Heurísticas para o problema de roteamento de veículos com minimização de emissão de gases poluentes

Igor de Andrade Junqueira

JUIZ DE FORA

AGOSTO, 2021

Heurísticas para o problema de roteamento de veículos com minimização de emissão de gases poluentes

Igor de Andrade Junqueira

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Stênio Sã Rosário Furtado Soares
Coorientadora: Lorenza Leão Oliveira Moreno

JUIZ DE FORA
AGOSTO, 2021

Heurísticas para o problema de roteamento de veículos com minimização de emissão de gases poluentes

Igor de Andrade Junqueira

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA , COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Stênio Sã Rosário Furtado Soares
Doutor em Computação - UFF

Lorenza Leão Oliveira Moreno
Doutora em Informática - PUC-RIO

Luciana Brugiolo Gonçalves
Doutora em Computação - UFF

Heder Soares Bernardino
Doutor em Modelagem Computacional - LNCC

JUIZ DE FORA
27 DE AGOSTO, 2021

Resumo

O problema de roteamento verde de veículos com frota heterogênea e janela de tempo é um problema em que o objetivo é minimizar a emissão de CO_2 considerando condições de tráfico ao longo do tempo, para realizar entregas a um conjunto de clientes. O modelo considera um grafo direcionado $G = (V, A)$, onde V é o conjunto que integra clientes e depósito, A é o conjunto de arcos e a função $f : (A, M) \rightarrow \mathbb{R}^+$ define a velocidade de cada arco $(i, j) \in A$ em cada período k do conjunto M de períodos ao longo do dia. O uso da informação dada pela função f contribui tanto para uma maior precisão quanto ao tempo de chegada do veículo no cliente, evitando atrasos e antecipações, quanto no planejamento de rotas que minimiza a poluição emitida pela frota de veículos. Este trabalho propõe um método híbrido eficiente que combina as metaheurísticas GRASP e ILS, que incluem também RVND e *Local Branching*. Experimentos mostraram que o algoritmo proposto consegue gerar soluções com qualidade, sendo três vezes mais rápido do que a literatura.

Palavras-chave: Roteamento Verde de Veículos, Logística Verde, Programação Inteira Mista, GRASP-ILS-RVND-MIP, *Local Branching*

Abstract

The Green Vehicle Routing Problem with heterogeneous fleet and time windows approached in this work aims to reduce CO_2 emissions by considering time-varying traffic conditions to deliver goods to a set of customers. In the model, there is a directed graph $G = (V, A)$, where V is the set of clients and deposit, A is a set of edges and the function $f : (A, M) \rightarrow \mathbb{R}^+$ define the velocity for every edge (i, j) and period k in the set M of periods. The f function contributes to a more precise arrival time of the vehicles to the clients, reducing arrival late and pollution emission of the fleet of vehicles. The proposal combines GRASP and ILS metaheuristics, as well as RVND and Local Branching matheuristic to solve the problem. Experiments showed that the algorithm can achieve high quality solutions three times faster than other literature results.

Keywords: Green Vehicle Routing Problem, Green Logistic, Mixed Integer Programming, GRASP-ILS-RVND-MIP, Local Branching

Agradecimentos

Aos professores Stênio Sã, Lorenza Leão e Luciana Brugiolo, por todos os conselhos, pela ajuda e pela paciência com a qual guiaram o meu aprendizado.

Ao meu Pai e Irmã, por todo o incentivo, amor e carinho. À minha Mãe (*In Memorian*), que me ensinou como se reerguer diante das adversidades da vida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
GEE	Gases do Efeito Estufa
ppm	Partes por milhão
ppb	Partes por bilhão
Gt	Gigatonelada
mm	milímetro
$PFC's$	Compostos perfluorados
NO_2	Óxido Nitroso
CH_4	Metano
$CFC's$	Clorofluorcarbonetos
G-VRPTW	Problema de roteamento verde de veículos com janela de tempo
MDGVRP	Problema de roteamento verde de veículos com múltiplos depósitos
Grasp	*Greedy Randomized Adaptive Search Procedure*
MIP	Programação inteira mista
PSO	*Particle Swarm Optimization*
MOPSO	*multi-objective particle swarm optimization*
MDOVRPTW	*multi-depot open vehicle routing problem time window*
AG	Algoritmo Genético
SA	*Simulated Annealing*
RVND	*Randomized Variable Neighborhood Descent*
ILS	*Iterated local search*
HGVRSP	*Green Vehicle Routing and Scheduling Problem*
LB	*Local Branching*
HFVRPTW	Problema de roteamento de veículos com janelas de tempo e frota heterogênea
G-VRP	*Green Vehicle Routing Problem*
VNS	*Variable Neighbour Source*

1 Introdução

O efeito estufa, segundo [Le Treut et al., 2007], é um processo natural gerado pelo acúmulo de gases do efeito estufa (GEE), como dióxido de carbono (CO_2), metano (CH_4), óxido nitroso (N_2O) e compostos perfluorados ($PFC's$), na atmosfera, que aquecem a superfície do planeta. O aquecimento global é o efeito estufa acelerado pela ação humana através da queima de combustíveis fósseis, gerando CO_2, de más práticas na agricultura, como o uso excessivo de fertilizantes, gerando óxido nitroso(NO_2), do uso de Clorofluorcarbonetos ($CFC's$) para refrigeração (proibido em vários países) e da liberação de hidrocarbonetos, que destroem a camada de ozônio [Forster et al., 2007].

Várias ações humanas contribuem para o aumento de GEE. Pode-se observar pela Figura 1.1 que até 1750, no início da primeira revolução industrial, a concentração de GEE era estável, com pequenas ocilações. Porém, após o início da industrialização, é possível ver um aumento significativo desses gases na atmosfera.

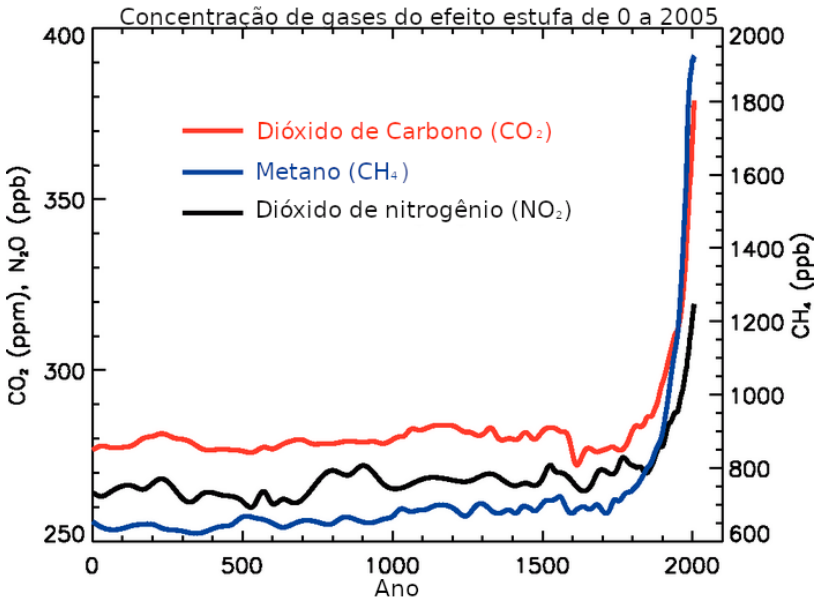

Figura 1.1: Concentração de GEE nos últimos 2000 anos.
Fonte: Traduzido de [Forster et al., 2007]

Algumas consequências do aquecimento global para o planeta são o derretimento de gelo de plataformas continentais e aumento dos níveis dos oceanos. Isso pode ser

constatado nos seguintes dados da *NASA* [1]: aumento da média global do nível do mar em 3,3 mm/ano, redução de gelo na Groenlândia de 278 Gt/ano, redução de gelo na Antártida de 150 Gt/ano, aumento da massa do oceano de 2,1 mm/ano. No Brasil, o setor de transportes é responsável por 44 % das emissões de CO_2 de combustíveis fósseis, segundo [Solaymani, 2019].

Como o transporte é responsável por uma parcela significativa de emissão de CO_2, a logística verde, diferente da logística que tem como objetivo reduzir custos, considera o impacto ambiental ao minimizar emissão de GEE em problemas de roteamento de veículos a combustão, otimizar rotas com uso de veículos elétricos considerando a autonomia de baterias, otimizar o uso de energia elétrica em processos de manufatura, minimizar o desperdício de recursos, etc. Segundo [Paskannaya and Shaban, 2019], diminuir a emissão de CO_2 é um dos principais objetivos da logística verde.

Neste trabalho é tratado o problema de roteamento verde de veículos com frota heterogênea que considera a variação de velocidade ao longo do tempo (HGVRSP, acrônimo de *Heterogeneous Green Vehicle Routing and Scheduling Problem*) e tem como objetivo minimizar a emissão de CO_2 por uma frota de veículos a combustão. Para resolver o HGVRSP, é proposto um método híbrido que utiliza as metahueurísticas GRASP (*Greedy Randomized Adaptive Search Procedure*) e ILS (Iterated local search) combinadas com LB (*Local Branching*). A abordagem proposta consegue resultados com diferença na média de apenas 0,1% do melhor resultado da literatura, porém, com tempos de execução 2,5 mais rápidos.

O restante do trabalho está assim organizado: o Capítulo 2 apresenta a descrição formal do problema; o Capítulo 3 mostra uma revisão bibliográfica no contexto do problema estudado; Os métodos computacionais são apresentados no Capítulo 4, enquanto os resultados e a comparação com literatura são apresentados no Capítulo 5. Por último, as conclusões do trabalho são apresentadas no Capítulo 6.

[1] Último acesso: 21/01/2021

2 Descrição do Problema

O problema *Heterogeneous Green Vehicle Routing and Scheduling Problem* (HGVRSP), descrito em [Xiao and Konak, 2016], é baseado em um problema clássico de otimização combinatória: o problema de roteamento de veículos com janelas de tempo e frota heterogênea (HFVRPTW). O HFVRPTW tem como objetivo encontrar rotas de custo mínimo para atender demandas de transporte dos clientes de forma que (1) todas as demandas sejam atendidas, (2) todos os clientes sejam visitados apenas uma vez, por um só veículo, no intervalo de tempo correspondente à sua janela de atendimento e (3) a capacidade de cada veículo seja respeitada.

Para considerar aspectos ambientais, o HGVRSP incorpora características adicionais ao HFVRPTW, que permitem estimar a poluição gerada pelos veículos ao longo das rotas. Particularmente, o HGVRSP concentra-se no impacto da velocidade sobre os níveis de poluição, como ilustrado na Figura 2.1. Observa-se que em velocidades baixas, menores que 20 km/h, e velocidades altas, maiores que 90 km/h, os veículos têm as maiores taxas de emissão de gás carbônico, cerca de 1 kg de CO_2/km. Por outro lado, com velocidade próxima a 60 km/h a taxa de emissão aproxima-se de 0.5 kg de CO_2/km, isto é, metade da emissão gerada por quilômetro nos casos anteriores. Portanto, considerar a velocidade dos veículos ao longo das rotas é fundamental para reduzir as emissões de CO_2.

A partir das observações do gráfico da Figura 2.1, os autores [Xiao and Konak, 2016], modelaram a emissão de poluentes com base no consumo de combustível. Deste modo, utilizaram taxas de conversão que permitem (1) estimar o consumo de combustível a partir da velocidade, do tempo de percurso, da carga dos veículos e da distância do percurso, e (2) estimar a poluição causada pelo veículo com base no consumo de combustível.

Como a velocidade do veículo ao fazer um percurso varia ao longo do dia, por exemplo, trechos com engarrafamentos (reduz consideravelmente as velocidades), na definição do problema, o dia foi dividido em diferentes períodos de tempo. Em cada período de tempo, para cada trecho da instância, é indicada a velocidade média de um veículo para

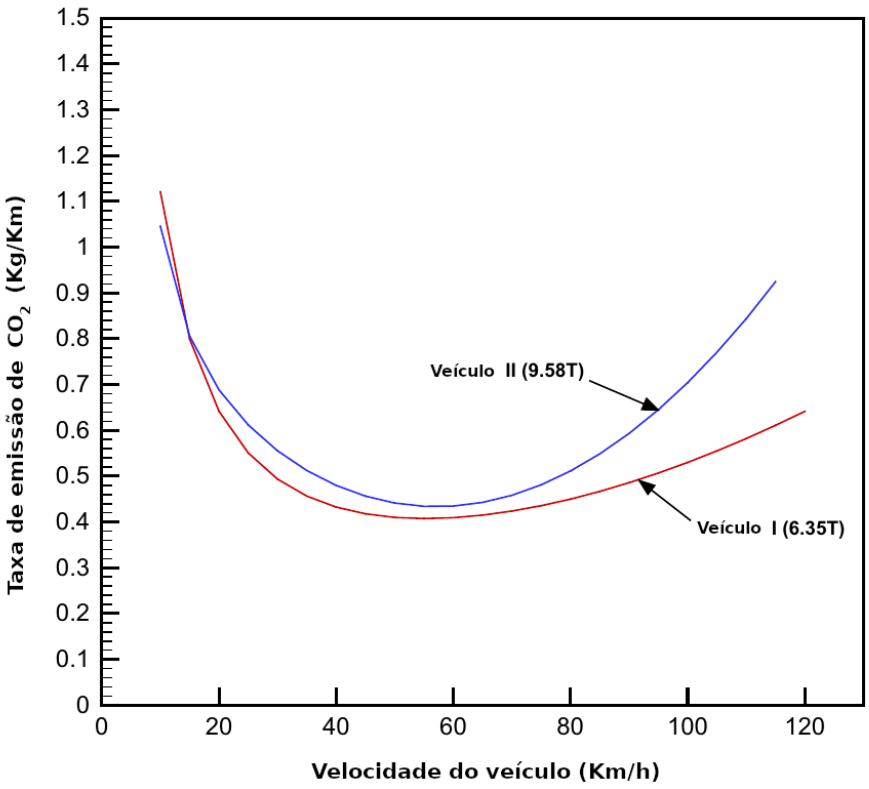

Figura 2.1: Gráfico velocidade $\times$ CO_2,
Fonte: Modificado de [Xiao and Konak, 2016]

percorrer o respectivo trecho. Esse problema não considera a escolha da velocidade, assim, se o veículo percorre um trecho em um determinado período, ele deve necessariamente percorrê-lo com a velocidade média.

As rotas dos veículos podem incluir paradas, para que seja possível esperar um momento mais oportuno (velocidade com taxa mínima de emissão de poluentes) para percorrer um trecho. Neste problema, as paradas podem acontecer nos clientes, ou mesmo durante o percurso de um arco.

Ao percorrer um arco, um veículo possui quatro possibilidades quanto a utilização dos períodos, que são: (1) utiliza-se somente um período para percorrer o arco, como é ilustrado na Figura 2.2. (2) para percorrer o arco é necessário utilizar mais de um período, como é ilustrado na Figura 2.3. (3) é adicionado um tempo a mais ao tempo de saída, como ilustrado na Figura 2.4. (4) o veículo não utiliza um período, ou parte dele, no meio da rota, como é ilustrado na Figura 2.4.

Todas as constantes e variáveis utilizadas na descrição estão em tabelas no Apêndice A.

