Ministério da Educação
Universidade Federal de Ouro Preto
Escola de Minas
Departamento de Engenharia de Produção, Administração e Economia

Milena Fernandes França

Aplicação da Meta-Heurística ILS no planejamento do transporte escolar intermunicipal na cidade de Itabirito - MG

Ouro Preto
2023

Milena Fernandes França

Aplicação da Meta-Heurística ILS no planejamento do transporte escolar intermunicipal na cidade de Itabirito – MG

Monografia apresentada ao Curso de Engenharia de Produção da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Produção

Orientador: Prof. Dr. Aloisio de Castro Gomes Junior

Ouro Preto
2023

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA DE PRODUCAO, ADMINISTRACAO E ECON

FOLHA DE APROVAÇÃO

Milena Fernandes França

Aplicação da Meta-heurística ILS no planejamento do transporte escolar intermunicipal na cidade de Itabirito-MG

Monografia apresentada ao Curso de Engenharia de Produção da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Produção.

Aprovada em 22 de agosto de 2023

Membros da banca

[Doutor] - Aloísio de Castro Gomes Júnior - Orientador(a) (Universidade Federal de Ouro Preto)
[Doutor] - Helton Cristiano Gomes- (Universidade Federal de Ouro Preto)
[Doutora] - Irce Fernandes Gomes Guimarães - (Universidade Federal de Ouro Preto)
[Graduada] - Larissa Aparecida Lopes de Souza - (Universidade Federal de Ouro Preto)

O Prof. Aloísio de Castro Gomes Júnior, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 28/08/2023.

Documento assinado eletronicamente por **Aloisio de Castro Gomes Junior**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 28/08/2023, às 16:14, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0582111** e o código CRC **418FB012**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591540 - www.ufop.br

Resumo

A Prefeitura de Itabirito disponibiliza transporte gratuito, nos turnos da manhã e noite, para os residentes da cidade que estão matriculados em instituições de ensino localizadas em cidades vizinhas, com destaque para Belo Horizonte, destino que será tratado neste trabalho. O principal desafio enfrentado pela Secretaria Municipal de Transportes é a definição das rotas, levando em consideração as restrições que envolvem instituições de ensino com demandas que excedem a capacidade dos veículos disponíveis e as limitações relacionadas ao tempo de chegada dos alunos devido à grade de horários das aulas. Essas restrições classificam o problema como um "Problema de Roteamento de Veículos com Entrega Fracionada e Janela de Tempo" (PRVEFJT). Dada a complexidade desse problema, que é classificado como NP-difícil, este estudo propõe abordagens para resolver o PRVEFJT por meio da meta-heurística denominada "*Iterated Local Search*" (ILS). A avaliação do desempenho do algoritmo é conduzida através do uso de cinco pequenas instâncias geradas de maneira aleatória, com auxílio da biblioteca "*random*" da linguagem de programação Python. São comparados os resultados do algoritmo ILS e do método exato para essas instâncias. A análise do "gap" entre as soluções ótimas e as soluções geradas pelo algoritmo ILS revela que o algoritmo demonstra competitividade em relação ao método exato, exibindo resultados que se aproximam das soluções ideais, ao mesmo tempo em que mantém um baixo tempo de execução. Com base nesses resultados, o algoritmo é aplicado à situação real do problema, gerando soluções que consistem em 15 rotas no período da manhã (totalizando 1910 km percorridos) e 23 rotas no período noturno (totalizando 2822 km percorridos).

Palavras-chave: Transporte escolar, otimização, problema de roteamento de veículos, *Iterated Local Search*.

Abstract

The Itabirito City Hall provides free transportation in the morning and evening shifts for residents of the city who are enrolled in educational institutions located in neighboring cities, with a focus on Belo Horizonte, a destination that will be addressed in this study. The main challenge faced by the Municipal Department of Transportation is the definition of routes, considering the constraints involving educational institutions with demands that exceed the capacity of the available vehicles, as well as limitations related to the students' arrival time due to the class schedule. These constraints categorize the problem as a "Vehicle Routing Problem with Split Deliveries and Time Windows" (VRPSDTW). Given the complexity of this problem, which is classified as NP-hard, this study proposes approaches to solve the VRPSDTW using the metaheuristic called "Iterated Local Search" (ILS). The performance evaluation of the algorithm is conducted using five small instances generated randomly, with the assistance of the "random" library in the Python programming language. The results of the ILS algorithm and the exact method for these instances are compared. The analysis of the gap between the optimal solutions and the solutions generated by the ILS algorithm reveals that the algorithm demonstrates competitiveness compared to the exact method, yielding results that approach ideal solutions while maintaining a low execution time. Based on these results, the algorithm is applied to the real situation of the problem, generating solutions consisting of 15 routes in the morning period (totaling 1910 km traveled) and 23 routes in the evening period (totaling 2822 km traveled).

Keywords: School transportation, optimization, vehicle routing problem, Iterated Local Search.

Lista de figuras

Lista de tabelas

Sumário

1. Introdução

A busca por uma educação de qualidade e o acesso à educação superior têm se tornado cada vez mais relevantes para o desenvolvimento social e econômico das comunidades. Nesse contexto, o Brasil enfrenta desafios significativos na democratização do acesso e da permanência dos alunos nas instituições de ensino superior. O relatório "Desafios e Perspectivas da Educação Superior Brasileira para a Próxima Década" elaborado pela UNESCO (2012) levantou alguns desafios que a educação brasileira poderia enfrentar ao longo da década de 2010, sendo os mais relevantes relacionados ao ingresso de jovens nas universidades e à conclusão dos cursos. Esses obstáculos se dão principalmente pela falta de democratização do acesso e de políticas para permanência dos alunos nas instituições. Segundo o Censo da Educação Superior, emitido pelo Ministério da Educação (2022), o Plano Nacional de Educação, tem como uma de suas metas garantir que ao menos 33% dos jovens brasileiros de 18 a 24 anos estejam matriculados em uma instituição de ensino superior até o ano de 2024, porém, em 2020, essa taxa era de apenas 23,8%.

Reconhecendo a importância do acesso à educação superior, a Prefeitura Municipal de Itabirito oferece transporte gratuito aos residentes que estão matriculados em instituições de ensino superior localizadas nas cidades vizinhas. O principal destino dos estudantes é Belo Horizonte, situada a 58 km de distância da cidade. O serviço de transporte é essencial para viabilizar o acesso dos estudantes às universidades da capital, permitindo que eles busquem uma formação de qualidade.

Os esforços da prefeitura de Itabirito são refletidos nos dados fornecidos pela Secretaria de Transportes do município. No segundo semestre do ano de 2022, 556 alunos, distribuídos em 27 universidades diferentes, utilizaram o serviço de transporte oferecido apenas para Belo Horizonte. Esses números demonstram o impacto positivo do programa de transporte gratuito, que facilita o acesso à educação superior para muitos estudantes do município.

Entretanto, o desafio logístico envolvido na gestão do transporte escolar é significativo, considerando a quantidade de variáveis que afetam a roteirização dos veículos, como distância entre as localidades, horário das aulas e capacidade de transporte. Decisões inadequadas em um processo de definição das rotas podem acarretar em dificuldades operacionais e subutilização do sistema (GOLDBARG, E.; GOLDBARG, M. e LUNA, 2016). Nesse sentido, a aplicação de técnicas de otimização combinatória, como algoritmos heurísticos e meta-heurísticos, pode

auxiliar na tomada de decisões, visando a redução de custos e deslocamento, além de melhorar a qualidade de vida dos usuários do serviço por meio da diminuição do tempo de viagem.

O objetivo geral deste trabalho é aplicar métodos de resolução do problema de roteamento de veículos, mais especificamente, o algoritmo *Iterated Local Search*, no planejamento do transporte escolar intermunicipal da cidade de Itabirito, MG, com o propósito de tornar o processo decisório do departamento de transportes mais eficiente. Para alcançar esse objetivo, são estabelecidos os seguintes objetivos específicos: definir e apresentar algoritmos heurísticos e meta-heurísticos aplicáveis ao problema, bem como implementá-los, identificando os parâmetros mais adequados, a fim de encontrar rotas que minimizem o número de veículos utilizados e a distância total percorrida por turno. Com isso, busca-se contribuir para o aprimoramento do transporte gratuito oferecido aos estudantes e, consequentemente, para o incentivo ao acesso e permanência na educação superior, promovendo o desenvolvimento educacional e social da comunidade de Itabirito.

A justificativa para este trabalho surge da percepção da necessidade de aprimoramento da gestão do transporte escolar intermunicipal como forma de promover uma maior equidade no acesso à educação superior. Reconhecendo os desafios inerentes, como a limitação de recursos e a complexidade logística, que precisam ser superados para garantir que mais estudantes tenham a oportunidade de buscar uma formação acadêmica de qualidade. A adoção de técnicas de otimização, como a *Iterated Local Search*, se apresenta como uma abordagem simples, prática e eficaz para enfrentar essas barreiras, proporcionando soluções eficientes e impactando positivamente a vida dos estudantes e a sociedade como um todo. Portanto, este trabalho busca contribuir para a melhoria do transporte escolar na cidade de Itabirito, em busca de uma educação inclusiva e ao desenvolvimento sustentável do município.

O presente trabalho está estruturado da seguinte forma: o capitulo 2 apresenta uma revisão da literatura, com os principais aspectos sobre o Problema de Roteamento de Veículos, aprofundando no Problema de Roteamento de Veículos Escolares e Problema de Roteamento de Veículos com Entrega Fracionada e Janela de Tempo e seus respectivos métodos de solução, como as Heurísticas e Meta-Heurísticas. No capítulo 3 está a apresentação do problema estudado de maneira detalhada e as estratégias utilizadas para resolução, onde são apresentados os métodos para preparação dos dados e algoritmos propostos. O capítulo 4 apresenta e discute os resultados obtidos através do método apresentado no capítulo anterior. As considerações finais estão colocadas no capítulo 5.

2. Referencial teórico

2.1. Problema de Roteamento de Veículos Escolares

A grande quantidade de variáveis envolvidas no planejamento e gestão do transporte escolar dificulta tomadas de decisão, principalmente no processo de definição de rotas. A Otimização Combinatória pode auxiliar na resolução deste problema, através de modelos computacionais que representam situações reais, favorecendo a utilização eficiente dos recursos disponíveis e garantindo a qualidade do serviço de acordo com as restrições e critérios importantes para o tomador de decisão.

Neste contexto, surge o Problema do Roteamento de Veículos Escolares (também conhecido na literatura como *School Bus Routing Problem* – SBRP). Este problema aborda o roteamento de veículos aplicado ao transporte escolar, envolvendo também decisões como definição de pontos de ônibus e horários das rotas de acordo com a grade escolar (MELO, 2022).

O Problema do Roteamento de Veículos Escolares pode ser resolvido através de 5 etapas, chamadas também de subproblemas: preparação de dados, seleção de ponto de ônibus (atribuição do aluno às paradas), geração de rotas de veículos, ajuste do horário do sino da escola e rota de agendamento, sendo as últimas duas etapas necessárias apenas em casos onde há mais de uma escola como destino. (PARK e KIM, 2010)

Alves (2015), descreve estes subproblemas da seguinte forma:

I. Preparação dos dados: este subproblema refere-se à coleta e organização dos dados do problema. Nesta etapa, a rede de rotas possíveis é especificada e são definidos os conjuntos de dados: estudantes, escolas, veículos e matriz origem-destino (OD). A matriz origem-destino armazena dados como tempo de deslocamento ou distância entre localidades e pode ser construída com o auxílio de ferramentas de informação geográfica.
II. Seleção de ponto de ônibus: este subproblema refere-se à seleção de pontos de ônibus e atribuição destes pontos aos estudantes. Em muitos dos casos os pontos de ônibus já são pré-estabelecidos e os estudantes são alocados ao ponto mais conveniente.
III. Planejamento e desenho das rotas de ônibus: este subproblema refere-se à criação de rotas entre os pontos de ônibus e as escolas. O planejamento das rotas pode seguir estratégias como:

a. Ônibus dedicado para o cluster: os estudantes são agrupados em clusters de acordo com as restrições do problema e cada veículo é designado a um cluster.
b. Ônibus dedicado para a escola: cada veículo é designado a uma escola apenas.
c. Conexão com o terminal: combinação entre os modelos de "ônibus dedicado para o cluster" e "ônibus dedicado para a escola" onde são definidas rotas até um terminal e deste terminal, são feitas rotas até as escolas.
d. Estratégias híbridas: estratégia que combina todos os três modelos. Normalmente adotada quando há um grande volume de alunos e escolas.

IV. Sincronização com os horários da escola: é possível tratar o início e término dos horários escolares como restrições ou como variáveis de decisão, a fim de encontrar um tempo de início e de término ótimo que maximize o aproveitamento dos veículos.

V. Refinamento das rotas: neste subproblema são especificados os horários de início e fim de cada rota, além de definir quais rotas cada veículo da frota irá percorrer.

Spada, Bierlaire e Liebling (2005) detalham duas classificações de decomposição do problema: uma baseada na escola e outra baseada na localização dos alunos (casa). Na abordagem escolar, um conjunto de rotas é gerado para cada escola de acordo com o horário de aula e essas rotas são atribuídas à frota de veículos de tal forma que, alunos de escolas diferentes não viagem ao mesmo tempo no mesmo veículo. A segunda abordagem avalia uma parada na rota por vez, considerando diversas escolas ao mesmo tempo. Neste caso, a condição de inserção de uma escola na rota é o custo de inserção. Ao contrário da abordagem escolar, a abordagem domiciliar permite que alunos de escolas diferentes estejam no mesmo veículo ao mesmo tempo.

Do ponto de vista da modelagem, o objetivo mais comum do Problema de Roteamento de Ônibus Escolares é a minimização dos custos totais e consequentemente, do número de ônibus. Podem também ser consideradas combinações do número de ônibus e tempo total de viagem, assim como outros objetivos.

2.2. Problema de roteamento de veículos

Em 1959, Dantzig e Ramser introduziram o conceito do Problema de Roteamento de Veículos, uma abordagem amplamente reconhecida na pesquisa operacional. Essa abordagem tem como objetivo determinar a rota mais curta que visita todos os n pontos de demanda exatamente uma vez, começando e terminando no mesmo local. Os modelos e algoritmos desenvolvidos para resolver esses problemas podem ser aplicados de maneira eficaz não apenas a questões relacionadas à entrega e coleta de mercadorias, mas também para solucionar uma variedade de problemas do mundo real que envolvem sistemas de transporte (ALVES, 2015). Algumas das aplicações mais comuns desse tipo de problema incluem:

- Serviços de entrega;
- Coleta de resíduos,
- Transporte público;
- Manutenção de redes.

