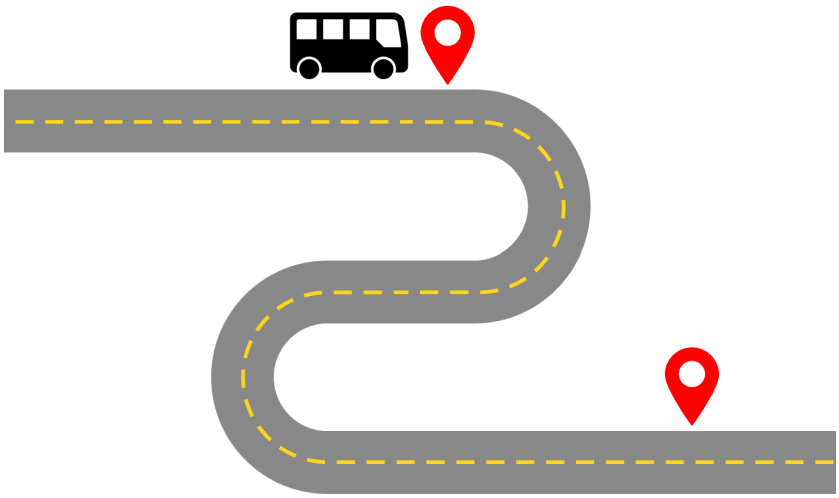


Planeamento de Afetação de Veículos de Transporte Público

CATARINA JIN
(Licenciada)

Relatório de Estágio para obtenção do grau de Mestre em Matemática Aplicada para a Indústria, na Área de Especialização em Tratamento de Dados

Orientadores:

Doutor José Firmino Aguilar Madeira

Doutora Teresa Maria de Araújo Melo Quinteiro

Júri:

Presidente: Doutor Luís Manuel Ferreira da Silva

Vogais:

Doutora Iola Maria Silvério Pinto

Doutor José Firmino Aguilar Madeira

setembro de 2024

Planeamento de Afetação de Veículos de Transporte Público

CATARINA JIN
(Licenciada)

Relatório de Estágio para obtenção do grau de Mestre em Matemática Aplicada para a Indústria, na Área de Especialização em Tratamento de Dados

Orientadores:
Doutor José Firmino Aguilar Madeira, ISEL
Doutora Teresa Maria de Araújo Melo Quinteiro, ISEL

Júri:
Presidente: Doutor Luís Manuel Ferreira da Silva, ISEL
Vogais:
Doutora Iola Maria Silvério Pinto, ISEL
Doutor José Firmino Aguilar Madeira, ISEL

setembro de 2024

Agradecimentos

Este estágio realizou-se, através do protocolo de cooperação com o ISEL, na empresa Tecnologias de Microeletrónica, sob a supervisão local do Engenheiro António Marcelo.

Gostaria de expressar minha sincera gratidão aos meus orientadores, sem os quais este projeto não teria sido possível. Agradeço ao professor José Madeira por me guiar nas técnicas de otimização, proporcionando perceções valiosas e direções claras ao longo do processo. À professora Teresa Quinteiro, meu agradecimento pelo apoio constante, motivação e pelas correções que contribuíram para o meu relatório coerente. O engenheiro António Marcelo da Tecmic foi fundamental na minha compreensão do problema apresentado pela empresa, e sou muito grato por sua ajuda e orientação.

Aproveito também para agradecer aos meus colegas de turma pelo apoio psicológico que foi essencial durante esta jornada. A colaboração e amizade entre nós tornaram este percurso mais leve e significativo.

Obrigado a todos!

Declaração de integridade

Declaro que esta(e) dissertação / trabalho de projeto / relatório de estágio é o resultado da minha investigação pessoal e independente. O seu conteúdo é original e todas as fontes listadas nas referências bibliográficas foram consultadas e estão devidamente mencionadas no texto. Mais declaro que todas as referências científicas e técnicas relevantes para o desenvolvimento do trabalho estão devidamente citadas e constam das referências bibliográficas.

O autor

Catarina Jin

Lisboa, 26 de 11 de 24

Resumo

O presente relatório aborda o problema apresentado pela Tecmic, empresa com a qual elaborei no âmbito de um estágio curricular no semestre de verão de 2023/24. Este projeto trata o problema de planeamento da afetação de veículos de transporte público, com o objetivo de minimizar o número de veículos necessários para cumprir os horários predefinidos de diversas rotas. Através da utilização de dados reais fornecidos pela Carris, e da aplicação de técnicas de otimização como o *Simulated Annealing* e o algoritmo NSGA-II, com o apoio computacional de Python, foi possível propor soluções boas que equilibram múltiplos critérios, tais como a redução do número de veículos e a melhoria da distribuição de tarefas.

Para este fim, foram explorados diferentes cenários de atribuição de viagens com tempos de pausa variáveis, simulando situações reais. A implementação dos algoritmos permitiu otimizar as soluções minimizando o tempo de inatividade, bem como o equilíbrio do número de viagens atribuídas aos veículos. O projeto apresenta a importância da sincronização de horários e da distribuição equilibrada de veículos entre as rotas. Os resultados obtidos mostram que as técnicas aplicadas conseguem reduzir o número de veículos em circulação, ao mesmo tempo que garantem o cumprimento rigoroso dos horários.

Palavras-chave

Afetação de Veículos; GTFS; Atribuição de Viagens; Recozimento Simulado; NSGA-II

Abstract

This report addresses the issue presented by Tecmic, the company where I interned during the summer semester of 2023/24. This project addresses the problem of planning the allocation of public transport vehicles, with the aim to minimizing the number of vehicles needed to meet the predefined schedules of various routes. Through the use of real data provided by Carris and applying optimization techniques such as Simulated Annealing and the NSGA-II algorithm, with the computacional support of Python, it was possible to propose good solutions that balance multiple criteria, such as reducing the number of vehicles and improving task distribution.

For this purpose, different trip assignment scenarios with variable break times were explored, simulating real operations. The implementation of the algorithms allowed us to optimize solutions by minimizing downtime, as well as balancing tasks among vehicles. The project presents the importance of synchronizing schedule and balanced distribution of vehicles between routes. The results obtained show that the techniques applied can reduce the number of vehicles in circulation, while ensuring strict compliance with schedules.

Keywords

Vehicle Allocation; GTFS; Trip Assignment; Simulated Annealing; NSGA-II

Índice

Índice de figuras

Índice de tabelas

Índice de algoritmos

Simbologia e abreviaturas

Latinas

f função objetivo

d distância de aglomeração

Abreviaturas

GTFS	*General Transport Feed Specification*
SA	*Simulated Annealing*
NSGA	*Non-dominated Sorting Genetic Algorithm*
SBX	*Simulated Binary Crossover*
PM	*Polynomial Mutation*

Capítulo 1

Introdução

Com o avanço constante das cidades, e o impulso pelo crescimento populacional e pela urbanização acelerada, surgiram novos desafios para a gestão e a organização dos espaços urbanos. As metrópoles modernas, que atualmente abrigam milhões de pessoas, exigem soluções cada vez mais complexas para garantir a mobilidade eficiente dos seus habitantes. Nesse contexto, os transportes públicos desempenham um papel fundamental, não apenas como um meio da deslocação sustentável, mas também como uma resposta aos problemas do congestionamento e poluição.

Este trabalho tem como objetivo explorar o tema do planeamento de afetação de veículos, abordando as metodologias e ferramentas utilizadas para otimizar o sistema dos transportes públicos.

1.1 Objetivo do Trabalho

O problema proposto no estágio centra-se na otimização da utilização do veículo do transporte público, com o objetivo de minimizar o número total de veículos necessários para cumprir os horários predefinidos de partida e chegada nos diferentes percursos. Esta tarefa envolve a análise detalhada dos dados relativos aos horários, com o intuito de desenvolver um plano da afetação eficiente.

1.2 Contexto

A eficiência dos transportes públicos é indispensável para o funcionamento equilibrado de qualquer cidade. Os sistemas do transporte bem planeados permitem reduzir o tráfego, diminuir as emissões de gases de efeito estufa e proporcionar uma alternativa económica e acessível aos cidadãos. No entanto, para que esses sistemas funcionem de forma otimizada, é imprescindível um planeamento cuidadoso e estratégico. A afetação adequada dos veículos, isto é, a distribuição e gestão eficiente da frota de transportes, desempenha um papel crucial para assegurar que a oferta se adequa à procura e melhorar a qualidade de vida nas cidades.

A sincronização precisa dos horários garante que os passageiros possam planear as

suas viagens de forma confiável, ao mesmo tempo minimizar o tempo de espera nas paragens e evitar sobrecargas nos veículos. Uma gestão adequada dos horários também contribui para a otimização da frota, que assegura o número dos veículos em operação seja suficiente para atender à demanda, sem desperdício dos recursos.

Além disso, uma boa gestão dos horários permite uma maior previsibilidade do serviço, o que é fundamental para aumentar a satisfação dos utilizadores e promover a utilização do transporte público em desfavor do transporte individual. Quando os horários são bem planeados, a eficácia operacional melhora, reduzindo os custos em combustível, a manutenção e as horas do trabalho. O impacto ambiental diminui devido à maior necessidade dos transportes públicos em circulação.

No entanto, a atribuição eficiente dos horários apresenta desafios significativos, especialmente quando o objetivo é minimizar o número dos veículos utilizados e, simultaneamente, os tempos de espera dos passageiros. O principal desafio deste trabalho reside na necessidade de conciliar horários de partida e chegada, considerando as variações na demanda ao longo do dia, bem como as limitações operacionais, como o tempo de pausa necessário entre viagens e as condições do tráfego variáveis.