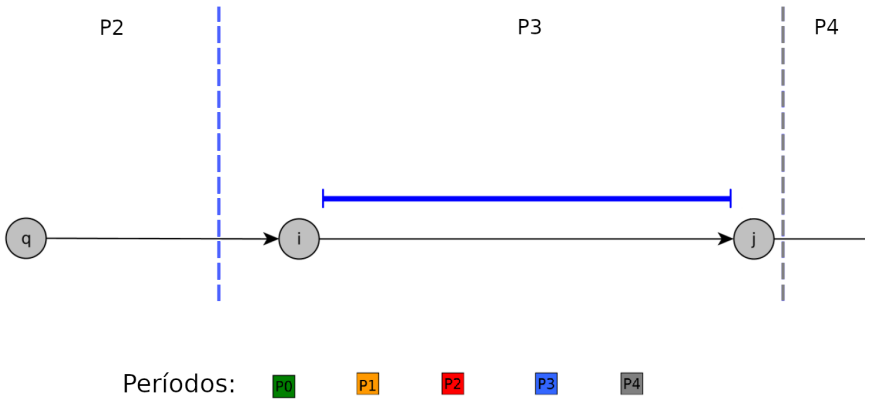

Figura 2.2: Exemplo da utilização de um período ao percorrer um arco

Fonte: Modificado de [RAYLAN, 2018]

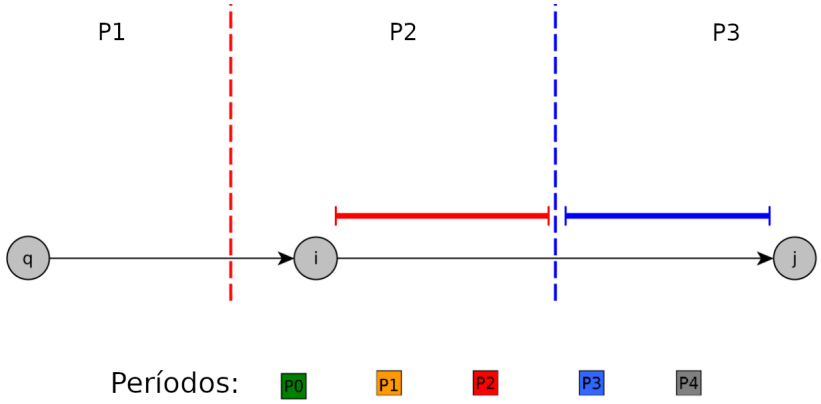

Figura 2.3: Exemplo da utilização de mais de um período ao percorrer um arco,

Fonte: Modificado de [RAYLAN, 2018]

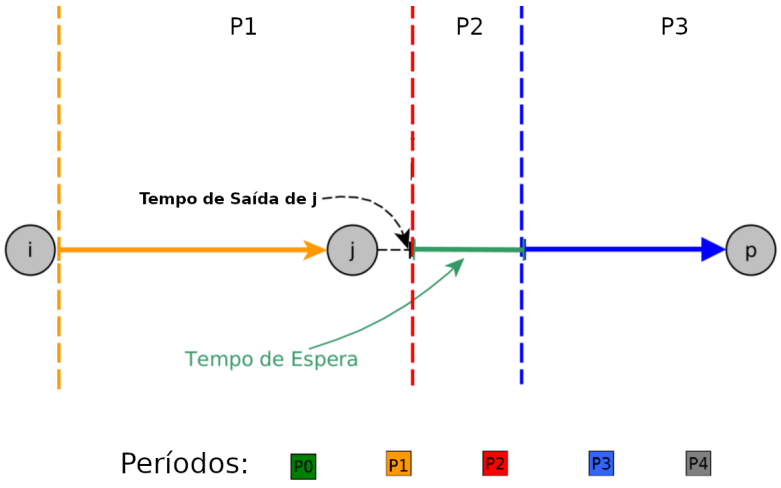

Figura 2.4: Exemplo com espera ao percorrer um arco,

Fonte: Modificado de [RAYLAN, 2018]

O HGVRSP é definido sobre um grafo orientado $G = (V, A)$, onde $V = \{0\} \cup C$ é o conjunto de vértices (em que 0 representa o depósito e $C = \{1, \ldots n\}$ o conjunto de clientes) e $A = \{(i, j) \mid i \neq j \text{ e } i, j \in V\}$ é o conjunto de arcos. A cada arco $(i, j) \in A$ está associada uma distância $D_{i,j}$ relativa ao caminho que liga i e j e a cada nó $i \in C$ está associada uma demanda Dem_i, uma janela de tempo $[T_i^{Ini}, T_i^{Fim}]$ e um tempo de serviço TS_i.

O horizonte de tempo é dividido em partes iguais e é definido pelo conjunto $M = \{0, 1, \ldots, m\}$ de períodos . O início e fim de cada um é definido para todo $k \in M$ como $[T_k^{Ini}$ e $T_k^{Fim}]$, respectivamente. A frota de veículos a combustão é heterogênea, composta por um conjunto de veículos $W = \{0, \ldots, v_{max} - 1\}$. Todo veículo $v \in W$ possui as características: capacidade de carga C_v, capacidade do tanque de combustível Cb_v^{max} e janela de tempo $[T_v^{Ini}, T_v^{Fim}]$.

A velocidade $Vel_{i,j}^k$ de um veículo v varia com o tempo, sendo definida em função do arco (i, j) e do período k. Como indica a Figura 2.3, no percurso de um arco (i, j), são permitidas mudanças de períodos, mas é importante ressaltar que para dois veículos saindo nos tempos t_a e t_b, $t_a \leq t_b$ considera-se que o primeiro veículo sempre chega antes do segundo, independente das velocidades dos períodos. Essa propriedade é conhecida como *non-passing* (ou *FIFO*) descrito por [Ahn and Shin, 1991].

O consumo de combustível q_{comb}^v de um veículo v depende da distância $d_{ij}^{k,v}$ percorrida em cada período k (calculada a partir da velocidade $Vel_{i,j}^k$ e do tempo utilizado para percorrer o arco (i, j) no período k). Depende ainda, da carga variável f_{ij}^v levada pelo veículo v no arco (i, j) e da taxa de consumo de combustível $Cb_{i,j}^{k,v}$ (estimativa de consumo). O consumo de CO_2 pode ser estimado pela Equação 2.2, que utiliza a estimativa de combustível q_{comb}^v e a taxa de conversão de combustível por CO_2, $Tx_v^{CO_2}$.

$$q_{comb}^v = \sum_{(i,j) \in A_r} \sum_{k \in M} Cb_{ij}^{k,v} . d_{ij}^{k,v} + \sum_{(i,j) \in A_r} Cb_v^+ . D_{ij} . f_{ij}^v \tag{2.1}$$

$$CO_2 = Tx_v^{CO_2} . q_{comb}^v \tag{2.2}$$

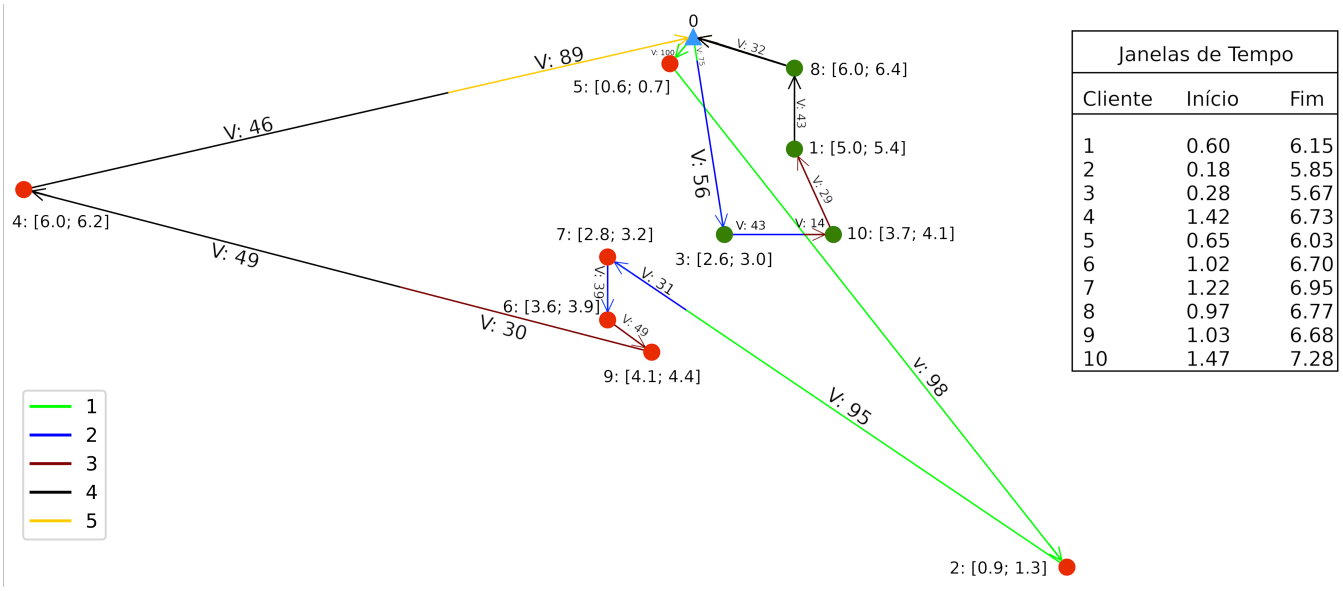

Figura 2.5: Exemplo de uma solução.

O problema consiste em definir rotas de veículos para atendimento dos clientes. Para cada rota, são indicados os clientes visitados, bem como os instantes de saída, de chegada e das paradas do veículo. A solução deve atender às restrições de janela de tempo e de demanda dos clientes, do período de disponibilidade dos veículos, da capacidade de carga e da autonomia de combustível dos veículos. O objetivo do HGVRSP é minimizar o CO_2 gerado pelas rotas, estimado a partir do consumo de combustível de cada veículo.

A Figura 2.5 apresenta uma solução para a instância UK_10x5_1 com 10 clientes, indicando as posições dos clientes, seus respectivos tempos de chegada e saída, assim como uma tabela com as janelas de tempo dos clientes. A velocidade é apresentada para cada arco e períodos utilizados. Ao percorrer o arco $(5, 2)$ somente o 1^{o} período é utilizado, já no arco $(4, 0)$ é necessário utilizar o 4° e 5° períodos. Todas as velocidades indicadas na figura estão em km/h. É possível observar a diferença expressiva das velocidade dos arcos nos períodos, por exemplo, no 1° período as velocidades são: 100, 98, 95 e 75, por outro lado, as velocidades no 3° e 4° períodos são significamente menores, alguns exemplos são: 49, 30, 14, 29, 43 e 32. No último período, 5°, existe um aumento na velocidade, com o único exemplo do arco $(4, 0)$ de 89.

2.1 Modelo Matemático

O modelo matemático descrito a seguir, para o problema HGVRSP, foi inspirado no modelo proposto em [Xiao and Konak, 2016]. A Tabela 2.1 apresenta os dados de entrada do modelo matemático. As variáveis do modelo são descritas na Tabela 2.2, sendo as variáveis X_{ij}, x_{ij}^{v} e x_{ij}^{kv} são binárias e as demais contínuas.

Tabela 2.1: Parâmetros do modelo

Parâmetros	Descrição
V	Conjunto de clientes e depósito
V^{+}	Conjunto de clientes (excluindo o depósito)
A	Conjunto de arestas do subgrafo induzido por V
i, j	Índices dos clientes
M	Conjunto dos períodos
k, m	Índices dos períodos, sendo que m representa o último período
W	Conjunto de veículos
v	Índice do veículo
$D_{i,j}$	Distância entre aresta (i, j)
$Dem_{\,i}$	Demanda do cliente i
$TS_{\,i}$	Tempo de serviço do cliente i
$[T_i^{Ini}, T_i^{Fim}]$	Janela de tempo do cliente i
$[T_k^{Ini}, T_k^{Fim}]$	Início e fim do período k
$[T_v^{Ini},\ T_v^{Fim}]$	Janela de tempo do veículo v
$Vel_{\,ij}^{\,k}$	Velocidade do veículo do arco (i, j) no período k
$Cb_{\,v}^{Max}$	Capacidade máxima do tanque de combustível do veículo v
$Cb_{\,ij}^{k,v}$	Taxa de consumo de combustível ao percorrer o arco (i, j) no período k
$Cb_{\,v}^{+}$	Consumo adicional de combustível para uma unidade a mais de carga
$Tx_{\,v}^{CO_2}$	Taxa de conversão combustível / CO_2

Tabela 2.2: Variáveis do modelo

Variáveis	Descrição
X_{ij}	Indica se o arco (i, j) é percorrido ou não
x_{ij}^{v}	Indica se o veículo v percorre o arco (i, j) ou não
x_{ij}^{kv}	Indica se o veículo v percorre o arco (i, j) em um período k
d_{ij}^{kv}	Indica a distância percorrida do arco (i, j) no período k pelo veículo v
τ_{ij}^{kv}	Indica o tempo gasto no arco (i, j) no período k pelo veículo v
t_i^{sai}	Indica o instante de tempo em que o veículo sai do cliente i
t_i^{chg}	Indica o instante de tempo em que o veículo chega no cliente i
f_{ij}^{v}	Indica a carga do veículo v ao percorrer o arco (i, j)
q_{comb}^{v}	Indica a quantidade total de combustível utilizada na rota do veículo v

O modelo é apresentado nas equações 4.1 à 4.21.

$$\text{Minimizar} \qquad CO_2 = \sum_{v \in W} Tx_v^{CO_2} . q_{comb}^v \tag{2.3}$$

A função objetivo minimiza o CO_2 produzido pelos veículos ao percorrer as rotas, que é calculado pela taxa de conversão de combustível por CO_2 vezes o combustível.

$$\begin{aligned}
\text{Sujeito a} \qquad \sum_{j \in V} x_{ij}^v = \sum_{j \in V} x_{ji}^v \quad & \forall i \in V, v \in W && (2.4)\\
x_{ij}^{kv} \leq x_{ij}^v \quad & \forall (i,j) \in A_r, k \in M, && \\
& \forall v \in W && (2.5)\\
\sum_{j \in V} X_{ij} = 1 \quad & \forall i \in V^+ && (2.6)\\
\sum_{i \in V} X_{ij} = 1 \quad & \forall j \in V^+ && (2.7)\\
\sum_{j \in V^+} x_{0j}^v \leq 1 \quad & \forall v \in W && (2.8)\\
x_{ij}^v \leq \sum_{k \in M} x_{ij}^{kv} \quad & \forall (i,j) \in A, k \in M, && \\
& \forall v \in W && (2.9)\\
X_{ij} = \sum_{v \in W} x_{ij}^v \quad & \forall (i,j) \in A, v \in W && (2.10)
\end{aligned}$$

As restrições 2.4 garantem para todo veículo que o total de arestas saindo é igual ao total de arestas chegando em cada vértice. As restrições 2.5 são inseridas no modelo para que as variáveis x_{ij}^{kv} sejam positivas somente se as variáveis x_{ij}^v forem positivas.

As restrições 2.6 e 2.7 garantem que todo cliente tenha somente uma aresta saindo e chegando. As restrições 2.8 garantem que um veículo tenha no máximo uma aresta de saída do depósito.

As restrições 2.9 são inseridas no modelo para que as variáveis x_{ij}^k sejam positivas somente se as variáveis x_{ij}^{kv} forem positivas. As restrições 2.10 fazem com que as variáveis X_{ij} indiquem a utilização de uma aresta por uma rota.

$$\sum_{l \in V, l \neq i,j} x_{jl}^{k'v} \leq 1 - \sum_{i \in V_r, i \neq j} x_{ij}^{kv} \quad \forall k', k \in M, k' < k, \; \forall j \in V^+, v \in W \tag{2.11}$$

$$d_{ij}^{kv} \leq D_{ij}.x_{ij}^{kv} \quad \forall (i,j) \in A, k \in M, \; \forall v \in W \tag{2.12}$$

$$\sum_{k \in M} \sum_{v \in W} d_{ij}^{kv} = D_{ij}.X_{ij} \quad \forall (i,j) \in A \tag{2.13}$$

As restrições 2.11 impedem a formação de sub-ciclos ao garantir que, se o veículo v chega em um cliente j no período k, não pode sair deste mesmo cliente em um período anterior a k. Cada variável de distância é limitada superiormente ao valor da distância da respectiva aresta em 2.12. As restrições 2.13 controlam as variáveis d_{ij}^{kv} para que a distância total percorrida na mesma aresta (i, j) no veículo v, em diferentes períodos, seja exatamente a distância da aresta (i, j).

$$\sum_{i \in V, i \neq j} \sum_{v \in W} f_{ij}^{v} - \sum_{i \in V, i \neq j} \sum_{v \in W} f_{ji}^{v} = Dem_j \quad \forall j \in V \tag{2.14}$$

$$f_{ij}^{v} \leq (\sum_{l \in V^+} Dem_l).X_{ij} \quad \forall (i,j) \in A, v \in W \tag{2.15}$$

Para cada cliente j, as restrições 2.14 garantem que o veículo v, ao passar por j, tenha uma redução em sua carga (representada pelas variáveis f_{ij}) equivalente à demanda de j. Em 2.15, cada variável que representa a carga em um arco é limitada à demanda total da rota. Este mesmo conjunto de restrições, reduz a carga transportada a zero no arco (i, j), se o veículo não passar pelo respectivo arco, isso é, caso a variável que indica se o veículo percorre este arco seja nula.

$$\tau_{ij}^{kv} = d_{ij}^{kv} / Vel_{ij}^{k} \quad \forall (i,j) \in A, k \in M, \; \forall v \in W \tag{2.16}$$

$$\sum_{(i,j) \in A} \tau_{ij}^{kv} \leq T_k^{Fim} - T_k^{Ini} \quad \forall k \in M, v \in W \tag{2.17}$$

$$t_i^{sai} \leq T_k^{Fim} - \tau_{ij}^{kv} + T_m^{Fim}(1 - x_{ij}^{kv}) \qquad \forall (i,j) \in A, k \in M, \forall v \in W \tag{2.18}$$

$$t_j^{chg} \geq T_k^{Ini} + \tau_{ij}^{kv} - T_m^{Ini}(1 - x_{ij}^{kv}) \qquad \forall (i,j) \in A, k \in M, \forall v \in W \tag{2.19}$$

$$t_j^{chg} \geq t_i^{sai} + \sum_{k \in M} \sum_{v \in W} \tau_{ij}^{kv} - T_m^{Fim}(1 - X_{ij}) \quad \forall (i,j) \in A \tag{2.20}$$

$$t_i^{chg} + TS_i \leq t_i^{sai},\ T_i^{Ini} \leq t_i^{chg} \leq T_i^{Fim} \qquad \forall i \in V \tag{2.21}$$

As restrições 2.16 obtém o tempo de percurso de cada arco e veículo a partir da distância percorrida, considerando a velocidade estimada para o trecho no período indicado. O tempo de percurso de qualquer arco em um período k é limitado ao tempo total deste mesmo período, em 2.17.

