ARTHUR HENRIQUE SOUSA CRUZ

UM ESTUDO SOBRE O PROBLEMA DINÂMICO DE ROTEAMENTO DE VEÍCULOS

LAVRAS – MG

2022

ARTHUR HENRIQUE SOUSA CRUZ

UM ESTUDO SOBRE O PROBLEMA DINÂMICO DE ROTEAMENTO DE VEÍCULOS

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para a obtenção do título de Mestre.

Prof. Mayron César de Oliveira Moreira
Orientador

Profa. Franklina Maria Bragion de Toledo
Coorientadora

LAVRAS – MG
2022

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Cruz, Arthur Henrique Sousa.
Um estudo sobre o Problema Dinâmico de Roteamento de Veículos / Arthur Henrique Sousa Cruz. - 2022.
110 p. : il.

Dissertação (mestrado) - Universidade Federal de Lavras, 2022.
Orientador: Prof. Mayron César de Oliveira Moreira.
Bibliografia.

1. Problema Dinâmico de Coleta e Entrega com Janelas de Tempo. 2. Paradigma de Orientação à Objetos. 3. Heurística. 4. Metaheurística. I. Moreira, Mayron César de Oliveira. II. Toledo, Franklina Maria Bragion de. III. Título.

ARTHUR HENRIQUE SOUSA CRUZ

UM ESTUDO SOBRE O PROBLEMA DINÂMICO DE ROTEAMENTO DE VEÍCULOS

A STUDY ON THE DYNAMIC VEHICLE ROUTING PROBLEM

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para a obtenção do título de Mestre.

APROVADA em 30 de Agosto de 2022.

Prof. Dr. Mayron César de Oliveira Moreira	UFLA
Prof. Dr. Paulo Afonso Parreira Júnior	UFLA
Prof. Dr. Fábio Luiz Usberti	UNICAMP
Prof. Dra. Franklina Maria Bragion de Toledo	ICMC/USP,

gov.br Documento assinado digitalmente
MAYRON CESAR DE OLIVEIRA MOREIRA
Data: 13/12/2022 23:20:24-0300
Verifique em https://verificador.iti.br

Prof. Mayron César de Oliveira Moreira
Orientador

Profa. Franklina Maria Bragion de Toledo
Co-Orientadora

LAVRAS – MG
2022

AGRADECIMENTOS

Agradeço à minha família por sempre acreditar e apoiar minha jornada acadêmica.
Agradeço aos meus amigos que estiveram ao meu lado durante os bons e maus momentos.
Agradeço ao meu orientador, professor Dr. Mayron César de Oliveira Moreira e à minha co-orientadora, professora Dra. Franklina Maria Bragion de Toledo, pela paciência, incentivo e conselhos.
Agradeço à Universidade Federal de Lavras por disponibilizar recursos para a realização do trabalho.
Agradeço ao financiamento provido pela Fundação de Desenvolvimento Científico e Cultural - UFLA (FUNDECC) - acordo 032/2020
O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

RESUMO

Nos últimos anos, o volume de entregas diárias aumentou de forma significativa, demandando ferramentas de apoio a decisão baseadas em algoritmos de otimização. Uma das questões mais relevantes nesse cenário é a definição de rotas de entrega com o objetivo de minimizar custos. Este problema é um clássico da otimização combinatória, conhecido como o Problema de Roteamento de Veículos (PRV). O problema conta com variantes, entre elas o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT). No PDCEJT, os pedidos são recebidos ao longo do dia durante a designação de rotas. Além disso, um pacote deve ser coletado em um ponto e entregue a um destino diferente. Por fim, a coleta ou entrega de cada ponto deve ser realizada em um determinado intervalo de tempo. Outras variantes são derivadas desse problema, como o PDCEJT/UR, que conta com a classificação de pontos como rurais e urbanos. Este trabalho tem inspiração em um caso real de uma empresa de logistíca, e propõe a adaptação de um algoritmo da literatura para a solução do PDCEJT. Além da solução do problema, descreve-se metodologias que facilitam a generalização de soluções para diferentes variantes do PDCEJT. No intuito de exemplificar as abstrações feitas, propõe-se uma abordagem para resolver o PDCEJT/UR. Experimentos são realizados para o PDCEJT e para o PCEJT, em que extensões da heurística de solução implementada são comparadas. A melhor variação é utilizada para a resolução do PDCEJT/UR. Os resultados apontam que a solução para o PDCEJT e sua adaptação para o PDCEJT/UR são igualmente eficientes.

Palavras-chave: Problema Dinâmico de Coleta e Entrega com Janelas de Tempo. Paradigma de Orientação à Objetos. Heurística. Metaheurística.

ABSTRACT

The number of package deliveries has increased significantly in recent years, demanding optimization algorithms-based tools for decision support. In this scenario, minimizing costs while defining delivery routes is relevant. This problem is a classic for combinatorial, known as Vehicle Routing Problem (VRP). There are many variants in the literature for VRP, such as the Dynamic Pickup and Delivery Problem (DPDPTW). In DPDPTW, the requests arrive during the route's designation. Furthermore, a package must be picked up at one point and delivered to another. Finally, the attendance to a point must be within a time interval. The DPDPTW/UR is a variant derivated from this problem and classifies the points into rural or urban and restricts vehicle attendance. Inspired by a real-world problem, this work adapts a literature algorithm to solve the DPDPTW. Additionally, it proposes methods to simplify the solutions generalization. Further, the approach for the DPDPTW is adjusted for DPDPTW/UR as an example. Extensions of the implemented heuristics are compared in experiments with DPDPTW and its static version. Finally, the DPDPTW/UR is tested with the best variation, and the results show that the methods proposed for DPDPTW and DPDPTW/UR are equally efficient.

Keywords: Dynamic Pickup and Delivery Problem with Time Windows. Object Oriented Paradigm. Heuristic. Metaheuristic.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SÍMBOLOS

P	conjunto de pontos de coletas
n	número de pedidos
D	conjunto de pontos de entregas
N	conjunto de todos os pontos (incluindo depósito)
R	conjunto de pedidos
s_u	tempo de atendimento do ponto $u \in N$
$[e_u, l_u]$	janela de tempo do ponto $u \in N$
$\overline{h}$	tempo total de serviço
F	conjunto de veículos
m	número de veículos
t_{uv}	tempo de viagem entre os pontos $u, v \in N$
H	conjunto de períodos
b	número de períodos
τ	tempo de um período ($\frac{\overline{h}}{b}$)
R_j	pedidos recebidos durante o período $H_j \in H$
$\mathbb{S}$	solução do PDCEJT
$\mathbb{S}_f$	rota do veículo $f \in F$ da solução $\mathbb{S}$
$\overline{m}_f$	número de pontos de coleta e entrega de uma rota
$\mathbb{S}_{fk}$	*k-ésimo* ponto da rota do veículo $f \in F$ da solução $\mathbb{S}$
T	tempo para execução do algoritmo de solução
S_j	*j-ésima* solução parcial do PDCEJT
S_{jf}	rota do veículo $f \in F$ da solução parcial S_j
S_{jfk}	*k-ésimo* ponto da rota do veículo $f \in F$ da solução parcial S_j
$\overline{S}_j$	*j-ésimo* conjunto de rotas parciais do PDCEJT
$\overline{S}_{jf}$	rota do veículo $f \in F$ do conjunto de rotas parciais $\overline{S}_j$
$\underline{m}_f$	número de pontos de coleta e entrega de uma rota parcial f
$\overline{S}_{jfk}$	*k-ésimo* ponto da rota do veículo $f \in F$ da conjunto de rotas parciais $\overline{S}_j$
c_{uv}	custo de viagem entre os pedidos $u, v \in N$
$C(\mathbb{S}_f)$	custo da rota $\mathbb{S}_f$
$C(S_{jf})$	custo da rota S_{jf}
I_j	instância de entrada parcial gerada pelo método de solução
it	número de iterações da *matheurist*
itm	número de iterações sem melhoria da *matheuristic*
S	solução qualquer para um problema de roteamento
S^{PC}	solução obtida pelo PLIM da *matheuristic*
S^*	melhor solução da *matheuristic*
$\overline{R}$	pedidos já inseridos durante a heurística de construção
it_ages	número de perturbações realizadas na componente AGES

S'	solução após remoção de pedidos (componentes AGES e LNS)
$\mathbb{E}$	pilha utilizada pela componente AGES
$\rho[r]$	penalidades do pedido $r \in R$
$\bar{r}$	pedido removido $\mathbb{E}$
R'	pedidos removidos de uma solução S (utilizada na componente LNS)
it	numero de iterações sem melhoria da componente LNS
S^+	melhor solução obtida pela LNS
m^*	número mínimo de rotas para se atender todos os pedidos de uma instância
R^*	pedidos recebidos durante o intervalo $[(j-1)\tau, j\tau - T]$
W	número de melhores rotas a serem avaliadas na Heurística de Arrependimento W
$\mathbb{P}$	conjunto de boas rotas utilizado no PLIM para particionamento de conjuntos
Δ_{rf}	valor binário que indica se um pedido r está na rota f
δ_{MA}	probabilidade de uso da Mudança Aleatória como operação de perturbação
δ_{TA}	probabilidade de uso da Troca Aleatória como operação de perturbação
λ	porcentagem de enviesamento na Mudança Enviesada
$\mathscr{K}$	número de inserções utilizadas para a comparação da Heurística de Arrependimento
δ_{ME}	probabilidade de uso da Mudança Enviesada
μ	valor suficientemente pequeno
$\mathscr{P}_t$	rotas atendidas por um veículo do tipo t
$\mathbb{F}_t$	tamanho da frota do tipo de veículo $t \in \{1...\mathbb{T}\}$
R^-	subconjunto de pedidos fixos $(R^- \subset R)$
ω	distância limite do centro para pontos urbanos
ϕ	proporção de pontos rurais em uma instância qualquer

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, o volume de entregas diárias aumentou de forma significativa (ATTANASIO et al., 2007; GHIANI et al., 2009; BENDER; KALCSICS; MEYER, 2020). Diante desse novo cenário, a utilização de ferramentas de apoio à decisão baseadas em algoritmos de otimização são capazes de melhorar a competitividade de empresas de diferentes setores (VIDAL; LAPORTE; MATL, 2019), dada a minimização de custos e o gerenciamento eficaz de seus recursos.

Neste contexto, uma das questões relevantes é a definição de rotas de entrega eficientes. Esse problema foi modelado por Dantzig e Ramser (1959) e é conhecido na literatura como o Problema de Roteamento de Veículos (PRV). O PRV é um problema clássico de otimização combinatória e, devido às mudanças do mercado global, é muito estudado e tem ganhado diversas variações ao longo dos anos (LAPORTE, 2007; VIDAL; LAPORTE; MATL, 2019).

Em especial, devido ao comércio eletrônico, cada vez mais os clientes realizam pedidos de maneira dinâmica. Logo, o conjunto de demandas de entrega pode aumentar ao longo do período. Isso dificulta a adaptação de rotas já obtidas considerando os pedidos anteriores, arranjados de forma estática (GHIANI et al., 2009; ATTANASIO et al., 2007). Esse cenário levou ao surgimento de uma nova variação do PRV, o Problema de Roteamento Dinâmico de Veículos (PRDV), que lida com a adição de novos pedidos dinamicamente ao longo de uma jornada de trabalho (PILLAC et al., 2013; PSARAFTIS; WEN; KONTOVAS, 2016).

Dentro desse contexto dinâmico, encontra-se o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT), uma variante que adiciona a necessidade de atendimento de pedidos em que determinada carga deve ser coletada em um ponto e entregue em outro. O atendimento de cada ponto deve ser realizado dentro de uma determinado período, ou janela de tempo. Por se tratar de um problema dinâmico, informações relacionadas ao pedido são obtidas de forma dinâmica no decorrer do período de atendimento dos pontos. O objetivo é atender os pedidos com o menor custo possível (MITROVIĆ-MINIĆ; LAPORTE, 2004; HOLBORN, 2013).

Variantes do PDCEJT também podem ser encontradas na literatura (PILLAC et al., 2013; PSARAFTIS; WEN; KONTOVAS, 2016; Ojeda Rios et al., 2021; BERBEGLIA; CORDEAU; LAPORTE, 2010), porém casos reais pode-se ter um grande número de extensões ou problemas com restrições muito específicas, o que afasta o problema dos já reportados na lite-

ratura. A motivação para este estudo advêm de um projeto em cooperação com uma empresa da área de logística, que lida com uma grande variedade de restrições de problemas dinâmicos de roteamento.

O Problema Dinâmico de Coleta e Entrega com Janelas de Tempo e pontos Urbanos e Rurais (PDCEJT/UR) é um exemplo de problema com restrições baseadas na empresa. Essa variação do PDCEJT considera o atendimento de pedidos com pontos em zonas rurais. Essa restrição impede o atendimento de pontos da zona rural por determinados veículos, devido a impossibilidade de transitar por vias rurais. Essa restrição também pode ser usada para impedir veículos de atenderem certos pontos em zonas urbanas, por exemplo, em uma rua onde o acesso de veículos de grande porte é proibido.

O PDCEJT/UR, apesar de poder ser convertido para o PDCEJT com frotas heterogêneas, foi a inspiração para a busca de uma solução para o PDCEJT que seja flexível para suas variantes, assim como outras restrições presentes na empresa que não puderam ser encontradas na literatura.

Na sequência deste Capítulo, na Seção 1.1, apresenta-se motivação para o estudo. Em seguida, nas Seções 1.2 e 1.3 tem-se, respectivamente, os objetivos gerais e específicos. Por fim, na Seção 1.4, são sumarizados os principais pontos do capítulo e a estrutura do trabalho é apresentada.

1.1 Motivação

A literatura para problemas dinâmicos de roteamento baseados em casos reais é vasta (PILLAC et al., 2013; BERBEGLIA; CORDEAU; LAPORTE, 2010; Ojeda Rios et al., 2021) (FONSECA-GALINDO et al., 2020; MITROVIĆ-MINIĆ; LAPORTE, 2004; ULMER et al., 2021; HOLBORN, 2013). A empresa na qual este trabalho foi inspirado conta com a necessidade de atender problemas dinâmicos de roteamento de diferentes clientes e restrições. Há casos onde o número de restrições do problema é grande ou que as restrições são muito específicas e não podem ser encontradas na literatura.

O estudo de novos métodos de solução de problemas de roteamento é uma demanda frequente. Como problemas reais podem não estar presentes na literatura, a adaptação de métodos de solução pode se tornar um trabalho árduo. Os objetivos deste trabalho visam o desenvolvi-

mento de uma ferramenta para a solução de problemas dinâmicos em um cenário baseado na empresa.

1.2 Objetivos Gerais

Em casos reais de problemas dinâmicos de roteamento é importante que o método de solução permita adaptações das heurísticas para novas soluções, visto que as variações de problemas de roteamento que podem surgir são imprevisíveis.

Este trabalho visa avançar a pesquisa no rumo do desenvolvimento de ferramentas que são flexíveis para a solução de problemas dinâmicos de roteamento. Mais especificamente, busca-se responder a seguinte pergunta:

a) "É possível desenvolver uma implementação flexível que permita a adição de novas restrições ou métodos de solução para o PDCEJT?"

1.3 Objetivos Específicos

Para alcançar uma solução flexível para o PDCEJT, este trabalho busca alcançar os seguintes objetivos específicos:

a) **algoritmos**: propor a adaptação de uma heurística para o resolver o PDCEJT;
b) **abstração da solução**: apresentar uma metodologia baseada em orientação a objetos para abstrair os principais elementos de uma solução para problemas de roteamento;
c) **implementação**: detalhar como a implementação pode auxiliar na generalização do de métodos de solução.

1.4 Considerações

Este trabalho tem como foco de estudo o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT). O objetivo é o desenvolvimento de um método solução que seja adaptável para variações do problema. O método de solução foi baseado no proposto por Sartori e Buriol (2020) para a versão estática do problema, que considera que todos os pedidos já são conhecidos no início da execução do algoritmo.

Além da adaptação para o contexto dinâmico, foram propostas variações no método de construção e na utilização de uma das componentes da *matheuristic* de Sartori e Buriol (2020).

A implementação foi feita em *Python* e foram utilizados conceitos de Orientação a Objetos para generalizar a solução. Para exemplificar o uso da ferramenta, propõe-se a adaptação da solução para resolver o PDCEJT/UR.

Experimentos foram realizados com base em instâncias da literatura para o PDCEJT e PCEJT com variações no método de construção . A melhor combinação das componentes foi utilizada na solução do PDCEJT/UR e os resultados mostraram que o método de solução é tão eficiente para o PDCEJT/UR quanto o a adaptação proposta para o PDCEJT é para seu próprio contexto.

Na sequência deste trabalho, no Capítulo 2 apresenta-se o referencial teórico e trabalhos relacionados. O PDCEJT e seu cenário estudado neste trabalho é detalhado no Capítulo 3. No Capítulo 4, o método proposto por Sartori e Buriol (2020) e as adaptações feitas para o contexto dinâmico são explicados. No Capítulo 5 são discutidas as abstrações e técnicas de codificação utilizadas para a generalização da implementação. O Capítulo 6 reporta os experimentos práticos realizados. Por fim, o Capítulo 7 apresenta as considerações finais deste trabalho e as possibilidades para a sua continuação.

2 REFERENCIAL TEÓRICO

A revisão taxonômica de Elshaer e Awad (2020) mostra que o PRV proposto por Dantzig e Ramser (1959) deu origem a diversas variantes. Entre as mais estudadas, destacam-se aquelas compostas por restrições de janelas de tempo (SOLOMON, 1987) e coleta e entrega de pedidos (TOTH et al., 2014, Capítulo 6).

O Problema de Coleta e Entrega (PCE) é uma generalização do PRV, em que um pedido deve ser retirado de um ponto de coleta e deixado em um ponto de entrega. O problema clássico pode ser visto como um problema de coleta e entrega em que todos os pontos de coleta (ou entrega) são o depósito.

Comumente o PCE é acompanhado por restrições de Janela de Tempo (PCEJT), como pode ser visto nos trabalhos propostos por Ropke e Pisinger (2006) e Sartori e Buriol (2020). As janelas adicionam um período de tempo no qual o pedido deve ser coletado de (ou entregue a) um ponto. Este trabalho considera ambas as restrições, contudo, como descrito no Capítulo 3, o problema se encontra no contexto dinâmico.

A diferença do PDRV e suas variantes de contexto estático se dá pela presença de um ou mais pedidos que não são conhecidos antes do início do planejamento das rotas, o que torna a geração de rotas ainda mais complexa (PILLAC et al., 2013).

Visto que este trabalho tem como objetivo a resolução do Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT), o referencial apresentado está focado no contexto dinâmico de problemas de roteamento de veículos. Para mais detalhes sobre o problema estático recomenda-se os trabalhos de Laporte (2007), Laporte (2009), Toth et al. (2014), Vidal, Laporte e Matl (2019) e Elshaer e Awad (2020).

Laporte (2007) e Laporte (2009) apresentam os principais métodos exatos, heurísticos e meta-heurísticos para solução do PRV. Toth et al. (2014), por sua vez, descrevem métodos de solução presentes na literatura do PRV e de suas principais variações. Vidal, Laporte e Matl (2019) apontam a necessidade de se ter mais estudos envolvendo variantes do PRV proeminentes na atualidade. Por fim, Elshaer e Awad (2020) fazem uma revisão taxinômica de 299 trabalhos relacionados ao PRV e suas extensões.

O referencial foi levantado a partir da busca por artigos relacionados ao PDRV e variantes em revistas da área. Revisões (*Surveys*) também foram utilizadas para encontrar estudos

relacionados. Por fim, foram buscadas produções destacadas pelos trabalhos encontrados a partir dos dois métodos anteriores.

A seguir, apresenta-se na Seção 2.1 uma breve referencial para PRV em contexto dinâmico, enquanto a Seção 2.2 considera trabalhos que buscam resolver PDRV ou alguma de suas variantes. A Seção 2.3 aborda trabalhos relacionados especificamente à variante estudada neste trabalho: o PDCEJT. Por fim, a Seção 2.4 sumariza as principais informações reportadas no Capítulo 2.

2.1 Problema Dinâmico de Roteamento de Veículos

Como destacado por Psaraftis, Wen e Kontovas (2016), o Problema Dinâmico de Roteamento de Veículos (PDRV) e suas extensões têm ganhado uma maior relevância na literatura de problemas de logística.

Pillac et al. (2013) revisa trabalhos relacionados ao PDRV e os classifica a partir da evolução e qualidade da informação. A evolução da informação indica se o problema é dinâmico (informações mudam durante a execução) ou estático (toda a informação é conhecida anteriormente). Os autores focam seu estudo no caso dinâmico.

Quanto à qualidade da informação, Pillac et al. (2013) classificam os trabalhos como determinísticos ou estocástico. No caso do primeiro grupo, o método de solução se utiliza de informações estocásticas conhecidas como, por exemplo, um histórico de pedidos. Para o segundo grupo, por outro lado, não se tem dados históricos e as informações sobre novos pedidos são obtidas somente durante a execução das rotas (PILLAC et al., 2013).

Berbeglia, Cordeau e Laporte (2010) fazem uma revisão bibliográfica do Problema de Coleta e Entrega Dinâmico (PDCE), destacando os estudos relacionados ao PDCEJT. Os autores definem as principais estratégias para a resolução dessa classe de problemas. Entre elas, é citada a reotimização periódica (também contemplada nas revisões de Pillac et al. (2013) e Psaraftis, Wen e Kontovas (2016)), que consiste da utilização de algoritmos para o problema estático no início do horizonte de planejamento ou toda vez que um novo pedido é recebido. Heurísticas de inserção e busca local atualizam as rotas calculadas anteriormente.

A revisão taxonômica de Psaraftis, Wen e Kontovas (2016) classifica trabalhos relacionados a PDRV's de acordo com 11 critérios: contexto logístico, função objetivo, modo de transporte, restrições de capacidade do veículo, tamanho da frota, restrições de tempo, habili-

dade de rejeitar pedidos, tipo do problema, método de solução, natureza do elemento dinâmico e natureza da estocacidade. Além disso, os autores identificam um aumento significativo no contexto de roteamento dinâmico a partir dos anos 2.000. Outro ponto importante levantado pelos autores é a importância da evolução tecnológica em *Big Data*, roteamento de veículos elétricos, *Drones* e veículos não tripulados, indicando possibilidades para futuras pesquisas para o contexto dinâmico.

Ojeda Rios et al. (2021) analisam e classificam 80 novos trabalhos no contexto do PDRV, com duas novas categorias, além das utilizadas por Psaraftis, Wen e Kontovas (2016): modo (*online* ou *offline*) e aplicação. A análise revelou que 65% dos novos trabalhos abordam problemas dinâmicos e estocásticos, enquanto 35% estudam problemas dinâmicos determinísticos. Isso indica que recentemente é mais comum as empresas manterem dados históricos relacionados às incertezas dos seus respectivos problemas de roteamento. Assim como no trabalho de Psaraftis, Wen e Kontovas (2016), os autores apontam o crescimento no número de estudos em roteamento dinâmico. Também é evidenciado que a maior parte dos algoritmos propostos são adaptações de variações estáticas do problema e indicam oportunidades de pesquisas futuras relacionadas à Mineração de Dados, *Big Data*, Modelos Preditivos e Aprendizado de Máquina.

Este trabalho se inspira em uma parceria de pesquisa com uma empresa que atua na área de consultoria logística. Devido às questões contratuais, não foi possível a utilização de dados históricos de rotas de clientes. No entanto, foi possível caracterizar or problema estudado como um PDRV que considera uma abordagem dinâmica e determinística. Por esse motivo, a revisão bibliográfica apresentada nas seções a seguir não contém detalhes sobre problemas de roteamento dinâmicos estocásticos. Para mais sobre esta classe de problemas, recomenda-se a leitura de Pillac et al. (2013, Seção 4.2), de Berbeglia, Cordeau e Laporte (2010) e de Ojeda Rios et al. (2021). Os estudos de Ritzinger, Puchinger e Hartl (2016), Ulmer et al. (2021) e Côté et al. (2021) também se destacam no contexto dinâmico e estocástico.

2.2 Métodos de Solução para o Problema Dinâmico de Roteamento de Veículos

Psaraftis (1980) foi um dos primeiros autores a abordar o Problema Dinâmico de Roteamento de Veículos (PDRV) determinístico. Seu trabalho foca em solucionar o *Dynamic Dial-A-Ride Problem* (DDARP), que consiste em designar rotas para usuários com origem (ponto de coleta) e destino (ponto de entrega). O estudo introduz o conceito de pedidos imediatos, que

considera a satisfação do cliente quanto ao tempo de espera do atendimento. O autor utiliza a estratégia de reotimizações periódicas através um método baseado em programação dinâmica. A desvantagem dessa abordagem consiste em sua escalabilidade na atualização da rota de dos clientes antes já designados a um veículo.

Chen, Chen e Gao (2017) aplicam uma adaptação da meta-heurística *Monarch Butterfly Optimization* (WANG; DEB; CUI, 2019) para o PDRV com soluções iniciais factíveis sendo geradas de forma aleatória. O trabalho aborda o contexto dinâmico com reotimizações periódicas. Uma simulação foi realizada com instâncias da literatura contendo até 199 clientes, e o algoritmo foi capaz de encontrar 12 novas melhores soluções. Okulewicz e Mańdziuk (2017) propõem uma variação da meta-heurística *Particle Swarm Optimization* (KENNEDY; EBERHART, 1995) para o mesmo problema. No trabalho, são resolvidas instâncias clássicas com até 385 vértices. Para 18 das 22 instâncias foram encontradas soluções com resultados médios melhores que a literatura.

AbdAllah, Essam e Sarker (2017) desenvolvem uma adaptação de um Algoritmo Genético (REEVES, 2010) para o PDRV. O algoritmo foi testado com instâncias da literatura de até 150 clientes. A abordagem é competitiva com outros métodos propostos na literatura, visto que melhorou 17 soluções já reportadas anteriormente.

Sabar et al. (2021) propõem uma Busca Local Iterada (ILS, do inglês *Iterated Local Search*) para solucionar o PDRV. A meta-heurística utiliza como busca local uma Busca Descendente em Vizinhança Distorcida (SVND, do inglês *Skewed Variable Neighborhood Descent*) (HANSEN et al., 2010) com 7 operadores diferentes. A perturbação é feita com operadores múltiplos, tendo foco de exploração do espaço de busca no início da execução, porém afunilando as perturbações em espaços de busca menores com o decorrer do tempo. Soluções de maior qualidade sempre são aceitas, e soluções de qualidade mais baixas tem probabilidade de aceitação calculada com base no critério de Monte-Carlo Exponencial (MCE) (AYOB; KENDALL, 2003). Experimentos práticos são realizados com 21 instâncias da literatura e o método proposto encontra novas melhores soluções conhecidas para 17 delas.

Abdirad, Krishnan e Gupta (2020) trabalham com o PDRV com frota homogênea limitada. Para tratar o problema, é proposta uma abordagem de dois estágios. No primeiro estágio, são executadas três heurísticas construtivas para a instância agrupada geograficamente por meio de um algoritmo de *clusterização*, resultando em diferentes soluções. Na sequência, aplica-se

três meta-heurísticas, desconsiderando os agrupamentos, nas distintas soluções para se obter uma nova solução. As heurísticas construtivas utilizadas são a *Path Cheapest Arc*, que expande a solução escolhendo o ponto mais próximo para inserir na rota; *Global Cheapest Arc*, que adiciona o ponto cuja distância é minimal. As meta-heurísticas são Busca Tabu, Busca Local Guiada e Recozimento Simulado. Nota-se que todos os métodos usados são clássicos na literatura de roteamento de veículos. O resultado final do algoritmo é a combinação entre a primeira e segunda etapa que obteve melhor resultado. Caso exista a demanda de um novo cliente, o método é reexecutado à partir da construção com as posições dos veículos atualizadas. Testes com instâncias de até 200 clientes foram realizados, com parte dos pedidos sendo estáticos. Nos resultados, foi possível observar que diferentes combinações entre heurísticas e meta-heurísticas se adéquam a instâncias de diferentes tamanhos.