No Problema de roteamento de veículos, a rede de estradas onde o transporte é realizado geralmente é representada por um grafo, onde os arcos são os segmentos das estradas e os vértices representam cruzamentos, depósitos e localizações dos clientes. Os arcos podem ser direcionados ou não, isso significa que podem ser percorridos em uma única direção (como vias de sentido único) ou em ambas as direções. Cada arco é associado a um custo que pode representar a distância ou o tempo de percurso. Os clientes são caracterizados de acordo com as seguintes informações, conforme observado em Toth e Vigo (2002):

- O vértice do grafo onde o cliente está localizado.
- A quantidade de bens (demanda), que pode ser de diferentes tipos, que devem ser entregues ou coletados pelo cliente.
- A janela de tempo, ou seja, os períodos do dia em que o cliente pode ser atendido, levando em conta a disponibilidade do cliente ou restrições de tráfego.
- O tempo necessário para carga e descarga de mercadorias, que depende do tipo de veículo utilizado.
- O subconjunto de veículos disponíveis que podem ser atribuídos ao cliente, levando em consideração as limitações de acesso ou restrições específicas do cliente para carga e descarga.

Uma das formulações mais conhecidas para o PRV foi apresentada por Fisher e Jaikumar (1981), a abordagem considera os parâmetros:

$K \equiv$ Número de veículos.

$n \equiv$ Número de clientes para os quais uma entrega deve ser realizada. Os clientes possuem índices de 1 a n e o índice 0 representa o depósito.

$Q_k \equiv$ Capacidade (peso ou volume) do veículo k.

$q_i \equiv$ Tamanho a entrega para o cliente i.

$c_{ij} \equiv$ Custo da viagem direta do cliente i para o cliente j.

São apresentadas as variáveis de decisão:

$$x_{ijk} \equiv \begin{cases} 1, & quando\ o\ veículo\ k\ visita\ o\ cliente\ k\ imediatamente\ após\ o\ cliente\ i \\ 0, & caso\ contrário \end{cases}$$

$$y_{ik} \equiv \begin{cases} 1, & quando\ o\ cliente\ é\ visitado\ pelo\ veículo\ k \\ 0, & caso\ contrário \end{cases}$$

Considerando os parâmetros e variáveis apresentadas, o problema pode ser modelado matematicamente da seguinte forma:

$$\min Z = \sum_{i,j,k} (c_{ij} x_{ijk}) \tag{1.0}$$

Sujeito a:

$$\sum_i q_i y_{ik} \leq Q_k \qquad k = 1, \dots, K \tag{1.1}$$

$$\sum_k y_{ik} = \begin{cases} K, & i = 0 \\ 1 & i = 1, \dots, n \end{cases} \tag{1.2}$$

$$y_{ik} \in \{0,1\} \qquad i = 1, \dots, n \qquad k = 1, \dots, K \tag{1.3}$$

$$\sum_j x_{ijk} = \sum_j x_{jik} = y_{ik} \qquad i = 1, \dots, n \qquad k = 1, \dots, K \tag{1.4}$$

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1 \quad \forall\, S \subseteq \{1, \dots, n\} \qquad 2 \leq |S| \leq n - 1 \quad k = 1, \dots, m \tag{1.5}$$

$$x_{ijk} \in \{0,1\} \qquad i = 1, \dots, n \qquad k = 1, \dots, m \tag{1.6}$$

As restrições (1.1), (1.2) e (1.3) correspondem às restrições de um problema de atribuição generalizada e garantem respectivamente: que a carga atribuída a um veículo esteja dentro de sua capacidade, que cada rota comece e termine no depósito (cliente 0) e que todos os clientes sejam atendidos por algum veículo. As restrições (1.4) garantem que as rotas comecem e terminem no depósito enquanto as restrições (1.5) garantem que não sejam criadas subrotas. As restrições (1.6) garantem que a variável de decisão x_{ijk} seja binária.

Os problemas clássicos de roteamento de veículos estão em constante evolução devido à sua ampla aplicabilidade e aos bons resultados que podem proporcionar. Há um longo histórico de esforços para sistematizar o conjunto de modelos que combinam esses problemas clássicos com outros objetivos. Como resultado, continuam surgindo constantemente novas variantes desses problemas. Os modelos mais recentes vão além da simples programação de rotas, frota, capacidade e horários de chegada dos veículos, considerando atividades nos vértices e arestas. Os problemas enriquecidos podem abranger uma série de processos, como carregamento, embalagem e acondicionamento de cargas, gerenciamento de estoque, programação de equipes, manutenção de veículos, consideração de riscos ambientais, transferência de carga entre veículos, composição de unidades de veículos (comboios e reboques), estacionamento de veículos e outros aspectos relacionados. Essa diversidade de problemas enriquecidos demonstra a aplicabilidade e complexidade cada vez maior do campo do roteamento de veículos (GOLDBARG, E.; GOLDBARG, M. e LUNA, 2016).

2.3. Problema de roteamento de veículos com entrega fracionada e janela de tempo (PRVEFJT)

No problema de roteamento de veículos convencional, cada cliente é visitado uma única vez por um entregador, onde este deve suprir toda a demanda de atendimento. Entretanto, podem haver casos onde as capacidades dos veículos disponíveis não sejam suficientes para atender de uma única vez toda a demanda de um cliente, sendo necessário, portanto, mais de uma viagem àquele local.

O Problema de Roteamento de Veículos com Entrega Fracionada (PRVEF), introduzido por Dror e Trudeau (1989) é uma relaxação do PRV, que aborda a possibilidade de divisão da carga dos clientes em mais de um veículo. Os autores indicam que, fracionar a carga, pode trazer economias significativas na distância total percorrida e do número de veículos.

No PRVEF, para atender à demanda, uma frota de veículos homogêneos e capacitados está disponível para atender um conjunto de clientes. Neste caso, cada cliente pode ser visitado mais de uma vez e a demanda de cada cliente pode ser maior do que a capacidade de um único veículo. Cada rota da solução deve ter início e fim no mesmo depósito (ARCHETTI e SPERANZA, 2008).

Uma outra variação do PRV é o Problema de roteamento de veículos com janela de tempo (PRVJT). O PRVJT é uma generalização do PRV, onde cada cliente é associado a um intervalo de tempo, que restringe o período em que veículo deve realizar o atendimento. Cada

arco fica associado à variável t_{ij}, que corresponde ao custo em unidades de tempo do nó i ao nó j. A chegada do veículo a cada nó está associada ao acúmulo do tempo gasto no deslocamento pela rota. É imposta pelo problema uma janela de tempo $[a_i, b_i]$, sendo assim, um cliente não pode ser atendido antes do tempo a_i, nem depois do tempo b_i (GOLDBARG, E.; GOLDBARG, M. e LUNA, 2016). Problemas deste tipo são importantes principalmente em casos onde não se pode tolerar atrasos ou não se deseja fazer entregas muito antes do necessário.

Outra abordagem importante, mas ainda pouco explorada na literatura é o Problema de Roteamento de Veículo com Entrega Fracionada e Janela de Tempo (PRVEFJT). Segundo McNabb *et al.* (2015), essa é mais uma relaxação do PRV, que combina características do PRVEF e PRVJT, buscando representar de forma mais precisa aplicações do mundo real ao Problema de Roteamento de Veículos.

Belfiore e Yoshizaki (2013), apresentam o PRVEFJT considerando os parâmetros:

$K \equiv$ Número de veículos.

$n \equiv$ Número de clientes para os quais uma entrega deve ser realizada. Os clientes possuem índices de 1 a n e o índice 0 representa o depósito.

$t_{ij} \equiv$ Tempo de viagem do cliente i ao cliente j.

$d_{ij} \equiv$ Distância entre o cliente i e o cliente j.

$Q_k \equiv$ Capacidade do veículo k.

$f_k \equiv$ Custo fixo de uso do veículo k.

$g_k \equiv$ Custo variável por unidade de distância do veículo k.

$c_{ijk} \equiv$ Custo do veículo k percorrer o trajeto do cliente i ao cliente j. É obtido multiplicando a distância d_{ij} pelo custo variável g_k.

$q_i \equiv$ Demanda do cliente i.

$e_i \equiv$ Início da janela de tempo do cliente i (menor horário permitido para iniciar o atendimento do cliente).

$l_i \equiv$ Fim da janela de tempo do cliente i (maior horário permitido para iniciar o atendimento do cliente).

$s_i \equiv$ Tempo de atendimento do cliente i.

$M_{ij} \equiv$ Número suficientemente grande. Pode ser, por exemplo $l_i + t_{ij} - e_j$.

E as variáveis de decisão:

$$x_{ijk} \equiv \begin{cases} 1, & quando\ o\ veículo\ k\ visita\ o\ cliente\ k\ imediatamente\ após\ o\ cliente\ i \\ 0, & caso\ contrário \end{cases}$$

$b_{ik} \equiv$ Momento em que o veículo k começa o serviço no cliente i.

$y_{ik} \equiv$ Fração da demanda do cliente i atendida pelo veículo k.

O objetivo do modelo é minimizar a soma dos custos fixos dos veículos e dos custos de deslocamento para atender aos clientes dentro do limite da janela de tempo. Considerando as variáveis descritas, o problema é modelado da seguinte forma:

$$\min Z = \sum_{k=1}^{K} f_k \sum_{j=1}^{n} x_{0jk} + \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=1}^{K} c_{ijk} x_{ijk} \tag{2.0}$$

Sujeito a:

$\sum_{j=1}^{n} x_{0jk} = 1 \qquad k = 1, \dots, K$ (2.1)

$\sum_{i=0}^{n} x_{ipk} - \sum_{j=0}^{n} x_{pjk} = 0 \qquad p = 0, \dots, n; \quad k = 1, \dots, K$ (2.2)

$\sum_{k=1}^{K} y_{ik} = 1 \qquad i = 1, \dots, n$ (2.3)

$\sum_{i=1}^{n} q_i y_{ik} \leq Q_k \qquad k = 1, \dots, K$ (2.4)

$y_{ik} \leq \sum_{j=0}^{n} x_{jik} \qquad i = 1, \dots, n;\ k = 1, \dots, K$ (2.5)

$b_{ik} + s_i + t_{ij} - M_{ij}(1 - x_{ijk}) \leq b_{jk}$

$i = 1, \dots, n;\ \ j = 1, \dots, n;\ \ k = 1, \dots, K$ (2.6)

$e_i \leq b_{ik} \leq l_i \qquad i = 1, \dots, n;\ \ k = 1, \dots, K$ (2.7)

$y_{ik} \geq 0 \qquad i = 1, \dots, n;\ \ k = 1, \dots, K$

$b_{ik} \geq 0 \qquad i = 1, \dots, n;\ \ k = 1, \dots, K$ (2.8)

$x_{ijk} \in \{0,1\} \qquad i = 0, \dots, n;\ \ j = 0, \dots, n;\ \ k = 1, \dots, K$ (2.9)

As restrições (2.1) e (2.2) estabelecem que cada veículo deve começar e finalizar sua rota no depósito. As restrições (2.3) garantem o atendimento completo da demanda dos clientes enquanto (2.4), garantem que a capacidade de cada veículo não seja excedida. As restrições

(2.5) asseguram que a demanda de um cliente só será atendida quando houver visita de um veículo. As restrições (2.6) estabelecem um tempo mínimo para o início do atendimento ao cliente j em uma rota determinada e também elimina a possibilidade de criação de sub rotas. As restrições (2.7) restringem o atendimento do cliente à sua janela de tempo. As equações em (2.8) garantem que as variáveis de decisão y_{ik} e b_{ik} sejam positivas e (2.9) que x_{ijk} seja binária.

Assim como o PRV e muitas de suas variações, o PRVEFJT é um problema NP-Difícil, fazendo com que métodos exatos exijam alto esforço computacional e tempos de processamento extremamente longos (HO e HAUGLAND, 2004). Diferentes abordagens já foram exploradas na literatura como alternativas para solucionar o problema. Feillet *et al.* (2002) apresentam um algoritmo *Branch-and-cut* para solucionar exemplos do PRVEFJT de forma exata. Ho e Haugland (2004) apresentam a Busca Tabu como alternativa para solução do PRVEFJT. Campos, Yoshizaki e Belfiore (2006) elaboraram algoritmos para a solução do PRVEFJT através de adaptações da heurística de Clarke & Wright e da meta-heurística Algoritmos Genéticos.

2.4. Métodos de solução de problemas

2.4.1. Heurística

Muitos problemas de programação inteira, como o Problema de Roteamento de Veículos e algumas de suas relaxações, são NP-difíceis. Isso significa que, dada a sua complexidade, não podem ser resolvidos computacionalmente em tempo razoável. (BELFIORE e FÁVERO, 2013)

Diante da insuficiência de recursos para utilização de métodos exatos, nas últimas décadas, tem-se feito esforços significativos para criação e aprimoramento de estratégias aproximativas para a solução de problemas NP-difíceis (HILLIER e LIEBERMAN, 2010). Estas estratégias são chamadas de heurísticas, que segundo Colin (2018, p. 374), são "critérios, métodos ou princípios para decidir qual, dentre muitas alternativas de ação, aparenta ser a melhor para o atingimento de um objetivo". Para Golbarg, E.; Goldbarg, M. e Luna, (2016, p. 72), heurística é uma técnica de resolução aproximativa que tem como objetivo encontrar uma solução aceitável para um problema representado computacionalmente e que pode garantir a viabilidade ou se aproximar da solução ótima do problema, empregando esforço computacional razoável.

As heurísticas podem ser classificadas quanto à sua abordagem para fazer escolhas durante o processo de busca. Uma heurística gulosa (*Greedy*) faz escolhas locais que parecem

ser as melhores no momento, sem considerar o impacto total na solução final. Essas heurísticas priorizam a opção mais promissora em cada etapa. Já as heurísticas aleatórias (*Randomized*), fazem escolhas baseadas em alguma aleatoriedade, o que pode ser utilizado para escapar de ótimos locais. Além dessas classificações, as heurísticas também são classificadas em relação ao tipo de construção: podem ser construtivas ou de melhoramento (SOUZA, 2022).

2.4.1.1. Heurísticas construtivas

As heurísticas construtivas são algoritmos que constroem uma solução viável passo a passo, adicionando componentes ou elementos de maneira sequencial para formar uma solução completa. Esse tipo de heurística geralmente começa com uma solução vazia e adiciona elementos de forma iterativa, seguindo regras heurísticas para guiar a construção. Essas regras podem ser projetadas para favorecer a exploração de áreas promissoras do espaço de busca (MELO, 2022).