As restrições 2.18 e 2.19 determinam o tempo de saída e de chegada do veículo v nos clientes, t_i^{sai} e t_i^{chg}. Nas restrições 2.18, o tempo de saída do cliente i é limitado ao instante final do período k menos o tempo τ_{ij}^{kv} gasto para percorrer o arco no veículo v. De forma similar, as restrições 2.19 limitam o tempo de chegada no cliente j ao valor do início do período k somado ao tempo gasto para percorrer o arco. Nos dois casos, a restrição só é imposta se o veículo v sai (ou chega) em i no período k considerado (quando x_{ij}^{kv} é nula, uma constante é somada ao lado direito para que a limitação não seja imposta).

Para todo arco (i,j) percorrido pelo veículo, as restrições 2.20 asseguram que a chegada a j ocorre a partir do momento correspondente ao tempo de saída de i mais o tempo gasto para percorrer o arco (i,j). Nas restrições 2.21, os tempos de saída e de chegada do veículo nos clientes, t_i^{sai} e t_i^{chg}, são forçados a respeitar o tempo de serviço no cliente i, TS_i e a janela de atendimento do cliente, de T_i^{Ini} a T_i^{Fim}.

$$\sum_{(i,j) \in A} \sum_{k \in M} Cb_{ij}^{k,v}.d_{ij}^{kv} + \sum_{(i,j) \in A} Cb_v^{+}.D_{ij}.f_{ij}^{v} = q_{comb}^{v} \leq Cb_v^{Max} \quad \forall v \in W \tag{2.22}$$

A quantidade de combustível q_{comb}^{v} consumida pelo veículo v é calculada pela restrição 2.22. A primeira parcela é referente à distância percorrida, com base na estimativa de consumo de combustível $Cb_{ij}^{k,v}$ de cada arco (i,j) e período k. A segunda é referente ao

peso transportado pelo veículo v, considerando a distância D_{ij} do arco (i, j), o peso f_{ij}^{v} do veículo v e o coeficiente Cb_{v}^{+}, que indica o consumo de uma unidade de combustível para uma unidade de carga. Na mesma restrição 2.22, a quantidade de combustível é limitada à capacidade do tanque.

$$T_k^{Fim} - \sum_{j \in V^+} \tau_{0j}^{kv} \geq T_v^{Ini} - T_m^{Fim} + \sum_{j \in V^+} T_m^{Fim} x_{0j}^{kv} \quad \forall k \in M, v \in W \tag{2.23}$$

$$t_j^{chg} - \tau_{0j}^{kv} \geq T_v^{Ini} - T_m^{Fim} + T_m^{Fim} x_{0j}^{kv} \quad \forall k \in M, j \in V^+, \; \forall v \in W \tag{2.24}$$

$$T_k^{Ini} + \sum_{i \in V^+} \tau_{i0}^{kv} \leq T_v^{Fim} + T_m^{Fim} - \sum_{i \in V^+} T_m^{Fim} x_{i0}^{kv} \quad \forall k \in M, v \in W \tag{2.25}$$

$$t_i^{sai} + \tau_{i0}^{kv} \leq T_v^{Fim} + T_m^{Fim} - T_m^{Fim} x_{i0}^{kv} \quad \forall k \in M, i \in V, \; \forall v \in W \tag{2.26}$$

As restrições 2.23 garantem que o horário de saída do veículo do depósito é maior ou igual do que o início da janela de tempo do veículo quando mais de um período de tempo são utilizados. O lado esquerdo, com somente um termo τ_{0j}^{kv} diferente de zero, representa o instante de saída do depósito.

Quando o veículo utiliza somente um período para sair do depósito é necessário utilizar o tempo de chegada do primeiro cliente da rota. As restrições 2.24 asseguram que o horário de saída é maior que início da janela de tempo do veículo.

As restrições 2.25 garantem que o horário de chegada do veículo ao depósito é menor ou igual do que o final da janela de tempo do veículo quando mais de um período de tempo são utilizados. O lado esquerdo, com somente um termo τ_{i0}^{kv} diferente de zero, representa o instante de chegada ao depósito.

Quando o veículo utiliza somente um período para chegar ao depósito é necessário utilizar o tempo de saída do cliente da rota. As restrições 2.26 asseguram que o horário de chegada é menor ou igual que final da janela de tempo do veículo.

$$x_{ij}^{kv} \in \{0,1\}, \tau_{ij}^{kv} \geq 0, d_{ij}^{kv} \geq 0 \quad \forall (i,j) \in A, k \in M, \forall v \in W \tag{2.27}$$

$$x_{ij}^{v} \in \{0,1\}, f_{ij}^{v} \geq 0 \quad \forall (i,j) \in A, v \in W \tag{2.28}$$

$$t_i^{chg} \geq 0, t_i^{sai} \geq 0 \quad \forall i \in V \tag{2.29}$$

$$X_{ij} \in \{0,1\} \quad \forall (ij) \in A \tag{2.30}$$

$$q_{comb}^{v} \geq 0 \quad \forall v \in W \tag{2.31}$$

As restrições 2.27 a 2.31 definem os domínios das variáveis. O tempo de saída e chegada ao depósito são facilmente calculados após o resultado ser recuperado do resolvedor, pois, acrescentar isso ao modelo tornaria o mesmo mais complexo e suas respectivas variáveis não são usadas no modelo.

3 Revisão Bibliográfica

Esse capítulo apresenta os trabalhos relacionados a: problemas de otimização em logística verde, cálculo de emissão de CO_2 e dependência de tempo. O objetivo é mostrar os modelos utilizados pela literatura, assim como, as técnicas de inteligência computacional e exatas utilizadas.

Em [Macrina et al., 2020], os autores apresentam uma classificação do G-VRP bem como das abordagens propostas para as variações do problema no período de 2011 a 2019. Por esta classificação, existem duas classes de G-VRP (*Green Vehicle Routing Problem*): os problemas que tratam de veículos à combustão, escopo deste trabalho, e os que tratam de veículos que usam outros combustíveis.

Ao se considerar a dependência de tempo em problemas de roteamento, dois tipos de abordagem são vistos na literatura. Pode-se tratar a dependência de tempo ao se incorporar ao problema a variação de velocidade nas vias ao longo do período de atendimento, ou pode-se considerar que as distâncias entre dois vértices podem sofrer alterações conforme o horário em que o arco é utilizado. O modelo em que para cada arco (i, j) e para cada fração do horizonte de tempo existe uma velocidade permite uma maior precisão nos tempos de saída e chegada e no consumo de combustível, já que o consumo depende da velocidade.

Em [Ahn and Shin, 1991] uma função contínua $A_{i,j}(x)$ foi utilizada para calcular o horário de chegada em j, a continuidade de A permite a realização de *shift* no tempo de chegada, o que contribui com o menor tempo de execução (comparado a refazer a rota completa). A função de tempo possuí duas linhas horizontais e um pico, apesar da premissa está correta (dois períodos com tempo menor e um pico com tempo maior), o tempo não é sempre constante nos período de menor tempo, podem existem variações significativas nesses períodos. [Hill and Benton, 1992] utilizaram as velocidades médias no tempo de saída e chegada em torno dos clientes para calcular a velocidade do arco.

No trabalho de [Xiao and Konak, 2016], os autores utilizaram valores tabelados de velocidades para cada período de cada arco. A propriedade de *non-passing* só é respeitada

quando não existem paradas, uma vez que eles consideraram a mudança de velocidade na interseção de dois períodos. Para resolver o problema, os autores utilizaram um método híbrido com *Variable Neighbour Source* (VNS) e *Mixed Integer Programming* (MIP), onde uma solução de inicio é gerada por heurísticas construtivas e a abordagem exata (modelo matemático) apresenta vários métodos que fixam parte das variáveis (por exemplo, para a variável x: x=0 ou x=1) do modelo de forma aleatória, para resolver o MIP com tempos de execução razoáveis , caso contrário, não seria possível resolve-lo para instâncias maiores que 15 clientes.

Já no trabalho de [Rylan et al., 2017], os autores estendem o problema descrito em [Xiao and Konak, 2016] para permitir que um cliente possa ser atendido por múltiplos veículos (*Split Delivery*) e apresentam um algoritmo ILS *mult start* com a busca local baseada em RVND (*Randomized Variable Neighborhood Descent*). Ao permitir o atendimento do cliente por múltiplos veículos, faz com que seja reduza a emissão de CO_2.

[Küçükoğlu et al., 2015] propuseram um algoritmo para calcular o consumo de combustível para o problema de roteamento verde de veículos com janela de tempo - G-VRPTW, onde o consumo de combustível é tratado como função objetivo e são levados em consideração a aceleração e desaceleração de um veículo até atingir uma velocidade constante, a distância, o peso da carga e especificações do veículo. Para resolver o problema, os autores apresentaram um algoritmo *Simulated Annealing* (SA) e propõem uma estratégia que permite manter uma memória de rotas que guarda o consumo já calculado, evitando recálculos.

Uma perspectiva diferente no universo de problemas de roteamento verde é tratada em [Shen et al., 2018], onde considera-se uma cota máxima de emissão CO_2 e é possível comprar créditos, caso a emissão de CO_2 ultrapasse a cota, ou vender créditos caso contrário. Os autores apresentaram um algoritmo de duas fases usando *PSO (Particle Swarm Optimization)* e busca tabu (*TS*) para resolver o MDOVRPTW (*Multi-Depot Open Vehicle Routing Problem Time Window)*, pelo qual os veículos não precisam retornar aos múltiplos depósitos e os clientes têm janela de tempo.

[Ye et al., 2018] trataram o roteamento de veículos com dependência de tempo e janela de tempo. A função de tempo por velocidade dos pares de clientes (i, j) é discreta

e diferente do descrito no Capítulo 2, já que não permite paradas entre clientes. Na função objetivo desse problema são considerados dois critérios em prioridade, minimizar: i) número total das rotas; ii) distância total percorrida, tempo total de viagem e tempo de espera. O trabalho apresenta duas abordagens do tipo Colônias de Formigas: uma para lidar com a minimização do número de veículos; e outra que busca minimizar a distância e tempo de viagem. As duas colônias compartilham a melhor solução pela atualização do feromônio de forma adaptativa. Quanto maior a média da velocidade mais veículos são utilizados.

Uma abordagem multiobjetivo foi proposta em [Rezaei et al., 2019] para um problema de roteamento verde onde considera-se frota heterogênea quanto à capacidade de carga, custo e consumo de combustível, janela de tempo nos clientes, abastecimento ao longo da rota e tempo máximo de viagem dos veículos. O aspecto multiobjetivo refere-se à minimização dos custos de transporte e da emissão de CO_2. Os autores consideraram, no cálculo do total de emissão de CO_2, a distância percorrida e o peso da carga. Para resolver o problema, propuseram AG (Algoritmo genético) com *crossover* de um ponto e a troca de duas posições aleatórias como mutação. Além do AG, também utilizaram um SA (*Simulated Annealing*) baseado em população. A análise de sensibilidade mostra que ao considerar o reabastecimento em rota reduz o custo, a emissão de CO_2 e o número de veículos.

[Wang et al., 2019] tratam o problema de roteamento verde de veículos com múltiplos depósitos MDGVRP (*Multiple Depot Green Vehicle Routing Problem*) com frota homogênea, onde os depósitos compartilham recursos e cada veículo, após terminar uma rota, pode ir até o depósito mais próximo e ser alocado para uma próxima rota, sendo que os clientes da segunda rota são sempre associados ao depósito de partida. O problema considera uma função contínua que associa a cada arco uma velocidade e atribui penalização por atrasos ou antecipações em relação às janelas de tempo dos clientes. O problema é tratado através de um algoritmo MOPSO (*multi-objective particle swarm optimization*) considerando dois critérios na função objetivo: minimizar a emissão de poluição e o total das penalidades. A minimização da distância não reduz, necessariamente, a emissão de carbono e custo.

[Zhao et al., 2020] descrevem um problema de roteamento de veículos para alimentos frescos com variação no tempo. Os veículos são refrigerados e durante o trajeto de entrega precisam manter a carga em uma determinada faixa de temperatura, a temperatura determina a segurança e descarte de alimento. Cada arco (i,j) é particionado em segmentos de tamanho ϑ (0.2 km) em que a velocidade é constante para um segmento e os períodos de tempo utilizados. Mudanças de velocidades ao longo do trajeto entre dois vértices no arco (i, j) são permitidas entre diferentes segmentos do mesmo e diferentes períodos de tempo. O problema consiste na minimização dos custos, que são: custo fixo por veículo, do tempo, das penalidades por atrasos e antecipações sobre a janela de tempo do cliente, consumo de combustível e emissão de CO_2 do transporte, combustível e emissão de CO_2 no processo de refrigeração, além do custo relacionado à segurança e descarte dos alimentos, a função é mono-objetivo. Um algoritmo ACO (*Ant Colony Optimization*) foi proposto.

Variações do problema de roteamento verde de veículos com velocidade variando ao longo do tempo são encontradas também quando se considera a frota como sendo de veículos elétricos. Em [Lu et al., 2020] foi proposto o TDEVRP(*Time-dependent Electrical Vehicle Routing Problem*), que consiste em um problema de roteamento de veículos elétricos onde existe um depósito, um conjunto de clientes, um conjunto de veículos elétricos homogêneos, e um conjunto de estações de recarga. Considerou-se a recarga total nas estações de recarga, tempo de atendimento e janela de tempo nos clientes. A dependência de tempo trata a velocidade em função do arco e do tempo de saída. Os autores particionaram o horizonte de planejamento em períodos e em cada período a velocidade de um arco é considerada constante. Como considera-se a mudança de período ao percorrer cada arco, a propriedade *non-passing* é respeitada. O consumo de energia da bateria leva em conta a velocidade, o peso do veículo e a distância percorrida. A função objetivo é minimizar o custo total, que consiste no custo de transporte, custo por utilização do veículo, custo do motorista e custo relacionado à energia consumida. Para resolver o problema foi proposta uma abordagem IVNS (*iterated variable neighbourhood search*) e uma solução inicial é gerada pela heurística de economias de *Soloman*.

[Foroutan et al., 2020] descrevem o problema GVRSP (*Green Vehicle Routing and*

Scheduling Problem) com coleta e entrega, cujo objetivo é reduzir os custos operacionais e ambientais. A frota de veículos considerada foi heterogênea, diferenciando-se na capacidade. Os clientes têm demandas potenciais de produtos retornáveis e janela de tempo. Entregas realizadas fora da janela de tempo são penalizadas e o horizonte de planejamento de oito horas foi dividido em intervalos iguais. Para cada arco e cada intervalo de tempo, a velocidade do veículo é constante. São considerados mudança de período, portanto, a propriedade de *non-passing* é respeitada. O cálculo do CO_2 considera a velocidade, distância percorrida e peso do veículo. O cálculo do combustível está em função do CO_2 e a função objetivo minimiza o custo total, que é: total de emissão de CO_2, custo fixo por veículo utilizado, custo variável do veículo e custo de combustível utilizado. Os autores empregaram AG e SA para resolver o problema.

Após a revisão da literatura, é possível concluir que existem diferentes abordagens para a dependência de tempo (velocidade ou tempo de deslocamento) e que isso influência no tempo de processamento dos algoritmos, porque, se o tempo de deslocamento é contínuo, o cálculo dos tempos de chegada dos clientes e a verificação de viabilidade são obtidos rapidamente. Por outro lodo, caso a função é a velocidade (tabelada) em um período de tempo, o cálculo se torna mais custoso, já que é necessário faze-lo em trechos. O cálculo da emissão de CO_2 está diretamente ligado ao consumo de combustível, e esse pode-se considerar diferentes características, como, a velocidade, peso, aceleração e desaceleração, entre outros. As principais técnicas utilizada pela literatura são metaheurísticas e a sua combinação com métodos exatos.