Gendreau et al. (1999) abordam a variante do PDRV que considera Janelas de Tempo (PDRVJT). O algoritmo proposto soluciona o problema utilizando reotimizações contínuas, mantendo as melhores soluções em um conjunto de soluções que é atualizado ao longo da busca. Sempre que há um novo pedido, uma Busca Tabu paralelizada é executada, e as rotas armazenadas no conjunto de soluções são utilizadas como solução inicial para cada *thread* da meta-heurística.

Kilby, Prosser e Shaw (1998) e Montemanni et al. (2005) abordam duas variantes do PDRV de forma semelhante: o dia é dividido em intervalos de tempo (períodos), e as atualizações das rotas são feitas ao fim de cada um desses intervalos. Kilby, Prosser e Shaw (1998) consideram o Problema de Coleta e Entrega Dinâmico (PDCE), em que a coleta e entrega são tratadas separadamente. Os autores propuseram um método heurístico baseado na ideia de inserção e melhoria para tratar o problema ao fim de cada período. Instâncias com até 100 pedidos baseados em trabalhos da literatura do PRV estático foram apresentadas. Os autores analisaram como a alteração de parâmetros relacionados a dinamicidade das requisições podem impactar o desempenho do método.

Montemanni et al. (2005), por sua vez, buscam solucionar o PDRVJT. Além disso, é considerado que os veículos partem de um depósito, mas não precisam retornar a ele ao fim de sua rota. A proposta de solução consiste na utilização de um Algoritmo de Colônia de Formigas ao fim de cada intervalo de tempo. Experimentos computacionais com até 200 pedidos foram realizados, usando instâncias geradas com base na literatura e em instâncias já existentes. O

método proposto foi comparado uma implementação própria da Busca Gulosa Aleatorizada e Adaptativa (*GRASP*, do inglês *Greedy Randomized Adaptive Search Procedure*) (RESENDE; RIBEIRO, 2010), e obteve resultados melhores para a maioria das instâncias testadas.

Ninikas e Minis (2014) tratam do Problema de Roteamento de Veículos Dinâmico considerando *Linehauls* (clientes onde os veículos realizam entregas) e *Backhauls* (clientes onde são feitas coletas). Os autores consideram que os *Linehauls* e *Backhauls* podem ser percorridos em qualquer ordem, desde que suas janelas de tempo sejam respeitadas. É proposto um algoritmo *Branch-and-Price* para obtenção de soluções, além de um nova heurística de inserção baseada em geração de colunas para instâncias mais complexas. Tal abordagem obteve resultados próximos da otimalidade para instâncias da literatura com até 100 pedidos.

2.3 Métodos de Solução para o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo

No contexto dinâmico do Problema de Coleta e Entrega com Janelas de Tempo (PDCEJT), destaca-se o trabalho de Mitrović-Minić e Laporte (2004). Nele são estudadas estratégias de espera para o PDCEJT. Em problemas com janela de tempo, é possível que um veículo espere para que as janelas de tempo dos clientes sejam respeitadas. Essa espera pode ocorrer ao chegar em um cliente a ser atendido, ou logo após atender um cliente. Tais estratégias são denominadas, respectivamente, *drive-first* e *wait-first*. Algoritmos estáticos geralmente fazem uso da primeira, por ser mais vantajosa neste contexto. Contudo, para o contexto dinâmico, obtém-se rotas menores com a estratégia *wait-first*. Em contrapartida, há grande aumento no número de veículos necessários para atender aos clientes. Os autores desenvolvem métodos que balanceiam o uso de ambas as estratégias: a estratégia de espera dinâmica e a estratégia de espera dinâmica avançada. Testes foram realizadas para instâncias de até 1.000 pedidos. Ambas as abordagens mostraram maior eficiência quando comparadas às estratégias *drive-first* e *wait-first*.

Pankratz (2005b) resolve o PDCEJT com o algoritmo proposto em Pankratz (2005a) para a versão estática do problema. O método utilizado foi um *Grouping Genetic Algorithm* (GGA), uma meta-heurística que considera um grupo de pedidos como um único cromossomo, sendo ele atribuído a um veículo. Consequentemente, o número de veículos é igual ao número de cromossomos. Os cromossomos, contudo, não armazenam informações da rota do veículo,

e por isso uma heurística baseada em inserção é aplicada para obter a rota. Para a solução do problema dinâmico, o GGA foi combinado com uma estrutura de memória adaptativa, armazenando a população encontrada ao fim da execução do algoritmo genético. Sempre que um novo conjunto de pedidos chega, o seguinte processo é realizado: (i) uma cópia da população presente na memória adaptativa é efetuada; (ii) os pedidos são inseridos em todos os indivíduos na memória adaptativa; e (iii) o GGA é executado partindo da nova população. Para a avaliação do método foram geradas 5.600 instâncias baseadas na literatura, contendo de 50 a 53 pedidos, com diferentes graus de dinamicidade. O autor desenvolve duas heurísticas para fins de comparação com o GGA. Os resultados comprovam que o GGA obtém resultados melhores que os outros dois algoritmos.

O trabalho de Gendreau et al. (2006) também aborda o PDCEJT considerando as janelas de tempo como restrições fracas. A abordagem, assim como em Gendreau et al. (1999), utiliza de uma memória adaptativa, onde soluções com menores valores de função objetivo para pedidos já roteados são armazenados. Um novo pedido que chega é inserido em cada solução presente na memória, e a melhor solução resultante é submetida a uma busca local baseada em Cadeias de Ejeção (*ejection chain*). A memória adaptativa é atualizada e, até que haja um novo pedido ou que um serviço seja concluído, uma Busca Tabu executa continuamente sua etapa de melhoria a fim de aprimorar a solução atual.

O trabalho de Pureza e Laporte (2008) busca resolver o PDCEJT através de uma estratégia de espera e armazenamento (*wating and buffering*). Uma heurística construtiva-desconstrutiva gera rotas para o conjunto de pedidos não dinâmicos. Os pedidos dinâmicos são inseridos de acordo com o limite de sua janela de tempo, podendo ser criada, se necessário, uma nova rota. A solução é comparada utilizando instâncias de até 100 pedidos (estáticos e dinâmicos) geradas aleatoriamente. Os resultados mostram as vantagens de cada estrategia em termos de pedidos atendidos e número de veículos.

Holborn (2013) estuda o PDCEJT e estratégias para sua resolução. Após um levantamento de métodos da literatura para a resolução do problema, a autora propõe um algoritmo que utiliza Busca Tabu e uma heurística baseada em *Branch-and-Bound* para tratar o problema. A abordagem é comparada com o método de Mitrović-Minić e Laporte (2004), apresentando melhorias para maior parte das instâncias testadas. O algoritmo proposto foi adaptado para um problema baseado em um caso real no contexto de sistemas de saúde. Nesse caso, o objetivo é

planejar a coleta e a entrega de materiais e equipamentos. O algoritmo não consegue alcançar melhorias significativas nesse contexto, mas permitiu uma nova visão em relação a incorporação de pedidos dinâmicos nesse tipo de serviço, além de mostrar que há potencial para estudos de roteamento dinâmico no campo da saúde.

Ferrucci e Bock (2014) estudam o PDCEJT considerando as janelas de tempo como restrições fracas. É apresentado um controlador que adapta a rota enquanto ela é percorrida pelos veículos em tempo-real à medida em que surgem novos pedidos. Uma Busca Tabu é utilizada para rearranjar a rota. O problema tem como objetivo primário a minimização do atraso na realização dos serviços. Para testes, são feitas simulações em instâncias de 300 e 500 pedidos. As instâncias são testadas em três ambientes: fácil, normal e difícil, classificadas de acordo com o tamanho das janelas de tempo dos pedidos.

No trabalho de Karami, Vancroonenburg e Vanden Berghe (2020), os autores consideram o PDCEJT permitindo violações na janela de tempo, mas minimizando o atraso da entrega. Os autores propõem um modelo Linear Inteiro Misto para a versão estática do problema e o utilizam em uma abordagem de reotimizações. O estudo apresenta uma heurística de agendamento composta de dois passos: (i) realiza-se uma inserção mais barata de um novo pedido e, em seguida, (ii) uma busca local é aplicada.

Através de instâncias com até 180 pedidos, com diferentes graus de dinamicidade e níveis de urgência, foi feito um estudo do comportamento da heurística. Para comparação, o problema foi resolvido utilizando o modelo proposto considerando que as informações eram conhecidas com antecedência (como um problema estático). Os resultados mostram que o desempenho do algoritmo piora a medida que a dinamicidade aumenta e o nível de urgência diminui.

2.4 Considerações

Neste capítulo, foi apresentada uma revisão bibliográfica para o Problema de Roteamento de Veículos com foco no contexto dinâmico. Na Seção 2.1, foi apresentado um breve referencial ao PDRV. O problema estudado neste trabalho foi classificado como dinâmico e determinístico, seguindo os critérios definidos por Pillac et al. (2013).

Nas Seções 2.2 e 2.3, foram introduzidos trabalhos relacioandos ao PDRV e PDCEJT, respectivamente. Destacam-se os trabalhos de Mitrović-Minić e Laporte (2004) e de Holborn (2013), que estudam o PDCEJT em um contexto semelhante ao deste trabalho.

É importante perceber que, como apontado por Psaraftis, Wen e Kontovas (2016) e Ojeda Rios et al. (2021), após os anos 2.000, houve um grande aumento em trabalhos relacionados ao PDRV. As buscas por trabalhos relacionados ao PDRV realizadas neste trabalho reafirmam este fato, afinal, a grande maioria dos trabalhos descritos nas Seções 2.2 e 2.3 são de anos seguintes a 2.000.

Neste trabalho, pretende-se resolver o PDCEJT, porém utilizando de uma implementação flexível da *matheuristic* proposta por Sartori e Buriol (2020). O propósito da implementação é a facilitação de adaptações do método para problemas com restrições muito específicas que não são comuns ou não estão presentes na literatura, como por exemplo, a restrição de limitação da visita de veículos a pontos rurais e urbanos.

3 PROBLEMA ESTUDADO

Neste trabalho, é estudado o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT). Em linhas gerais, o problema consiste em planejar as rotas de veículos para coleta e entrega de encomendas, sendo que não se tem conhecimento de todos os pedidos no início do planejamento das rotas. O atendimento de cada pedido deve respeitar janelas de tempo pré-estabelecidas, dentro de um horário de serviço fixo. O objetivo consiste em minimizar a distância total percorrida pelos veículos.

As características do problema estudado são detalhadas na Seção 3.1 e um modelo matemático é descrito na Seção 3.2. A Seção 3.3 apresenta um exemplo para ilustrar o problema. Por fim, a Seção 3.4 sumariza as principais discussões do capítulo.

3.1 Definições

Sejam $P = \{1,2,...,n\}$ e $D = \{n+1,n+2,...,2n\}$ os conjuntos de pontos de coleta e de entrega, respectivamente. Considera-se $N = \{0\} \cup P \cup D$ o conjunto de todos os pontos do problema, em que 0 identifica o ponto de origem de todos os veículos. O conjunto de pedidos R é composto por tuplas $r = (i, i+n)$, em que $i \in P$ é o ponto de coleta e $i+n \in D$ é o ponto de entrega do pedido.

Define-se s_u como o tempo de serviço necessário para se realizar a tarefa (coleta ou entrega) no ponto $u \in N$, sendo $s_0 = 0$. O serviço de um ponto deve ser iniciado em um horário contido na janela de tempo $[e_u, l_u]$. Considera-se que a janela de tempo do ponto de origem se inicia no tempo 0 e termina no final do horário de serviço $\overline{h}$, ou seja, $e_0 = 0$ e $l_0 = \overline{h}$.

Os pedidos são atendidos por uma frota homogênea de veículos F, tal que $F = \{1,2...,m\}$. No contexto estudado, os objetos a serem coletados e entregues não têm volume suficiente para encher um veículo. Portanto, assume-se que a capacidade dos veículos é ilimitada. Um veículo demanda um tempo t_{uv} para viajar entre dois pontos $u, v \in N$.

Os pedidos chegam dinamicamente com o decorrer do tempo durante o horário de serviço $\overline{h}$. Portanto, para tratar o problema divide-se o horário de serviço em um conjunto de períodos iguais $H = \{H_1, H_2, ..., H_b\}$, ou seja, cada período tem dimensão $\tau = \frac{\overline{h}}{b}$. Logo, o primeiro período é dado por $H_1 = [0,\ \tau]$ e os períodos seguintes por $H_j = [(j-1)\tau, j\tau]$, com $j = 2,3,...,b$. Consequentemente, os pedidos são divididos em subconjuntos, ou seja, $R_j \subseteq R$ é

o conjunto de pedidos recebidos no período H_j, de forma que $R_b = R$. Assume-se que nenhum pedido é conhecido antes de H_1.

Estabelecidos o problema estudado e a forma de tratar os pedidos dinâmicos, define-se que uma solução $\mathbb{S}$ para o PDCEJT consiste em um conjunto de rotas $\mathbb{S} = \{\mathbb{S}_1, \mathbb{S}_2, ..., \mathbb{S}_m\}$. Considera-se $\mathbb{S}_f = [0, u_1, u_2, ..., u_{\overline{m}_f}, 0]$ como a rota do veículo f sendo $\overline{m}_f$ o número de pontos de coleta e entrega da rota f (desconsidera-se apenas o depósito). Utiliza-se da notação $\mathbb{S}_{fk}$ para indicar o *k-ésimo* ponto da rota $\mathbb{S}_f$. Uma solução factível para o problema deve respeitar as restrições apresentadas a seguir.

a) **precedência de coleta e entrega**: para um pedido $r = (i, i+n)$, o ponto de entrega $(i+n) \in D$ não pode ser visitado antes de seu correspondente ponto de coleta $i \in P$;
b) **origem e horário de serviço**: cada veículo $f \in F$ deve partir e retornar do ponto de origem 0 no intervalo de sua janela de tempo ($[0, \overline{h}]$);
c) **janelas de tempo**: O tempo de chegada do veículo f ao ponto $u \in N$ não pode exceder l_u. Caso o motorista do veículo f chegue antes de e_u, o mesmo deve esperar até e_u para realizar o atendimento do ponto;
d) **obrigatoriedade e exclusividade de visita**: para cada pedido r, os pontos $i, i+n \in N$ devem ser visitado exatamente uma vez;
e) **atendimento de pedido**: em um pedido $r = (i, i+n)$, a visita ao ponto $i \in N$ por um veículo $f \in F$ torna obrigatório que o atendimento ao ponto $i+n \in N$ seja feito pelo mesmo veículo.

Cada período H_j tem uma solução parcial S_j contendo rotas geradas com os pedidos recebidos até $j\tau - T$, ou seja, T instantes antes do final do período. O intervalo T é utilizado para gerar uma solução para o problema. Cada rota é representada por uma lista de pontos pelos quais o veículo $f \in F$ deve passar. As definições das rotas e dos pontos de uma rota de uma solução parcial S_j são análogas às de uma solução $\mathbb{S}$. Utiliza-se da notação S_{jf} para indicar a rota do veículo f e S_{jfk} a posição do ponto k na rota f. Se durante um período H_j um veículo $f \in F$ iniciar o atendimento do ponto de coleta $i \in P$, S_j e todas as soluções de todos os períodos subsequentes devem incluir o ponto de entrega $i+n \in D$.

Define-se $\overline{S}_j$ como o conjunto de rotas parciais do período H_j, com $j = 1, 2, ..., b-1$. Cada rota parcial $\overline{S}_{jf}$, com $f \in F$, contém apenas os pontos com atendimentos iniciados até o final do período H_j, de forma que $\overline{S}_{jf} = [0, u_1, u_2, ..., u_{\underline{m}_f}]$ e $S_{j-1f} = [0, u_1, u_2, ..., u_{\underline{m}_f}, ..., 0]$,

sendo $\underline{m}_f$ o número de pontos de coleta e entrega cujo atendimento foi inciado no período H_j. Analogamente à $\mathbb{S}_{jfk}$ e S_{jfk}, $\overline{S}_{jfk}$ representa o *k-ésimo* ponto de $\overline{S}_{jf}$. É importante explicitar que os pedidos com coletas não iniciadas ao fim de H_j podem ter sua rota modificada, ou seja, pares de coletas e entregas após $u_{\overline{m}_f}$ da rota S_{j-1f} podem ser removidos.

Uma solução S_j deve conter os pedidos que têm o atendimento iniciado até o final do período H_j acrescido dos pedidos recebidos até o tempo $(j)\tau - T$, ou seja, $S_{jf} = [0, u_1, u_2, ..., u_{\underline{m}_f}, v_1, v_2, ..., v_{\overline{m}_f - \underline{m}_f}, 0]$, sendo, $[v_1, v_2, ..., v_{\overline{m}_f - \underline{m}_f}, 0]$ a lista de pontos adicionadas à $\overline{S}_{jf}$ na rota S_{jf}. Tal lista pode ser vista como a nova continuação da rota do veículo após o final do período H_j.

Com as definições feitas, pode-se afirmar que a última solução parcial S_b contém todos os pedidos recebidos ao longo do horário de serviço $\overline{h}$, implicando que a solução final para o problema $\mathbb{S}$ será igual à S_b

Para o PDCEJT estudado, tenta-se minimizar a soma dos custos das rotas percorridas pelos veículos. O custo de uma rota é dado pela soma dos custos de viagem entre os pontos da rota, sendo que o custo de viagem entre dois pontos $u, v \in N$ é igual a c_{uv}. Portanto, o custo de uma solução qualquer, pode ser descrita como o somatório dos custos de suas rotas ($C(\mathbb{S}_f)$). Logo, define-se a seguinte função objetivo para o problema:

$$\min \sum_{f \in F} C(\mathbb{S}_f), \tag{3.1}$$

em que $C(\mathbb{S}_f) = \sum_{k=1}^{\overline{m}_f+1} c_{\mathbb{S}_{fk}, \mathbb{S}_{fk+1}}$ é o custo da rota $\mathbb{S}_f$, ou seja, o custo de uma rota é dado pela soma do custo de viagem entre os pontos que percorre. O custo de uma solução parcial S_j, é calculado de forma análoga, assim como o custo de uma de suas rotas $C(S_{jf})$. Além disso, Como $\mathbb{S} = S_b$, tem-se que o custo da solução $\mathbb{S}$ é igual o custa da última solução parcial S_b

O DPDPTW é NP-Difícil, já que o PRV é um caso especial do problema onde:

a) todos os pedidos chegam no instante inicial;
b) o ponto de coleta de todas as entregas é o depósito;
c) o tempo de serviço de todos os pontos é zero;
d) o horizonte de planejamento é suficientemente grande para que um único veículo atenda todos os pedidos;
e) todas as janelas de tempo iniciam no instante inicial e têm tamanho igual ao horizonte de planejamento.

3.2 Modelo Matemático para o Problema Estático

O PDCEJT estudado neste trabalho é resolvido por reotimizações periódicas, como detalhado no Capítulo 4. Portanto, o problema estático é resolvido a cada período considerando os pedidos recebidos até o tempo $j\tau - T$. A seguir, apresenta-se a modelagem matemática baseada em (ROPKE; PISINGER, 2006) do problema resolvido a cada período, sendo as variáveis definidas no Quadro 3.1.

$$\min \sum_{f \in F} \sum_{u \in N} \sum_{v \in N} c_{uv} x_{uvf} \tag{3.2}$$

sujeito à:

$$\sum_{f \in F} \sum_{u \in N \setminus \{0\}} \sum_{v \in N \setminus \{0\}} x_{uvf} = 1 \tag{3.3}$$

$$\sum_{u \in N} x_{0uf} = 1, \forall f \in F \tag{3.4}$$

$$\sum_{u \in N} x_{u0f} = 1, \forall f \in F \tag{3.5}$$

$$\sum_{u \in N \setminus \{0\}} x_{uvf} - \sum_{u \in N \setminus \{0\}} x_{vuf} = 0, \forall v \in N, \forall f \in F \tag{3.6}$$

$$\psi_{0f} = 0, \forall f \in F \tag{3.7}$$

$$\begin{gathered} x_{uvf} \implies \psi_{uf} + s_u + t_{uv} \leq \psi_{vf}, \\ \forall u \in N \setminus \{0\}, \forall v \in N \setminus \{0\}, \forall f \in F \end{gathered} \tag{3.8}$$

$$\begin{gathered} e_u \leq \psi_{uf} \leq l_u, \forall u \in N \setminus \{0\}, \\ \forall f \in \{i \mid i \in F, \sum_{v \in N} x_{uvi} + \sum_{v \in N} x_{vui} >= 1\} \end{gathered} \tag{3.9}$$

$$\psi_{uf} + s_u + t_{u0} \leq \overline{h}, \forall f \in F, u = \overline{S}_{jf\underline{m}_f} \tag{3.10}$$

$$\psi_{if} \leq \psi_{i+n,f}, i \in P, f \in F \tag{3.11}$$

$$x_{uvf} = 1, f \in F, u = \underline{S}_{jfk}, v \in \overline{S}_{jfk+1}, k = 1, ..., \underline{m}_f \tag{3.12}$$

$$x_{uvf} \in \{0,1\}, \forall u \in N, \forall v \in N, \forall f \in F \tag{3.13}$$

$$\psi_{uf} \in \mathbb{N}, \forall u \in N \forall f \in F \tag{3.14}$$

Fonte: Baseado em Ropke e Pisinger (2006)

Quadro 3.1 – Variáveis do modelo matemático

Variável	**Significado**
x_{uvf}	indica se o veículo f viaja do ponto u para o ponto v;
ψ_{uf}	instante de tempo que o veículo f chega no ponto u;

Fonte: Do autor (2022).

A função objetivo (3.2) calcula o custo total das rotas, sendo equivalente à função (3.1). As restrições (3.3) garantem que todos os pontos de coleta e entrega sejam percorridos pelo menos uma vez, enquanto as restrições (3.4) e (3.5) obrigam que o veículo f parta e retorne ao depósito. As restrições (3.6) fazem com que caminhos consecutivos sejam criados.

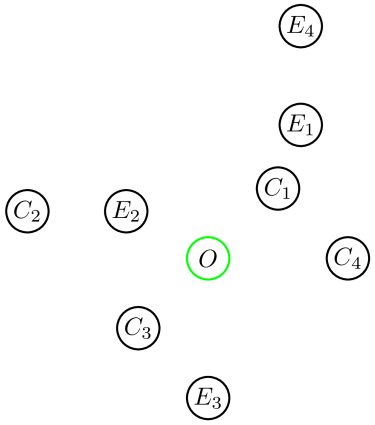
O tempo de chegada no ponto (ψ_{uf}) é calculado a partir das restrições (3.7) e (3.8), enquanto as restrições (3.9) limitam o tempo de chegada às janelas de tempo. As restrições (3.10) obrigam que o atendimento seja limitado ao horizonte de planejamento. As restrições (3.11) garantem que cada ponto de entrega seja atendido após seu ponto de coleta e impede que sejam criados subciclos. As restrições (3.12) forçam o atendimento de pedidos cuja coleta foi iniciada. Por fim, as restrições (3.13) e (3.14) definem, respectivamente, o espaço de busca das variáveis x_{uvf} e ψ_{uf}.

3.3 Exemplo Numérico

Para ilustrar a dinamicidade do PDCEJT, apresenta-se nesta seção um exemplo simplificado do problema. Considere que haja três períodos de tempo, ou seja, $H = \{H_1, H_2, H_3\}$ e um conjunto de pedidos $R = \{(C_1, E_1), (C_2, E_2), (C_3, E_3), (C_4, E_4), (C_5, E_5), (C_6, E_6), (C_7, E_7), (C_8, E_8), (C_9, E_9)\}$. Tem-se o ponto O como depósito de onde partirão 3 veículos.

Por simplicidade, considera-se que não há pedidos antes do primeiro período. Na Figura 3.1, são ilustrados os pedidos (C_1, E_1), (C_2, E_2), (C_3, E_3) e (C_4, E_4) que chegaram até tempo $\tau - T$, onde τ é o tempo de cada período e T o tempo necessário para definição das rotas. Durante o tempo T um algoritmo é executado para os pedidos disponíveis e as rotas de coleta e entrega são definidas. O resultado obtido pode ser visto na Figura 3.2.

Figura 3.1 – Pedidos recebidos em H_1 até o tempo $\tau - T$. Em verde o ponto de origem dos veículos.

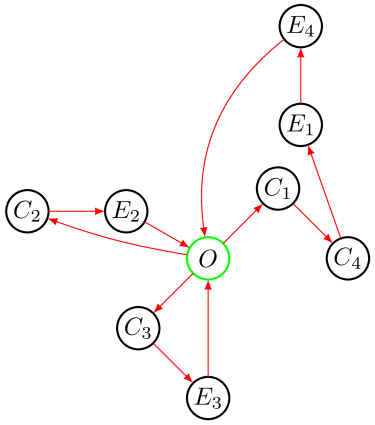


Fonte: Do autor (2022).

Após o período H_1 os veículos iniciam as entregas seguindo as rotas planejadas enquanto novos pedidos são recebidos. Na Figura 3.3, tem-se os novos pedidos (em azul) (C_5, E_5) e (C_6, E_6) que chegaram até o tempo $2\tau - T$, ou seja, antes do início da definição das novas rotas. Note que é possível prever onde o veículo estará ao final de H_2 (pontos em verde) utilizando os tempos de viagem entre pontos (t_{uv} com $u, v \in N$) e os tempos de serviço (s_u com $u \in N$).

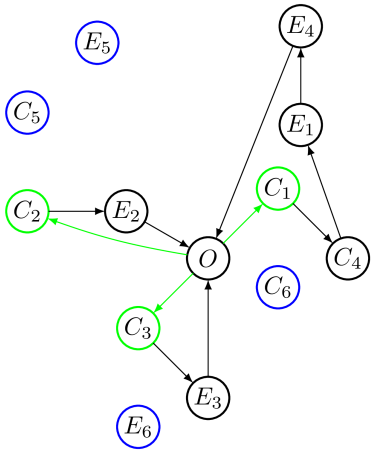
Os pedidos de soluções anteriores cujo o início do atendimento foi (ou será) feito durante H_2 não podem ser modificados. Observando a Figura 3.3 percebe-se que somente o pedido (C_4, E_4) poderia ter sua rota modificada, visto que os outros pedidos do período H_1 têm atendimento iniciado antes do final de H_2.

Figura 3.2 – Solução gerada para H_1. Em verde o ponto de origem. As arestas em vermelho representam o caminho gerado na solução.



Fonte: Do autor (2022).

Figura 3.3 – Pedidos recebidos até o tempo $2\tau - T$. Em azul tem-se os pedidos que chegaram após o tempo $\tau - T$. Pontos verdes são os pontos onde estarão os veículos ao fim de H_2. Arestas já percorridas em verde.

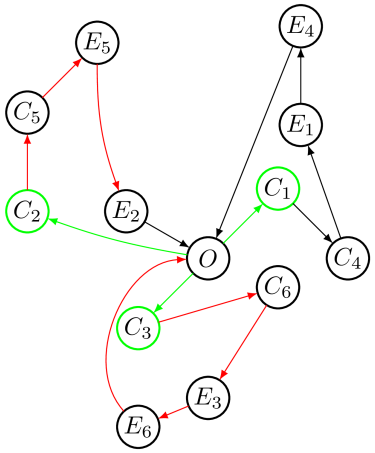


Fonte: Do autor (2022).

Após a execução do algoritmo tem-se o resultado apresentado pela Figura 3.4. Esta nova solução acrescentou os pedidos recebidos às rotas dos veículos. Durante o período seguinte cada veículo seguirá as novas rotas partindo dos pontos C_1, C_2 e C_3.