1.3 Tecmic - Tecnologias de Microeletrónica S.A.

No âmbito do meu estágio, tive a oportunidade de colaborar com a Tecmic S.A., uma empresa multinacional portuguesa fundada em 1988. A empresa atua: nos principais setores da atividade económica as soluções inteligentes de comando, controlo, gestão de frotas e recursos, equipas e operações, meios e ocorrências, com grandes vantagens na redução de custos e no sucesso dos seus clientes (1). Destaca-se como líder nos sistemas destinados a grandes empresas e corporações, com elevada qualificação das equipas comerciais, desenvolvimento, engenharia e assistência técnica. Os distribuidores e parceiros internacionais espalham-se pelo mundo, principalmente na América, Europa e África.

A Tecmic, com sua abordagem inovadora e compromisso com o desenvolvimento de soluções que transcendem as expetativas, desempenha um papel fundamental na otimização de processos e na preparação de empresas para enfrentar os desafios competitivos nos mercados mais difíceis. A empresa foca as suas competências em:

- Soluções de mobilidade e gestão de equipas: planeamento, otimização, gestão de ativos e equipas;
- Gestão integrada da recolha resíduos: monitorização remota do nível dos contentores de resíduos;
- Gestão de frotas e localização GPS;
- Sistemas de comando e controlo de forças de segurança e emergência: visão global do terreno e intervenção completa;

- Sistemas de gestão de transportes públicos de passageiros: informação ao público em tempo real;
- Sistemas de segurança eletrónica: registo de assiduidades e segurança de entradas e saídas.

1.4 Estrutura do relatório

Este relatório está dividido em três partes principais:

- Descrição do Problema e Objetivo: A primeira parte apresenta uma descrição detalhada do problema em questão, bem como os objetivos do trabalho. Inclui também o pré-processamento dos dados, a abordagem básica para a atribuição das viagens aos veículos e analisa diferentes cenários das rotas. Nesta secção, é discutido a implementação das estratégias de atribuição e apresentam-se os resultados obtidos em diferentes casos, considerando diversas condições iniciais.
- Primeira Estratégia de Otimização – *Simulated Annealing*: A segunda parte foca na primeira estratégia de otimização, que utiliza o algoritmo do recozimento simulado. Este método baseia-se na atribuição das viagens a partir da primeira viagem disponível e realiza a otimização por meio de permutações nas ordens dos veículos. Neste capítulo, é detalhada a implementação do algoritmo e apresentam-se os resultados de diferentes cenários.
- Segunda Estratégia de Otimização – *NSGA-II*: A terceira parte aborda a segunda estratégia de otimização, que se concentra na técnica de múltiplos objetivos. Utilizamos o algoritmo NSGA-II para identificar várias soluções que podem ser contraditórias entre si. Esta secção também inclui a implementação do programa e os resultados obtidos.

Capítulo 2

Metodologia

À medida que os sistemas de transporte evoluem, as exigências operacionais tornam-se mais sofisticadas. A combinação dos modelos de otimização com algoritmos avançados permite não só minimizar o número dos veículos, como também melhorar a qualidade do serviço.

Assim, surge a otimização multi-objetivo, uma abordagem que visa equilibrar múltiplos critérios do desempenho simultaneamente. A otimização multi-objetivo permite lidar com os conflitos existentes entre diferentes objetivos. Esta abordagem é particularmente importante porque melhorar um objetivo pode piorar outro.

Ao integrar diferentes dimensões num único modelo de otimização, é possível criar soluções mais robustas e flexíveis, que atendam às necessidades operacionais e às restrições práticas do sistema. A utilização dos algoritmos avançados, como algoritmos genéticos, programação linear multi-objetivo, ou métodos heurísticos, facilita a exploração de soluções complexas, permitindo que as decisões do planeamento sejam tomadas com base em compromissos ótimos entre os vários objetivos.

2.1 Descrição do problema

Neste trabalho, aborda-se a otimização da afetação dos veículos num sistema do transporte público, onde o objetivo é minimizar o número dos veículos necessários para cumprir um conjunto predefinido dos horários de viagens entre duas paragens terminais, designadas por A e B. As viagens realizam-se nos dois sentidos: da paragem A para a paragem B (direção $A-B$) e da paragem B para a paragem A (direção $B-A$). O desafio consiste em garantir que todas as viagens sejam realizadas de acordo com os horários estipulados, utilizando o menor número possível dos veículos.

A premissa central é, após a conclusão de uma viagem, o veículo necessita de um tempo de pausa antes de iniciar a próxima viagem na direção oposta. Este intervalo, denominado exatamente "tempo de pausa", permite compensar os atrasos e preparar para a viagem subsequente. O tempo de pausa é, portanto, um requisito essencial para garantir a continuidade e a segurança do serviço de transporte.

Este trabalho procura, assim, fornecer uma solução computacional que permita atingir

este objetivo, com base nos pressupostos e nas condições operacionais definidas, garantindo que o serviço seja prestado de forma eficaz e com o mínimo dos veículos necessários.

2.1.1 Caso exemplo

Para ilustrar o problema, considera-se um cenário com horários predefinidos em ambas as direções. As partidas na direção $A - B$ ocorrem de 10 em 10 minutos, começando às $7:00$ e as partidas na direção oposta $B - A$ seguem um padrão idêntico.

Para exemplificar o modelo, considera-se o seguinte conjunto simplificado dos horários, representados na Figura 2.1:

- **Partidas na direção A-B:** 7:00, 7:10, 7:20, 7:30, 7:40, 7:50, 8:00...
- **Partidas na direção B-A:** 7:00, 7:10, 7:20, 7:30, 7:40, 7:50, 8:00...

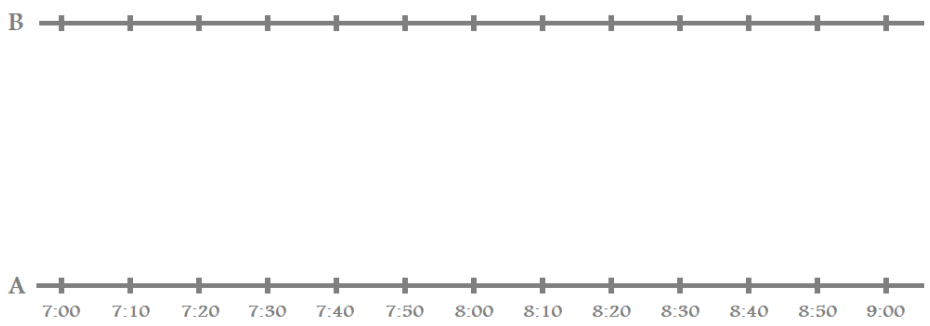


Figura 2.1 Gráfico do caso exemplo com os horários.

Assumindo um tempo de viagem de 50 minutos entre as paragens A e B, os horários da chegada à paragem terminal oposta seriam:

- **Chegadas na direção A-B:** 7:50, 8:00, 8:10, 8:20, 8:30, 8:40, 8:50...
- **Chegadas na direção B-A:** 7:50, 8:00, 8:10, 8:20, 8:30, 8:40, 8:50...

Tornando-se a viagem das $7:00$, o veículo que parte do ponto A chegará ao ponto B às $7:50$, como ilustrado no gráfico da Figura 2.2.

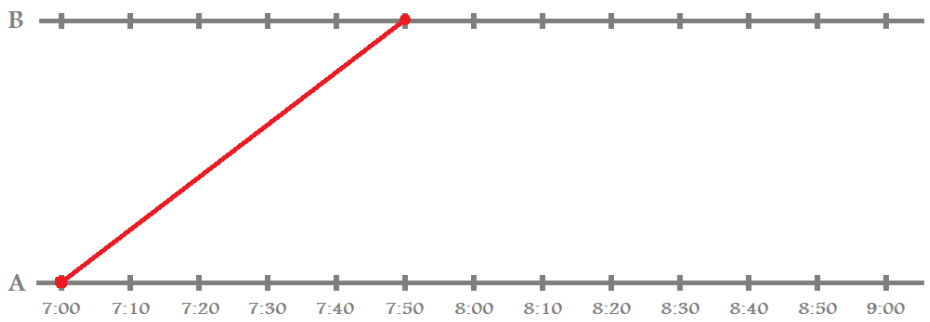


Figura 2.2 Gráfico do caso exemplo com os horários e a atribuição da primeira viagem.

Define-se um tempo de pausa de 10 minutos após a chegada do veículo à paragem terminal. Assim, o tempo total necessário para completar uma viagem (incluindo a pausa) será de $50 + 10 = 60$ minutos. A atribuição das viagens aos veículos pode ser efetuada de forma eficiente, desde que o horário da partida de uma viagem, somado ao tempo total da viagem, seja inferior ao horário da partida da viagem seguinte na direção oposta.

Por exemplo, se um veículo realiza a viagem das $7:00$ na direção $A - B$, ele estará disponível para realizar as viagens a partir das $8:00$ na direção oposta ($B - A$). Neste exemplo, realiza a viagem das $8:00$ seguida, tal como representado na Figura 2.3 para a primeira viagem das $7:00$.

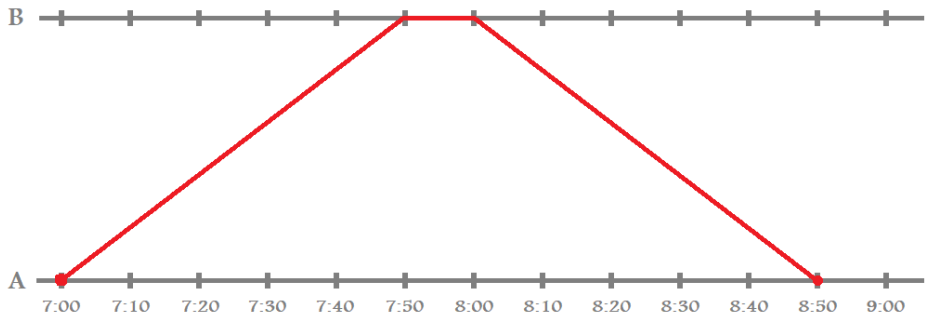


Figura 2.3 Gráfico do caso exemplo com os horários, a atribuição da primeira viagem e a pausa.