4 Métodos Computacionais

Nesta seção, são descritos os algoritmos propostos para o TD-VRPTW com frota heterogênea. Em linhas gerais, a estratégia utiliza duas metaheurísticas de forma sequencial: um GRASP (*Greedy Randomized Adaptive Search Procedure*), utilizado para gerar diferentes soluções viáveis para o problema e, a partir da melhor solução encontrada, é inicializado um algoritmo ILS (*Iterated Local Search*).

O GRASP proposto utiliza uma heurística de construção e um RVND (*Randomized Variable Neighborhood Descent*) para busca local. Os dois métodos utilizam uma heurística de verificação de viabilidade e inserção de tempo de parada nos clientes.

Para o ILS, um RVND simplificado foi adotado, além de um algoritmo tipo *Local Branching*, baseada em modelos de programação linear inteira mista (MIPs), para otimização das rotas e inserção de tempo de parada nos arcos.

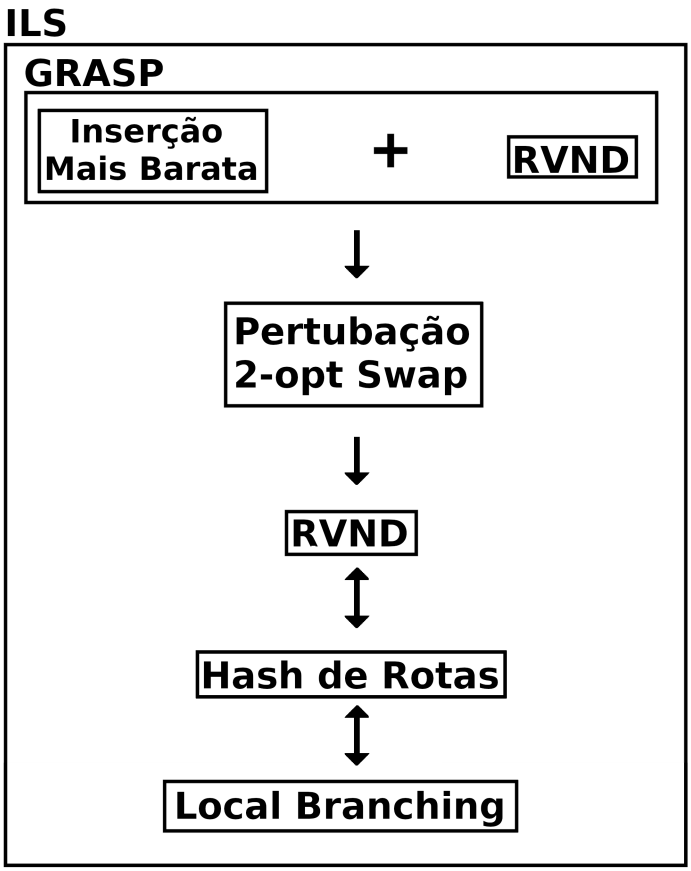

Figura 4.1: Visão geral dos métodos computacionais

A Figura 4.1 apresenta a sequência de aplicação das metaheurísticas. O GRASP gera uma solução para inicializar o ILS, que é composto por uma pertubação 2-opt swap inter rotas. O RVND é utilizado como busca local, e as rotas geradas pelo *Local Branching* são compartilhadas com o RVND pela hash de rotas.

4.1 Heurística construtiva

Para construir uma solução, a heurística de inserção mais barata é utilizada. Inicialmente, todos os veículos estão vazios e a cada iteração um cliente não visitado é inserido em uma rota. No início de cada iteração, para cada cliente candidato a entrar na solução, determina-se o veículo e a posição de inserção que minimize a distância percorrida para atendimento do cliente. Caso não exista uma rota viável para receber o cliente, considera-se inserir este cliente em um veículo fictício (a busca local pode conseguir viabilizar a solução).

Considerando a solução corrente e um dado cliente, o Algoritmo 4.1 descreve a busca pela melhor posição de inserção. Para cada veículo (linha 5) e para cada posição de inserção (linha 7), a possibilidade de atender o cliente é verificada através da estratégia descrita na Subseção 4.1.1 (linha 8). A função *insereCliente* verifica a possibilidade de inserir o cliente na posição indicada pelos parâmetros, analisando viabilidade da janela de tempo e possíveis paradas nos clientes.

Algoritmo 4.1: Escolhe Melhor Inserção

```
1   input: solucao, cliente
2   output: rota
3   rota ← geraRotaVazia();
4   setDistancia(rota, Inf);
5   foreach(veiculo in solucao)
6       posicao ← 1;
7       while(posicao < |veiculo|)
8           resultado ← insereCliente(veiculo, posicao, cliente);
9           if(getDistancia(resultado) < getDistancia(rota))
10              rota ← resultado;
11              atualizaDistancia(rota);
12          posicao ← posicao + 1;
13      end while
14  end foreach
15  return rota;
```

O Algoritmo 4.2 descreve a heurística de inserção mais barata. A cada iteração (linha 6), cada um dos clientes candidatos é analisado (linha 10). A lista *candidatos* registra as tuplas (cliente, rota) indicando a melhor inserção possível de cada cliente na

solução atual. Se não existir uma inserção viável para algum cliente, o mesmo é inserido no veículo fictício, (linhas 14 e 15). Na linha 21, a lista *candidatos* é ordenada considerando poluição ou distância, de acordo com o parâmetro *criterio*. O cliente a ser inserido é selecionado na linha 22, de forma aleatória, entre os primeiros $|candidatos| \times \alpha$ candidatos da lista. Em seguida, a solução é atualizada (linha 23), acrescentando o cliente na solução. Ao final do algoritmo, uma solução viável é construída ou existem um ou mais clientes no veículo fictício.

Algoritmo 4.2: Heurística

```
1   input: clientes, criterio, alfa
2   output: solucao
3   solucao ← ∅;
4   iteracoes ← |Clientes| − 1;
5   i ← 0;
6   while(i < iteracoes)
7       candidatos ← ∅;
8       cliente ← getPrimeiroCliente(clientes);
9       while(cliente ≠ ∅)
10          rota ← EscolheMelhorSolucao(solucao, cliente);
11          if(rota ≠ ∅)
12              candidatos ← candidatos + (cliente, rota);
13          else
14              adicionaVeiculoFicticio(cliente);
15              removeCliente(clientes, cliente);
16          end else
17          cliente ← cliente + 1;
18      end while
19      i ← i + 1;
20      if(candidatos ≠ ∅)
21          sort(candidatos, criterio);
22          candidato ← escolheCandidato(candidatos, alfa);
23          atualizaSolucao(Solucao, getRota(candidato));
24          removeCliente(clientes, getCliente(candidato));
25      end if
26      else
27          break;
28  end while
29  return Solucao;
```

4.1.1 Algoritmo de verificação de viabilidade e inserção de tempo de parada nos clientes

Considerando uma rota de um dado veículo, nesta seção é descrita a estratégia de verificação de viabilidade, no que se refere a janela de tempo, bem como a possibilidade de inserção de tempo de parada nos clientes.

Para checar a viabilidade antes da inserção do tempo de parada, considera-se uma rota r direta, verificando-se a possibilidade de atender os clientes dentro de suas janelas de tempo. O combustível não é considerado (a rota final respeita a capacidade total de combustível), porque, realizando as paradas pode existir uma solução viável.

Considerando o tempo de retorno do veículo ao depósito após a visitação dos clientes, calcula-se o tempo de saída mais tarde, levando-se em conta que o tempo é calculado pela função discreta $Vel_{ij}^{k,v}$, não existindo uma função de tempo e que para calcular o tempo de saída mais tarde é necessário existir a função inversa do tempo. É possível concluir que não existe uma função para calcular o tempo de saída mais tarde, porém, é possível aproximar esse valor com um tempo de saída viável (t_0) para um arco (i, j), adicionar um incremento de tempo q ($t_0 + r.q$, para $r \in \{0, 1, ...\}$). O tempo de

saída mais tarde é $(t_0 + r_{max-1}.q)$ em que $(t_0 + r_{max}.q)$ é o primeiro tempo inviável. Esse tempo é calculado para todos os clientes. Esta informação será importante para definição de possíveis tempos de espera.

Para auxiliar neste processo, um grafo direcionado é construído utilizando cada arco (i, j) da rota, partindo do depósito. Para analisar a possibilidade de inserção de tempos de parada nós clientes, são testadas variações nos tempos de saída (de k em k instantes) entre cada par de clientes. Considerando TS_i o tempo de saída no cliente i, inicialmente calcula-se este valor sem considerar a inserção de tempos de parada. Em seguida, para cada par de clientes (i, j) consecutivos da rota, são gerados novos tempos de saída ($TS_i + k.t$, para $t \in \{1, 2, \dots\}$), sendo que não é permitido ultrapassar o tempo de saída mais tarde já calculado. Para cada valor de t tem-se um tempo de chegada distinto no próximo cliente. Com esse atraso, espera-se que a utilização dos períodos seja alterada, podendo mudar a emissão de CO_2. Opções consecutivas de t que têm o mesmo custo, isto é, causam a mesma poluição e gasto de combustível, são descartados. Neste caso, apenas o percurso com saída mais cedo é mantido. Desta forma, algumas opções de percurso para a mesma aresta da rota original são adicionadas ao grafo, com tempos e custos distintos, todas respeitando a janela de atendimento do cliente.

A Figura 4.2 ilustra este processo. Para a rota (0-8-1-0), inicialmente é inserida no grafo a rota direta, sem paradas nos clientes, representada na figura pelos vértices de id 0, 1, 2, 9. Cada vértice do grafo é rotulado com os atributos *TS* (tempo de saída), *Comb* (combustível), *Pol* (poluição), *predId* (*id* do predecessor) e *TC* (tempo de chegada) e para os arcos na horizontal, que representam o deslocamento do veículo, são registrados *Comb* (combustível) e *pol* (poluição). Os arcos na diagonal representam a inserção de tempo de espera, gerando novos vértices associados a um mesmo cliente. Por exemplo, o vértice id 1 representa a chegada do veículo ao cliente 8, possibilitando a partida do veículo no tempo TS: 1,3 para o próximo cliente da rota (cliente 1). Ao considerar aguardar um tempo $k = 0, 3$ no cliente 8 é gerado um novo vértice (id 3), onde o tempo de saída para o cliente 8 seria $TS : 1, 6$. Da mesma forma, os vértices de id 5 e 7 são gerados, considerando aguardar $2k$e $6k$ instantes de tempo no cliente 8. Os vértices correspondentes a k = $\{3, 4, 5\}$ não foram inseridos no grafo porque apresentariam a mesma poluição da aresta

$(5, 6)$. Todos os vértices associados a um mesmo cliente possuem a mesma cor no grafo. Os vértices de id 4, 6, 8 e 10, 11, 12 foram gerados de forma semelhante.

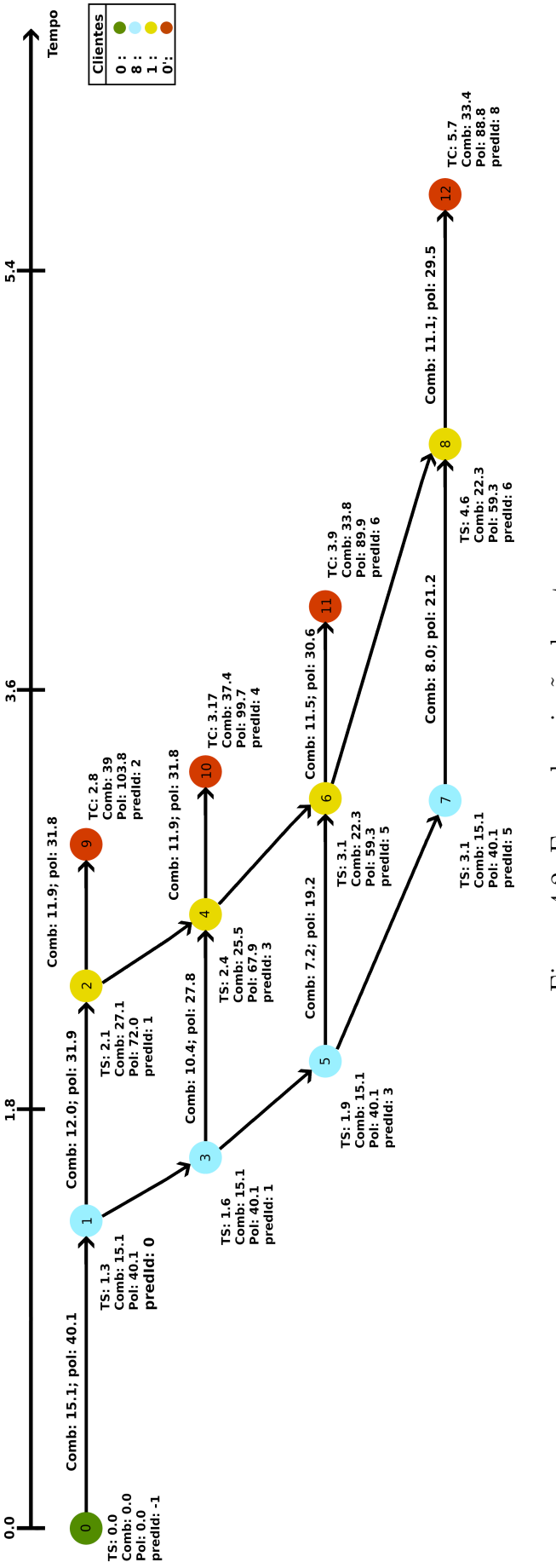

Figura 4.2: Exemplo criação de rotas

O Algoritmo 4.3 implementa o método descrito anteriormente. Na linha 3 verifica-se a existência de uma rota direta. Na linha 8 gera-se o primeiro nó do grafo, partindo do depósito com o tempo de saída igual ao respectivo veículo da rota. Em seguida, na linha 12, são calculados os tempos de saída mais tarde para todos os clientes da rota. O *while* na linha 14 percorre os arcos, o *foreach* na linha 18 cria os percursos para cada nó no conjunto ordenado *clienteI*, os nós do cliente j do arco são adicionados ao conjunto ordenado *clienteJ*. O *while* na linha 34 gera novos percursos adicionando o incremento k ao último nó, enquanto o tempo de saída é menor ou igual ao tempo limite do cliente i do arco.

Algoritmo 4.3: Construção do grafo com tempo de parada

```
1   input: instancia, rota, instanteK
2   output: grafo G(V, E)
3   viavel ← verificaExistenciaRotaDireta(instancia, rota);
4   if(!viavel)
5       return ∅;
6   grafo ← ∅, nextId ← 0;
7   veiculo ← getTipoVeiculo(instancia, rota);
8   no ← criaNo(id: nextId, cliente: 0, ts: getTempoSaida(veiculo));
9   adicionaNo(grafo, no);
10  clienteI ← [], clienteJ ← [];
11  clienteI ← clienteI + no, nextId ← nextId + 1;
12  tempoSaidaMaisTarde ← criaTempoSaidaMaisTarde(rota, instancia);
13  iteracao ← 0;
14  while(iteracao < getNumArcos(rota))
15      arco ← getArco(rota, iteracao);
16      noAnterior ← ∅;
17      proximoNo ← ∅;
18      foreach(no ∈ clienteI)
19          noAux ← criaNo(id: nextId, cliente: getClienteJ(arco));
20          if(proximoNo = ∅)
21              proximoNo ← noAux;
22          adicionaNo(grafo, noAux), nextId ← nextId + 1;
23          percurso ← criaPercurso(no, noAux);
24          criaAresta(grafo, no, noAux, percurso);
25          if(noAnterior ≠ ∅)
26              while(verificaDeferencaTempoSaida(noAnterior, no, instanteK)
27                  criaPercursoIntermediario(noAnterior, no, instanteK, instancia, grafo);
28              percurso ← criaPercursoZero();
29              criaAresta(grafo, noAnterior, no, percurso);
30          end if
31      end foreach
32      tempoLimite ← getTempoLimite(tempoSaidaMaisTarde, getClienteI(arco));
33      proximoTs ← getTempoSaida(no) + instanteK;
34      while(proximoTs ≤ tempoLimite)
35          noAux ← criaNo(id: nextId, cliente: getClienteI(arco), tempoSaida: proximoTs);
36          adicionaNo(grafo, noAux);
37          nextId ← nextId + 1;
38          percurso ← criaPercursoZero();
39          criaAresta(grafo, no, noAux, percurso);
40          no ← noAux;
41          noAux ← criaNo(id: nextId, cliente: getClienteJ(arco));
42          adicionaNo(grafo, noAux), nextId ← nextId + 1;
43          percurso ← criaPercurso(no, noAux);
44          criaAresta(grafo, no, noAux, percurso);
45          proximoTs ← getTempoSaida(no) + instanteK;
46      end while
47      no ← proximoNo, iteracao ← iteracao + 1;
48  end while
49  return grafo;
```

Após o grafo ser gerado, um algoritmo de caminho mais curto (algoritmo de *Dijkstra* proposto em [Dijkstra et al., 1959]) encontra a sequência de tempo de parada minimizando a poluição.