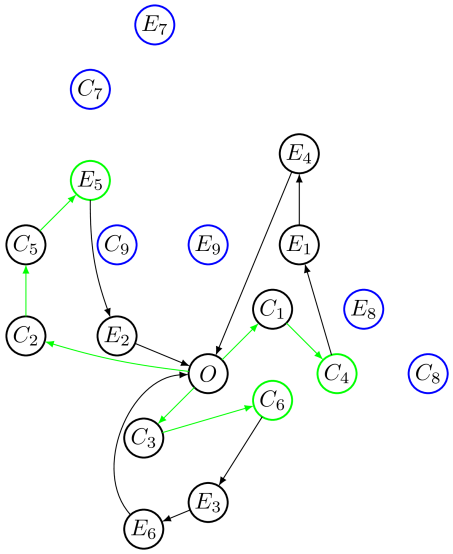
Durante o último período para recebimento de pedidos (H_3), recebe-se os pedidos (C_7, E_7), (C_8, E_8) e (C_9, E_9) até o tempo $3\tau - T$. Na Figura 3.5, é possível visualizar os novos pedidos (em azul) e as posições previstas dos veículos (pontos em verde). Note que desta vez nenhum pedido da solução anterior pode ter sua rota alterada, pois todos os pontos de coleta terão iniciado o atendimento em seu ponto de coleta até o final de H_3.

Figura 3.4 – Nova solução gerada no período de H_2. Arestas em vermelho indicam as mudanças no trajeto feito em H_1. Pontos verdes indicam o ponto de partida dos veículos. Arestas já percorridas em verde.



Fonte: Do autor (2022).

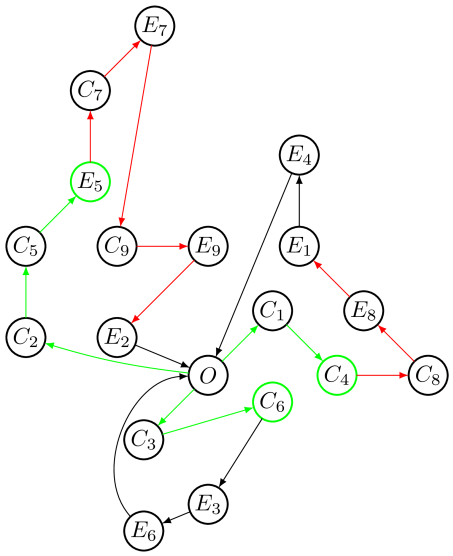
Figura 3.5 – Pedidos recebidos até o tempo $3\tau - T$. Em azul tem-se os pedidos que chegaram após o tempo $2\tau - T$. Pontos verdes são os pontos onde estarão os veículos ao fim de H_3. Arestas já percorridas em verde.



Fonte: Do autor (2022).

Ao fim do tempo de execução tem-se a solução final para o problema, como ilustra a Figura 3.6. Os pontos de partida de cada veículo estão em verde e as modificações nas rotas em vermelho. Ao final da execução os veículos seguem o atendimento aos pedidos das novas rotas.

Figura 3.6 – Solução para o PDCEJT gerada no período de H_3. Arestas em vermelho indicam as mudanças no trajeto feito em H_2. Pontos verdes indicam o ponto de partida dos veículos. Arestas já percorridas em verde.



Fonte: Do autor (2022).

3.4 Considerações

Neste capítulo, foram apresentadas as características do PDCEJT estudado neste trabalho e um exemplo foi utilizado para ilustrar a dinamicidade do problema.

Como definido, o objetivo do problema é a minimização dos custos das rotas realizadas pelos veículos. Uma solução do problema deve respeitar as seguintes restrições:

a) o ponto de entrega não pode ser visitado antes do de coleta;

b) todo veículo deve partir do depósito e retornar a ele dentro do intervalo de tempo $[0, \overline{h}]$;

c) um veículo deve atender um ponto em um horário dentro de sua janela de tempo;

d) cada ponto deve ser visitado exatamente uma vez;

e) se a coleta de um pedido for iniciada, sua entrega deve ser realizada pelo mesmo veículo.

A chegada de pedidos ocorre ao longo do horizonte de planejamento e há a necessidade de rever as rotas estabelecidas para incluir os novos pedidos a cada período de tempo. Problemas de roteamento apresentam um alto grau de dificuldade de resolução, com as restrições adicionais torna-se importante que se tenha um método de resolução eficiente. Portanto, no pró-

ximo capítulo, é proposto um método de resolução baseado em reotimizações periódicas para o PDCEJT estudado.

4 MÉTODO DE RESOLUÇÃO

Neste trabalho, propõe-se desenvolver um método de resolução para o Problema de Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT). Como destacado na literatura, a estratégia de reotimizações periódicas é uma das estratégias mais utilizadas para sua resolução. Ela consiste na divisão do horizonte de planejamento em períodos, como descrito no Capítulo 3. Para cada período de tempo H_j é resolvido um subproblema estático que considera apenas parte dos pedidos. A única restrição adicional é que os pedidos cujo atendimento da coleta foi iniciado não podem ter seu veículo de entrega modificado.

Algoritmo 1: Algoritmo geral para as reotimizações periódicas

```
1  início
2      b =| H |;
3      para j = 1,2,3,...,b faça
4          R_j = recebe_pedidos(H_j, T);
5          se R_j > 1 então
6              R_j = R_j ∪ R_{j-1}
7          fim
8          I_j = gera_instancia_parcial(R_j, H_j, T);
9          S_j = SB(I_j, T);
10     fim
11     S = S_b;
12     retorna S;
13 fim
```

O Algoritmo 1 ilustra como a reotimização é realizada. Como entrada tem-se o conjunto de períodos de tempos $H = \{H_1, H_2, ..., H_b\}$ e o tempo de execução T. Percorre-se os períodos de tempo do conjunto H (Linhas 3-10) na repetição principal. A cada iteração, os pedidos são recebidos e armazenados no conjunto R_j durante um período de tempo igual à $\tau - T$ (Linha 4), sendo τ o tempo total de cada período.

Em seguida, pedidos recebidos anteriormente são adicionados ao conjunto R_j (Linhas 5-7) e uma instância parcial do problema é gerada (Linha 8). Ao final da iteração (Linha 9), uma adaptação da *matheuristic* proposta por Sartori e Buriol (2020) é executada pelo tempo

limite T utilizando a entrada parcial I_j. O algoritmo finaliza retornando a solução S, que é a solução gerada na última iteração da repetição (Linhas 11-12).

Neste capítulo, detalha-se a meta-heurística proposta por Sartori e Buriol (2020) para a versão estática do PDCEJT (Seção 4.1) e as adaptações realizadas para a resolução do PDCEJT (Seção 4.2). Ao final do capítulo, na Seção 4.3, sumariza-se os principais pontos do capítulo.

4.1 Solução do Problema Estático

A *matheuristic* proposta por Sartori e Buriol (2020) combina heurísticas da literatura para a resolução do PCEJT. Os autores buscam minimizar dois objetivos de forma hierárquica: (i) a quantidade de veículos necessários para o roteamento; (ii) o custo de viagem. Este trabalho foca na minimização do custo de viagem.

A ideia principal é combinar heurísticas clássicas da literatura com modelos matemáticos auxiliares. Os métodos heurísticos utilizados foram a Busca em Grandes Vizinhanças (*Large Neighborhood Search* - LNS), baseada na proposta de Ropke e Pisinger (2006), e Busca Adaptativa por Ejeção Guiada (*Adaptive Guided Ejection Search* - AGES), de Curtois et al. (2018). No contexto de modelagem matemática, o PCEJT foi convertido para o problema de Particionamento de Conjuntos (PC), e um Programa Linear Inteiro Misto (PLIM) foi proposto para resolvê-lo, como apresentado por Dumas, Desrosiers e Soumis (1991).

O Algoritmo 2 resume a *matheuristic* de Sartori e Buriol (2020). A resolução do problema começa com a geração de uma solução inicial S e um conjunto $\mathbb{P}$ de rotas (Linhas 2-3) que é utilizado no PLIM na linha 10. Em seguida, nas Linhas 4 e 5 são inicializados os contadores it e itm, que representam, respectivamente, o número de iterações realizadas e o número de iterações sem melhoria.

Algoritmo 2: *Matheuristic* de Sartori e Buriol

```
1  início
2      S = constroi_solucao_inicial(I);
3      ℙ = inicializa_conjunto_de_rotas(S);
4      it = 0;
5      itm = 0;
6      repita
7          S = AGES(S);
8          S = LNS(S);
9          S^PC = PC(S, ℙ);
10         S = aceita_solucao(S, S^PC, it, itm);
11         S = perturbe(S, S*);
12     até criterio_de_parada_cumprido();
13     S* = melhor_solucao_obtida();
14     retorna S*
15 fim
```

Nas Linhas 6-12, tem-se a repetição principal do algoritmo. Primeiramente, a solução passa por melhorias feitas pelo AGES e pela LNS (Linhas 7 e 8). A AGES visa reduzir o número de veículos e a LNS o custo de viagem. Na Linha 9, é resolvido o Problema de Particionamento de Conjuntos com as rotas do conjunto $\mathbb{P}$ e, na Linha 10, é verificado se a solução obtida será aceita. Ao final da iteração, a solução sofre uma perturbação para gerar diversidade de soluções (Linha 11).

O critério de parada da repetição (Linha 12) pode ser o número de iterações, o número de iterações sem melhoria, o tempo computacional, entre outros. Ao cumprir tal critério, a melhor solução obtida pelo método é retornada (Linhas 13-14).

Na sequência desta seção, cada componente da *matheuristic* é descrita em mais detalhes. Primeiramente, discute-se a heurística construtiva utilizada para obter uma solução inicial (Seção 4.1.1). Em seguida, as heurísticas AGES e LNS são exploradas, nas Seções 4.1.2 e 4.1.3, respectivamente. Na Seção 4.1.4, o PLIM para o Problema de Particionamento de Conjuntos é apresentado. Por fim, o critério de aceitação (Seção 4.1.5) e a estratégia de perturbação de uma solução (Seção 4.1.6) são definidos.

4.1.1 Heurística Construtiva

A heurística construtiva utilizada no trabalho de Sartori e Buriol (2020) é uma adaptação da heurística proposta por Solomon (1987), descrita no Algoritmo 3.

Algoritmo 3: Adaptação do Algoritmo Construtivo

```
1  início
2      S = ∅;
3      R̄ = ∅;
4      repita
5          f = cria_rota();
6          para cada r ∈ R tal que r ∉ R̄ faça
7              tenta_inserir(r, f);
8          fim
9          R̄ = R̄ ∪ f;
10         S = S ∪ {f};
11     até R ⊆ R̄;
12     retorna S;
13 fim
```

Enquanto houver pedidos que não foram inseridos em uma rota (Linhas 4-11), cria-se uma rota vazia f (Linha 5) e tenta-se inserir cada pedido na melhor posição factível da rota f (Linhas 6-8). Após as tentativas de inserção, armazena-se a rota e, caso todos os pedidos já tenham sido inseridos, retorna-se a solução (Linhas 10-12). Caso contrário, o processo se repete a partir da criação de uma nova rota na Linha 5.

4.1.2 Busca Adaptativa por Ejeção Guiada

Na *matheuristic*, a AGES (Curtois et al. (2018)) é a principal responsável pela minimização do número de rotas. Sua ideia básica é: dada uma solução inicial, remove-se uma rota e reinsere seus pontos nas outras rotas da solução.

Como pode ser visto no Algoritmo 4, a AGES inicia sua repetição principal na Linha 3. Em seguida, na Linha 4, a rota de índice f é removida da solução inicial S, gerando uma nova solução incompleta S'. A rota S_f é utilizada para criar um pilha de pedidos removidos $\mathbb{E}$ e, para

cada $r \in R$, cria-se uma penalidade $\rho[r]$ com valor inicial 1. A penalidade indica o quão difícil é inserir um pedido na solução (Linhas 5 e 6).

Nas Linhas 7-18 inicia-se a repetição de reinserção dos pedidos. Um pedido é removido da pilha e, caso haja uma ou mais posições factíveis para inserção, o pedido é reinserido (Linhas 8-11) ao se escolher uma posição e rota aleatória.

Caso não haja posições factíveis (Linhas 12-17), a penalidade do pedido é aumentada, remove-se k elementos da solução e os insere na pilha. A remoção é feita de forma a minimizar as penalidades dos pedidos removidos. Quando $k = 1$, tenta-se remover um pedido $r \in R$ que minimize $\rho[r]$. Por outro lado, quando $k = 2$, tenta-se remover dois pedidos $q, r \in R$ da mesma rota de forma que se minimize $\rho[q] + \rho[r]$. Sartori e Buriol (2020) limitam o valor de k para dois pedidos e, caso não seja possível reinserir o pedido, ele é recolocado na pilha $\mathbb{E}$. A ideia é utilizar as penalidades para remover pedidos que são mais facilmente reinseridos no futuro. Após a reinserção a solução, é perturbada pelo método apresentado na Seção 4.1.6.

Algoritmo 4: Adaptação da componente AGES

```
1  início
2      it_ages = 0;
3      enquanto pode_perturbar(it_ages) faça
4          f, S' = remove_rota_aleatoria(S);
5          𝔼 = cria_pilha(S_f);
6          ρ[r] = 1, r ∈ R;
7          enquanto 𝔼 ≠ ∅ e pode_perturbar(it_ages) faça
8              r̄ = remove(𝔼);
9              se tem_insercao_factivel(r̄, S') então
10                 insere(r̄, S');
11             fim
12             senão
13                 ρ[r̄] = ρ[r̄] + 1;
14                 ejeta_e_insere(r̄, S', 𝔼);
15                 perturbe(S');
16                 it_ages = it_ages + 1;
17             fim
18         fim
19         se 𝔼 ≠ ∅ então
20             S = S';
21             it_ages = 0;
22         fim
23     fim
24 fim
```

O critério de parada se baseia no número de perturbações realizadas, permitindo que o algoritmo mantenha sua execução enquanto melhorar ou que finalize rapidamente caso não esteja conseguindo diminuir o número de rotas.

A variável *it_ages* mantém armazenado o número de perturbações executadas. Após uma perturbação ser executada, seu valor é incrementado (Linha 16). Nas Linhas 19-22, é verificado se houve melhoria na solução após a reinserção. Se a pilha $\mathbb{E}$ estiver vazia, significa

que todos os pedidos da rota removida foram reinseridos, ou seja, a solução foi melhorada. Nesse caso, a solução é atualizada e o contador *it_ages* é reinicializado. Caso a pilha não esteja vazia, o algoritmo é finalizado, pois significa que o número de perturbações realizadas alcançou o limite, terminando a sua execução na próxima verificação da Linha 3.

4.1.3 Busca em Grandes Vizinhanças

Sartori e Buriol (2020) propõem uma Busca em Grandes Vizinhanças (LNS) com base na Busca Adaptativa em Grandes Vizinhanças (*Adaptative Large Neighborhood Search* - ALNS) de Ropke e Pisinger (2006) e na LNS de Curtois et al. (2018). O conceito principal desta busca é a destruição e reconstrução de soluções. O Algoritmo 5 apresenta a adaptação para o PCEJT de Sartori e Buriol (2020). O método se resume na consecutiva remoção e reinserção de pedidos. As Linhas 3-13 apresentam a repetição principal da heurística.

Inicialmente, ocorre a destruição da solução atual (Linha 4) através de uma de três heurísticas: Remoção de Shaw, Remoção do Pior ou Remoção Aleatória, que removem um determinado número de pedidos da solução. A primeira remove pedidos com base no seu relacionamento espacial (onde os pontos estão localizados), temporal (quando os pedidos devem ser atendidos) e na relação de suas demandas (quão semelhante é sua demanda). A Remoção do Pior busca remover o pedido que mais incrementa no custo da solução. Por fim, a Remoção Aleatória seleciona os pedidos a serem removidos de forma aleatória. A cada iteração da repetição uma das heurísticas é escolhida, cada uma tendo uma probabilidade $\frac{w_z}{w_s+w_p+w_a}$, com $z \in \{s, p, a\}$, sendo w_s, w_p e w_a valores constantes relacionados a Remoção de Shaw, Remoção do Pior e a Remoção Aleatória, respectivamente. Para mais detalhes sobre os métodos de remoção, recomenda-se a leitura de Ropke e Pisinger (2006).

Algoritmo 5: Adaptação da componente LNS

```
1  início
2      it = 0;
3      repita
4          R', S' = remove_pedidos(S);
5          S' = adiciona_pedidos(R', S');
6          S = aceita_solucao(S, S');
7          se melhorou_solucao(S) então
8              it = 0;
9          senão
10             it = it + 1;
11         fim
12     até it_sem_melhora_alcancadas(it);
13     S+ = melhor_solucao_obtida();
14     retorna S+;
15 fim
```

Após a remoção, a solução deve ser reconstruída, ou seja, os pedidos devem ser reinseridos. Na Linha 5, os pedidos são adicionados utilizando a Heurística de Arrependimento. Inicialmente, calcula-se o custo mínimo de inserção do pedido em cada uma das rotas, ou seja, a posição de inserção do pedido que menos aumenta o custo total da rota. Quando a inserção não for possível, atribui-se um valor grande o suficiente. O pedido a ser inserido será aquele que maximizar a soma das diferenças do custo de sua inserção mais barata para as outras $\mathscr{K}$ inserções mais baratas calculadas. Logo, pedidos com um número menor de possíveis inserções terão prioridade e poderão ser inseridos com mais facilidade. Portanto, evita-se o arrependimento de não se realizar a inserção quando foi possível. Note que quando $\mathscr{K} = 1$, tem-se a heurística inserção gulosa utilizada na construção.

Após a reconstrução da solução, é verificado se a solução obtida será mantida. A heurística para a aceitação de solução utilizada por Sartori e Buriol (2020) foi a *Late Acceptance Hill-Climbing* (LAHC). Esse método pode ser entendido como uma extensão do clássico *Hill-Climbing* que aceita soluções apenas quando são melhores que a última. A grande dife-

rença é que a LAHC utiliza-se do histórico de soluções obtidas. Para detalhes sobre a LAHC recomenda-se a leitura de Burke e Bykov (2017).

O algoritmo termina após um determinado número de iterações sem melhoria. A variável *it* definida na Linha 2 mantém a contagem do número iterações sem soluções melhoradas. Sempre que for encontrada uma nova solução, ela é zerada (Linhas 7-11). Isso permite que a LNS continue sendo aplicada enquanto for eficiente. Após finalizar a repetição, o algoritmo retorna a melhor solução encontrada durante sua execução (Linhas 13-14).

4.1.4 Particionamento de Conjuntos

Dumas, Desrosiers e Soumis (1991) utilizam uma formulação para o Problema de Particionamento de Conjuntos como um método de geração de colunas para resolver o PCEJT. Considere $\mathbb{P}$ como um conjunto que contém as rotas de todas as possíveis soluções factíveis para uma instância do problema. Pode-se afirmar que a solução ótima é um subconjunto de $\mathbb{P}$.

Considere Δ_{rf} como um valor binário cujo valor 1 indica que a rota (ou veículo) $f = 1,2,...,\mid \mathbb{P} \mid$ atende o pedido $r \in R$, e 0, caso contrário. Tem-se a variável binária y_f cujo valor 1 indica que a rota de índice f está presente na solução do problema, e 0, caso não seja escolhida. Formula-se o seguinte PLIM para resolver PCEJT.

$$\min \sum_{f=1}^{\mathbb{P}} y_f C(\mathbb{P}_f) \tag{4.1}$$

sujeito à

$$\sum_{f=1}^{\mathbb{P}} y_f \Delta_{rf} = 1, \forall r \in R \tag{4.2}$$

$$y_f \in \{0,1\}, f = 1,2,...,\mid \mathbb{P} \mid \tag{4.3}$$

A função objetivo (4.1) minimiza a soma dos custos das rotas da solução. As Restrições (4.2) indicam que todo pedido deve ser atendido por exatamente uma rota. Por fim, o domínio das variáveis é dado pelas Restrições (4.3)

O uso do conjunto $\mathbb{P}$ pode ser inviável, pois o número de rotas possíveis para uma instância do PCEJT é exponencial. Contudo, é possível utilizar métodos heurísticos para gerar um conjunto $\mathscr{P} \subseteq \mathbb{P}$ de rotas factíveis para serem utilizadas pelo método, como feito por Sartori e Buriol (2020). A cada iteração do Algoritmo 2, rotas obtidas pelas heurísticas anteriores são

adicionadas ao conjunto e utilizadas pelo PLIM para se obter uma nova solução que reduz os custos de viagens.

4.1.5 Critério de Aceitação

Após uma nova solução ser encontrada pela formulação apresentada na Seção 4.1.4, verifica-se se ela será aceita (Linha 10 do Algoritmo 2). Caso a solução gerada seja melhor que a solução incumbente, ela é aceita. Caso contrário, ela é aceita com probabilidade $\frac{itm}{it}$, sendo itm o número de iterações sem melhoria e it o número de iterações decorridas do Algoritmo 2. Caso ela não seja aceita, a solução obtida na LNS é utilizada para dar continuidade ao método.

4.1.6 Estratégia de Perturbação

Para gerar diversidade, Sartori e Buriol (2020) testaram dois algoritmos de perturbação. O primeiro, proposto por Nagata e Kobayashi (2010), conta com duas operações aleatórias para gerar novas soluções. A Mudança Aleatória seleciona um pedido qualquer r de uma rota f e o insere em uma posição aleatória factível de outra rota. A Troca Aleatória sorteia dois pedidos e os troca de rotas, reinserindo na melhor posição possível de suas novas rotas. A escolha da operação se dá com probabilidade δ_{MA} e δ_{TA}, sendo δ_{MA} e δ_{TA} as probabilidades de escolher a Mudança Aleatória e Troca Aleatória, respectivamente.

Como segundo algoritmo de perturbação, os autores propõem a Mudança Enviesada, que sugere uma alteração na Mudança Aleatória. Mantém-se a ideia de remover um pedido aleatório e reinserí-lo em uma outra rota também aleatória. Contudo, entre as todas as posições de inserções, sorteia-se $\lambda\%$ e escolhe-se a melhor posição entre elas. Nota-se que utilizando $\lambda = 100$, teria-se o método original.

4.2 Adaptação para o Problema Dinâmico

Para resolver o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PD-CEJT) foi proposta uma adaptação da *matheuristic* descrita na seção anterior. Estas adaptações são necessárias devido às mudanças na função objetivo, à adição da dinamicidade dos pedidos e aos veículos terem capacidade ilimitada.

Na Seção 4.2.1, explica-se como o problema e o algoritmo foram adaptados para atender a restrição de frota ilimitada. As modificações da heurística construtiva são descritas na Seção 4.2.2, enquanto, na Seção 4.2.3, é proposta uma alteração do método de inserção utilizado no algoritmo. A descrição das restrições relacionadas as modificações de rotas geradas em períodos anteriores é apresentada na Seção 4.2.4. Na sequência, a Seção 4.2.5 analisa o uso da componente AGES e as modificações na LNS propostas na Seção 4.2.6. São discutidos, na Seção 4.2.7, os motivos de não serem necessárias as adaptações do PLIM proposto por Sartori e Buriol (2020) para o Particionamento de Conjuntos. Por fim, a Seção 4.2.8 finaliza apontando as adaptações do algoritmo de perturbação.

4.2.1 Frota Ilimitada

A primeira adaptação no algoritmo é feita para abordar uma frota ilimitada. No contexto do problema estudado, considera-se que há um número muito maior de veículos do que o necessário para atender os pedidos. Por isso, diz-se que a frota é ilimitada. Assim, considerando m^* como o número mínimo de veículos necessário para atender os pedidos, soluciona-se o problema com a restrição de frota limitada de tamanho m, com $m \gg m^*$. O número de rotas geradas ao final da solução é limitada a m.

4.2.2 Heurística Construtiva Adaptada

A cada passo do PDCEJT, são conhecidos os pedidos que chegaram até o instante $j\tau - T$, sendo que parte deles podem estar presentes em soluções parciais de períodos anteriores. A fim de respeitar as novas restrições tanto de dinamicidade quanto de frota ilimitada, propõe-se o Algoritmo 6, que é uma adaptação da heurística construtiva descrita na Seção 4.1.1.

A primeira mudança em relação ao Algoritmo 3 se dá na Linha 2, onde define-se S_j como o conjunto de rotas do período anterior ao invés um conjunto vazio. Como se têm um número de veículos limitados, rotas vazias são criadas antes da repetição principal na Linha 3, ao invés de serem alocadas de acordo com a necessidade, como foi feito no trabalho de Sartori e Buriol (2020).

Algoritmo 6: Algoritmo Construtivo para o PDCEJT

```
1 início
2     S_j = S_{j-1};
3     S_j = adiciona_rotas_vazias(S_j);
4     para cada r ∈ R* faça
5         insere(r, S_j);
6     fim
7     retorna S_j;
8 fim
```

Nas Linhas 4-6, apresenta-se a repetição para inserir pedidos na solução S_j. A repetição itera sobre o conjunto R^*, que é o conjunto de pedidos recebidos no período de tempo $[(j-1)\tau, j\tau - T]$. Nota-se que pedidos que chegaram antes do instante $(j-1)\tau$ estarão presentes em S_{j-1} e, por isso, não é necessário inseri-los.

Como $m \gg m^*$, espera-se que a inserção realizada na Linha 5 seja sempre bem sucedida, pois sempre será possível inserir em uma rota vazia. O algoritmo finaliza retornando a solução S_j, para que as buscas locais sejam executadas.

4.2.3 Estratégia de Inserção de Pedidos

Como será reportado nos testes computacionais (Capítulo 6), a implementação flexível (Capítulo 5) acarretou em perda de eficiência computacional das heurísticas adaptadas. Esse agravante teve grande impacto na etapa de construção, devido ao fato de a inserção de Sartori e Buriol (2020) ser gulosa, ou seja, verificar todas as posições factíveis para a inserção e escolher a que tem o menor custo. Portanto, neste trabalho, também foi avaliada a estratégia de inserção na primeira posição factível encontrada. Note que o operador de inserção da construção foi implementado como a Heurística de Arrependimento utilizando $\mathscr{K} = 1$. No Capítulo 6, as duas estratégias de inserção são comparadas.

4.2.4 Restrição de Pedidos Fixos

Diz-se que que pedidos cujo ponto de coleta teve atendimento iniciado até o final de um período H_j são pedidos fixos e eles devem ser mantidos na rota do mesmo veículo. Portanto,

durante a alteração de uma rota $S_{jf} \in S_j$, impede-se a remoção de pedidos cujo ponto de coleta $i \in P$ está em uma rota parcial $\overline{S}_{jf}$. Além disso, bloqueia-se a inserção de pontos em uma posição $k \leq | \overline{S}_{jf} |$ em uma rota S_{jf}, pois os pontos em $\overline{S}_{jf}$ já foram percorridos.

4.2.5 Utilização da Componente AGES

A componente AGES tem o objetivo específico de reduzir o número de rotas, porém a função objetivo do PDCEJT busca minimizar o custo de viagem dos veículos. Contudo, não é garantido que uma solução com grande número de rotas tenha um custo de viagem menor. Logo, a componente AGES pode ser vista como uma forma de perturbação para forçar a geração de rotas mais longas. A fim de testar esse comportamento, no Capítulo 6 apresenta-se um estudo comparativo do uso ou não uso da AGES.

4.2.6 Adaptações na LNS

A LNS utiliza a Heurística de Arrependimento para a reinserção de pedidos. Da mesma forma que a inserção gulosa (utilizada na construção) foi impactada pela flexibilização da implementação, a Heurística de Arrependimento teve um aumento no custo computacional. A fim de amenizar o impacto da flexibilização, propõe-se a Heurística de Arrependimento W, uma variação da Heurística de Arrependimento que, para cada pedido, calcula o custo mínimo de inserção apenas nas W melhores rotas da solução ao invés de todas as rotas. Nos experimentos realizados, utiliza-se $W = \mathscr{K}$, sendo $\mathscr{K}$ o número de inserções mais baratas a serem utilizados no cálculo para a comparação da Heurística de Arrependimento.

4.2.7 Particionamento de Conjuntos

O PLIM utilzado para o Particionamento de Conjuntos não sofre alterações. Visto que o número de rotas é muito maior do que o necessário, a restrição de número de rotas não precisa ser adicionada. As restrições de dinamicidade são mantidas durante as operações de inserção e remoção da rota. Logo, as rotas presentes no conjunto $\mathbb{P}$ continuam factíveis.

