of the IRP where a set of customers faces daily demands for items in a discrete planning horizon. The demands are satisfied by items already held in stock and replenished by joint orders for a depot that serve a subset of customers simultaneously. In the Tree JRP, the supply chain is represented by a tree whose leaves are the customers, and the cost of an order for a subset of customers is determined by the cost of a minimum subtree that connects them to the root. The goal is to decide when to make the orders and which customers will be part of each order, so the total ordering and storage costs are minimized. In this work, we start the study of the variant where the orders have limited capacity, and provide a 5-approximation that is based on the LP rounding technique. We emphasize that this is the first approximation for the splittable version, when a demand can be divided into multiple trips, as well as for the unsplittable version, when one period's demand must be delivered in a single trip.

Sumário

Capítulo 1

Introdução

Algoritmos de aproximação são algoritmos com complexidade polinomial para problemas de otimização combinatória que devolvem uma solução com valor dentro de um fator da solução ótima. Em particular, algoritmos de aproximação são muito utilizados para lidar com problemas NP-difíceis, já que eles não podem ser resolvidos de forma exata por algoritmos polinomiais, a não ser que P = NP. A ideia é desenvolver algoritmos viáveis para esses problemas considerando o tempo de execução e a memória utilizada, mesmo que a qualidade da solução obtida seja comprometida. Encontrar aproximações boas para problemas NP-difíceis pode ser trabalhoso; contudo, algumas técnicas e estratégias gerais já foram desenvolvidas. Neste trabalho, estamos interessados principalmente na estratégia que consiste em formular o problema como um programa linear inteiro ou misto; obter e resolver um programa linear relaxado; encontrar uma solução inteira para o problema original com custo limitado por um fator do valor da solução do programa linear.

Problemas de inventário e de roteamento são observados em diversos setores da indústria, uma vez que gerenciamento de estoques e planejamento de entregas (domínios do âmbito de logística) estão presentes no dia-a-dia de médias e grandes empresas e, em geral, contribuem com uma parcela considerável dos custos operacionais. Dado que ambos os tipos de problemas são muito relacionados, modelos unificados são necessários a fim de minimizar os custos do processo como um todo [63]. Contudo, muitos desses problemas já são NP-difíceis mesmo quando vistos individualmente. Por exemplo, se considerarmos apenas decisões de inventário, temos o *Lot-Sizing Problem* [68] ou o *Joint Replenishment Problem* [38] e, se considerarmos apenas decisões de roteamento, temos o *Facility Location Problem* [49] ou o *Vehicle Routing Problem* [34], entre outros. Este tipo de complicação faz com que encontrar soluções exatas para os problemas unificados seja impraticável, sendo necessário resolvê-los muitas vezes com auxílio de heurísticas ou algoritmos de aproximação.

Os denominados *Inventory Routing Problems* [18] são exemplos de problemas que integram variáveis de estoque e roteamento. Nesses tipos de problemas, dado um ou mais depósitos e uma frota de veículos, deve-se encontrar a melhor política de entregas de forma a servir um conjunto de clientes, que, por sua vez, se deparam com um plano de horizonte de demandas e custos de armazenamento associados. Atualmente, não são conhecidos algoritmos de aproximação com fator constante para as versões mais gerais do IRP, que consideram os custos das entregas a partir de uma função do conjunto de

clientes envolvidos na entrega, e não se sabe se essas versões do IRP admitem aproximação com fator constante. Entre os resultados conhecidos, temos aproximações com fatores sub-logarítmicos para a versão do IRP que considera um único depósito, vários clientes e cujas funções de custo de entrega são submodulares ou equivalentes ao custo de um rota de comprimento mínimo do conjunto de clientes atendidos [59, 23]. Por causa disso, estudos recentes voltaram-se para casos particulares do IRP, logrando algoritmos de aproximação com fatores constantes para essas versões.

Uma dessas simplificações consiste em considerar o roteamento direto, ou seja, todas as viagens são realizadas diretamente do depósito até o cliente, fornecendo rotas em formato de estrela. Essa simplificação corresponde ao *Star Inventory Routing Problem with Facility Location* (SIRPFL), uma versão de IRP com vários depósitos e clientes e que tem por objetivo minimizar custos de abertura de depósitos, armazenamento e roteamento. O SIRPFL foi introduzido recentemente por Jiao e Ravi [44], que forneceram os primeiros algoritmos com aproximações constantes, e por Byrka e Lewandoski [25], que melhoraram esses fatores. Em ambos os trabalhos, foram abordadas 3 versões para o SIRPFL, diferenciadas pelo fator capacidade de entrega, são elas: *Uncapacitated SIRPFL* (versão com entregas sem limite de capacidade), *Capacitated Splittable SIRPFL* (versão com capacidades e possibilidade de dividir a demanda de um cliente em um período utilizando várias entregas) e *Capacitated Unsplittable SIRPFL* (versão com capacidades e necessidade de entregar a demanda de um cliente em um período utilizando uma única entrega). Os autores citados anteriormente observaram que algoritmos para a versão particular do SIRPFL com apenas um depósito e um cliente, chamada de *Inventory Access Problem* (IAP), podem ser utilizados como sub-rotinas para as versões mais gerais do problema. Atualmente, os algoritmos para o SIRPFL com os melhores fatores de aproximação conhecidos são baseados nessa estratégia.

Outra simplificação de IRP modela a cadeia de fornecimento como uma árvore, de forma que a raiz represente o depósito e que as folhas representem os clientes. Desse modo, o custo de roteamento pode ser estimado pelo custo de conectar a raiz a um subconjunto de folhas dessa árvore, dando origem ao chamado *Tree Joint Replenishment Problem* (*Tree* JRP). Esse problema admite uma 3-aproximação baseada em arredondamento de programa linear (PL) [30] e o melhor algoritmo de aproximação conhecido tem fator 2 [60]. Uma premissa do *Tree* JRP é que se pressupõe um único veículo com capacidade de carregamento ilimitada, o que não é realista para determinadas aplicações.

Neste trabalho, realizamos uma revisão bibliográfica do estado da arte do *Inventory Routing Problem* sob o aspecto de algoritmos de aproximação. Entre as nossas contribuições, está um algoritmo de aproximação para o IAP que melhora o menor fator conhecido. Como consequência, também melhoramos o fator de aproximação de duas versões capacitadas do SIRPFL. Além disso, desenvolvemos um algoritmo de aproximação com fator constante para uma generalização do *Tree* JRP que considera capacidades de carregamento. De acordo com nossa pesquisa, esse é o primeiro trabalho a obter aproximações para versões capacitadas do *Tree* JRP ou do JRP tradicional.

No Capítulo 2, fornecemos o ferramental teórico básico necessário e técnicas de algoritmos de aproximação utilizados neste trabalho. Para isso, definimos formalmente o que é um algoritmo de aproximação, enunciamos alguns conceitos básicos da área e, em

seguida, apresentamos as principais técnicas de programação linear e de algoritmos de aproximação utilizadas na literatura. No final desse capítulo, também fornecemos um exemplo de algoritmo de aproximação.

No Capítulo 3, definimos todos os problemas relacionados ao IRP abordados neste trabalho, mais especificamente, o problema genérico IRP, o SIRPFL e versões particulares como o IAP e o *Tree* JRP. Nesse mesmo capítulo, também introduzimos o *Capacitated Tree JRP*, uma generalização do *Tree* JRP com capacidades de carregamento nas entregas e suas variantes, o *Capacitated Splittable Tree JRP* e o *Capacitated Unsplittable Tree JRP* (versões que se diferem pela possibilidade ou não de dividir a demanda de um cliente em um período em entregas diferentes). No final do capítulo, também enumeramos alguns problemas relacionados ao IRP encontrados na literatura que consideramos relevantes para o estudo do problema, desde casos particulares até generalizações e extensões.

No Capítulo 4, damos uma visão geral do IRP na literatura para algoritmos exatos e heurísticas e, como foco principal, apresentamos uma revisão bibliográfica detalhada do IRP na área de algoritmos de aproximação, desde os primeiros algoritmos de aproximação encontrados na literatura até o estado da arte de cada variante existente.

No Capítulo 5, aprimoramos os algoritmos de aproximação das versões capacitadas do IAP desenvolvidos por Jiao e Ravi [44], mais especificamente, o *Capacitated Splittable* IAP e o *Capacitated Unsplittable* IAP. No início desse capítulo, realizamos uma revisão dos algoritmos de aproximação de Jiao e Ravi e de alguns resultados técnicos do SIRPFL realizados por Byrka e Lewandoski [25]. Em seguida, apresentamos uma melhoria no algoritmo de aproximação do *Capacitated Splittable* IAP que possibilitou a redução do fator de aproximação do algoritmo de 3 para 2,619 e, como consequência direta, do fator de aproximação do *Capacitated Unsplittable IAP* de 6 para 4,562. No final desse capítulo também observamos que, de acordo com os resultados recentes de Byrka e Lewandoski, as nossas melhorias realizadas para os algoritmos de aproximação do IAP reduzem de forma direta os melhores fatores de aproximação do *Capacitated Splittable* SIRPFL e do *Capacitated Unsplittable* SIRPFL. Os resultados deste capítulo serão publicados na revista *Operations Research Letters* [1].

No Capítulo 6, desenvolvemos os primeiros algoritmos de aproximação para o *Capacitated Tree* JRP utilizando a técnica de arredondamento de PL. No início do capítulo, fazemos uma revisão da 3-aproximação para o *Tree* JRP de Cheung et al. [30]. Em seguida, baseando-nos nesse algoritmo, apresentamos uma 6-aproximação para o *Capacitated Splittable Tree* JRP e uma 5-aproximação para o *Capacitated Unsplittable Tree* JRP, sendo a segunda aproximação também válida para o *Capacitated Splittable Tree* JRP. A ideia geral dos algoritmos consiste de duas etapas. Na primeira, determinamos em que períodos serão agendadas entregas e quais clientes participarão de entregas nos períodos. Para isso, utilizamos uma sub-rotina de arredondamento de Cheung et al., mas adaptada para o *Capacitated Tree* JRP. Na segunda, para cada período, determinamos quantas entregas e quais clientes serão incluídos em cada entrega, utilizando um algoritmo de roteamento para cada versão do *Capacitated Tree* JRP explorada. Um resumo dos resultados deste capítulo foi publicado nos Anais do VI Encontro de Teoria da Computação [5] e pretendemos submeter uma versão completa a uma revista.

Finalmente, no Capítulo 7, apresentamos conclusões gerais, perguntas em aberto e

possíveis trabalhos futuros no estudo de algoritmos de aproximação voltados para o IRP, particularmente aqueles para versões com capacidades. Os resultados contidos nesta dissertação estão resumidos na Tabela 1.1.

Tabela 1.1: Resumo dos resultados realizados nesta pesquisa

Problema	Resultado anterior	Nosso resultado
Capacitated Splittable IAP	3 [44]	2,619
Capacitated Unsplittable IAP	6 [44]	4,562
Capacitated Splittable SIRPFL	3,236 [25]	2,905
Capacitated Unsplittable SIRPFL	6,029 [25]	4,649
Capacitated Splittable Tree JRP	-	5
Capacitated Unsplittable Tree JRP	-	5

Capítulo 2

Métodos de otimização combinatória

2.1 Algoritmo de aproximação

As definições a seguir foram adaptadas do livro de Williamson e Shmoys [69] e representam alguns conceitos básicos sobre algoritmos de aproximação.

Considere um problema de otimização $\mathcal{P}$ e seja $\mathcal{A}$ um algoritmo para $\mathcal{P}$. Dada uma instância $\mathcal{I}$ do problema $\mathcal{P}$, denote por $\mathcal{A}(\mathcal{I})$ o valor da solução obtida por $\mathcal{A}$ para $\mathcal{I}$ e por $OPT(\mathcal{I})$ o valor da solução ótima de $\mathcal{I}$.

Definição 1. *Um algoritmo é uma α-aproximação para um problema de otimização se ele for um algoritmo polinomial tal que, para toda instância do problema, ele produz uma solução cujo valor está dentro de um fator α do valor da solução ótima. Chamamos de α a performance garantida do algoritmo, também chamada de fator de aproximação ou razão de aproximação.*

Em outras palavras, $\mathcal{A}$ é uma α-aproximação para $\mathcal{P}$ se $\mathcal{A}$ é um algoritmo polinomial e se, para toda instância $\mathcal{I}$ do problema $\mathcal{P}$, temos que $\mathcal{A}(\mathcal{I}) \leq \alpha \cdot OPT(\mathcal{I})$ para um problema de minimização ou $\mathcal{A}(\mathcal{I}) \geq \alpha \cdot OPT(\mathcal{I})$ para um problema de maximização. Note que, de acordo com esta convenção, temos que $\alpha \geq 1$ para problemas de minimização e que $\alpha \leq 1$ para problemas de maximização.

Também é possível estender a definição de algoritmos de aproximação para algoritmos aleatorizados.

Definição 2. *Um algoritmo é uma α-aproximação aleatorizada se ele for um algoritmo polinomial tal que a esperança do valor da solução esteja dentro de um fator α do valor da solução ótima.*

Neste caso, $\mathcal{A}$ é uma α-aproximação aleatorizada para $\mathcal{P}$ se $\mathcal{A}$ é algoritmo polinomial e se $\mathbb{E}[\mathcal{A}(\mathcal{I})] \leq \alpha \cdot OPT(\mathcal{I})$ para um problema de minimização ou $\mathbb{E}[\mathcal{A}(\mathcal{I})] \geq \alpha \cdot OPT(\mathcal{I})$ para problema de maximização, sendo $\mathbb{E}[\mathcal{A}(\mathcal{I})]$ a esperança do custo da solução de $\mathcal{A}$ para $\mathcal{I}$.

Finalmente, temos os esquemas de aproximação polinomiais.

Definição 3. *Um esquema de aproximação de tempo polinomial (PTAS) é uma família de algoritmos $\{\mathcal{A}_\epsilon\}_{\epsilon>0}$ tal que, para cada $\epsilon > 0$, $\mathcal{A}_\epsilon$ é uma $(1+\epsilon)$-aproximação para problemas de minimização ou uma $(1-\epsilon)$-aproximação para problemas de maximização.*

2.2 Programação linear e algoritmos de aproximação

Os seguintes conceitos e resultados de programação linear foram extraídos do livro de programação linear do Bazaraa et al. [16].

Programação linear é uma técnica para formulação de problemas de otimização combinatória onde a *instância* é formada por um vetor $c \in \mathbb{Q}^n$, um vetor $b \in \mathbb{Q}^m$ e uma matriz $A = (a_{ij}) \in \mathbb{Q}^{m \times n}$ cuja *tarefa* é representada pela decisão de devolver exatamente um dos seguintes resultados possíveis:

1. encontrar um vetor $x \in \mathbb{Q}_+^n$ tal que $Ax \geq b$ e $c^T x$ é mínimo, ou
2. verificar que $\{x \in \mathbb{Q}_+^n : Ax \geq b\}$ é vazio, ou
3. verificar que para todo $\alpha \in \mathbb{R}$ existe um $x \in \mathbb{Q}^n$ com $Ax \geq b$ e $c^T x < \alpha$.

O vetor x é chamado de variável do problema. Qualquer x que satisfaça as restrições é chamado de solução viável e, se existir pelo menos algum x no problema, o problema é dito viável; caso contrário, é dito inviável. Uma instância do problema é chamada pelo nome de programa linear. Tanto o problema quanto uma instância do problema são frequentemente abreviados por PL. Programação Linear é um problema bem resolvido e pertence à classe P.

Também é possível definir um programa linear com variáveis inteiras. Na programação linear inteira (PLI), uma *instância* é composta de um vetor $c \in \mathbb{Q}^n$, um vetor $b \in \mathbb{Q}^m$ e uma matriz $A = (a_{ij}) \in \mathbb{Q}^{m \times n}$. A *tarefa* é similar, encontrar um vetor $x \in \mathbb{Z}^n$ tal que $Ax \geq b$ e $c^T x$ é mínimo, ou verificar que $\{x \in \mathbb{Z}^n : Ax \geq b\}$ é vazio, ou verificar que a solução é ilimitada.

Diferentemente de PL, PLI é um problema NP-difícil [48]. Quando a instância é composta tanto por variáveis racionais arbitrárias quanto por variáveis inteiras, o problema é denominado Programação Linear Mista. Ao transformar as variáveis inteiras de um problema linear inteiro ou misto em variáveis racionais, o problema resultante é um programa linear chamado de relaxação linear do problema original. Note que a solução ótima da relaxação linear não necessariamente corresponde a uma solução viável do problema original. Para um problema de minimização, o valor ótimo da relaxação linear de um problema linear inteiro ou misto é um limitante inferior para o valor da solução do problema original e, para o caso de maximização, um limitante superior.

Um conceito muito importante em programação linear é o conceito de dualidade. Considere o programa linear abaixo, que está escrito na forma canônica para um problema de minimização, i.e, todas as variáveis do problema são não negativas e todas as restrições são do tipo maior ou igual. Denominamos esse programa de problema primal:

$$\begin{array}{lll} P: & \min & cx \\ & s.a & Ax \geq b \\ & & x \geq 0 \end{array} \tag{2.1}$$

O seguinte programa linear é especialmente relacionado com o problema acima e está escrito na forma canônica para problema de maximização. Esse programa é chamado de

problema dual:

$$\begin{array}{lll} D: & \max & wb \\ & s.a & wA \leq c \\ & & w \geq 0 \end{array} \tag{2.2}$$

Sejam x_0 e w_0 quaisquer soluções viáveis para os problemas primal e dual, respectivamente. A seguinte propriedade sempre é válida: $cx_0 \geq w_0Ax_0 \geq w_0b$. Essa propriedade é conhecida como dualidade fraca e quer dizer, em outras palavras, que o valor de qualquer solução viável do problema dual é um limitante inferior para o valor da solução ótima do problema primal correspondente. Além dessa propriedade, outros dois resultados muito importantes existem sobre problemas primais e duais, descritos a seguir:

Teorema 1 (Teorema fundamental da dualidade)**.** *Com relação aos problemas primal e dual, apenas um dos seguintes itens é verdadeiro:*

1. *Ambos os problemas possuem solução ótima x^* e w^* com $cx^* = w^*b$.*
2. *Um problema possui valor da função-objetivo ilimitado e o outro problema é inviável.*
3. *Ambos problemas são inviáveis.*

Teorema 2 (Teorema das folgas complementares)**.** *Sejam x^* e w^* quaisquer soluções viáveis dos problemas primal e dual na forma canônica. As soluções correspondentes são ótimas se, e somente se,*

1. $(c_j - w^*a_j)x_j^* = 0$ *para* $j = 1, ..., n$ *e*
2. $w_i^*(a^ix^* - b_i) = 0$ *para* $i = 1, ..., m$.

Diversos algoritmos de aproximação são baseados em técnicas de programação linear, visto que grande parte dos problemas de otimização combinatória são enunciados na forma de programação linear inteira ou mista. Logo, os resultados acima são frequentemente utilizados na análise e projeto de algoritmos de aproximação para problemas de otimização combinatória.

Dentre algumas das principais técnicas de programação linear utilizadas para desenvolvimento de algoritmos de aproximação temos o arredondamento de PL, que consiste em utilizar a solução ótima da relaxação linear do problema linear inteiro ou misto do problema estudado e de alguma forma arredondar os valores das variáveis fracionárias para valores inteiros. Assim, o caminho para se encontrar um algoritmo de aproximação é descrever um processo de arredondamento que seja polinomial em função da entrada de tal modo que o custo da nova solução obtida seja no máximo um fator da solução ótima da relaxação, que é um limitante inferior para a solução ótima do problema original.

A técnica primal-dual consiste em analisar as variáveis de ambos problemas primal e dual da solução relaxada e fazer uma sequência de decisões (muitas vezes a partir de um critério guloso) até encontrar uma solução viável do problema original limitado por algum fator das soluções do primal e do dual. Uma técnica que utiliza a perspectiva primal-dual, por exemplo, é a técnica do *dual-fitting*. Considerando-se um problema de minimização,

ela consiste em iniciar o procedimento com uma solução dual w e uma solução primal integral inviável x e iterativamente fazer atualizações nas soluções primal e dual até que a solução primal se torne viável e de forma a garantir que o valor das duas soluções sejam iguais. Uma vez que a solução primal integral viável for encontrada, a análise consiste em encontrar um fator $\gamma \geq 1$ de forma que w/γ seja viável. Desse modo, utilizando o teorema da dualidade, o custo da solução primal integral obtida é limitado por:

$$cx = bw = \gamma \cdot b\frac{w}{\gamma} \leq \gamma \cdot b\hat{w} = \gamma \cdot c\hat{x},$$

onde $\hat{w}$ e $\hat{x}$ são soluções ótimas dual e primal, respectivamente. Como o valor da solução do programa relaxado é um limitante para o valor do programa inteiro, isso implica que o valor da solução obtida com essa estratégia é uma γ-aproximação para o problema.

Além de programação linear, muitas outras técnicas também são utilizadas na análise e projeto de algoritmos de aproximação, como busca local, algoritmos gulosos, programação dinâmica, arredondamento de dados, *factor revealing*, arredondamento aleatório, *chernoff bounds*, *semidefinite programming*, PCP, entre outros. Direcionamos o leitor interessado ao livro de algoritmos de aproximação de Williamson e Shmoys [69].