Para o exemplo da Figura 4.2 o menor caminho é o 0, 1, 3, 5, 6, 8, 12 que corresponde há inserção de tempo de parada nos clientes 8 e 1.

4.2 Programação Inteira e *Local branching*

Ao gerar uma rota, a heurística da seção anterior restringe algumas ações do veículo, por exemplo, ao não considerar a possibilidade do veículo fazer uma parada ao longo do percurso entre clientes. Na tentativa de incorporar todas as ações permitidas pela definição do problema no processo de solução, optou-se por utilizar técnicas de programação inteira para otimizar as rotas. Assim, para uma determinada solução gerada heuristicamente, cada rota será otimizada através de um algoritmo de busca local definido sobre um problema de programação matemática, conhecido como *Local Branching*.

O modelo matemático é descrito a seguir e, posteriormente, o algoritmo de *local branching* será detalhado.

4.2.1 Modelo matemático

O modelo a seguir foi adaptado da Seção 2.1 para otimizar uma única rota r, previamente gerada de forma heurística, permitindo paradas em qualquer ponto do trajeto. Somente uma rota é considerada, porque, devido à complexidade do modelo, só é possível resolver instâncias pequenas (e mesmo assim com tempo de execução elevado).

O modelo considera um subconjunto do grafo original $G(V, A)$ de uma instância, restrito aos pontos que compõem uma rota específica r. O modelo mantém o mesmo veículo da rota indicada (isso é, o veículo é fixo), mas permite que a ordem de visitação dos clientes e, consequentemente, os instantes de tempo relacionados sejam modificados. A Tabela 4.1 apresenta os dados de entrada do modelo matemático. As variáveis do modelo são descritas na Tabela 4.2, sendo as variáveis x_{ij} e x_{ij}^k binárias e as demais contínuas.

Tabela 4.1: Parâmetros do modelo

Parâmetros	Descrição
V_r	Conjunto de pontos (depósito e clientes) da rota r
V_r^+	Conjunto de clientes da rota r (excluindo o depósito)
A_r	Conjunto de arestas do subgrafo induzido por V_r
A_r^*	Subconjunto de arestas de A_r composto pelas arestas originais da rota r
M	Conjunto dos períodos
i, j	Índices dos clientes
k, m	Índices dos períodos, sendo que m representa o último período
$D_{i,j}$	Distância entre aresta (i, j)
$Dem_{\,i}$	Demanda do cliente i
$TS_{\,i}$	Tempo de serviço do cliente i
$[T_i^{Ini}, T_i^{Fim}]$	Janela de tempo do cliente i
$[T_k^{Ini}, T_k^{Fim}]$	Início e fim do período k
$[T_v^{Ini},\ T_v^{Fim}]$	Janela de tempo do veículo
$Vel_{\,ij}^{\,k}$	Velocidade do veículo do arco (i, j) no período k
$Cb^{\,Max}$	Capacidade máxima do tanque de combustível do veículo
v	Tipo de veículo v associado a rota r.
$Cb_{\,ij}^{\,k,v}$	Taxa de consumo de combustível ao percorrer o arco (i, j) no período k
Cb_v^+	Consumo adicional de combustível para uma unidade a mais de carga
$Tx_v^{CO_2}$	Taxa de conversão combustível / CO_2

Tabela 4.2: Variáveis do modelo matemático que otimiza uma rota r

Variáveis	Descrição
x_{ij}	Indica se o arco (i, j) é percorrido ou não
x_{ij}^k	Indica se um arco (i, j) é percorrido em um período k
d_{ij}^k	Indica a distância percorrida do arco (i, j) no período k
τ_{ij}^k	Indica o tempo gasto no arco (i, j) no período k
t_i^{sai}	Indica o instante de tempo em que o veículo sai do cliente i
t_i^{chg}	Indica o instante de tempo em que o veículo chega no cliente i
f_{ij}	Indica a carga do veículo ao percorrer o arco (i, j)
q_{comb}	Indica a quantidade total de combustível utilizada na rota.

O modelo é apresentado nas equações 4.1 à 4.21.

$$\text{Minimizar} \qquad CO_2 = Tx_v^{CO_2}.q_{comb} \tag{4.1}$$

$$\text{Sujeito a} \qquad \sum_{j \in V_r} x_{ij} = \sum_{j \in V_r} x_{ji} = 1 \qquad \forall i \in V_r \tag{4.2}$$

$$x_{ij}^k \leq x_{ij} \qquad \forall (i,j) \in A_r, k \in M \tag{4.3}$$

$$\sum_{l \in V_r, l \neq i,j} x_{jl}^{k'} \leq 1 - \sum_{i \in V_r, i \neq j} x_{ij}^k \qquad \forall k', k \in M, k' < k, j \in V_r \tag{4.4}$$

$$d_{ij}^k \leq D_{ij}.x_{ij}^k \qquad \forall (i,j) \in A_r, k \in M \tag{4.5}$$

$$\sum_{k \in M} d_{ij}^k = D_{ij}.x_{ij} \qquad \forall (i,j) \in A_r \tag{4.6}$$

$$\sum_{i \in V_r, i \neq j} f_{ij} - \sum_{i \in V_r, i \neq j} f_{ji} = Dem_j \qquad \forall j \in V_r \tag{4.7}$$

$$f_{ij} \leq (\sum_{l \in V_r^+} Dem_l).x_{ij} \qquad \forall (i,j) \in A_r \tag{4.8}$$

$$\tau_{ij}^k = d_{ij}^k / Vel_{ij}^k \qquad \forall (i,j) \in A_r, k \in M \tag{4.9}$$

$$\sum_{(i,j) \in A_r} \tau_{ij}^k \leq T_k^{Fim} - T_k^{Ini} \qquad \forall k \in M \tag{4.10}$$

$$t_i^{sai} \leq T_k^{Fim} - \tau_{ij}^k + T_m^{Fim}(1 - x_{ij}^k) \qquad \forall (i,j) \in A_r, k \in M \tag{4.11}$$

$$t_j^{chg} \geq T_k^{Ini} + \tau_{ij}^k - T_m^{Ini}(1 - x_{ij}^k) \qquad \forall (i,j) \in A_r, k \in M \tag{4.12}$$

$$t_j^{chg} \geq t_i^{sai} + \sum_{k \in M} \tau_{ij}^k - T_m^{Fim}(1 - x_{ij}) \qquad \forall (i,j) \in A_r \tag{4.13}$$

$$t_i^{chg} + TS_i \leq t_i^{sai}, \; T_i^{Ini} \leq t_i^{chg} \leq T_i^{Fim} \qquad \forall i \in V_r \tag{4.14}$$

$$\sum_{(i,j) \in A_r} \sum_{k \in M} Cb_{ij}^{k,v}.d_{ij}^k + \sum_{(i,j) \in A_r} Cb_v^+.D_{ij}.f_{ij} = q_{comb} \leq Cb^{Max} \tag{4.15}$$

$$T_k^{Fim} - \sum_{j \in V_r^+} \tau_{0j}^k \geq T_v^{Ini} - T_m^{Fim} + \sum_{j \in V_r^+} T_m^{Fim} x_{0j}^k \qquad \forall k \in M \tag{4.16}$$

$$t_j^{chg} - \tau_{0j}^k \geq T_v^{Ini} - T_m^{Fim} + T_m^{Fim} x_{0j}^k \qquad \forall k \in M, \forall j \in V_r^+ \tag{4.17}$$

$$T_k^{Ini} + \sum_{i \in V_r^+} \tau_{i0}^k \leq T_v^{Fim} + T_m^{Fim} - \sum_{i \in V_r^+} T_m^{Fim} x_{i0}^k \qquad \forall k \in M \tag{4.18}$$

$$t_i^{sai} + \tau_{i0}^k \leq T_v^{Fim} + T_m^{Fim} - T_m^{Fim} x_{i0}^k \qquad \forall k \in M, \forall i \in V_r^+ \tag{4.19}$$

$$x_{ij}^k \in \{0,1\}, \tau_{ij}^k \geq 0, d_{ij}^k \geq 0 \qquad \forall i,j \in V_r, \forall k \in M \tag{4.20}$$

$$x_{ij} \in \{0,1\}, t_i^{chg} \geq 0, t_i^{sai} \geq 0, f_{ij} \geq 0, \qquad \forall i,j \in V_r \tag{4.21}$$

As restrições 4.2 asseguram que o veículo visita todos os vértices da rota, garantindo que são selecionadas exatamente uma aresta saindo e uma aresta chegando em cada vértice.

As restrições 4.3 são inseridas no modelo para que as variáveis x_{ij}^k sejam positivas somente se as variáveis x_{ij} forem positivas.

As restrições 4.4 impedem a formação de sub-ciclos ao garantir que, se o veículo chega em um cliente j no período k, não pode sair deste mesmo cliente em um período anterior a k. Cada variável de distância é limitada superiormente ao valor da distância da respectiva aresta em 4.5. As restrições 4.6 controlam as variáveis d_{ij}^k para que a distância total percorrida na mesma aresta (i, j), em diferentes períodos, seja exatamente a distância da aresta (i, j).

Para cada cliente j, as restrições 4.7 garantem que o veículo, ao passar por j, tenha uma redução em sua carga (representada pelas variáveis f_{ij}) equivalente à demanda de j. Em 4.8, cada variável que representa a carga em um arco é limitada à demanda total da rota. Este mesmo conjunto de restrições, reduz a carga transportada a zero no arco (i, j), se o veículo não passar pelo respectivo arco, isso é, caso a variável que indica se o veículo percorre este arco seja nula.

As restrições 4.9 obtém o tempo de percurso de cada arco a partir da distância percorrida, considerando a velocidade estimada para o trecho no período indicado. O tempo de percurso de qualquer arco em um período k é limitado ao tempo total deste mesmo período, em 4.10.

As restrições 4.11 a 4.12 determinam o tempo de saída e de chegada do veículo nos clientes, t_i^{sai} e t_i^{chg}. Nas restrições 4.11, o tempo de saída do cliente i é limitado ao instante final do período k menos o tempo gasto para percorrer o arco, τ_{ij}^k. De forma similar, as restrições 4.12 limitam o tempo de chegada no cliente j ao valor do início do período k somado ao tempo gasto para percorrer o arco. Nos dois casos, a restrição só é imposta se o veículo sai (ou chega) em i no período k considerado (quando x_{ij}^k é nula, uma constante é somada ao lado direito para que a limitação não seja imposta).

Para todo arco (i, j) percorrido pelo veículo, as restrições 4.13 asseguram que a chegada a j ocorre a partir do momento correspondente ao tempo de saída de i mais o tempo gasto para percorrer o arco (i, j). Nas restrições 4.14, os tempos de saída e de chegada do veículo nos clientes, t_i^{sai} e t_i^{chg}, são forçados a respeitar o tempo de serviço no cliente i, TS_i e a janela de atendimento do cliente, de T_i^{Ini} a T_i^{Fim}.

A quantidade de combustível consumida pelo veículo q é calculada pela restrição 4.15. A primeira parcela é referente à distância percorrida, com base na estimativa de consumo de combustível $Cb_{ij}^{k,v}$ de cada arco (i, j) e período k. A segunda é referente ao peso transportado pelo veículo, considerando a distância D_{ij} do arco (i, j), o peso no veículo f_{ij} e o coeficiente Cb_v^+, que indica o consumo de uma unidade de combustível para uma unidade de carga. Na mesma restrição 4.15, a quantidade de combustível é limitada à capacidade do tanque.

As restrições 4.16 garantem que o horário de saída do veículo do depósito é maior ou igual do que o início da janela de tempo do veículo quando mais de um período de tempo são utilizados. O lado esquerdo, com somente um termo τ_{0j}^k diferente de zero, representa o instante de saída do depósito.

Quando o veículo utiliza somente um período para sair do depósito é necessário utilizar o tempo de chegada do primeiro cliente da rota. As restrições 4.17 asseguram que o horário de saída é maior que início da janela de tempo do veículo.

As restrições 4.18 garantem que o horário de chegada do veículo ao depósito é menor ou igual do que o final da janela de tempo do veículo quando mais de um período de tempo são utilizados. O lado esquerdo, com somente um termo τ_{i0}^k diferente de zero, representa o instante de chegada ao depósito.

Quando o veículo utiliza somente um período para chegar ao depósito é necessário utilizar o tempo de saída do cliente da rota. As restrições 4.19 asseguram que o horário de chegada é menor ou igual que final da janela de tempo do veículo.

A função objetivo minimiza o CO_2 produzido pelo veículo ao percorrer a rota, que é calculado pela taxa de conversão de combustível por CO_2 vezes o combustível.

O tempo de saída e chegada ao depósito são facilmente calculados após o resultado ser recuperado do resolvedor, pois, acrescentar isso ao modelo tornaria o mesmo mais complexo e suas respectivas variáveis não são usadas no modelo.

4.2.2 O algoritmo de *local branching*

Local branching (LB) é um algoritmo de busca local proposto por [Fischetti and Lodi, 2003], que utiliza um modelo de programação linear inteira (MIP) para encontrar no-

vas soluções viáveis para determinado problema. Uma restrição é inserida no MIP para restringir o espaço de busca do problema a uma vizinhança em torno de uma solução de referência, usualmente, limitando o número de modificações que podem ser realizadas nesta solução.

$$\sum_{(i,j)\in A_r^*} x_{ij} \geq |A_r^*| - 3 \tag{4.22}$$

A restrição 4.22 de *local branching* permite que até três arestas (i, j) da rota original sejam substituídas durante a otimização. Para isso, a soma das variáveis x_{ij} correspondentes às arestas (i, j) da rota r, que inicialmente equivale ao número de arestas da rota, pode ter redução de até 3 unidades. Com a restrição, a rota original continua sendo viável, mas pode também ser substituída por uma rota de menor custo à medida que alguma aresta (i, j) deixe de ser utilizada (e a respectiva variável passe a valer 0).

Como todo algoritmo de busca local, LB redefine a vizinhança reiteradamente. A cada otimização em que uma nova solução é retornada, a solução de referência é substituída e a restrição de LB é atualizada. Eventualmente, quando a otimização não é capaz de encontrar uma nova solução, considera-se que o algoritmo atingiu um ótimo local.

Algoritmo 4.4: *Local Branching*

```
1   input: instancia, rota, numMaxIteracoes
2   output: rota
3   modelo ← criaModelo(instancia);
4   criaRestricaoLB(modelo, rota);
5   novaRota ← mip(modelo, rota);
6   while(novaRota != rota)
7       rota ← novaRota;
8       atualizaRestricaoLB(modelo, rota);
9       novaRota ← mip(modelo, rota);
10  end while
11  return rota;
```

O Algoritmo 4.4 ilustra o *local branching*. A função *criaModelo*, utilizada na linha 3, cria o modelo apresentado na Subseção 4.2.1, considerando apenas os pontos visitados pela respectiva rota. A função *criaRestricaoLB* insere a restrição de LB 4.22 tendo como referência a rota inicial. A cada nova rota encontrada pelo algoritmo, a restrição de LB é reescrita pela função *atualizaRestricaoLB*.

Na função *mip*, o modelo é otimizado até atingir um valor menor ou igual à

15% do *gap* de integralidade, porque, para atingir o ótimo, é necessário muito tempo de computação e pode não existir uma solução inteira com esse valor. Quando a solução é recuperada, verifica-se se a mesma é diferente da rota atual. Se sim, a restrição de LB é atualizada com a nova rota e o algoritmo continua. Caso contrário, a rota atual é considerada a melhor rota que o modelo foi capaz de encontrar na vizinhança definida. Neste caso, o algoritmo é encerrado, retornando a rota atual.

Por exemplo, considerando a rota $0 - a - b - c - 0$, o somatório do lado esquerdo incluiria as variáveis x_{0a}, x_{ab}, x_{bc} e x_{c0}, permitindo que até 3 fossem substituídas durante a otimização. Com isso, o modelo irá selecionar a opção de menor custo entre as rotas (0-b-a-c-0), (0-a-c-b-0), (0-b-c-a-0), (0-c-a-b-0) e a rota original (0-a-b-c-0).

O modelo, ao selecionar a melhor sequência vizinha a uma rota de referência, considera todas as possíveis combinações de paradas nas arestas, determinando instantes de partida e chegada do veículo. Toda rota gerada respeita as restrições de janela de tempo dos clientes, capacidade de combustível e janela de atendimento do veículo.