4.2.8 Adaptações na Perturbação

Utiliza-se os operadores apresentados na Seção 4.1.6 com probabilidades δ_{MA}, δ_{TA} e δ_{ME}, que são, respectivamente, as probabilidades de escolher a Mudança Aleatória, Troca Aleatória e Mudança Enviesada. A ideia é permitir que a Mudança Enviesada realize movimentos de perturbação menores também sejam feitos para evitar um grande distanciamento da solução gerada. O distanciamento pode ser ruim, pois pode ocorrer demora na geração de rotas melhores que são necessárias para o Particionamento de Conjuntos. Optou-se por esse método devido ao impacto da flexibilização na eficiência do algoritmo e à necessidade de ter resoluções mais rápidas.

4.3 Considerações

A proposta de solução para o PDCEJT deste trabalho é baseada em reotimizações periódicas e utiliza uma adaptação do algoritmo proposto por Sartori e Buriol (2020). A *matheuristic* foi proposta para o contexto estático do problema e foi detalhada na Seção 4.1. Devido às diferenças nas restrições e na função objetivo do PDCEJT, foram propostas adaptações do método na Seção 4.2.

Quanto à restrição de frota ilimitada, aborda-se como uma flexibilização da restrição de frota limitada, onde o tamanho da frota é muito maior que a necessária para o atendimento dos pedidos. A dinamicidade, por sua vez, bloqueia movimentos de remoção de pedidos cuja coleta foi inciada e de inserção em posições já percorridas.

Observou-se que a componente AGES pode ou não ser tão eficaz para o contexto deste trabalho e propôs-se um estudo de seu impacto no PDCEJT que é descrito no Capítulo 6. Também foram propostas modificações nos métodos de inserção para o algoritmo de construção e para a LNS, devido ao impacto gerado pela flexibilização da implementação.

As técnicas utilizadas para tal implementação são descritas no Capítulo 5. Como exemplo, a ferramenta implementada é utilizada para a resolução do Problema Dinâmico de Coleta e Entrega com Janelas de Tempo com pontos Urbanos e Rurais (PDCEJT/UR).

5 GENERALIZAÇÃO DA IMPLEMENTAÇÃO

As versões dinâmicas do Problema Roteamento de Veículos (PRV) são reportadas frequentemente na literatura (PILLAC et al., 2013; PSARAFTIS; WEN; KONTOVAS, 2016; Ojeda Rios et al., 2021; BERBEGLIA; CORDEAU; LAPORTE, 2010). Em casos reais, as variantes podem ser tão específicas que não se encontra um estudo que as introduzam. Outra possível situação é o problema ter um grande número de restrições, tornando tal variante distante daquelas abordadas na literatura.

Há trabalhos que buscam criar ferramentas para a resolução de diferentes versões do PRV. Vidal et al. (2014) apresenta um *framework* para a resolução de 29 variantes do PRV. O método implementado foi uma *Unified Hybrid Genetic Search* (UHGS), baseado no trabalho proposto por Vidal et al. (2013). O algoritmo contém operadores que são independentes entre si, permitindo sua componentização e generalização. Nos experimentos realizados, o método obteve bons resultados comparado aos estados-da-arte da época de sua publicação, encontrando soluções que se tornaram referência para todas as variações.

Abordando um contexto mais genérico, Silva et al. (2019) desenvolveram um *framework* que utiliza de aprendizado baseado em multi-agentes para problemas de otimização combinatória. A ideia é que agentes implementem heurísticas e meta-heurísticas diferentes. A ação individual de cada agente e sua comunicação com outros agentes permite a hibridização dos métodos implementados pelos agentes, tendendo a utilizar as melhores combinações de heurísticas e meta-heurísticas. O modelo foi testado para dois problemas clássicos, sendo um deles o Problema de Roteamento de Veículos com Janelas de Tempo. O algoritmo se mostrou versátil na resolução do problema, apesar de encontrar limitantes superiores distantes das melhores soluções conhecidas.

Perron (2011), por sua vez, apresenta o desenvolvimento da ferramenta *OR-tools* que atualmente conta com ferramentas para a resolução do PRV e algumas de suas variantes (Google Developers, 2021). A ferramenta implementa algoritmos estado da arte e pode ser utilizada em diferentes linguagens de programação.

O trabalho de Oliveira et al. (2021) apresentam uma API para facilitar a implementação de métodos de solução de problemas de otimização. O método proposto pelos autores apresenta uma interface simples para a utilização de um *Biased Random-Key Genetic Algorithm* (BRKGA). Esse método é baseado no Algoritmo Genético, porém apresenta uma vantagem: a

factibilidade das soluções geradas pelo cruzamento é mais facilmente mantida, pois as soluções são codificadas em um vetor de chaves aleatórias. Portanto, é possível resolver diferentes problemas de otimização modificando apenas o decodificador, que transforma o vetor de chaves aleatórias em soluções válidas. Utilizando de conceitos de Orientação a Objetos, é feita uma implementação em C++. Testes realizados com algoritmos de decodificação com diferentes complexidades de tempo apontam a eficiência do método.

Até onde se sabe, os trabalhos de Oliveira et al. (2021) e Vidal et al. (2014) são os que mais se aproximam das intenções deste trabalho. Deseja-se a generalização de soluções para o PDCEJT, não apenas com o intuito de resolver diferentes variantes com um mesmo método, mas também de facilitar a modificação de algoritmos, heurísticas e meta-heurísticas utilizadas para a solução do problema. Ao mesmo tempo, tenta-se facilitar a utilização do mesmo método para quando há pequenas variações no problema a ser resolvido, como por exemplo, a adição da restrição de capacidade ao PDCEJT , ou a modificação de sua função objetivo.

Para cumprir este objetivo, utiliza-se flexibilidade da linguagem *Python3* (ROSSUM; DRAKE, 2009) e de propriedades de Orientação a Objetos (OO). Na Seção 5.1, apresenta-se as estratégias relacionadas a Orientação a Objetos para a flexibilização da implementação, enquanto a Seção 5.2 aprofunda em detalhes das estratégias de implementação em Python. Para exemplificar a aplicação das estratégias, na Seção 5.3, propõe-se métodos para a resolução de uma outra variante do PDCEJT: o *Problema Dinâmico de Coleta e Entrega com Janelas de Tempo e pontos Urbanos e Rurais* (PDCEJT/UR). Por fim, na Seção 5.4, apresenta-se as considerações finais deste Capítulo.

5.1 Orientação a Objetos

Para uma implementação flexível baseada em Orientação à Objetos, os conceitos de herança e polimorfismo são essenciais. Tais estratégias de codificação são interessantes, pois o primeiro conceito torna possível a generalização de classes com características semelhantes, enquanto o segundo introduz a sobrescrita, que permite que uma classe filha reimplemente um método definido em sua classe pai.

Nas Seções a seguir explica-se as abstrações realizadas para a criação da estrutura de classes utilizadas. Frisa-se que os diagramas representam simplificações das classes, omitindo

ou alterando parte das informações, como parâmetros, atributos, métodos auxiliares, entre outros.

A Seção 5.1.1 apresenta a generalização realizada para a leitura de instâncias. Na sequência, na Seção 5.1.2 é discutida a representação das restrições, enquanto a Seção 5.1.3 explica as abstrações das funções objetivos. Prossegue-se com os detalhes relacionados aos vértices (que representam os pontos) e as rotas, na Seção 5.1.4 e na Seção 5.1.5, respectivamente. Na Seção 5.1.6, aborda-se os operadores de modificação de rotas. A Seção 5.1.7 apresenta a classe que representa as soluções. Na Seção 5.1.8, detalha-se a abstração dos algoritmos, heurísticas e meta-heurísticas. Por fim, a Seção 5.1.9 aborda as classes que representam os métodos de solução para cada problema a ser resolvido.

5.1.1 Leitor de Dados

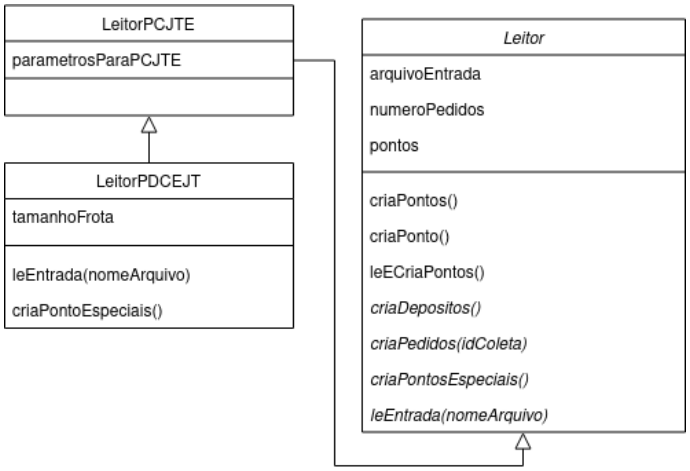
Para cada problema, há a necessidade de parâmetros diferentes e, por isso, a leitura de dados foi generalizada utilizando a estrutura da Figura 5.1. Foi criada uma classe abstrata *Leitor*, responsável pela leitura de um arquivo que irá conter os parâmetros do problema a ser resolvido. Para permitir a adaptação às especificidades de cada problema, quatro métodos abstratos estão presentes na classe leitor, sendo eles listados a seguir:

a) *leEntrada*: responsável pela leitura do arquivo com os dados da instância a ser resolvida;
b) *criaPedidos*: cria os vértices relacionados ao pedido;
c) *criaDepósito*: cria o(s) vértice(s) que indica o depósito;
d) *criaPontosEspeciais*: permite a criação de vértices extras ou a adição de atributos específicos do problema aos vértices já criado.

O método *leEntrada* é chamado durante o método *leECriaPontos* e faz processamento de informações tais como número de pedidos, custos de viagem, tempos de viagem, entre outros. Na sequência de sua finalização, é chamado o método *criaPontos*, que por sua vez chama, respectivamente, os métodos *criaDeposito*, *criaPedidos* e *criaPontosEspeciais*.

A ideia da generalização da criação do Depósito é permitir que o problema seja adaptado para casos onde não haja depósito ou que haja mais de um depósito. O método *criaPedidos* é responsável pela criação dos pontos a serem transportados. Nele deve ser feita a chamada do método *criaPonto*, implementado na classe *Leitor*. Esse método possibilita a inicialização de

Figura 5.1 – Diagrama de Classes simplificado para representar leitores de instâncias.



Fonte: Do autor (2022).

uma instância da classe *Vertice* (Seção 5.1.4) utilizada para o problema. Isso permite que pedidos de diferentes problemas criem seus respectivos vértices de diferentes formas, facilitando, por exemplo, a criação de uma classe para a leitura de um problema sem coleta ou sem entrega.

Após a criação dos vértices dos pedidos, é chamado o método *criaPontosEspecificos*. Esse método é importante para a adição das especificidades dos pontos do problema, como por exemplo, a atribuição de um valor que indica se um ponto pertence a um vértice (ou ponto) está em uma rota parcial de uma solução anterior.

Na implementação atual (encontrada em: <https://github.com/thuzax/vrp-solver/tree/dev/solver>), a classe *Leitor* tem apenas uma filha, a classe *LeitorPCJTE*, que lê um arquivo de entrada para o PCEJT estático. Nela são implementadas as classes abstratas de seu pai, além de adicionar métodos de processamento que foram omitidos no diagrama. A classe *LeitorPDCEJT* herda tais métodos de seu pai e reimplementa apenas os métodos *leEntrada* e *criaPontosEspecificos*, para processar os parâmetros adicionais do problema dinâmico e adicionar os vértices que pertencem à uma rota de uma solução anterior. Para manter a generalidade do leitor, considera-se que no arquivo de entrada PDCEJT haverá capacidade limitada e demanda de pontos. Contudo, como será visto na Seção 5.2, a verificação de factibilidade da restrição de capacidade pode não ser incluída na execução do algoritmo.

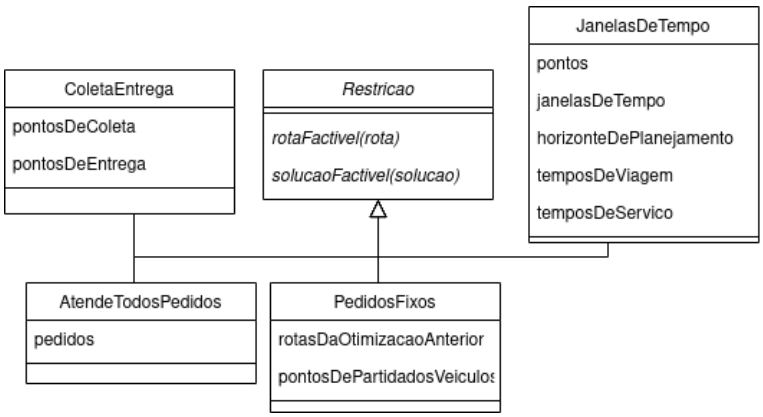
Atualmente, encontra-se implementado apenas a leitura para problemas com coleta e entrega considerando apenas um depósito. Contudo, os métodos abstratos da classe *Leitor*

permitem variação nesses quesitos, além de permitir a leitura de outros dados relacionados a diferentes problemas.

5.1.2 Restrições

Em geral, durante a execução de um algoritmo para resolver roteamento, é necessário verificar se uma rota ou se uma solução cumpre determinada restrição. Logo, uma restrição pode ser vista como uma entidade que pratica duas ações: verificar se uma rota é factível e verificar se uma solução é factível. A Figura 5.2 apresenta a estrutura utilizada para a implementação das restrições, sendo *Restricao* uma classe abstrata que solicita que suas filhas implementem os métodos *rotaFactivel* e *solucaoFactivel*. As classes filhas implementam as seguintes restrições do PDCEJT:

Figura 5.2 – Diagrama de Classes simplificado para representar restrições



Fonte: Do autor (2022).

a) *AtendeTodosPedidos*: todos os pontos de coleta e entrega devem ser atendidos;
b) *ColetaEntrega*: um ponto de coleta deve ser visitado antes do seu ponto de entrega;
c) *JanelasDeTempo*: um ponto deve ser atendido durante uma janela de tempo específica. A restrição de horário de serviço também é satisfeita com esta restrição pois considera-se que a janela de tempo do depósito é $[0, \overline{h}]$;
d) *PedidosFixos*: um ponto de coleta cujo atendimento já foi iniciado por um veículo deve ter seu ponto de entrega atendido pelo mesmo.

Cada classe filha tem como atributos os dados necessários para a implementação de seus métodos de verificação. Com a generalização feita, durante a implementação das heurís-

ticas, é possível utilizar somente os métodos definidos na classe *Restricao* para verificações de factibilidade.

5.1.3 Função Objetivo

A função objetivo do problema também pode ser enxergada como um objeto. Uma função objetivo precisa ser capaz de calcular o custo de uma rota, de uma solução ou de um pedido inserido em uma rota. Além disso, também é preciso ser possível identificar se uma rota (solução) tem melhor valor para a função do que outra rota (solução).

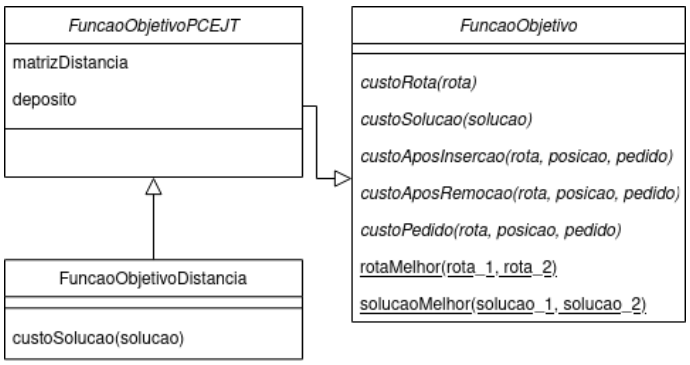
A Figura 5.3 apresenta as classes implementadas para a representação da função objetivo. A classe abstrata *FuncaoObjetivo* implementa dois métodos estáticos: *rotaMelhor* e *solucaoMelhor*, que verificam se seus respectivos primeiros parâmetros (*rota_1* ou *solucao_1*) são melhor que o segundo (*rota_2* ou *solucao_2*). Na implementação dos métodos de comparação, considera-se por padrão que uma solução é melhor se seu custo for menor. O método *solucaoMelhor* implementa uma função hierárquica, onde primeiro compara-se o custo da solução (número de veículos, número de pedidos atendidos, custo de viagem, entre outros) e, em seguida, caso o custo das soluções sejam iguais, compara-se a o custo das rotas, que, nas heurísticas implementadas, é calculado pelo somatório dos custos de cada rota.

Os métodos abstratos da classe são implementados a fim de permitir a alteração nos métodos de cálculos de custo. Define-se, a seguir, os métodos da classe *FuncaoObjetivo*, que devem ser implementados nas classes filhas:

a) *custoRota*: deve calcular o custo total de uma rota;
b) *custoSolucao*: deve calcular o custo de uma solução;
c) *custoPedido*: deve calcular o custo de um pedido dentro de uma rota, ou seja, quanto do custo da rota é incrementado ao se ter o pedido inserido;
d) *custoAposInsercao*: deve calcular quanto é aumentado do custo de uma rota após a inserção determinado pedido;
e) *custoAposRemocao*: deve calcular quanto é diminuído do custo de uma rota após a remoção determinado pedido.

Em grande parte dos Problemas de Coleta e Entrega com Janelas de Tempo, tem-se o custo de viagem como o custo de uma rota. Esse custo pode ser representado em uma matriz de distância e seu cálculo é padrão: o custo de viagem de uma rota é a soma do custo de viagem

Figura 5.3 – Diagrama de Classes Simplificado para representar funções objetivos.



Fonte: Do autor (2022).

entre os pontos da rota. A classe abstrata *FuncaoObjetivoPCEJT* tem como função implementar o cálculo do custo de viagem e o custo de um pedido. Devido a isso, ela implementa os métodos *custoRota* e *custoPedido*. Há, também, a implementação dos métodos *custoAposInsercao* e *custoAposRemocao*, pois eles podem ser obtidos a partir do método *custoPedido*, já que o primeiro é o calculo simulando que a inserção foi feita e o segundo é o valor oposto ao custo do pedido.

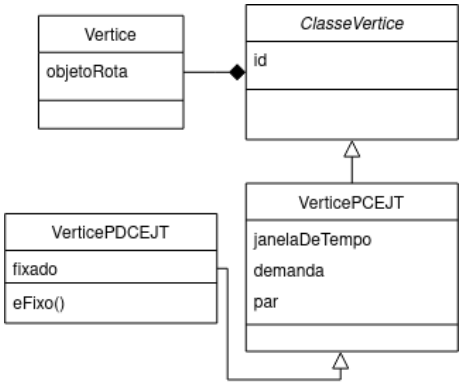
O custo de uma solução, diferentemente do custo de uma rota, costuma sofrer variações. Por exemplo, no trabalho de Sartori e Buriol (2020), o custo de uma solução é dada pelo número de veículos, enquanto neste trabalho o custo da solução é o somatório do custo de viagem das rotas. A classe *FuncaoObjetivoDistancia* representa a função objetivo do PDCEJT, implementando o método *custoSolucao* como o somatório dos custos da rotas.

A função objetivo indica o quão boa é uma solução e guia o método de solução na direção de melhoria. Em problemas de roteamento, pode-se dizer que a função objetivo pode ser calculada em função das rotas ou de outros fatores (número de veículos, número de pedidos, entre outros). Neste trabalho, optou-se em generalizar a função objetivo como uma classe abstrata que calcula separadamente o custo de uma rota e o custo da solução, sendo hierárquico o meio de comparação padrão entre soluções. Apesar disso, a sobrescrita desses métodos torna possível a utilização de outros meios de comparação. Os métodos abstratos permitem a implementação de diferentes funções objetivos tornando seu uso genérico. Utilizando os métodos estáticos e abstratos da classe *FuncaoObjetivo*, é possível calcular e comparar diferentes funções objetivos utilizando a mesma chamada.

5.1.4 Vértices

Vértices são a representação dos pontos de coleta ou entrega. Sua modelagem como objeto permite o acesso a certos atributos que pertencem a um ponto, como sua janela de tempo ou demanda. Neste trabalho, a abstração de um vértice é feita como mostrada na Figura 5.4. A *ClasseVertice* e suas filhas representam um ponto e a classe *Vertice* seria o meio de acesso aos seus objetos.

Figura 5.4 – Diagrama de Classes Simplificado para representar os pontos do problema e seus atributos.



Fonte: Do autor (2022).

A implementação foi feita dessa forma para permitir que a chamada do construtor classe seja padrão, variando somente seus parâmetros. O construtor retorna um objeto descendente de *ClasseVertice*, permitindo a utilização dos atributos e métodos definidos. Na implementação atual, a chamada do método *Vertice()* ocorre apenas no método *criaPonto* da classe abstrata *Leitor*. Se parâmetros adicionais forem necessários, a função equivalente deve ser sobrescrita. A passagem de parâmetros no construtor, contudo, pode ser contornada utilizando de atribuições através de métodos *set*. Na classe *Leitor*, é possível fazer essas atribuições no método *criaVerticesEspeciais*.

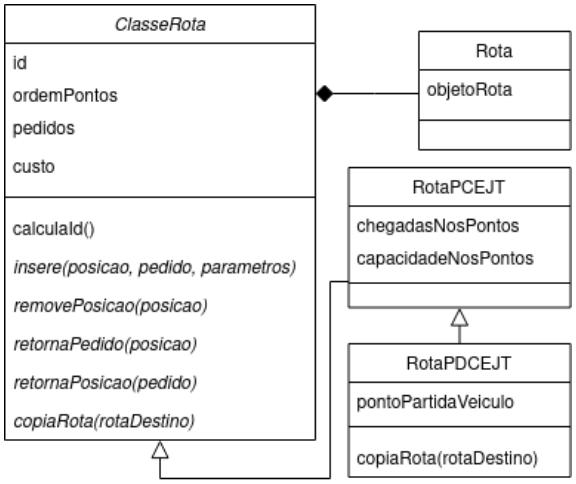
A classe *VerticePCEJT* representa um vértice do problema estático, armazenando dados como a janela de tempo, demanda e o par (de coleta ou entrega) do ponto. Sua filha *VerticePDCEJT*, por sua vez, adiciona um valor que indica se o ponto pode ou não ser removido de sua rota, seguindo de acordo com restrições de dinamicidade do PDCEJT.

A representação de um ponto como vértice permite a adição de características específicas do ponto como suas janelas de tempo e demanda. Ao mesmo tempo, a utilização da classe *Vertice* como meio de acesso ao objeto propicia uma chamada padronizada de seu construtor.

5.1.5 Rotas

A abstração das rotas pode ser feita de forma semelhante a dos vértices. Como pode ser visto na Figura 5.5, tem-se a classe *Rota* como meio de acesso a objetos do tipo *ClasseRota*, que representam a rota de um determinado problema. Cada rota possui um valor de identificação representado pela ordem de de visita dos pontos. Após inserções e remoções, esse identificador é atualizado pela chamada do método *calculaId*.

Figura 5.5 – Diagrama de Classes Simplificado para representar rotas



Fonte: Do autor (2022).

Os métodos abstratos *insere* e *remove* são responsáveis, respectivamente, pela inserção e remoção de pedidos nas rotas. A forma como um pedido é inserido ou removido depende da representação de um pedido. Para o PCEJT e suas variantes, um pedido é um par de pontos, porém em um problema sem coleta ou sem entrega, o pedido pode ser representado por um único ponto. O mesmo é válido para a representação de suas posições e, por isso, os métodos *retornaPedido* e *retornaPosicao* também são abstratos.

Durante a execução de heurísticas, pode ocorrer a necessidade de se clonar um objeto de forma que alterações na nova rota não alterem a original. Devido a isso, o método *copiaRota* deve ser implementado pelas classes filhas. A classe *RotaPCEJT* implementa esse e os outros

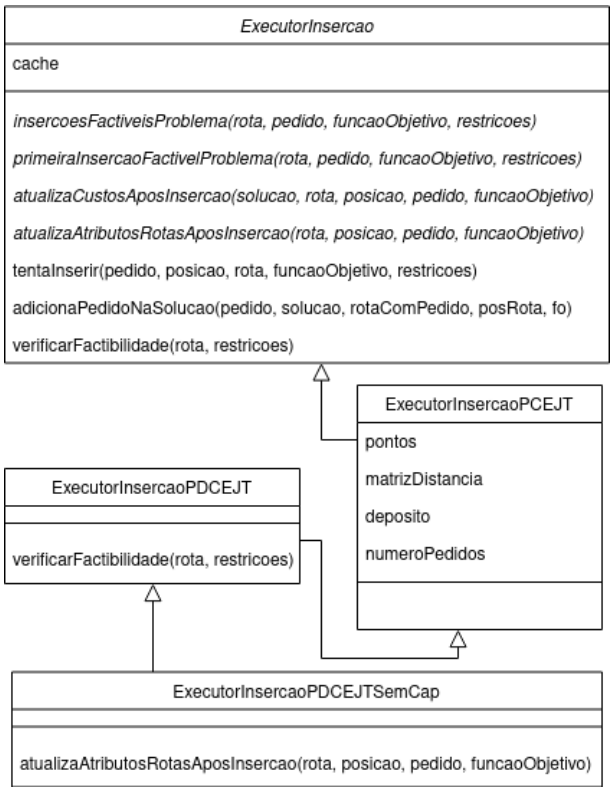
métodos classe rota para o PCEJT, enquanto sua filha *RotaPDCEJT* adiciona um atributo para indicar o ponto da rota de onde o veículo partirá no contexto do PDCEJT.

Assim como a classe um *Vertice*, a classe *Rota* permite uma chamada padronizada do construtor e a criação de rotas durante os métodos de solução podem ser feitas de forma mais genérica. A modificação de rotas também fica padronizada devido aos métodos abstratos das classes, facilitando a generalização da inserção e remoção de pedidos.

5.1.6 Operadores de Rota

As principais operações que modificam uma rota são a inserção e remoção de pedidos. Essas operações podem ser separadas das heurísticas e abstraídas das rotas para que verificações de factibilidade e cálculos de custos sejam feitos separadamente.

Figura 5.6 – Diagrama de Classes Simplificado para representar os as classes que realizam a inserção de pedidos em rotas



Fonte: Do autor (2022).

A Figura 5.6 apresenta a representação de classes que realizam operações de inserção nas rotas. A classe abstrata *ExecutorInsercao* tem como atributo uma *cache* que armazena possibilidades de inserção de um pedido em uma rota. Essa *cache* é preenchida na medida que

tentativas de inserções são feitas com o método *tentaInserir*. Esse método realiza a inserção de um pedido em uma posição específica de uma rota chamando, em seguida, o método *verificaFactibilidade* que verifica se a rota é factível com o método *rotaFactivel* de cada um dos objetos da classe *Restrição* (Seção 5.1.2) recebidas por parâmetro. Ela também atualiza o custo da rota de acordo com o objeto da classe (Seção 5.1.3) que for passado na chamada do método.

Os métodos abstratos *insercoesFactiveisProblema* e *primeiraInsercaoFactivelProblema* foram criados para buscar inserções factíveis de acordo com cada problema. O primeiro retorna todas as posições factíveis da rota, enquanto o segundo retorna apenas a primeira posição factível encontrada. Esses métodos poderm ser utilizados por heurísticas de inserção tais como a Heurística de Arrependimento e a Primeira Inserção.

Os métodos *atualizaCustoAposInsercao* é abstrato pelo mesmo motivo das duas anteriores. Ele é responsável por atualizar o custo do pedido e o custo da solução. Esse método junto a *insercoesFactiveisProblema* e *primeiraInsercaoFactivelProblema* são abstratos pois pedidos podem ser representados de diferentes formas, o que pode modificar o cálculo seu custo também pode variar.

O último método abstrato, *atualizaAtributosRotasAposInsercao*, deve ser implementado de forma que os atributos de rota específicos de cada problema sejam atualizados. Por exemplo, na Figura 5.5, percebe-se que a classe *RotasPCEJT* tem os atributos *chegadasNosPontos* e *capacidadeNosPontos*. Esses atributos armazenam, respectivamente, o instante em que o veículo chega em cada ponto e a capacidade do veículo que é ocupada após a finalização do serviço. Toda vez que uma rota é modificada, esses atributos devem ser atualizados para considerar os pontos do novo pedido. A atualização é realizada pelo método *atualizaAtributosRotasAposInsercao*.