No gráfico seguinte estão representados as atribuições de todas as viagens entre as $7:00$ a $9:00$. Cada cor corresponde a um veículo. Tem-se então neste período 4 veículos com 2 viagens e 8 veículos com 1 viagem.

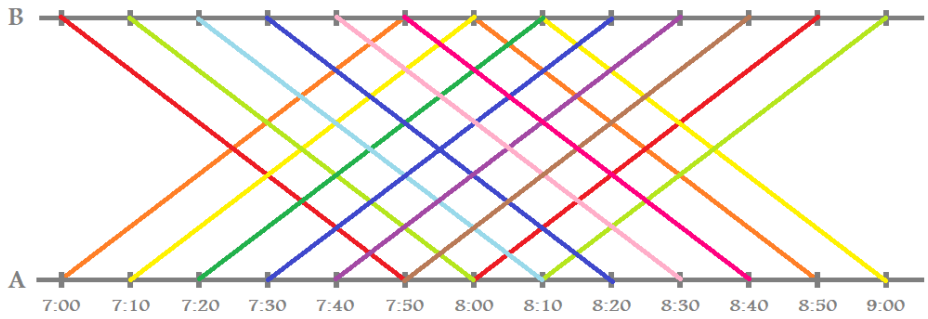


Figura 2.4 Gráfico do caso exemplo com os horários e a atribuição de todas as viagens.

Apesar de este exemplo ilustrar a ideia básica do problema, ele simplifica a realidade, onde os horários de partida e chegada raramente são tão regulares e previsíveis. Na prática, a variabilidade nos horários, tempos da viagem, e até mesmo imprevistos operacionais, tornam o problema de planeamento mais complexo. Este trabalho procura assim fornecer uma solução computacional que permita atingir o objetivo, ao mesmo tempo garantir que o serviço seja prestado com o mínimo dos veículos necessários.

2.1.2 Conceitos chave

No contexto deste relatório, é fundamental definir e compreender os seguintes termos, que são essenciais para a análise e o desenvolvimento do planeamento da afetação de veículos:

Rota: Uma rota refere-se a um percurso específico que é apresentado aos passageiros. Este conceito inclui um conjunto de viagens que seguem um trajeto comum, ligando diferentes pontos (paragens) ao longo de um corredor de transporte. Na Figura 2.5 está exemplificada o mapa geográfico da rota `60_0`, do Cais de Sodré ao Linda-a-Velha.

Viagem: Uma viagem é definida como a sequência de movimento de um veículo entre duas ou mais paragens ao longo de uma rota, ocorrendo num período de tempo específico. Cada viagem é caracterizada por um horário da partida e da chegada, assim como pela direção da viagem, e faz parte de um serviço contínuo ao longo do dia. No contexto operacional, as viagens precisam ser cuidadosamente planeadas e atribuídas aos veículos disponíveis para garantir um serviço contínuo.

Serviço: O termo serviço refere-se ao tipo da operação realizada num dia específico, o que pode variar em função do calendário (por exemplo, dias úteis, fins de semana, feriados). Diferentes tipos do serviço podem ter diferentes níveis de frequência e cobertura, ajustando-se às necessidades dos utilizadores em diferentes dias ou horários.

Direção: refere-se ao sentido que uma viagem ocorre entre duas paragens terminais, designadas por *A* e *B* por exemplo. Para simplificar a representação das direções no modelo, são utilizados os valores binários, onde `0` e `1` indicam direções opostas. Por exemplo, o valor `0`

pode representar viagens na direção $A - B$, enquanto o valor 1 representa viagens na direção $B - A$.

Tempo de Pausa: O tempo de pausa é o intervalo mínimo necessário entre o terminal de uma viagem e o início da próxima viagem no sentido oposto. Este período é utilizado para compensar atrasos da próxima partida devido a possíveis trânsito e acidentes da última viagem e a preparação do veículo e descanso do motorista. O planeamento adequado do tempo de pausa é fundamental para garantir que os veículos estejam prontos para a próxima viagem sem comprometer a pontualidade e a fiabilidade do serviço.

Tempo da Inatividade: O tempo da inatividade refere-se ao período de tempo que um veículo permanece inativo entre duas viagens após ter sido atribuído um conjunto de viagens. Este conceito é importante na gestão da frota, uma vez que tempos da inatividade prolongados podem indicar uma subutilização dos recursos.

Atribuição de viagens: refere-se ao processo de designar veículos específicos para cobrir as viagens programadas dentro de uma rota e serviço. Esta alocação deve otimizar o uso da frota, minimizando o número de veículos necessários.

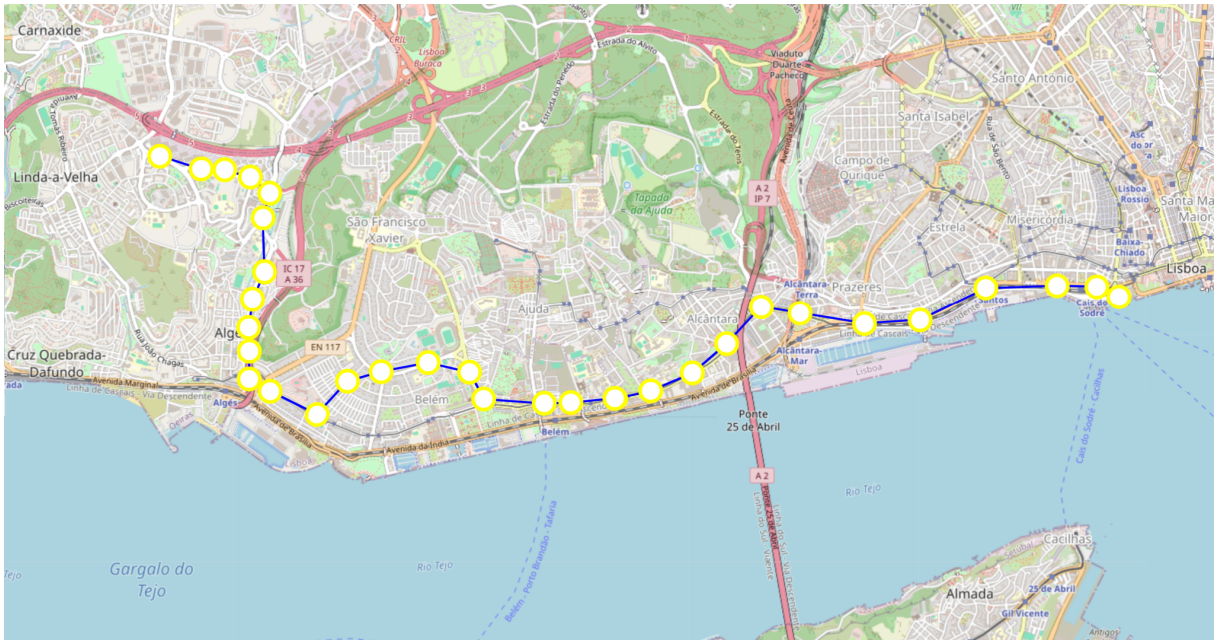


Figura 2.5 Mapa geográfico da rota 60_0, utilizada com o pacote *folium* (2).

2.1.3 Dados reais do problema

A análise e o planeamento eficazes no setor do transporte público dependem da disponibilidade dos dados precisos e abrangentes. Neste contexto, a Tecmic forneceu os dados necessários para a realização da pesquisa, os dados reais da Carris, a empresa responsável pela operação do transporte público na região de Lisboa. A disponibilidade desses dados reflete a precisão da situação real, garantindo a relevância das conclusões. Neste capítulo serão apresentados os dados de Carris de 26 de fevereiro a 8 de abril de 2024.

Os dados fornecidos pela Tecmic são em formato GTFS (*General Transit Feed Specifi-*

cation), um padrão aberto que define um formato comum para a troca de informações sobre horários, rotas e outras informações relacionadas ao transporte público. Ao adotar o GTFS, as empresas de gestão e de solução podem reunir uma variedade dos dados relevantes de diversas fontes e integrá-los de forma coesa nos seus processos de análise e planeamento. Isso não apenas simplifica a manipulação e o tratamento dos dados, como também promove uma abordagem mais abrangente e precisa na conceção de estratégias da afetação dos veículos. A descrição dos ficheiros GTFS está apresentada no Anexo A.1.

Os exemplos do Anexo A.2 só foram possíveis de obter devido à consistência dos dados dos vários ficheiros GTFS fornecidos, consistência essa que valida a precisão das informações recolhidas. Com essa validação, torna-se possível extrair de forma confiável os dados necessários de cada rota, em função dos diferentes tipos de serviço, conforme apresentado no Anexo A.3. Esta capacidade da recolha dos dados detalhados para cada rota e serviço é fundamental para a fase consecutiva da análise e otimização, garantindo que as decisões operacionais se baseiem nas informações precisas e completas.

2.2 Abordagem inicial

Inicia-se com a estratégia para a atribuição sequencial dos horários da partida, onde cada serviço é alocado a um veículo da maneira sequencial e objetiva.

Para simplificar o fluxo do trabalho e evitar problemas decorrentes dessas transformações, convertem-se os valores do tempo, representando os minutos ou segundos em decimais, isto é, aplicam-se aos dados de *Start_time* e *End_time* a função *to_float* (Ver Anexo B.4). Esta abordagem uniformiza o tratamento das variáveis temporais e facilita as operações aritméticas, como somas e subtrações, no contexto do programa da atribuição ou otimização dos veículos. Ao usar valores numéricos, eliminam-se as ambiguidades associadas aos diferentes formatos de tempo.