2.3 Um exemplo de algoritmo de aproximação

Em seguida, ilustramos a aplicação de técnicas de programação linear para projetar um algoritmo de aproximação para o Problema da Cobertura de Conjuntos (*Set Cover Problem, SC*). Diversos problemas de IRP podem ser definidos por formulações baseadas em cobertura como descrito no Capítulo 3, assim, as ideias do algoritmo de aproximação para o SC podem ser adaptadas para problemas de IRP. Construiremos um algoritmo de aproximação para o SC utilizando a técnica de arredondamento de PL. Para isso, será necessário projetar um algoritmo polinomial em função da entrada que devolva uma solução viável com custo no máximo um fator do custo da solução ótima do PL.

No SC, uma entrada consiste de um conjunto $E = \{e_1, ..., e_n\}$ de elementos, uma família $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ de subconjuntos de E e, para cada $S_j \in \mathcal{S}$, um peso não negativo $w_j \geq 0$. O objetivo é encontrar uma coleção de subconjuntos de custo mínimo que cobre todos os elementos de E. Em outras palavras, queremos um conjunto de índices $I \subseteq \{1, 2, ..., m\}$ que minimize $\sum_{j \in I} w_j$ sujeito a $\bigcup_{j \in I} S_j = E$.

Dada uma solução qualquer I, defina para cada $j \in \{1, ..., m\}$,

$$x_j = \begin{cases} 1 & \text{se } j \in I, \\ 0 & \text{se } j \notin I. \end{cases}$$

Note que cada vetor binário $x \in \{0, 1\}^m$ corresponde a uma solução I. Portanto, podemos

reescrever o problema enunciado acima como um PLI:

$$\begin{aligned} \min \quad & \sum_{j=1}^{m} w_j x_j \\ s.a \quad & \sum_{j:e_i \in S_j} x_j \geq 1, \quad \forall i \in \{1,2,...,n\}, \\ & x_j \in \{0,1\}, \quad \forall j \in \{1,2,...,m\}. \end{aligned} \tag{2.3}$$

A função-objetivo calcula a soma dos custos dos conjuntos escolhidos na solução. A primeira restrição garante que cada elemento i é coberto por pelo menos um conjunto S_j. Seja RL a relaxação linear do PLI acima, que é obtida ao substituirmos as restrições $x_j \in \{0,1\}$ por $x_j \geq 0$, para cada $j \in \{1,2,...,m\}$. Defina $f_i = |\{j : e_i \in S_j\}|$, ou seja, f_i é o número de conjuntos que contêm i e defina $f = \max\{f_i : e_i \in E\}$. Considere o seguinte algoritmo:

Algoritmo 1 Arredondamento-SC($\{e_1, e_2, ..., e_n\}$, $\{S_1, S_2, ..., S_m\}$):

1: Construa RL
2: Obtenha uma solução ótima x^*
3: $I \leftarrow \{j : x_j^* \geq \frac{1}{f}\}$

Vamos provar o seguinte teorema:

Teorema 3. *Arredondamento-SC é uma f-aproximação para SC.*

Demonstração. Arredondamento-SC é um algoritmo polinomial, visto que é possível construir e resolver a relaxação RL em tempo polinomial, assim como obter a solução I a partir de x^* em tempo linear em função da entrada. Dado qualquer $e_i \in E$, temos pela primeira restrição do programa linear que $\sum_{j:e_i \in S_j} x_j^* \geq 1$, que por sua vez é a soma de f_i termos. Portanto, para algum j com $e_i \in S_j$, o valor de x_j^* é maior ou igual que a média dessa soma, ou seja, $x_j^* \geq \frac{1}{f_i} \geq \frac{1}{f}$. Logo, por construção, para cada $e_i \in E$, existe índice j em I tal que $e_i \in S_j$, ou seja, I é uma cobertura. O custo da solução é dado por

$$w(I) = \sum_{j \in I} 1 \cdot w_j \leq \sum_{j \in I} (x_j^* f) \cdot w_j \leq f \sum_{j=1}^{m} w_j x_j^* = f \cdot z_{RL} \leq f \cdot OPT,$$

onde z_{RL} é o custo da solução ótima da relaxação linear RL, que por sua vez é um limitante inferior para o custo da solução ótima OPT do problema. □

Capítulo 3

Definição dos problemas

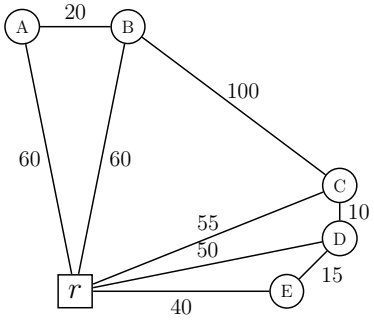
3.1 Inventory Routing Problem

No *Problema de Estoque e Roteirização* (*Inventory Routing Problem*, IRP), a entrada inclui um grafo $G = (V, E)$ e uma função de distância w que associa um número não negativo para cada par de vértices $i, j \in V$. Temos um depósito $r \in V$ especificado e cada elemento $v \in V \setminus \{r\}$ é chamado de cliente. O depósito possui capacidade de produção infinita, o que significa que o depósito pode fornecer tantas unidades de demanda quantas forem necessárias. Também, é dado um plano de horizonte com T períodos numerados de 1 até T. Para cada cliente i e período t, existe uma demanda por d_{it} unidades que deve ser atendida pelo depósito usando uma frota de veículos que pode ou não ter uma capacidade de carregamento. Uma solução corresponde a um conjunto de rotas em cada período do planejamento. Uma rota é uma sequência de vértices de V (clientes ou depósito) que devem ser visitados por um veículo que começa no depósito, faz o carregamento dos produtos e visita os clientes entregando os produtos. Na versão capacitada, o veículo não pode carregar mais unidades de produtos do que sua capacidade, C.

Para cada cliente i, existe um custo $h^i_{st} \geq 0$ que é pago para manter uma unidade de demanda no estoque do cliente i do período s até o período t. Assim, o custo para manter a demanda d_{it} nesse período é dado por $H^i_{st} = h^i_{st} d_{it}$. A única suposição que temos sobre a função de custo de armazenamento é monotonicidade, ou seja, para cada cliente i e quaisquer três períodos $u \leq s \leq t$, devemos ter $h^i_{ut} \geq h^i_{st}$. Como os custos de armazenamento são monotônicos, é preferível satisfazer demandas com itens entregues mais recentemente. Para a versão não capacitada do IRP, isso implica que entregas são feitas apenas para clientes com inventário vazio (cf. [27]).

Uma entrega consiste de um conjunto de demandas, sendo o custo de uma entrega equivalente ao comprimento de uma rota entre o depósito e todos os clientes envolvidos na entrega. Uma solução do IRP consiste de um conjunto de entregas para cada período do plano de horizonte de forma que esse conjunto de entregas atenda todas as demandas dos clientes antes dos seus períodos de expiração, i.e., para cada cliente i, deve haver pelo menos d_{it} itens em estoque no período t. Dada uma solução x do problema, a soma de todos os custos de armazenamento para todos os clientes no horizonte é o custo total de armazenamento (*holding cost*) $h(x)$, a soma de todas as distâncias percorridas em todas as rotas em todos os períodos do horizonte é o custo total de entrega (*ordering cost*) $r(x)$

e o custo total da solução é $c(x) = h(x) + r(x)$. O objetivo do problema é determinar uma solução com o menor custo $c(x)$ possível.



(a) Grafo ponderado

	Plano de horizonte			
Cliente	1	2	3	4
A	0	400	800	0
B	400	0	0	600
C	0	0	800	0
D	200	800	200	0
E	0	200	200	200

(b) Demandas por cliente e período

Figura 3.1: Exemplo de instância do IRP. Do lado esquerdo temos o grafo representando o depósito r e os clientes, de A a E. Do lado direito temos uma tabela com as demandas d_{it} de cada cliente para um plano de horizonte de $T = 4$ períodos. Neste exemplo, definimos $h^i_{st} = 0.1 \times (t - s)$ para todo $i \in V \setminus \{r\}$. Por exemplo, se os itens demandados pelo cliente A no período $t = 3$ forem entregues no período $s = 1$, então o custo de armazenamento dessa demanda será $H^i_{st} = 0.1 \times (3 - 1) \times 800 = 160$.

O IRP aparece diversas vezes na literatura na forma de programa linear inteiro. Um exemplo de formulação, descrita a seguir, foi proposta por Nagarajan e Shi [59]. Seja $\mathcal{C} = V \setminus \{r\}$ o conjunto de clientes, $[T] = \{1, 2, ..., T\}$ o plano de horizonte de T períodos, $\mathcal{D} = \{(i, t) \in \mathcal{C} \times [T] : d_{it} > 0\}$ o conjunto de demandas não nulas e $f(S) : 2^{\mathcal{C}} \to \mathbb{R}_+$ uma função que representa o custo da entrega para um conjunto de clientes $S \subseteq \mathcal{C}$ (note que essa função generaliza o custo de transporte do IRP). Nessa formulação, temos as seguintes variáveis binárias:

$$y^S_s = \begin{cases} 1, & \text{se o subconjunto } S \subseteq \mathcal{C} \text{ foi escolhido para se tornar uma rota no período } s, \\ 0, & \text{caso contrário;} \end{cases} \quad \text{e}$$

$$x^i_{st} = \begin{cases} 1, & \text{se a demanda } (i, t) \in \mathcal{D} \text{ foi satisfeita usando uma entrega no período s,} \\ 0, & \text{caso contrário.} \end{cases}$$

Com essa interpretação das variáveis, o IRP pode ser formulado como:

$$\min \sum_{S \subseteq \mathcal{C}} \sum_{s=1}^{T} f(S) y^S_s + \sum_{(i,t) \in \mathcal{D}} \sum_{s=1}^{t} H^i_{st} x^i_{st}$$

$$\text{(IRP)} \quad \text{s.a.} \quad \sum_{s=1}^{t} x^i_{st} = 1, \qquad (i, t) \in \mathcal{D} \tag{3.1}$$

$$x^i_{st} \leq \sum_{S : i \in S \subseteq \mathcal{C}} y^S_s, \qquad (i, t) \in \mathcal{D}, s \leq t \tag{3.2}$$

$$x^i_{st}, y^S_s \in \{0, 1\}. \qquad (i, t) \in \mathcal{D}, s \leq t, S \subseteq \mathcal{C} \tag{3.3}$$

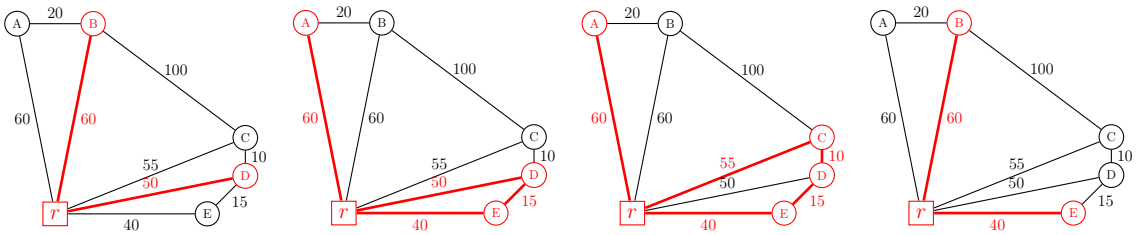
Dada uma solução (x, y), o primeiro termo da função-objetivo corresponde ao custo

de entrega $r(x, y)$ e o segundo termo corresponde ao custo de armazenamento $h(x, y)$. A restrição (3.1) garante que cada demanda (i, t) é satisfeita antes do seu período de expiração. A restrição (3.2) garante que a rota definida por S deve conter o elemento i se alguma demanda (i, t) foi atendida no período s. Tradicionalmente, a função $f(S)$ corresponde ao custo mínimo de uma rota envolvendo o conjunto $S \cup \{r\}$, ou ao custo mínimo de um conjunto de rotas, todas partindo de r e envolvendo coletivamente S. Em outras palavras, a função $f(S)$ corresponde ao valor de uma solução ótima para o TSP com os vértices $S \cup \{r\}$ ou ao valor de uma solução ótima para o VRP com depósito r e clientes S (ver descrições sobre o TSP e o VRP na Seção 3.4).

Nas Figuras 3.2, 3.3 e 3.4, temos três exemplos de soluções para o IRP considerando que a função $f(S)$ corresponde ao custo de uma solução ótima para o VRP com capacidade ilimitada. As arestas em destaque nas figuras correspondem às rotas realizadas em cada solução. A primeira solução minimiza os custos de armazenamento realizando entregas em todos os períodos de forma a não manter produtos em estoque. A segunda minimiza os custos de roteamento entregando todas as demandas do plano de horizonte no primeiro período. A terceira fornece uma melhor solução com relação as duas primeiras buscando ponderar adequadamente os custos de armazenamento e estoque.

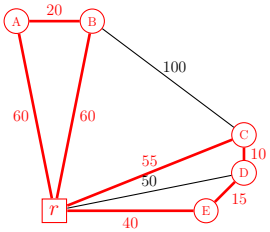
O problema descrito acima é apenas uma configuração possível das inúmeras variantes do IRP. Nesta descrição, o plano de horizonte é finito, temos apenas um depósito e vários clientes, o depósito possui capacidade infinita de produção, o roteamento é feito por uma frota ilimitada de veículos idênticos com capacidade infinita, as demandas dos clientes são previamente conhecidas e não são permitidas perdas nem atrasos nas mercadorias demandadas. Das principais formas com que o IRP pode ser estruturado, os seguintes critérios são os mais encontrados na literatura e servem para definir variantes do IRP:

- **Plano de horizonte:** Pode ser finito, infinito ou periódico (demandas dos clientes se repetem a cada certa frequência).
- **Número de depósitos x clientes:** Pode ser um para um, um para muitos ou muitos para muitos.
- **Capacidade de produção no depósito:** Pode ser limitada (existe uma quantidade máxima de produtos que pode ser entregue por período) ou ilimitada (o depósito tem tantos produtos quantos forem necessários em qualquer período).
- **Tipo de roteamento:** Roteamento direto (veículo sai do depósito, entrega mercadoria para apenas um cliente e volta para o depósito), roteamento múltiplo (veículo sai do depósito, entrega mercadorias para diversos clientes e volta para o depósito), ou roteamento contínuo (veículo nunca volta para o depósito).
- **Número de veículos:** Apenas um veículo, múltiplos veículos ou número ilimitado de veículos.
- **Capacidade dos veículos:** Pode ser limitada (existe uma quantidade máxima de produtos que cada veículo pode carregar), ou ilimitada (o veículo pode carregar quantos produtos forem necessários).



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$

Figura 3.2: Uma solução para o exemplo da Figura 3.1 minimizando custos de armazenamento. Em cada período, visitamos todos os clientes com alguma demanda positiva entregando todos os itens solicitados. Note que, nesse caso, o custo de armazenamento é 0. O custo de roteamento será 220 para o período $t = 1$, 225 para $t = 2$, 240 para $t = 3$ e 200 para $t = 4$. Portanto, o custo total da solução é $220 + 225 + 240 + 200 = 885$.

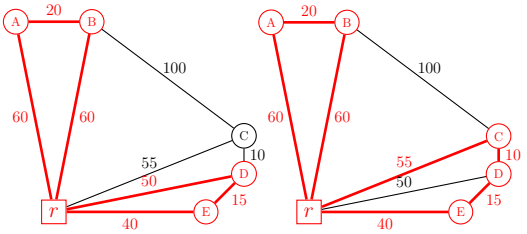


(a) $t = 1$

	Plano de horizonte			
Cliente	1	2	3	4
A	0	400	800	0
B	400	0	0	600
C	0	0	800	0
D	200	800	200	0
E	0	200	200	200

(b) Demandas por cliente e período

Figura 3.3: Uma solução para o exemplo da Figura 3.1 minimizando custos de roteamento. Realizamos apenas uma entrega em $t = 1$ satisfazendo todas as demandas dos clientes no plano de horizonte. O custo de roteamento no período $t = 1$ é de 260 e de 0 para os demais. Os custos de armazenamento são iguais a $40 + 160 = 200$ para o cliente A, 180 para o cliente B, 160 para o cliente C, $80 + 40 = 120$ para o cliente D e $20 + 40 + 60 = 120$ para o cliente E. O custo total da solução é $260 + 780 = 1040$.



(a) $t = 1$ (b) $t = 3$

	Plano de horizonte			
Cliente	1	2	3	4
A	0	400	800	0
B	400	0	0	600
C	0	0	800	0
D	200	800	200	0
E	0	200	200	200

(c) Demandas por cliente e período

Figura 3.4: Uma solução para o exemplo da Figura 3.1 com custo total menor que as anteriores. Satisfazemos todas as demandas acumuladas até $t = 2$ em $t = 1$ e de $t = 3$ até $t = 4$ em $t = 3$. Os custos de roteamento em $t = 1$ e $t = 3$ são iguais a 245 e 260. Os custos de armazenamento por cliente em $t = 2$ são iguais a 40 para A, 80 para D, 20 para E e 0 para os demais. Os custos de armazenamento por cliente em $t = 4$ são iguais a 60 para B, 20 para E e 0 para os demais. O custo total é $505 + 220 = 725$.

- **Composição da frota de veículos:** Homogênea (veículos idênticos), ou heterogênea (veículos com configurações diferentes, por exemplo, diferentes capacidades).
- **Capacidade máxima de inventário:** Pode ser limitada (existe uma quantidade máxima de produtos que um cliente consegue manter em estoque por período) ou ilimitada (um cliente pode manter quantos produtos forem necessários em estoque).
- **Capacidade mínima de inventário:** Pode ser limitada (o cliente precisa de uma quantidade mínima de produtos disponível em estoque por período) ou ilimitada (o cliente pode manter uma quantidade arbitrária de produtos em estoque).
- **Demandas:** Determinísticas (todas as demandas no plano de horizonte são previamente conhecidas), estocásticas (as demandas são conhecidas a partir de uma distribuição probabilística), ou dinâmicas (parte das demandas são conhecidas e são reveladas de tempos em tempos).

3.2 Star Inventory Routing Problem with Facility Location

No *Star Inventory Routing Problem with Facility Location* (SIRPFL), temos um grafo não direcionado $G = (V, E)$ com pesos nas arestas w_e para cada $e \in E$ e um plano de horizonte de T períodos numerados de 1 até T. Seja $\mathcal{F} \subseteq V$ um conjunto que representa os depósitos (também chamados de instalações dependendo da aplicação) e $\mathcal{C} \subseteq V$ um conjunto que representa os clientes (também chamados de varejistas). Cada depósito $j \in \mathcal{F}$ possui um custo de abertura f_j. Para cada cliente $i \in \mathcal{C}$ e período $t \in \{1, ..., T\}$, associamos uma demanda $d_{it} \geq 0$ que deve ser entregue em um período $s \leq t$ por um depósito aberto j através da aresta (i, j). Seja h^i_{st} o custo de armazenamento por unidade de demanda do período s até t para o cliente i. Assim como no IRP, a única suposição que temos sobre os custos de armazenamento é a monotonicidade.

O objetivo do SIRPFL é abrir um subconjunto de depósitos $F \subseteq \mathcal{F}$ que podem ser utilizados em todos os períodos do plano de horizonte e definir o conjunto de demandas que será entregue em cada período e para cada cliente, minimizando os custos totais de abertura dos depósitos, armazenamento das demandas e conexão dos clientes. Note que uma vez decididos os depósitos que serão abertos, as conexões entre depósito e clientes já são determinadas, bastando conectar cada cliente no depósito aberto mais próximo, formando rotas em formato de estrela (ver Figura 3.5). Os custos de conexão podem ser interpretados como custos de roteamento do veículo que realizará a entrega das demandas (neste problema, não há limite de número do veículos que podem ser utilizados). Note que a versão particular do SIRPFL com apenas um período e capacidade ilimitada nos veículos reduz-se ao clássico *Uncapacitated Facility Location Problem* (UFL) [49].

Consideramos duas versões para o SIRPFL, são elas: *Uncapacitated SIRPFL* (veículos não possuem capacidade, permitindo fazer a entrega de todas as demandas de um cliente em uma única viagem, caso seja necessário) e *Capacitated SIRPFL* (os veículos possuem uma capacidade máxima de produtos que podem ser transportados, logo, o veículo pode

precisar fazer mais do que uma viagem para entregar um conjunto de demandas para um cliente). Na versão capacitada do SIRPFL que consideramos, todos os veículos possuem a mesma capacidade, denotada por U. A versão capacitada possui ainda duas variações, são elas: *Capacitated Splittable SIRPFL* (a demanda d_{it} pode ser dividida em várias viagens) e *Capacitated Unsplittable SIRPFL* (a demanda d_{it} deve ser entregue em uma única viagem). Neste último problema, presumimos que todas as demandas possuem tamanho menor ou igual que U.

O *Inventory Access Problem* (Problema de Acesso ao Inventário, IAP), também introduzido por Jiao e Ravi, é um caso particular do SIRPFL no qual temos apenas um depósito e um cliente, bastando apenas decidir quando fazer a entrega de cada demanda. Assim como o SIRPFL, existem três versões para o IAP, cuja interpretação é análoga às versões do SIRPFL, são elas: *Uncapacitated IAP*, *Capacitated Splittable IAP* e *Capacitated Unsplittable IAP*.