Souza (2022) detalha algumas das heurísticas mais aplicadas na resolução do Problema de Roteamento de Veículos:

I. Heurística do Vizinho Mais Próximo: essa abordagem começa a construir a solução a partir do depósito e, em cada etapa, adiciona-se uma localidade k ainda não visitada, cuja distância até a última localidade visitada é a menor possível. Esse processo de construção continua até que todos os nós tenham sido visitados, momento em que se estabelece a ligação entre o último nó visitado e a origem.
II. Heurística de Bellmore e Nemhauser: neste método, acrescenta-se à rota atual o nó k ainda não visitado que esteja mais próximo dos extremos da subrota. Em outras palavras, o nó k é conectado a um nó que esteja no início ou fim da subrota. Esse procedimento é repetido até que todos os nós tenham sido visitados.
III. Heurística da Inserção Mais Barata: nesta heurística, o processo de construção parte de uma subrota inicial que inclui três nós. Em cada passo, um nó k ainda não visitado é inserido entre dois outros nós i e j da subrota existente, de modo que o custo de inserção (distância ou tempo) seja o menor possível.

2.4.1.2. Heurísticas de busca local

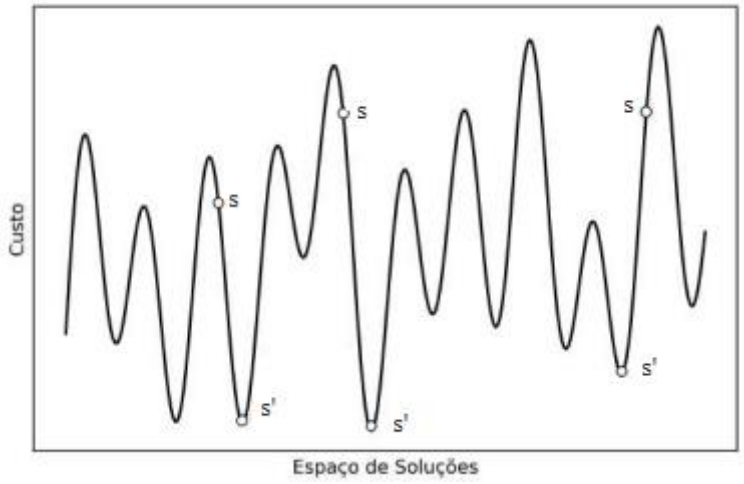
As heurísticas de busca local (também conhecidas como heurísticas de melhoramento) realizam alterações em uma solução em busca de melhores resultados. Estes movimentos

realizados exploram as proximidades da solução em que a busca foi realizada, este espaço de soluções próximas é conhecido como vizinhança. (MELO, 2022)

A escolha da solução vizinha para substituir a solução corrente pode ser feita utilizando diferentes técnicas, como *Best Improvement (BI)* e *First Improvement (FI)*. Em *Best Improvement*, todo o conjunto de soluções vizinhas é explorado e a melhor solução é escolhida caso apresente melhora em relação à solução corrente. Já em *First Improvement*, a busca é interrompida assim que uma solução melhor que a solução corrente for encontrada.

O funcionamento de uma heurística de busca local pode ser visualizado na Figura 1. O gráfico apresenta o universo de possíveis soluções para um problema e seu custo associado. Quando aplicada uma heurística construtiva, uma solução s é obtida. A partir dessa solução, realiza-se uma busca local, que tem como resultado um ótimo local s'. Este processo pode ser repetido até que se encontre um novo ótimo local.

Figura 1: Representação gráfica de uma heurística de busca local



Fonte: (MELO, 2022)

O processo de construção de soluções vizinhas é determinado principalmente pelo operador de vizinhança aplicado, que determina quais tipos de alterações podem ser feitas em uma solução para gerar uma solução vizinha. Aleman, Zhang e Hill (2010) apresentam alguns operadores de vizinhança para o PRVEF aplicados em um algoritmo de memória adaptativa. McNabb *et al.* (2015) apresenta diferentes combinações de operadores de busca local aplicados ao Problema de Roteamento de Veículos com Entrega Fracionada e Janela de Tempo (PRVEFJT), no contexto de uma meta-heurística de otimização baseada em colônia de formigas. Os autores detalham movimentos de busca local escolhidos devido seu uso

generalizado na obtenção de boas soluções para o PRV e suas variantes e/ou por apresentarem resultados promissores no PRVJT e PRVEF. A seguir, são apresentados movimentos de troca aplicáveis ao PRVEFJT.

I. *Swap*: considerando um par de entregas pertencentes a diferentes rotas $g_i \in R_a$ e $g_j \in R_b$. g_i é removida de R_a e inserida em R_b , assim como g_j é removida de R_b e inserida em R_a . Essa operação é viável quando ambas as rotas têm capacidade para receber a entrega após realizada a troca.

II. *Relocate*: considerando duas entregas $g_i, g_j \in R_a$. g_i é removida de sua posição original e inserida após g_j.

III. *Split-to-single*: considerando um par de entregas pertencentes a diferentes rotas, mas de mesmo cliente $g_i \in R_a$ e $g_j \in R_b$. Essas duas entregas g_i e g_j são combinadas e uma nova rota é criada.

IV. *2-opt**: considerando duas entregas $g_i \in R_a$ e $g_j \in R_b, (a \neq b)$. As arestas que conectam g_i a g_{i+1} e g_j a g_{j+1} são removidas. Duas novas arestas são adicionadas conectando g_i a g_{j+1} e g_j a g_{i+1} .

V. *Or-opt*: considerando três entregas $g_i, g_{i+\delta} \in R_a (\delta \geq 2)$ e $g_j \in R_b, (a \neq b)$. A sequência que começa com g_{i+1}e termina com $g_{i+\delta-1}$ é removida de R_a. Uma aresta é então adicionada em R_a de forma que g_i e $g_{i+\delta}$ se tornem entregas consecutivas. O segmento que foi removido de R_a é inserido em R_b de forma que g_j preceda g_{i+1} e $g_{i+\delta-1}$ preceda g_{j+1}.

VI. *Cross Exchange*: considerando quatro entregas $g_i, g_{i+\delta} \in R_a$ e $g_{j,} g_{j+\varepsilon} \in R_b, (a \neq b; \delta, \varepsilon \geq 2)$. A sequência que começa com g_{i+1}e termina com $g_{i+\delta-1}$ é removida de R_a. Da mesma forma, a sequência que começa com g_{j+1}e termina com $g_{j+\varepsilon-1}$ é removida de R_b. Quatro novas arestas são adicionadas conectando os seguintes pares de entregas: g_i a g_{j+1}, g_j a g_{i+1}, $g_{i+\delta-1}$ a $g_{j+\varepsilon}$, e $g_{j+\varepsilon-1}$ a $g_{i+\delta}$.

VII. *2-split-interchange*: considerando $g_i \in R_c$ e um par de rotas R_a e R_b $(a \neq c, b \neq c, a \neq b)$ que não possuem capacidade para g_i. A entrega de g_i é então dividida entre as duas rotas R_a e R_b, de forma que a quantidade máxima possível seja transferida para R_a e o restante para R_b. Cada entrega dividida é inserida na primeira localização viável após sair do depósito em sua nova rota.

VIII. *Combine*: considerando duas entregas $g_i \in R_a$ e $g_j \in R_b$ que pertencem a um mesmo cliente. As estregas são combinadas em uma das duas entregas existentes, sendo g_i a primeira escolha.

IX. *Shift**: considerando um par de entregas $g_i \in R_a$ e $g_j \in R_b, (a \neq b)$ em que o veículo que atende R_b tenha capacidade para g_i, mas o veículo que atende R_a não tem capacidade para g_j. Então, g_i é inserida em R_b na primeira localização viável após sair do depósito. Em seguida, g_j é dividida de forma que uma entrega parcial permaneça em R_b enquanto o restante é inserido em R_a. Ambas as entregas são simplesmente inseridas na primeira localização que resulte em uma solução viável e melhorada.

As heurísticas de busca local podem apresentar diferentes configurações e os movimentos apresentados podem ser aplicados em todas estas. Souza (2022) apresenta algumas das heurísticas de refinamento clássicas:

I. Método da descida/subida: nesta técnica, inicia-se com uma solução qualquer e, em cada etapa, analisa-se todos os possíveis vizinhos dessa solução. Em seguida, seleciona-se apenas o vizinho que proporciona uma melhoria no valor atual da função de avaliação. Esta heurística também é referida como *Best Improvement Method (BI)*. O algoritmo é encerrado quando um ótimo local é encontrado, ou seja, quando não é possível encontrar uma solução vizinha que melhore o resultado atual.

II. Método de primeira melhora: uma abordagem alternativa ao método da descida/subida é o método de primeira melhora, também conhecido como *First Improve Method* (FI). Nesta heurística, a busca local é interrompida assim que um vizinho melhor é encontrado.

III. Método da descida/subida randômico: diferentemente do método da descida/subida, esta heurística não explora toda a vizinhança. O método analisa um vizinho qualquer e somente o aceita em caso de melhora na solução, caso contrário, a solução corrente permanece inalterada e um novo vizinho é gerado aleatoriamente. O procedimento é interrompido após um número fixo de iterações sem melhora.

IV. Descida em vizinhança variável: também conhecido como VND, este método explora o espaço de soluções por meio de trocas sistemáticas nas estruturas de vizinhança utilizadas, aceitando somente soluções de melhora da solução corrente e retornando à primeira estrutura quando uma solução melhor é encontrada. Uma extensão deste método é o RVND, que adiciona aleatoriedade no algoritmo, ao

selecionar randomicamente uma vizinhança para explorar em cada iteração, no lugar de uma sequência fixa.

2.4.2. Meta-heurísticas

Os esforços para criação de novos métodos aproximados para solução de problemas resultaram no surgimento de uma nova classe de estratégias voltadas para o direcionamento do processo de construção de heurísticas. Estas estratégias são chamadas meta-heurísticas. Para Blum e Roli (2003), as meta-heurísticas são combinações de métodos heurísticos básicos em arquiteturas de alto nível, que visam aumentar a eficiência e eficácia na exploração do espaço de busca. Golbarg, E.; Goldbarg, M. e Luna (2016, p. 72) classificam as meta-heurísticas em função da estrutura de vizinhança utilizada e da estratégia de obtenção das soluções:

I. Métodos em função da estrutura de vizinhança empregada:
    a. Método com estrutura de vizinhança fixa: a estrutura de vizinhança é definida ao início do procedimento, não sofrendo mais alterações. Exemplos: *Greedy Randomized Adaptive Search Procedure* (GRASP) e *Simulated Annealing* (SA).
    b. Métodos com estrutura de vizinhança flexível: neste método o conceito de busca por vizinho não fica claro ou a estrutura de vizinhança pode ser alterada ao longo da busca. Exemplos: Busca Tabu e *Scatter Search* (SS).
    c. Métodos com estrutura de vizinhança variável: métodos que exploram a possibilidade de avaliar diferentes estruturas de vizinhança de forma sistemática. Exemplo: *Variable Neighborhood Search* (VNS), *Iterated Local Search* (ILS).

II. Métodos em função da estratégia de obtenção das soluções:
    a. Soluções construtivas: as soluções são organizadas passo a passo em ordem crescente, podendo ser refinadas em fases posteriores. Exemplo: GRASP, colônia de formigas.
    b. Evolutivas: soluções são alcançadas através da formação de outras soluções já conhecidas. Exemplo: algoritmos genéticos, meméticos, culturais, simbióticos, transgenéticos e outros.
    c. De decomposição: as soluções são ordenadas com base em subproblemas de solução mais fácil. Exemplo: decomposição heurística de Dantzig-Wolfe, decomposição heurística de Benders.

d. De informação compartilhada: as soluções se organizam com base em um grupo de métodos ou fatores que alteram as soluções ao longo do processo

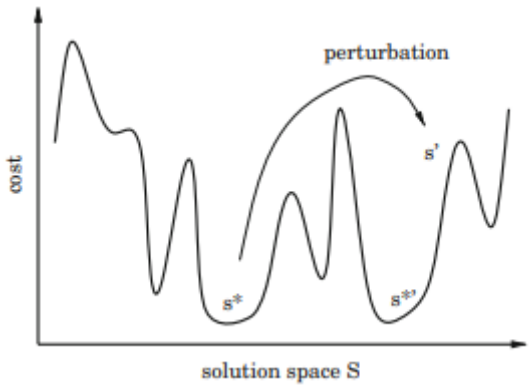
2.4.3. Iterated Local Search (ILS)

Ao projetar uma meta-heurística, é importante buscar a simplicidade tanto em sua concepção quanto na prática. O método *Iterated Local Search* (ILS) é uma abordagem que possui várias características desejáveis de uma meta-heurística, como simplicidade, robustez, efetividade e facilidade de implementação (LOURENÇO; MARTIN e STÜTZLE, 2018).

O ILS é baseado na ideia de que a busca local pode ser aprimorada gerando-se novas soluções de partida através de perturbações na solução corrente. Essa abordagem consiste em iterativamente construir sequências de soluções geradas por uma busca local aplicada a uma solução inicial. A busca local visa melhorar a solução corrente, enquanto perturbações são introduzidas para explorar outras regiões do espaço de soluções. Após a aplicação da busca local e perturbação, uma etapa de aceitação é utilizada para determinar se a solução intermediária será aceita ou se a busca continua a partir da solução corrente (SOUZA, 2022).

É possível utilizar qualquer método de busca local, no entanto, o desempenho do ILS em termos da qualidade final da solução e da velocidade de convergência depende significativamente da escolha desses componentes. A perturbação deve ser intensa o suficiente para escapar de um ótimo local, mas fraca o suficiente para guardar caraterísticas do ótimo local corrente. A Figura 2 representa o funcionamento do algoritmo ILS, começando com um mínimo local s*, aplicamos uma perturbação que leva a uma solução s'. Após a aplicação da busca local, encontramos um novo mínimo local s*' que pode ser melhor do que s*.

Figura 2: Representação gráfica do funcionamento da meta-heurística ILS



Fonte: (LOURENÇO, MARTIN e STÜTZLE, 2018)

O ILS pode ser aplicado em uma grande diversidade de problemas de otimização dada a sua simplicidade e efetividade, e estudos recentes têm explorado o método aplicado na resolução de Problemas de Roteamento de Veículos. Penna, Subramanian e Ochi (2011) introduzem na literatura o procedimento ILS integrado com RVND na fase de busca local para o Problema de Roteamento de Veículos com frota heterogênea. Silva, Subramanian e Ochi (2015) também aplicam o método ILS em conjunto com RVND para resolução de um Problema de Roteamento de Veículo com Entrega Fracionada e Janela de Tempo. Melo (2022), aplica a meta-heurística ILS na resolução do Problema do Roteamento de Veículos Escolares.

3. Metodologia

3.1. Descrição do problema

A Prefeitura Municipal de Itabirito disponibiliza transporte gratuito para todos os moradores da cidade que estejam matriculados em alguma instituição de ensino técnico ou superior localizadas nas cidades de Ouro Preto, Mariana, Ouro Branco e Belo Horizonte. Para fazer parte do programa basta estar com a matrícula ativa na instituição de ensino e realizar o cadastro na Secretaria de Transportes de Itabirito.