Até agora, foi descrita a preparação necessária para a atribuição das viagens. A atribuição direta é feita sem a preocupação com a possibilidade da otimização. É centrada na ideia de atribuir cada viagem ao primeiro veículo disponível que possa realizá-la e seguir a sequência temporal das viagens garantindo que cada veículo completa o maior número possível das viagens antes de passar para o veículo seguinte.

Mais especificamente, a atribuição direta no planeamento dos veículos de transporte público segue uma abordagem sequencial e coerente para garantir a funcionalidade básica. O processo inicia-se com a seleção da primeira viagem disponível no conjunto de viagens a efetuar. Esta viagem é então atribuída ao primeiro veículo disponível. Após esta atribuição inicial, as restantes viagens são verificadas para identificar quais podem ser associadas ao mesmo veículo. Esta verificação baseia-se em dois critérios principais: a hora do início da próxima viagem deve ser superior à hora do fim da viagem atual, mais um tempo de pausa.

O processo da verificação e atribuição continua, associando o maior número possível de viagens ao mesmo veículo. Quando não for possível atribuir mais viagens ao primeiro veículo,

o mesmo procedimento é repetido para o próximo veículo disponível. Este processo continua até que todas as viagens tenham sido atribuídas aos veículos, garantindo que cada viagem seja coberta. Esta abordagem da atribuição direta permite uma alocação clara e ordenada dos veículos às respetivas rotas e viagens.

Algoritmo 1 Pseudocódigo da atribuição de viagens a partir da primeira viagem disponível. (Ver código no Anexo B.6

Entrada: Tabela de dados *df*, Tempo de pausa *pausa*

Saída: Atribuição de viagens a cada veículo *dic_bus*

$init \leftarrow 1$

enquanto *df não estiver vazio* **faça**
    redefinir os índices de *df*
    $dic\_bus[init] \leftarrow$ primeira viagem disponível
    para *i em df* **faça**
        $start\_time \leftarrow i[5]$
        $end\_time \leftarrow i[6]$
        $direction \leftarrow i[-1]$
        $depois\_pausa \leftarrow end\_time$ da última viagem de $dic\_bus[init]+$ *pausa*
        se $start\_time >= depois\_pausa$ ***e*** $direction! = direction$ *da última viagem* **então**
            $dic\_bus[init] \leftarrow dic\_bus[init] + i$
            $df \leftarrow df - i$
        fim
    fim
    $init \leftarrow init + 1$
fim

Explica-se o algoritmo usando como exemplo a rota 60_0, no dia 15 de março de 2024, o tipo de serviço correspondido é *Inverno_Util_20240229*. Considera-se o primeiro veículo, chamado '1', atribuindo-lhe a primeira viagem, 12649_20240229 _60_0_2 que parte de Linda-a-velha às 0 horas da madrugada e chega ao Cais do Sodré às 0 : 25 com 25 minutos do percurso. O veículo estará disponível no Cais do Sodré para a viagem seguinte. Caso seja definido 5 minutos como o tempo de pausa após uma viagem efetuada, então a próxima viagem escolhida deverá começar no Cais do Sodré e partir depois das 0 : 30. Ora a primeira viagem que satisfaz estas condições é 12649_20240229_60_0_3, que parte de Cais do Sodré às 0 : 30 e chega a Linda-a-velha às 00 : 55 com 25 minutos do percurso.

Seguindo este raciocínio, e com auxílio do computador, determinam-se as viagens pertencentes ao veículo '1', que estão descritas no Anexo B.7.

Para mais fácil compreensão das soluções, são representadas num gráfico baseado no serviço *rota_servico_60_0_Inverno_Util_20240229* (Figura 2.6). O código para desenhar os gráficos está no Anexo B.10. No eixo horizontal (eixo x) estão representadas as horas, e no eixo vertical (eixo y) a direção de cada viagem, 0 ou 1. Neste exemplo, a direção 0 corresponde ao Cais de Sodré à Linda-a-Velha e a direção 1 o sentido oposto. O fundo do gráfico apresenta

uma grade cinza, a cada hora uniformemente dividida em 6 partes, cada divisão representando um intervalo de 10 minutos. Isso facilita a visualização do tempo total gasto em cada viagem e os intervalos entre elas.

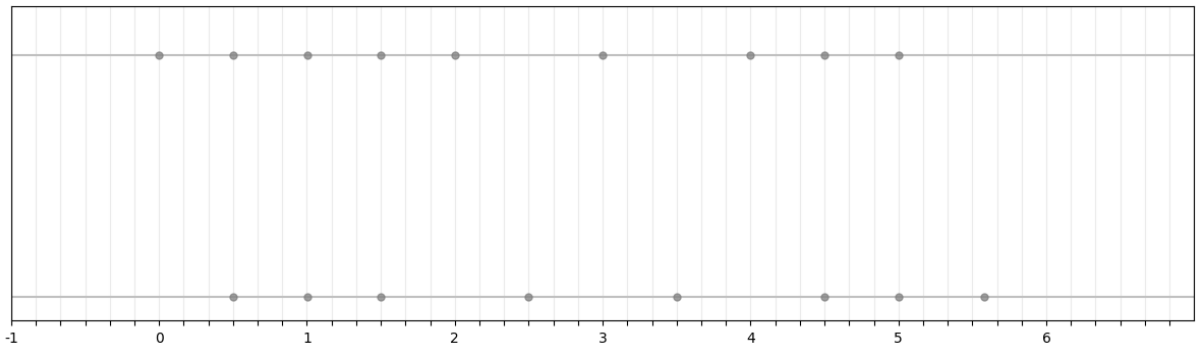


Figura 2.6 Representação gráfica das viagens da rota 60_0 do tipo de serviço *Inverno_Util _20240229*.

Na Figura 2.7 está a representação gráfica para a atribuição das viagens ao veículo '1', onde as linhas coloridas mostram o período em que o veículo está em movimento. O ponto inicial das linhas indica a hora da partida e o ponto final a hora da chegada ao destino. A legenda localizada à direita do gráfico especifica a cor correspondente ao veículo, e o número próximo ao ponto da partida do veículo no gráfico representa a soma das viagens realizadas pelo veículo.

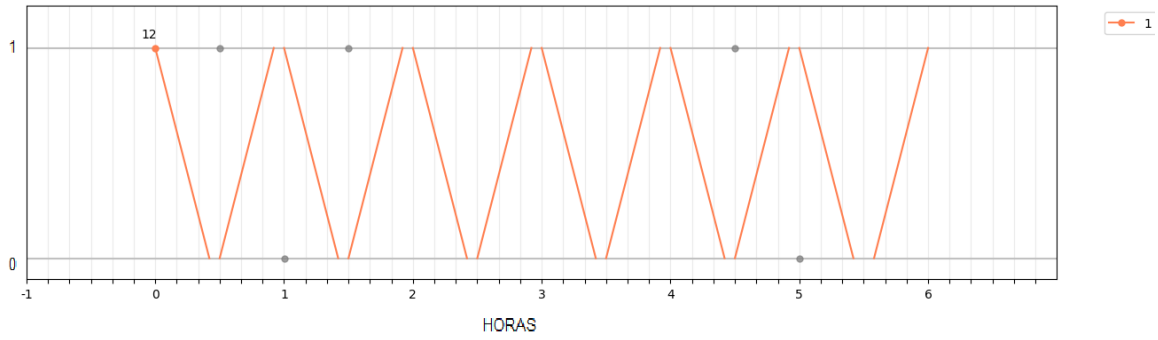


Figura 2.7 Representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço $Inverno_Util$_20240229 do veículo '1', com tempo de pausa de 5 minutos.

Se se aumentar o tempo de pausa para 10 minutos, prevê-se que a quantidade de viagens deste veículo irá diminuir, pois logo a segunda viagem atribuída no caso anterior (5 minutos de tempo de pausa) não vai ser incluída neste caso (Ver Apêndice B.7).

Uma observação em relação ao tempo da inatividade entre as viagens neste caso: a diferença entre o fim de uma viagem e o início da viagem a seguir é superior a meia hora. Isto deve-se provavelmente à pequena dimensão da base de dados, visto não haver outras viagens possíveis a serem escolhidas. Portanto para a rota 60_0, o veículo '1', com o critério da seleção da primeira viagem disponível, a atribuição é igual para o tempo de pausa de 10 minutos como para o tempo de pausa de 35 minutos.

Considere-se o tempo de pausa 5 minutos. Após a atribuição de 12 viagens para o primeiro veículo, ainda restam 5 viagens a serem distribuídas (apresentadas no Anexo B.7).

Estas 5 viagens poderiam ser satisfeitas por um único veículo. Neste trabalho optou-se de não considerar o caso de veículos que chegam ao terminal e voltam à paragem inicial para as viagens do mesmo sentido que realizou, isto é, excluímos a hipótese de um veículo poder circular sem passageiros.

Portanto, no nosso caso as 5 viagens terão de ser realizadas por 2 veículos. Ora, se se atribuir ao segundo veículo a primeira viagem possível, então a composição das viagens para este veículos será a que se encontra na Figura 2.8. A viagem 12671_20240229_60_0_8 fica automaticamente para o terceiro veículo (Fig. 2.9).

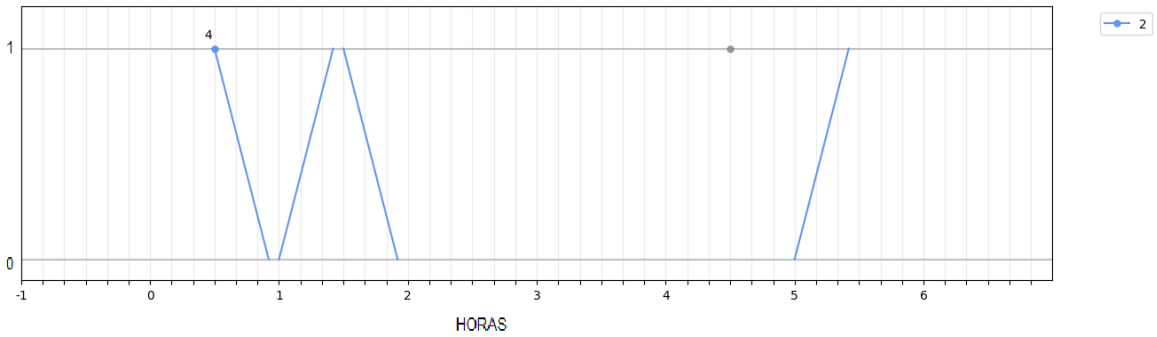


Figura 2.8 Representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço *Inverno_Util*_20240229 do veículo '2', com tempo de pausa de 5 minutos.