4.2.3 Extensão do modelo para duas rotas

No intuito de aumentar a vizinhança definida pela busca local, o modelo apresentado anteriormente foi adaptado para otimizar duas rotas simultaneamente. Neste caso, além de permitir uma permutação da sequência de clientes visitados em uma rota e determinação de novos instantes de partidas, chegadas e paradas, o modelo também explora a troca de clientes entre as rotas.

Pequenas adaptações são necessárias no modelo. Em relação às variáveis, todas recebem um índice adicional para identificar a respectiva rota. São criadas ainda variáveis adicionais para cada rota, incluindo arestas e vértices relativos aos clientes visitados na outra rota (ou seja, para cada rota, os índices i e j passam a considerar o conjunto de clientes de ambas as rotas).

Há também adaptações nas restrições do modelo. Um somatório relativo às rotas é incorporado às expressões da função objetivo e das restrições 4.2 de cada cliente. Todas as demais restrições, de 4.3 a 4.21, são duplicadas, isto é, todo este conjunto de restrições é criado separadamente para cada rota.

O algoritmo de *local branching* passa a receber um par de rotas da solução e, a cada passo, acrescenta uma restrição 4.22 para cada rota. Ao final, as rotas obtidas substituem as originais na solução heurística.

4.3 *Hash* de rotas

Para evitar recalculo do modelo da Subseção 4.2 a hash djb2, criada por Dan Bernstein [Bernstein, 1991], foi modificada para funcionar com uma rota. Originalmente essa hash é utilizada em cadeias de caracteres, o algoritmo 4.5 recebe uma string, inicia o valor da hash com 5381 e em cada carácter multiplica o valor atual da hash e soma o valor da tabela *ASCII* do mesmo.

Algoritmo 4.5: Hash dbj2

```
1        input: string
2        output: hash
3        hash ← 5381;
4        i ← 0;
5        while(i < string.size())
6             hash ← hash * 33 + string[i];
7             i ← i + 1;
8        end while
9        return hash;
```

A hash de uma rota é realizada utilizando cada cliente como um carácter. O Algoritmo 4.6 não utiliza a primeira e a última posição, porque é o depósito, o identificador do cliente é utilizado no lugar do carácter.

Algoritmo 4.6: Hash dbj2 rotas

```
1        input: rota
2        output: hash
3        hash ← 5381;
4        i ← 1;
5        while(i < rota.size()-1)
6             hash ← hash * 33 + rota[i];
7             i ← i + 1;
8        end while
9        Return hash;
```

Uma tabela com uma lista ligada para tratar as colisões foi utilizado para implementar a memória de rotas, no modelo da Subseção 4.2 uma rota r_0 é otimizada gerando uma rota r_1, se a sequencia r_1 é diferente de r_0 as duas rotas são inseridas, a rota r_0 aponta para rota r_1. Quando a rota r_0 é encontrada, é a rota r_1 que é recuperada. Quando as sequencias r_0 e r_1 são iguais só a rota r_1 é inserida. Como o modelo da Subseção 4.2 permite que as rotas troquem clientes, só as rotas após o modelo são inseridas.

4.4 RVND

O VND (*Variable Neighborhood Descent*) é uma abordagem que permite a combinação de diferentes movimentos de busca local. Enquanto no VND é necessário indicar a ordem de exploração em cada uma das vizinhanças, no RVND (*Randomized Variable Neighborhood Descent*) isso não é necessário. Quando o RVND é chamado, define-se aleatoriamente a ordem que as vizinhanças são analisadas.

A seguir são apresentados os movimentos que foram utilizados na busca local. Para exemplificar é considerada a solução ilustradas na Figura 4.3, composta por duas rotas (0-6-3-1-0) e (0-7-4-8-2-5-0).

No movimento *shift* intra rota, um cliente é escolhido de uma rota e inserido em qualquer lugar da mesma. A Figura 4.4 mostra a aplicação do movimento, com o cliente 1 sendo inserido no arco (0-6), gerando a rota: (0-1-6-3-0). No movimento *swap* intra rota, dois clientes de uma rota são trocados. A Figura 4.5 mostra o resultado do movimento com os clientes 4 e 8, gerando a rota: (0-7-8-4-2-5-0).

No movimento *shift* inter rotas, um cliente é escolhido de uma rota e inserido em qualquer posição de uma segunda rota. A Figura 4.6 mostra que cliente 2 foi inserido no arco (3,1), gerando as rotas: (0-6-3-2-1-0) e (0-7-4-8-5-0). Já pelo movimento *swap* inter rotas, dois clientes são trocados de rotas distintas. A Figura 4.7 mostra que os clientes 1 e 5 são trocados, gerando as rotas: (0-6-3-5-0) e (0-7-4-8-2-1-0).

No movimento 2-opt intra rota, duas arestas de uma rota são retiradas, a sequência de clientes entre as arestas é invertida e as arestas são criadas. A Figura 4.8 destaca as arestas (5-0) e (7-4), invertendo a sequência (4-8-2) para (2-8-4) e adicionando as arestas (7-5) e (4-0), gerando a rota: (0-7-5-8-8-4-0). No movimento 2-opt inter rotas, duas arestas de duas rotas distintas são retiradas, as sequências (i,...,0) e (j,...,0) são trocadas. A Figura 4.9 mostra que as arestas (8-2) e (3-1) são escolhidas e as sequências (2-5) e (1) são trocadas, gerando as rotas: (0-7-4-8-1-0) e (0-6-3-2-5-0).

No movimento inverte rota, uma rota selecionada tem seu sentido invertido. A Figura 4.10 mostra a rota invertida: (0-1-3-6-0). Por outro lado, no movimento troca o tipo de rota, duas rotas de tipos distintos são selecionadas, e seus tipos são trocados.