O método *adicionaPedidosNaSolucao*, por sua vez, é responsável por alterar a solução. O método *tentaInserir* gera uma nova rota com o pedido adicionado. Contudo, essa rota não é alterada diretamente na solução para evitar que seja necessário desfazer a operação a fim de remover infactibilidades. Ao ser chamado, o método *adicionarPedidosNaSolucao* substitui a rota na qual o vértice foi inserido e adiciona o pedido no conjunto de pedidos do objeto da classe *Solucao* (Seção 5.1.7).

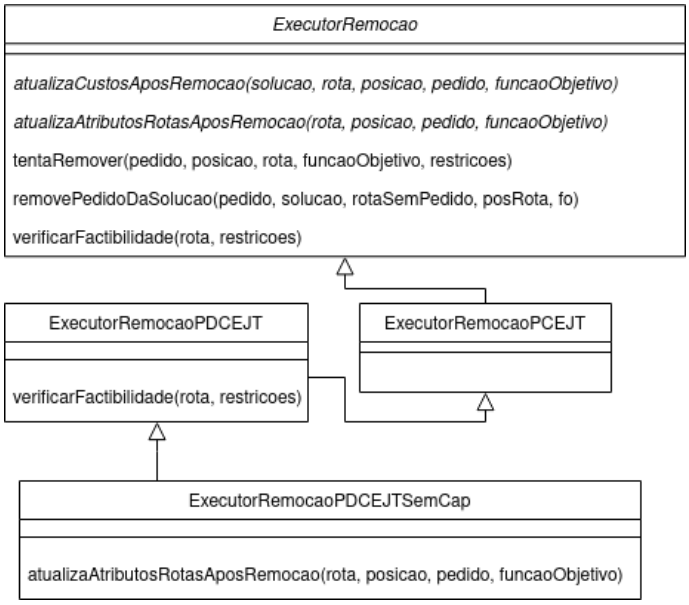
A classe filha *ExecutorInsercaoPCEJT* implementa os métodos abstratos da classe *ExecutorInsercao*, utilizando dos atributos relacionados ao PCEJT para a realização das inserções

e atualizações de valores. A classe *ExecutorInsercaoPDCEJT*, por sua vez, sobrescreve apenas o método *verificarFactibilidade* implementado pela classe *ExecutorInsercao*. Ele altera a verificação para que ela analise a viabilidade das rotas a partir do ponto de partida do veículo. Note que isso reduz o processamento de restrições que podem precisar verificar a rota toda para assegurar a factibilidade.

Por fim, a classe *ExecutorInsercaoPDCEJTSemCap* sobrescreve o método *atualizaAtributosRotasAposInsercao* implementado no *ExecutorInsercaoPCEJT*, para que a atualização do atributo *capacidadeNosPontos* seja ignorada, evitando processamento desnecessário.

Opondo-se ao *ExecutorInsercao*, tem-se o *ExecutorRemocao*, que é responsável pela remoção de pedidos das rotas. A maioria de seus métodos funcionam de forma semelhante ao do *ExecutorInsercao*. Os métodos abstratos *atualizaCustoAposRemocao* é análoga ao método *atualizaCustoAposInsercao*, porém considerando a remoção de um pedido ao invés da inserção. O mesmo é válido para os outros métodos *atualizaAtributosRotasAposRemocao* e *atualizaAtributosRotasAposInsercao*, *removerPedidosNaSolucao* e *adicionarPedidosNaSolucao*, e *tentaRemover* e *tentaInserir*. Os métodos *verificarFactibilidade* funcionam da mesma forma em ambos *ExecutorInsercao* e *ExecutorRemocao*.

Figura 5.7 – Diagrama de Classes Simplificado para representar os operadores de remoção de pedidos das rotas



Fonte: Do autor (2022).

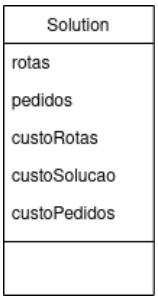
Os métodos sobrescritos nas descendentes de *ExecutorRemocao* também seguem a mesma ideia dos métodos descendentes de *ExecutorInsercao*: *verificarFactibilidade* é alterado em *ExecutorRemocaoPDCEJT* para verificar apenas a partir do ponto de partida do veículo e *atualizaAtributosRotasAposRemocao* em *ExecutorRemocaoPDCEJTSemCap* para que a atualização da capacidade seja ignorada.

As classes *ExecutorInsercao* e *ExecutorRemocao* diminuem o impacto causado por modificações em heurísticas ou nas rotas, visto que o algoritmo para realização das alterações nas rotas podem não precisar sofrer alterações. Por exemplo, como será visto na Seção 5.3, para representação da rota para o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo com pontos Urbanos e Rurais (PDCEJT/UR), há a adição de novos atributos nas rotas e há modificações em partes dos métodos de solução. Contudo, não é necessário adicionar novas modificações aos métodos das classes (ou suas descendentes) *ExecutorInsercao* e *ExecutorRemocao*.

5.1.7 Solução

Na implementação atual, uma solução é representada pela classe apresentada na Figura 5.8. Uma solução armazena as rotas que a compõe e os pedidos que compõem as rotas. Além disso, são armazenados os custos de cada pedido em cada rota, os custos de viagens de cada uma das rotas e o custo da solução.

Figura 5.8 – Diagrama de Classes Simplificado para representar uma solução.



Fonte: Do autor (2022).

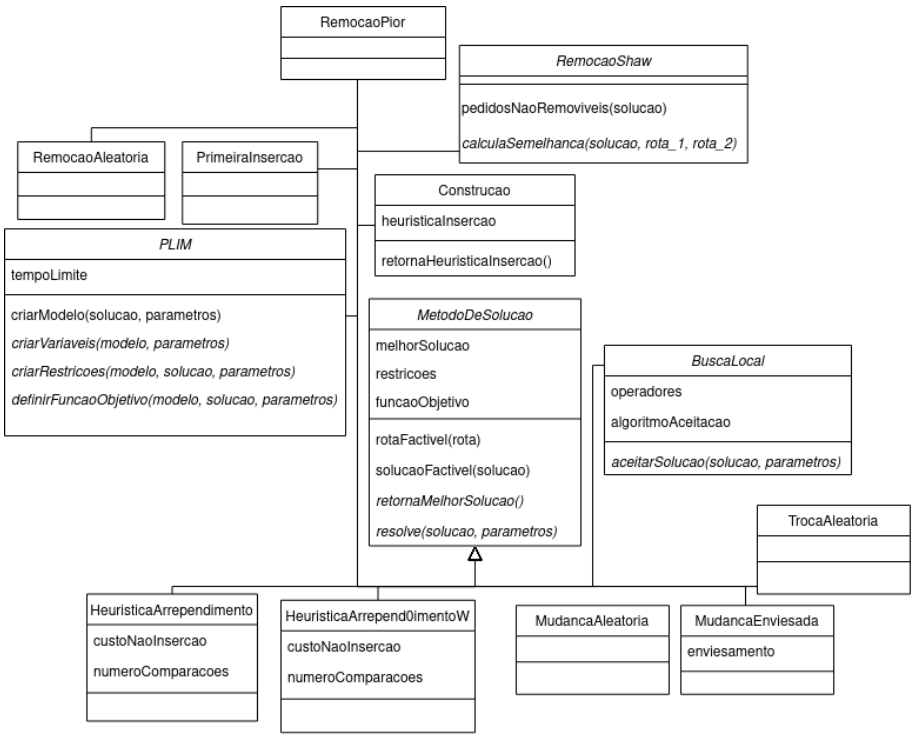
Acredita-se que a representação atual consegue ser utilizada para uma grande variedade de heurísticas e problemas e, por isso, a classe ainda não foi abstraída. Apesar disso, pretende-se generalizar a representação da solução utilizando uma abordagem semelhante às classes *Vertice*

e *Rota* (Seções 5.1.4 e 5.1.5), utilizando uma classe de externa como meio de acesso aos objetos que representam as soluções.

5.1.8 Métodos de Solução

Todo método que gera uma solução para um problema de roteamento pode ser abstraído como um objeto. Isso é válido até mesmo para algoritmos que geram soluções infactíveis, como por exemplo a Remoção Aleatória para o PDCEJT, que gera soluções que não respeitam a restrição de atendimento de todos os pedidos. Logo, todas as heurísticas propostas no Capítulo 4 podem ser vistas como uma única classe *MetodoDeSolucao*. A Figura 5.9 apresenta essa classe e os métodos de geração de solução que dela descendem.

Figura 5.9 – Diagrama de Classes Simplificado para representar a generalização os métodos de solução.



Fonte: Do autor (2022).

Na abstração feita, um método de solução tem três principais atributos: um objeto da classe *FuncaoObjetivo*, uma lista de objetos da classe *Restricao* e um objeto da classe *Solucao*. O primeiro é usado para calcular os custos da solução ou para guiar a heurística na direção da otimalidade. Para diferentes métodos de solução, podem ser atribuídas diferentes *FuncoesObjetivos*. Por exemplo, pode ser permitido que a classe *LNS* utilize uma função objetivo

que minimize a distância e que a *AGES* minimize o número de veículos. Logo, permite-se que diferentes componentes de uma metaheurística busquem a otimização de aspectos diferentes do problema.

O mesmo é válido para a lista de objetos da *Restricao*. Não faria sentido, por exemplo, que heurística de Remoção Aleatória, representada pela classe *RemocaoAleatoria*, precisasse verificar se todos os pedidos são atendidos, visto que a heurística busca remover pedidos da solução.

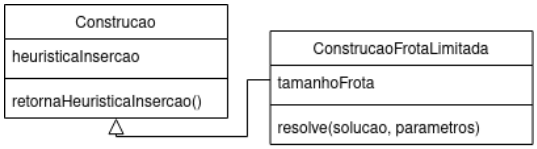
O atributo *melhorSolucao*, objeto da classe *Solucao*, por sua vez, tem função diferente. A execução do programa é limitada por tempo e, quando seu limite é alcançado, a execução é interrompida. O método abstrato *retornaMelhorSolucao* é chamado e a melhor solução é então buscada na heurística utilizada para solução e então, retornada para que possa ser salva em um arquivo externo ao programa. Por padrão, o método *retornaMelhorSolucao* foi implementado na classe pai e retorna o atributo *melhorSolucao*. Contudo, esse método pode ser modificado, por exemplo, para se buscar a melhor solução nas componentes da *matheuristic* de Sartori e Buriol (2020).

Entre os procedimentos de um *MetodoDeSolucao*, tem-se *rotaFactivel* e *solucaoFactivel* que realizam, respectivamente, as verificações de factibilidade da rota e da solução. O método *resolve*, por sua vez, é o mais importante, pois ele é chamado para que seja feita a resolução do problema. Esse método espera a recepção dos parâmetros necessários para a execução do algoritmo implementado e deve retornar um objeto da classe *Solucao*.

Os métodos implementados pelas descendentes do *MetodoDeSolucao* permitem que componentes de heurísticas sejam criadas sem grande esforço, sendo necessário trocar somente os parâmetros de entrada do algoritmo.

Destaca-se algumas classes filhas que estão presentes na Figura 5.9. A primeira delas é a classe *Construcao*. Nessa classe foi implementada a heurística de construção proposta por Sartori e Buriol (2020), apresentada na Seção 4.1, e o atributo *heuristicaInsercao* indica se será usado a Heurística de Arrependimento ou a Primeira Inserção para a escolha das posições de inserção de pedidos nas rotas. Como pode ser vista na Figura 5.10, o método *resolve* implementado nessa classe é sobrescrito pela classe *ConstrucaoFrotaLimitada* para que a heurística de construção tenha uma frota limitada.

Figura 5.10 – Diagrama de Classes Simplificado para representar a generalização dos algoritmos de construção.

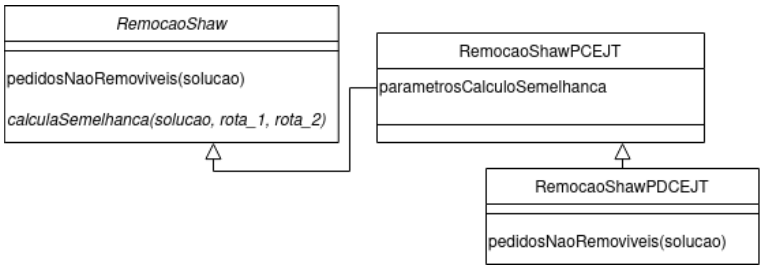


Fonte: Do autor (2022).

A classe *RemocaoShaw*, por sua vez, implementa uma classe abstrata para representar a heurística Remoção de Shaw, apresentada na Seção 4.1.3. A heurística pode ser utilizada para diversas variantes do PRV ao se variar a forma como o cálculo de relação entre pedidos é feito. A modificação desse cálculo pode ser feita sobrescrevendo o método *calculaSemelhanca*. Como pode ser visto na Figura 5.11, esse método é reimplementado pela sua classe filha *RemocaoShawPCEJT*, que calcula a semelhança entre pedidos para o PCEJT de acordo com o proposto por Sartori e Buriol (2020), explicado na Seção 4.2.6.

O método abstrato *pedidosNaoRemoviveis*, por sua vez foi adicionado para proibir que um certo subconjunto de pedidos se tornem candidatos para a remoção de suas rotas. Na Figura 5.11, pode ser visto que esse método é sobrescrito pela classe *RemocaoShawPDCEJT* a fim de evitar que pedidos fixos sejam candidatos a remoção.

Figura 5.11 – Diagrama de Classes Simplificado para representar a generalização da heurística Remoção de Shaw.

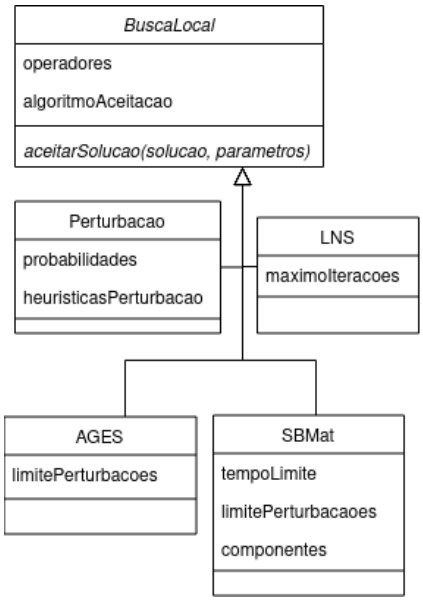


Fonte: Do autor (2022).

A classe abstrata *BuscaLocal*, herdeira de *MetodoDeSolucao*, representa buscas locais ou meta-heurísticas que visam gerar novas soluções a partir de uma já construída. A Figura 5.12 apresenta suas classes filhas, sendo elas as representações das componentes AGES e LNS (classes *AGES* e *LNS*), da heurística de perturbação implementada (classe *Perturbacao*) e da metaheurística proposta por Sartori e Buriol (2020) (classe *SBMat*). A grande diferença entre um objeto da classe *BuscaLocal* para um objeto da classe *MetodoDeSolucao* é que uma *Bus-*

caLocal conta com operadores para realizar busca em vizinhanças, tais como as heurísticas de remoção da LNS ou as componentes da meta-heurística de Sartori e Buriol (2020), sendo eles descendentes da classe *MetodoDeSolucao*.

Figura 5.12 – Diagrama de Classes Simplificado para representar a generalização métodos de geração de novas soluções, como perturbações, buscas locais e meta-heurísticas.

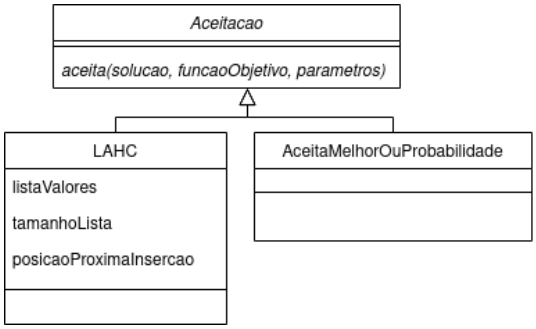


Fonte: Do autor (2022).

A segunda diferença é a necessidade de um algoritmo para a aceitação ou recusa de soluções. Esse algoritmo também pode ser visto como uma entidade e, consequentemente, ser representado como um objeto. A Figura 5.13 apresenta a estrutura de classes para a implementação de algoritmos para aceitação de soluções. A classe pai *Aceitacao* contém o método *aceita*, que deve ser implementado de forma a verificar se uma solução será aceita ou recusada. As duas heurísticas de aceitação do trabalho Sartori e Buriol (2020), apresentadas na Seção 4.1.5 deste trabalho, foram implementadas como filhas da classe *Aceitacao*. Uma classe filha genérica que aceita qualquer solução (omitida da Figura 5.13), também foi implementada para permitir que algoritmos que permitem a piora sem critérios, como por exemplo, na heurística de perturbação implementada (representada pela classe *Perturbacao* na Figura 5.12).

A classe abstrata *PLIM*, também herdeira de *MetodoDeSolucao*, foi implementada como uma forma de generalizar o Programa Linear Inteiro Misto proposto por Dumas, Desrosiers e Soumis (1991), apresentado na Seção 4.1.4. Entre os parâmetros recebidos pelo método *resolve* sobrescrito pelo *PLIM*, deve estar presente o conjunto de rotas factíveis $\mathbb{P}$, para que o parti-

Figura 5.13 – Diagrama de Classes Simplificado para representar a generalização de algoritmos de aceitação de soluções.



Fonte: Do autor (2022).

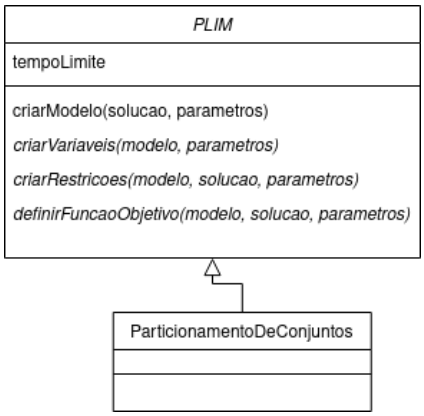
cionamento possa ser realizado. Como um PLIM é específico para cada problema, objetos da classe *Restricao* e *FuncaoObjetivo* não podem ser utilizados durante a execução da componente. Portanto, os seguintes métodos definem suas respectivas etapas na criação do PLIM:

a) *criarVariaveis*: cria as variáveis do PLIM;

b) *criarRestricoes*: cria as restrições lineares do PLIM;

c) *definirFuncaoObjetivo*: define a função objetivo do PLIM.

Figura 5.14 – Diagrama de Classes Simplificado para representar a generalização do PLIM proposto por Dumas, Desrosiers e Soumis (1991).



Fonte: Do autor (2022).

Como será visto na Seção 5.2, a implementação dos modelos matemáticos foi feita com o pacote Python-MIP (<https://www.python-mip.com/> que permite a utilização de diferentes algoritmos de otimização para a solução de Problemas de Otimização. No método *criarModelo*, um objeto da classe *Model* importada desse pacote é inicializado. Portanto as classes que

herdam de PLIM devem implementar os métodos abstratos *criarVariaveis*, *criarRestricoes* e *definirFuncaoObjetivo*, seguindo utilizando os recursos do Python-MIP.

Como pode ser visto na Figura 5.14, a classe *ParticionamentoDeConjuntos* herda de *PLIM*, implementando seus métodos abstratos de acordo com a formulação matemática de Dumas, Desrosiers e Soumis (1991), apresentada na Seção 4.1.4.

As generalizações feitas para os algoritmos, heurísticas e meta-heurísticas permitem flexibilidade para a componentização dos próprios métodos. Por exemplo, a meta-heurística implementada pela classe *SBMat* utiliza-se de objetos das classes *LNS*, *AGES*, *PLIM* e *Perturbacao* como operadores, enquanto a classe *Perturbacao* utiliza-se como operadores objetos das classes *TrocaAleatoria*, *MudancaAleatoria* e *MudancaEnviesada*, que representam as heurísticas Troca Aleatória, Mudança Aleatória e Mudança Enviesada (Seção 4.1.6). Isso permite o reaproveitamento de heurísticas como partes umas das outras.

A única dificuldade da generalização são os parâmetros recebidos pelo método *resolve*, pois podem haver variações dependendo de cada método de solução. Por exemplo, na classe *PLIM*, o método espera o conjunto de rotas $\mathbb{P}$, enquanto na classe *RemoçãoShaw* espera-se o número de pedidos a serem removidos. Esse fato traz dificuldades para a componentização, porém isso só ocorre para parâmetros que são definidos durante a execução do algoritmo de solução, pois parâmetros com valores constantes podem ser atribuídos durante a criação dos objetos de cada classe, como será visto na Seção 5.2.

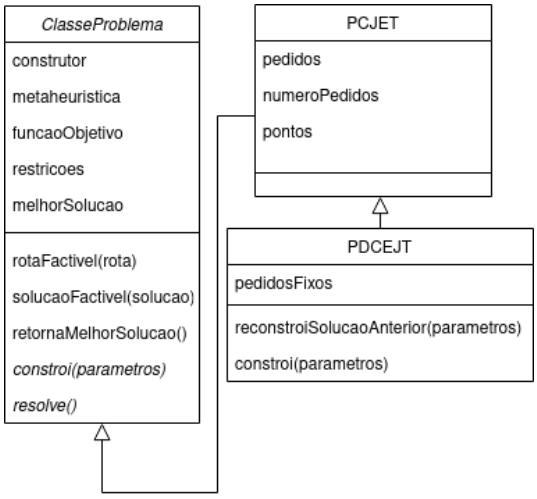
5.1.9 Problema

Na implementação feita, cada problema a ser resolvido é representado por uma classe do tipo *ClasseProblema*. A função principal dos objetos dessa classe é realizar a conexão entre o construtor e a meta-heurística.

Como pode ser visto na Figura 5.15, as filhas dessa classe herdam dois métodos abstratos: *constroi* e *resolve*. O primeiro deve inicializar uma variável do tipo *Solucao* e chamar o algoritmo de construção do problema. Esse algoritmo deve ser um objeto do tipo *MetodoDeSoucao* e deve estará armazenado no atributo *construtor*. A chamada é padronizada como *construtor.resolve(solucao, parametros)*.

O método *resolve*, por sua vez, liga a construção à meta-heurística. A solução retornada pelo método abstrato *constroi* é utilizada na chamada do objeto do tipo *BuscaLocal*, armazenado

Figura 5.15 – Diagrama de Classes Simplificado para representar a estrutura de classes que representa o problema a ser resolvido.



Fonte: Do autor (2022).

no atributo *metaheuristica*. Analogamente à chamada do algoritmo de construção, a busca local para a melhoria da solução tem chamada padronizada *metaheuristica.resolve(solucao, parametros)*.

Os métodos não abstratos *rotaFactivel* e *solucaoFactivel* são responsáveis pela verificação de factibilidade das soluções retornadas pelo construtor e pela busca local. A verificação é de acordo com uma lista de objetos do tipo *Restricao* armazenada no atributo *restricoes*. O método *retornaMelhorSolucao* deve retornar a solução final do problema. Na implementação atual, retorna a melhor solução obtida a partir da comparação da solução do construtor com a solução retornada pelo método *retornaMelhorSolucao* do objeto armazenado em *metaheuristica*. A *ClasseProblema* também armazena um objeto do tipo *FuncaoObjetivo*, permitindo que o cálculo de custos de pedidos, rotas ou soluções sejam feitas.

A classe *PCEJT* sobrescreve os métodos de *ClasseProblema* para resolver o Problema de Coleta e Entrega com Janelas de Tempo no contexto estático. Sua filha, *PDCEJT*, sobrescreve seu construtor para que ele faça a reconstrução da solução obtida na execução do período anterior a partir do método *reconstroiSolucaoAnterior*.

A classe problema tem como premissa fazer a ligação do método de construção com a heurística de melhoramento de solução. Com as chamadas generalizadas a partir dos atributos da classe, facilita-se a modificação da estratégia de solução utilizada para cada problema. A implementação dessa classe é necessária somente se houver modificações nos parâmetros pas-

sados para os métodos de solução, como por exemplo, a necessidade de reconstrução da solução anterior do PDCEJT.

5.2 Detalhes de Implementação

A linguagem *Python* foi escolhida devido a sua flexibilidade na implementação. A principal vantagem que a linguagem apresenta para a generalização é permitir a instanciação de objetos de uma classe a partir de uma *string* com seu nome. Isso é possível devido à função *built-in getattr*, que retorna o valor de um atributo dado um objeto e o nome do atributo a ser procurado. Por exemplo: *getattr(Solucao(), "pedidos")* retorna os pedidos armazenados no atributo *pedidos* de um objeto da classe *Solucao* (Figura 5.8, Seção 5.1.7). Isso também é possível para arquivos, já que todo *script* em *Python* é visto como um módulo, que por sua vez é um objeto.

Figura 5.16 – Código para a instanciação de objetos a partir de *strings*.

```
import src
def create_class_by_name(class_name, class_data):
    class_type = getattr(
        src,
        class_name
    )
    class_object = class_type()
    for attribute, value in class_data.items():
        if (getattr(class_object, attribute) is None):
            class_object.set_attribute(attribute, value)
    return class_object
```

Fonte: Do autor (2022).

A criação de uma instância das classes apresentadas na Seção 5.1 é feita com o código apresentado na Figura 5.16, que pode ser encontrado no arquivo *objects_creation_manager.py*, presente no repositório <https://github.com/thuzax/vrp-solver/blob/dev/solver/src/>. Como pode ser visto na figura, importa-se inicialmente o módulo *src*, que é um diretório que contém as classes para a resolução do problema e *scripts* com métodos auxiliares. A função *create_class_by_name* recebe como parâmetros o nome da classe a ser inicializada e um dicionário com atributos que representam os parâmetros de entrada da classe, tais como o nome da função objetivo de um *MetodoDeSolucao* (Seção 5.1.8). A criação, contudo, exige que o módulo a ser

buscado possa ser identificado no diretório *src*. Por isso é necessário a definição do módulos nos seus respectivos arquivos *__init__.py* (<https://docs.python.org/3/tutorial/modules.html>).

A classe é buscada pelo método *getattr* e seu objeto é instanciado. Após isso, o dicionário é iterado e seus valores são atribuídos aos atributos do objeto criado. Note que os atributos também são buscados pela função *getattr*. Com a instanciação de classes utilizando uma *string*, permite-se a criação das classes seja feita a partir de um arquivo de configuração passado como entrada.

No código desenvolvido, um arquivo de configuração deve ser passado como entrada e deve conter as classes que são usadas para resolver o problema. O arquivo é do tipo *json* (<https://www.json.org/json-en.html>), pois ele permite a descrição de itens complexos como classes e dicionários a partir de forma serializada e padronizada. Sua descrição pode ser feita por uma lista ou a partir de coleções composta por um par chave/valor. O primeiro se comporta como um dicionário da linguagem *Python*, enquanto o segundo se comporta como uma lista.

O arquivo *json* utilizado como configuração deve conter uma coleção com as seguintes chaves:

a) *objective*: conjunto classes que representam as *FuncoesObjetivos* (Seção 5.1.3) a serem utilizadas.
b) *solution_methods*: conjunto de algoritmos, heurísticas, meta-heurísticas e outros *MetodosDeSolucao* (Seção 5.1.8);
c) *constraints*: conjunto de classes *Restricao* (Seção 5.1.2) do problema a ser resolvido;
d) *solver*: classe *ClasseProblema* (Seção 5.1.9) para o problema a ser resolvido;
e) *reader*: classe *Leitor* usado para a leitura das instâncias de entrada;
f) *writer*: classe para a escrita de dados (recomenda-se a classe *WriterLiLimPDPTW*);
g) *route_class*: classe utilizada para a representação das rotas (Seção 5.1.5);
h) *route_class*: classe utilizada para a representação de vértices (Seção 5.1.4);
i) *insertion_operator*: classe *ExecutorInsercao* (Seção 5.1.6), que será utilizada para fazer a inserção de um pedido em um rota;
j) *removal_operator*: classe *ExecutorRemocao* (Seção 5.1.6), que será utilizada para fazer a remoção de um pedido em um rota.