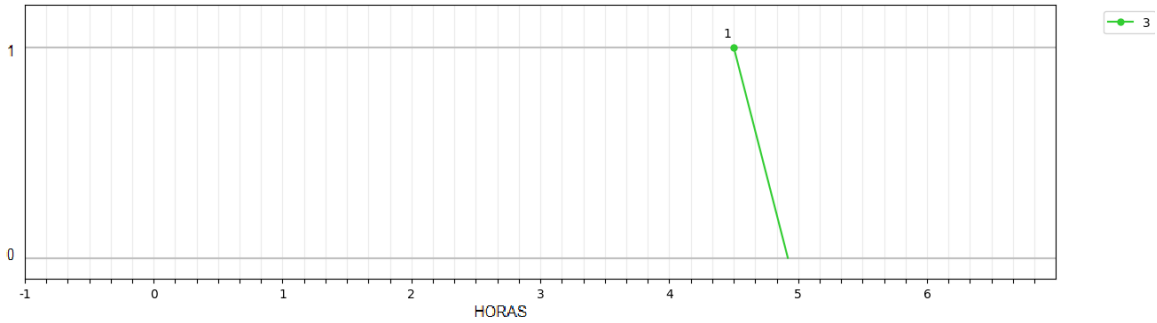


Figura 2.9 Representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço *Inverno_Util*_20240229 do veículo '3', com tempo de pausa de 5 minutos.

A representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço *Inverno_Util_20240229* é ilustrada na Figura 2.10, com a integração da atribuição das viagens de cada veículo.

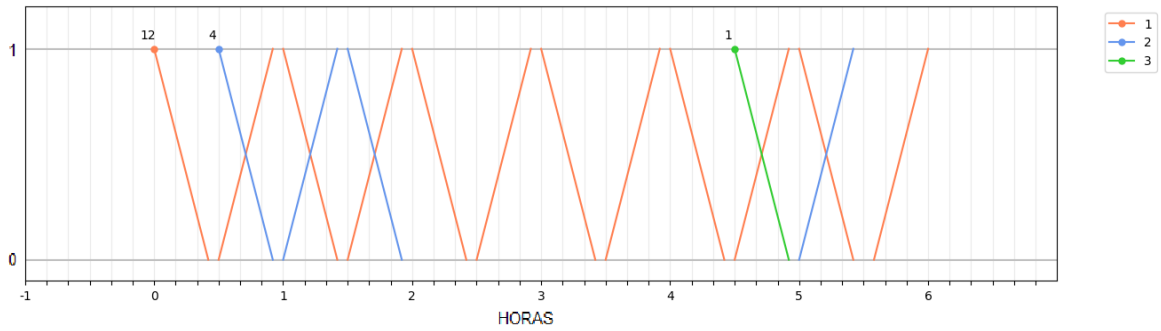


Figura 2.10 Representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos.

A partir do gráfico, observa-se que, por um lado o critério da seleção da primeira viagem possível poderá minimizar a quantidade de veículos, por outro a atribuição é desequilibrada entre o número de viagens dos três veículos. Além disso, o tempo da inatividade para o segundo veículo é demasiado longo para a realidade.

No que se refere ao tempo de pausa, para o caso de 10 minutos ou 35 minutos, a atribuição é representada na Figura 2.11, onde a linha descontínua corresponde ao veículo que começa no ponto da partida com sentido 0.

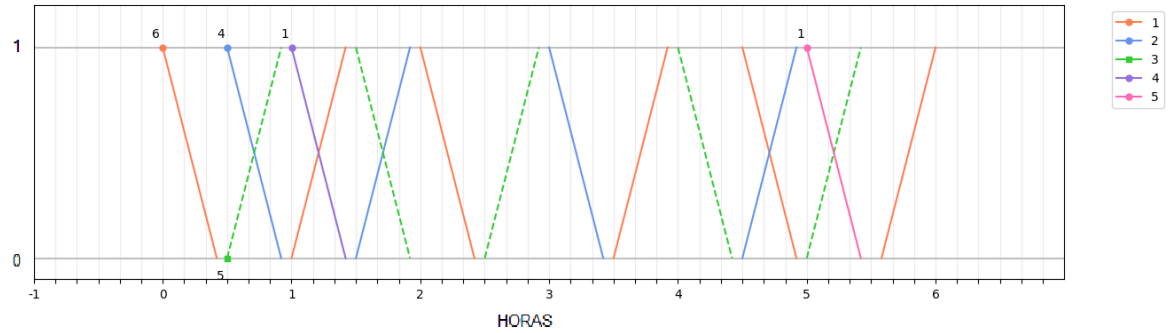


Figura 2.11 Representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 10 minutos.

Neste exemplo, com a extensão do tempo de pausa, a quantidade de veículos necessários também se estende, de 3 veículos para 5.

Para melhor se salientar a diferença dos resultados para tempos de pausa diferentes, considera-se uma base de dados maior: por exemplo a rota 78_0, do Cais Sodré ao Cemitério da Ajuda, e tipo de serviço *Inverno_Util_20240229*, com 76 viagens associadas. Com tempo de pausa de 5 minutos, são necessários 5 veículos, conforme representado na Figura 2.12.

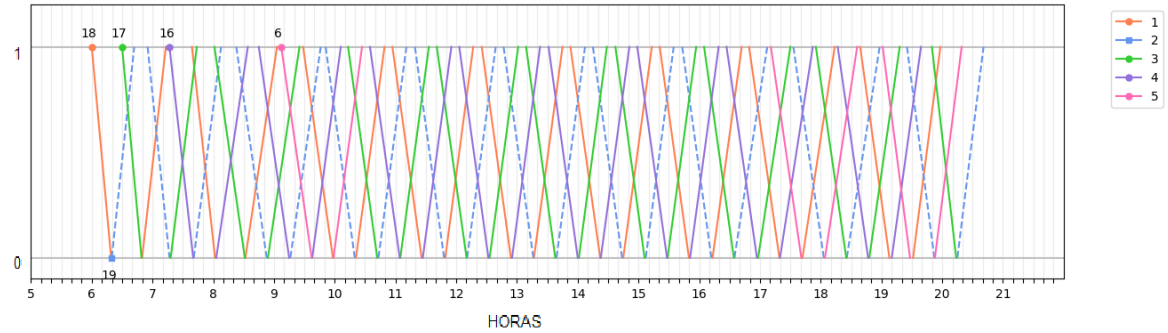


Figura 2.12 Representação gráfica da atribuição das viagens da rota 78_0 do tipo de serviço *Inverno_Util_20240229*, com tempo de pausa de 5 minutos.

A evolução do número dos veículos de acordo com diferentes tempo de pausa está ilustrada na Tabela 2.1.

Tempo pausa	1	2	3	4	5	···	19	20	21	22	23
Nº veículos	3	4	5	5	5	···	5	5	5	5	6

Tabela 2.1 Tabela com a evolução do número de veículos da atribuição conforme o tempo de pausa, da rota 78_0 do tipo de serviço *Inverno_Util_20240229*.

A descrição da tabela é simples. Com tempo de pausa de 1 minuto, são apenas necessários 3 veículos, implicando não haver tempos de compensação de possíveis atrasos, situação representada graficamente na Figura 2.13. Para o caso de tempo de pausa de 2 minutos, já é preciso mais um veículo. E a partir do tempo de pausa de 3 minutos são sempre necessários 5 veículos, o que parece ser um mistério. Como é que com o aumento do tempo de pausa, a quantidade de veículos ainda é a mesma? Será como o caso da rota 60_0?

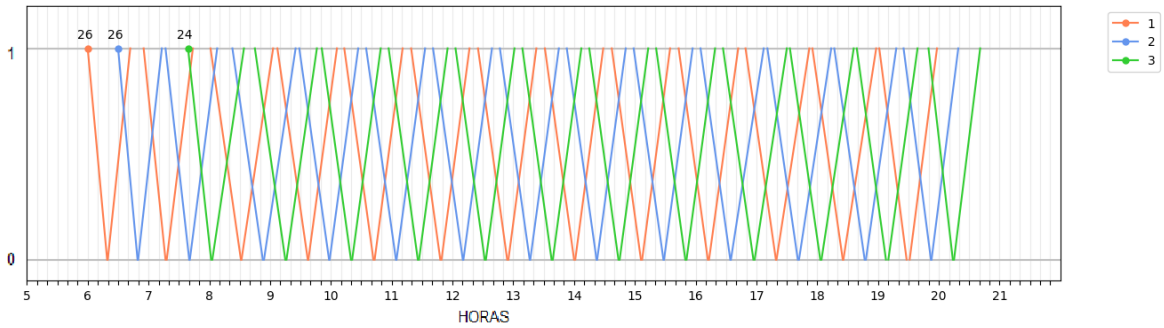


Figura 2.13 Representação gráfica da atribuição das viagens da rota 78_0 do tipo de serviço *Inverno_Util_20240229*, com tempo de pausa de 1 minuto.