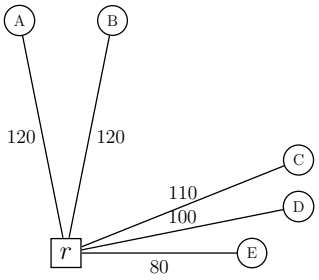
Nas Figuras 3.5 e 3.6 temos, respectivamente, um exemplo de instância e um exemplo de solução do SIRPFL com apenas um depósito, que por sua vez, é uma simplificação do exemplo da Figura 3.1 (IRP tradicional) utilizando roteamento direto. Note que, para essa configuração com apenas um depósito, o SIRPFL se reduz a resolver independentemente uma instância de IAP para cada cliente.

3.3 Tree Joint Replenishment Problem

No *Tree Joint Replenishment Problem* (Problema de Reabastecimento em Conjunto, JRP), a entrada contém uma árvore $\mathcal{T} = (V, E)$ cuja raiz representa o depósito r e as folhas representam os clientes, N. Para cada cliente $i \in N$, existe uma demanda $d_{it} \geq 0$ para todo $t \in [T] = \{1, 2, ..., T\}$. Para cada $i \in N$, denote por $path(i)$ o conjunto de vértices no único caminho de i até a raiz r. Para cada vértice $j \in V$, existe um custo $K^{(j)} \geq 0$ relacionado a visitar o vértice j. O custo de fazer uma entrega para um subconjunto $S \subseteq N$ de clientes é dado por $\sum_{j \in \bigcup_{i \in S} path(i)} K^{(j)}$, ou seja, é a soma dos custos de todos os vértices nos caminhos da raiz a cada vértice $i \in S$. Além disso, há também um número $h^i_{st} \geq 0$ que corresponde ao custo de armazenamento de um item do período s até t para o cliente i, sendo que a única suposição que temos dos custos de armazenamento, mais uma vez, é a monotonicidade.

Uma solução consiste dos períodos em que se realizam entregas e do conjunto de clientes reabastecidos em cada entrega, de forma a satisfazer todas as demandas com itens disponíveis no inventário. O objetivo do problema é satisfazer todas as demandas dos clientes antes dos seus períodos de expiração e minimizar a soma dos custos das entregas e dos custos de armazenamento.

As Figuras 3.7 e 3.8 ilustram, respectivamente, um exemplo de instância e uma solução para o *Tree* JRP. Esse exemplo consiste mais uma vez de uma simplificação do exemplo da Figura 3.1 (IRP tradicional), mas é também uma extensão do exemplo da Figura 3.5 (SIRPFL), visto que nessa versão é possível economizar custos de roteamento realizando rotas através da árvore, ao invés de apenas conexões diretas entre o depósito e os clientes. Nesse exemplo, nem todos os clientes são representados como nós folhas, mas

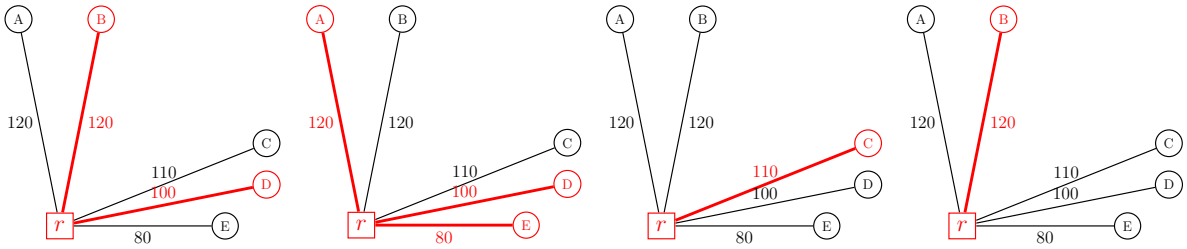


(a) Grafo ponderado

	Plano de horizonte			
Cliente	1	2	3	4
A	0	400	800	0
B	400	0	0	600
C	0	0	800	0
D	200	800	200	0
E	0	200	200	200

(b) Demandas por cliente e período

Figura 3.5: Exemplo da Figura 3.1 na versão SIRPFL. Note que o peso das arestas foram dobrados para corresponder ao custo de ida e volta do veículo entre o depósito e o respectivo cliente.



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$

Figura 3.6: Uma solução para o exemplo da Figura 3.5. Por inspeção, é possível obter a solução ótima dessa instância, resolvendo, para cada cliente, o IAP correspondente. Na solução ótima: A somente é visitado em $t = 2$, gerando um custo de $120 + 80 = 200$; B é visitado em $t = 1$ e $t = 4$, gerando um custo de 240; C é visitado em $t = 3$, gerando um custo de 110; D é visitado em $t = 1$ e $t = 2$, gerando um custo de $200 + 20 = 220$; E é visitado em $t = 2$, gerando um custo de $80 + 60 = 140$. O custo total da solução é portanto $200 + 240 + 110 + 220 + 140 = 910$.

essa representação é equivalente ao problema definido, visto que podemos criar uma instância correspondente adicionando uma nova folha ligada a cada cliente com uma aresta de custo 0 e transportar a demanda desse cliente para a nova folha criada. Note também que no *Tree* JRP, diferentemente do IRP e do SIRPFL, o grafo é ponderado nos vértices, mas podemos fazer uma correspondência entre pesos de arestas para pesos em vértices, bastando definir peso $K^{(r)} = 0$ para raiz r e associar, para cada cliente j, o peso da aresta incidente nele e em seu pai j' na árvore $\mathcal{T}$, isso é, definir $K^{(j)} = w_{j'j},$.

O *Capacitated Tree JRP* é uma extensão do *Tree* JRP. Neste problema, além de uma entrada do *Tree* JRP, recebemos uma capacidade de entrega, U. No *Capacitated Tree* JRP, uma entrega (conjunto de demandas entregues por um veículo) não pode exceder a capacidade U do veículo, podendo eventualmente ser necessário realizar mais do que uma entrega no mesmo período, ao contrário do *Tree* JRP, em que as demandas planejadas para um período podem ser sempre entregues por um único veículo. O objetivo do problema é o mesmo do *Tree* JRP, contudo, além de determinar o conjunto de demandas que será

satisfeito em cada período, também é necessário particionar estas demandas em diversas entregas de capacidade máxima U. A versão capacitada do *Tree* JRP ainda possui duas variações, são elas: *Capacitated Splittable Tree JRP* (a demanda d_{it} pode ser dividida em várias entregas em um mesmo período) e *Capacitated Unsplittable Tree JRP* (a demanda d_{it} deve ser entregue em uma única entrega). Um exemplo de solução para a versão *Splittable* do *Capacitated Tree* JRP é dada na Figura 3.9.

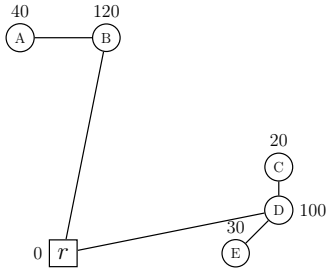
3.4 Problemas relacionados

Além de todas as variantes definidas neste capítulo, existem diversos outros problemas relacionados ao IRP, que incluem casos particulares, generalizações ou extensões. A seguir, apresentamos alguns desses problemas, separados em duas partes. A primeira parte, abrange os casos particulares do IRP, onde se encontram o VRP e JRP, problemas que já são muito bem estudados na literatura e que serviram de base para começar o estudo do IRP. A segunda parte, abrange generalizações e extensões do IRP, ou seja, versões do IRP mais genéricas que permitem resolver um conjunto de variantes do IRP, ou ainda, problemas que integram o IRP com algum outro problema conhecido.

3.4.1 Casos particulares

Vehicle Routing Problem, VRP: Problema clássico onde temos um depósito e vários clientes localizados em um espaço métrico. Os clientes precisam ser visitados para atender suas demandas e, para isso, o depósito dispõe de uma frota de veículos. Para o caso em que os veículos possuem uma capacidade limitada ou por alguma outra restrição do problema, é necessário realizar várias rotas, portanto, a solução é um conjunto de rotas, uma por veículo, com o menor custo de transporte. Diferentemente do IRP, não existem custos de inventário (*holding cost*) visto que o plano de horizonte é de apenas um período. Vale relembrar que o VRP é uma generalização do *Problema do Caixeiro Viajante* (*Traveling Salesman Problem*, TSP [33]), cuja tarefa é determinar a rota de comprimento mínimo para um conjunto de clientes dadas as distâncias entre cada par de clientes, com melhor fator de aproximação encontrado de $1{,}5-\epsilon$, para um certo ϵ constante muito pequeno [45]. Das diversas variantes do VRP, muitas delas já foram exploradas na área de algoritmos de aproximação, por exemplo, o *Generalized Multiple Depot Multiple Travelling Salesman Problem* [55], o *Capacitated VRP* [28], o *VRP with time window* [15], o *Dial-a-Ride Problem* [41], entre outros.

Joint Replenishment Problem, JRP: Outro problema clássico que envolve um conjunto de clientes que enfrentam demandas diárias por um item em um horizonte de planejamento. As demandas são satisfeitas por itens previamente mantidos no estoque de cada cliente e reabastecidos por meio de um pedido conjunto para um depósito de um subconjunto de clientes. Cada entrega possui um custo de preparação fixo (*setup cost*), independente do número de itens entregues, além do custo dos itens pertencentes à entrega. A solução é um conjunto de entregas que satisfaça todas as demandas a tempo e o objetivo é decidir quando fazer os pedidos e quais clientes serão atendidos em cada

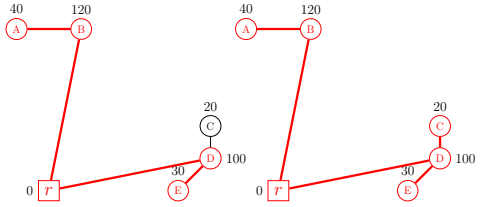


(a) Árvore $\mathcal{T}$

	Plano de horizonte			
Cliente	1	2	3	4
A	0	400	800	0
B	400	0	0	600
C	0	0	800	0
D	200	800	200	0
E	0	200	200	200

(b) Demandas por cliente e período

Figura 3.7: Exemplo da Figura 3.1 para uma versão *Tree* JRP. Os pesos das arestas da versão IRP foram correspondidos a pesos nos vértices para o *Tree* JRP (note que o peso dos vértices foram dobrados para corresponder ao custo de ida e volta do veículo através das arestas da árvore).

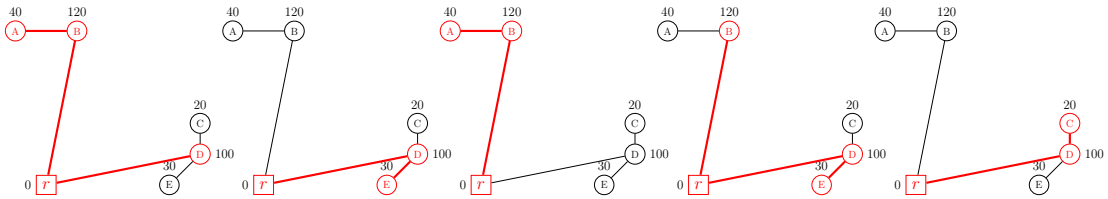


(a) $t = 1$ (b) $t = 3$

	Plano de horizonte			
Cliente	1	2	3	4
A	0	400	800	0
B	400	0	0	600
C	0	0	800	0
D	200	800	200	0
E	0	200	200	200

(c) Demandas por cliente e período

Figura 3.8: Uma solução para o exemplo da Figura 3.7. Neste caso, satisfazemos todas as demandas acumuladas até $t = 2$ em $t = 1$ e de $t = 3$ até $t = 4$ em $t = 3$. Os custos de roteamento em $t = 1$ e $t = 3$ são iguais a 290 e 310, respectivamente. Os custos de armazenamento por cliente em $t = 2$ são iguais a 40 (cliente A), 80 (cliente D), 20 (cliente E) e 0 para os demais clientes. Os custos de armazenamento por cliente em $t = 4$ são iguais a 60 (cliente B), 20 (cliente E) e 0 para os demais clientes. O custo total da solução é portanto $600 + 220 = 820$.



(a) $t = 1$ (rota 1) (b) $t = 1$ (rota 2) (c) $t = 3$ (rota 1) (d) $t = 3$ (rota 2) (e) $t = 3$ (rota 3)

Figura 3.9: Uma possível correção da solução da Figura 3.8 de forma a torná-la viável para o *Capacitated Tree* JRP. Neste exemplo, assumimos que $U = 1000$. Em $t = 1$ são necessárias duas rotas: a primeira entrega 400 unidades para A, 400 unidades para B e 200 unidades para D e a segunda entrega 800 unidades para D e 200 unidades para E. O novo custo de roteamento em $t = 1$ é 390. Em $t = 3$ são necessárias três rotas: a primeira entrega 800 unidades para A e 200 unidades para B, a segunda entrega 400 unidades para B, 200 unidades para D e 400 unidades para E e a terceira entrega 800 unidades para C. O novo custo de roteamento em $t = 3$ é 530. O novo custo da solução é portanto $920 + 220 = 1140$. Note que nesse exemplo, foi necessário dividir a demanda $t = 4$ de B, logo, esta solução somente é válida para o *Capacitated Splittable Tree* JRP.

pedido, de modo a minimizar os custos totais de entrega e armazenamento. Atualmente, existem diversos algoritmos de aproximação para o JRP [52, 53, 51]. O melhor fator de aproximação para esse problema é de 1,791, obtido por Bienkowski et al. 2013 [21].

3.4.2 Generalizações e extensões

Production Inventory Routing Problem, PIRP: Também conhecido como *Production Routing Problem*, PRP [4], este problema integra o VRP com outro problema clássico, o Problema de Dimensionamento de Lotes (*Lot-Sizing Problem*, LSP [68]). Neste problema, temos a mesma entrada do IRP, contudo, além de decidir o conjunto de rotas e as quantidades de produtos a serem enviadas em cada entrega, devemos decidir quando produzir os produtos envolvidos, adicionando novos custos de produção e restrições de estoque do lado do depósito. O PRP é mais geral que o IRP, visto que considera planejamento de produção além do planejamento de distribuição considerado no IRP.

Consistent Inventory Routing Problem, CIRP: O CIRP consiste de um IRP com a adição de restrições que buscam melhorar a qualidade do serviço de distribuição, de forma a gerar soluções mais consistentes com relação a quantidades de produtos entregues, frequência das entregas e gestão da força de trabalho. A motivação para o estudo deste problema é que muitas vezes (como comentado em [32]), a solução ótima do IRP pode gerar inconveniências tanto para o fornecedor e para o cliente. Dentre alguns exemplos de situações que buscam ser evitadas pelo CIRP temos: pequenas entregas para o mesmo cliente em curtos períodos de tempo, entregas muito grandes em longos períodos de tempo, veículos pouco carregados e veículos sobrecarregados.

Green Inventory Routing Problem, GIRP: Este problema, introduzido por Cheng et al. [29], é uma extensão do IRP que considera impactos ambientais e frotas de veículos heterogêneas. O propósito do problema é minimizar os custos de inventário e de transporte, dado que neste último caso são integrados custos adicionais de poluição devido à emissão de CO_2 dependendo do tipo de veículo ou combustível utilizado nas rotas.

Inventory Location Routing Problem, ILRP: O ILRP, discutido por Liu e Lee [54], é uma generalização do SIRPFL dada pela substituição das rotas diretas pela presença de rotas de comprimento mínimo, assim como o IRP tradicional. Além disso, o problema também é uma generalização do *Location Routing Problem* [56], que por sua vez, integra o UFL com o VRP. Versões intermediárias entre o SIRPFL e o ILRP (com relação a estrutura de roteamento) também existem, como por exemplo, o *tree* IRPFL [44], cujas rotas possuem uma estrutura em árvore.

Capítulo 4

Revisão bibliográfica

4.1 Inventory Routing Problem

A versão clássica do IRP [18] consiste de um problema que integra administração de inventário, roteamento de veículos e planejamento de entregas. Como observado antes, mesmo quando as decisões de inventário ou de roteamento são consideradas individualmente, recaímos em problemas NP-difíceis, como o Problema de Reabastecimento Conjunto (*Joint Replenishment Problem*, JRP [47]) e o Problema de Roteirização de Veículos (*Vehicle Routing Problem*, VRP [34]), que são casos particulares do IRP.

O IRP surgiu por volta da década de 80, estudado como uma variação do VRP com a adição de custos de inventário. O IRP foi introduzido por Bell et al. [18] com a finalidade de resolver o problema de abastecimento de gases e produtos químicos para usos industriais. A motivação para estudar o IRP começou na observação de que resolver o problema de inventário e roteamento separadamente pode gerar sub-otimalidades (ou seja, custos adicionais desnecessários para as empresas) e que existia uma intuição de que, se fosse criado um modelo unificado para ambos os problemas, soluções mais próximas da solução ótima seriam encontradas (reduzindo custos operacionais das empresas). No artigo de Bell et al., a nova abordagem consistiu de um algoritmo que utiliza relaxação lagrangiana para resolver um programa linear misto com cerca de 800 mil variáveis e 200 mil restrições, levando a economia de 6% a 10% dos custos operacionais. Por sua vez, Sarmiento e Nagi [63] defendem exaustivamente a importância de unificar modelos de diferentes decisões nos sistemas de produção e distribuição exibindo relatos de diversas empresas que utilizaram modelos unificados (todos eles baseados em heurísticas) e economizaram milhões de dólares em custos operacionais. Além do mais, Sarmiento, Ana e Nagi comentaram que até aquele momento (1999) não existiam muitos estudos unindo as análises de inventário e distribuição e que nenhuma metodologia para resolução exata do problema que fosse tratável na indústria havia sido desenvolvida.

Para verificar o estado da arte das formas de resolução do IRP, separamos os trabalhos em três vertentes: algoritmos exatos, heurísticas e algoritmos de aproximação. Para a referência bibliográfica das duas primeiras vertentes, buscamos identificar conceitos e temas que podem ser úteis na área de algoritmos de aproximação, além de identificar limitações que possam ajudar a comparar os resultados obtidos em aproximação até o momento e permitir identificar lacunas no estado da arte do IRP.

4.1.1 Algoritmos exatos

Algoritmos exatos do IRP não eram explorados até meados do ano 2000 devido à dificuldade de tratar o problema, mesmo para pequenas instâncias [31]. Assim, muitos desses algoritmos envolviam apenas versões simplificadas do IRP, ou seja, casos especiais do problema. Por exemplo, em 1984, Federgruen e Zipkin [39] desenvolveram um algoritmo exato para o IRP com apenas um período, utilizando Decomposição de Benders [19].

Archetti et al. 2007 [9] implementaram o primeiro algoritmo de *branch-and-cut* do IRP para o caso com apenas um veículo, resolvendo instâncias de 50 clientes com 3 períodos e 30 clientes com 6 períodos em cerca de 2 horas. Posteriormente, esse resultado foi melhorado por Solyalı e Süral 2011 [66] utilizando uma formulação mais forte. A versão com vários veículos foi estendida por Coelho e Laporte 2013 [31] e Adulyasak et al. 2013 [3] a partir do modelo de Archetti et al. As soluções consistiram na formulação do problema como um programa linear inteiro com variáveis $x_{i,j}^{k,t}$ indicando o número de vezes que uma aresta (i, j) no grafo de entrada G é utilizada na rota de um veículo k em um período t (definindo as rotas de cada veículo), variáveis $y_i^{k,t}$ indicando se um veículo é visitado por um veículo k no período t e variáveis $q_i^{k,t}$ denotando a quantidade de produto enviada do depósito ao cliente i utilizando o veículo k no período t. A partir da definição dessas variáveis, é possível modelar um programa linear inteiro com restrições análogas do LSP (restrições que calculam e mantêm a conformidade do inventário do depósito) e do TSP (restrições de grau de um vértice e de eliminação de subciclo para cada rota, que é definida por um veículo k e um período t).

Após o trabalho de Coelho e Laporte, o número de trabalhos considerando o IRP de maneira exata se manteve baixo. Em 2016, Desaulniers et al. [35] propuseram uma nova formulação matemática para o IRP e desenvolvem, segundo eles, um algoritmo de *branch-price-and-cut* que é o melhor dentre os algoritmos de *branch-price-and-cut* já vistos até 2016, obtendo soluções ótimas para diversas instâncias grandes que estavam em aberto até então. Mais recentemente, em 2019, Archetti et al. [10] resolveram de forma exata o IRPLR, variante do IRP que, ao invés de minimizar o custo total de distribuição, minimiza a razão do custo total de distribuição e a quantidade total de produtos enviada.

4.1.2 Algoritmos heurísticos

As primeiras formas de se resolver o IRP surgiram pela definição de heurísticas e na tentativa de criar políticas de entregas que garantissem uma melhora no *trade-off* entre o custo de armazenamento e o custo de distribuição, integrando o gerenciamento de pedidos e a entrega de produtos.