Dentre todos estes destinos, Belo Horizonte representa o maior desafio para o planejamento do transporte escolar por conta do grande volume de alunos e instituições de ensino. Segundo dados enviados pela Secretaria Municipal de Transporte de Itabirito, no segundo semestre letivo do ano de 2022, 556 alunos matriculados em instituições de ensino localizadas em Belo Horizonte se beneficiaram do serviço, sendo 224 pela manhã e 332 à noite. Para realizar o serviço, são disponibilizadas vans, de características semelhantes, todas com capacidade para 15 pessoas, que fazem o percurso de ida e volta nos turnos da manhã e noite. A Tabela 1 detalha a distribuição de alunos por Instituição de Ensino e turnos das aulas.

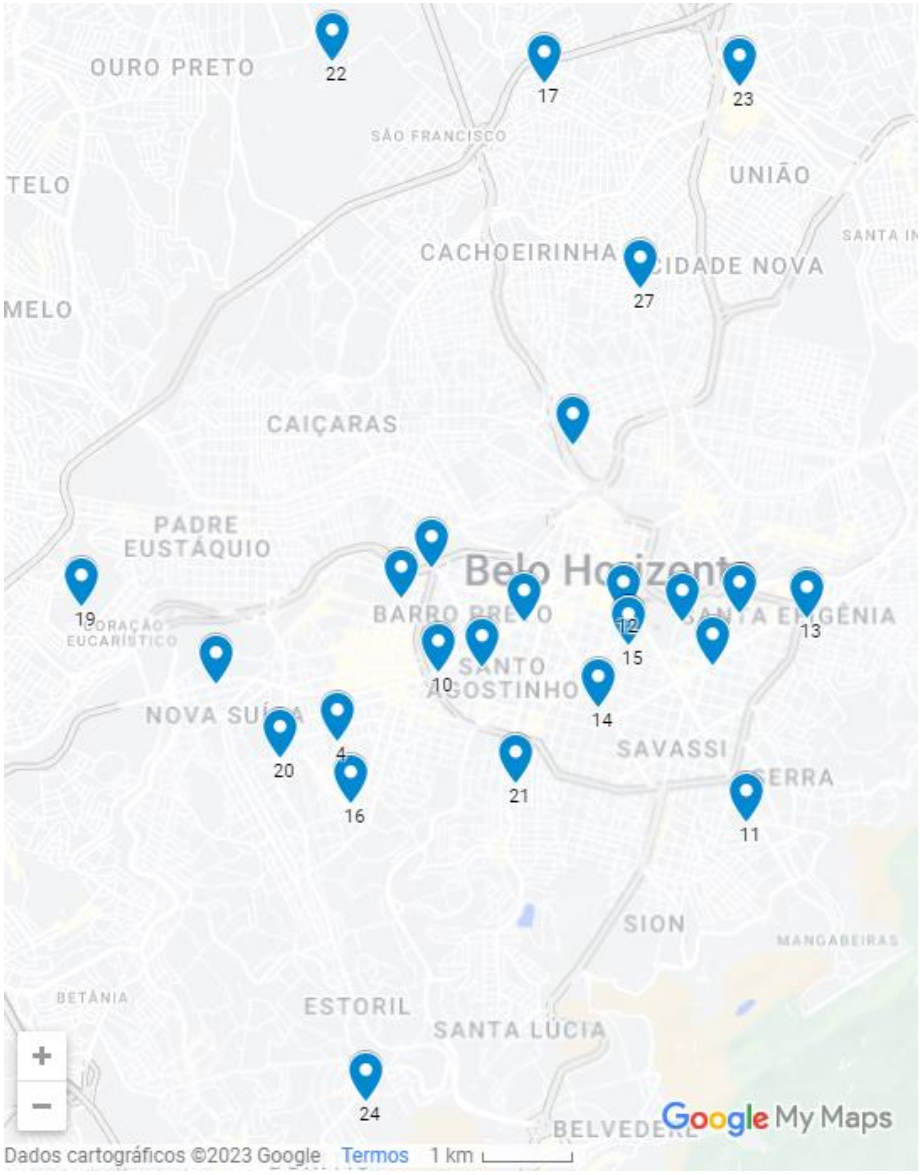
Tabela 1: Distribuição de alunos por Instituição de Ensino no turno da manhã e noite

Instituição de ensino	Quantidade de alunos Manhã	Quantidade de alunos Noite
1	6	8
2	6	1
3	2	3
4	4	1
5	2	-
6	-	2
7	-	2
8	2	14
9	-	2
10	-	1
11	10	16
12	-	3
13	1	2
14	1	4
15	-	3
16	22	42
17	15	46
18	-	3
19	16	25

20	4	-
21	1	6
22	82	34
23	6	37
24	44	70
25	-	4
26	-	1
27	-	2

A Figura 3 detalha a distribuição geográfica das Instituições de Ensino na cidade de Belo Horizonte. O mapa foi construído com o auxílio da ferramenta Google Maps™.

Figura 3: Distribuição geográfica das Instituições de Ensino



Fonte: elaborado pela autora em Google Maps™

A prefeitura segmenta estas instituições de ensino em três classificações, de acordo com a região que a universidade está localizada (Sul, Centro ou Norte). Cada van é direcionada para uma região e o percurso realizado é definido pela cooperativa que realiza o transporte.

3.2. Preparação dos dados

A Secretaria Municipal de Transporte da cidade de Itabirito-MG cedeu o quantitativo de alunos por universidade e turno, além do contrato de prestação de serviços da cooperativa, onde estão descritas as características dos veículos utilizados para o transporte dos alunos e o custo das rotas por região de atendimento (Sul, Centro e Norte). A fórmula de cálculo do custo não é descrita no contrato, entretanto, percebe-se que há certa proporcionalidade em relação à distância dessas regiões à origem.

Para o cálculo das distâncias entre as localidades, foi utilizada a funcionalidade "Rotas" da ferramenta Google Maps™ e em todas as situações, considerou-se a rota de menor distância entre dois locais. Para o tempo de deslocamento entre os destinos foi considerada uma média de 60km/h para rotas intermunicipais e 30km/h para as rotas urbanas (deslocamentos dentro da cidade de Belo Horizonte). Através dessas informações, foram construídas as matrizes de distância e tempo entre as localidades, considerando também a origem.

As informações para o problema foram armazenadas da seguinte forma:

- Capacidade dos veículos (número inteiro);
- Demanda dos clientes (vetor considerando o cliente 0 como o depósito com uma demanda igual a 0);
- Início da janela de tempo (vetor considerando o cliente 0 como o depósito com o início da janela igual a 0);
- Fim da janela de tempo (vetor considerando o cliente 0 como o depósito com o fim da janela igual a 0);
- Matriz de distâncias (matriz de dimensão m x m, onde é m é o total de clientes mais a origem);
- Matriz de tempo (matriz de dimensão m x m, onde é m é o total de clientes mais a origem).

Além dos dados reais do problema, para realizar alguns testes no algoritmo proposto, foram criadas cinco pequenas instâncias de dados fictícios gerados com auxílio da biblioteca "*random*", da linguagem de programação Python. Essa biblioteca gera números aleatórios

usando algoritmos pseudoaleatórios (PRNGs). Esses algoritmos não produzem números verdadeiramente aleatórios, mas sim sequências de números que parecem ser aleatórias, o suficiente para muitos propósitos práticos. Os PRNGs começam com um valor inicial chamado de "*seed*" (semente) e, a partir daí, geram uma sequência determinística de números que parece aleatória. A semente padrão da linguagem Python é baseada no relógio do sistema no momento em que o programa é iniciado (AGUIAR; MAURI e SILVA, 2018).

O algoritmo criado para geração de instâncias com dados fictícios para o Problema de Roteamento de Veículos com Entrega Fracionada e Janela de Tempo, cria matrizes de distâncias e tempos de deslocamento entre os clientes com valores aleatórios, assim como a demanda dos clientes. Todas as matrizes são geradas com o auxílio da função "*randint*" da biblioteca "*random*", gerando valores inteiros dentro de um intervalo de valores determinado pelo executor do código. Para cada instância, foram determinados intervalos de valores para essas matrizes de maneira arbitrária, assim como os intervalos para as janelas de tempo e capacidade dos veículos, a fim de manter o caráter aleatório da base de dados.

Esses dados foram então utilizados na criação de dois tipos de arquivos: um para implementação do algoritmo ILS em Python e outro para modelagem matemática no *software* AMPL.

3.3. Seleção de ponto de ônibus

A prefeitura de Itabirito pré-estabelece pontos de ônibus dentro da cidade. Fica à critério do aluno escolher o ponto de ônibus mais próximo de sua casa. Sendo assim, estes pontos foram mantidos e para simplificação da resolução do problema, o último ponto de ônibus foi considerado como origem para os modelos, já que todos os alunos já estariam dentro da van neste momento.

3.4. Sincronização com os horários da escola

O horário de início das aulas pode variar de acordo com a política da instituição de ensino, mas para simplificação do problema, foi considerada uma janela de tempo fixa para todos os destinos de 40 minutos para a noite, iniciando se às 18:20 e finalizando às 19 horas.

3.5. Planejamento e desenho das rotas

Para resolução deste problema foi adotada uma estratégia híbrida, detalhada por Alves (2015), onde sempre que possível, os veículos ficam dedicados a um único cliente, mas, caso um veículo não consiga suprir toda a necessidade de um cliente, fracionamentos de carga são

permitidos. Considerando ainda a limitação de horários de aula e tamanho da demanda de cada universidade, o problema pode ser classificado como um problema de Roteamento de Veículos com Entrega Fracionada e Janela de Tempo. Além disso, como todos os veículos possuem a mesma capacidade e características semelhantes, considera-se que este é um PRVEJT com frota homogênea.

Para planejamento e desenho das rotas das vans, foi utilizada na modelagem matemática do problema uma adaptação do modelo apresentado por Belfiore e Yoshizaki (2013) detalhado na seção 2.3. Considerando como parâmetros do problema:

$K \equiv$ Número de veículos.

$n \equiv$ Número de clientes para os quais uma entrega deve ser realizada. Os clientes possuem índices de 1 a n e o índice 0 representa o depósito.

$t_{ij} \equiv$ Tempo (em min) de viagem do cliente i ao cliente j.

$Q \equiv$ Capacidade dos veículos.

$c_{ij} \equiv$ Custo (em km de distância) para percorrer o trajeto do cliente i ao cliente j.

$q_i \equiv$ Demanda do cliente i.

$e_i \equiv$ Início da janela de tempo do cliente i (menor horário permitido para iniciar o atendimento do cliente).

$l_i \equiv$ Fim da janela de tempo do cliente i (maior horário permitido para iniciar o atendimento do cliente).

$s_i \equiv$ Tempo de atendimento do cliente i.

$M_{ij} \equiv$ Número suficientemente grande. Pode ser, por exemplo $l_i + t_{ij} - e_j$.

São variáveis de decisão:

$$x_{ijk} \equiv \begin{cases} 1, & quando\ o\ veículo\ k\ visita\ o\ cliente\ j\ imediatamente\ após\ o\ cliente\ i \\ 0, & caso\ contrário \end{cases}$$

$b_{ik} \equiv$ Momento em que o veículo k começa o serviço no cliente i.

$y_{ik} \equiv$ Fração da demanda do cliente i atendida pelo veículo k.

O objetivo do modelo é minimizar o deslocamento total, ou seja, a soma da distância de todas as rotas geradas pelo problema. Essas rotas devem atender aos clientes dentro do limite da janela de tempo, respeitando também a capacidade dos veículos. Considerando os

parâmetros e variáveis descritas, o problema pode ser modelado matematicamente da seguinte forma:

$$\min Z = \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=1}^{K} c_{ij}x_{ijk} \tag{3.0}$$

Sujeito a:

$\sum_{j=1}^{n} x_{0jk} = 1 \qquad k = 1, \dots, K$ (3.1)

$\sum_{i=0}^{n} x_{ipk} - \sum_{j=0}^{n} x_{pjk} = 0 \qquad p = 0, \dots, n; \quad k = 1, \dots, K$ (3.2)

$\sum_{k=1}^{K} y_{ik} = 1 \qquad i = 1, \dots, n$ (3.3)

$\sum_{i=1}^{n} q_i y_{ik} \leq \mathrm{Q} \qquad k = 1, \dots, K$ (3.4)

$y_{ik} \leq \sum_{j=0}^{n} x_{jik} \qquad i = 1, \dots, n; \; k = 1, \dots, K$ (3.5)

$b_{ik} + s_i + t_{ij} - M_{ij}(1 - x_{ijk}) \leq b_{jk}$

$i = 1, \dots, n; \;\; j = 1, \dots, n; \;\; k = 1, \dots, K$ (3.6)

$e_i \leq b_{ik} \leq l_i \qquad i = 1, \dots, n; \;\; k = 1, \dots, K$ (3.7)

$y_{ik} \geq 0 \qquad i = 1, \dots, n; \;\; k = 1, \dots, K$

$b_{ik} \geq 0 \qquad i = 1, \dots, n; \;\; k = 1, \dots, K$ (3.8)

$x_{ijk} \in \{0,1\} \qquad i = 0, \dots, n; \;\; j = 0, \dots, n; \;\; k = 1, \dots, K$ (3.9)

As restrições (3.1) e (3.2) estabelecem que cada veículo deve começar e finalizar sua rota no ponto de origem. As restrições (3.3) garantem que todos os alunos de todas as universidades sejam atendidos enquanto as restrições (3.4), garantem que as viagens respeitem a capacidade dos veículos. As restrições (3.5) asseguram que a demanda de um cliente só será atendida quando houver visita de um veículo. As restrições (3.6) e (3.7) tratam das janelas de tempo, estabelecendo um horário mínimo e máximo para chegada dos veículos aos destinos. As equações em (3.8) garantem que as variáveis de decisão y_{ik} e b_{ik} sejam positivas e (3.9) que x_{ijk} seja binária.

O modelo utilizado no trabalho difere do apresentado na seção 2.3 em dois pontos: em primeiro lugar, como no problema estudado, todos os veículos possuem características semelhantes, foi considerada uma frota homogênea, onde todos os veículos possuem a mesma

capacidade Q. A outra divergência em relação ao modelo da seção 2.3 está na função objetivo. O modelo apresentado por Belfiore e Yoshizaki (2013) considera os gastos fixos de uso de um veículo, bem como um gasto variável atrelado à distância percorrida por cada veículo. Como o problema apresentado tem por objetivo minimizar a distância total percorrida pelas rotas, considerou-se o custo somente como a distância entre as localidades. Desta forma, a função objetivo do problema (3.0) é representada pela minimização da soma das distâncias percorridas (custo) por todos os veículos.

3.5.1. Algoritmo proposto

O PRVEFJT é considerado um problema NP-Difícil, ou seja, requer tempos computacionais elevados à medida que o tamanho do problema aumenta. Por isso, o uso de métodos heurísticos se torna uma abordagem eficaz para enfrentar esse desafio.