Ora, pela Figura 2.14, a resposta às perguntas é explícita: o número das viagens efetuadas pelos veículos está mais equilibrado. Enquanto que no caso de 5 minutos de tempo de pausa há uma diferença de 13 viagens entre o veículo que realiza mais viagens e o que realiza menos, no caso de 20 minutos de tempo de pausa essa mesma diferença é de apenas

2 viagens, o que é bastante mais razoável e será um dos nossos objetivos.

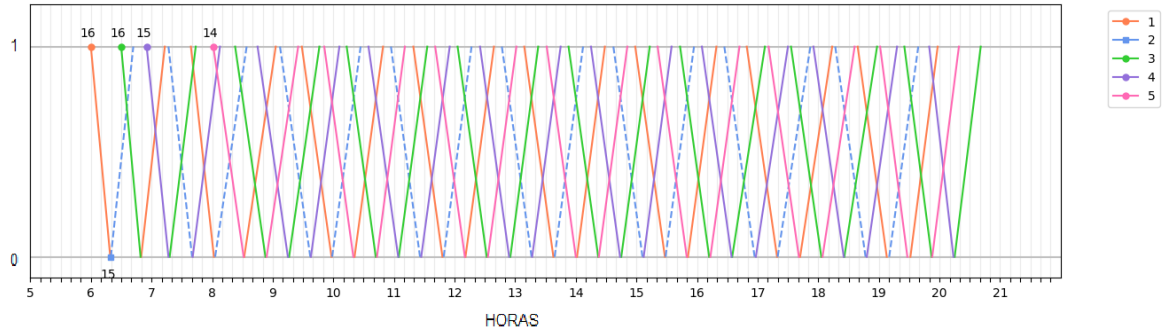


Figura 2.14 Representação gráfica da atribuição das viagens da rota 78_0 do tipo de serviço *Inverno_Util_20240229*, com tempo de pausa de 20 minutos.

2.2.1 Viagens Circulares

À medida que o projeto continua, exploram-se outros tipos de rotas da *Carris*, como é o caso de viagens circulares.

As viagens circulares têm como característica principal o fato da primeira paragem ser a mesma que a última. Toma-se como exemplo a rota 242_0, com paragem de partida e de chegada no Campo de Cebolas. Como é uma rota circular, a direção é sempre de 0.

O algoritmo que se segue permite alterar os valores da coluna da direção quando realiza as iterações da atribuição das viagens. Mais explicitamente, numa iteração com uma das viagens a ser atribuída, modifica-se a direção de forma binária, isto é, de 0 para 1 e de 1 para 0, para garantir que todas as direções possíveis sejam consideradas no processo da atribuição.

Algoritmo 2 Pseudocódigo da atribuição de viagens a partir do primeiro horário disponível para as rotas circulares.

Entrada: Tabela de dados *df*, Tempo de pausa *pausa*

Saída: Atribuição de viagens a cada veículo *dic_bus*

$init \leftarrow 1$

$direction\_list \leftarrow df['Direction']$

enquanto *df não estiver vazio* **faça**

  redefinir os índices de *df*

  $dic\_bus[init] \leftarrow$ primeira viagem disponível

  para *i em df* **faça**

    ...

    $df['Direction'] \leftarrow 1 - df['Direction']$

    se $start\_time >= depois\_pausa$ ***e*** $direction != direction$ *da última viagem* **então**

      $dic\_bus[init] \leftarrow dic\_bus[init] + i$

      $df \leftarrow df - i$

    fim

    $df['Direction'] \leftarrow direction\_list$

  fim

  $init \leftarrow init + 1$

fim

Após a atribuição das viagens, obtém-se o seguinte gráfico da rota 242_0.

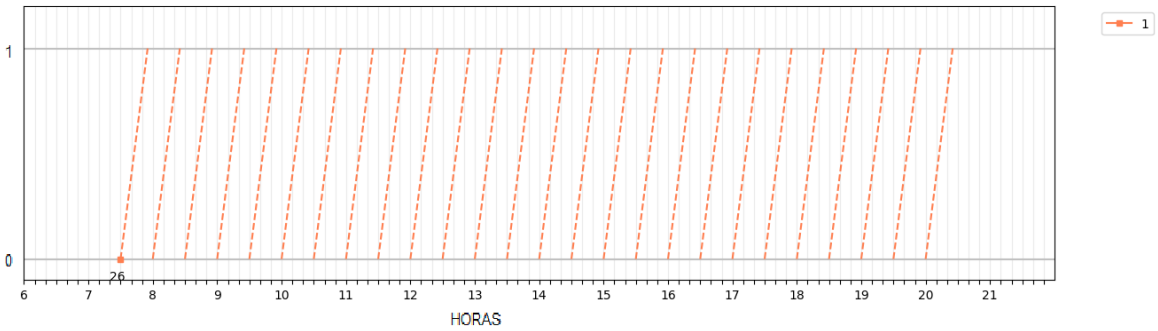


Figura 2.15 Representação gráfica da atribuição das viagens da rota 242_0 do tipo de serviço *Inverno_Util_20240229*, com tempo de pausa de 5 minutos.

Como a direção da rota é sempre 0, então neste caso o eixo y já não tem o mesmo significado. Cada linha a tracejado do gráfico corresponde a uma viagem circular. Com o tempo de pausa a 5 minutos, um único veículo poderá realizar todas as viagens. Se se alterar o tempo de pausa para 10 minutos, precisam-se de dois veículos, que circulam alternadamente.

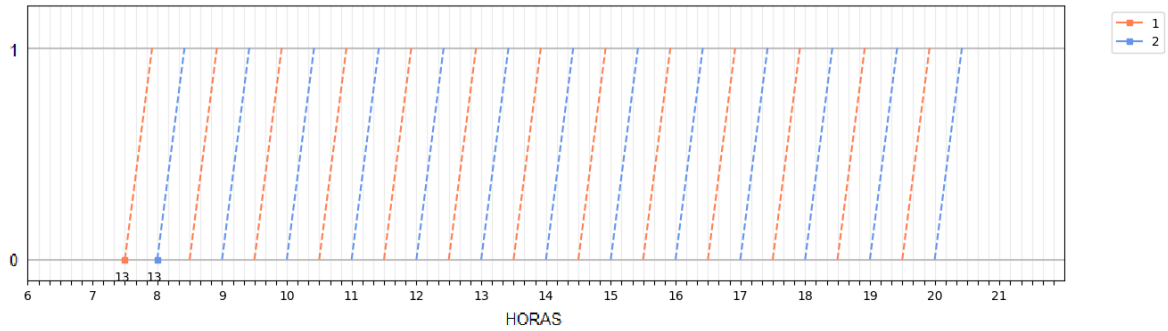


Figura 2.16 Representação gráfica da atribuição das viagens da rota 242_0 do tipo de serviço *Inverno_Util_20240229*, com tempo de pausa de 10 minutos.

2.2.2 Avaliação das soluções com *Trip_init* e *Direction_init*

Até agora, a abordagem adotada para atribuição de viagens tem sido atribuir a primeira viagem disponível aos veículos e depois preencher as restantes viagens sequencialmente. No entanto, esta estratégia pode não ser a mais apropriada. Por exemplo, se o primeiro veículo começar na segunda viagem disponível, a solução resultante pode ser significativamente diferente. Além disso, a definição da direção da primeira viagem do veículo também consegue influenciar diretamente a eficiência global do sistema da atribuição. Dessa forma, o objetivo deste capítulo é apresentar uma nova metodologia para a atribuição de viagens, baseada nos conceitos de *trip_init* e *direction_init*.

Seja *trip_init* o índice da viagem que será iniciada pelo primeiro veículo. Este índice pode variar de 0 até o número total de viagens disponíveis. Através de *trip_init*, a primeira viagem do primeiro veículo é determinada com base no seguinte código:

```
dic_bus[1] = [(df_[`Trips_id'].values[int(trip_init)],
               df_[`Direction'].values[int(trip_init)])]
```

Neste código, *trip_init* é o índice da viagem inicial atribuída ao veículo, e *df_* é a tabela contendo os dados das viagens. Este código define a primeira viagem do primeiro veículo de acordo com *trip_init*, com a informação da id da viagem e a direção correspondente.

Para outras viagens e veículos, o processo da atribuição segue o mesmo método adotado anteriormente. Ou seja, uma vez definida a viagem inicial, as viagens subsequentes são alocadas de acordo com a lógica da atribuição já estabelecida.

A Tabela 2.2 apresenta a quantidade de viagens realizadas por diferentes veículos, conforme a primeira viagem escolhida, na rota 60_0 do tipo de serviço *Inverno_Util_20240229*, com um tempo de pausa de 5 minutos.

Trip_init	0	1	2	3	4	5	···	8	···	14	15	16
Veículo 1	12	10	11	9	10	8	···	7	···	2	1	1
Veículo 2	4	6	4	6	4	6	···	6	···	10	12	11
Veículo 3	1	1	1	1	3	3	···	3	···	4	3	4
Veículo 4			1	1			···	1	···	1	1	1

Tabela 2.2 Tabela de quantidade de viagens relativas a diferente veículos conforme a primeira viagem escolhida, da rota 60_0 do tipo de serviço *Inverno_Util_20240229*, com tempo de pausa de 5 minutos.

Para cada veículo, é apresentada a quantidade de viagens realizadas em função do índice da primeira viagem (*trip_init*).

Para o primeiro veículo, observa-se uma variação significativa no número de viagens atribuídas, dependendo da viagem inicial. Por exemplo, com *trip_init* igual a 0, o veículo realiza 12 viagens, enquanto com *trip_init* igual a 15, realiza apenas 1 viagem. À medida que *trip_init* aumenta, a quantidade de viagens atribuídas aos veículos '2' e '3' também aumentam. O ponto mais equilibrado para a quantidade de viagens a realizar para os veículos 1 e 2 é de *trip_init*$= 8$, é necessário um novo veículo (Veículo '4').