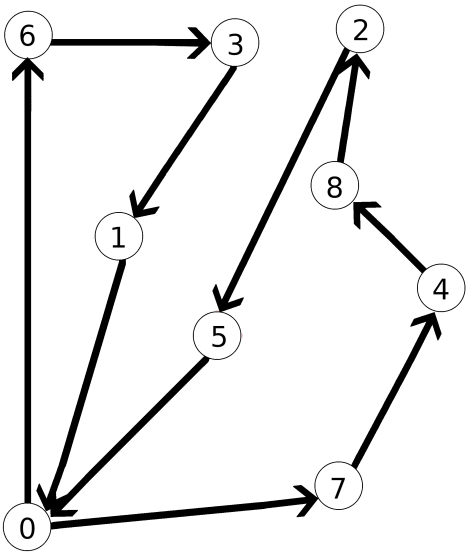

Figura 4.3: Rota inicial

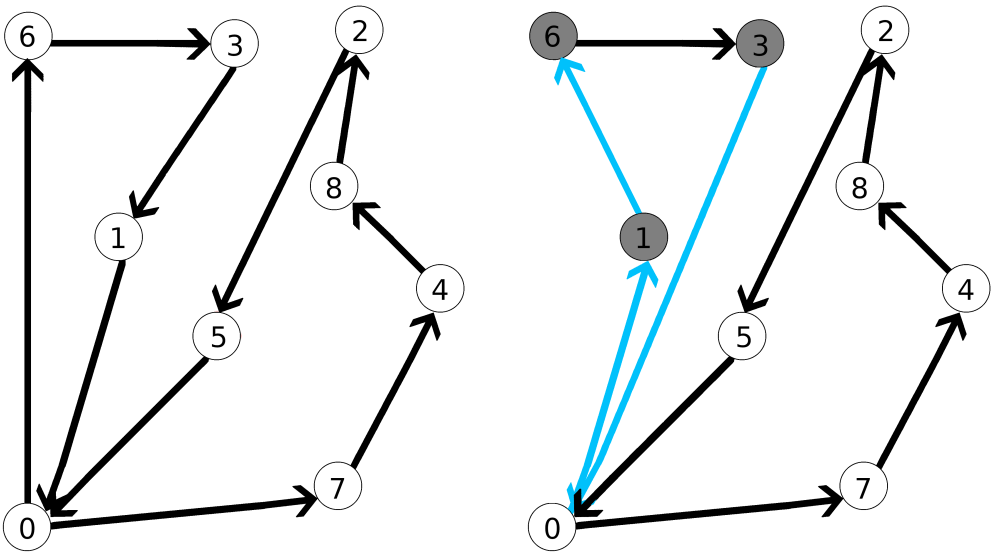

Figura 4.4: Movimento shift intra rota

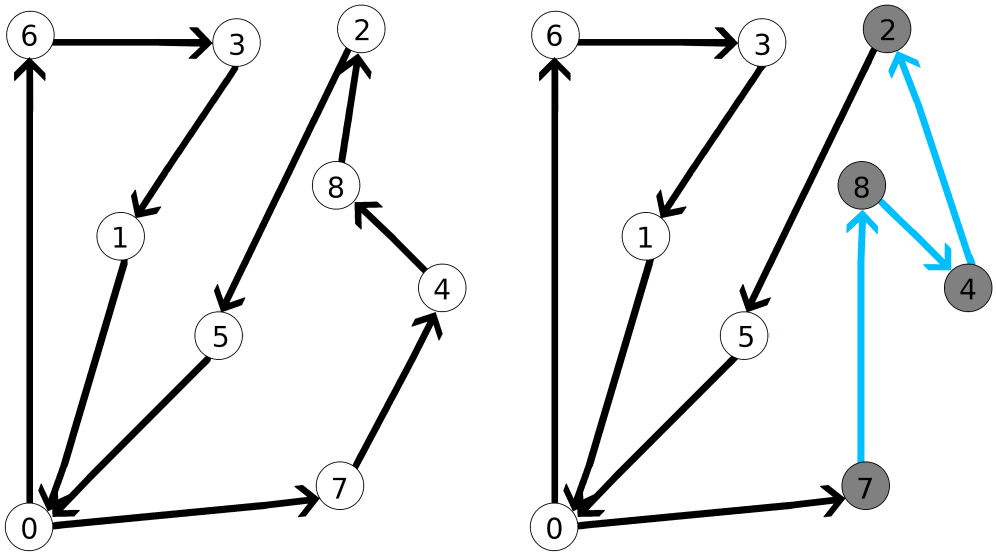

Figura 4.5: Movimento swap intra rota

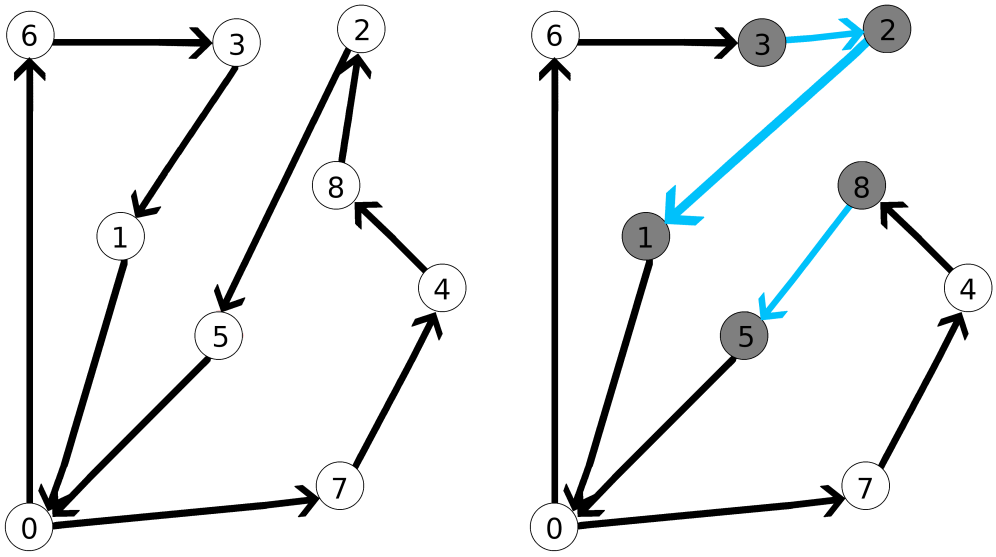

Figura 4.6: Movimento shift inter rotas

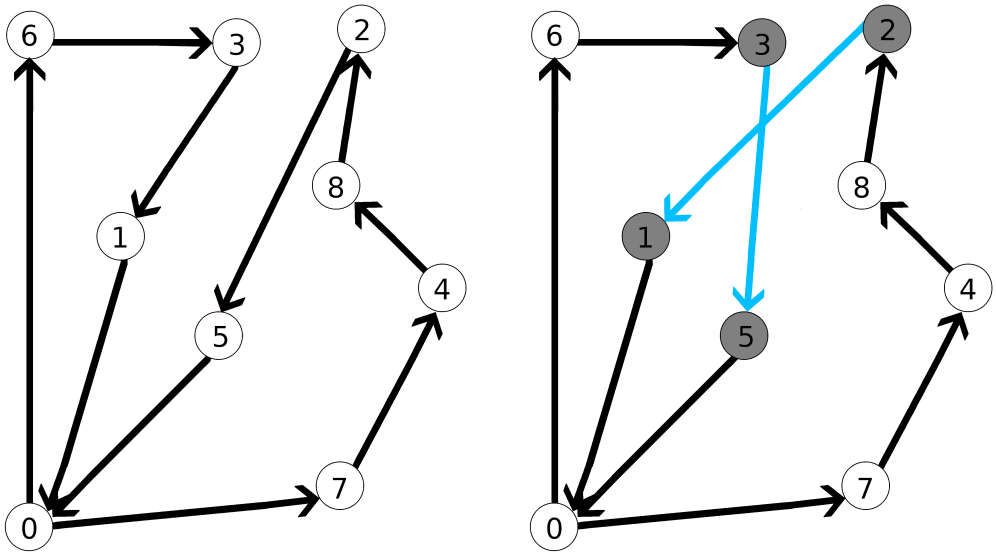

Figura 4.7: Movimento swap inter rotas

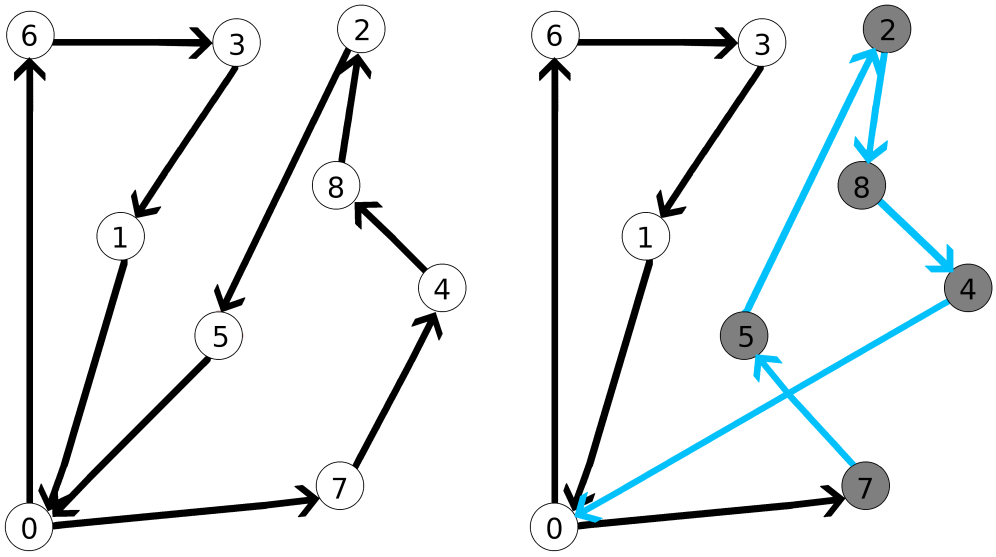

Figura 4.8: Movimento 2-opt intra rota

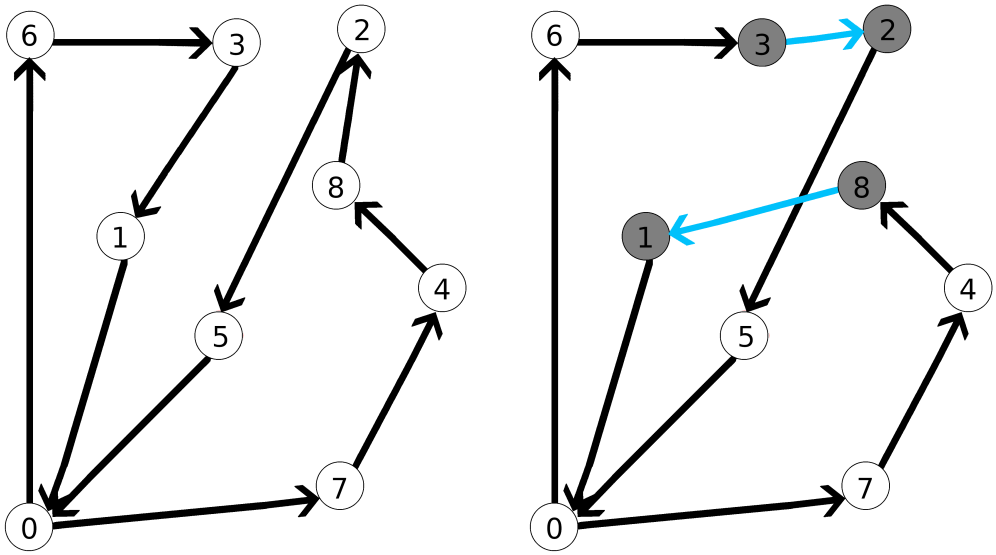

Figura 4.9: Movimento 2-opt inter rotas

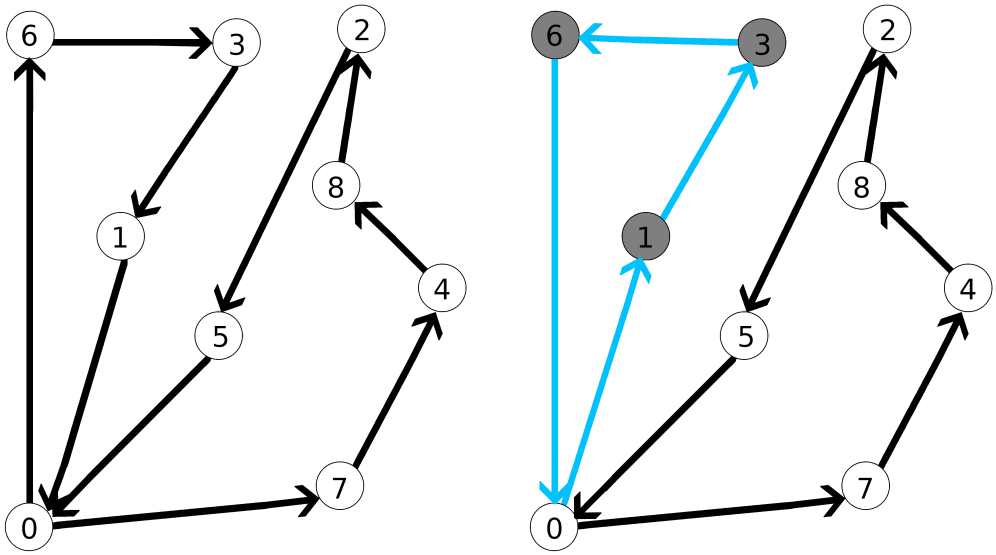

Figura 4.10: Movimento Inverte Rota

A heurística descrita na Subseção 4.1 pode gerar soluções inviáveis. Considerando $\overline{C}$ o conjunto dos clientes que estão associados ao veículo fictício e uma taxa de penalização ρ, a função objetivo considerada inclui uma parcela $|\overline{C}| \times \rho$ para penalização da inviabilidade na função objetivo. A primeira parte da equação, $Tx^{CO_2} \times q$, consiste no computo do CO_2 emitido pelos veículos da solução. O ρ foi definido de forma empírica para cada grupo de instância como o valor médio do CO_2 utilizando o método construtivo.

$$\text{Minimizar} \qquad F = Tx^{CO_2} \times q + |\overline{C}| \times \rho \tag{4.23}$$

Na tentativa de reduzir a inviabilidade da solução, quando esta ocorre, o RVND fixa os movimentos *shift* e *swap* inter rotas como movimentos iniciais, visando a remoção dos clientes da rota do veículo fictício. A ordem de execução dos demais movimentos é definida de maneira aleatória. Se não houver nenhum cliente no veículo fictício, a ordenação é totalmente aleatória para todas as vizinhanças consideradas.

A cada rota gerada pelos movimentos, é verificado se ela está na memória de rotas da Subseção 4.3. Se a rota encontrada tem diferença na ordem de visitação dos clientes, porque as rotas que passam pelo Mip são conectadas, ainda sim são utilizadas.

4.5 Grasp Reativo

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma metaheurística que combina construção e busca local num processo iterativo [Talbi, 2009]. A versão utilizada neste trabalho é baseada no trabalho [Prais and Ribeiro, 2000], onde o grau de aleatoriedade da heurística de construção é ajustado durante a execução do algoritmo, denominada GRASP Reativo.

Nesta abordagem um conjunto de valores de α são considerados durante a execução do algoritmo. Inicialmente, todos os valores têm a mesma probabilidade de serem escolhidos. Mas, a cada bloco de iterações, estas probabilidades são ajustadas de forma a favorecer a escolha dos valores de α responsáveis pelos melhores resultados.

Em cada iteração, uma solução é construída utilizando a estratégia apresentada na Seção 4.1. Inicialmente o critério de ordenação da Lista de Candidatos, parâmetro da Heurística Construtiva, é a poluição resultante da inserção do cliente da rota. Mas, quando o número de iterações sem melhora supera 150 iterações inicia-se uma fase de alternância no critério de ordenação da lista de candidatos. A cada bloco de 150 iterações o critério começa uma alternância entre o critério apresentado inicialmente e a distância percorrida.

A fim de refinar a solução obtida pela estratégia de construção, esta é refinada usando o RVND descrito na Seção 4.4.

O Algoritmo 4.7 mostra a implementação do GRASP, a linha 3 inicia a probabilidade de cada alfa como $\frac{1}{|listaAlfas|}$, a linha 9 o alfa é escolhido, a linha 10 a solução é

gerada pela heurística da Subseção 4.1 e na linha 11 a busca local realizada pelo RVND da Subseção 4.4. A linha 8 a solução *best* é atualizada se sol for melhor que a *best* e a linha 17 a probabilidade é atualizada em um intervalo de iterações. O critério da heurística de construção é atualizado na linha 19.

Algoritmo 4.7: Grasp reativo

```
1   input: instancia, listaAlfas, numIteracoes, intervaloAtualizacaoProb
2   output: solucao
3   prob ← iniciaProb(listaAlfas);
4   best ← ∅;
5   i ← 0;
6   criterio ← inicializaCriterio();
7   ultimaAtual ← −1;
8   while(i < numIteracoes)
9       alfa ← escolheAlfa(prob, listaAlfas);
10      sol ← gerasolucao(instancia, alfa, criterio);
11      rvnd(instancia, sol);
12      if(sol < best)
13          best ← sol;
14          ultimaAtual ← i;
15      end if
16      if(i mod intervaloAtualizacaoProb) = 0)
17          atualizaProb(prob, listaAlfas);
18      end if
19      criterio ← atualizaCriterio(criterio, i, numIteracoes, ultimaAtual);
20      i ← i + 1;
21  end while
22  return best;
```

4.6 ILS

Segundo [Talbi, 2009] a metaheurística ILS(*Iterated local search*) consiste em gerar uma perturbação em uma solução inicial e a aplicação de uma busca local. O GRASP da Subseção 4.7 é utilizado como solução inicial. O movimento 2 opt-swap inter rotas apresentado na Subseção 4.4 é utilizado como pertubação, a memória de rotas não é utilizada com a pertubação. A busca local utilizada é o RVND da Subseção 4.4 retirando o movimento 2 opt-swap inter rotas, para evitar ciclos.

Os modelos da Subseção 4.2 são utilizados a cada x iterações do ILS. Uma lista com máximo de x soluções é criada, para uma solução entrar na lista é necessário que uma das regras seja verdadeira: a solução está pelo menos a 10% da melhor solução ou se está entre 10% e 15% possui 40% de chance de ser inserida na lista. Uma solução da lista é escolhida de forma aleatória, todas as rotas são verificadas se foram geradas pelos modelos, senão passam pelo modelo com uma rota 4.2. Para o modelo que utiliza duas rotas, um limite de chamadas é definido e as rotas são escolhidas de forma aleatória. Só

pode existir repetição se uma das duas rotas for atualizada.

O Algoritmo 4.8 recebe como parâmetros: instância, número de iterações, número de iterações sem melhora, primeira chamada mip, número de iterações Grasp, intervalo mip (bloco de iterações para chamada do mip) e ϵ (o *gap* de 0.01 para solução corrente ser considerada igual a *best*). A linha 3 a solução inicial é gerada pelo Grasp da Subseção 4.5, a linha 10 o mip é chamado para a solução *best* quando o número de interações chega na primeira chamada do mip.

A linha 13 a solução corrente é perturbada pelo movimento 2 opt-swap seguido do RVND' (não utiliza o movimento de perturbação). A linha 16 verifica a iteração atual é maior que a iteração para começar o mip. A função *atualizaLista* utiliza o *gap* para incluir a solução corrente se a mesma estiver suficientemente próxima da melhor solução. O fluxo de execução atinge o bloco do *if* da linha 18 se a iteração atual for múltiplo de *intervaloMip*, a linha 19 escolhe uma solução aleatoriamente da lista de soluções ou retorna a solução corrente se o *gap* é menor que ϵ. O *if* da linha 21 atualiza a solução ótima se a solução resultante do mip for melhor que *best*. A linha retorna a melhor solução se o número de interações sem melhora é maior do que o parâmetro número de iterações sem melhora, caso esse ponto não seja alcançado a linha 31 retorna a melhor solução.

Algoritmo 4.8: ILS

```
1   input: instancia, numIteracoes, itSemMelhora, itMip, itGRASP, intervaloMip, ϵ
2   output: solucao
3   best ← GRASP(instancia, itGRASP);
4   i ← 0; ultimaAtualizacao ← 0;
5   solCorrente ← best;
6   memoriaRotas ← ∅; listaSolucoes ← ∅;
7   while(i < numIteracoe)
8       if(i == itMip)
9           poluicao ← getPoluicao(best);
10          MIP(best, memoriaRotas);
11          atualizaSolucaoCorrente(best, poluicao, solCorrente, i, ultimaAtualizacao);
12      end if
13      solCorrente ← pertubacao(instancia, solCorrente);
14      rvnd'(instancia, solCorrente, memoriaRotas);
15      gap ← calculaGap(solCorrente, best);
16      if(i > itMip)
17          atualizaLista(listaSolucoes, solucaoCorrente, gap);
18          if((i%intervaloMip) = 0)
19              sol ← escolheSolucao(listaSolucoes, solCorrente, gap, ϵ);
20              MIP(sol, memoriaRotas);
21              if(atualizaSolucao(sol, best))
22                  best ← sol; solucaoCorrente ← sol; ultimaIt ← i;
23          end if
24      end if
25      if(atualizaSolucao(solucaoCorrente, best))
26          best ← solucaoCorrente; ultimaIt ← i;
27      if(verificaItSemMelhora(i, itMip, ultimaIt, itSemMelhora))
28          return best;
29      i++;
30  end while
31  return best;
```

5 Experimentos Computacionais

Nesse capítulo, os experimentos computacionais são descritos e os resultados são analisados. As características das instâncias são apresentadas na seção 5.1, na seção 5.2 os ajustes de parâmetros dos algoritmos são detalhados, e na seção 5.4 os resultados são comparados com a literatura.

5.1 Descrição das instâncias

Para avaliar as abordagens propostas, foram utilizadas instâncias clássicas para problemas de roteamento verde, que consideram distâncias reais entre cidades escolhidas aleatoriamente do Reino Unido, disponíveis em [Demir et al., 2012] e que foram modificadas e disponibilizadas por [Xiao and Konak, 2016].

As instâncias são divididas em sete grupos conforme o número de clientes, a saber: 10, 15, 20, 25, 50, 75 e 100 clientes. Cada grupo consiste em 20 instâncias. O horizonte de planejamento é de nove horas e foi dividido em cinco períodos de 1,8 horas. A frota apresenta dois tipos de veículos, que diferem entre si pela capacidade máxima de carga, capacidade de combustível e início e fim da janela de tempo. Pelo fato de se utilizar os clientes como cidades e distâncias reais, a matriz com as distâncias é assimétrica. E uma vez que pode não existir um caminho direto entre duas cidades, considerou-se, neste caso, distância igual a infinito.

5.2 Ajuste de parâmetros

O ajuste de parâmetros dos métodos apresentados no Capítulo 4 foi realizado através do pacote *Irace* [López-Ibáñez et al., 2016]. Para tal, os seguintes parâmetros e seus respectivos conjuntos de valores foram submetidos ao software;

- GRASP:

  numIteracoes - número máximo de iterações: $\{200, 500, 600\}$.

- ILS:

  numIteracoes - número de iterações: $\{2000, 2500, 3000, 3500\}$;

  itSemMelhora - número de iterações sem melhoras: $\{300, 500, 700, 900\}$;

  itMip - primeira chamada mip: $\{500, 1000, 1400\}$;

  intervaloMip - intervalo para chamada mip: $\{50, 100, 200, 400\}$;

  itMip2Rotas - número de chamadas para gerar duas rotas: $\{2, 4, 7, 10\}$;

Configurou-se o Irace para fazer um total de 1000 execuções considerando um conjunto de 30 instâncias selecionadas do conjunto de instâncias do *benchmark* do experimento. A melhor configuração de parâmetros indicada pelo Irace foi:

- número de iterações Grasp: 500;
- número de iterações ILS: 2500;
- número de iterações sem melhora ILS: 500;
- primeira chamada mip: 500;
- intervalo para chamada mip: 50;
- número de chamadas para gerar duas rotas: 7.

5.3 Ambiente de teste e configuração do experimento

O algoritmo GRASP_ILS foi desenvolvido na linguagem *C++*, enquanto o modelo matemático foi implementado usando o resolvedor Gurobi (versão 9) [Gurobi Optimization, 2021]. Os testes foram executados em uma máquina Intel i5-8250U com frequência base de 1,6 Ghz (*tubo* de 3,4 Ghz), 8GB de memória RAM, Sistema Operacional Debian 11 com kernel linux na versão 5.10.0-2amd64. Além disso, utilizou-se o compilador g++ na versão 10.2.1 e Gurobi na versão 9. [Xiao and Konak, 2016] utilizam uma máquina Intel i5-2400S com frequência base de 2,5 Ghz (*tubo* de 3,3 Ghz), resolvedor CPLEX (versão 12.4.0.1) e linguagem de programação AMPL. As informações sobre sistema operacional, memória RAM e compilador não foram fornecidos pelos autores.

Tabela 5.1: Comparação entre os processadores com o *benchmark PassMark*

Métricas (MOps/S)	**i5-8250U**	**i5-2400S**	Razão
Operações de números inteiros	21.364	11.126	1,9
Operações de ponto flutuante	13.066	5.745	2,3
Single Thread	1.948	1.468	1,3

Na Tabela 5.1 são apresentadas métricas do *benchmark PassMark* para comparar as velocidades dos processadores. Utilizando o pior caso, operações de ponto flutuante, o processador utilizado nos testes desse trabalho é 2,3 vezes mais rápido do que o processador utilizado por [Xiao and Konak, 2016]. Essa informação permite uma comparação mais precisa dos tempos de processamento.

Cada instância foi executada 30 vezes. Para obter o resultado por grupo de instâncias, o seguinte procedimento foi modificado de [Xiao and Konak, 2016]: cada instância é executada um número λ de vezes e calculam-se as seguintes métricas: médias de tempo e de total de emissão de poluição, o menor tempo e o menor valor de emissão de poluição para cada instância. Os resultados por grupo são obtidos pela média de cada uma destas métricas considerando todas as 10 instâncias do grupo. A diferença do experimento apresentado em [Xiao and Konak, 2016] é que os autores usaram $\lambda = 5$ nas as instâncias dos grupos de 10 a 25 clientes e $\lambda = 1$ para os grupos de 50 a 100 clientes. Além disso, apenas o tempo total médio para a obtenção das soluções do grupo é apresentado, faltando o tempo médio das melhores soluções obtidas nas 20 instâncias do grupo. Destaca-se que já foi provado por [Xiao and Konak, 2016], a partir do uso de resolvedor do modelo matemático, que a instância UK_15x5_11 é inviável. Para todas as demais instâncias existem soluções viáveis.

5.4 Análise dos resultados

A Tabela 5.2 mostra uma comparação entre os resultados obtidos pelo algoritmo proposto, coluna ILS_GRAS_MIP, com aqueles apresentados em [Xiao and Konak, 2016] (coluna P-MIP-INS) e em [Rylan et al., 2017] (coluna Raylan 2017) em relação ao melhor resultado obtido para o valor da função objetivo (coluna Poluição). A coluna GAP apresenta a diferença percentual em relação ao melhor resultado dentre as três abordagens. Os

melhores resultados são destacados em negrito.

Pode-se observar que o algoritmo ILS proposto nesse trabalho e o algoritmo P-MIP-INS apresentaram resultados muito próximos para todas as instâncias, diferindo, na média dos GAPs, em apenas $0,1\%$ para o conjunto de instâncias, ao passo que abordagem de [Rylan et al., 2017] não se mostrou eficaz, tendo os piores resultados para seis dos sete grupos de instâncias testadas.