Uma solução para o PRVEFJT pode ser representada através de um conjunto de rotas, que sempre tem início e fim na origem. Cada uma destas rotas possui uma carga atrelada a si, que representa a quantidade de alunos transportados de cada uma das localidades visitadas pela van. A carga total de uma rota deve ser menor ou igual a capacidade do veículo utilizado.

Para avaliar a qualidade de uma solução, é somada a distância percorrida por todas as rotas. Quanto menor a distância total percorrida, melhor. Essa distância pode ser calculada através da matriz c_{ij}, que armazena as distâncias entre todas as localidades i e j presentes no problema. Sendo assim, considerando n como o número total de clientes e K como o número total de veículos, pode-se definir a função objetivo do problema da seguinte maneira:

$$Função\ objetivo = \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=1}^{K} c_{ij}x_{ijk}$$

Onde,

$$x_{ijk} \equiv \begin{cases} 1, & quando\ o\ veículo\ k\ visita\ o\ cliente\ j\ imediatamente\ após\ o\ cliente\ i \\ 0, & caso\ contrário \end{cases}$$

O método proposto divide a construção da solução em dois momentos. No primeiro momento, um algoritmo chamado “CriaRotasUnicas” constrói rotas para atender somente aqueles clientes que possuem demanda maior que a capacidade dos veículos, de tal forma que o maior número de rotas com um destino e ocupação máxima possíveis sejam criadas. Em seguida, para atender a demanda restante dos clientes, é aplicada a meta-heurística ILS, onde as rotas criadas seguem o princípio de fracionamento de carga.

O pseudocódigo **Algoritmo 1** detalha o procedimento "CriaRotasUnicas," que tem como dados de entrada o número de clientes "n_c", o vetor de demanda "dem", além da capacidade dos veículos "cap_v".

A função inicializa os vetores vazios "s" e "s_{cg}", que representam o conjunto de rotas e a distribuição de cargas atrelada à cada uma dessas rotas, respectivamente. O vetor "$c_{Atendidos}$" também é inicializado com "n_c" valores "Falso". Este vetor representa o status de atendimento de um cliente. Logo em seguida, o algoritmo percorre cada cliente (linhas 4 – 10), verificando se sua demanda é maior que a capacidade do veículo, caso a afirmação seja verdadeira, uma rota com destino ao cliente "i" é criada. Esse procedimento é repetido até que a demanda do cliente seja menor que a capacidade do veículo e, portanto, não seja mais possível criar uma rota que tenha um único destino e ainda assim ocupe totalmente um veículo.

Sempre que um destino é inserido em uma rota, a carga alocada do cliente - que neste caso é exatamente a capacidade do veículo "cap_v" - é retirada da demanda correspondente. Caso após uma alocação em rota, a demanda do cliente passe a ser igual a 0, seu status em "$c_{Atendidos}$" muda para "Verdadeiro". O procedimento retorna as rotas criadas, demanda residual dos clientes e status de atendimento dos clientes.

Algoritmo 1: CriaRotasUnicas

```
1   Procedimento CriaRotasUnicas (nc, dem, capv)
2       s, scg ← [Ø]
3       cAtendidos ← Falso para i,..., nc
4           para i ← 1,..., nc + 1 faça
5               enquanto dem[i] ≥ capv faça
6                   s ← adiciona a rota [i] na solução; scg ← adiciona a carga [capv] na rota
7                   dem[i] ← dem[i] – capv
8                   se dem [i] = 0, faça cAtendidos[i] = Verdadeiro
9               fim
10          fim
11      retorne s, dem, cAtendidos
12  fim
```

A segunda etapa, leva como *input* a demanda residual dos clientes após passarem pelo procedimento "CriaRotasUnicas". Esta estratégia foi adotada para evitar algumas realocações que possam piorar a qualidade da solução, considerando que, o melhor cenário, sempre será ter um veículo totalmente carregado e com um único destino.

Para este segundo momento, propôs-se um algoritmo baseado no método *Iterated Local Search* (ILS), apresentado por Silva, Subramanian e Ochi (2015). Nele, as soluções iniciais são geradas utilizando uma heurística de inserção aleatória gulosa, enquanto a busca local é

realizada utilizando o *Randomized Variable Neighborhood Descent* (RVND). Sempre que uma solução fica presa em um ótimo local, ou seja, nenhum dos operadores de busca local é capaz de melhorá-la, é aplicado um mecanismo de perturbação sobre essa solução.

O **Algoritmo 2** apresenta o pseudocódigo da meta-heurística ILS aplicada. O algoritmo executa um número "I_{Max}" de iterações (linhas 3 – 26) e em cada uma delas, uma solução inicial é gerada usando uma heurística de inserção aleatória gulosa (linha 4). Essa solução inicial é modificada aplicando um procedimento RVND na fase de busca local (linha 9). As estruturas de vizinhança utilizadas no RVND são apresentadas na seção 3.5.3. Caso alguma dessas soluções encontradas na etapa de busca local ultrapasse o número mínimo de veículos necessários "k_{min}" é aplicado o procedimento "*EmptyRoutes*" (consulte a seção 3.5.4.6), projetado para esvaziar as rotas até que se chegue ao menor número de veículos possível. Se uma solução aprimorada for encontrada, o "$iter_{ILS}$" é reinicializado (linhas 12 - 15). A melhor solução atual é então perturbada (linha 16), usando o mecanismo de perturbação descrito na seção 3.5.5. O nível da perturbação varia de acordo com a quantidade de iterações sem melhora na melhora na solução, quanto menor o número de iterações sem melhora, menor é a perturbação. Para os problemas da manhã e noite, o nível "k" de perturbação assumiu os valores 5, 6 e 7. A cada iteração dentro de "I_{max}", a solução corrente é avaliada pela função "CalculaDistancia", que retorna a distância total das rotas da solução "D_". Caso a solução corrente apresente menor distância total que a melhor solução até então, "s*" é atualizado com o valor de "s_" (22 – 25).

Algoritmo 2: Split ILS

```
1  Procedimento SplitILS(I_max, I_ILS)
2      D* ← ∞; s* ← [Ø]
3      para i ← 1, ..., I_max faça
4          s ← solucaoInicial( )
5          s_ ← s
6          D_ ← CalculaDistanciaTotal(s_)
7          iter_ILS ← 0
8          enquanto iter_ILS < I_ILS faça
9              s ← RVND(s)
10             s ← EmptyRoutes(s)
11             D ← CalculaDistanciaTotal(s)
12             se d < d_ então
13                 s_ ← s
14                 iter_ILS ← 0
15             fim
16             se iter_ILS = 0 faça: s ← perturba(s_,k)
17             se iter_ILS > 0 e <=10 faça: s ← perturba(s_,k+1)
```

```
18          se iter_ILS > 10 faça: s ← perturba(s_,k+2)
19          iter_ILS ← iter_ILS + 1
20        fim
21        D_← CalculaDistanciaTotal(s_)
22        se D_ < D* então
23          s* ← s_
24          D* ← D_
25        fim
26      retorne s*, D*
27  fim
```

3.5.2. Solução inicial

O algoritmo "solucaoInicial" constrói uma solução inicial para o problema de roteamento de veículos com carga fracionada e janela de tempo através de uma abordagem construtiva aleatória. O método inicializa os vetores de solução "s", carga "s_{cg}", além de "s_" e "$s_{cg\_}$" que armazenam as rotas e cargas temporariamente para inclusão na solução (linha 2). São inicializados também o espaço disponível nos veículos "e" e índice do veículo "v" (linha 3), além das variáveis "c_0" e "c_1" que indicam clientes subsequentes em uma rota.

O algoritmo percorre toda a lista de clientes até que todos sejam atendidos. A cada iteração, um cliente é adicionado temporariamente na última posição da rota corrente e sua inserção no percurso é avaliada pela função "Saida" (linhas 9 – 11). Caso a adição do cliente não infrinja a restrição de janela de tempo, o mesmo é adicionado a uma lista de candidatos (linhas 12 – 14).

Se ao percorrer todos os clientes, a lista de candidatos continuar vazia, o procedimento avalia a rota corrente. Caso a rota corrente esteja vazia, significa que não há solução viável para o problema, assim, o algoritmo é encerrado. Caso contrário, uma nova rota é criada e o procedimento retorna a avaliar a lista de clientes (linhas 17 – 24).

Em seguida, um valor "k" é selecionado aleatoriamente em um intervalo de valores pré-determinado. Os clientes armazenados no vetor "CL" são ordenados por distância e somente os k menores valores são mantidos na lista de candidatos. Um destes k valores é selecionado aleatoriamente para ser inserido na rota atual "s_" (linhas 26 – 29).

Para alocar a carga, verifica-se se a demanda do cliente é menor que o espaço disponível no veículo. Caso a demanda residual seja menor que o espaço, isso significa que o veículo pode atender todo o cliente, por isso, toda a sua demanda é alocada para o veículo e o cliente muda

seu status para atendido. Após essa operação, o cliente inserido na rota corrente passa a ocupar o espaço de "c_0" (linhas 30 – 32).

Caso o espaço do veículo seja menor ou igual à demanda do cliente, a maior carga possível é alocada na rota e um novo percurso é iniciado. Em casos onde a demanda residual é exatamente igual ao espaço disponível, o cliente muda seu status para atendido. As cargas e espaço do veículo são atualizados sempre que algum cliente é inserido na rota corrente (linhas 33 – 40).

Algoritmo 3: SolucaoInicial

```
1  Procedimento SolucaoInicial ()
2      s, s_cg, s_, s_cg_ ← [Ø]
3      e ← cap_v; v ← 0
4      c0 ← 0; c1 ← Ø
5      enquanto houver cliente não atendidos, faça:
6          CL ← [Ø]
7          para i ← 1,..., nc+1 faça
8              se c_atendido[i] for falso faça
9                  s_ ← adiciona cliente i na rota
10                 h_ ← saída(s_)
11                 s_ ← retira cliente i da rota
12                 se h_ não for falso faça
13                     d ← dist[c0][i]; CL ← CL + [(i,d)]
14                 fim
15             fim
16         fim
17         se CL = [Ø] faça
18             se c0 = 0 significa que não há solução viável, encerre o procedimento
19             caso contrário, faça:
20                 s ← insere a rota s_ no conjunto de soluções
21                 s_cg ← insere a distribuição de cargas da rota s_ no conjunto de cargas.
22                 v ← v + 1 ; e ← cap_V ; s_, s_cg_← Ø ; c0 ← 0
23                 retorne para a linha 8
24             fim
25         fim
26         CL ← ordena de maneira crescente em relação à distância d
27         k ← seleciona aleatoriamente um valor entre [g_i,..., g_i+α]
28         c1 ← seleciona aleatoriamente um dos k primeiros elementos de CL
29         s_ ← adiciona o cliente c1 na última posição da rota
30         se dem[c1] < e faça
31             e ← e – dem[c1]; s_cg_← s_cg_+ dem[c1]
32             dem[c1] ← 0; c_Atendidos ← Verdadeiro; c0 ← c1
33         caso contrário, faça
34             dem[c1] ← dem[c1] - e; s_cg_← s_cg_ + e
35             e ← 0
36                 se dem[c1] = 0 faça c_Atendidos ← Verdadeiro
```

```
37          s ← insere a rota s_ no conjunto de soluções
38          scg ← insere a distribuição de cargas da rota s_ no conjunto de cargas.
39          v ← v + 1 ; e ← capV ; s_, scg_← Ø ; c0 ← 0
40        fim
41      fim
42    retorne s, scg, dem
43    fim
```

3.5.3. Heurística de Busca Local

A busca local é realizada por meio da heurística RVND (*Randomized Variable Neighborhood Descent*). O método é uma variação do *Variable Neighborhood Search* (VNS), onde um conjunto de operadores de vizinhança é usado para intensificar a busca por soluções de maior qualidade em diferentes regiões do espaço de soluções do problema, sem a necessidade de uma etapa de perturbação.

O **Algoritmo 4** apresenta o pseudocódigo do procedimento. No loop principal, o algoritmo executa várias iterações de busca local, controladas pelos parâmetros "I_{RVND}" e "inv_{max}", que significam o número máximo de iterações e o número máximo de operações inválidas aceitas, respectivamente. O operador de busca local é selecionado aleatoriamente (*Swap* ou *Shift**) e através desse movimento uma solução vizinha é gerada. Em seguida, a função "AvaliaBusca" avalia se a mudança feita pelo operador gera uma solução válida. Caso a nova solução respeite todas as restrições, avalia-se o custo total (soma das distâncias percorridas de todas as rotas). Se o custo total da solução corrente for menor que o menor custo encontrado até então "D*", a solução corrente é armazenada como melhor solução até então "s*". O procedimento é interrompido quando o número máximo de iterações sem melhora é alcançado ("I_{RVND}") ou quando "inv_{max}" soluções inválidas forem geradas.

Algoritmo 4: RVND

```
1   Procedimento RVND (IRVND, invmax)
2     D* ← ∞; s*, scg* ← Ø; iter, inv ← 0
3     enquanto iter < IRVND e inv < invmax faça
4       D ← 0
5       vizinhança ← escolha aleatoriamente entre 'Swap' e 'Shift*'
6       se vizinhança = 'Swap' faça s_, scg_,s1_ s2_= swap(s, scg)
7       se vizinhança = 'Shift*' faça s_, scg_, s1_, s2_= shift*(s, scg)
8       valido ← AvaliaBusca()
9       se valido, faça
10        iter ← iter + 1
11        para j ← 0,...,tamanho s faça
12          D ← D + calculaDistancia(s_)
```

```
            fim
            se D < D* faça
                D* ← D; s* ← s_; scg* ← scg_,
            fim
        caso contrário, faça inv ← inv + 1
        fim
    fim
    retorne s*, scg*
fim
```

3.5.3.1. Swap

O operador *Swap* realiza uma troca entre dois clientes pertencentes a duas rotas diferentes. O **Algoritmo 5** apresenta o pseudocódigo do movimento, que recebe como entrada o conjunto de rotas, a distribuição de cargas e a matriz de distâncias. Duas rotas diferentes são selecionadas aleatoriamente e têm seus dados armazenados. Posteriormente, um cliente de cada uma dessas rotas também é selecionado aleatoriamente e tem seus dados armazenados. Os clientes são removidos das suas rotas originais e a função "custoInserção" calcula as posições de menor custo para realizar as inserções nas novas rotas. Os clientes são inseridos nas novas rotas e a solução é atualizada.