Em 1985, Burns et al. [24] compararam as estratégias de *direct shipping* (fazer uma entrega por cliente) e de *peddling* (fazer uma entrega para vários clientes) e desenvolveram métodos analíticos para calcular parâmetros que são úteis na avaliação das tomadas de decisão no processo de criar as entregas. Os métodos consistem de fórmulas matemáticas que, segundo eles, facilitam a análise de sensibilidade da mudança de algum parâmetro do problema, já que podem ser realizadas por qualquer um com uma simples calculadora, diferente de um modelo matemático, onde esses parâmetros são escondidos das pessoas e dificultam no entendimento do que está acontecendo no sistema do fornecedor.

No mesmo ano, Dror et al. [36] resolveram o problema (retornando as rotas dos veículos de fato, diferentemente de Burns et al.) formulando diferentes programas lineares inteiros e decompondo o problema em sub-problemas conhecidos, dentre eles o *Generalized Assignment Problem*, GAP [58], o VRP e o TSP. Na primeira abordagem, primeiro atribuímos os clientes nos períodos utilizando o GAP e, posteriormente, resolvemos o VRP para cada período e o conjunto de clientes selecionado nesse período. Na segunda abordagem, atribuímos os clientes em um veículo de algum determinado período e resolvemos o TSP para cada veículo. Em cada abordagem, alterou-se levemente a formulação do GAP utilizada para gerar o comportamento desejado. Diversos outros estudos consideraram a ideia de decomposição do IRP em sub-problemas conhecidos e separação do IRP em etapas. Campbell e Savelsbergh [26] citam os trabalhos mais relevantes relacionados a esse tipo de abordagem até 2002 e desenvolveram uma metodologia de duas fases para resolver o IRP, tentando escalar essa metodologia para instâncias da vida real. A primeira fase utilizou programação linear inteira e a segunda fase efetuou o roteamento dos veículos com auxílio de heurísticas.

Antes do conceito de gerenciamento da cadeia de fornecimento, era habitual (e ainda é, em muitos casos) a utilização da política RMI (*retailer-managed inventory*) onde o cliente decide a quantidade e o período da entrega, fazendo que o fornecedor organizasse apenas as rotas de distribuição [62, 12, 57]. Contudo, a nova política utilizada pelo IRP é a chamada VMI (*vendor-managed inventory*) onde o fornecedor conhece o plano de horizonte de demandas do cliente e administra tanto a criação de pedidos (quantidade do produto e quando entregar) quanto a distribuição (criação de jornadas de entrega em veículos). Uma comparação entre políticas RMI e VMI pode ser encontrada em [11] onde Archetti e Speranza modelam ambas políticas matematicamente e analisam as políticas teoricamente (provando formalmente que a política VMI é melhor que a RMI) e experimentalmente (implementando algoritmos e fazendo testes com instâncias de outros trabalhos).

Com relação às políticas de inventário e estratégias de roteamento, Bertazzi e Speranza [20] dão uma boa noção das diferentes políticas e estratégias existentes, algumas das mais utilizadas e conhecidas são: *Zero Inventory Ordering ZIO*, em que um cliente é reabastecido somente quando seu estoque se torna vazio; *Periodic* quando toda operação (entrega) para um cliente deve ser feita periodicamente dado uma frequência fixa; *Full load* quando entregas devem ser feitas somente com veículos lotados; *Maximum level, ML* quando é definida uma quantidade máxima de estoque que pode ser atingida para cada cliente; *Partition-based* quando o conjunto de clientes são particionados em subconjuntos e as rotas somente podem visitar clientes dentro de um conjunto destes subconjuntos. Chan et al. [27] descrevem algumas destas políticas, traz uma boa revisão das políticas não estudadas na sua pesquisa, enumerando trabalhos e resultados obtidos até aquele momento (1998), além de contribuir com um estudo teórico, provando um limitante inferior para políticas genéricas e criando uma *fixed partition policy* e limitantes superiores para a mesma.

Como existem inúmeros trabalhos resolvendo o IRP através de diversos tipos de técnicas e heurísticas, é difícil avaliar qual delas é a mais eficiente e/ou melhor em termos de qualidade nas soluções, uma vez que o conjunto de instâncias e de restrições consideradas pelos trabalhos podem variar muito. Segue uma lista de alguns trabalhos do IRP e

relacionados, divididos por técnica/metodologia:

- **Algoritmos genéticos:** Aziz e Mom [14], Sofianopoulou [65], Hiassat et al. [43]
- **Busca tabu:** Supithak [67], Archetti et al. [8], Seifbarghy e Samadi [64]
- **Busca local:** Lau et al. [50], Santos et al. [61], Alvarez et al. [7]
- **GRASP:** Dubedout et al. [37], Guemri et al. [42]
- **Outros:** Askin e Xia [13], Abdelmaguid et al. [2], Kheiri [46]

Métodos mais sofisticados em algoritmos heurísticos ainda estão sendo desenvolvidos como o método de Alkaabneh et al. [6] de 2020, que utiliza Decomposição de Benders e meta-heurísticas para resolver o IRP com produtos perecíveis e custos ambientais.

4.1.3 Algoritmos de aproximação

São poucos os trabalhos que contribuem com algoritmos de aproximação para alguma versão do IRP. Nesta seção, listamos todos os trabalhos em que encontramos um algoritmo de aproximação para alguma variante do problema. Um resumo dos resultados pode ser observado na Tabela 4.1.

O primeiro trabalho publicado foi desenvolvido por Fukunaga et al. em 2014 [40] e oferece as duas primeiras aproximações com fator constante para o IRP com apenas um depósito, demandas determinísticas, plano de horizonte finito e com a restrição de clientes serem atendidos a partir de uma *periodic policy* específica. A primeira é uma 2,55-aproximação para o caso em que os veículos possuem capacidade infinita e a segunda é uma 4,55-aproximação para o caso em que todos os veículos possuem uma mesma capacidade constante. As técnicas utilizadas no desenvolvimento dos dois algoritmos são uma simples redução para o problema chamado *Prize-Collecting Vehicle Routing Problem* e arredondamento de PL aleatório. Vale ressaltar que no trabalho de Fukunaga et al., os custos de entrega foram modelados através do comprimento da árvore de custo mínimo que conecta o depósito e os clientes envolvidos na entrega. Contudo, segundo os autores, a aproximação encontrada também se estende para o caso em que o custo das entregas são definidas pelo custo de uma rota mínima, dada por uma solução ótima de TSP.

O segundo trabalho, desenvolvido no ano seguinte por Nagarajan e Shi [59], ofereceu o primeiro algoritmo de aproximação com fator sub-logarítmico para o IRP com um depósito, demandas determinísticas, plano de horizonte finito e custos de entrega definidos pelo custo de rotas mínimas. Nesse trabalho, nenhuma restrição de política de entrega é necessária e os custos de armazenamento são generalizados para funções polinomiais. A modelagem do problema foi unificada com o problema chamado *Submodular Joint Replenishment Problem* (SJRP) que, por sua vez, possui a mesma entrada do IRP, embora os custos de entrega sejam determinados por uma função submodular qualquer em função do subconjunto de clientes atendidos na entrega. O algoritmo desenvolvido por Nagarajan e Shi rendeu uma $O(\frac{\log T}{\log \log T})$-aproximação, onde T é o número de períodos do plano de horizonte. Nagarajan e Shi comentam que o algoritmo também é uma $O(\log T)$-aproximação para a mesma versão do problema com custos de armazenamento arbitrários

(monótonos) fazendo alguns ajustes no algoritmo. A técnica utilizada por Nagarajan e Shi é um arredondamento de PL mais sofisticado, construindo o que eles chamam de *shadow intervals* associados às variáveis binárias relacionadas a cada demanda (i, t) do PLI construído (o mesmo enunciado na Subseção 3.1). Em seguida, as demandas são particionadas de acordo com estes intervalos e as entregas são finalmente definidas. Dado que a formulação (IRP) possui um número exponencial de variáveis, parte do trabalho de Nagarajan e Shi dedica-se a determinar uma forma de resolver a relaxação linear de (IRP) em tempo polinomial. Contudo, para o propósito do algoritmo de aproximação de Nagarajan e Shi, bastou provar que a relaxação pode ser resolvida de forma aproximada em tempo polinomial para o caso em que a função de custo das entregas é equivalente ao custo de um TSP. Para isso, os autores analisam o problema dual da formulação (IRP) (que por sua vez, possui um número exponencial de restrições) e determinam uma rotina de separação polinomial para resolver o dual de forma aproximada. No caso da função de custo das entregas ser aquela do SJRP, os autores mostraram que o PL pode ser resolvido de maneira exata utilizando um oráculo de separação polinomial.

Visto que o SJRP tem grande relação com o IRP, vale citar um terceiro trabalho, desenvolvido por Cheung et al. [30] meses após o trabalho de Nagarajan e Shi (ainda em 2015), que apresenta algoritmos de aproximação para versões particulares do SJRP, são elas: *Tree* JRP, *Laminar* JRP e *Cardinality* JRP. Essas versões diferem-se pela estrutura das funções submodulares utilizadas. As modelagens dos problemas são muito parecidas com as de Nagarajan e Shi e a técnica utilizada no *Tree* JRP e *Cardinality* JRP também consistiu em arredondamento de PL, proporcionando uma 3-aproximação e uma 5-aproximação para esses dois problemas, respectivamente. Para a versão *Laminar* JRP, foi desenvolvido um algoritmo exato baseado em programação dinâmica. Dentre as três versões estudadas por Cheung et al., destacamos o *Tree* JRP, problema também conhecido pelos nomes de *Multilevel JRP* [60] e *Multi-Level Aggregation Problem* [22]. O primeiro algoritmo de aproximação para o *Tree* JRP foi dado por Bechetti et al. 2006 [17], a partir de uma 2-aproximação baseada em arredondamento de PL para a versão particular do *Tree* JRP com *deadlines*, ou seja, com custo de armazenamento zero (caso a demanda seja satisfeita a tempo) ou infinito (caso a demanda seja satisfeita fora do prazo de expiração). Para a versão mais geral do *Tree* JRP, o primeiro algoritmo de aproximação com fator constante foi obtido pela própria 3-aproximação de Cheung et al. comentada anteriormente. Logo em seguida, Pedrosa [60] demonstrou que é possível reduzir o *Tree* JRP para o problema chamado *Multistage Assembly Problem*, que por sua vez, admite uma 2-aproximação dada por Levy et al., baseada numa abordagem primal-dual [51], obtendo dessa forma, o melhor algoritmo de aproximação para o *Tree* JRP encontrado até o momento, com fator de aproximação $2 + \epsilon$ (para um $\epsilon > 0$ arbitrário).

Após esses três trabalhos, nenhum outro estudo parecido foi realizado até metade e fim de 2019. Em maio de 2019, Jiao e Ravi [44] introduzem e desenvolvem as primeiras aproximações para o SIRPFL e o IAP. Nesse trabalho, os autores introduziram todas as variantes não capacitadas e capacitadas do SIRPFL e do IAP e forneceram algoritmos de aproximação com fator constante para cada versão, utilizando técnicas de arredondamento de PL. Como comentado por Jiao e Ravi em seu trabalho, a versão não capacitada do IAP se resume a resolver um *Lot-Sizing Problem* com apenas um item, que por sua vez, é pos-

Tabela 4.1: Melhores fatores de aproximação encontrados para diversas variantes do IRP

Problema	Resultado anterior	Melhor resultado
IRP *with any (monotone) holding cost*	$O(\log T)$ [59]	$O(\log\log\min(N,T))$ [23]
IRP *with polynomial holding cost*	$O(\frac{\log T}{\log\log T})$ [59]	$O(\log\log\min(N,T))$ [23]
Uncapacitated Periodic IRP	-	2,55 [40]
Capacitated Periodic IRP	-	4,55 [40]
Capacitated Splittable IAP	-	3 [44]
Capacitated Unsplittable IAP	-	6 [44]
Uncapacitated SIRPFL	12 [44]	1,488 [25]
Capacitated Splittable SIRPFL	24 [44]	3,236 [25]
Capacitated Unsplittable SIRPFL	48 [44]	6,029 [25]
Tree JRP	3 [30]	$2+\epsilon$ [60]

sível de ser resolvido na otimalidade em tempo polinomial a partir de uma programação dinâmica [68]. Já para as versões capacitadas do IAP, os autores fornecem os primeiros algoritmos de aproximação com fator constante, obtendo uma 3-aproximação para o *Capacitated Splittable IAP* e uma 6-aproximação para o *Capacitated Unsplittable IAP*, sendo o segundo algoritmo reduzido a partir do primeiro (veja os detalhes no Capítulo 5). Estendendo os resultados do IAP, Jiao e Ravi adaptaram as formulações matemáticas e os algoritmos de arredondamento para tratar as versões do SIRPFL, derivando aproximações constantes para o *Uncapacitated* SIRPFL, *Capacitated Splittable* SIRPFL e *Capacitated Unsplittable* SIRPFL, com fatores de 12, 24 e 48, respectivamente.

No fim de 2019, dois trabalhos melhoraram os resultados obtidos anteriormente. No primeiro, desenvolvido por Bosman e Olver [23], é desenvolvido um algoritmo de aproximação que resolve a mesma versão do IRP explorada por Nagarajan e Shi (resolvendo consequentemente o SJRP) com um fator de $O(\log\log\min(N,T))$, exponencialmente melhor que a antiga $O(\frac{\log T}{\log\log T})$-aproximação obtida por Nagarajan e Shi. Para obter esse resultado, eles utilizaram a mesma formulação que Nagarajan e Shi, mas associada com um problema chamado *Submodular Cover Over Time*, reduzindo o IRP para esse problema e desenvolvendo um algoritmo de arredondamento de PL aleatório a partir dele. No processo de arredondamento, também foi utilizado o conceito de intervalos, contudo, a estrutura utilizada é totalmente diferente da utilizada no trabalho de Nagarajan e Shi.

No segundo trabalho de 2019, Byrka e Lewandowski [25] melhoram as mesmas versões do SIRPFL estudadas por Jiao e Ravi. Neste trabalho, os autores fizeram uma simples redução do SIRPFL para uma versão do UFL chamada *Per-Client Nondecreasing Concave Connection Cost Facility Location* (NCC-FL), que por sua vez, pode ser reduzida ao UFL sem perda no fator de aproximação (ver detalhes em [25]). Na redução de cada versão do SIRPFL para o NCC-FL, Byrka e Lewandowski utilizaram como sub-rotina os algoritmos de aproximação para as versões correspondentes do IAP para computar os custos de conexão entre cada cliente e depósito (mais detalhes no Capítulo 5). A redução apresentada por Byrka e Lewandowski rendeu uma 1,488-aproximação para o *Uncapacitated* SIRPFL, uma 3,236-aproximação para o *Capacitated Splittable* SIRPFL e uma 6,029-aproximação para o *Capacitated Unsplittable* SIRPFL.

Capítulo 5

Algoritmo de aproximação para o Capacitated Splittable IAP

Neste capítulo, apresentamos uma melhoria na 3-aproximação para o *Capacitated Splittable* IAP de Jiao e Ravi [44], o que permite reduzir o fator de aproximação do algoritmo para 2,619 e, como consequência direta, melhorar as aproximações das versões capacitadas do SIRPFL. Para isso, iniciamos o capítulo com uma revisão das aproximações desenvolvidas por Jiao e Ravi para as versões capacitadas do IAP. Em seguida, apresentamos alguns resultados técnicos do SIRPFL provados por Byrka e Lewandoski [25] e mostramos como é possível obter algoritmos de aproximação para o SIRPFL a partir de qualquer algoritmo de aproximação para o IAP. Finalmente, apresentamos a nossa melhoria para o algoritmo de aproximação para o *Capacitated Splittable* IAP e apresentamos as consequências desse resultado para o *Capacitated Unsplittable* IAP, o *Capacitated Splittable* SIRPFL e o *Capacitated Unsplittable* SIRPFL.

5.1 Uma 3-aproximação para o Capacitated Splittable IAP

Como descrito no Capítulo 3, relembre que o IAP é a versão particular do SIRPFL com apenas um depósito e um cliente. Considere uma instância do IAP em que W é o peso da aresta que conecta o depósito e o cliente, d_t é a demanda do cliente no período t e h_{st} é o custo de armazenamento do cliente por unidade de demanda do período s até t. Definimos $H_{st} = h_{st}d_t$ correspondendo ao custo do cliente armazenar os d_t itens de s até t. Considere também variáveis binárias x_{st} cujos valores interpretamos como: $x_{st} = 1$ caso a demanda d_t seja satisfeita no período s e $x_{st} = 0$, caso contrário. Além disso, seja y_s a variável que indica o número de vezes que o cliente é visitado no período s. O *Capacitated Splittable* IAP pode ser formulado como o seguinte programa linear inteiro:

$$\min \sum_{s=1}^{T} W y_s + \sum_{t=1}^{T} \sum_{s=1}^{t} H_{st} x_{st}$$

$$\text{s.a.} \quad \sum_{s=1}^{t} x_{st} \geq 1, \qquad t \leq T \tag{5.1}$$

(CS-IAP)

$$y_s \geq \sum_{t=s}^{T} \frac{x_{st} d_t}{U}, \qquad s \leq T, \tag{5.2}$$

$$y_s \geq x_{st}, \qquad t \leq T, s \leq t, \tag{5.3}$$

$$x_{st} \geq 0, \qquad t \leq T, s \leq t \tag{5.4}$$

$$y_s \in \mathbb{Z}^+, \qquad s \leq T. \tag{5.5}$$

As restrições (5.1) garantem que toda demanda deve ser satisfeita antes do seu período de expiração. As restrições (5.2) fornecem um limitante inferior do número de viagens que podem ser feitas considerando a capacidade do veículo. As restrições (5.3) são desigualdades válidas para o modelo uma vez que impõem a condição de que, se o cliente não é visitado nenhuma vez no período s, então nenhuma demanda pode ser satisfeita no período s.

Seja (x, y) uma solução ótima da relaxação linear da formulação (CS-IAP) e, para cada $t \leq T$, defina s_t como o maior período tal que $\sum_{s=s_t}^{t} x_{st} \geq \frac{1}{2}$. O algoritmo de arredondamento de PL proposto por Jiao e Ravi é dado pelo Algoritmo 2.

Algoritmo 2 Regra de visitas para o *Capacitated Splittable* IAP:

1: Inicialize $A \leftarrow \emptyset$, $S \leftarrow \emptyset$
2: **enquanto** existe alguma demanda não satisfeita **faça**
3: Seja t uma demanda não satisfeita que possui o maior s_t
4: $A \leftarrow A \cup \{t\}$
5: $S \leftarrow S \cup \{s_t\}$
6: Satisfaça a demanda t no período s_t
7: **para** demandas não satisfeitas com $\hat{t} \geq s_t$ **faça**
8: Satisfaça a demanda $\hat{t}$ no período s_t
9: **devolve** conjunto de períodos de visita S

Uma vez definidos os períodos de visita, é fácil alocar as demandas: basta servir uma demanda t no maior período $s \in S$ tal que $s \leq t$. As demandas em A são chamadas de *âncoras* no algoritmo de Jiao e Ravi. Note que cada demanda $t \in A$ foi satisfeita em s_t e que todas as demandas $\hat{t}$ que foram satisfeitas em s_t possuem $s_{\hat{t}} \leq s_t$, uma vez que, se tivéssemos $s_{\hat{t}} > s_t$, então $s_{\hat{t}}$ seria processado antes da âncora s_t. Note também que todos os intervalos $[s_t, t]$ para toda âncora $t \in A$ são disjuntos entre si, pela construção do algoritmo. Por fim, como todas as demandas são servidas antes da sua data de expiração, o algoritmo acima devolve uma solução viável para o problema. Como vamos analisar o custo da solução na próxima seção para o algoritmo melhorado, optamos por não reproduzir a análise feita por Jiao e Ravi.

No trabalho realizado por Jiao e Ravi [44], provou-se que o custo de armazenamento de uma solução obtida pelo Algoritmo 2 é no máximo 2 vezes o custo de armazenamento ótimo da relaxação linear e o custo de entrega possui no máximo 3 vezes o custo de entrega ótimo da mesma relaxação, implicando que o algoritmo dos autores é uma 3-aproximação. Para obter uma aproximação melhorada, gostaríamos de equilibrar os fatores que multiplicam o custo de armazenamento e o custo de entrega. Intuitivamente, pode-se querer diminuir o custo de entrega, embora possivelmente aumentando o custo de armazenamento.

5.2 Uma 6-aproximação para o Capacitated Unsplittable IAP

Para obter a 6-aproximação do *Capacitated Unsplittable* IAP, Jiao e Ravi demonstraram que é possível transformar qualquer solução viável do *Capacitated Splittable* IAP em uma solução viável do *Capacitated Unsplittable* IAP que gasta no máximo duas vezes o número de visitas da solução do *Capacitated Splittable* IAP. Mais precisamente, os autores provaram a seguinte resultado:

Lema 1. *Se existe uma ρ-aproximação para o Capacitated Splittable IAP, então existe uma 2ρ-aproximação para o Capacitated Unsplittable IAP.*

Demonstração. Considere uma solução viável do *Capacitated Splittable* IAP. Para cada período de visita s da solução, seja D^s o conjunto de períodos das demandas que foram servidas no período s e particione D^s nos conjuntos $D^s_{\leq 1/2} = \{t \in D^s : d_t \leq U/2\}$ e $D^s_{>1/2} = \{t \in D^s : d_t > U/2\}$. Para cada período t em $D^s_{>1/2}$, crie uma nova viagem contendo a demanda d_t. Em seguida, atribua as demandas de cada período t em $D^s_{\leq 1/2}$ nas viagens criadas anteriormente utilizando uma estratégia gulosa de tal forma que nenhuma demanda d_t seja dividida em duas viagens e que nenhuma viagem possua mais do que U itens (criando novas viagens, caso seja necessário).