Tabela 5.2: Média dos melhores resultados

		P-MIP-INS		Raylan 2017		ILS_GRASP_MIP	
Grupo	Melhor	Poluição	GAP	Poluição	GAP	Poluição	GAP
10	**557,9**	**557,9**	**0,0%**	597,9	7,2%	561,0	0,5%
15	**705,0**	746,1	5,8%	**705,0**	**0,0%**	745,9	5,8%
20	**954,3**	958,7	0,5%	1078,1	13,0%	**954,3**	**0,0%**
25	**936,0**	**936,0**	**0,0%**	1054,3	12,6%	937,7	0,2%
50	**1698,9**	1710,2	0,7%	2066,0	21,6%	**1699,0**	**0,0%**
75	**2483,5**	**2483,5**	**0,0%**	2932,3	18,1%	2487,2	0,2%
100	**3149,8**	**3149,8**	**0,0%**	3748,5	19,0%	3182,2	1,0%
Média	-	-	**1,0%**	-	13,1%	-	1,1%

A principal contribuição das abordagens propostas neste trabalho é o fato de combinar um método baseado no modelo matemático com metaheurísticas sem comprometer a eficiência de tempo do algoritmo proposto. A Tabela 5.3 apresenta os tempos de processamento, em minutos, apenas para os algoritmos P-MIP-INS e ILS_GRASP_MIP, já que os resultados apresentados em [Rylan et al., 2017] não trazem os tempos de processamento. Os resultados para o algoritmo P-MIP-INS foram retirados de [Xiao and Konak, 2016], sendo que, como já dito, para os grupo de até 25 clientes, os autores apresentam a média dos melhores resultados para apenas cinco execuções por instância. Já para os grupos de 50, 75 e 100 clientes, os resultados referem-se à média do grupo considerando uma execução por instância.

Os tempos de processamento da literatura foram convertidos usando a razão de 2,3 da tabela 5.1. Ou seja, para se obter uma estimativa de tempo de execução do P-MIP-INS no processador utilizado nesse trabalho, o tempo do mesmo foi dividido por 2,3.

Pode-se observar que o algoritmo ILS proposto apresenta desempenho 2,5 vezes melhor que a melhor abordagem da literatura, isso com um GAP de apenas $0,1\%$ acima em relação à qualidade das soluções apresentadas pelos dois algoritmos.

Tabela 5.3: Média dos melhores resultados tempo

Grupos	P-MIP-INS(Min)	P-MIP-INS Conversão(Min)	ILS_GRASP_MIP(Min)	*Speedup*
10	7,7	3,3	**0,9**	**3,7**
15	12,9	5,6	**2,4**	**2,3**
20	18,0	7,8	**5,2**	**1,5**
25	31,1	13,5	**9,2**	**1,5**
50	59,0	25,6	**11,0**	**2,3**
75	89,0	38,7	**11,2**	**3,5**
100	98,0	42,6	**15,0**	**2,8**
Média	45,1	19,6	**7,8**	**2,5**

Tabela 5.4: Média dos resultados para 30 execuções

Grupos	P-MIP-INS			ILS_GRASP_MIP			
	Poluição	Tempo(Min)	Tempo Conv.(Min)	Poluição	GAP	Tempo(Min)	*Speedup*
10	**558,6**	7,7	3,3	561,3	0,5%	**0,9**	**3,7**
15	770,9	12,9	5,6	**748,9**	**-2,8%**	**2,2**	**2,5**
20	971,2	18,0	7,8	**969,9**	**-0,1%**	**6,5**	**1,2**
25	**952,3**	31,1	13,5	959,2	0,7%	**6,8**	**2,0**
50	1710,2	59,0	25,6	1764,4	3,2%	**9,8**	**2,6**
75	2483,5	89,0	38,7	2598,3	4,6%	**9,3**	**4,2**
100	3149,8	98,0	42,6	3312,9	5,2%	**11,6**	**3,7**
Média	1513,8	45,1	19,6	1559,3	0,02	**6,7**	**3,0**

Ao se considerar o tempo total médio dos algoritmos (não apenas os tempos das melhores execuções), a Tabela 5.4 apresenta uma comparação das médias dos resultados da abordagem híbrida proposta com os resultados do P-MIP-INS quanto ao valor da função objetivo (poluição) e ao tempo em minutos. A coluna GAP apresenta a diferença percentual do resultado da abordagem proposta em relação apresentado pelo algoritmo P-MIP-INS. Convém destacar que, para os grupos com 50, 75 e 100 clientes, a comparação não é precisa, dado que P-MIP-INS só foi executado uma vez para cada instância nesses grupos.

A Tabela 5.4 mostra que, mesmo considerando todas as execuções do experimento (não apenas aquelas que levaram aos melhores resultados por instância), a abordagem proposta apresenta um comportamento que valida o objetivo pretendido, já que apresenta uma diferença percentual de apenas $0,02\%$ com um *Speedup* de 3. Ou seja, o uso de um método exato, que em geral demanda tempos excessivos até obter boas soluções, quando combinado com metaheurísticas garantiu a eficácia com uma melhor eficiência de tempo.

6 Conclusão e Trabalhos Futuros

Esse trabalho propôs um método híbrido que combina as técnicas de Inteligência Computacional GRASP, ILS e RVND com um algoritmo de *local branching* para resolver o problema de roteamento verde de veículos com frota heterogênea e variação de velocidade ao longo do tempo conforme as condições de tráfego, em que são permitidas paradas em qualquer ponto ao longo da rota dos veículos.

Para resolver o problema, três métodos foram propostos: uma heurística construtiva que considera somente paradas nos clientes, e dois métodos exatos que consideram as paradas em qualquer lugar de um arco, e que permitem a troca de clientes. Para inicializar a solução do problema pela abordagem ILS, utilizou-se um algoritmo GRASP que emprega, na fase de construção, uma heurística de inserção mais barata, e, na fase de busca local, um método RVND.

Na definição do algoritmo RVND, foram utilizados movimentos de *shifit*, *swap*, *2-opt* inter e intra rotas, além de movimentos que permitem inverter o sentido de uma rota, bem como o movimento que permite a troca dos tipos veículos entre duas rotas. Por outro lado, no ILS proposto utilizou-se o movimento 2-opt inter rotas como pertubação, combinado com o RVND sem este movimento, além de chamadas periódicas dos métodos exatos.

Utilizando o modelo matemático do problema, um método do tipo *local branching* foi utilizado para encontrar novas soluções viáveis em torno de uma rota de referência, sendo que, o espaço de busca é limitado a um subconjunto das possíveis sequências. Esse método foi estendido para permitir a utilização de duas rotas, permitindo a troca de clientes entre as rotas, assim, aumentando o espaço de busca.

Os algoritmos propostos foram submetidos a um conjunto de instâncias da literatura e foi possível obter soluções competitivas sem contudo depreciar a eficiência. Os resultados mostraram que a abordagem proposta consegue resultados similares aos apresentados na literatura, porém com ganho expressivo em termos de tempo de processamento, chegando a tempos três vezes inferiores mesmo considerando as diferenças dos

processadores no pior caso.

Como trabalhos futuros, uma vez que os algoritmos propostos apresentam eficiência de tempo de processamento expressiva, sugere-se explorar outros movimentos como pertubação no ILS. Isto porque a pertubação com o movimento *2-opt* inter rotas implementada considera sequencialmente as rotas e os clientes após o início. Essas escolhas podem ser feitas sempre de forma aleatória.

Além disso, movimentos mais simples, onde seja priorizada a escolha de duas rotas com veículos diferentes no modelo para gerar duas rotas pode ser interessante. Já que existem diferenças entre os veículos em relação à taxa de emissão de combustível conforme o arco e o período, priorizar a escolha de veículos específicos pode levar a uma menor emissão de CO_2. Os modelos também podem ser modificados para permitirem a troca de tipos de veículos distintos, assim, aumentando o espaço de busca do *local branching*. A penalização da função objetivo, o ρ pode ser definido de forma adaptativa para cada execução.

A Constantes e variáveis do modelo

Tabela A.1: Parâmetros do modelo

Parâmetros	Descrição
V	Conjunto de clientes e depósito
V^+	Conjunto de clientes (excluindo o depósito)
A	Conjunto de arestas do subgrafo induzido por V
i, j	Índices dos clientes
M	Conjunto dos períodos
k, m	Índices dos períodos, sendo que m representa o último período
W	Conjunto de veículos
v	Índice do veículo
$D_{i,j}$	Distância entre aresta (i, j)
$Dem_{\,i}$	Demanda do cliente i
$TS_{\,i}$	Tempo de serviço do cliente i
$[T_i^{Ini}, T_i^{Fim}]$	Janela de tempo do cliente i
$[T_k^{Ini}, T_k^{Fim}]$	Início e fim do período k
$[T_v^{Ini},\ T_v^{Fim}]$	Janela de tempo do veículo v
$Vel_{\,ij}^{\,k}$	Velocidade do veículo do arco (i, j) no período k
$Cb_{\,v}^{\,Max}$	Capacidade máxima do tanque de combustível do veículo v
$Cb_{\,ij}^{\,k,v}$	Taxa de consumo de combustível ao percorrer o arco (i, j) no período k
Cb_v^+	Consumo adicional de combustível para uma unidade a mais de carga
$Tx_v^{CO_2}$	Taxa de conversão combustível / CO_2

Tabela A.2: Variáveis do modelo

Variáveis	Descrição
X_{ij}	Indica se o arco (i, j) é percorrido ou não
x_{ij}^v	Indica se o veículo v percorre o arco (i, j) ou não
x_{ij}^{kv}	Indica se o veículo v percorre o arco (i, j) em um período k
d_{ij}^{kv}	Indica a distância percorrida do arco (i, j) no período k pelo veículo v
τ_{ij}^{kv}	Indica o tempo gasto no arco (i, j) no período k pelo veículo v
t_i^{sai}	Indica o instante de tempo em que o veículo sai do cliente i
t_i^{chg}	Indica o instante de tempo em que o veículo chega no cliente i
f_{ij}^v	Indica a carga do veículo v ao percorrer o arco (i, j)
q_{comb}^v	Indica a quantidade total de combustível utilizada na rota do veículo v

B Constantes e variáveis do modelo matemático para uma rota

Tabela B.1: Parâmetros do modelo matemático para uma rota r

Parâmetros	Descrição
V_r	Conjunto de pontos (depósito e clientes) da rota r
V_r^+	Conjunto de clientes da rota r (excluindo o depósito)
A_r	Conjunto de arestas do subgrafo induzido por V_r
A_r^*	Subconjunto de arestas de A_r composto pelas arestas originais da rota r
M	Conjunto dos períodos
i, j	Índices dos clientes
k, m	Índices dos períodos, sendo que m representa o último período
$D_{i,j}$	Distância entre aresta (i, j)
$Dem_{\,i}$	Demanda do cliente i
$TS_{\,i}$	Tempo de serviço do cliente i
$[T_i^{Ini}, T_i^{Fim}]$	Janela de tempo do cliente i
$[T_k^{Ini}, T_k^{Fim}]$	Início e fim do período k
$[T_v^{Ini},\ T_v^{Fim}]$	Janela de tempo do veículo
$Vel_{\,ij}^{\,k}$	Velocidade do veículo do arco (i, j) no período k
$Cb^{\,Max}$	Capacidade máxima do tanque de combustível do veículo
v	Tipo de veículo v associado a rota r.
$Cb_{\,ij}^{\,k,v}$	Taxa de consumo de combustível ao percorrer o arco (i, j) no período k
Cb_v^+	Consumo adicional de combustível para uma unidade a mais de carga
$Tx_v^{CO_2}$	Taxa de conversão combustível / CO_2

Tabela B.2: Variáveis do modelo matemático que otimiza uma rota r

Variáveis	Descrição
x_{ij}	Indica se o arco (i, j) é percorrido ou não
x_{ij}^k	Indica se um arco (i, j) é percorrido em um período k
d_{ij}^k	Indica a distância percorrida do arco (i, j) no período k
τ_{ij}^k	Indica o tempo gasto no arco (i, j) no período k
t_i^{sai}	Indica o instante de tempo em que o veículo sai do cliente i
t_i^{chg}	Indica o instante de tempo em que o veículo chega no cliente i
f_{ij}	Indica a carga do veículo ao percorrer o arco (i, j)
q_{comb}	Indica a quantidade total de combustível utilizada na rota.

Bibliografia

B.-H. Ahn and J.-Y. Shin. Vehicle-routeing with time windows and time-varying congestion. *Journal of the Operational Research Society*, 42(5):393–400, 1991.

D. Bernstein. O. yigit. hash functions. `http://www.cse.yorku.ca/~oz/hash.html`, 1991. Accessed: 2021-06-07.

E. Demir, T. Bektaş, and G. Laporte. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2): 346–359, 2012.

E. W. Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

M. Fischetti and A. Lodi. Local branching. *Mathematical programming*, 98(1):23–47, 2003.

R. A. Foroutan, J. Rezaeian, and I. Mahdavi. Green vehicle routing and scheduling problem with heterogeneous fleet including reverse logistics in the form of collecting returned goods. *Applied Soft Computing*, 94:106462, 2020.

P. Forster, V. Ramaswamy, P. Artaxo, T. Berntsen, R. Betts, D. Fahey, J. Haywood, J. Lean, D. Lowe, G. Myhre, J. Nganga, R. Prinn, G. Raga, M. Schulz, R. Dorland, G. Bodeker, O. Boucher, W. Collins, T. Conway, and T. Whorf. *Changes in Atmospheric Constituents and in Radiative Forcing*, pages 135, 137. 10 2007.

L. Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL `http://www.gurobi.com`.

A. V. Hill and W. C. Benton. Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *Journal of the Operational Research Society*, 43(4):343–351, 1992.

İ. Küçükoğlu, S. Ene, A. Aksoy, and N. Öztürk. A memory structure adapted simulated annealing algorithm for a green vehicle routing problem. *Environmental Science and Pollution Research*, 22(5):3279–3297, 2015.

H. Le Treut, R. Somerville, U. Cubasch, Y. Ding, C. Mauritzen, A. Mokssit, T. Peterson, and M. Prather. Historical overview of climate change science chapter 1. 2007.

J. Lu, Y. Chen, J.-K. Hao, and R. He. The time-dependent electric vehicle routing problem: model and solution. *Expert Systems with Applications*, 161:113593, 2020.

M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002.

G. Macrina, L. D. P. Pugliese, and F. Guerriero. The green-vehicle routing problem: A survey. *Modeling and Optimization in Green Logistics*, 2020.

T. Paskannaya and G. Shaban. Innovations in green logistics in smart cities: USA and EU experience. , 1:173–181, 2019.

M. Prais and C. C. Ribeiro. Parameter variation in grasp procedures. *Investigación Operativa*, 9(1):1–20, 2000.

M. RAYLAN. Problema de roteamento de veículos voltado para redução de emissões de carbono. Master's thesis, UFF, Universidade Federal Fluminense, Niterói, Fevereiro 2018.

N. Rezaei, S. Ebrahimnejad, A. Moosavi, and A. Nikfarjam. A green vehicle routing problem with time windows considering the heterogeneous fleet of vehicles: two metaheuristic algorithms. *European Journal of Industrial Engineering*, 13(4):507–535, 2019.

M. Rylan, Y. Abitbol, and L. Satoru. Problema de roteamento de veículos voltado para redução de emissões de carbono. In *XLIX SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL*, pages 1037–1041. SOBRAPO, 2017.

L. Shen, F. Tao, and S. Wang. Multi-depot open vehicle routing problem with time windows based on carbon trading. *International journal of environmental research and public health*, 15(9):2025, 2018.

S. Solaymani. Co2 emissions patterns in 7 top carbon emitter economies: The case of transport sector. *Energy*, 168:989–1001, 2019.

E. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley, 2009. ISBN 9780470496909. URL `https://books.google.com.br/books?id=SIsa6zi5XV8C`.

Y. Wang, K. Assogba, J. Fan, M. Xu, Y. Liu, and H. Wang. Multi-depot green vehicle routing problem with shared transportation resource: Integration of time-dependent speed and piecewise penalty cost. *Journal of Cleaner Production*, 232:12–29, 2019.

Y. Xiao and A. Konak. The heterogeneous green vehicle routing and scheduling problem with time-varying traffic congestion. *Transportation Research Part E: Logistics and Transportation Review*, 88:146–166, 2016.

D. Ye, Z. Wanhong, L. Hongwei, and Z. Yonghui. Multi-type ant system algorithm for the time dependent vehicle routing problem with time windows. *Journal of Systems Engineering and Electronics*, 29(3):625–638, 2018. doi: 10.21629/JSEE.2018.03.20.

Z. Zhao, X. Li, and X. Zhou. Optimization of transportation routing problem for fresh food in time-varying road network: Considering both food safety reliability and temperature control. *PloS one*, 15(7):e0235950, 2020.