Algoritmo 5: *Swap*

```
1   Procedimento Swap(s, scg,dist)
2     λs1, λs2 ← seleciona aleatoriamente duas rotas diferentes
3     s1 ← s[λs1]; s1cg ← scg[λs1]
4     s2 ← s[λss]; s2cg ← scg[λs2]
5     λc1, λc2 ← seleciona aleatoriamente um cliente de cada uma das rotas
6     c1 ← s1[λc1]; scg^c1 ← s1cg[λc1];
7     c2 ← s2[λc2]; scg^c2 ← s1cg[λc2];
8     remove s[λs1], scg[λs1], s[λss], scg[λs2]
9     p1,d1 ← custoInsercao(s1, c2, scg^c2, dist); p2,d2 ← custoInsercao(s2, c1, scg^c1, dist)
10    s1[p1] ← c2; s1cg[p1] ← scg^c2
11    s2[p2] ← c1; s2cg[p2] ← scg^c1
12    retorne s, scg, λs1, λs2
13  fim
```

3.5.3.2. Shift*

O operador *Shift**, apresentado por McNabb *et al.* (2015), realiza uma troca inter-rotas. A operação começa selecionando aleatoriamente duas rotas diferentes. Em seguida, escolhe um cliente aleatoriamente da primeira rota para ser movido para a segunda rota. Se a adição do cliente na segunda rota ultrapassar a capacidade do veículo, é realizada uma redistribuição de

cargas entre os clientes na rota, até que a adição seja possível. O **Algoritmo 6** detalha melhor o movimento *Shift** através de um pseudocódigo.

Após selecionar as rotas a serem modificadas e cliente a ser realocado, o procedimento armazena as informações dessas rotas e cliente (linhas 2 – 6). Em seguida, o cliente é removido de sua rota original (linha 7) e a capacidade da rota de destino é avaliada. Se a inserção do novo cliente na rota infringir o limite de capacidade do veículo, os clientes de maior carga na rota são realocados total ou parcialmente para a rota 1, até que seja possível adicionar o cliente 1 na rota 2 (linhas 8 – 14). O cliente 1 é inserido na rota 2 e o procedimento retorna à solução atualizada, além do índice das rotas que sofreram modificações.

Algoritmo 6: *Shift**

```
1  Procedimento Shift*(s, s_cg, cap_v)
2      λs1, λs2 ← seleciona aleatoriamente duas rotas diferentes
3      s1 ← s[λs1]; s1cg ← s_cg[λs1]
4      s2 ← s[λss]; s2cg ← s_cg[λs2]
5      λc1 ← seleciona aleatoriamente um cliente da rota 1
6      c1 ← s1[λc1]; s_cg^c1 ← s1cg[λc1];
7      remove s[λs1], s_cg[λs1]
8      se s2cg + s_cg^c1 > cap_v faça
9          gap ← s2cg + s_cg^c1 - cap_v
10         CL ← ordena clientes de s2 de acordo com sua carga na rota de forma decrescente
11         para i ← 0, ..., tamanho CL faça
12             se s2cg[i] > carga transfira todo o cliente i para s1 e vá para o próximo cliente
13             caso contrário, transfira somente o necessário do cliente i para s1
14         fim
15     fim
16     s[λs2] ← s[λs2] + c1; s_cg[λs2] ← s_cg[λs2] + s_cg^c1
17     retorne s, s_cg, λs1, λs2
18 fim
```

3.5.4. Procedimentos auxiliares

3.5.4.1. Verificação de janela de tempo e definição do horário de saída da origem

O algoritmo “Saida” é um procedimento auxiliar responsável por verificar a viabilidade de um determinado trajeto garantindo que todas as restrições de tempo sejam atendidas. Recebe como entrada um vetor contendo a rota a ser analisada “s_”, a matriz de tempo de deslocamento “t”, o início e fim da janela de tempo de cada cliente “$JT_{i,f}$” e o tempo de serviço “ts”.

O procedimento é inicializado definindo o horário de saída mais cedo possível que atenda à janela de tempo do primeiro cliente. O horário atual “h_” é inicializado como o horário

de saída e o primeiro cliente da rota como a origem. O algoritmo percorre toda a lista de clientes e verifica o atendimento da janela de tempo com base no horário atual. Se a condição for atendida, o horário atual é atualizado, somando o tempo de viagem entre origem e cliente atual com tempo de serviço. A origem também é atualizada, recebendo o cliente atual.

Se todos os clientes forem atendidos dentro de suas janelas de tempo, a função retorna o horário que a rota deve sair da origem para cumprir a restrição. Caso contrário, a função retorna o valor "Falso" indicando que o trajeto não respeita a restrição de janela de tempo.

Algoritmo 7: Saída

```
1  Procedimento Saída (s_, t, JT_{i,f}, ts)
2      h ← JT_i[s_[0]] – t[0][ s_[0]]
3      h_ ← h
4      c_o ← 0
5      para i ← 1,..., tamanho r_ faça
6          c_1 ← s_[i]
7          se JT_i[c_1] <= h_ + t[c_o][ c_1] e JT_f [c_1] >= h_ + t[c_o][ c_1] + ts
8              h_ ← h_ + t[c_o][ c_1] + ts; c_o ← c_1
9          caso contrário, faça
10             retorne Falso
11         fim
12     fim
13     retorne h
14 fim
```

3.5.4.2. Cálculo do custo de uma rota

O procedimento "calculaDistancia" tem como objetivo calcular a distância total percorrida em uma rota específica, o que é útil para avaliar a qualidade das soluções encontradas pelos algoritmos e comparar diferentes rotas.

O algoritmo recebe como entrada a rota "s_" representada por uma lista de clientes, e uma matriz de distância "dist" que contém os custos "distâncias" entre os clientes. Primeiro, a função modifica a rota adicionando a origem no início e no final da lista. Em seguida, o procedimento percorre a lista de clientes na rota, de um cliente ao próximo, somando a distância entre esses clientes. Ao percorrer toda a rota, a função retornará a distância total percorrida, incluindo as viagens de ida e volta à origem.

Algoritmo 8: CalculaDistância

```
1  Procedimento CalculaDistância(s_,dist)
2      s_ ← [0] + s_ + [0]
3      d ← 0
5      para i ← 1,..., tamanho s_ - 1 faça
```

```
5  |  |  |  c0 ← s_[i]
6  |  |  |  c1 ← s_[i+1]
7  |  |  |  d ← d + dist[c1][c2]
8  |  |  fim
9  |  |  retorne d
10 |  fim
```

3.5.4.3. Cálculo do custo total da solução

O algoritmo "calculaDistanciaTotal" tem como objetivo calcular a distância total percorrida por todas as rotas de uma solução. Este procedimento permite avaliar a qualidade de uma solução, já que a minimização do custo total das rotas é a função objetivo do problema.

Como entrada, são levados o conjunto de rotas "s" como entrada e a matriz de distância "dist" que contém os custos "distâncias" entre os clientes. A função percorre cada rota presente na lista e para cada rota, chama o procedimento "calculaDistancia" para calcular a distância percorrida nessa rota específica. Assim, a distância de todas as rotas é somada, representando o custo total da solução que é retornado pela função através da variável "D".

Algoritmo 9: CalculaDistânciaTotal

```
1  Procedimento CalculaDistânciaTotal(s,dist)
2  |  D ← 0
3  |  para i ← 1,..., tamanho s_  faça
4  |  |  D ← D + CalculaDistância(s[i])
5  |  fim
6  |  retorne D
7  fim
```

3.5.4.4. Cálculo do custo de inserção de um cliente em uma rota

O procedimento custoInsercao tem como objetivo analisar a viabilidade de inserção de um cliente em uma determinada posição dentro de uma rota, além de calcular o custo dessa inserção. Ele recebe como entrada uma lista representando uma rota existente "s_", o cliente que se deseja inserir na rota "c" e a matriz de distâncias "dist". Além disso, também são inseridos como dados de entrada informações para análise das restrições, como início e fim da janela de tempo "$JT_{i,f}$" e capacidade do veículo "cap_v".

O objetivo do algoritmo é encontrar a posição de menor custo para inserção. Ele percorre todas as posições possíveis na rota existente (incluindo a posição inicial e final) e insere o cliente temporariamente em cada uma dessas posições. Em seguida, verifica se a inserção atende às restrições de janela de tempo utilizando a função "Saida".

Se a inserção for viável (ou seja, atender às restrições de janela de tempo), o custo da rota com a inserção é calculado utilizando o procedimento “calculaDistancia”. Se esse custo for menor que o menor custo registrado até o momento, a posição de inserção e o novo menor custo são atualizados. Após verificar todas as posições possíveis na rota, a função retorna a melhor posição onde o cliente deve ser inserido “p*” e o custo para realizar essa inserção “d*’.

Algoritmo 10: CustoInsercao

```
1  Procedimento CustoInsercao(s_, c, dist)
2      d* ← ∞
3      p* ← Ø
4      para i ← 1,..., tamanho s_  faça
5          s_[i] ← inserir cliente c na posição i da rota
6          h_ ← saída(s_)
7          se h_ não for falso, faça
8              d_ ← calculaDistancia(d_)
9              se d_ < d, faça
10                 d* ← d_
11                 p* ← i
12             fim
13         fim
14         s_[i] ← remove cliente c da posição i da rota
15     fim
16     retorne p*, d*
17 fim
```

3.5.4.5. Inserção de cliente em uma rota

O procedimento *SplitReisertion* proposto por Silva, Subramanian e Ochi (2015) tem como base uma generalização da estrutura de vizinhança *k-split* proposta por Boudia, Prins e Reghioui (2007). Seu objetivo é remover o cliente da rota original r_p e encontrar uma nova posição para inseri-lo em outra rota. A remoção do cliente pode ocorrer totalmente ou parcialmente, dependendo da capacidade disponível nas outras rotas.

Este procedimento tem como dados de entrada o conjunto de rotas da solução “s”, uma rota proibida “r_p” e um cliente desta rota proibida “c”. Além disso, também são levados como entrada informações para análise das restrições como início e fim da janela de tempo, “$JT_{i,f}$” capacidade “cap_v” e distribuição de cargas de cada veículo “s_{cg}”.

Primeiro, o algoritmo calcula o espaço disponível em cada veículo e verifica se esse espaço remanescente é suficiente para que o cliente a ser realocado tenha toda sua demanda atendida. Caso não haja espaço, a função retorna as rotas e cargas inalteradas, já que não há capacidade suficiente no sistema para realizar a movimentação.

Em seguida, todas as rotas diferentes de r_p e que possuem algum espaço disponível são percorridas. Para cada uma dessas rotas, é calculado o custo de inserção do cliente através da função "custoInsercao," que calcula o custo de inserir um cliente em uma rota, retornando a melhor posição para alocação. Além disso, a função "custoInsercao" avalia se a inserção do cliente infringe a restrição de janela de tempo, caso essa restrição não seja respeitada, o movimento é descartado. As rotas que possuem viabilidade para inserção do cliente são inseridas em um vetor "CL", que posteriormente é ordenado de forma crescente pelo seu custo.

Uma nova verificação de carga é feita, agora somente em relação às rotas que tiveram suas restrições validadas e são candidatas à inserção do cliente. Caso a soma dos espaços vazios nessas rotas não sejam suficientes para atender a demanda do cliente, a função retorna as rotas e cargas inalteradas.

No próximo passo, o algoritmo itera a lista de rotas candidatas à inserção e realiza a inserção do cliente na melhor posição encontrada. Se a rota escolhida não tiver capacidade para atender toda a demanda do cliente, a carga é alocada parcialmente e o processo é repetido para as outras rotas da lista de candidatas até que a demanda do cliente seja completamente atendida.

Após a realocação, o cliente é removido de sua rota original r_p. A função retorna as rotas e cargas atualizadas.

Algoritmo 11: *SplitReisertion*

```
1  Procedimento SplitReisertion(s, s_cg, cap_v, r_p, c)
2      α ← posição do cliente c na rota r_p
3      s_cg^c ← cargas[r_p][ α]
4      CL ← [Ø]; e, n ← 0
5      para i ← 1,..., tamanho s  faça
6          se i ≠ r_p, faça e ← e + (cap_v – carga total rota i)
7      fim
8      se e < s_cg^c retorne s, s_cg
9      para j ← 1,..., tamanho r  faça
10         se j ≠ r_p, faça
11             p, d = custoInsercao(s[j], c)
12             se a inserção for viável, faça
13                 CL ← CL ∪ {'rota': j, 'posição': p, 'custo': d}
14             fim
15         fim
16     fim
17     CL ← ordena em relação ao custo d de forma crescente
18     para k ← 1,..., tamanho CL  faça
19         e ← e + (cap_v – carga total rota k)
20     fim
21     se e < s_cg^c retorne s, s_cg
```

```
22  enquanto s_cg^c > 0 e tamanho L > n
23      CL_n ← CL[n]; λ_s ← CL_n['rota']; λ_p ← CL_n['posição']
24      s[λ_s][ λ_p] ← inserir cliente c na rota λ_s, na posição λ_p
25      e_λ ← somatório s_cg[λ_s]
26      se e_λ < s_cg^c faça
27          s_cg^c ← s_cg^c - e_λ; cargas[λ_s] ← insere e_λ na posição λ_p; e_λ ← 0
28          n ← n + 1
29      caso contrário, faça
30          cargas[λ_s] ← insere e_λ na posição λ_p; e_λ ← e_λ - s_cg^c; s_cg^c ← 0
31      fim
32  fim
33  s ← remove cliente c da rota r_p; s_cg ← remove s_cg^c do cliente c na rota r_p
34  retorne s, s_cg
35  fim
```

3.5.4.6. Esvaziamento de rotas

Silva, Subramanian e Ochi (2015) apresentam o procedimento "*EmptyRoutes"* que tem por objetivo esvaziar rotas que possuem capacidade ociosa em um problema de roteamento de veículos. A ideia é combinar os clientes que estão na rota a ser esvaziada em outras rotas para reduzir o número total de veículos utilizados e melhorar a eficiência da solução

O procedimento, detalhado pelo pseudocódigo do **Algoritmo 12**, recebe como entrada o conjunto de rotas da solução "s" e suas respectivas cargas "s_{cg}", a demanda total dos clientes "dem", capacidade do veículo "cap_v", matriz de distâncias "dist", além de informações para verificação da restrição de janela de tempo.

O algoritmo inicia definindo o valor mínimo de veículos necessários "k_{min}' com base na demanda total dos clientes dividida pela capacidade dos veículos. Enquanto o número de rotas "k' for maior que o número mínimo de veículos necessários "k_{min}" o procedimento entra em um loop. Ele calcula as cargas totais de cada rota e as ordena de forma crescente pela carga.

Em seguida, o procedimento seleciona a rota de menor carga "r_p". Enquanto a carga dessa rota for diferente de zero (ou seja, a rota não estiver vazia), o procedimento começa a retirar clientes dessa rota por meio da função "*splitReinsertion*".

A função "*splitReinsertion"* tenta inserir os clientes da rota selecionada "r_p" em outras rotas para otimizar o aproveitamento dos veículos. Se a tentativa de inserção não resultar em uma melhoria na solução (ou seja, a rota permanece inalterada), o procedimento sai do loop para tentar esvaziar outra rota.