Uma vez que não alteramos os períodos de entrega das demandas, o custo de armazenamento da solução não é alterado. Portanto, para provar o lema, basta mostrar que o número de viagens foi, no máximo, dobrado. Denote por $n(s)$ o número de visitas realizadas no período s pela solução original e por $n'(s)$ o número de viagens obtidas após executar o procedimento descrito anteriormente. Mostraremos que $n'(s) \leq 2n(s)$.

É fácil observar que, no fim do procedimento, todas as viagens (com exceção de no máximo uma) são preenchidas com mais da metade da capacidade U. Se não existirem viagens com capacidade acima de $\frac{U}{2}$, então $n'(s) = n(s) = 1$. Caso contrário, a quantidade de produtos enviadas no período s é estritamente maior que $(n'(s) - 1) \cdot \frac{U}{2}$, ou seja, $\sum_{t \in D^s} d_t > (n'(s) - 1) \cdot \frac{U}{2}$. Mas, uma vez que cada viagem na solução original respeita a capacidade U, então $n(s) \geq \frac{\sum_{t \in D^s} d_t}{U}$, portanto, concluímos que $n(s) > \frac{n'(s)-1}{2}$. Visto que $n'(s)$ é inteiro, segue que $n'(s) < 2n(s) + 1$ e, portanto, $n'(s) \leq 2n(s)$. □

Agora, considere uma (λ_h, λ_r)-aproximação para o *Capacitated Splittable* IAP sendo um algoritmo de aproximação bi-fator que devolve uma solução com um custo de armazenamento de no máximo λ_h vezes o custo de armazenamento ótimo de uma solução ótima

e com custo de entrega de no máximo λ_r vezes o custo de entrega dessa mesma solução ótima. Dado que a prova do Lema 1 não altera o custo de armazenamento da solução gerada, o seguinte resultado também é válido:

Lema 2. *Se existe uma (λ_h, λ_r)-aproximação para o Capacitated Splittable IAP, então existe uma $(\lambda_h, 2\lambda_r)$-aproximação para o Capacitated Unsplittable IAP.*

5.3 Algoritmos de aproximação para o SIRPFL

No trabalho de Byrka e Lewandoski [25], os autores demonstram que é possível obter algoritmos de aproximação para cada versão do SIRPFL a partir de algoritmos de aproximação para uma variante do UFL chamada *Per-Client Nondecreasing Concave Connection Costs Facility Location* (NCC-FL), algoritmos estes também desenvolvidos pelos autores.

Lembrando que na versão clássica do UFL, a entrada contém um conjunto de depósitos $\mathcal{F}$ e um conjunto de clientes $\mathcal{C}$ e a tarefa é abrir um subconjunto de depósitos e conectar cada cliente ao depósito aberto mais próximo. O custo de abrir um depósito j é f_j e o custo de conectar o cliente i a um depósito j é a distância d_{ij}. Neste problema, supomos que as distâncias d_{ij} são definidas sobre um espaço métrico. O objetivo do UFL é minimizar os custos totais de abertura e de conexão.

Já no NCC-FL, a entrada é idêntica à do UFL, com a diferença que, para cada cliente i, contém também uma função côncava não decrescente g_i. O objetivo ainda é minimizar os custos totais de abertura e de conexão, mas o custo de conexão entre um cliente i e um depósito à distância x é dado por $g_i(x)$. Byrka e Lewandoski mostraram que é possível reduzir todas as versões do SIRPFL para o NCC-FL de forma que uma aproximação para NCC-FL implica em uma aproximação para cada versão do SIRPFL. Primeiramente, vamos provar a redução do *Uncapacitated* SIRPFL para o NCC-FL com o seguinte lema:

Lema 3. *Uncapacitated SIRPFL pode ser reduzido ao NCC-FL em tempo polinomial, preservando o fator de aproximação.*

Demonstração. Considere uma instância do *Uncapacitated* SIRPFL. Para cada cliente $i \in \mathcal{C}$ e depósito $j \in \mathcal{F}$, resolva a instância correspondente do *Lot-Sizing Problem* que considera apenas o depósito j e o cliente i, com custo $w_{i,j}$ por entrega e tal que armazenar uma unidade de demanda no inventário do período s a t custa h^i_{st}. Isso pode ser feito de maneira ótima em tempo polinomial [68]. Em seguida, construa uma função $g_i(x)$ definindo $g_i(w_{i,j})$ como o valor da solução obtida para cada depósito $j \in \mathcal{F}$ e interpolando linearmente os demais valores do domínio de $g_i(x)$.

É fácil notar que $g_i(x)$ é uma função côncava crescente. Isso segue do fato que a solução ótima do *Lot-Sizing Problem* com custo de entrega x também é uma solução viável para a mesma instância do problema com custo de entrega $\alpha \cdot x$ para $\alpha \geq 1$. Além do mais, o valor dessa solução com custo de entrega aumentado é no máximo α vezes o valor da solução ótima da instância com custo de entrega $\alpha \cdot x$.

Dessa forma, é possível resolver uma instância do *Uncapacitated* SIRPFL utilizando um algoritmo para o NCC-FL. Note que, após aplicar o algoritmo do NCC-FL, sabendo quais são os depósitos abertos, basta utilizar a solução ótima calculada pela programação

dinâmica do respectivo cliente e depósito conectado para satisfazer as demandas de cada cliente. Dessa maneira, tem-se que o custo da solução do NCC-FL é igual ao custo da solução construída para o SIRPFL e, portanto, o lema segue. □

Dado que existe uma 1,488-aproximação para o NCC-FL [25], o resultado do Lema 3 implica no seguinte resultado:

Corolário 1. *Existe uma* 1,488*-aproximação para o Uncapacitated SIRPFL.*

A seguir, provamos outro resultado de Byrka e Lewandowski que proporciona algoritmos de aproximação para ambas as versões do *Capacitated* SIRPFL a partir de qualquer algoritmo de aproximação da respectiva versão do *Capacitated* IAP. No lema seguinte, dizemos que um algoritmo é uma $(\lambda_f, \lambda_{h+r})$-aproximação para o SIRPFL se o custo de abertura da solução devolvida é de no máximo λ_f vezes o custo de abertura de uma solução ótima e a soma do custo de armazenamento e entrega da solução devolvida é de no máximo λ_{h+r} vezes a soma do custo de armazenamento e entrega dessa mesma solução ótima.

Lema 4. *Se existe uma ρ-aproximação para o Capacitated IAP (versão Splittable ou Unsplittable), então existe uma $(\lambda_f, \rho(1 + 2e^{-\lambda_f}))$-aproximação para a respectiva versão do Capacitated SIRPFL, para todo $\lambda_f \geq 1{,}6774$.*

Demonstração. A abordagem realizada neste caso é parecida com a prova do Lemma 3. Para cada cliente $i \in \mathcal{C}$ e depósito $j \in \mathcal{F}$, resolva a instância correspondente do *Capacitated* IAP que considera apenas o depósito j e o cliente i com custo de entrega $w_{i,j}$ e custos de armazenamento dados por h^i_{st} utilizando uma ρ-aproximação. Note que não podemos definir $g_i(w_{i,j})$ como o custo da solução obtida pelo algoritmo de aproximação, visto que a função resultante não seria necessariamente côncava.

Portanto, a construção de $g_i(x)$ será feita de uma maneira ligeiramente diferente. Primeiro, relembre que uma solução para o *Capacitated* IAP (tanto para a versão *Splittable* quanto para a *Unsplittable*) pode ser representada por um conjunto de períodos de visita. Agora, fixe um cliente i e considere uma instância correspondente do *Capacitated* IAP considerando um custo de entrega x tal que armazenar uma unidade de demanda no inventário do período s a t custa h^i_{st}. Dada uma solução S para essa instância, defina $V(S, x)$ como o valor da solução S com custo de entrega x e $A(x)$ como a solução dessa instância instância obtida a partir de uma ρ-aproximação para o *Capacitated* IAP.

Agora, seja $\mathcal{F} = \{1, 2, \ldots, n\}$ o conjunto de depósitos de uma instância do *Capacitated* SIRPFL e suponha, sem perda de generalidade, que $w_{i,1} \leq w_{i,2} \leq \cdots \leq w_{i,n}$. Note que para $x < y$, a solução $A(x)$ é viável para a mesma instância do *Capacitated* IAP com custo de entrega y. Além disso, o custo dessa solução para essa instância com custo de entrega y é no máximo $\frac{y}{x}$ vezes mais cara do que o custo dessa solução para a instância com custo de entrega x, ou, mais precisamente, $V(A(x), y) \leq \frac{y}{x} \cdot V(A(x), x)$. A seguir, construímos uma sequência de soluções. Seja $S_1 = A(w_{i,1})$. Para cada $j \in \{1, \ldots, n-1\}$ definimos:

$$S_{j+1} = \begin{cases} S_j, & \text{se } V(S_j, w_{i,j+1}) < V(A(w_{i,j+1}), d_{i,j+1})), \\ A(w_{i,j+1}), & \text{caso contrário.} \end{cases}$$

Finalmente, definimos $g(w_{i,j}) = V(S_i, w_{i,j})$ e interpolamos linearmente os valores da função $g_i(x)$ para cada $j \in \mathcal{F}$. Nesse caso, pode ser observado facilmente que $g_i(x)$ é uma função côncava não decrescente. Note também que, cada custo de conexão desta instância do NCC-FL é no máximo ρ vezes o custo de conexão ótimo para o respectivo cliente e depósito. Portanto, dado que existe uma $(\lambda_f, 1+2e^{-\lambda_f})$-aproximação para o NCC-FL [25] e uma vez que também perdemos um fator ρ com relação ao custo de conexão, obtemos o lema. □

5.4 Algoritmo melhorado para o Capacitated Splittable IAP

Para melhorar a 3-aproximação de Jiao e Ravi, queremos igualar os fatores das contribuições do custo de armazenamento e do custo de entrega. Na aproximação melhorada, usamos o mesmo algoritmo de arredondamento de PL dado pelo Algoritmo 2, com a exceção de que definimos s_t como sendo o maior período tal que $\sum_{s=s_t}^{t} x_{st} \geq \alpha$, para algum parâmetro $0 < \alpha < 1$. Note que o algoritmo original corresponde à escolha de $\alpha = 1/2$.

A seguir, analisamos o custo da solução do algoritmo modificado. Seja (x, y) uma solução ótima da relaxação linear da formulação (CS-IAP) e considere uma solução fornecida pelo Algoritmo 2 modificado. Primeiro, limitamos o custo de armazenamento com o seguinte lema:

Lema 5. *O custo de armazenamento da solução é de no máximo* $\frac{1}{1-\alpha} \sum_{t=1}^{T} \sum_{s=1}^{t} H_{st} x_{st}$.

Demonstração. Considere uma demanda qualquer para algum período t. Primeiro, nós analisamos o caso em que $t \in A$. Nesse caso, t é servido no período s_t, gerando um custo de armazenamento $H_{s_t t}$. Uma vez que $\sum_{s=1}^{t} x_{st} = 1$, temos que $\sum_{s=1}^{s_t} x_{st} \geq 1 - \alpha$, caso contrário, s_t não seria maximal. Uma vez que H_{st} é monotonicamente decrescente em s, isso implica que

$$H_{s_t t}(1-\alpha) \leq H_{s_t t} \sum_{s=1}^{s_t} x_{st} \leq \sum_{s=1}^{s_t} H_{st} x_{st} \leq \sum_{s=1}^{t} H_{st} x_{st}. \tag{5.6}$$

Agora, supomos que $t \notin A$. Seja s' o último período em S tal que $s' \leq t$. Nesse caso, o custo de armazenamento de t é $H_{s't}$. Uma vez que t não foi escolhido para ser uma âncora, nós sabemos que $s_t \leq s'$. Mais uma vez, temos que $\sum_{s=1}^{s_t} x_{st} \geq 1 - \alpha$, logo, $\sum_{s=1}^{s'} x_{st} \geq 1 - \alpha$. Portanto, como anteriormente, temos que

$$H_{s't}(1-\alpha) \leq H_{s't} \sum_{s=1}^{s'} x_{st} \leq \sum_{s=1}^{s'} H_{st} x_{st} \leq \sum_{s=1}^{t} H_{st} x_{st}. \tag{5.7}$$

Somando (5.6) ou (5.7) para todo período t, o lema segue. □

O custo de entrega da solução é limitada pelo seguinte lema:

Lema 6. *O custo de entrega da solução é de no máximo* $(1 + \frac{1}{\alpha}) \sum_{s=1}^{T} W y_s$.

Demonstração. Seja D^s o conjunto de demandas satisfeitas no período s pelo Algoritmo 2. O número de viagens no período $s \in S$, digamos $n(s)$, pode ser calculado como

$$n(s) = \left\lceil \sum_{t \in D^s} \frac{d_t}{U} \right\rceil \leq \sum_{t \in D^s} \frac{d_t}{U} + 1. \tag{5.8}$$

Note que os conjuntos D^s formam uma partição sobre o conjunto de todas as demandas do cliente. Logo, o número total de viagens na solução, digamos $n(S)$, é

$$n(S) = \sum_{s \in S} n(s) \leq \sum_{s \in S} (\sum_{t \in D^s} \frac{d_t}{U} + 1) = \sum_{s \in S} \sum_{t \in D^s} \frac{d_t}{U} + |S| = \sum_{t=1}^{T} \frac{d_t}{U} + |S|. \tag{5.9}$$

De acordo com as restrições (5.1) e (5.2), nós podemos limitar o primeiro termo do lado direito de (5.9) por

$$\sum_{s=1}^{T} y_s \geq \sum_{s=1}^{T} \sum_{t=s}^{T} \frac{d_t}{U} x_{st} = \sum_{t=1}^{T} \sum_{s=1}^{t} \frac{d_t}{U} x_{st} = \sum_{t=1}^{T} \frac{d_t}{U}. \tag{5.10}$$

Para o segundo termo do lado direito de (5.9), usamos o fato de que $|S| = |A|$. Por construção, os intervalos $[s_t, t]$ são disjuntos para $t \in A$ e $\sum_{s=s_t}^{t} x_{st} \geq \alpha$, logo

$$\frac{1}{\alpha} \sum_{s=1}^{T} y_s \geq \frac{1}{\alpha} \sum_{t \in A} \sum_{s=s_t}^{t} y_s \geq \frac{1}{\alpha} \sum_{t \in A} \sum_{s=s_t}^{t} x_{st} \geq \frac{1}{\alpha} \sum_{t \in A} \alpha = |A| = |S|, \tag{5.11}$$

onde a segunda desigualdade segue pela restrição (5.3).

Combinando (5.10) e (5.11) e substitutindo em (5.9), temos que $n(S) \leq (1+\frac{1}{\alpha}) \sum_{s=1}^{T} y_s$. Uma vez que cada viagem gera um custo de W, o lema segue. □

De acordo com os Lemas 5 e 6, o seguinte lema é imediato:

Lema 7. *O Algoritmo 2 é uma* $(\frac{1}{1-\alpha}, 1 + \frac{1}{\alpha})$*-aproximação para o Capacitated Splittable IAP.*

Para otimizar o fator de aproximação, igualamos os fatores do custo de armazenamento e do custo de entrega fixando $\alpha = \frac{\sqrt{5}-1}{2} \approx 0{,}618$. Substituindo esse valor de α no Lema 7, nós obtemos o seguinte resultado:

Teorema 4. *Existe uma* 2,619*-aproximação para o Capacitated Splittable IAP.*

5.5 Consequências para problemas relacionados

Se existe uma (λ_h, λ_r)-aproximação para o *Capacitated Splittable* IAP, então existe uma $(\lambda_h, 2\lambda_r)$-aproximação para o *Capacitated Unsplittable* IAP (de acordo com o Lema 2). Aplicando o mesmo procedimento do Lema 1 para converter uma solução do *Capacitated Splittable* IAP em uma solução viável do *Capacitated Unsplittable* IAP e, desta vez, utilizando a nossa $(\frac{1}{1-\alpha}, 1 + \frac{1}{\alpha})$-aproximação para o *Capacitated Splittable* IAP, obtemos o seguinte resultado:

Lema 8. *Existe uma* $(\frac{1}{1-\alpha}, 2+\frac{2}{\alpha})$*-aproximação para o Capacitated Unsplittable IAP.*

Otimizando os fatores de aproximação fixando $\alpha = \frac{\sqrt{17}-1}{4} \approx 0.781$, o seguinte resultado é imediato:

Corolário 2. *Existe uma* 4,562*-aproximação para o Capacitated Unsplittable IAP.*

Além do mais, se existe uma ρ-aproximação para o *Capacitated* IAP, então existe uma $(\lambda_f, \rho(1+2e^{-\lambda_f}))$-aproximação para o *Capacitated* SIRPFL (de acordo com o Lema 4). Logo, escolhendo o valor de λ_f de maneira apropriada e, para cada aproximação melhorada obtida para o *Capacitated* IAP, obtemos os seguintes resultados:

Corolário 3. *Existe uma* 2,905*-aproximação para o Capacitated Splittable SIRPFL.*

Corolário 4. *Existe uma* 4,649*-aproximação para o Capacitated Unsplittable SIRPFL.*

Capítulo 6

Algoritmos de aproximação para o Capacitated Tree JRP

Neste capítulo, apresentamos os primeiros algoritmos de aproximação com fator constante para o *Capacitated Tree* JRP. A primeira fase de nossos algoritmos é baseada no algoritmo de arredondamento de PL para o *Tree* JRP desenvolvido por Cheung et al [30], que é utilizado para realizar o agendamento de entregas e decidir em que períodos serão realizadas entregas. Na segunda fase, executamos um algoritmo de roteamento de entregas distinto para cada versão do problema, que decide quantos e quais clientes farão parte de cada entrega.

Iniciamos o capítulo revisando o algoritmo de Cheung et al. Em seguida, descrevemos dois algoritmos para o *Capacitated Tree* JRP. Primeiro, apresentamos uma 6-aproximação com um procedimento de roteamento mais simples e que vale para a versão *Splittable* do problema apenas. Depois, melhoramos esse algoritmo utilizando um procedimento de roteamento ligeiramente mais elaborado e obtendo uma 5-aproximação que vale tanto para a versão *Splittable* quanto para a versão *Unsplittable* do problema.

6.1 Uma 3-aproximação para o Tree JRP

Considere uma instância do *Tree* JRP. Denote por $\mathcal{D}$ o conjunto de todos os pares (i,t) com $d_{it} > 0$ e defina o valor fixo $H^i_{st} = h^i_{st} d_{it}$, i.e., H^i_{st} é o custo de armazenar os d_{it} itens do período s até t por um cliente i. Considere também variáveis binárias x^i_{st} cujos valores interpretamos como: $x^i_{st} = 1$ caso a demanda $(i,t) \in \mathcal{D}$ seja satisfeita usando uma entrega no período s e $x^i_{st} = 0$, caso contrário. Além disso, considere variáveis binárias y^j_s interpretadas como: $y^j_s = 1$ se o vértice j foi visitado no período s e $y^j_s = 0$, caso contrário. Segue a formulação de programação linear inteira de Cheung et al. para o *Tree* JRP [30]:

$$\min \sum_{j \in V(\mathcal{T})} \sum_{s=1}^{T} K^{(j)} y_s^j + \sum_{(i,t) \in \mathcal{D}} \sum_{s=1}^{t} H_{st}^i x_{st}^i$$

$$\text{(T-JRP)} \quad \text{s.a.} \quad \sum_{s=1}^{t} x_{st}^i = 1, \qquad (i,t) \in \mathcal{D} \tag{6.1}$$

$$x_{st}^i \leq y_s^j, \qquad (i,t) \in \mathcal{D}, s \leq t, j \in path(i) \tag{6.2}$$

$$y_s^j, x_{st}^i \in \{0,1\}, \qquad (i,t) \in \mathcal{D}, s \leq t, j \in V(\mathcal{T}). \tag{6.3}$$

As restrições (6.1) garantem que todas as demandas são entregues antes dos seus períodos de expiração. As restrições (6.2) garantem que, para a demanda (i,t) ser entregue no período s, todos os vértices no caminho $path(i)$ devem ser visitados no período s e, além do mais, caso um vértice j não seja visitado em s, então nenhuma demanda (i,t) tal que $j \in path(i)$ pode ser satisfeita em s. As restrições (6.3) definem o domínio das variáveis.