Dentro de cada uma dessas coleções, são passadas outras coleções, cada uma representando uma classe. Em geral, todas as classes podem ser descritas no arquivo de configuração

seguindo o formato da coleção apresentado pela Figura 5.17. A chave que representa cada uma delas é o nome da classe e seus atributos são listados dentro de outra coleção. Os valores são atribuídos durante a criação do objeto, feita pela função apresentada pela Figura 5.16.

Figura 5.17 – Código para a instanciação de objetos a partir de *strings*.

```
"nome da classe" : {
    "atributo 1" : valor,
    "atributo 2" : valor,
    ...
}
```

Fonte: Do autor (2022).

A Figura 5.18 apresenta um exemplo simplificado de um arquivo de configuração. Note que apenas três das chaves listadas anteriormente foram utilizadas, mesmo que todas sejam obrigatórias. Além disso, parte dos parâmetros das classes a serem geradas foram omitidos, sendo que os nomes das classes e atributos apresentadas foram adaptados para ficarem coerentes com as da Seção 5.1. Um exemplo completo de um arquivo de configuração para o PCEJT estático pode ser encontrado em <https://github.com/thuzax/vrp-solver/blob/dev/solver/configuration-static.json>. Apresenta-se, também, um exemplo para o PDCEJT, por meio do arquivo *config_dpdptw_no_cap_with_ages.json*, encontrado em <https://github.com/thuzax/vrp-solver/blob/dev/server_solver/configurations/>.

A Figura 5.18 apresenta como função objetivo a classe *FuncaoObjetivoPCEJT* (Figura 5.3, Seção 5.1.3). Como única restrição do problema tem-se a classe *AtendeTodosPedidos* (Figura 5.2, Seção 5.1.5). Por fim, como método de solução, utiliza-se da *BuscaLocal LNS* (Figuras 5.9 e 5.12, Seção 5.1.8), cujo o atributo *funcaoObjetivo* será um objeto da classe *FuncaoObjetivoPCEJT*, o atributo *restricoes* será uma lista contendo apenas um objeto da classe *AtendeTodosPedidos* e o atributo *algoritmoAceitacao* será um objeto da classe *LAHC* (Figura 5.13, Seção 5.1.8). O algoritmo de aceitação, por sua vez, tem o valor 3, passado como parâmetro ao atributo *tamanhoLista*.

Os atributos presentes no arquivo de configuração são constantes relacionadas às suas respectivas classes, como o número de iterações de uma heurística ou o nome de uma busca local de uma meta-heurística. Dados relacionados à instância são lidos através da classe passada na coleção *reader* do arquivo de configurações. Para acessar esses dados, uma heurística deve implementar dois métodos abstratos: *inicializa_atributos* e *relacao_de_atributos_leitor*.

Figura 5.18 – Exemplo simplificado do arquivo de configuração.

```
"objective" : {
    "FuncaoObjetivoPCEJT" : {
    }
},
"constraints" : {
    "AtendeTodosPedidos" : {
    }
},
"solution_methods": {
    "LNS" : {
        "funcaoObjetivo" : "FuncaoObjetivoPCEJT",
        "restricoes" : [
            "AtendeTodosPedidos"
        ],
        "algoritmoAceitacao" : {
            "LAHC" : {
                "tamanhoLista" : 3
            }
        }
    }
}
```

Fonte: Do autor (2022).

O primeiro deve implementar a criação e inicialização das variáveis da classe. O segundo deve retornar um dicionário cuja chave é o nome do atributo do objeto *Leitor* do problema e o valor é o nome do atributo na classe. Com essas funções implementadas, faz-se a passagem de parâmetros que sempre serão utilizados na classe. Por exemplo, a classe *JanelasDeTempo* sempre precisará das janelas de tempo para a execução de sua verificação, mas com o método *relacao_de_atributos_leitor*, este valor poderá ser armazenado no objeto durante a criação das classes do arquivo de configurações.

A instanciação das classes por *strings* permite que todas as classes sejam inicializadas por um mesmo atributo. Além disso, utilizando do padrão *json*, é possível criar uma forma padronizada para a definições das classes a serem utilizadas na solução de um determinado problema. O mesmo é válido para a atribuição de parâmetros. Valores constantes podem ser passados no arquivo de configuração e valores de instâncias podem ser obtidos através do método *relacao_de_atributos_leitor*.

5.3 Exemplo com o PDCEJT/UR

O Problema Dinâmico de Coleta e Entrega com Janelas de Tempo e pontos Urbanos e Rurais (PDCEJT/UR) é uma variante do PDCEJT baseada em um caso real onde são considerados pontos de coleta ou entrega que estão na zona rural cujo acesso de certos veículos não é possível. Esse problema pode ser visto como um caso específico problema com frota heterogênea (como o estudado por Ropke e Pisinger (2006)), contudo existem apenas dois tipos de veículos: (i) que conseguem atender pedidos urbanos e rurais e (ii) que conseguem atender somente pontos urbanos.

O objetivo do PDCEJT/UR estudado neste trabalho é, em primeira instância, maximização de pedidos considerando uma fota limitada. Secundariamente, minimiza-se a distância total percorrida. Consequentemente, a restrição de atendimento de todos os pedidos é removida. Este problema é uma versão simplificada do caso real, o qual inspirou não somente o estudo em problemas dinâmicos de roteamento, mas também o desenvolvimento da ferramenta apresentada neste capítulo. Devido a isso, o PDCEJT/UR foi escolhido para exemplificar a generalização do código.

Na sequência, na Seção 5.3.1, explica-se as mudanças necessárias na metodologia apresentada para o PDCEJT (Seção 4.2). A Seção 5.3.2 apresenta como a generalização da implementação auxiliou na modificação dos métodos já implementados.

5.3.1 Adaptações do Método de Solução

O método usado para a resolução foi o mesmo aplicado para o PDCEJT (Seção 4.2). Contudo, foi necessário realizar algumas adaptações. Altera-se a heurística construtiva e o Modelo Linear Intero Misto (PLIM), apresentados, respectivamente, na Seção 4.2.2 e na Seção 4.1.4. A modificação da heurística de construção se deu para que as rotas fossem criadas nas quantidades de acordo com o tamanho da frota de cada tipo.

O PLIM, por sua vez, teve de ser adaptado para o novo PDCEJT/UR. Foi preciso mudar sua função objetivo, além de adicionar restrições relacionadas à frota. Portanto, define-se o seguinte PLIM para o PDCEJT/UR da seguinte forma:

$$\max \sum_{r \in R} x_r - \mu \frac{\sum_{k=1}^{|\mathbb{P}|} C(\mathbb{P}_k) y_k}{\sum_{f \in F} C(S_f)} \tag{5.1}$$

$$\sum_{k \in \mathscr{P}_t} y_k \leq \mathbb{F}_t, t \in \{1...T\}; \tag{5.2}$$

$$x_r = \sum_{k \in \Delta_r} y_k, \forall r \in R \tag{5.3}$$

$$x_r = 1, \forall r \in R^- \tag{5.4}$$

$$x_r \in \{0,1\}, \forall r \in R \tag{5.5}$$

$$y_k \in \{0,1\}, k = 1... \mid \mathbb{P} \mid \tag{5.6}$$

Os parâmetros presentes no programa são listados no Quadro 5.1. Como variáveis do PLIM, tem-se x_r que é igual a 1 se o pedido $r \in R$ foi atendido pela solução, ou 0 caso contrário. A variável y_k é igual a 1 se a *k-ésima* rota de $\mathbb{P}$ está na solução, ou 0 caso não esteja.

Quadro 5.1 – Parâmetros necessários para a o programa linear adaptado

Parâmetro	**Significado**
R	conjunto de todos os pedidos;
R^-	subconjunto de pedidos fixos ($R^- \subset R$);
$\mathbb{P}$	conjunto de rotas factíveis geradas ao longo da meta-heurística;
Δ_{rf}	igual a 1 se o pedido $r \in R$, é atendido pela rota $k \in \mathbb{P}$, 0 c.c.;
$C(g)$	custo de uma rota qualquer g;
F	frota de veículos;
S	solução inicial;
μ	valor suficientemente pequeno;
$\mathscr{P}_t$	rotas atendidas por um veículo do tipo t, tal que $\mathscr{P}_t \in \mathbb{P}$;
$\mathbb{F}_t$	tamanho da frota do tipo de veículo $t \in \{1...\mathbb{T}\}$.

Fonte: Do autor (2022).

A função objetivo é descrita pela Função (5.1). Tenta-se contemplar os objetivos hierárquicos do problema: (i) maximização de pedidos e (ii) minimização da distância. A primeira parte da função $\left(\sum_{r \in R} x_r\right)$ soma o número de pedidos atendidos, enquanto a segunda $\left(\frac{\sum_{k=1}^{|\mathbb{P}|} C(\mathbb{P}_k) y_k}{\sum_{f \in F} C(S_f)}\right)$ calcula a razão entre o custo da solução gerada e o custo da solução recebida como parâmetro de entrada no método *resolve* (Seção 5.1.8). A razão é multiplicada por um fator muito pequeno μ para que seu valor esteja entre 0 e 1, diminuindo seu impacto no cálculo do número de pedidos atendidos. Com um valor de μ suficiente pequeno, a prioridade da função pode ser dada para a maximização do número de pedidos atendidos, e o custo das rotas se tornaria um critério de desempate.

As Restrições (5.2) obrigam que a quantidade de veículos do tipo t não ultrapasse o limite estabelecido. É garantido pelas Restrições (5.3) que um pedido não será atendido por mais de uma rota. As Restrições (5.4), por sua vez, forçam a adição dos pedidos fixos à solução. Por fim, as Restrições (5.5) e (5.6) delimitam, respectivamente, o espaço de busca das variáveis $x_r, r \in R$ e $y_k, k = 1, 2..., \mid \mathbb{P} \mid$.

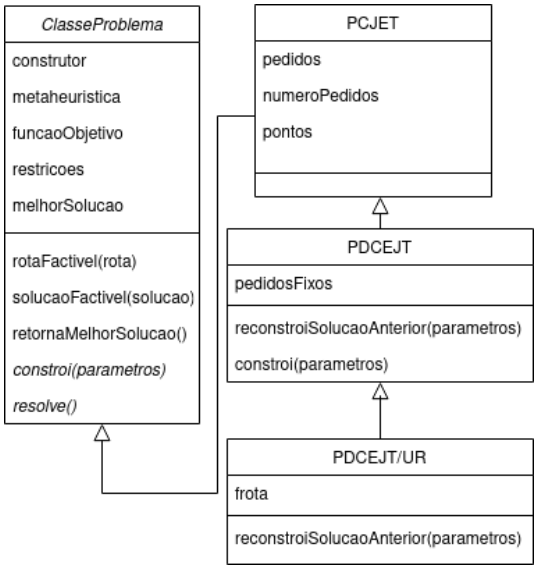
5.3.2 Modificações no Diagrama de Classe

Os métodos discutidos na Seção 5.3.1 são implementados utilizando a generalização apresentada pelas Seções 5.1 e 5.2. A explicação das alterações feitas para atender aos requisitos do PDCEJT/UR são feitas com base nos diagramas e conceitos previamente discutidos.

A primeira modificação feita para resolver o novo problema é a adição de uma classe descendente da *ClasseProblema*. A Figura 5.19 apresenta a nova estruturação das classes que representam o problema (Seção 5.1.9). Foi adicionada a classe *PDCEJT/UR* como filha de *PDCEJT*, com o atributo adicional *frota*, que indica quais tipos de pedidos cada um dos veículos pode atender. Além disso, o método *reconstroiSolucaoAnterior* da classe *PDCEJT* foi sobrescrita para que a reconstrução da solução do período anterior seja feita de acordo com as restrições do PDCEJT/UR.

A classe *Leitor* também teve uma descendente adicionada, como pode ser visto na Figura 5.20. Um objeto do tipo *LeitoriPDCEJT/UR* terá como atributo adicional *frota*, para armazenar dados relacionados a frota limitada, e sobrescreve os métodos *leEntrada* e *criaPontoEspecial*

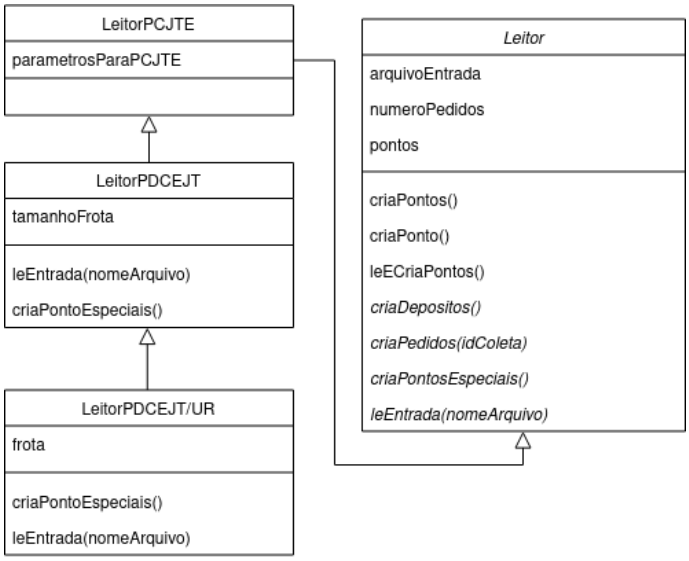
Figura 5.19 – Diagrama de Classes Simplificado com a adição da classe PDCEJT/UR que representa o PDCEJT com pontos Urbanos e Rurais.



Fonte: Do autor (2022).

de seu pai *LeitorPDCEJT*. O objetivo é alterar a leitura e a criação de pontos para adicionar dados relacionados à frota e aos tipos de pedidos.

Figura 5.20 – Diagrama de Classes Simplificado com a adição da classe LeitorPDCEJT/UR que faz a leitura de instâncias do PDCEJT/UR.

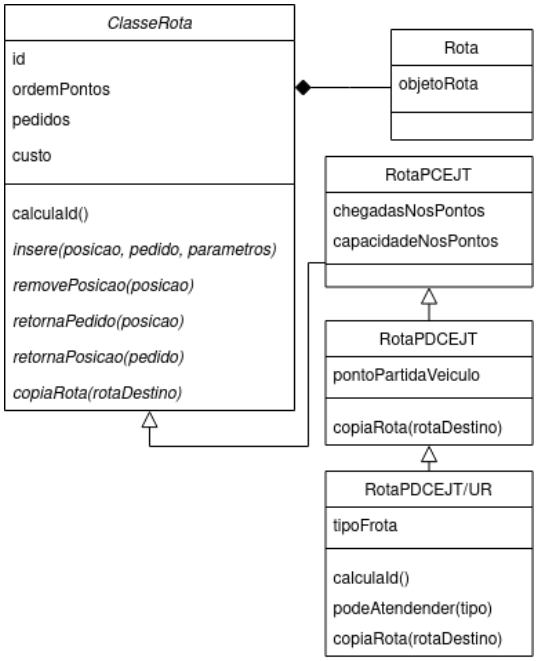


Fonte: Do autor (2022).

As classes *Rota* e *Vertice* também devem ser compatíveis com o novo problema. Assim, através da Figura 5.21, apresenta-se a classe *RotaPDCEJT/UR*. Para a resolução do problema, cada rota teve a adição do tipo de sua frota. Além disso, o cálculo de seu identificador foi

alterado para que haja a diferenciação de rotas de tipos diferentes cujos pontos inseridos sejam os mesmos.

Figura 5.21 – Diagrama de Classes Simplificado com a adição da classe que representa as rotas do PDCEJT/UR.



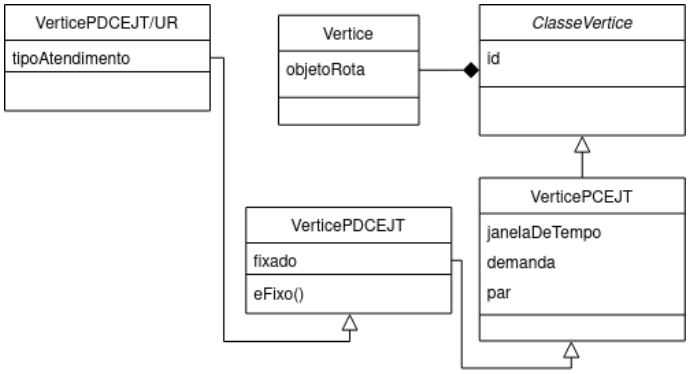
Fonte: Do autor (2022).

O método de cópia foi sobrescrito para que o novo atributo possa ser copiado junto do restante dos dados relacionados à rota. Por fim, um novo método foi adicionado para verificar se um tipo de atendimento pode ser realizado pela rota. Os outros métodos das classes ancestrais de *RotaPDCEJT/UR* são utilizadas em sua implementação original. A Figura 5.22, por sua vez, apresenta a adição da classe *VerticesPDCEJT/UR* para representar um ponto do PDCEJT/UR. A única adição à classe foi um atributo indicando se seu atendimento é urbano ou rural.

A Figura 5.23 apresenta a adição função objetivo para o PDCEJT/UR. A classe *FuncaoObjeivoPedidos* sobrescreve apenas o método *custoSolucao* da classe abstrata *FuncaoObjetivo*, visto que o custo da rota se mantém igual ao da *FuncaoObjetivoPCEJT*. O método *custoSolucao* foi reimplementado a fim de maximizar o número de pedidos atendidos por uma solução.

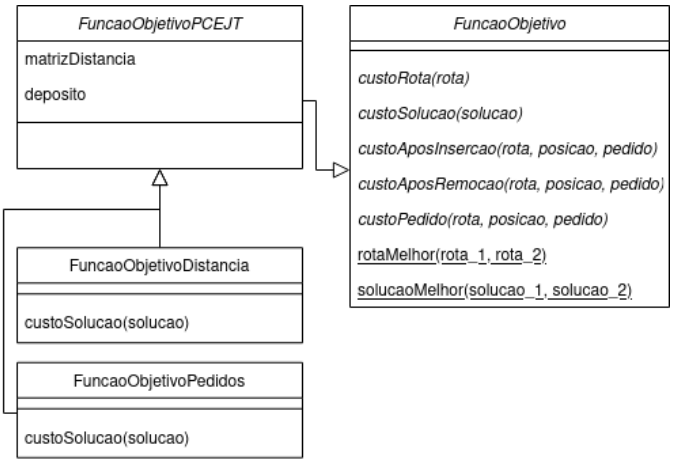
A restrição *FrotaUR*, filha de *Restricao*, implementa os métodos *rotaFactivel* e *solucaoFactivel* de forma a impedir que veículos atendam tipos para os quais não são qualificados. Além disso, o número de rotas com pedidos de um determinado tipo não ultrapasse o número de veículos que podem atendê-lo.

Figura 5.22 – Diagrama de Classes Simplificado com a adição da classe para representação dos vértices do PDCEJT/UR.



Fonte: Do autor (2022).

Figura 5.23 – Diagrama de Classes Simplificado com a adição da classe que representa a função objetivo de maximização de pedidos do PDCEJT/UR.

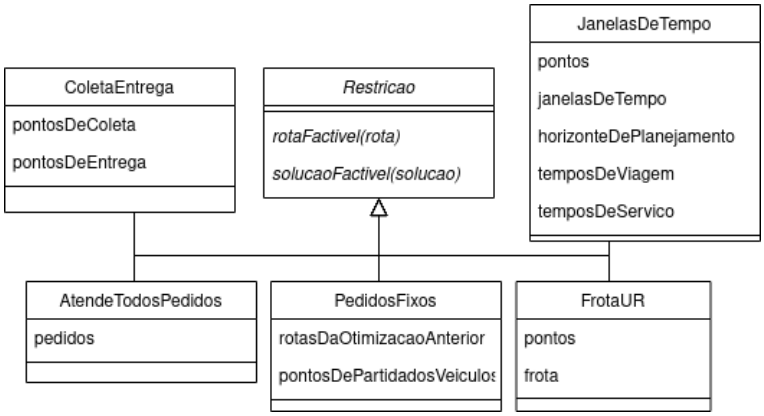


Fonte: Do autor (2022).

A Figura 5.25 apresenta a classe *Construcao* e suas descendentes, incluindo a classe *ConstrucaoPDCEJT/UR*, que realiza a implementação do algoritmo de construção adicionando a criação de rotas compatíveis com a frota.

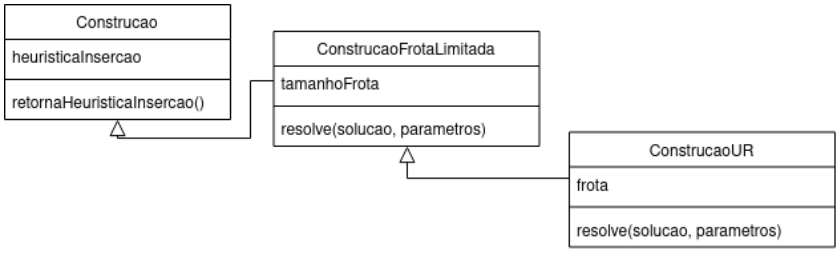
Por fim, a Figura 5.26 apresenta as descendentes adicionadas à classe *PLIM* (Seção 5.1.8). A classe *PartitionamentoUR* implementa o PLIM detalhado na Seção 5.3.1. Essa classe herda de *ParticionamentoMaxPedidos*, que implementa os métodos abstratos de sua classe pai para atender um problema que considera maximização de pedidos e frota limitada. Devido à semelhança do problema com o PDCEJT/UR, optou-se pela herança, tornando necessário somente modificar as restrições do problema, visto que a função objetivo é a mesma.

Figura 5.24 – Diagrama de Classes Simplificado com a adição da classe que representa a restrição de frota do PDCEJT/UR.



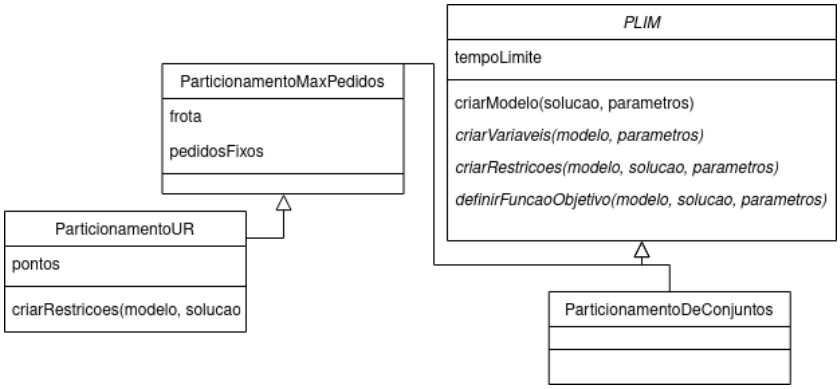
Fonte: Do autor (2022).

Figura 5.25 – Diagrama de Classes Simplificado com a adição da classe que apresenta a classe do construtor para o PDCEJT/UR.



Fonte: Do autor (2022).

Figura 5.26 – Diagrama de Classes Simplificado com a adição da classe que representa a o Programa Linear Inteiro Misto desenvolvido para o PDCEJT/UR.



Fonte: Do autor (2022).

As outras classes apresentadas durante a Seção 5.1 não sofreram alterações em sua implementação. Entre as heurísticas propostas para a solução do problema dinâmico, somente o construtor e o PLIM precisaram sofrer alterações, enquanto outros métodos foram utilizados com sua implementação para o PDCEJT.

5.4 Considerações

É perceptível na literatura a grande diversidade de problemas dinâmicos de roteamento (PILLAC et al., 2013; PSARAFTIS; WEN; KONTOVAS, 2016; Ojeda Rios et al., 2021; BERBEGLIA; CORDEAU; LAPORTE, 2010). Porém, ainda existem casos reais em que as especifidades de uma variante ainda não tenham sido estudadas, ou que haja um grande número de variantes que distanciam o problema a ser resolvido daqueles presentes na literatura.

Por outro lado, existem trabalhos que buscam criar *frameworks* para mais de uma variante de problemas de otimização. Considera-se os trabalhos de Vidal et al. (2014) e Silva et al. (2019) como os mais próximos do almejado neste trabalho. Neste estudo, aborda-se a generalização para a resolução de PDCEJT e suas variantes ao mesmo tempo em que se tenta facilitar a implementação de novos métodos de solução, restrições, funções objetivos, ou outras partes de um método de solução para problemas de roteamento.

Para dar flexibilidade na implementação de métodos de solução de problemas de roteamento, buscou-se padronizar a forma de implementá-los e utilizá-los. A Seção 5.1 apresenta como conceitos de herança e polimorfismo do Paradigma de Programação Orientado a Objetos foram empregados para abstrair as diferentes características de um algoritmo para a resolução de um problema de roteamento. A Seção 5.2, por sua vez, explica como a linguagem *Python* auxilia na padronização da implementação e uso das classes implementadas para permitir a fácil mudança de componentes heurísticos, funções objetivos, restrições, entre outros elementos de um método de solução.

A Seção 5.3, por sua vez, exemplificou as mudanças realizadas para atender o PDCEJT/UR, que adiciona pontos urbanos e rurais ao PDCEJT. É perceptível que a maioria das mudanças feitas não foram nos algoritmos utilizados para a solução do problema. O funcionamento da maior parte do código executado é em "caixa-preta", o que facilita alterações na forma como um problema é resolvido.

A ferramenta implementada sofreu impactos na sua eficiência devido à linguagem *Python*. A fim de verificar o quão bem a ferramenta proposta se comporta com a variação das componentes implementadas, apresenta-se, na sequência deste trabalho, experimentos práticos para a resolução do PDCEJT e do PDCEJT/UR.

6 EXPERIMENTOS COMPUTACIONAIS

Neste trabalho propôs-se uma adaptação da *matheuristic* de Sartori e Buriol (2020) para resolver o Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT). A fim de verificar a eficiência do método, foram executados testes com 10 das instâncias propostas por Mitrović-Minić e Laporte (2004). Cada instância contém 100 pedidos e seus pontos são distribuídos em uma área de 60×60 km^2, sendo o depósito localizado no ponto $(20, 30)$. As instâncias são armazenadas no diretório <https://drive.google.com/drive/folders/1MaluGrBbSeZxU6YqHV7_TlU1VfTCQbWj?usp=sharing>. Optou-se por experimentos com um menor número de instâncias e pequena quantidade de pontos para focar os testes nas diferentes componentes dos algoritmos, e nas duas variantes estudadas neste trabalho.

Os custos de viagem foram calculados a partir da distância euclidiana dos pontos, com precisão de uma casa decimal. Considerou-se a velocidade do veículo como $60 km/h$ para o cálculo dos tempos de viagem entre pontos. Para a simulação, o horário de serviço $\overline{h}$ foi de 10 horas, sendo dividido em dez períodos de tempo de tamanho τ iguais a 1 hora. Foram considerados 20 veículos e o tempo para a resolução ao final de cada período foi de 10 minutos. Esse cenário foi pensado considerando o problema da empresa que inspirou este trabalho, visto que era inviável que o motorista de um veículo aguardasse por muito tempo os pontos que deveria atender. Infelizmente não foi encontrado na literatura um cenário similar ao deste trabalho. A simulação foi feita em um modelo *cliente-servidor*, onde o cliente pelo gerenciamento de novos e antigos pedidos e o servidor é responsável pela resolução de cada problema parcial. O servidor funciona como uma API para a execução do algoritmo com configurações para diferentes problemas.

Diferentes cenários e componentes foram testados. O Quadro 6.1 apresenta uma codificação para os diferentes contextos e combinações de componentes aplicados a cada um dos experimentos feitos. Para os problemas dinâmicos, foi considerado a utilização ou não da componente AGES e a construção utilizando a Inserção na Primeira Posição ou a Heurística de Arrependimento. Para o contexto estático, foram feitos experimentos considerando a componente AGES e variando apenas as heurísticas de Inserção na Primeira Posição e de Arrependimento.