Se a rota ficar vazia após tentar retirar os clientes, ela é removida da lista de rotas, e a quantidade de rotas k é atualizada.

O processo de realocação dos clientes das rotas de menor carga é repetido até que o número de rotas k seja igual ao número mínimo de veículos necessários "k_{min}" ou que não sejam mais possíveis realizar realocações de clientes devido às restrições de janela de tempo. Quando isso acontece, o procedimento termina e retorna as rotas otimizadas.

Algoritmo 12: *EmptyRoutes*

```
1  Procedimento EmptyRoutes(s, s_cg, dem, cap_v, dist)
2      k_min ← soma dem / cap_v
3      k ← tamanho s; i ← 0
4      enquanto k > k_min faça
5          CL ← ordena as rotas pelo tamanho de sua carga do menor para o maior
6          r_p ← i rota de menor carga
7          enquanto carga total de r_p ≠ 0 faça
8              c ← r_p [0]
9              s_, s_cg_ ← SplitReisertion(s, s_cg, cap_v, r_p, c)
10         se carga total r_p = 0, remova a rota de s
11         k ← k – 1; i ← i + 1
12     fim
13     retorne s, s_cg
14 fim
```

3.5.5. Perturbação

Soluções localmente ótimas são perturbadas pelo procedimento "*Multiple-K-Split"*, apresentado por Silva, Subramanian e Ochi (2015). Os autores compararam o procedimento com estruturas de vizinhança muito utilizadas para o PRV, como o "*Swap"* e "*Eject Point"*. Experimentos computacionais mostraram que o "*Multiple-k-Split"* é mais eficaz na geração de soluções diversificadas quando se considera o PRVEF, principalmente porque esse operador explora a característica de divisão do problema.

O "*Multiple-k-Split"*, descrito pelo **Algoritmo 13** seleciona um número "k" de clientes, identifica as rotas que estes clientes fazem parte e os realoca para novas rotas através do procedimento "*SplitReinsertion"*. Esse "k" número de clientes é o nível de perturbação do método e pode ser ajustado conforme o número de soluções sem melhoria no problema.

Algoritmo 13: *Multiple K-Split*

```
1  Procedimento Multiple-k-split(s, s_cg, dem, cap_v, dist, k)
2      CL ← escolha aleatoriamente k clientes
5      para c em CL faça
6          para i ← 0,..., tamanho s
7              se c estiver em s[i] faça s, s_cg ← SplitReisertion(s, s_cg, cap_v, s[i], c)
8          fim
9      fim
```

```
10 |   retorne s, s_cg
11 fim
```

4. Resultados

A presente seção apresenta os resultados obtidos a partir da implementação do algoritmo ILS (*Iterated Local Search*) aplicado aos dados reais do problema apresentados na seção 3.5.1. Os experimentos foram executados em um computador equipado com processador Intel® Core™ i3-8130U CPU @ 2.20GHz 2.21 GHz, 4 gigabytes de memória RAM e sistema operacional Windows™ 10 64 bits. Os algoritmos heurísticos e meta-heurísticos foram implementados na linguagem de programação Python 3.9.13. O modelo matemático apresentado também foi implementado utilizando a linguagem matemática AMPL™ (AMPL IDE versão 3.6.10) e o solver CPLEX® versão 22.1.1 para a resolução do problema.

Por se tratar de um problema NP-Difícil, instâncias muito grandes não podem ser resolvidas através do modelo matemático. Portanto, para tornar possível a avaliação da qualidade do método e também auxiliar no ajuste dos parâmetros, foram geradas 5 pequenas instâncias com dados gerados randomicamente através do método apresentado na seção 3.2, mantendo características semelhantes às informações reais do problema.

A avaliação das soluções foi conduzida tanto através do software AMPL IDE utilizando o solver CPLEX, como pelo algoritmo *Iterated Local Search*, implementado na linguagem Python. O ILS foi executado sob critérios predefinidos, sendo $I_{RVND} = 50$, $I_{max} = 50$ e $I_{ILS} = 100$. Em cada instância, o algoritmo ILS foi aplicado por dez vezes, com a melhor solução sendo selecionada para análise.

A As colunas intituladas "GAP (km)" e "GAP (%)" revelam as disparidades entre os custos determinados pelo algoritmo ILS e pelo modelo matemático. Na primeira coluna, a discrepância é expressa em quilômetros, enquanto na segunda, é evidenciada a proporção percentual desse GAP em relação ao custo obtido pela solução ótima.

Tabela 2 apresenta os resultados encontrados comparando estes dois métodos. A coluna “ID” identifica a instância utilizada. As colunas “Nº de clientes”, “Nº de veículos”, “Capacidade veículos” e “Demanda total” detalham o tamanho de cada uma das instâncias, representando respectivamente: o total de clientes de cada instância, total de veículos utilizados para solução, capacidade dos veículos da frota (considerando uma frota homogênea) e soma das demandas de todos os clientes da instância. As colunas “Tempo ILS (s)” e “Custo ILS (km)” representam o tempo gasto em segundos na implementação do método e custo total (soma das distâncias das rotas) em quilômetros da solução encontrada pelo algoritmo proposto apresentado na seção 3.5.1. As colunas “Tempo ótimo (s)” e “Custo ótimo (km)” apresentam o resultado encontrado

pelo modelo matemático apresentado na seção 3.5. As colunas intituladas "GAP (km)" e "GAP (%)" revelam as disparidades entre os custos determinados pelo algoritmo ILS e pelo modelo matemático. Na primeira coluna, a discrepância é expressa em quilômetros, enquanto na segunda, é evidenciada a proporção percentual desse GAP em relação ao custo obtido pela solução ótima.

Tabela 2: Resultados das instâncias geradas nos métodos ILS e método exato

ID	Nº de clientes	Nº de veículos	Capacidade veículos	Demanda total	Tempo ILS (s)	Custo ILS (km)	Tempo ótimo (s)	Custo ótimo (km)	GAP (km)	GAP (%)
1	5	6	15	83	16	769	3442	760	9	1,2%
2	5	4	20	67	14	503	219	501	2	0,4%
3	4	5	15	68	12	652	17	636	16	2,5%
4	5	7	15	98	12	886	208	883	3	0,3%
5	5	6	15	78	14	756	596	750	6	0,8%

Nas cinco instâncias analisadas, o algoritmo ILS não alcançou a solução ótima. No entanto, em três delas, a diferença em relação ao método exato foi inferior a 1%, um valor consideravelmente pequeno quando se leva em conta o custo total da solução. Além disso, é importante notar que, em termos de tempo de execução, a meta-heurística demonstrou um desempenho notável, com tempo médio de execução de 14 segundos e desvio padrão de 1,5 segundos.

Comparativamente, o método exato revelou uma performance mais lenta, levando mais de uma hora para finalizar em todos os cenários, exceto na instância com apenas quatro clientes. O tempo médio de execução do método exato foi de 896 segundos e desvio padrão de 1438,5 segundos.

Tendo em vista o custo das rotas otimizadas e o esforço necessário para aplicar as duas abordagens, a meta-heurística proposta se destaca como uma alternativa eficaz. Os resultados obtidos com o ILS se aproximam consideravelmente do ideal, ao mesmo tempo em que requer um tempo de implementação muito curto, oferecendo um equilíbrio entre a qualidade das soluções e a eficiência computacional.

Apesar de sua característica NP-Difícil, o modelo matemático também foi aplicado para resolução do problema real, mas mesmo após 10.000 segundos de execução, o método não obteve sucesso, não encontrando nenhuma solução viável. A partir destes resultados, aplicou-se o algoritmo ILS para resolução dos problemas do turno da manhã e noite. Para cada turno, o algoritmo foi aplicado por dez vezes para quatro combinações de parâmetros diferentes. A Tabela 3 apresenta os resultados obtidos através do custo médio por combinação e desvio

padrão. Nesta tabela, a coluna “ID” identifica a combinação dos parâmetros utilizada, as colunas “I_{RVND}”, “I_{max}” e “I_{ILS}” representam, respectivamente, o número de iterações sem do método RVND, o número de iterações do algoritmo ILS e o número máximo de iterações sem melhora após a busca local dentro do método ILS. As colunas “Tempo médio (s)” e “Desvio padrão tempo” apresentam, respectivamente, a média de tempo e desvio padrão em segundos das execuções do algoritmo ILS. Já as colunas “Custo médio (km)” e “Desvio padrão custo” apresentam a média de custo encontrado pelas soluções geradas pelo método ILS e seu desvio padrão em quilômetros.

Tabela 3: Resultado do algoritmo ILS no turno da manhã considerando o custo médio

ID	I_{RVND}	I_{max}	I_{ILS}	Tempo médio (s)	Desvio padrão tempo	Custo médio (km)	Desvio padrão custo
1	100	50	100	22,95	0,91	2010,10	36,35
2	100	100	100	45,27	2,59	2005,70	31,97
3	50	100	100	77,68	38,13	2013,10	6,12
4	100	100	50	37,74	18,90	1966,50	45,98

Percebe-se que a combinação 4, com I_{RVND} = 100, I_{max} = 100 e I_{ILS} = 50 apresentou o menor custo médio, seguida da combinação 2, com parâmetros I_{RVND} = 100, I_{max} = 100 e I_{ILS} = 100.

Dentre as 40 aplicações do método, a melhor solução encontrada para cada combinação de parâmetros é descrita na Tabela 4. Na tabela, a coluna “Tempo (s)” apresenta o tempo de execução da solução de menor custo encontrada para cada combinação de parâmetros e a coluna “Custo (km)” representa o menor custo encontrado pelas soluções geradas com a combinação de parâmetros referida pelas quatro primeiras colunas.

Tabela 4: Resultados do algoritmo ILS no turno da manhã considerando o menor custo total

ID	I_{RVND}	I_{max}	I_{ILS}	Tempo (s)	Custo (km)
1	100	50	100	22,32	1910
2	100	100	100	44,49	1916
3	50	100	100	45,18	2006
4	100	100	50	66,19	1910

Percebe-se que o menor custo encontrado é de 1910 km. Essa solução foi encontrada por duas vezes, utilizando a combinação de parâmetros 1 (22,32 segundos) e a combinação de parâmetros 4 (66,19 segundos), sendo a combinação 1 mais eficiente dado o menor tempo de execução.

Esta solução classificada como mais eficaz resultou em 15 rotas de custo médio 127,33 km e desvio padrão de 12,20 km. A Tabela 5 detalha as rotas encontradas pela solução. Na

tabela, a coluna “Rota” apresenta o número da rota e a coluna “Descrição da Rota” o percurso realizado. Neste caso, o índice “0” representa a origem e os demais índices, representam os clientes (conforme índices apresentados na Tabela 1). A coluna “Nº de clientes atendidos” destaca o número de clientes atendidos por rota e a coluna “Distância Percorrida”, a distância total da rota em quilômetros. Por fim, a coluna “Ocupação do veículo” apresenta o percentual de ocupação do veículo, que é a relação entre total alunos atendidos pela rota e a capacidade do veículo.

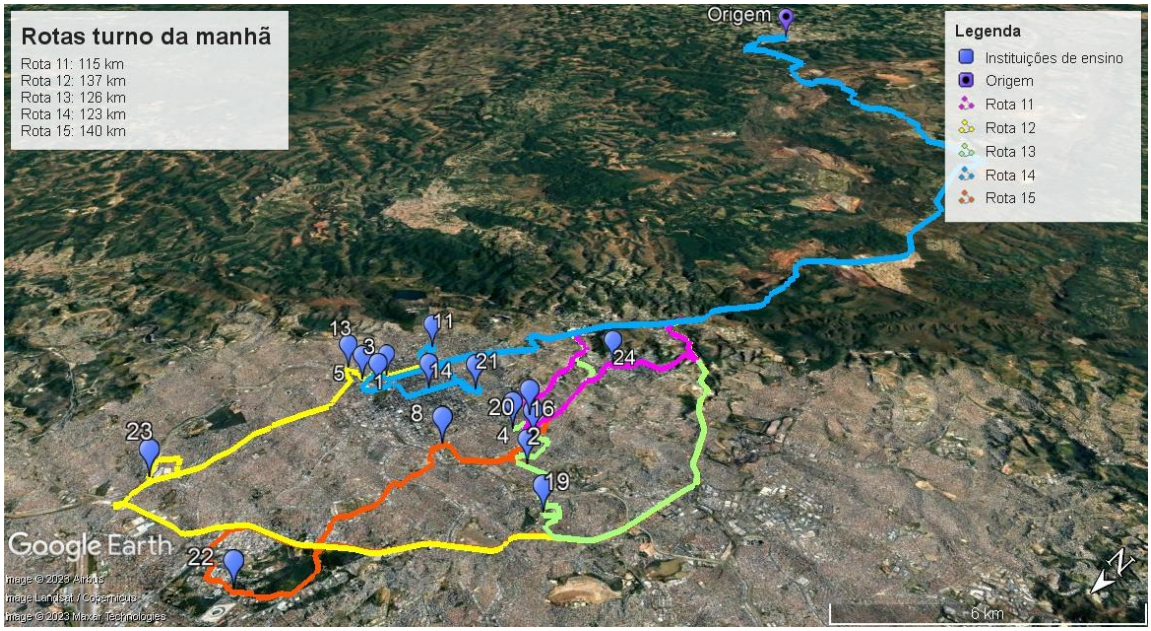
Tabela 5: Detalhamento da melhor solução encontrada para o turno da manhã

Rota	Descrição da Rota	Nº de clientes atendidos	Distância Percorrida (km)	Ocupação do veículo
1	0 – 19 - 0	1	122	100%
2	0 – 16 - 0	1	114	100%
3	0 – 24 - 0	1	106	100%
4	0 – 24 - 0	1	106	100%
5	0 – 22 - 0	1	137	100%
6	0 – 22 - 0	1	137	100%
7	0 – 22 - 0	1	137	100%
8	0 – 22 - 0	1	137	100%
9	0 – 22 - 0	1	137	100%
10	0 – 17 - 0	1	136	100%
11	0 – 24 – 16 – 0	2	115	100%
12	0 – 1 – 5 – 13 – 23 – 0	4	137	100%
13	0 – 4 – 16 – 20 – 19 – 0	4	126	100%
14	0 – 11 – 21 – 14 – 3 – 0	4	123	93%
15	0 – 22 – 8 – 2 – 0	3	140	100%

Percebe-se que as 10 primeiras rotas atendem a somente um cliente. Este tipo de rota, chamado de “rota única” é possível quando a demanda de uma instituição de ensino é maior que a capacidade de um veículo, assim, é possível criar rotas que ocupem 100% do veículo.

Por outro lado, as 5 últimas rotas visitam mais de uma instituição de ensino. Estas rotas, também chamadas de “rotas compartilhadas” atendem aos clientes que não possuem demanda maior que a capacidade de um veículo, e portanto, não podem realizar uma rota única, ou aos clientes que mesmo após a construção de rotas únicas, ainda possuem alguma demanda residual a ser atendida. As rotas compartilhadas estão detalhadas também na Figura 4, que apresenta a distribuição geográfica das Instituições de Ensino.