O problema dual da relaxação linear da formulação é denotado por (D-T-JRP). Ele é usado na análise do fator de aproximação do algoritmo de arredondamento proposto.

$$\max \sum_{(i,t) \in \mathcal{D}} B_t^i$$

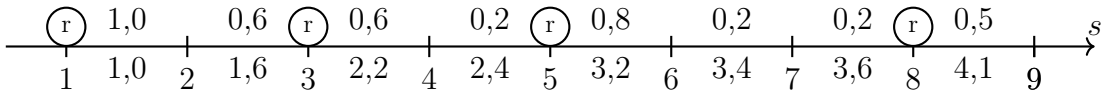
$$\text{(D-T-JRP)} \quad \text{s.a.} \quad B_t^i \leq H_{st}^i + \sum_{j \in path(i)} L_{st}^{ij}, \qquad (i,t) \in \mathcal{D}, s \leq t \tag{6.4}$$

$$\sum_{i: j \in path(i)} \sum_{t=s}^{T} L_{st}^{ij} \leq K^j, \qquad s \leq T, j \in V(\mathcal{T}) \tag{6.5}$$

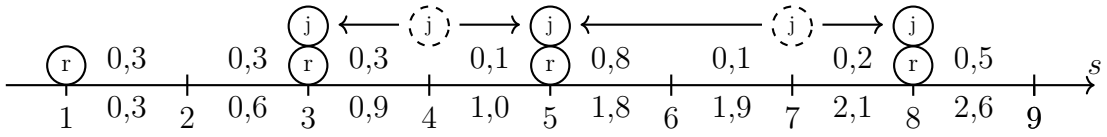
$$L_{st}^{ij} \geq 0. \qquad (i,t) \in \mathcal{D}, s \leq t, j \in V(\mathcal{T}) \tag{6.6}$$

O procedimento de arredondamento do algoritmo considera apenas os valores das variáveis y_s^j da relaxação linear de (T-JRP) de forma a obter os períodos em que cada vértice será visitado. Uma vez decididos os vértices que serão visitados em cada período, basta satisfazer cada demanda (i,t) no maior período $s \leq t$ tal que i é visitado.

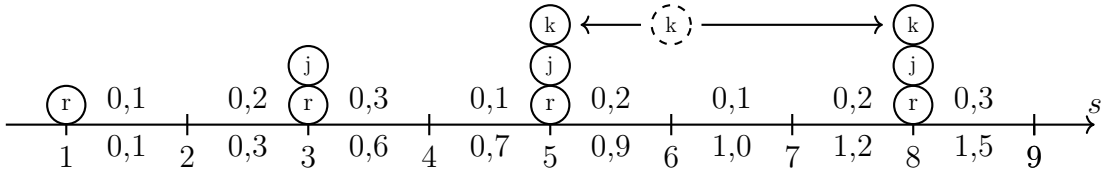
O algoritmo de arredondamento inicia criando entregas na raiz r em todos os períodos $s \leq T$ tais que o intervalo $(\sum_{u=1}^{s-1} y_u^r, \sum_{u=1}^{s} y_u^r]$ contém um inteiro. Para os demais vértices, utilizando qualquer percurso da árvore $\mathcal{T}$ em pré-ordem, criam-se tentativas de entrega para cada $j \in V(\mathcal{T}) \setminus \{r\}$ em cada período s tal que $(\sum_{u=1}^{s-1} y_u^j, \sum_{u=1}^{s} y_u^j]$ contém um inteiro. Seja j' o pai do vértice j (que já foi processado pela construção do algoritmo). Para cada tentativa de entrega s do vértice j, criam-se no máximo duas entregas em j (se existirem), são elas: no último período de entrega de j' em $[1,s]$ e no primeiro período de entrega de j' em $(s,T]$. A Figura 6.1 ilustra o procedimento de arredondamento para os três primeiros vértices de uma árvore. O pseudocódigo do algoritmo de arredondamento de PL proposto para o *Tree* JRP por Cheung et al. é dado pelo Algoritmo 3.



(a) Criação de entregas para a raiz r.



(b) Criação de entregas para um vértice j filho de r.



(c) Criação de entregas para um vértice k filho de j.

Figura 6.1: Exemplo do procedimento de arredondamento de PL de Cheung et al. para a criação de entregas. Nesse exemplo, o plano de horizonte é de 8 períodos. Os círculos pontilhados representam as tentativas de entrega para determinado vértice e os círculos contínuos representam as entregas definitivas. Os valores acima de cada intervalo representam os valores das variáveis y_s^j, por exemplo, $y_5^r = 0{,}8$, $y_4^j = 0{,}1$ e $y_2^k = 0{,}2$. Os valores abaixo de cada intervalo representam os valores acumulados das variáveis y_s^j ao decorrer do plano de horizonte.

Algoritmo 3 Algoritmo de arredondamento de PL para o *Tree* JRP:

1: Resolva a relaxação linear de (T-JRP) e obtenha (x, y)
2: **para** cada período s de 1 até T **faça**
3: **se** o intervalo $(\sum_{u=1}^{s-1} y_u^r, \sum_{u=1}^{s} y_u^r]$ contém um inteiro **então**
4: Crie uma entrega em r no período s
5: **para** cada nó $j \in V(\mathcal{T}) \setminus \{r\}$ em pré-ordem **faça**
6: **para** cada período s de 1 até T **faça**
7: **se** o intervalo $(\sum_{u=1}^{s-1} y_u^j, \sum_{u=1}^{s} y_u^j]$ contém um inteiro **então**
8: Crie uma tentativa de entrega em j no período s
9: **para** cada período s que contém uma tentativa de entrega de j **faça**
10: **se** existe uma entrega para $pai(j)$ em $(s, T]$ **então**
11: Crie uma entrega em j na primeira entrega de $pai(j)$ em $(s, T]$
12: **se** existe uma entrega para $pai(j)$ em $[1, s]$ **então**
13: Crie uma entrega em j na última entrega de $pai(j)$ em $[1, s]$
14: Satisfaça cada demanda (i, t) na entrega mais próxima com relação a t que contém i

Para provar que o Algoritmo 3 sempre devolve uma solução viável para o problema, é necessário provar o seguinte lema:

Lema 9. *Para cada vértice $j \in V(\mathcal{T})$ e intervalo de períodos $[s, t]$ onde $\sum_{u=s}^{t} y_u^j \geq 1$, o Algoritmo 3 cria uma entrega no vértice j em algum período em $[s, t]$.*

Demonstração. Primeiramente, mostramos a seguinte propriedade da solução ótima (x, y): para cada par de vértices j e j' tal que $j' \in path(j)$ e qualquer período u, temos que $y_u^j \leq y_u^{j'}$. Para isso, considere um vértice j não folha com filhos $\mathcal{C}$. Observe que, para cada período u, temos que $y_u^j = \max_{k \in \mathcal{C}} y_u^k$, caso contrário, seria possível diminuir o valor da função-objetivo diminuindo y_u^j e mantendo a viabilidade da solução. Portanto, para cada par de vértices j e j' tal que $j' \in path(j)$ e qualquer período u, temos que $y_u^j \leq y_u^{j'}$. Levando isto em consideração, vamos provar o lema por indução na profundidade do nó j:

Caso base (nó raiz): Para qualquer intervalo $[s, t]$ onde $\sum_{u=s}^{t} y_r^j \geq 1$, o intervalo $(\sum_{u=1}^{s-1} y_u^r, \sum_{u=1}^{t} y_u^r]$ contém um inteiro. Portanto, por construção do algoritmo, uma entrega na raiz deve existir em algum período em $[s, t]$.

Passo de indução: Considere um vértice j e qualquer intervalo $[s, t]$ tal que $\sum_{u=s}^{t} y_u^j \geq 1$. Seja j' o pai do vértice j. Uma vez que $y_u^j \leq y_u^{j'}$ para todo u, temos então que $\sum_{u=s}^{t} y_u^{j'} \geq 1$. Usando a hipótese de indução, temos que existe uma entrega para o nó j' em algum período em $[s, t]$; vamos chamar este período de u'. Assim como no caso base e, pelos passos do Algoritmo 3, temos que existe uma tentativa de entrega para j em algum período em $[s, t]$; vamos chamar este período de u. Escolhendo u' como a entrega de j' mais próxima de u que está contido em $[s, t]$, sabemos que o algoritmo criará uma entrega em u', porque o algoritmo cria entregas em ambos os lados da tentativa de entrega em u. Dessa forma, o passo de indução está provado e, portanto, o enunciado do lema segue. □

Vamos continuar a prova da viabilidade do Algoritmo 3. Para isso, considere uma demanda qualquer $(i, t) \in \mathcal{D}$. De acordo com as restrições (6.1) e (6.2), concluímos que $\sum_{s=1}^{t} y_s^j \geq 1$. Logo, pelo Lema 9, existe uma entrega para i em algum período em $[1, t]$, implicando que todas as demandas são satisfeitas antes dos seus períodos de expiração e portanto, a solução dada pelo algoritmo é de fato viável para o *Tree* JRP.

Com relação ao custo da solução, separamos a análise do custo de entrega e do custo de armazenamento. Para limitar o custo de entrega, provamos o seguinte lema:

Lema 10. *O custo de entrega da solução do Algoritmo 3 é de no máximo* $2 \sum_{s=1}^{T} \sum_{j \in V(\mathcal{T})} K^{(j)} y_s^j$.

Demonstração. De acordo com o Algoritmo 3, o nó raiz é visitado no máximo $\lfloor \sum_{u=1}^{T} y_u^r \rfloor$ vezes e os demais nós são visitados no máximo $2\lfloor \sum_{u=1}^{T} y_u^j \rfloor$ vezes, uma vez que realizamos no máximo 2 entregas para cada tentativa de entrega de j. Uma vez que cada visita a um vértice j incorre em um custo de $K^{(j)}$, usando os limitantes para cada vértice e somando para todos os vértices, obtemos o lema. □

Com relação ao custo de armazenamento, mostramos que o custo de armazenamento da solução do Algoritmo 3 é de no máximo $\sum_{(i,t) \in \mathcal{D}} B_t^i$, o valor ótimo da solução do problema dual (D-T-JRP).

Lema 11. *O custo de armazenamento da solução do Algoritmo 3 é de no máximo* $\sum_{(i,t) \in \mathcal{D}} B_t^i$.

Demonstração. Para cada demanda $(i, t) \in \mathcal{D}$, considere o conjunto de entregas que servem (i, t) de maneira fracionária, de acordo com a solução (x, y) da relaxação linear de (T-JRP), i.e., o conjunto de períodos s tal que $x_{st}^i > 0$. Seja s_1 o menor período desse

conjunto de entregas fracionárias. Dizemos que $[s_1, t]$ é o *intervalo ativo* da demanda (i, t). Assim, se $x^i_{st} > 0$, então $s \geq s_1$ e vale $\sum_{s=s_1}^{t} x^i_{st} = 1$.

Uma vez que $x^i_{s_1 t} > 0$, temos que, por folgas complementares, a restrição do problema dual correspondente a esta variável ser justa, ou seja, $B^i_t = H^i_{s_1 t} + \sum_{j \in path(i)} L^{ij}_{s_1 t}$. Dado que as variáveis $L^{ij}_{s_1 t}$ são não negativas, temos que $H^i_{s_1 t} \leq B^i_t$. Entretanto, sabemos que H^i_{st} é monotonicamente decrescente em s, logo, para qualquer período s no intervalo ativo de (i, t), temos que $H^i_{st} \leq B^i_t$. Logo, para provar o lema, basta mostrar que existe uma entrega para i no intervalo ativo da demanda (i, t). Pela própria definição de um intervalo ativo e pelas restrições (6.1) e (6.2), temos que $\sum_{s=s_1}^{t} y^i_s \geq 1$. Portanto, utilizando o Lema 9 sobre o intervalo $[s_1, t]$, temos que existe alguma entrega em i no intervalo ativo de (i, t), como desejado. □

Finalmente, de acordo com os Lemas 10 e 11 e pelo fato de que as soluções ótimas da relaxação linear de (T-JRP) e do problema dual (D-T-JRP) serem limitantes inferiores para a solução ótima do *Tree* JRP, obtemos o seguinte resultado:

Teorema 5. *O Algoritmo 3 é uma* 3*-aproximação para o Tree JRP.*

6.2 Uma 6-aproximação para o Capacitated Splittable Tree JRP

Assim como o algoritmo de Cheung et al., nosso algoritmo é baseado em arredondamento de PL. A seguir, denote por $\mathcal{T}_j$ a sub-árvore maximal de $\mathcal{T}$ enraizada em $j \in \mathcal{T}$. No PL misto abaixo, temos uma variável inteira y^j_s que indica o número de vezes que o vértice j é visitado no período s. Além disso, temos uma variável racional x^i_{st} que representa a fração dos d_{it} itens da demanda $(i, t) \in \mathcal{D}$ que é satisfeita por uma entrega no período s. Uma solução para o programa abaixo é um limitante inferior para o valor de uma solução ótima de qualquer versão capacitada do *Tree* JRP.

$$\min \sum_{j \in V(\mathcal{T})} \sum_{s=1}^{T} K^{(j)} y^j_s + \sum_{(i,t) \in \mathcal{D}} \sum_{s=1}^{t} H^i_{st} x^i_{st}$$

(CT-JRP)

$$\text{s.a.} \quad \sum_{s=1}^{t} x^i_{st} = 1, \qquad (i, t) \in \mathcal{D} \tag{6.7}$$

$$x^i_{st} \leq y^j_s, \qquad (i, t) \in \mathcal{D}, s \leq t, j \in path(i) \tag{6.8}$$

$$\sum_{i \in V(\mathcal{T}_j) \cap N} \sum_{t=s}^{T} \frac{d_{it}}{U} x^i_{st} \leq y^j_s, \qquad s \leq T, j \in V(\mathcal{T}) \tag{6.9}$$

$$y^j_s \in \mathbb{Z}^+, x^i_{st} \geq 0, \qquad (i, t) \in \mathcal{D}, s \leq t, j \in V(\mathcal{T}) \tag{6.10}$$

A principal diferença da formulação acima para a versão sem capacidades descrita na seção anterior é a adição das restrições (6.9). Elas limitam o número de vezes que cada vértice j deve ser visitado no período s em função do número de demandas servidas no período.

Nosso algoritmo consiste de duas etapas. Na primeira, determinamos em que períodos serão agendadas entregas e quais clientes participarão de entregas nos períodos, no qual realizamos um procedimento parecido com o do algoritmo do Tree JRP apresentado anteriormente. Para isso, utilizamos uma sub-rotina de arredondamento de Cheung et al., mas adaptada para (CT-JRP). Na segunda, para cada período, determinaremos quantas entregas e quais clientes serão incluídos em cada entrega, utilizando um algoritmo de roteamento.

Agendamento de entregas. Dada uma solução ótima da relaxação linear, determinaremos em que períodos cada vértice será visitado. Primeiro, criamos um conjunto de entregas tentativas. Para cada $j \in V(\mathcal{T})$, criamos uma entrega tentativa em todo período $s \in \{1, 2, \ldots, T\}$ tal que o intervalo $(\sum_{u=1}^{s-1} y_u^j, \sum_{u=1}^{s} y_u^j]$ contém um inteiro. Intuitivamente, podemos pensar que o total de entregas fracionárias y_u^j acumuladas entre duas entregas tentativas consecutivas é de no máximo um. Depois, selecionamos entregas definitivas. Escolhemos as próprias entregas tentativas para a raiz r e processamos os demais vértices j iterativamente em um percurso *pré-ordem*. Seja j' o pai do vértice j (que já foi processado antes pelo algoritmo). Para cada entrega tentativa em um período s, escolhemos no máximo duas entregas definitivas em j: no último período de entrega definitivo de j' em $[1, s]$ e no primeiro período de entrega definitivo de j' em $(s, T]$.

Roteamento de entregas. Para cada período s, queremos construir um conjunto de entregas incluindo os vértices com entregas agendadas nesse período. Note que uma entrega corresponde a uma subárvore enraizada em r e que várias entregas diferentes no mesmo período podem conter vértices diferentes. O algoritmo construirá todas as entregas simultaneamente processando a árvore de maneira *bottom-up*, i.e., processamos os vértices de $\mathcal{T}$ que têm entrega agendada em s em um percurso em *pós-ordem*.

Para uma folha i de $\mathcal{T}$, que corresponde a um cliente, seja s' o período da próxima entrega agendada para i, ou $T+1$ se não houver. A demanda acumulada de i no período s é definida como $D_i^s := \sum_{t=s}^{s'-1} d_{it}$. Criamos $\lfloor \frac{D_i^s}{U} \rfloor$ entregas cheias para i com U itens cada e, possivelmente, uma entrega parcial com o número de itens restantes da demanda, diga-se, R_i^s.

Para um vértice interno j, definimos D_j^s como a soma das demandas residuais R_ℓ^s de seus filhos ℓ. Analogamente, neste caso combinamos as entregas parciais dos filhos em $\lfloor \frac{D_j^s}{U} \rfloor$ entregas cheias e, possivelmente, uma entrega parcial com o restante da demanda, R_j^s. Na etapa de combinar as entregas parciais dos filhos em entregas cheias, devemos combiná-las na ordem em que cada filho foi processado na árvore (este detalhe será importante quando formos analisar o algoritmo). Por simplicidade, se um vértice j não tiver entrega agendada para um período s, definimos $D_j^s := 0$ e $R_j^s := 0$.

O pseudocódigo do procedimento de roteamento de entregas é dado pelo Algoritmo 4.

Algoritmo 4 Roteamento de entregas para o *Capacitated Splittable Tree* JRP:

para cada período s de 1 até T **faça**
  para cada nó $j \in V(\mathcal{T})$ em pós-ordem **faça**
    se j é folha **então**
      Crie $\lfloor \frac{D_j^s}{U} \rfloor$ entregas cheias e guarde a demanda residual em R_j^s
    senão
      $D_j^s \leftarrow 0$
      para cada filho ℓ de j **faça**
        $D_j^s \leftarrow D_j^s + R_\ell^s$
      Crie $\lfloor \frac{D_j^s}{U} \rfloor$ entregas cheias e guarde a demanda residual em R_j^s
  se $R_r^s > 0$ **então**
    Crie uma entrega para a demanda residual em R_r^s

Agora analisamos a viabilidade da solução gerada pelo algoritmo. Para isso, precisamos demonstrar que o algoritmo de arredondamento de Cheung et al., quando aplicado ao programa (CT-JRP) (na etapa de agendamento de entregas), visita cada cliente i antes de cada demanda não nula, i.e., se $d_{it} > 0$, então existe uma entrega em i em um período s com $s \leq t$. Para isso, podemos utilizar o mesmo resultado do Lema 9 do *Tree* JRP. É necessário provar que a solução ótima da formulação (CT-JRP) possui a mesma propriedade da solução ótima da formulação (T-JRP) com relação as variáveis y_s^j, assim como foi considerado na prova do Lema 9. Para a formulação (CT-JRP), a prova da propriedade é ligeiramente mais complicada que a prova original, como segue:

Lema 12. *Existe uma solução ótima* (x, y) *para a relaxação linear de* (CT-JRP) *tal que, para cada par de vértices* j *e* j' *com* $j' \in \text{path}(j)$ *e para cada* $s \in \{1, 2, \ldots, T\}$*, vale* $y_s^j \leq y_s^{j'}$.

Demonstração. Considere uma solução viável tal que $y_s^j > y_s^{j'}$. Vamos provar que é possível diminuir a variável y_s^j para $y_s^{j'}$ e manter a viabilidade da solução. Como reduzir o valor de y_s^j só pode diminuir o valor da função-objetivo, isso é suficiente para demonstrar o lema.

Como $j' \in path(j)$, temos $\mathcal{T}_j \subseteq \mathcal{T}_{j'}$, daí $\sum_{i \in V(\mathcal{T}_j) \cap N} \sum_{t=s}^{T} \frac{d_{it}}{U} x_{st}^i \leq \sum_{i \in V(\mathcal{T}_{j'}) \cap N} \sum_{t=s}^{T} \frac{d_{it}}{U} x_{st}^i$. Como o segundo termo é limitado por $y_s^{j'}$, temos que $\sum_{i \in V(\mathcal{T}_j) \cap N} \sum_{t=s}^{T} \frac{d_{it}}{U} x_{st}^i \leq y_s^{j'}$. Portanto, a restrição (6.9) continua satisfeita se substituirmos o valor de y_u^j por $y_u^{j'}$.

Além disso, como $\mathcal{T}_j \subseteq \mathcal{T}_{j'}$, sempre que temos uma restrição $x_{st}^i \leq y_s^j$ da forma (6.8) em que y_s^j aparece, também temos uma restrição da forma $x_{st}^i \leq y_s^{j'}$. Mas como $y_s^j > y_s^{j'}$, podemos diminuir o valor de y_s^j para $y_s^{j'}$ de forma que todas as desigualdades continuem satisfeitas. □

O próximo lema (que é uma adaptação do Lema 9 de Cheung et al.) afirma que há, pelo menos, uma entrega agendada em cada intervalo cujo número fracionário de demandas acumulado seja superior a um. A prova desse lema é idêntica à prova do Lema 9, com a exceção de que utilizamos o Lema 12 para provar a propriedade da solução ótima da formulação (CT-JRP).