As heurísticas foram parametrizadas com a ferramenta *hyperopt* (BERGSTRA et al., 2015) com 7 instâncias propostas por Mitrović-Minić e Laporte (2004). As instâncias estão disponíveis no seguinte diretório: <https://drive.google.com/drive/folders/12qxkQFWUbAgUK_

Quadro 6.1 – Configurações utilizadas para os contextos dinâmico e estático.

Código	Significado
P1-C1	Dinâmico + Inserção na Primeira Posição
P1-C2	Dinâmico + AGES + Inserção na Primeira Posição
P1-C3	Dinâmico + Heurística de Arrependimento
P1-C4	Dinâmico + AGES + Heurística de Arrependimento
P2-C2	Estático + AGES + Inserção na Primeira Posição
P2-C4	Estático + AGES + Heurística de Arrependimento

Fonte: Do autor (2022).

rGw0Haim7oiefIFIyp?usp=sharing>. A parametrização foi feita no cenário estático do problema, ou seja, considerando o PCEJT. Para que houvesse maior tempo de execução das componentes da meta-heurística, a construção foi feita utilizando a heurística de Inserção na Primeira Posição.

A parametrização foi feita considerando dois casos: com a componente AGES e sem a componente AGES. Deixou-se, a critério da ferramenta de calibração, a escolha da heurística de inserção da componente LNS, permitindo a Heurística de Arrependimento (Seção 4.1.3) ou a Heurística de Arrependimento W (Seção 4.2.6). Os resultados obtidos foram transformados em um arquivos de configuração (Seção 5.3) para cada um dos quatro cenários dinâmicos presentes no Quadro 6.1, e podem ser encontrados em <https://github.com/thuzax/vrp-solver/tree/dev/server_solver/configurations>. Os cenários estáticos utilizam os arquivos de configuração para as configurações dinâmica, porém consideram que todos os pedidos são conhecidos previamente. Logo, o comportamento do algoritmo é o mesmo do que seria para o estático.

Para todos os cenários testados, cada uma das instâncias foi executada 5 vezes em uma máquina com processador *i*7 da 7^a geração e 16GB de memória *RAM*. O repositório <https://drive.google.com/drive/folders/19peJUvOsI_D450W_X6xf2Kob5r8V5dIZ?usp=sharing> contém os resultados obtidos para todos os experimentos feitos. Na sequência deste capítulo, a Seção 6.1 apresenta os experimentos realizados para o PDCEJT. A Seção 6.2, por sua vez, apresenta testes para o problema no contexto estático. Em seguida, a Seção 6.3 detalha os experimentos realizados para a variante PDCEJT/UR. Por fim, a Seção 6.4 pontua as considerações finais do capítulo.

6.1 Experimentos com o PDCEJT

O PDCEJT foi testado com diferentes configurações. Os resultados obtidos pelos experimentos P1-C1, P1-C2, P1-C3 e P1-C4 são apresentados nas Tabelas 6.1, 6.2 e 6.3. A primeira apresenta as médias considerando as 5 execuções. A segunda evidencia as melhores soluções encontradas por cada configuração, enquanto a terceira apresenta o número de rotas gerados. As colunas das tabelas indicam cada uma das configurações enquanto as linhas mostram os resultados para cada instância. Nas Tabelas 6.1 e 6.3, a linha intitulada *Média* apresenta as médias das funções objetivos de cada configuração. Analogamente, na Tabela 6.1, a linha *Desvio Padrão* que mostra o desvio padrão de cada coluna.

Tabela 6.1 – Médias (em *km*) de funções objetivos obtidas pelas quatro configurações para o problema dinâmico. As linhas *Média* e *Desvio Padrão* apresentam a média e o desvio padrão de cada configuração. Melhores valores em negrito.

Instâncias	**P1-C1**	**P1-C2**	**P1-C3**	**P1-C4**
Rnd6_10h_100_000	3468,0	2772,7	2954,0	**2697,9**
Rnd6_10h_100_001	3564,4	2819,1	3148,7	**2747,6**
Rnd6_10h_100_002	3586,0	**2901,0**	3166,9	2916,3
Rnd6_10h_100_003	3560,4	**2791,9**	3166,2	2802,4
Rnd6_10h_100_004	3501,8	2832,3	3090,4	**2806,0**
Rnd6_10h_100_005	3342,9	2854,2	3158,5	**2821,1**
Rnd6_10h_100_006	3595,4	2866,5	3294,8	**2843,0**
Rnd6_10h_100_007	3468,6	**2861,4**	3104,0	2868,8
Rnd6_10h_100_008	3358,2	**2766,8**	3126,1	2783,0
Rnd6_10h_100_009	3484,3	2713,0	3294,5	**2691,9**
Média	3493,0	2817,9	3150,4	2797,8
Desvio Padrão	62,4	41,3	59,8	49,2

Fonte: Do autor (2022).

Percebe-se que a componente AGES tem um grande impacto na função objetivo. As configurações P1-C1 e P1-C3, nas quais a componente não foi utilizada, atingiram valores de função e tamanhos de rota piores que as configurações P1-C2 e P1-C4 para todas as instâncias. A diferença é ainda mais notável quando compara-se as configurações P1-C1 e P1-C2, pois a heurística de Inserção na Primeira Posição tende a utilizar mais rotas em suas soluções. Essa diferença é justificável pois, uma maior quantidade de rotas facilita que pontos distantes entre si sejam atendidos em sequência. Com um número menor de rotas, há mais variedade nas posições de inserção, o que possibilita a realização de melhores inserções no decorrer da *matheuristic*.

Tabela 6.2 – Melhores soluções obtidas pelas quatro configurações para o problema dinâmico. Valores em *km*. Melhores dentre as solução em negrito.

Instâncias	P1-C1	P1-C2	P1-C3	P1-C4
Rnd6_10h_100_000	3432,0	2735,0	2886,6	**2623,1**
Rnd6_10h_100_001	3488,4	2681,0	3128,0	**2632,2**
Rnd6_10h_100_002	3522,5	2888,8	3130,9	**2870,2**
Rnd6_10h_100_003	3459,3	**2671,6**	3095,5	2758,6
Rnd6_10h_100_004	3427,3	**2772,9**	3060,9	2784,3
Rnd6_10h_100_005	3192,7	2833,8	3107,0	**2764,1**
Rnd6_10h_100_006	3467,7	**2768,9**	3138,1	2771,8
Rnd6_10h_100_007	3378,2	**2776,8**	2948,6	2821,5
Rnd6_10h_100_008	3286,5	2733,6	3009,3	**2716,5**
Rnd6_10h_100_009	3310,1	2681,9	3261,1	**2641,3**

Fonte: Do autor (2022).

Tabela 6.3 – Médias de número de rotas obtidos pelas quatro configurações para o problema dinâmico. A linha *Média* apresenta a média de cada configuração. Melhores valores em negrito..

Instâncias	P1-C1	P1-C2	P1-C3	P1-C4
Rnd6_10h_100_000	12,0	**8,4**	9,8	9,6
Rnd6_10h_100_001	12,6	8,6	10,6	**8,4**
Rnd6_10h_100_002	10,6	**8,4**	10,8	8,8
Rnd6_10h_100_003	11,8	**8,6**	10,4	8,8
Rnd6_10h_100_004	10,8	**8,2**	10,2	**8,2**
Rnd6_10h_100_005	10,4	8,8	9,6	**8,4**
Rnd6_10h_100_006	12,2	8,6	11,0	**8,4**
Rnd6_10h_100_007	11,4	**9,0**	10,2	9,2
Rnd6_10h_100_008	12,0	**8,6**	10,4	**8,6**
Rnd6_10h_100_009	11,6	**8,6**	12,6	9,8
Média	11,5	**8,6**	10,6	8,8

Fonte: Do autor (2022).

Outro ponto interessante é a comparação da Heurística de Arrependimento com a Inserção na Primeira Posição. As comparações entre as configurações P1-C1 e P1-C3 demonstram a diferença entre os métodos. É clara a vantagem da Heurística de Arrependimento, visto que a configuração P1-C3 obteve valores médios de função objetivo melhores que P1-C1 para todas as instâncias.

A Heurística de Arrependimento também provou ser mais eficiente quando se usa a componente AGES, porém seu impacto nesse caso é menor. A componente P1-C4 encontrou soluções melhores para 6 das instâncias, enquanto as outras 4 foram encontradas pela compo-

nente P1-C2, que utiliza da heurística de Inserção na Primeira Posição. O mesmo ocorre para as médias das execuções.

Apesar da configuração P1-C4 ser superior, percebe-se que a configuração P1-C2 também obtém bons resultados. Isso é evidenciado pela Tabela 6.4, que apresenta a porcentagem do *Desvio* entre as médias das execuções de cada instância em relação a melhor solução encontrada para o problema dinâmico, sendo sua última linha a média de suas linhas anteriores. Fica claro a melhoria trazida pela componente AGES, visto que as os valores do *desvio* ultrapassam 10% para todas as instâncias das configurações P1-C1 e P1-C3, que não utilizam da componente. Por outro lado, as configurações P1-C2 e P1-C3 tem *desvio* inferior 8% para todas as instâncias.

Tabela 6.4 – *Desvios* (em %) das médias de soluções para cada configuração testada com PDCEJT em relação a melhor solução encontrada. Melhores valores em negrito..

Instâncias	**P1-C1**	**P1-C2**	**P1-C3**	**P1-C4**
Rnd6_10h_100_000	32,2	5,7	12,6	**2,9**
Rnd6_10h_100_001	35,4	7,1	19,6	**4,4**
Rnd6_10h_100_002	24,9	**1,1**	10,3	1,6
Rnd6_10h_100_003	29,5	**4,5**	18,5	4,9
Rnd6_10h_100_004	23,6	2,1	11,5	**1,2**
Rnd6_10h_100_005	20,9	3,3	14,3	**2,1**
Rnd6_10h_100_006	25,2	3,5	19,0	**2,7**
Rnd6_10h_100_007	21,7	**3,0**	11,8	3,3
Rnd6_10h_100_008	23,6	**1,9**	15,1	2,4
Rnd6_10h_100_009	31,9	2,7	24,7	**1,9**
Média	26,9	3,5	15,7	**2,7**

Fonte: Do autor (2022).

A média dos *desvios*, por sua vez, indicam a proximidade dos resultados obtidos das combinações P1-C2 e P1-C3, que utilizam, respectivamente, da Inserção na Primeira Posição e da Heurística de Arrependimento. Pode-se dizer que a *matheuristic* de Sartori e Buriol (2020) consegue resultados relativamente próximos mesmo ao variar o método de construção devido à componente AGES, pois a redução do número de rotas permite que as outras heurísticas tenham um comportamento melhor. Mais testes com as diferentes heurísticas de construção e a componente AGES são realizados com o PCEJT no contexto estático, e são discutidos com mais detalhes na sequência por meio da Seção 6.2.

6.2 Experimentos com o PCEJT

Realiza-se experimentos considerando o PCEJT para as mesmas instâncias, partindo do pressuposto que todos os pedidos são conhecidos no início do horário de serviço. Foram utilizadas duas diferentes configurações: com a Inserção na Primeira Posição (P2-C2) e com Heurística de Arrependimento (P2-C4). Em ambos os casos a componente AGES foi utilizada. O tempo de execução das configurações estáticas foi de 50 minutos.

As Tabelas 6.5 e 6.6 apresentam os resultados obtidos para as configurações P2-C2 e P2-C4. Suas colunas e linhas são análogas às Tabelas 6.1 e 6.2.

Tabela 6.5 – Médias (em *km*) de funções objetivos obtidas pelas quatro configurações para o problema estático. As linhas *Média* e *Desvio Padrão* apresentam a média e o desvio padrão de cada configuração. Melhores valores em negrito..

Instâncias	**P2-C2**	**P2-C4**
Rnd6_10h_100_000	1926,3	**1846,4**
Rnd6_10h_100_001	1959,7	**1921,1**
Rnd6_10h_100_002	2044,8	**1993,8**
Rnd6_10h_100_003	2051,6	**2001,7**
Rnd6_10h_100_004	**1860,7**	1939,8
Rnd6_10h_100_005	**1932,3**	1955,1
Rnd6_10h_100_006	**2042,1**	2100,9
Rnd6_10h_100_007	2062,1	**2046,5**
Rnd6_10h_100_008	2089,2	**2082,3**
Rnd6_10h_100_009	**1856,1**	1859,2
Média	1982,5	1974,7

Fonte: Do autor (2022).

É perceptível que a configuração P2-C4 encontrou um maior número de melhores soluções e de valores médios para a maioria dos casos. Contudo, fica claro que a diferença entre os valores de função objetivo entre ambas componentes estáticas é pequeno. Isso é evidenciado pela Figura 6.1, que por sua vez ilustra as os valores das médias de cada uma das componentes testadas. Percebe-se que os valores médios para ambas as configurações estáticas são muito próximos, tendo uma diferença de menos de 1%.

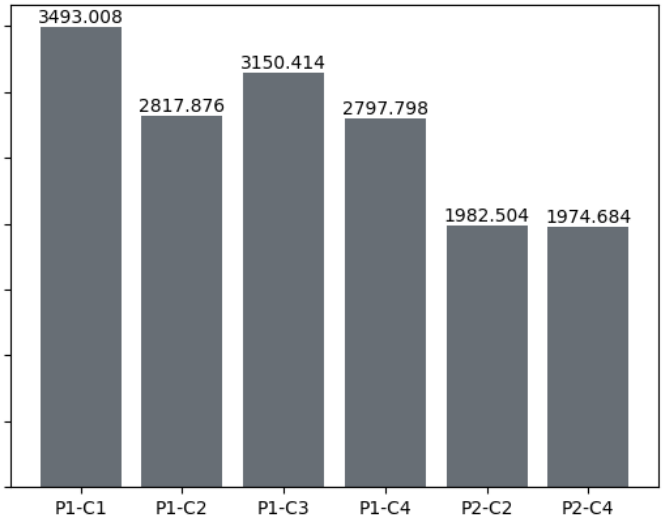
A Tabela 6.7 reafirma esse fato, apresentando as porcentagens do *desvio* das configurações estáticas e dinâmicas em relação à melhor soluções estática encontrada. A proximidade da eficiência das configurações P2-C2 e P2-C4 é ainda mais evidente, visto que ambas apresentam

Tabela 6.6 – Melhores soluções obtidas pelas quatro configurações para o problema estático. Valores em *km*. Melhores dentre as soluções em negrito.

Instâncias	P2-C2	P2-C4
Rnd6_10h_100_000	1869,4	**1798,4**
Rnd6_10h_100_001	1900,0	**1889,9**
Rnd6_10h_100_002	2017,4	**1945,9**
Rnd6_10h_100_003	1991,3	**1917,8**
Rnd6_10h_100_004	**1814,4**	1879,8
Rnd6_10h_100_005	**1892,2**	1921,5
Rnd6_10h_100_006	1988,2	**1971,5**
Rnd6_10h_100_007	2009,2	**1965,5**
Rnd6_10h_100_008	1996,9	**1978,8**
Rnd6_10h_100_009	1798,7	**1795,5**

Fonte: Do autor (2022).

Figura 6.1 – Médias de valores para as funções objetivos de cada combinação testada para o cenário estático e dinâmico.



Fonte: Do autor (2022).

valores de *desvio* com diferença de menos de 5% para todas as instâncias e seus *desvios* médio tem diferença menor que $0,5\%$.

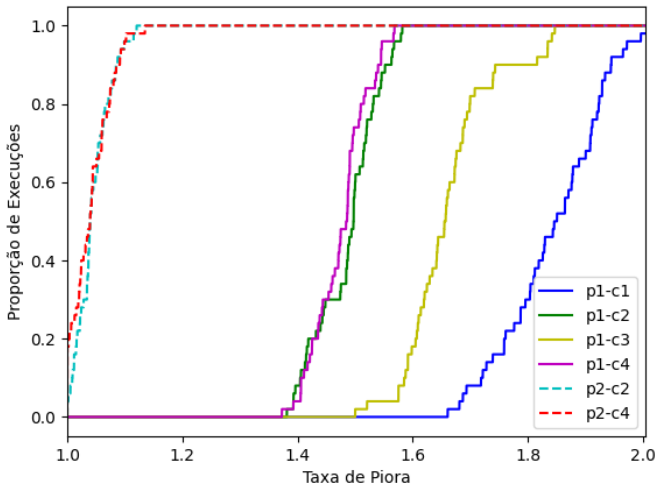
Analisando também as configurações dinâmicas, considerando a Figura 6.1 e a Tabela 6.7, por outro lado, vê-se a grande vantagem dos métodos estáticos. Na tabela tem-se valores altos, todos acima de 40%. A disparidade entre os problemas estáticos e dinâmicos fica ainda mais visível ao se analisar a Figura 6.2, que apresenta um gráfico *Performance x Profile* para os diferentes métodos de solução e problemas. Como pode ser visto, os métodos estáticos têm uma

curva de crescimento mais rápida. Isso significa que os desvios de suas soluções em relação à melhor é menor.

A diferença entre problemas estáticos e dinâmicos, contudo, é esperada, visto que o conhecimento de todos os pedidos antes do início do horário de atendimento acarreta em uma grande vantagem. Ao decorrer dos períodos de tempo o número de pedidos cuja rota é imutável aumenta, e isso faz com que as possibilidades de melhoria da solução diminuam.

Por outro lado, avaliando somente os métodos dinâmicos, fica claro a superioridade daqueles que se utilizam da componente AGES. Apesar de estarem distantes dos resultados para métodos estáticos, é visível que a taxa de crescimento de suas curvas é maior que ambos os métodos que não utilizam a componente. Por fim, percebe-se que a componente AGES tem um impacto maior quando se usa a Inserção na Primeira Posição.

Figura 6.2 – *Performance x Profile* entre as componentes estáticas (linhas tracejadas) e dinâmicas (linhas contínuas). Nota-se que quando a taxa de piora é 1, tem-se as execuções que obtiveram melhores soluções.



Fonte: Do autor (2022).

Ao analisar os resultados obtidos pelas P1-C2 e P1-C4 na Seção 6.1, estima-se que existe a possibilidade da *matheuristic* de Sartori e Buriol (2020) ser independente de seu método de construção. Essa hipótese é reforçada pelos resultados apresentados para o contexto estático do problema estudado. A fim de explorar mais essa característica, detalha-se na Tabela 6.8 a média de tempos de execução para as configurações P2-C2 e P2-C4, que utilizam, respectivamente, das heurísticas de Inserção na Primeira Posição e da Heurística de Arrependimento.

Tabela 6.7 – *Desvios* (em %) das médias de soluções para cada configuração testada em relação a melhor solução encontrada para o PCEJT no contexto estático. Melhores valores em negrito..

Instâncias	P1-C1	P1-C2	P1-C3	P1-C4	P2-C2	P2-C4
Rnd6_10h_100_000	92,8	54,2	64,3	50,0	7,1	**2,7**
Rnd6_10h_100_001	88,6	49,2	66,6	45,4	3,7	**1,7**
Rnd6_10h_100_002	84,3	49,1	62,7	49,9	5,1	**2,5**
Rnd6_10h_100_003	85,6	45,6	65,1	46,1	7,0	**4,4**
Rnd6_10h_100_004	93,0	56,1	70,3	54,7	**2,6**	6,9
Rnd6_10h_100_005	76,7	50,8	66,9	49,1	**2,1**	3,3
Rnd6_10h_100_006	82,4	45,4	67,1	44,2	**3,6**	6,6
Rnd6_10h_100_007	76,5	45,6	57,9	46,0	4,9	**4,1**
Rnd6_10h_100_008	69,7	39,8	58,0	40,6	5,6	**5,2**
Rnd6_10h_100_009	94,1	51,1	83,5	49,9	**3,4**	3,5
Média	84,4	48,7	66,2	47,6	4,5	**4,1**

Fonte: Do autor (2022).

O tempo gasto pela Heurística de Arrependimento é elevado, chegando a demorar, em média, mais de 5 minutos para 6 das instâncias. Isso provavelmente se deve à linguagem *Python* e às estratégias de implementação apresentadas no Capítulo 5. É interessante notar, porém, que esse fenômeno não ocorre no contexto dinâmico. A adaptação da heurística de construção apresentada na Seção 6 reduz o número de pedidos a serem inseridos, visto que a solução obtida em um período de tempo anterior é reconstruída.

Para uma análise mais completa do comportamento das heurísticas em relação ao tempo, apresenta-se o Apêndice A. As Figuras 1-10 são gráficos TTT para as execuções de cada uma das instâncias, apresentando a melhoria da função objetivo no decorrer do tempo de execução. Os gráficos reforçam as conclusões anteriores: a Inserção na Primeira Posição gera soluções iniciais de forma mais rápida. Como esperado, as soluções apresentam um valor de função objetivo maior, contudo, para grande parte dos casos, o valor de função é reduzido rapidamente para um valor similar ao obtido pela Heurística de Arrependimento. Essa redução rápida provavelmente se deve à componente LNS, que se utiliza da Heurística de arrependimento para a reinserção. Por fim, percebe-se que curva de melhoria da Inserção na Primeira Posição se torna similar a da Heurística de Arrependimento após a aproximação dos valores da função.

Mesmo com o custo de tempo elevado, a Heurística de Arrependimento obteve resultados melhores que a de Inserção na Primeira Posição. Isso traz o seguinte questionamento: a

matheuristic de Sartori e Buriol (2020) é independente do método de construção ou o alto custo de tempo da Heurística de Arrependimento enviesou os resultados?

A partir dos resultados deste trabalho, acredita-se que, em uma implementação mais eficiente, a Heurística de Arrependimento terá um valor levemente melhor que a Inserção na Primeira Posição em instâncias muito pequenas. Por outro lado, para instâncias muito grandes, os resultados das heurísticas tenderão a ser semelhantes, visto que o valor da Inserção na Primeira Posição decresce rapidamente no início da execução para um valor próximo ao da Heurística de Arrependimento, como pode ser visto Figuras 1-10.

Tabela 6.8 – Médias de tempo (em segundos) gasto na etapa de construção para o PCEJT utilizando a Inserção na Primeira Posição (configuração P2-C2) e a Heurística de Arrependimento (configuração P2-C4). Melhores valores em negrito..

Instâncias	P2-C2	P2-C4
Rnd6_10h_100_000	**19,5**	315,1
Rnd6_10h_100_001	**18,5**	210,2
Rnd6_10h_100_002	**16,6**	569,7
Rnd6_10h_100_003	**20,5**	123,8
Rnd6_10h_100_004	**20,1**	210,7
Rnd6_10h_100_005	**20,0**	523,6
Rnd6_10h_100_006	**17,0**	391,0
Rnd6_10h_100_007	**23,5**	238,6
Rnd6_10h_100_008	**21,7**	364,7
Rnd6_10h_100_009	**15,2**	305,4
Média	**19,3**	325,3

Fonte: Do autor (2022).

A possibilidade da ocorrência do primeiro caso se deve aos resultados obtidos para o PDCEJT, que, a cada período de tempo, considera a construção com uma parcela do total de pontos. Acredita-se que o segundo caso é devido ao tempo necessário para a execução da Heurística de Arrependimento. Em uma implementação mais eficiente, os tempos obtidos na Tabela 6.8 só seriam obtidos em instâncias muito grandes. Para a comprovação de ambas as hipóteses, contudo, seriam necessários mais testes.

6.3 Experimentos para o PDCEJT/UR

A fim de aprofundar os estudos no PDCEJT com pontos Urbanos e Rurais, realiza-se experimentos práticos com instâncias adaptadas de Mitrović-Minić e Laporte (2004). As 10

instâncias utilizadas nos testes anteriores foram adaptadas para que parte de seus pontos fossem considerados rurais. As instâncias para o PDCEJT/UR podem ser encontradas em <https://drive.google.com/drive/folders/1yRPmj_u4DdrogCJtJJ6NoK151WNlTryU?usp=sharing>.

A classificação de pontos urbanos e rurais foi feito através do seguinte algoritmo:

a) encontra-se um ponto central da instância através do algoritmo *K-Means*;
b) calcula-se a distância de todo ponto para o ponto central;
c) todo ponto cuja distância do centro é maior que um determinado valor ω é considerado rural. Caso contrário, o ponto é urbano.

O valor de ω é calculado em função da metade da maior das distâncias entre dois pontos da instância, denotada por c_{met}. Para as instâncias geradas, usou-se $\omega = 0.8c_{met}$. O tamanho da frota rural é dado por $\lceil \phi m \rceil$, sendo m o número de veículos da instância original e ϕ a proporção de pontos rurais na instância, ou seja, o número de pedidos rurais dividido pelo número total de pedidos. A frota urbana, por sua vez, tem tamanho $\lceil (1-\phi)m \rceil$. Após os experimentos realizados nas Seções 6.1 e 6.2, pode-se concluir que a utilização da AGES e da Heurística de Arrependimento é benéfica para o PDCEJT. Portanto, realiza-se os experimentos utilizando a mesma configuração P1-C4. O arquivo de configurações utilizado para o problema foi adaptado do arquivo gerado pela parametrização. Ele pode ser encontrado em <https://github.com/thuzax/vrp-solver/tree/dev/server_solver/configurations>

A Tabela 6.9 apresenta os resultados obtidos para as execuções. A coluna *Melhor Encontrado* apresenta dados relacionados a da melhor solução encontrada, a coluna *Média* detalha as médias das execuções e a coluna *Desvio Padrão* o desvio padrão. As três exibem valores relacionado à maximização de pedidos em *Ped.* e o custo total das rotas em *Dist.* A última coluna *Média Frota*, apresenta o número de rotas médio das soluções.

Observa-se que para todas as instâncias foram encontradas melhores soluções que atendem pelo menos 90% dos pedidos, enquanto a média do método foi superior a 89% em todos os casos. Nota-se também que os custos de distância são próximos, quando não menores, que as melhores soluções encontradas para o PDCEJT. Parte da redução se deve ao não atendimento de um subconjunto de pedidos. Contudo, é plausível dizer que as adaptações feitas a partir da *matheuristic* de Sartori e Buriol (2020) são igualmente eficientes para o PDCEJT/UR quanto para o PDCEJT.

Tabela 6.9 – Resultados obtidos para o PDCEJT/UR com as instâncias adaptadas. As colunas *Ped.* são dados referentes aos pedidos, enquanto as colunas *Dist.* são dados referentes ao custo das rotas. Os custos das rotas são dados em *km*.

Instâncias	Melhor Encontrado		Média		Média Frota
	Ped	Dist	Ped	Dist	
Rnd6_10h_100_000	95,0	2535,7	95,0	2535,7	9,0
Rnd6_10h_100_001	91,0	2380,8	90,2	2352,6	7,0
Rnd6_10h_100_002	96,0	2687,9	96,0	2694,4	8,0
Rnd6_10h_100_003	90,0	2483,6	89,0	2510,8	7,8
Rnd6_10h_100_004	91,0	2619,4	89,8	2540,2	7,8
Rnd6_10h_100_005	95,0	2740,0	94,8	2779,9	8,2
Rnd6_10h_100_006	97,0	2971,7	94,6	2886,9	7,8
Rnd6_10h_100_007	94,0	2678,9	92,8	2612,3	7,4
Rnd6_10h_100_008	95,0	2405,2	94,8	2464,5	7,2
Rnd6_10h_100_009	95,0	2509,0	93,6	2488,4	9,0

Fonte: Do autor (2022).

6.4 Considerações

Neste capítulo foram apresentados os experimentos práticos para os métodos apresentados no Capítulo 4 e implementadas utilizando as técnicas presentes no Capítulo 5. Os principal objetivo dos testes era a verificação do comportamento das implementadas em diferentes problemas.

Foram realizados testes para o PDCEJT utilizando 4 diferentes configurações, combinando a aplicação ou não da componente AGES com a Heurística de Arrependimento ou com a Inserção na Primeira Posição. Concluiu-se que o uso da componente AGES causa um grande impacto positivo no resultado. A heurística de Inserção na Primeira Posição, por sua vez, se mostrou inferior à Heurística de Arrependimento. Porém, quando a componente AGES foi utilizada, seus resultados foram mais próximos.