Figura 4: Rotas com destino a mais de uma Instituição de Ensino no turno da manhã



Fonte: elaborado pela autora em Google Earth™

Para o turno da noite, o mesmo procedimento da manhã foi adotado. Os resultados estão detalhados na Tabela 6. Neste caso, a combinação de menor custo médio foi a 1, com parâmetros I_{RVND} = 100, I_{max} = 50 e I_{ILS} = 100.

Tabela 6: Resultados do algoritmo ILS para o turno da noite considerando o custo médio

ID	I_{RVND}	I_{max}	I_{ILS}	Tempo médio (s)	Desvio padrão tempo	Custo médio (km)	Desvio padrão custo
1	100	50	100	28,77	0,56	2837,60	8,72
2	100	100	100	57,43	1,66	2838,70	14,58
3	50	100	100	58,41	3,19	2840,40	11,01
4	100	100	50	29,49	1,06	2842,50	13,01

Apesar da combinação 1 apresentar o menor custo médio, a melhor solução dentre todas as aplicações do método foi encontrada com os parâmetros da combinação 2 (I_{RVND} = 100, I_{max} = 100 e I_{ILS} = 100), representando um custo total de 2822 km e tempo de implementação 54,28 segundos. A Tabela 7 detalha as melhores soluções encontradas para cada combinação de parâmetros.

Tabela 7: Resultados do algoritmo ILS no turno da noite considerando o menor custo total

ID	I_{RVND}	I_{max}	I_{ILS}	Tempo (s)	Custo (km)
1	100	50	100	29,12	2828
2	100	100	100	54,28	2822
3	50	100	100	56,69	2823
4	100	100	50	29,20	2825

A melhor solução encontrada, que representa um custo total de 2822 km, resultou em 23 rotas de custo médio 123,57 km e desvio padrão de 12,57 km. A Tabela 8 detalha as rotas encontradas por essa solução.

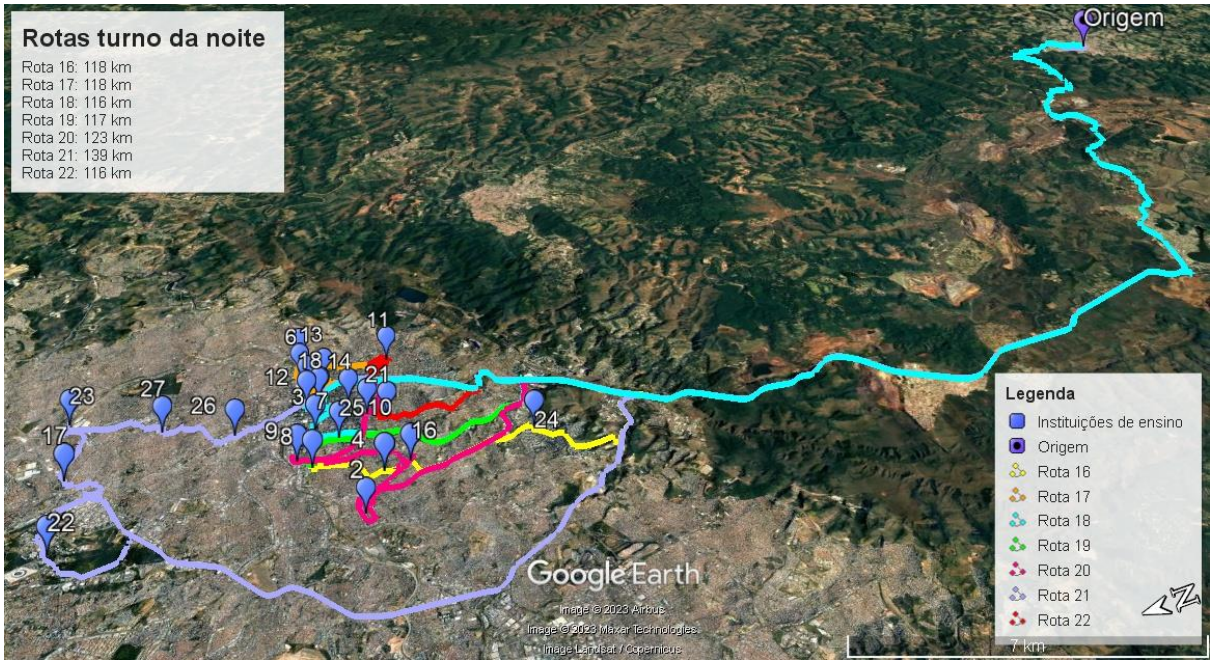
Tabela 8: Detalhamento da melhor solução encontrada para o turno da noite

Rota	Descrição da Rota	Nº de clientes atendidos	Distância Percorrida (km)	Ocupação do veículo
1	0 – 11 – 0	1	133	100%
2	0 – 16 – 0	1	114	100%
3	0 – 16 – 0	1	114	100%
4	0 – 17 – 0	1	136	100%
5	0 – 17 – 0	1	136	100%
6	0 – 17 – 0	1	136	100%
7	0 – 19 – 0	1	122	100%
8	0 – 22 – 0	1	137	100%
9	0 – 22 – 0	1	137	100%
10	0 – 23 – 0	1	142	100%
11	0 – 23 – 0	1	142	100%
12	0 – 24 – 0	1	106	100%
13	0 – 24 – 0	1	106	100%
14	0 – 24 – 0	1	106	100%
15	0 – 24 – 0	1	106	100%
16	0 – 24 – 4 – 8 – 0	3	118	100%
17	0 – 6 – 13 – 3 – 1 – 0	4	118	100%
18	0 – 7 – 14 – 12 – 15 – 18 – 0	5	116	100%
19	0 – 25 – 8 – 9 – 0	3	117	100%
20	0 – 10 – 9 – 16 – 2 – 0	4	123	100%
21	0 – 26 – 27 – 23 – 17 – 22 – 0	5	139	100%
22	0 – 11 – 21 – 0	2	116	47%
23	0 – 19 – 0	1	122	67%

As 15 primeiras rotas representam rotas únicas, enquanto as 7 seguintes, representam as rotas compartilhadas.

As rotas compartilhadas do turno da noite estão detalhadas também na Figura 5, que apresenta a distribuição geográfica das Instituições de Ensino.

Figura 5: Rotas com destino a mais de uma Instituição de Ensino no turno da noite



Fonte: elaborado pela autora em Google Earth™

No turno da noite, percebe-se que as duas últimas rotas possuem uma taxa de ocupação consideravelmente menor que as demais, que estão com 100% de utilização. Isto ocorre devido à demanda total do problema. No turno da noite, 332 alunos demandam transporte, o que representaria uma necessidade de 22,13 veículos. Como o número mínimo de veículos "k_{min}" deve ser inteiro, considera-se 23 veículos, o que somam 345 vagas (13 a mais do que o necessário para o atendimento da demanda). Estas vagas sem utilização, fazem com que a taxa média de ocupação dos veículos seja de 96,23%.

Como alternativa para aumentar a ocupação média dos veículos e otimizar o sistema, a Secretaria de Transportes pode, por exemplo, alterar a configuração da frota, substituindo alguns veículos pequenos, por outros de capacidade maior, de tal forma que se consiga alcançar a capacidade de 332 lugares com menos veículos. Este é apenas um exemplo de como os resultados da aplicação do algoritmo ILS podem auxiliar na tomada de decisão da Prefeitura quanto à prestação do serviço de transporte escolar. As soluções encontradas pelo método devem servir como *input* para definição de novas estratégias para melhorar a qualidade do sistema de transportes.

5. Conclusão

Este estudo se concentrou na concepção de um método para abordar o desafio da roteirização do transporte escolar dentro da esfera pública. A relevância deste tema reside na necessidade de encontrar soluções que tornem o transporte escolar mais eficiente reduzindo gastos públicos e viabilizando a expansão do serviço.

Com base em uma análise dos diversos métodos e variações do Problema de Roteamento de Veículos Escolares, o presente trabalho empregou princípios e técnicas da Pesquisa Operacional, como o conceito de meta-heurística para resolução do problema. No âmbito dessa abordagem, foi utilizada uma meta-heurística denominada "*Iterated Local Search* (ILS)", que se mostrou promissora na resolução do problema em questão.

O algoritmo foi submetido a testes em cinco instâncias de pequena escala, variando de quatro a cinco clientes, que foram geradas de maneira aleatória. Esses testes indicaram a eficácia do método proposto, especialmente em instâncias que permitiram a aplicação de métodos exatos de resolução.

No contexto de futuros desdobramentos, existe a oportunidade de ampliar ainda mais a abrangência do problema, considerando novas variações do Problema de Roteamento de Veículos Escolares, como definição de pontos de ônibus, tempo de viagem máximo, além de abordagens não determinísticas. Essa expansão poderia proporcionar um enfoque mais realista e adaptado às situações cotidianas de estudantes e profissionais envolvidos no gerenciamento do transporte escolar. A adição dessas variações poderia enriquecer o campo de estudo e oferecer soluções ainda mais relevantes para os desafios operacionais enfrentados por essas empresas.

Bibliografia

AGUIAR, Marcelo; MAURI, Geraldo R.; SILVA, Rodrigo F. **Introdução aos Métodos Heurísticos de Otimização com Python**. 1. ed. Alegre, ES: CAUFES, 2018.

ALEMAN, Rafael; ZHANG, Xinhui; HILL, Raymond R. An adaptive memory algorithm for the split delivery. **Journal of Heuristics**, v. 16, p. 441- 473, 2010.

ALVES, Fernando S. Problema de roteamento de veículos aplicados no planejamento logístico do transporte escolar da cidade de Coxim - MS. **Dissertação (Mestrado em Matemática Aplicada e Computacional) - Universidade Estadual de Campinas**, Campinas, 2015.

ARCHETTI, Claudia; SPERANZA, Maria G. The Split Delivery Vehicle Routing Problem: A survey. **The vehicle routing problem: latest advances and new challenges**, Nova Iorque, n. Springer, p. 103-122, 2008.

BELFIORE, Patrícia; FÁVERO, Luiz P. **Pesquisa Operacional - Para Cursos de Engenharia**. Rio de Janeiro: Elsevier, 2013.

BELFIORE, Patrícia; YOSHIZAKI, Hugo T. Y. Heuristic methods for the fleet size and mix vehicle routing problem. **Computers & Industrial Engineering**, v. 64, 2013.

BLUM, Christian; ROLI, Andrea. Metaheuristics in Combinatorial Optimization: Overview. **ACM Computing Surveys**, v. 35, n. 3, p. 268 - 308, 2003.

BOUDIA, Mourad; PRINS, Christian; REGHIOUI, Mohamed. An Effective Memetic Algorithm with Population Management for Split Delivery Vehicle Routing Problem. In: BARTZ-BEIELSTEN, Thomas, *et al.* **Hybrid Metaheuristics LNCS**. [S.l.]: Springer, v. 4771, 2007. p. 16 - 30.

BRASIL. **Censo da Educação Superior 2021**. Diretoria de Estatísticas Educacionais, Ministério da Educação. Brasília. 2022.

CAMPOS, Guilherme G.; YOSHIZAKI, Hugo T. Y.; BELFIORE, Patrícia P. Algoritmos Genéticos e Computação Paralela para Problemas de Roteirização de Veículos com Janelas de Tempo e Entregas Fracionadas. **Gestão & Produção**, v. 13, n. 2, p. 271 - 281, 2006.

COLIN, Emerson C. **Pesquisa Operacional - 170 Aplicações em Estratégia, Finanças, Logística, Produção, Marketing e Vendas**. 2ª. ed. São Paulo: Grupo GEN, 2018.

DROR, Moshe; TRUDEAU, Pierre. Savings by Split Delivery Routing. **Transportation Science**, v. 23, n. 2, p. 141-145, 1989.

FEILLET, Dominique *et al.* Vehicle Routing with Time Windows and Split, 2002.

FISHER, Marshal L.; JAIKUMAR, Ramchandran. A generalized assignment heuristic for vehicle routing. **Networks**, v. 11, p. 109-124, 1981.

GOLDBARG, Elizabeth; GOLDBARG, Marco Cesar; LUNA, Henrique P. L. **Otimização Combinatória e Meta-heurísticas - Algoritmos e Aplicações**. 1ª. ed. Rio de Janeiro: Elsevier, 2016.

HILLIER, Frederick S.; LIEBERMAN, Gerald J. **Introdução à Pesquisa Operacional**. 9ª. ed. Porto Alegre: Grupo A, 2010.

HO, S. C.; HAUGLAND, D. A tabu search heuristic for the vehicle routing problem with. **Computers & Operations Research**, v. 31, p. 1947 – 1964, 2004.

LOURENÇO, Helena R.; MARTIN, Olivier C.; STÜTZLE, Thomas. Iterated Local Search: Framework. In: GENDREAU, Michel; POTVIN, Jean-Yves **Handbook of Metaheuristics**. 3ª. ed. [S.l.]: [s.n.], 2018. Cap. 5.

MCNABB, Marcus E. *et al.* Testing local search move operators on the vehicle routing problem. **Computers & Operations Research**, v. 56, p. 93 - 109, 2015.

MELO, Igor E. S. Modelagem matemática e algoritmos heurísticos para o problema do roteamento de ônibus escolares envolvendo múltiplos períodos. **Dissertação (Mestrado em Engenharia de Produção) - Universidade Federal de Pernambuco**, 2022.

PARK, Junhyuk; KIM, Byung-In. The school bus routing problem: A review. **European Journal of Operational Research**, v. 202, p. 311 - 319, 2010.

PENNA, Puca H. V.; SUBRAMANIAN, Anand; OCHI, Luiz S. An Iterated Local Search heuristic for the. **Journal of Heuristics**, v. 19, n. 2, p. 201–232, 2011.

SILVA, Marcos M.; SUBRAMANIAN, Anand; OCHI, Luiz S. An iterated local search heuristic for the split delivery vehicle. **Computers & Operations Research**, v. 53, p. 234 - 249, 2015.

SOUZA, Marcone J. F. **Inteligência Computacional para Otimização**. Departamento de Computação, Instituto de Ciências Exatas e Biológicas, Universidade Federal de Ouro Preto. Ouro Preto. 2022.

SPADA, M.; BIERLAIRE, M.; LIEBLING, Th. M. Decision-Aiding Methodology for the School Bus. **Transportation Science**, v. 39, n. 4, p. 477 - 490, 2005.

SPELLER, Paulo; ROBL, Fabiane; MENEGHEL, Stela M. **Desafios e perspectivas da educação superior brasileira para a próxima década**. UNESCO, CNE, MEC. Brasília. 2012.

TOTH, Paolo; VIGO, Daniele. The Vehicle Routing Problem. **Siam**, 2002.