Lema 13. *Seja j um vértice de $\mathcal{T}$ e $[s,t]$ um intervalo de períodos tal que $\sum_{u=s}^{t} y_u^j \geq 1$. O algoritmo de agendamento cria uma entrega no vértice j em algum período em $[s,t]$.*

Agora, considere uma demanda $(i,t) \in \mathcal{D}$. De acordo com as restrições (6.7) e (6.8), concluímos que $\sum_{s=1}^{t} y_s^j \geq 1$. Logo, pelo Lema 13, existe uma entrega para i em algum período em $[1,t]$, o que implica que todas as demandas são satisfeitas antes dos seus períodos de expiração. Isso é suficiente para demonstrar que o algoritmo cria uma solução viável, já que cada entrega criada pelo algoritmo de roteamento em um período s que passa por um vértice j serve todas as demandas de clientes descendentes de j de períodos a partir de s e até a próxima entrega. Além disso, toda entrega criada respeita o limite de capacidade de U itens.

Em seguida, analisaremos o custo da solução gerada. Para limitar o custo de armazenamento, comparamos com o valor ótimo do problema dual, da mesma forma como foi realizado no trabalho de Cheung et al. Para isso, considere o problema dual da relaxação de (CT-JRP):

(D-CT-JRP)

$$\max \sum_{(i,t)\in\mathcal{D}} B_t^i$$

$$\text{s.a.} \quad B_t^i \leq H_{st}^i + \sum_{j\in path(i)} L_{st}^{ij} + \sum_{j:i\in V(\mathcal{T}_j)} \frac{d_{it}}{U} M_s^j, \qquad (i,t)\in\mathcal{D}, s\leq t \tag{6.11}$$

$$\sum_{i:j\in path(i)} \sum_{t=s}^{T} L_{st}^{ij} \leq K^{(j)} - M_s^j, \qquad s\leq T, j\in V(\mathcal{T}) \tag{6.12}$$

$$L_{st}^{ij} \geq 0, M_s^j \geq 0. \qquad (i,t)\in\mathcal{D}, s\leq t, j\in V(\mathcal{T}) \tag{6.13}$$

Usando folgas complementares e o Lema 13, podemos limitar o custo de armazenamento da solução gerada da mesma maneira como foi realizada no Lema 10. Dessa forma, o seguinte lema segue:

Lema 14. *O custo de armazenamento é de no máximo $\sum_{(i,t)\in\mathcal{D}} B_t^i$.*

A análise do custo de entrega é mais elaborada. Observe que o algoritmo de agendamento implica que o número de períodos em que uma entrega é agendada é $\lfloor \sum_{u=1}^{T} y_u^r \rfloor$ para a raiz e no máximo $2\lfloor \sum_{u=1}^{T} y_u^j \rfloor$ para os demais vértices. No entanto, como uma entrega tem capacidade limitada, um vértice pode ser visitado múltiplas vezes no mesmo período. Estimamos o número de vezes que um vértice é visitado a partir do Lema 15. A ideia do lema é limitar a quantidade de vezes que cada vértice j é visitado em função do menor número de entregas cheias possíveis de acordo com as demandas acumuladas associadas à subárvore de j.

Lema 15. *Seja n_j^s o número de vezes que um vértice j é visitado pelo algoritmo no período s. Para todo $j \in V(\mathcal{T})$, vale $n_j^s \leq \left\lfloor \sum_{i\in V(\mathcal{T}_j)\cap N} \frac{D_i^s}{U} \right\rfloor + 2$.*

Demonstração. Seja $\bar{n}_j^s$ o número de entregas cheias que passam por j no período s no momento imediatamente após processarmos o vértice j no algoritmo. Iremos mostrar por indução na altura do nó j que $\bar{n}_j^s = \left\lfloor \sum_{i\in V(\mathcal{T}_j)\cap N} \frac{D_i^s}{U} \right\rfloor$. Quando j é uma folha, temos que

$\left\lfloor \sum_{i \in V(\mathcal{T}_j) \cap N} \frac{D_i^s}{U} \right\rfloor = \left\lfloor \frac{D_j^s}{U} \right\rfloor$, então, neste caso, o lema vale pela construção do algoritmo de roteamento.

Considere agora um vértice interno j com filhos $\mathcal{C}$. Relembre que cada entrega cheia que passa por um nó filho também passa por j e que todas as entregas parciais dos nós filhos são combinadas para formar $\left\lfloor \frac{D_j^s}{U} \right\rfloor$ entregas cheias e, potencialmente, mais uma entrega parcial com R_j^s itens. Portanto, o número de entregas cheias que passa por j no período s é

$$\begin{aligned}
\bar{n}_j^s &= \sum_{\ell \in \mathcal{C}} n_\ell^s + \left\lfloor \frac{D_j^s}{U} \right\rfloor \\
&= \sum_{\ell \in \mathcal{C}} \left\lfloor \sum_{i \in V(\mathcal{T}_\ell) \cap N} \frac{D_i^s}{U} \right\rfloor + \left\lfloor \frac{D_j^s}{U} \right\rfloor, \\
&= \sum_{\ell \in \mathcal{C}} \left\lfloor \sum_{i \in V(\mathcal{T}_\ell) \cap N} \frac{D_i^s}{U} \right\rfloor + \left\lfloor \frac{\sum_{\ell \in \mathcal{C}} R_\ell^s}{U} \right\rfloor,
\end{aligned}$$

onde a penúltima igualdade vale pela hipótese de indução e a última igualdade vale pela definição de D_j^s. Multiplicando os dois lados por U, obtemos

$$\begin{aligned}
U \cdot \bar{n}_j^s &= \sum_{\ell \in \mathcal{C}} U \cdot \left\lfloor \sum_{i \in V(\mathcal{T}_\ell) \cap N} \frac{D_i^s}{U} \right\rfloor + U \cdot \left\lfloor \frac{\sum_{\ell \in \mathcal{C}} R_\ell^s}{U} \right\rfloor \\
&= \sum_{\ell \in \mathcal{C}} \left(\sum_{i \in V(\mathcal{T}_\ell) \cap N} D_i^s - R_\ell^s \right) + (\sum_{\ell \in \mathcal{C}} R_\ell^s - R_j^s) \\
&= \sum_{i \in V(\mathcal{T}_j) \cap N} D_i^s - R_j^s = U \cdot \left\lfloor \frac{\sum_{i \in V(\mathcal{T}_j) \cap N} D_i^s}{U} \right\rfloor.
\end{aligned}$$

Isso completa a indução.

Além das entregas cheias que passam por j criadas até a iteração que processamos j no algoritmo de roteamento, podemos ter entregas parciais para entregar os R_j^s itens restantes para clientes descendentes de j que serão combinados com outras demandas residuais em algum vértice posterior a j com relação ao percurso da árvore realizado pelo algoritmo. Contudo, dado que combinamos as demandas residuais dos filhos na ordem do percurso em pós-ordem utilizado pelo algoritmo, temos que os R_j^s itens restantes de j estão divididos em no máximo duas entregas. Portanto, $n_j^s \leq \bar{n}_j^s + 2$ e o lema segue. □

Com o auxílio do Lema 15, é possível provar o seguinte resultado:

Lema 16. *O custo de entrega é no máximo* $5 \sum_{j \in V(\mathcal{T})} \sum_{s=1}^{T} K^{(j)} y_s^j$.

Demonstração. Fixe um vértice j e defina $\mathcal{S}_j$ como o conjunto de períodos em que é realizada alguma entrega j. Utilizando o resultado do Lema 15, o número total de entregas

que incluem j, diga-se n_j, é limitado por

$$n_j \leq \sum_{s\in\mathcal{S}_j}\left(\sum_{i\in V(\mathcal{T}_j)\cap N}\frac{D_i^s}{U}+2\right)=\sum_{i\in V(\mathcal{T}_j)\cap N}\sum_{t=1}^{T}\frac{d_{it}}{U}+2|\mathcal{S}_j|, \tag{6.14}$$

onde a igualdade vale por construção do algoritmo de roteamento e pode ser verificada usando indução na altura de j em $\mathcal{T}$. Observe que por construção do algoritmo de agendamento temos que

$$2|\mathcal{S}_j| \leq 2\cdot 2\left\lfloor\sum_{s=1}^{T} y_s^j\right\rfloor \leq 4\sum_{s=1}^{T} y_s^j.$$

Então resta limitar o valor do termo de (6.14) com a soma. Somando-se todas as restrições do tipo (6.9) do programa linear para todo t e utilizando $\sum_{s=1}^{t} x_{st}^i = 1$ pela restrição (6.8),

$$\begin{aligned}\sum_{s=1}^{T} y_s^j &\geq \sum_{s=1}^{T}\sum_{i\in V(\mathcal{T}_j)\cap N}\sum_{t=s}^{T}\frac{d_{it}}{U}x_{st}^i\\ &= \sum_{i\in V(\mathcal{T}_j)\cap N}\sum_{t=1}^{T}\frac{d_{it}}{U}\sum_{s=1}^{t}x_{st}^i\\ &= \sum_{i\in V(\mathcal{T}_j)\cap N}\sum_{t=1}^{T}\frac{d_{it}}{U}.\end{aligned}$$

Juntando os dois limitantes e somando para todos os vértices, obtemos o lema. □

Como o valor ótimo do PL é um limitante para o ótimo, os Lemas 14 e 16 implicam:

Teorema 6. *Existe uma* 6*-aproximação para o Capacitated Splittable Tree JRP.*

Demonstração. Seja PL* o valor ótimo da relaxação linear de (CT-JRP). Somando os custos de armazenamento e de entregas dados pelos Lemas 14 e 16, o valor da solução é no máximo

$$\begin{aligned}&\sum_{(i,t)\in\mathcal{D}} B_t^i + 5\sum_{j\in V(\mathcal{T})}\sum_{s=1}^{T} K^{(j)}y_s^j\\ &= \mathrm{PL}^* + 5\sum_{j\in V(\mathcal{T})}\sum_{s=1}^{T} K^{(j)}y_s^j\\ &\leq \mathrm{PL}^* + 5\mathrm{PL}^* = 6\mathrm{PL}^*.\end{aligned}$$

□

6.3 Uma 5-aproximação para o Capacitated Unsplittable Tree JRP

O algoritmo de roteamento apresentado na subseção anterior permite que uma demanda d_{it} seja dividida em várias entregas, tornando o algoritmo válido apenas para a versão *Splittable* do *Capacitated Tree* JRP. A seguir, apresentamos uma adaptação do algoritmo de roteamento anterior de forma que toda demanda d_{it} seja entregue em uma única entrega.

Da mesma maneira que o algoritmo anterior, o novo algoritmo construirá todas as entregas simultaneamente de maneira *bottom-up*, ou seja, utilizando um percurso em *pós-ordem* da árvore.

Roteamento de entregas melhorado. Para uma folha i de $\mathcal{T}$, que corresponde a um cliente, seja s' o período da próxima entrega agendada para i, ou $T+1$ se não houver. Defina D_i^s como o conjunto de demandas acumuladas no período s, ou seja, defina $D_i^s = \{(i,t) \in \mathcal{D} : s \leq t \leq s'-1\}$. Em seguida, particione D_i^s em duas partes: $D_{i-}^s = \{t \in D_i^s : d_{it} \leq U/2\}$ e $D_{i+}^s = \{t \in D_i^s : d_{it} > U/2\}$. Para cada período t em D_{i+}^s, crie uma entrega parcial contendo toda a demanda d_{it}. Depois, atribua as demandas de cada período t em D_{i-}^s nas entregas parciais criadas antes iterativamente de tal forma que nenhuma demanda d_{it} seja dividida em duas entregas e que nenhuma entrega possua mais do que U itens (criando novas entregas parciais, caso seja necessário). É fácil observar que, no fim do procedimento, todas as entregas parciais (com exceção de no máximo uma) serão preenchidas com mais da metade da capacidade U. Por fim, crie entregas definitivas para as entregas parciais preenchidas com mais da metade da capacidade U e guarde as demandas da entrega parcial com menos da metade da capacidade U no conjunto R_i^s.

Para um vértice interno j, definimos D_j^s como a união de todos conjuntos das demandas residuais R_ℓ^s de seus filhos ℓ. Como no algoritmo anterior, também combinamos as entregas parciais dos filhos, embora neste caso consideramos que cada conjunto de demandas R_ℓ^s de um filho ℓ deva ser adicionado inteiramente em uma mesma entrega. Fazemos isso utilizando a mesma estratégia para um nó folha: criamos entregas parciais alocando demandas residuais iterativamente nas entregas parciais criadas anteriores ou criando uma nova entrega parcial, caso seja necessário. Novamente, criamos entregas definitivas para cada entrega parcial com mais da metade da capacidade U ocupada e guardamos as demandas da entrega parcial com menos da metade da capacidade U no conjunto R_j^s.

Após processar todos os vértices da árvore, criamos uma entrega definitiva para a entrega parcial com menos da metade da capacidade U obtida na raiz, se ela existir. Por simplicidade, se um vértice j não tiver entrega agendada para um período s, definimos $D_j^s := \emptyset$ e $R_j^s = \emptyset$.

A seguir, explicamos com mais detalhes a estratégia gulosa utilizada no algoritmo. Primeiro, descrevemos o procedimento para criar entregas na etapa que processamos os nós folhas da árvore. Considere o conjunto de entregas parciais criadas a partir do conjunto D_{i+}^s pelo algoritmo. Para cada período t em D_{i-}^s, adicione d_{it} na entrega parcial com a menor quantidade de itens, se a demanda couber nessa entrega parcial, caso contrário, crie uma nova entrega parcial contendo d_{it}. Um pseudocódigo deste procedimento é dado no Algoritmo 5.

O procedimento de criação de entregas para nós internos é análogo com relação ao Algoritmo 5, com a exceção de que processamos as demandas acumuladas de D_j^s agrupadas pelas demandas residuais R_ℓ^s de cada filho ℓ e por não haver a necessidade de particionar as demandas nos conjuntos D_{j+}^s e D_{j-}^s, dado que toda demanda residual R_ℓ^s já possui carregamento menor ou igual que $\frac{U}{2}$, pela construção do Algoritmo 5. O pseudocódigo do procedimento para nós internos é dado no Algoritmo 6.

Finalmente, o novo algoritmo de roteamento completo é dado no Algoritmo 7.

Algoritmo 5 Criação de entregas para nós folhas:

Entrada: Conjunto de demandas acumuladas D_i^s de uma folha i em s.
Saída: Conjunto de demandas R_i^s correspondente a demanda residual de i em s.

CriaEntregasFolhas(D_i^s)

$D_{i-}^s \leftarrow \{(i,t) \in D_i^s : d_{it} \leq \frac{U}{2}\}$
$D_{i+}^s \leftarrow \{(i,t) \in D_i^s : d_{it} > \frac{U}{2}\}$
para cada demanda $(i,t) \in D_{i+}^s$ **faça**
    Crie uma nova entrega parcial contendo a demanda (i,t)
para cada demanda $(i,t) \in D_{i-}^s$ **faça**
    $W \leftarrow$ quantidade de itens da entrega parcial menos carregada
    se $W + d_{it} \leq U$ **então**
        Adicione a demanda (i,t) na entrega parcial menos carregada
    senão
        Crie uma nova entrega parcial contendo a demanda (i,t)
$W \leftarrow$ quantidade de itens da entrega parcial menos carregada
se $W \leq \frac{U}{2}$ **então**
    $R_i^s \leftarrow$ demandas da entrega parcial menos carregada
senão
    $R_i^s \leftarrow \emptyset$
devolve R_i^s

Algoritmo 6 Criação de entregas para nós internos:

Entrada: Conjunto de demandas residuais D_j^s de um nó interno j em s.
Saída: Conjunto de demandas R_j^s correspondente a demanda residual de j em s.

CriaEntregasInternas(D_j^s)

Crie uma nova entrega parcial vazia
para cada filho ℓ de j **faça**
    $W \leftarrow$ quantidade de itens da entrega parcial menos carregada
    se $W + \sum_{(i,t) \in R_\ell^s \cap D_j^s} d_{it} \leq U$ **então**
        Adicione as demandas em R_ℓ^s na entrega parcial menos carregada
    senão
        Crie uma nova entrega parcial contendo as demandas em R_ℓ^s
$W \leftarrow$ quantidade de itens da entrega parcial menos carregada
se $W \leq \frac{U}{2}$ **então**
    $R_j^s \leftarrow$ demandas da entrega parcial menos carregada
senão
    $R_j^s \leftarrow \emptyset$
devolve R_j^s

Algoritmo 7 Roteamento de entregas para o *Capacitated Unsplittable Tree* JRP:

1: **para** cada período s de 1 até T **faça**
2: **para** cada nó $j \in V(\mathcal{T})$ em pós-ordem **faça**
3: **se** j é folha **então**
4: $R_j^s \leftarrow$ *CriaEntregasFolhas*(D_j^s)
5: **senão**
6: $D_j^s \leftarrow \emptyset$
7: **para** cada filho ℓ de j **faça**
8: $D_j^s \leftarrow D_j^s \cup R_\ell^s$
9: $R_j^s \leftarrow$ *CriaEntregasInternas*(D_j^s)
10: **se** $R_r^s \neq \emptyset$ **então**
11: Crie uma nova entrega parcial para as demandas em R_r^s

A análise do custo de entrega é semelhante àquela feita para a 6-aproximação. Mais uma vez, queremos estimar o número de vezes que um vértice é visitado em função do número mínimo de entregas cheias possíveis de acordo com as demandas associadas a subárvore de j. Contudo, desta vez, limitamos essa quantidade de visitas por duas vezes o número mínimo de entregas mais um, na forma do lema seguinte.

Lema 17. *Seja n_j^s o número de vezes que um vértice j é visitado pelo algoritmo no período s. Para todo $j \in V(\mathcal{T})$, vale $n_j^s \leq \lfloor 2 \sum_{i' \in V(\mathcal{T}_j) \cap N} \sum_{(i,t) \in D_{i'}^s} \frac{d_{it}}{U} - 2 \sum_{(i,t) \in R_j^s} \frac{d_{it}}{U} \rfloor + 1$.*

Demonstração. Seja $\bar{n}_j^s$ o número de entregas com carregamento acima de $\frac{U}{2}$ unidades que passam por j no período s no momento imediatamente após processarmos o vértice j no algoritmo. Também, defina $L_j^s = D_j^s \setminus R_j^s$, ou seja, L_j^s é o conjunto de demandas pertencentes às viagens com carregamento acima de $\frac{U}{2}$ criadas pela estratégia gulosa ao processar o vértice j no período s. Mostraremos por indução na altura do nó j que

$$\bar{n}_j^s \leq \lfloor 2 \sum_{k \in V(\mathcal{T}_j) \cap N} \sum_{(i,t) \in D_k^s} \frac{d_{it}}{U} - 2 \sum_{(i,t) \in R_j^s} \frac{d_{it}}{U} \rfloor.$$

Quando j é uma folha, temos que $\sum_{k \in V(\mathcal{T}_j) \cap N} \sum_{(i,t) \in D_k^s} \frac{d_{it}}{U} = \sum_{(i,t) \in D_j^s} \frac{d_{it}}{U}$. A quantidade de produtos enviada no período s por viagens com carregamento acima de $\frac{U}{2}$ é estritamente maior que $\bar{n}_j^s \cdot \frac{U}{2}$, ou seja, $\sum_{(i,t) \in L_j^s} d_{it} > \bar{n}_j^s \cdot \frac{U}{2}$. Como $\bar{n}_j^s$ é inteiro,

$$\bar{n}_j^s \leq \lfloor 2 \sum_{(i,t) \in L_j^s} \frac{d_{it}}{U} \rfloor = \lfloor 2 \sum_{(i,t) \in D_j^s} \frac{d_{it}}{U} - 2 \sum_{(i,t) \in R_j^s} \frac{d_{it}}{U} \rfloor.$$

Então, neste caso, o lema vale pela construção do algoritmo de roteamento.