A fim de comparar os resultados obtidos no PDCEJT e investigar mais os impactos da variação da construção, o método foi testado para o PCEJT. Duas configurações foram testadas, sendo que ambas utilizam a componente AGES, mas variam entre a Inserção na Primeira Posição e Heurística de Arrependimento. Os experimentos mostraram que os dois métodos de inserção geram resultados semelhantes, porém questionou-se o impacto do tempo gasto pela Heurística de Arrependimento. Com os resultados obtidos, foi possível afirmar que a *matheuristic* de Sartori e Buriol (2020) não sofre grandes impactos utilizando as diferentes inserções em

instâncias pequenas. Estima-se que o mesmo ocorra para instâncias muito grandes. Contudo, para provar ambas as conclusões, seriam necessários mais testes.

Por fim, o algoritmo foi testado para o PDCEJT/UR, cujos métodos de solução foram implementados para exemplificar as generalizações feitas no Capítulo 5. Alcançou-se taxas altas para o número de pedidos atendidos, e os resultados obtidos para os custos das rotas indicam que as adaptações para o PDCEJT e para o PDCEJT/UR são equivalentes em termos de eficiência.

Na sequência deste trabalho, no Capítulo 7, apresenta-se as conclusões obtidas a partir do estudo detalhados anteriormente e indica-se possibilidades de continuação para este trabalho.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O Problema Dinâmico de Coleta e Entrega com Janelas de Tempo (PDCEJT) consiste no atendimento de pedidos com determinada carga deve ser coletada em um ponto e entregue em outro, sendo que o atendimento ao ponto deve ser realizado dentro de um determinado período. Informações relacionadas ao pedido são obtidas de forma dinâmica no decorrer do período de atendimento dos pontos. O objetivo é atender os pedidos com o menor custo possível.

Inspirado em um caso real, este trabalho focou em resolver o PDCEJT ao mesmo tempo em que cria uma ferramenta para facilitar a adição, remoção ou alteração de restrições, heurísticas, funções objetivos, entre outros aspectos de problemas de otimização. O problema foi resolvido ao se adaptar a *matheuristic* proposta por Sartori e Buriol (2020) e a implementação foi feita utilizando conceitos de orientação a objetos e recursos da linguagem *Python*, a fim de padronizar a adição de código-fonte. Uma solução para o PDCEJT/UR foi implementada a fim de exemplificar a utilização da técnica de generalização proposta.

Para verificar o comportamento das diferentes componentes adaptadas para a solução do PDCEJT, experimentos foram realizados com um conjunto de 10 instâncias. Testou-se a utilização ou não da componente AGES junto à Heurística de Arrependimento ou à Inserção na Primeira Posição. Como resultados, percebeu-se que a componente AGES tem um grande impacto no resultado final do método. Além disso, ao utilizar a componente, os resultados obtidos pela Heurística de Arrependimento e pela Inserção na Primeira Posição foram mais próximos.

Também foram realizados experimentos para o contexto estático do problema com as duas combinações que utilizam a componente AGES. Como era de se esperar, os resultados para o problema estático foram melhores. Um detalhe importante, contudo, é que foi possível uma nova comparação entre a Heurística de Arrependimento e pela Inserção na Primeira Posição. A Heurística de Arrependimento se mostrou superior ao encontrar um maior número de melhores soluções, porém a diferença dos resultados médios foi próximo.

Ao analisar os tempos de execução das heurísticas, contudo, percebeu-se que a Heurística de Arrependimento sofreu grande impacto devido à implementação em *Python*. Isso traz o questionamento: a *matheuristic* de Sartori e Buriol (2020) é independente da sua heurística de construção ou houve um enviesamento dos resultados devido ao alto custo de tempo para a execução da Heurística de Arrependimento?

Os resultados obtidos indicam que em uma implementação mais eficiente, a Heurística de Arrependimento alcançará valores levemente melhor que a Inserção na Primeira Posição em instâncias muito pequenas. Por outro lado, em instâncias muito grandes, seus resultados tendem a ser semelhantes. Contudo, seriam necessários mais testes para a comprovação de ambas as hipóteses.

Para finalizar os experimentos, o método proposto para o PDCEJT/UR foi testada. As instâncias utilizadas nos experimentos com o PDCEJT e com o PCEJT foram adaptadas para a divisão de pontos urbanos e rurais. Os testes realizados obtiveram uma alta taxa de atendimento de pedidos e os custos das rotas apontam que o método é tão eficiente para o PDCEJT/UR quanto para o PDCEJT.

Dentre as possibilidades de continuação deste trabalho, tem-se a realização de mais experimentos com a Heurística de Arrependimento e com a Inserção na Primeira Posição para investigação dos impactos da construção na *matheuristic* de Sartori e Buriol (2020) para problemas dinâmicos de roteamento.

As futuras pesquisas também podem ser voltadas a generalização da implementação. Atualmente, a criação de classes e métodos de configuração são padronizados e isso abre oportunidade para a utilização de algoritmos para geração de código. Por exemplo, seria possível desenvolver uma interface gráfica para a criação de um novo método de solução ou para a criação de um novo arquivo de configuração. Outra opção seria fazer algo similar a (ERDOĞAN, 2017), e utilizar planilhas como arquivos de configuração. A utilização do padrão de projeto Fábrica Abstrata para facilitar a implementação de novos métodos também pode ser estudada. Decoradores (<https://python101.pythonlibrary.org/chapter25_decorators.html>) também podem contribuir para a generalização e reutilização do código. Outra possibilidade é a adição de mais métodos de solução tanto para o PDCEJT e suas variantes como para outras variantes dinâmicas.

Por fim, tem-se a melhoria na eficiência das heurísticas implementadas, visto que as tecnologias e estratégias apresentadas no Capítulo 5 causaram impactos na eficiência do método. Uma primeira forma de melhoria seria a implementação das partes críticas dos métodos de solução utilizando pacotes como *ctypes* (<https://docs.python.org/3/library/ctypes.html>), que permite a integração de código em linguagem *C*.

REFERÊNCIAS

ABDALLAH, A. M. F.; ESSAM, D. L.; SARKER, R. A. On solving periodic re-optimization dynamic vehicle routing problems. **Applied Soft Computing Journal**, Elsevier B.V., v. 55, p. 1–12, 2017. ISSN 15684946. Disponível em: <http://dx.doi.org/10.1016/j.asoc.2017.01.047>.

ABDIRAD, M.; KRISHNAN, K.; GUPTA, D. A two-stage metaheuristic algorithm for the dynamic vehicle routing problem in industry 4.0 approach. **Journal of Management Analytics**, Informa UK Limited, v. 8, n. 1, p. 69–83, Oct 2020. ISSN 2327-0039. Disponível em: <http://dx.doi.org/10.1080/23270012.2020.1811166>.

ATTANASIO, A. et al. Real-time fleet management at eCourier Ltd. **Operations Research/ Computer Science Interfaces Series**, v. 38, 2007. ISSN 1387666X.

AYOB, M.; KENDALL, G. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In: . [S.l.: s.n.], 2003. p. 132–141.

BENDER, M.; KALCSICS, J.; MEYER, A. Districting for parcel delivery services–a two-stage solution approach and a real-world case study. **Omega**, Elsevier, p. 102283, 2020.

BERBEGLIA, G.; CORDEAU, J. F.; LAPORTE, G. Dynamic pickup and delivery problems. **European Journal of Operational Research**, Elsevier B.V., v. 202, n. 1, p. 8–15, 2010. ISSN 03772217. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2009.04.024>.

BERGSTRA, J. et al. Hyperopt: a python library for model selection and hyperparameter optimization. **Computational Science & Discovery**, IOP Publishing, v. 8, n. 1, p. 014008, jul 2015. Disponível em: <https://doi.org/10.1088/1749-4699/8/1/014008>.

BURKE, E. K.; BYKOV, Y. The late acceptance hill-climbing heuristic. **European Journal of Operational Research**, v. 258, n. 1, p. 70–78, 2017. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221716305495>.

CHEN, S.; CHEN, R.; GAO, J. A monarch butterfly optimization for the dynamic vehicle routing problem. **Algorithms**, v. 10, n. 3, 2017. ISSN 19994893.

CÔTÉ, J.-f. et al. Dynamic Optimization Algorithms for Same-Day Dynamic Optimization Algorithms for Same-Day Delivery Problems. **CIRRELT**, CIRRELT-2021-17, 2021.

CURTOIS, T. et al. Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. **EURO Journal on Transportation and Logistics**, Springer, v. 7, n. 2, p. 151–192, 2018.

DANTZIG, G. B.; RAMSER, J. H. The Truck Dispatching Problem. **Management Science**, v. 6, n. 1, p. 80–91, 1959. ISSN 0025-1909.

DUMAS, Y.; DESROSIERS, J.; SOUMIS, F. The pickup and delivery problem with time windows. **European Journal of Operational Research**, v. 54, n. 1, p. 7–22, 1991. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/037722179190319Q>.

ELSHAER, R.; AWAD, H. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. **Computers and Industrial Engineering**, Elsevier, v. 140, n. November 2018, p. 106242, 2020. ISSN 03608352. Disponível em: <https://doi.org/10.1016/j.cie.2019.106242>.

ERDOĞAN, G. An open source spreadsheet solver for vehicle routing problems. **Computers & Operations Research**, v. 84, p. 62–72, 2017. ISSN 0305-0548. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0305054817300552>.

FERRUCCI, F.; BOCK, S. Real-time control of express pickup and delivery processes in a dynamic environment. **Transportation Research Part B: Methodological**, Elsevier Ltd, v. 63, p. 1–14, 2014. ISSN 01912615. Disponível em: <http://dx.doi.org/10.1016/j.trb.2014.02.001>.

FONSECA-GALINDO, J. C. et al. A Multi-Agent System for Solving the Dynamic Capacitated Vehicle Routing Problem with Stochastic Customers using Trajectory Data Mining. 2020. Disponível em: <http://arxiv.org/abs/2009.12691>.

GENDREAU, M. et al. Parallel tabu search for real-time vehicle routing and dispatching. **Transportation Science**, v. 33, n. 4, p. 381–390, 1999. ISSN 00411655.

GENDREAU, M. et al. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. **Transportation Research Part C: Emerging Technologies**, v. 14, n. 3, p. 157–174, 2006. ISSN 0968090X.

GHIANI, G. et al. Anticipatory algorithms for same-day courier dispatching. **Transportation Research Part E: Logistics and Transportation Review**, Elsevier Ltd, v. 45, n. 1, p. 96–106, 2009. ISSN 13665545. Disponível em: <http://dx.doi.org/10.1016/j.tre.2008.08.003>.

Google Developers. **Sobre OR-Tools**. [S.l.]: Google OR-Tools, 2021. Disponível em: <https://developers.google.com/optimization/introduction/overview>. Acessado em 22/06/2022.

HANSEN, P. et al. Variable neighborhood search. In: ____. **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p. 61–86. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_3>.

HOLBORN, P. L. Heuristics for Dynamic Vehicle Routing Problems with Pickups and Deliveries and Time Windows A thesis. n. May, p. 252, 2013. Disponível em: <http://orca.cf.ac.uk/47742/8/2013holbornplphdV.3.pdf>.

KARAMI, F.; VANCROONENBURG, W.; Vanden Berghe, G. A periodic optimization approach to dynamic pickup and delivery problems with time windows. **Journal of Scheduling**, Springer US, v. 23, n. 6, p. 711–731, 2020. ISSN 10991425. Disponível em: <https://doi.org/10.1007/s10951-020-00650-x>.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE. **Proceedings of ICNN'95-International Conference on Neural Networks**. [S.l.], 1995. v. 4, p. 1942–1948.

KILBY, P.; PROSSER, P.; SHAW, P. Dynamic {VRP}s: A Study of Scenarios. n. APES-06-1998, p. 1–11, 1998.

LAPORTE, G. What you should know about the vehicle routing problem. **Naval Research Logistics (NRL)**, Wiley Online Library, v. 54, n. 8, p. 811–819, 2007.

LAPORTE, G. Fifty Years of Vehicle Routing. **Transportation Science**, v. 43, n. 4, p. 408–416, 2009.

MITROVIĆ-MINIĆ, S.; LAPORTE, G. Waiting strategies for the dynamic pickup and delivery problem with time windows. **Transportation Research Part B: Methodological**, v. 38, n. 7, p. 635–655, 2004. ISSN 01912615.

MONTEMANNI, R. et al. Ant colony system for a dynamic vehicle routing problem. **Journal of Combinatorial Optimization**, Springer, v. 10, n. 4, p. 327–343, 2005.

NAGATA, Y.; KOBAYASHI, S. Guided ejection search for the pickup and delivery problem with time windows. In: SPRINGER. **European Conference on Evolutionary Computation in Combinatorial Optimization**. [S.l.], 2010. p. 202–213.

NINIKAS, G.; MINIS, I. Reoptimization strategies for a dynamic vehicle routing problem with mixed backhauls. **Networks**, v. 64, n. 3, p. 214–231, 2014. ISSN 10970037.

Ojeda Rios, B. H. et al. Recent dynamic vehicle routing problems: A survey. **Computers & Industrial Engineering**, v. 160, p. 107604, 2021. ISSN 0360-8352. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360835221005088>.

OKULEWICZ, M.; MAŃDZIUK, J. The impact of particular components of the PSO-based algorithm solving the Dynamic Vehicle Routing Problem. **Applied Soft Computing Journal**, v. 58, p. 586–604, 2017. ISSN 15684946.

OLIVEIRA, B. B. et al. A C++ application programming interface for co-evolutionary biased random-key genetic algorithms for solution and scenario generation. **Optimization Methods and Software**, p. 1–21, 2021. ISSN 10294937.

PANKRATZ, G. A grouping genetic algorithm for the pickup and delivery problem with time windows. **OR Spectrum**, v. 27, n. 1, p. 21–41, 2005. ISSN 01716468.

PANKRATZ, G. Dynamic vehicle routing by means of a genetic algorithm. **International Journal of Physical Distribution and Logistics Management**, v. 35, n. 5, p. 362–383, 2005. ISSN 09600035.

PERRON, L. Operations research and constraint programming at google. In: LEE, J. (Ed.). **Principles and Practice of Constraint Programming – CP 2011**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 2–2. ISBN 978-3-642-23786-7.

PILLAC, V. et al. A review of dynamic vehicle routing problems. **European Journal of Operational Research**, Elsevier B.V., v. 225, n. 1, p. 1–11, 2013. ISSN 03772217. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2012.08.015>.

PSARAFTIS, H. N. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. **Transportation Science**, INFORMS, v. 14, n. 2, p. 130–154, 1980.

PSARAFTIS, H. N.; WEN, M.; KONTOVAS, C. A. Dynamic vehicle routing problems: Three decades and counting. **Networks**, v. 67, n. 1, p. 3–31, 2016. ISSN 10970037.

PUREZA, V.; LAPORTE, G. Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. **INFOR: Information Systems and Operational Research**, Taylor & Francis, v. 46, n. 3, p. 165–175, 2008.

REEVES, C. R. Genetic algorithms. In: ____. **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p. 109–139. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_5>.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In: ____. **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010. p. 283–319. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_10>.

RITZINGER, U.; PUCHINGER, J.; HARTL, R. F. A survey on dynamic and stochastic vehicle routing problems. **International Journal of Production Research**, Taylor & Francis, v. 54, n. 1, p. 215–231, 2016. Disponível em: <https://doi.org/10.1080/00207543.2015.1043403>.

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. **Transportation Science**, INFORMS Inst.for Operations Res.and the Management Sciences, v. 40, n. 4, p. 455–472, 2006. ISSN 15265447.

ROSSUM, G. V.; DRAKE, F. L. **Python 3 Reference Manual**. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

SABAR, N. R. et al. Population-based iterated local search approach for dynamic vehicle routing problems. **IEEE Transactions on Automation Science and Engineering**, p. 1–11, 2021.

SARTORI, C. S.; BURIOL, L. S. A study on the pickup and delivery problem with time windows: Matheuristics and new instances. **Computers & Operations Research**, Elsevier, p. 105065, 2020.

SILVA, M. A. L. et al. A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. **Expert Systems with Applications**, v. 131, p. 148–171, 2019. ISSN 09574174.

SOLOMON, M. M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. **Operations Research**, v. 35, n. 2, p. 254–265, 1987.

TOTH, P. et al. **Vehicle Routing: Problems, Methods, and Applications, Second Edition**. USA: Society for Industrial and Applied Mathematics, 2014. ISBN 1611973589.

ULMER, M. W. et al. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. **Transportation Science**, v. 55, n. 1, p. 75–100, 2021. ISSN 15265447.

VIDAL, T. et al. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. **Computers and Operations Research**, v. 40, n. 1, p. 475–489, jan 2013. ISSN 03050548.

VIDAL, T. et al. A unified solution framework for multi-attribute vehicle routing problems. **European Journal of Operational Research**, Elsevier B.V., v. 234, n. 3, p. 658–673, 2014. ISSN 03772217. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2013.09.045>.

VIDAL, T.; LAPORTE, G.; MATL, P. A concise guide to existing and emerging vehicle routing problem variants. **European Journal of Operational Research**, Elsevier B.V., p. 1–16, 2019. ISSN 03772217.

WANG, G.-G.; DEB, S.; CUI, Z. Monarch butterfly optimization. **Neural computing and applications**, Springer, v. 31, n. 7, p. 1995–2014, 2019.

APÊNDICE A – GRÁFICOS TTT - HEURÍSTICA DE ARREPENDIMENTO X PRIMEIRA INSERÇÃO FACTÍVEL

Figura 1 – Gráfico TTT para instância Rnd6_10h_100_000. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

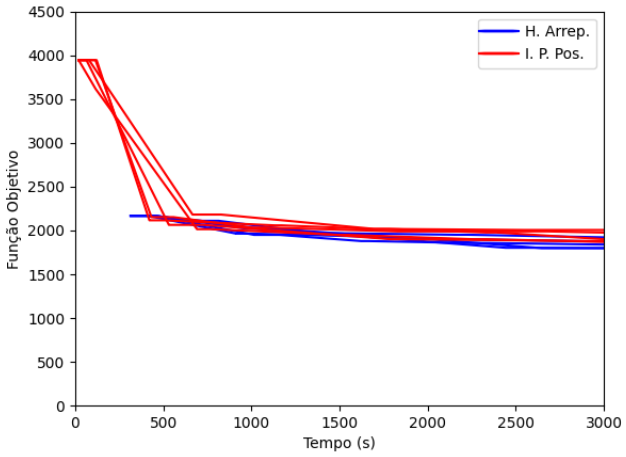


Figura 2 – Gráfico TTT para instância Rnd6_10h_100_001. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

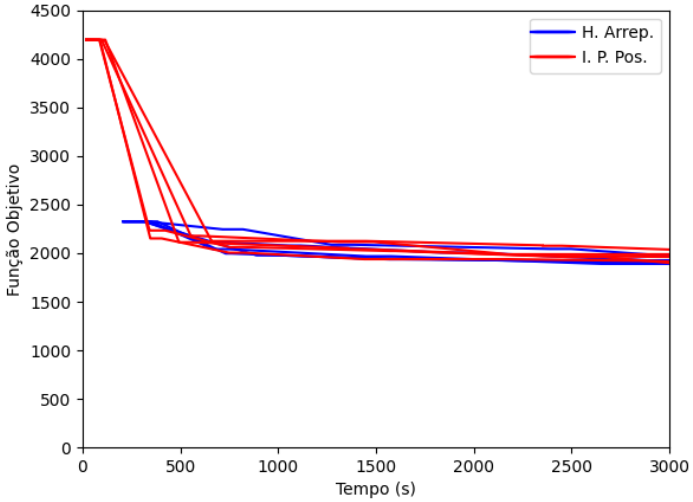


Figura 3 – Gráfico TTT para instância Rnd6_10h_100_002. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

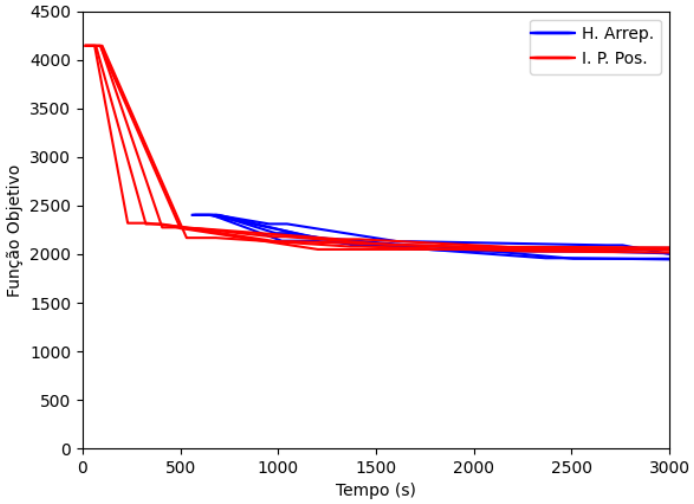


Figura 4 – Gráfico TTT para instância Rnd6_10h_100_003. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

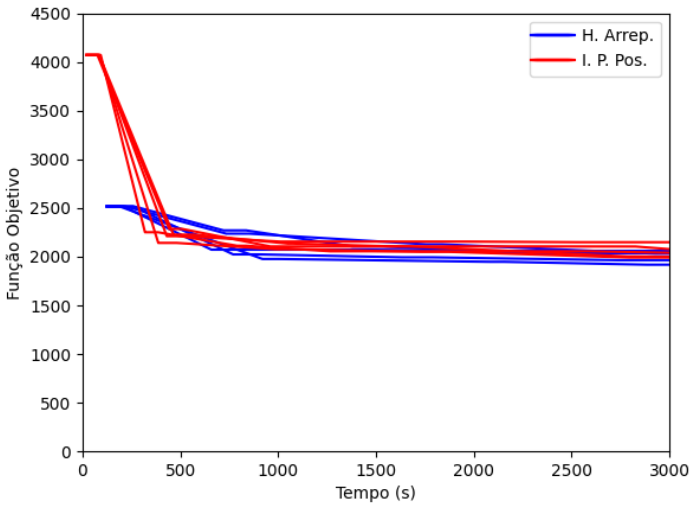


Figura 5 – Gráfico TTT para instância Rnd6_10h_100_004. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

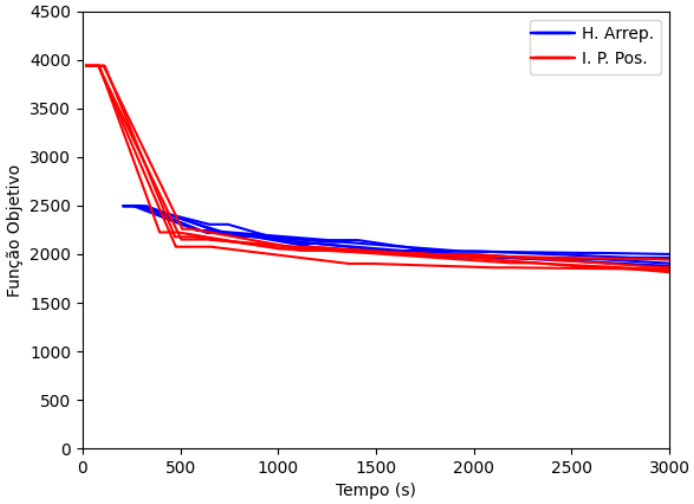


Figura 6 – Gráfico TTT para instância Rnd6_10h_100_005. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

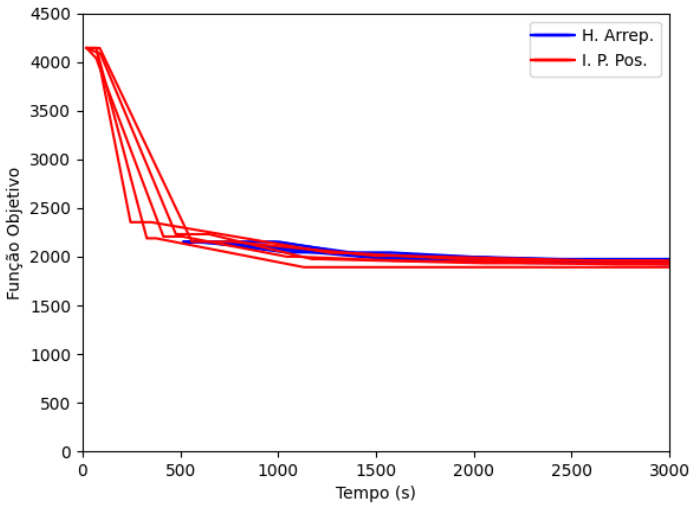


Figura 7 – Gráfico TTT para instância Rnd6_10h_100_006. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

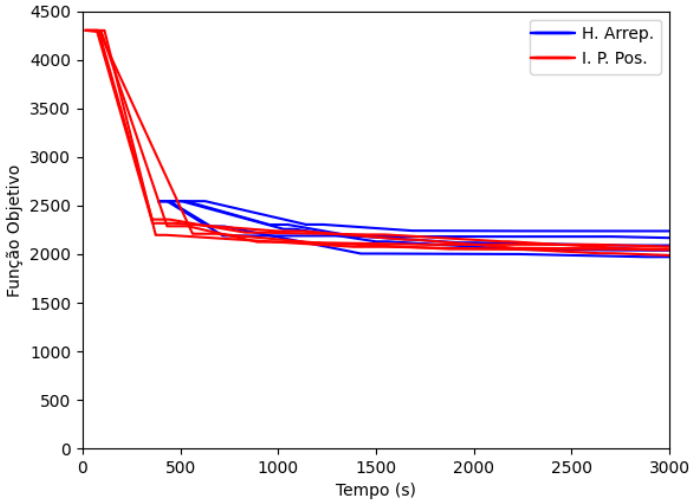


Figura 8 – Gráfico TTT para instância Rnd6_10h_100_007. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

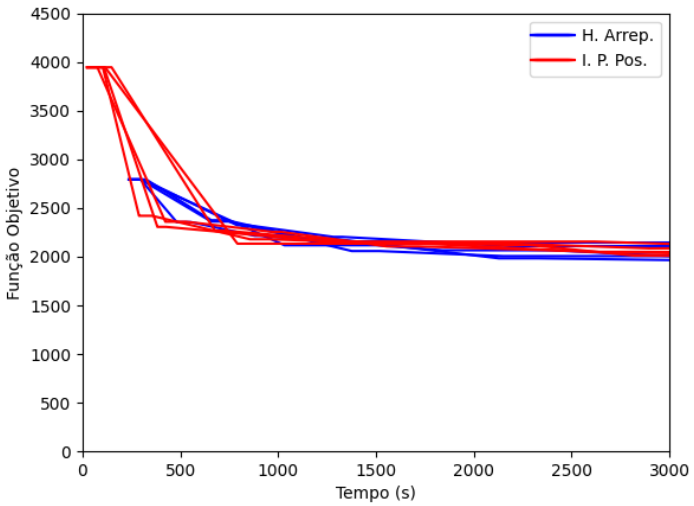


Figura 9 – Gráfico TTT para instância Rnd6_10h_100_008. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).

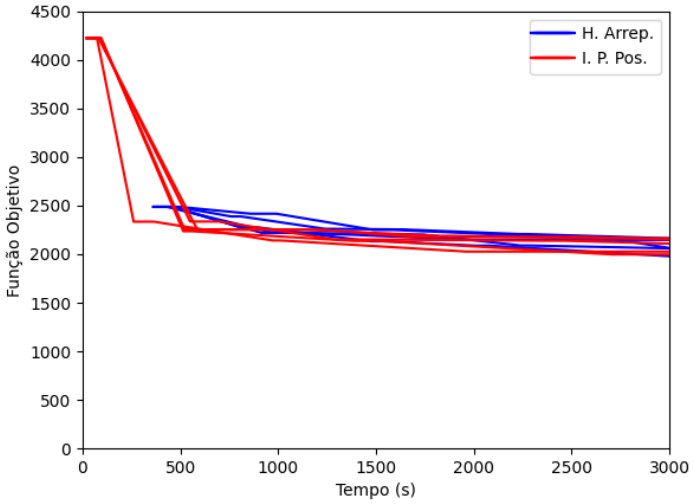


Figura 10 – Gráfico TTT para instância Rnd6_10h_100_009. Em azul (vermelho) é apresentado o comportamento de cada execução da Heurística de Arrependimento (Primeira Inserção Factível).