Portanto, a escolha da viagem inicial (*trip_init*) tem um impacto direto na quantidade de viagens realizadas por cada veículo. Estes resultados serão utilizados para otimizar a atribuição de viagens no contexto do serviço de transporte analisado. No capítulo 4 é abordado a metodologia com a escolha da viagem inicial de mais veículos.

Seja *direction_init* a direção da viagem que será iniciada pelo primeiro veículo. Este parâmetro complementa a *trip_init* ao determinar a direção inicial da primeira viagem atribuída ao primeiro veículo. Basicamente, *direction_init* especifica a direção desejada para a primeira viagem. Por exemplo, se introduzir *trip_init*$= 2$ e *direction_init*$= 0$, então a primeira viagem do primeiro veículo será a terceira viagem dos dados com o valor da direção 0.

Para aplicar *direction_init*, é necessário considerar a quantidade de viagens disponíveis em cada sentido. Se os dados contiverem apenas 4 viagens na direção 1, não é possível definir *trip_init*$= 4$ e *direction_init*$= 1$. É importante garantir que existam viagens suficientes na direção escolhida para a configuração inicial desejada.

O código desenvolvido para implementar este parâmetro é o seguinte:

```
dic_bus[1] = [(df_[df_[`Direction']==int(direction_init)].iloc[int(trip_init)]
  [`Trips_id'],int(direction_init))]
```

onde `df_[df_[`Direction']==int(direction_init)]` corresponde ao subconjunto de viagens com a direção *direction_init* a partir de *trip_init*. Usa-se *iloc* para garantir que a seleção seja feita por posição e não por índice.

Trip_init	0		1		2		···	6		7		8	
Direction_init	0	1	0	1	0	1	···	0	1	0	1	0	1
Veículo 1	11	12	9	10	9	10	···	1	4	1	2		2
Veículo 2	4	4	6	6	4	4	···	12	10	11	12		10
Veículo 3	1	1	1	1	3	3	···	3	3	4	3		4
Veículo 4	1		1		1		···	1		1			1

Tabela 2.3 Tabela de quantidade de viagens relativas a diferente veículos com diferentes direções iniciais conforme a primeira viagem escolhida, da rota 60_0 do tipo de serviço *Inverno_Util_20240229*, com tempo de pausa de 5 minutos.

A Tabela 2.3 revela a quantidade de viagens realizadas por cada veículo para combinações de *trip_init* e *direction_init* da rota 60_0. Tal como a conclusão tirada da Tabela 2.2, o segundo veículo tende a aumentar o número de viagens incluídas, enquanto que o primeiro veículo tende a diminuir, devido à escolha feita apenas para a viagem do primeiro veículo. O veículo '3' tem como funcionalidade tratar das viagens que os veículos anteriores não suportavam. O veículo '4' alterna regularmente entre 0 e 1 para as diferentes combinações de *trip_init* e *direction_init*, ou seja, este veículo é utilizado de forma intermitente quando a direção inicial da viagem escolhida é de 0.

Então, numa fase inicial da rota 60_0, conclui-se que a atribuição é mais vantajosa quando a primeira viagem do primeiro veículo tem direção 1, minimizando a quantidade de veículos necessários. Sendo assim, a quantidade de viagens realizadas por cada veículo depende tanto da viagem inicial escolhida quanto da direção inicial. Portanto a retribuição da variável *trip_init* poderia ser uma técnica de abordagem utilizada para a otimização multi-objetivo.

Capítulo 3

Otimização pela permutação das ordens

A partir das análises apresentadas no capítulo anterior, conclui-se que a ordem da atribuição das viagens desempenha um papel importante no processo da otimização da afetação de veículos. Adicionalmente, realiza-se a atribuição baseada em permutações das ordens das viagens, onde diferentes sequências da atribuição são analisadas para identificar a melhor configuração.

3.1 Avaliação exaustiva de todas permutações possíveis

3.1.1 A partir do número inicial de veículos

Para reduzir a quantidade de veículos em circulação, inicialmente, considerou-se a ideia de determinar os veículos necessários a partir das viagens iniciais do dia. A abordagem consiste em determinar quantos veículos são necessários até que o primeiro comece a sua segunda viagem, e as viagens não atribuídas antes desta viagem são designadas viagens iniciais. Enquanto o primeiro veículo realiza a sua rota inicial, é necessário adicionar mais veículos até que as viagens iniciais sejam completadas.

Considere-se a situação da rota 60_0 do tipo de serviço $Inverno_Util_$20240229 (Figura 2.10). Antes do veículo '1' realizar a sua segunda viagem, é necessário que um outro veículo efetue a viagem da outra direção. Logo, para este serviço são precisos pelo menos dois veículos, no entanto não é suficiente: é requerido um terceiro veículo, ao qual lhe será atribuída apenas uma viagem. Neste sentido, continuando com este critério da atribuição da primeira viagem disponível, a reordenação do veículo que inicia este processo pode ser importante. A única viagem do veículo '3' poderia eventualmente ser realizada pelo veículo '2' e obter desta forma a solução ótima para este serviço.

Assim, foi criado o seguinte algoritmo para a resolução do número de veículos iniciais.

Algoritmo 3 Pseudocódigo da resolução de número de veículos iniciais precisos. (Ver código no Anexo B.8.1)

Entrada: Tabela de dados *df*, Tempo de pausa *pausa*

Saída: Número de veículos *n_bus*, Atribuição de viagens iniciais *dic_bus_init*

início

- $n\_bus \leftarrow 1$
- $dic\_bus\_init[n\_bus] \leftarrow$ primeira viagem e sua direção
- $ultima\_viagem \leftarrow$ primeira viagem de $dic\_bus\_init[1]$
- $depois\_pausa \leftarrow$ fim da $ultima\_viagem\_1 +$ *pausa*
- **se** *a viagem não for circular* **então**
  - **para cada** *i em df* **faça**
    - **se** $start\_time \geq depois\_pausa$ ***e*** $direction \neq$ *direção inicial* **então**
      - $segunda\_viagem \leftarrow trips\_id$
      - $viagens\_anteriores \leftarrow$ viagens até a $segunda\_viagem$
    - **fim**
  - **fim**
  - **enquanto** $viagens\_anteriores$ *não estiver vazio* **faça**
    - $n\_bus \leftarrow n\_bus + 1$
    - $dic\_bus\_init[n\_bus] \leftarrow$ primeira viagem de $viagens\_anteriores$
    - Remover a viagem atribuída de $viagens\_anteriores$
    - **para cada** *i em* $viagens\_anteriores$ **faça**
      - $ultima\_viagem\_2 \leftarrow$ última viagem atribuída
      - $depois\_pausa\_2 \leftarrow$ fim da $ultima\_viagem\_2 +$ *pausa*
      - **se** $start\_time \geq depois\_pausa\_2$ $direction \neq$ *última direção* **então**
        - Remover a viagem de $viagens\_anteriores$
      - **fim**
    - **fim**
  - **fim**
- **fim**
- **senão**
  - **para cada** *i em df a partir da segunda viagem* **faça**
    - **se** $start\_time \geq depois\_pausa$ **então**
      - Quebra
    - **fim**
    - $n\_bus \leftarrow n\_bus + 1$
    - $dic\_bus\_init[n\_bus] \leftarrow (trips\_id, direction)$
  - **fim**
- **fim**

fim

Para as viagens não circulares, o primeiro passo consiste em identificar as viagens anteriores à segunda viagem do primeiro veículo. Inicialmente, são atribuídas duas viagens ao

primeiro veículo: a primeira viagem dos dados e uma segunda viagem na direção oposta, que deve satisfazer a condição da compatibilidade dos horários. Todas as viagens anteriores a esta segunda viagem são guardadas na variável $viagens\_anteriores$. A partir dessas viagens, calcula-se o número de veículos necessário para as realizar, utilizando o mesmo critério definido no Algoritmo 1.

Para as viagens circulares é apenas necessário determinar a quantidade de viagens anteriores à segunda viagem do primeiro veículo. Nesse caso, as viagens anteriores a esta segunda viagem não serão realizadas pelos mesmos veículos, o que simplifica o processo da atribuição.

Depois de se determinar o número de veículos iniciais, avança-se para a fase seguinte: a atribuição das viagens com diferentes ordens (Algoritmo 4).

Algoritmo 4 Pseudocódigo da atribuição das viagens em diferentes ordens com o número de veículos iniciais precisos. (Ver código no Anexo B.8.2)

Entrada: Permutação de veículos *arr_perm*, Atribuição inicial *dic_bus_init*, Tabela de dados *df*, Tempo de pausa *pausa*

Saída: Nova atribuição de viagens *solucao*, Lista de viagens atribuídas *list_trip*

```
início
    solucao ← dic_bus_init com apenas a primeira viagem de cada veículo
    list_trip ← lista vazia
    list_init ← lista de trips_id da primeira viagem de cada veículo
    para cada i em df faça
        se trip_id está em list_init então
            continue
        fim
        se trip_id já está na solução então
            continue
        fim
        senão
            para cada veículo j em arr_perm faça
                ultima_viagem ←última viagem atribuída ao veículo j
                depois_pausa ← adicionar pausa ao End_time da ultima_viagem
                se depois_pausa ≤ start_time e direcao_nova_viagem ≠ ultima_direcao en-
                 tão
                    Adiciona a nova viagem à solução para o veículo j
                    Adiciona trip_id à list_trip
                    quebra
                fim
            fim
        fim
    fim
fim
```

Quando se obtém os resultados da atribuição de diferentes ordens, é crucial compará-los. O mais prático para este problema é fazer comparação entre o número de veículos, se estes fossem iguais para as duas em comparação, compara-se pela quantidade de viagens restantes. A realização do processo está descrita no Algoritmo 5.