Considere agora um vértice interno j com filhos $\mathcal{C}$. Relembre que cada entrega com carregamento acima de $\frac{U}{2}$ que passa por um nó filho também passa por j e que cada entrega com carregamento menor ou igual que $\frac{U}{2}$ dos nós filhos são combinadas para formar, digamos, $\bar{m}_j^s$ entregas com carregamento acima de $\frac{U}{2}$ e, potencialmente, mais uma entrega com carregamento menor ou igual que $\frac{U}{2}$ para os itens em R_j^s. A quantidade de produtos enviadas no período s por viagens com carregamento acima de $\frac{U}{2}$ originadas pela combinação das demandas residuais dos filhos é estritamente maior que $\bar{m}_j^s \cdot \frac{U}{2}$, ou seja,

$\sum_{(i,t)\in L_j^s} d_{it} > \bar{m}_j^s \cdot \frac{U}{2}$, logo, temos $\bar{m}_j^s < 2\sum_{(i,t)\in L_j^s} \frac{d_{it}}{U}$. Portanto, utilizando a hipótese de indução, podemos limitar o número de entregas com carregamento acima de $\frac{U}{2}$ que passam por j no período s como

$$\begin{aligned}
\bar{n}_j^s &= \sum_{\ell\in\mathcal{C}} \bar{n_\ell^s} + \bar{m_j^s} \\
&< \sum_{\ell\in\mathcal{C}} \left\lfloor 2 \sum_{k\in V(\mathcal{T}_\ell)\cap N} \sum_{(i,t)\in D_k^s} \frac{d_{it}}{U} - 2 \sum_{(i,t)\in R_\ell^s} \frac{d_{it}}{U} \right\rfloor + 2 \sum_{(i,t)\in L_j^s} \frac{d_{it}}{U} \\
&\leq 2 \sum_{\ell\in\mathcal{C}} \sum_{k\in V(\mathcal{T}_l)\cap N} \sum_{(i,t)\in D_k^s} \frac{d_{it}}{U} - 2 \sum_{\ell\in\mathcal{C}} \sum_{(i,t)\in R_\ell^s} \frac{d_{it}}{U} + 2 \left(\sum_{(i,t)\in D_j^s} \frac{d_{it}}{U} - \sum_{(i,t)\in R_j^s} \frac{d_{it}}{U} \right) \\
&= 2 \sum_{\ell\in\mathcal{C}} \sum_{k\in V(\mathcal{T}_\ell)\cap N} \sum_{(i,t)\in D_k^s} \frac{d_{it}}{U} - 2 \sum_{\ell\in\mathcal{C}} \sum_{(i,t)\in R_\ell^s} \frac{d_{it}}{U} + 2 \sum_{\ell\in\mathcal{C}} \sum_{(i,t)\in R_\ell^s} \frac{d_{it}}{U} - 2 \sum_{(i,t)\in R_j^s} \frac{d_{it}}{U} \\
&= 2 \sum_{k\in V(\mathcal{T}_j)\cap N} \sum_{(i,t)\in D_k^s} \frac{d_{it}}{U} - 2 \sum_{(i,t)\in R_j^s} \frac{d_{it}}{U},
\end{aligned}$$

onde a segunda desigualdade segue da definição de L_j^s e a segunda igualdade, da definição de D_j^s. Como $\bar{n}_j^s$ é inteiro, temos que $\bar{n}_j^s \leq \left\lfloor 2\sum_{k\in V(\mathcal{T}_j)\cap N} \sum_{(i,t)\in D_k^s} \frac{d_{it}}{U} - 2\sum_{(i,t)\in R_j^s} \frac{d_{it}}{U} \right\rfloor$, o que completa a indução.

Além das entregas com carregamento acima de $\frac{U}{2}$ que passam por j, podemos ter entregas parciais para entregar os itens das demandas em R_j^s restantes para clientes descendentes de j que serão combinados com outras demandas residuais em algum vértice processado depois de j no percurso da árvore realizado pelo algoritmo. Contudo, neste caso, garantimos que as demandas restantes em R_j^s são entregues em exatamente uma entrega, pela construção do algoritmo. Portanto, $n_j^s \leq \bar{n}_j^s + 1$ e o lema segue. □

Com o auxílio do Lema 17, é possível provar o seguinte resultado:

Lema 18. *O custo de entrega é no máximo* $4\sum_{j\in V(\mathcal{T})} \sum_{s=1}^{T} K^{(j)} y_s^j$.

Demonstração. Fixe um vértice j e seja $\mathcal{S}_j$ o conjunto de períodos em que é realizada alguma entrega em j. Utilizando o resultado do Lema 17, o número total de entregas que incluem j, diga-se n_j, é limitado por

$$\begin{aligned}
n_j &\leq \sum_{s\in\mathcal{S}_j} \left(\left\lfloor 2\sum_{i'\in V(\mathcal{T}_j)\cap N} \sum_{(i,t)\in D_{i'}^s} \frac{d_{it}}{U} - 2\sum_{(i,t)\in R_j^s} \frac{d_{it}}{U} \right\rfloor + 1 \right) \\
&\leq \sum_{s\in\mathcal{S}_j} \left(2\sum_{i'\in V(\mathcal{T}_j)\cap N} \sum_{(i,t)\in D_{i'}^s} \frac{d_{it}}{U} \right) + |\mathcal{S}_j| \\
&= 2\sum_{i\in V(\mathcal{T}_j)\cap N} \sum_{t=1}^{T} \frac{d_{it}}{U} + |\mathcal{S}_j|,
\end{aligned}$$

onde a igualdade vale por construção do algoritmo de roteamento e pode ser verificada usando indução na altura de j em $\mathcal{T}$. Limitando as parcelas do lado direito da desigualdade

acima de maneira análoga como na prova do Lema 16, obtemos

$$\begin{aligned} n_j &\leq 2 \sum_{i \in V(\mathcal{T}_j) \cap N} \sum_{t=1}^{T} \frac{d_{it}}{U} + |\mathcal{S}_j| \\ &\leq 2 \sum_{s=1}^{T} y_s^j + 2 \sum_{s=1}^{T} y_s^j \\ &= 4 \sum_{s=1}^{T} y_s^j. \end{aligned}$$

Uma vez que cada visita a um vértice j incorre em um custo de $K^{(j)}$, usando o limitante acima e somando para todos os vértices, obtemos o lema. □

Como o valor ótimo do PL é um limitante para o ótimo, os Lemas 14 e 18 implicam:

Teorema 7. *Existe uma* 5*-aproximação para o Capacitated Unsplittable Tree JRP.*

Visto que uma solução viável para o *Capacitated Unsplittable Tree* JRP também é uma solução viável para o *Capacitated Splittable Tree* JRP e dado que o valor ótimo da relaxação linear de (CT-JRP) também é um limitante inferior para o custo da solução ótima do *Capacitated Splittable Tree*, o resultado abaixo é direto:

Teorema 8. *Existe uma* 5*-aproximação para o Capacitated Splittable Tree JRP.*

Capítulo 7

Considerações finais

Nesta dissertação, estudamos o *Inventory Routing Problem* sob a perspectiva de algoritmos de aproximação. Observamos que a literatura para este problema é muito reduzida na área de algoritmos de aproximação, resumindo-se a aproximações sub-logarítmicas para a versão mais geral do IRP [59, 23] e algumas aproximações constantes para simplificações do IRP, mais especificamente, para uma versão periódica do IRP [40], para o SIRPFL [44, 25] e para o *Tree* JRP [30]. Uma questão em aberto para o IRP tradicional é se ele admite aproximação com fator constante, ou se a análise do algoritmo sub-logarítmico do melhor resultado encontrado é justa.

Nossa primeira contribuição é a melhora de um algoritmo de aproximação para o IAP, versão particular do SIRPFL com apenas um cliente e um depósito. Para a versão do IAP chamada *Capacitated Splittable* IAP, melhoramos a 3-aproximação de Jiao e Ravi [44] para um fator de 2,619. Como consequência direta desse resultado, reduzimos os melhores fatores de aproximação do *Capacitated Unsplittable* IAP de 6 para 4,562, do *Capacitated Splittable* SIRPFL de 3,236 para 2,905 e do *Capacitated Unsplittable* SIRPFL de 6,029 para 4,649.

Como o SIRPFL é o primeiro problema a estender o IRP de forma aproximada com vários depósitos (apesar de forma simplificada) e pelo IAP estar diretamente relacionado com o SIRPFL, acreditamos que o IAP merece uma maior atenção, uma vez que não existem outros trabalhos para o IAP além do realizado por Jiao e Ravi. Uma alternativa para encontrar melhores aproximações para o IAP é desenvolver novos algoritmos utilizando outras técnicas além do arredondamento de PL. Como a formulação matemática do IAP é relativamente simples, uma possível abordagem para construir um novo algoritmo de aproximação que acreditamos ser promissora é a técnica primal-dual. Outro ponto a se considerar, com relação ao nosso algoritmo melhorado, é verificar se a análise que fizemos para a nossa versão do algoritmo é justa.

Nossa segunda contribuição é o desenvolvimento dos dois primeiros algoritmos de aproximação para versões do *Tree* JRP com capacidades de entrega, mais especificamente, uma primeira 6-aproximação para o *Capacitated Splittable Tree* JRP e uma 5-aproximação para os problemas *Capacitated Splittable* e *Capacitated Unsplittable Tree* JRP. Nossas aproximações são baseadas na 3-aproximação de Cheung et al. para o *Tree* JRP sem capacidades [30] e, embora exista uma 2-aproximação para esse problema [60], não sabemos se é possível adaptar esse algoritmo para resolver a versão capacitada, assim como foi feito

com o algoritmo de Cheung et al.

Observamos que a nossa 5-aproximação vale tanto para a versão *Splittable* quanto para a *Unsplittable* e não sabemos se a segunda é mais fácil de aproximar do que a primeira. Uma vez que uma soluçao para a versão *Splittable* é menos restrita do que uma solução para a versão *Unsplittable*, uma possibilidade é projetar um algoritmo que explore o fato de uma demanda poder ser dividida em várias entregas. Outra opção é verificar se é possível refinar a análise feita para a 6-aproximação do *Capacitated Splittable Tree* JRP ou melhorar o algoritmo de roteamento, já que é razoável pensar que um algoritmo de roteamento que busca criar entregas cheias deveria proporcionar soluções melhores do que um algoritmo que não efetua entregas cheias pela impossibilidade de dividir uma demanda em várias entregas. Algumas variantes interessantes também podem ser consideradas, como uma versão de *Tree* JRP com vários depósitos e capacidades, assim como o SIRPFL, ou até mesmo uma versao de *Tree* JRP na ausência da árvore $\mathcal{T}$ como entrada, o que, por sua vez, aproximaria a versão tradicional do IRP.

Por fim, esperamos que este trabalho possa ajudar a aumentar o interesse nos *Inventory Routing Problems* e em algoritmos de aproximação para pesquisadores de pesquisa operacional e áreas afins, visto que ainda existem assuntos não resolvidos do problema na área de algoritmos de aproximação e diversas possibilidades de estudo iniciadas e levantadas neste trabalho.

Referências Bibliográficas

[1] Operation research letters webpage. URL: `https://www.journals.elsevier.com/operations-research-letters`, last accessed on 21/08/2021.

[2] Tamer F. Abdelmaguid, Maged M. Dessouky, and Fernando Ordóñez. Heuristic approaches for the inventory-routing problem with backlogging. *Computers & Industrial Engineering*, 56(4):1519 – 1534, 2009.

[3] Yossiri Adulyasak, Jean-François Cordeau, and Raf Jans. Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. *INFORMS Journal on Computing*, 26(1):103–120, 2014.

[4] Yossiri Adulyasak, Jean-François Cordeau, and Raf Jans. The production routing problem: A review of formulations and solution algorithms. *Computers & Operations Research*, 55:141 – 152, 2015.

[5] Miguel Alarcón and Lehilton Pedrosa. Uma aproximação para o problema de reabastecimento em conjunto com capacidades. In *Anais do VI Encontro de Teoria da Computação*, pages 9–12, Porto Alegre, RS, Brasil, 2021. SBC.

[6] Faisal Alkaabneh, Ali Diabat, and Huaizhu Oliver Gao. Benders decomposition for the inventory vehicle routing problem with perishable products and environmental costs. *Computers & Operations Research*, 113:104751, 2020.

[7] Aldair Alvarez, Pedro Munari, and Reinaldo Morabito. Iterated local search and simulated annealing algorithms for the inventory routing problem. *International Transactions in Operational Research*, 25(6):1785–1809, 2018.

[8] C. Archetti, Luca Bertazzi, Alain Hertz, and M.Grazia Speranza. A hybrid heuristic for an inventory routing problem. *INFORMS Journal on Computing*, 24, 02 2012.

[9] Claudia Archetti, Luca Bertazzi, Gilbert Laporte, and Maria Grazia Speranza. A Branch-and-Cut Algorithm for a Vendor-Managed Inventory-Routing Problem. *Transportation Science*, 41(3):382–391, August 2007.

[10] Claudia Archetti, Leandro C. Coelho, and M. Grazia Speranza. An exact algorithm for the inventory routing problem with logistic ratio. *Transportation Research Part E: Logistics and Transportation Review*, 131:96 – 107, 2019.

[11] Claudia Archetti and M. Grazia Speranza. The inventory routing problem: the value of integration. *International Transactions in Operational Research*, 23(3):393–407, 2016.

[12] Arshinder, Arun Kanda, and S.G. Deshmukh. Supply chain coordination: Perspectives, empirical studies and research directions. *International Journal of Production Economics*, 115(2):316–335, 2008. Institutional Perspectives on Supply Chain Management.

[13] Ronald Askin and Mingjun Xia. Hybrid heuristics for infinite period inventory routing problem. *12th IMHRC Proceedings*, 2012.

[14] N. A. B. Aziz and N. H. Mom. Genetic algorithm based approach for the mufti product multi period inventory routing problem. In *2007 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 1619–1623, Dec 2007.

[15] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. *Proc. of STOC*, 04 2004.

[16] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear Programming and Network Flows*. Wiley, 2009.

[17] Luca Becchetti, Peter Korteweg, Alberto Marchetti-Spaccamela, Martin Skutella, Leen Stougie, and Andrea Vitaletti. Latency constrained aggregation in sensor networks. In Yossi Azar and Thomas Erlebach, editors, *Algorithms – ESA 2006*, pages 88–99, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] Walter J. Bell, Louis M. Dalberto, Marshall L. Fisher, Arnold J. Greenfield, R. Jaikumar, Pradeep Kedia, Robert G. Mack, and Paul J. Prutzman. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces*, 13(6):4–23, 1983.

[19] J.F. BENDERS. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[20] Luca Bertazzi and M. Grazia Speranza. Inventory routing problems: an introduction. *EURO Journal on Transportation and Logistics*, 1(4):307–326, Dec 2012.

[21] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, and Jiří Sgall. Better approximation bounds for the joint replenishment problem, 2013.

[22] Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Łukasz Jeż, Jiri Sgall, Nguyen Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. *Operations Research*, 68, 07 2015.

[23] Thomas Bosman and Neil Olver. Improved approximation algorithms for inventory problems, 2019.

[24] Lawrence D. Burns, Randolph W. Hall, Dennis E. Blumenfeld, and Carlos F. Daganzo. Distribution strategies that minimize transportation and inventory costs. *Operations Research*, 33(3):469–490, 1985.

[25] Jarosław Byrka and Mateusz Lewandowski. Concave connection cost facility location and the star inventory routing problem, 2019.

[26] Ann Melissa Campbell and Martin W. P. Savelsbergh. A decomposition approach for the inventory-routing problem. *Transportation Science*, 38(4):488–502, 2004.

[27] Lap Mui Ann Chan, Awi Federgruen, and David Simchi-Levi. Probabilistic analyses and practical algorithms for inventory-routing models. *Operations Research*, 46(1):96–106, 1998.

[28] Moses. Charikar, Samir. Khuller, and Balaji. Raghavachari. Algorithms for capacitated vehicle routing. *SIAM Journal on Computing*, 31(3):665–682, 2001.

[29] Chun Cheng, Peng Yang, Mingyao Qi, and Louis-Martin Rousseau. Modeling a green inventory routing problem with a heterogeneous fleet. *Transportation Research Part E: Logistics and Transportation Review*, 97:97 – 112, 2017.

[30] Maurice Cheung, Adam Elmachtoub, Retsef Levi, and David Shmoys. The submodular joint replenishment problem. *Mathematical Programming*, 158:1–27, 07 2015.

[31] Leandro Coelho and Gilbert Laporte. The exact solution of several classes of inventory-routing problems. *Computers & Operations Research*, 40:558–565, 02 2013.

[32] Leandro C. Coelho, Jean-François Cordeau, and Gilbert Laporte. Thirty years of inventory routing. *Transportation Science*, 48(1):1–19, 2014.

[33] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

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

[35] Guy Desaulniers, Jørgen G. Rakke, and Leandro C. Coelho. A branch-price-and-cut algorithm for the inventory-routing problem. *Transportation Science*, 50(3):1060–1076, 2016.

[36] Moshe Dror, M. Ball, and Bruce Golden. A computational comparison of algorithms for the inventory routing problem. *Annals of Operations Research*, 4:3–23, 12 1985.

[37] Hugues Dubedout, Pierre Dejax, Nicoleta Neagu, and Thomas G. Yeung. A GRASP FOR REAL LIFE INVENTORY ROUTING PROBLEM: APPLICATION TO BULK GAS DISTRIBUTION. In *9th International Conference on Modeling, Optimization & SIMulation*, Bordeaux, France, June 2012.

[38] Awi Federgruen and Yu-Sheng Zheng. The joint replenishment problem with general joint cost structures. *Operations Research*, 40(2):384–403, 1992.

[39] Awi Federgruen and Paul Zipkin. A combined vehicle routing and inventory allocation problem. *Operations Research*, 32(5):1019–1037, 1984.

[40] Takuro Fukunaga, Afshin Nikzad, and R. Ravi. Deliver or hold: Approximation Algorithms for the Periodic Inventory Routing Problem. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 209–225, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[41] Inge Li Gørtz, Viswanath Nagarajan, and R. Ravi. Minimum makespan multi-vehicle dial-a-ride. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, pages 540–552, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[42] Oualid Guemri, Abdelghani Bekrar, Bouziane Beldjilali, and Damien Trentesaux. Grasp-based heuristic algorithm for the multi-product multi-vehicle inventory routing problem. *4OR*, 14, 04 2016.

[43] Abdelhalim Hiassat, Ali Diabat, and Iyad Rahwan. A genetic algorithm approach for location-inventory-routing problem with perishable products. *Journal of Manufacturing Systems*, 42:93 – 103, 2017.

[44] Yang Jiao and R. Ravi. Inventory routing problem with facility location. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R Salavatipour, editors, *Algorithms and Data Structures*, pages 452–465, Cham, 2019. Springer International Publishing.

[45] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. *CoRR*, abs/2007.01409, 2020.

[46] Ahmed Kheiri. Heuristic sequence selection for inventory routing problem. *Transportation Science*, 2019.

[47] Moutaz Khouja and Suresh Goyal. A review of the joint replenishment problem literature: 1989–2005. *European Journal of Operational Research*, 186(1):1 – 16, 2008.

[48] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2012.

[49] Jakob Krarup and Peter Mark Pruzan. The simple plant location problem: Survey and synthesis. *European Journal of Operational Research*, 12(1):36 – 81, 1983.

[50] Hoong Chuin Lau, Qizhang Liu, and Hirotaka Ono. Integrating local search and network flow to solve the inventory routing problem. In *Eighteenth National Conference on Artificial Intelligence*, page 9–14, USA, 2002. American Association for Artificial Intelligence.

[51] Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54:763–776, 04 2008.

[52] Retsef Levi, Robin Roundy, and David B. Shmoys. A constant approximation algorithm for the one-warehouse multi-retailer problem. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, page 365–374, USA, 2005. Society for Industrial and Applied Mathematics.

[53] Retsef Levi, Robin O. Roundy, and David B. Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31(2):267–284, 2006.

[54] Shu-Chu Liu and S.B. Lee. A two-phase heuristic method for the multi-depot location routing problem taking inventory control decisions into consideration. *International Journal of Advanced Manufacturing Technology*, 22:941–950, 12 2003.

[55] Waqar Malik, Sivakumar Rathinam, and Swaroop Darbha. An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Oper. Res. Lett.*, 35:747–753, 11 2007.

[56] Yannis Marinakis. *Location routing problem*, pages 1919–1925. Springer US, Boston, MA, 2009.

[57] Guillaume Marquès, Caroline Thierry, Jacques Lamothe, and Didier Gourc. A review of vendor managed inventory (vmi): from concept to processes. *Production Planning & Control*, 21(6):547–561, 2010.

[58] Dolores Romero Morales and H. Edwin Romeijn. *The Generalized Assignment Problem and Extensions*, pages 259–311. Springer US, Boston, MA, 2005.

[59] Viswanath Nagarajan and Cong Shi. Approximation algorithms for inventory problems with submodular or routing costs. *Mathematical Programming*, 160, 04 2015.

[60] Lehilton L. C. Pedrosa. *Approximation Algorithms for Facility Location Problems and Other Supply Chain Problems*. PhD thesis, Instituto de Compução, Unicamp, 2014.

[61] Edcarllos Santos, Luiz Satoru Ochi, Luidi Simonetti, and Pedro Henrique González. A hybrid heuristic based on iterated local search for multivehicle inventory routing problem. *Electronic Notes in Discrete Mathematics*, 52:197 – 204, 2016. INOC 2015 – 7th International Network Optimization Conference.

[62] Kazim Sari. On the benefits of cpfr and vmi: A comparative simulation study. *International Journal of Production Economics*, 113(2):575–586, 2008. Special Section on Advanced Modeling and Innovative Design of Supply Chain.

[63] Ana Sarmiento and Rakesh Nagi. A review of integrated analysis of production-distribution systems. *IIE Transactions*, 31:1061–1074, 11 1999.

[64] Mehdi Seifbarghy and Zahra Samadi. A tabu search-based heuristic for a new capacitated cyclic inventory routing problem. *Int. J. of Mathematics in Operational Research*, 6:491 – 504, 01 2014.

[65] Stella Sofianopoulou. A heuristic approach for an inventory routing problem with backorder decisions. *Lecture Notes in Management Science*, 6:49–57, July 2014.

[66] Oğuz Solyalı and Haldun Süral. A branch-and-cut algorithm using a strong formulation and an a priori tour-based heuristic for an inventory-routing problem. *Transportation Science*, 45:335–345, 08 2011.

[67] Anchalee Supithak. A tabu search algorithm for integrated inventory and vehicle routing problem in one depot and multicustomers distribution system. *2011 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 200–205, 2011.

[68] Harvey M. Wagner and Thomson M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5(1):89–96, 1958.

[69] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, Cambridge, 2011.