Algoritmo 5 Pseudocódigo da atribuição das viagens ao número de veículos iniciais com todas as permutações de ordens. (Ver código no Anexo B.8.3)

Entrada: Tabela de dados *df*, Tempo de pausa *pausa*

Saída: Solução ótima *solucao_otima*, Ordem de permutação *ordem*, Número de viagens restantes *otima_length*, Atribuição final de veículos *dic_bus*

início

- $solucao\_otima \leftarrow$ dicionário vazio
- **se** *df vazia* **então**
  - **retorna** *dicionário vazio, lista vazia, 0, 0*
- **fim**
- $n\_bus, dic\_bus\_init \leftarrow$ *n_min_afetacao(df,pausa) [Alg. 3]*
- $arr\_perm \leftarrow$ todas as permutações possíveis dos veículos de 1 até *n_bus*
- *otima_length* $\leftarrow 999$
- *n_bus* $\leftarrow 999$
- *count* $\leftarrow 0$
- **para cada** *permutação i em arr_perm* **faça**
  - *count* $\leftarrow$ *count* $+ 1$
  - $solucao, list\_trip\_used \leftarrow$ *atrb_perm_one(i, dic_bus_init, df, pausa) [Alg. 4]*
  - **para cada** *veículo b de 1 até n_bus* **faça**
    - Adiciona a primeira viagem de cada veículo de *dic_bus_init* à *list_trip_used*
  - **fim**
  - $viagem\_restante \leftarrow$ *Trips_id* em *df* que não estão em *list_trip_used*
  - $df\_trip\_used \leftarrow$ subconjunto de *df* com as viagens não atribuídas
  - $dic\_bus\_new \leftarrow$ *atrb_direto(df_trip_used, pausa) [Alg. 1]*
  - **se** $viagem\_restante \leq otima\_length$ ***e*** *número de veículos na nova solução* $\leq$ *n_bus* **então**
    - $otima\_length \leftarrow$ *número de viagens restantes*
    - $solucao\_otima \leftarrow$ *solucao*
    - $n\_bus \leftarrow$ *número de veículos na nova solução*
    - $dic\_bus \leftarrow$ *dic_bus_new*
    - $ordem \leftarrow i$
  - **fim**
- **fim**

fim

Retomando ao caso prático, existem dois casos possíveis de ordenação, $[1, 2]$ e $[2, 1]$. O número mínimo de veículos já foi determinado com as viagens iniciais do serviço. Assim a primeira viagem de cada um dos veículos mantém-se, ou seja, a diferença na ordem não

interfere na primeira viagem de cada um. Os resultados para as ordens de atribuição $[1, 2]$ e $[2, 1]$ estão representados graficamente nas Figuras 3.1 e 3.2 respetivamente.

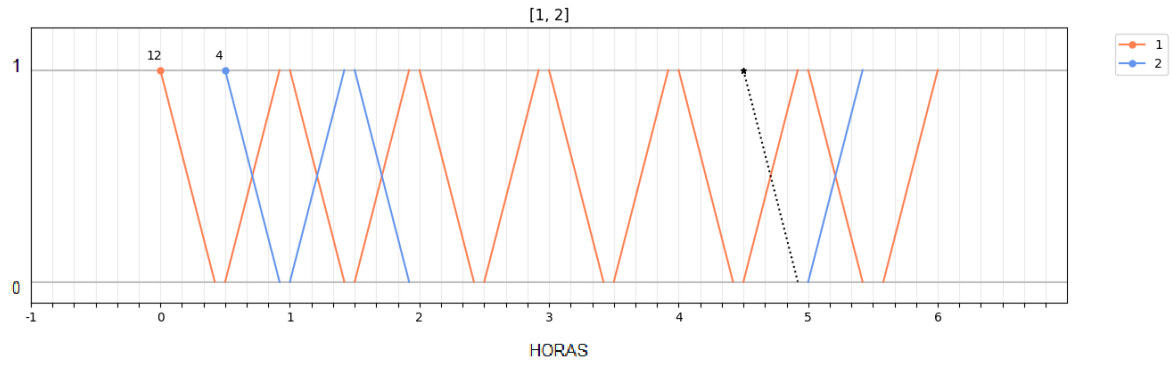


Figura 3.1 Representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos, com ordem de $[1, 2]$.

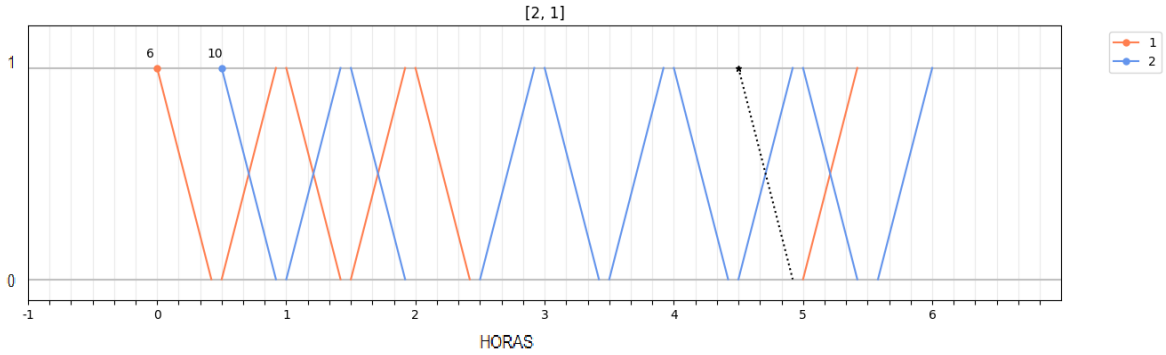


Figura 3.2 Representação gráfica da atribuição das viagens da rota 60_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos, com ordem de $[2, 1]$.

Em ambas as atribuições resta uma viagem que necessita de um novo veículo. Observa-se que, com a técnica da nova ordem da atribuição, a partir da oitava viagem, aquelas que foram atribuídas ao veículo '1' na ordem $[1, 2]$ passam a pertencer ao veículo '2' na ordem $[2, 1]$, e vice-versa. A distribuição de viagens pelos dois veículos está mais equitativa na ordem $[2, 1]$.

Pelo exemplo ilustrado nos dois gráficos concluiu-se que esta metodologia não é uma opção para a resolução do problema, pois ainda restam viagens a serem atribuídas. Na próxima secção é introduzida uma metodologia que tem em conta o número total de veículos obtido da atribuição sequencial (Secção 2.2).

3.1.2 A partir de número completo de veículos

Por vezes, o número dos veículos não é o único fator a considerar; outros aspetos igualmente importantes incluem a distribuição equilibrada das viagens entre os veículos, a definição do tempo de pausa razoável e a gestão do tempo de inatividade entre as viagens.

Ao invés de calcular o número de veículos, como foi feito no Algoritmo 5, nesta resolução este número de veículos é obtido pela Função 1 da atribuição das viagens a partir da primeira viagem disponível. Assim garante-se que todas as viagens sejam atribuídas.

O processo deste algoritmo é igual ao do algoritmo das permutações de ordens com veículos iniciais (Algoritmo 5), mas com um maior número de veículos. À medida que o número de veículos aumenta, é necessário acrescentar algumas restrições ou novos atributos para uma metodologia mais complexa, nomeadamente, a introdução do número de veículos, a condição de interromper o processo a partir de uma variável, a ordem que inicia a permutação e a variável da aleatoriedade. O código pode ser visualizado no Anexo B.8.4.

Definir o número dos veículos

O número de veículos pode ser obtido pelo comprimento do dicionário do Algoritmo 1, ou também ser definido como parâmetro da função neste caso. Computacionalmente foi adicionado um programa de código ao início da função:

```
if n_bus_init:
    for i in np.arange(1,len(dic_bus_init)+1):
      if i < n_bus_init+1:
        dic_bus_init[i] = [dic_bus_init[i][0]]
      else:
        del dic_bus_init[i]
```

onde `dic_bus_init=atrb_direto(df,pausa,0,'')`. Ou seja, se for introduzido um valor neste parâmetro, o dicionário com apenas as primeiras viagens será reduzido na dimensão do valor introduzido, caso não se aplique, este dicionário ficará com todos veículos.

A definição do número dos veículos, ou seja, o comprimento do vetor das ordens, também permite testes dos melhores resultados. Isto é, para uma rota com resultado de 5 veículos na abordagem inicial, neste programa é possível tentar percorrer a todas as permutações de comprimento 4, verificando assim se existem soluções melhores, quer seja no sentido da possibilidade de redução do número de veículos, quer seja no equilíbrio entre a quantidade de viagens atribuídas a todos veículos.

Interrupção do programa

No decorrer do processo, os resultados de cada permutação das ordens torna-se repetitivo: muitas das vezes as permutações diferentes dão os mesmos resultados. Então torna-se precisa uma interrupção do programa, com o parâmetro chamado $solucao\_igual\_max$. Com este parâmetro, o programa é interrompido quando o número de soluções consecutivas com o mesmo número de viagens restantes e o mesmo número de veículos atinge um limite predefinido. Para $solucao\_igual\_max = 5$, o programa pára e retorna resultados quando os resultados de 5 permutações consecutivos forem iguais.

É adicionado uma condição à função recursiva do *for* quando se percorrem todas as permutações:

```
if viagem_restante == anterior_length and len(dic_bus_new) == n_bus_anterior:
  solucao_igual += 1
```

```
if solucao_igual == solucao_igual_max-1:
  return solucao,i,viagem_restante,dic_bus_new
```

onde *anterior_length* é inicializado com 0 e atualizado com valores de *viagem_restante* a cada iteração. Esta condição adicional permite que o programa não continue a testar as permutações que não melhoram o resultado: simultaneamente aumenta a eficiência e reduz o tempo de execução, retornando a solução atual mais cedo.

Definir a ordem inicial

As permutações são geradas em ordem crescente dos valores da lista por padrão. No entanto, ao interromper a execução do programa, é provável que essa interrupção ocorra logo no início do processo, o que pode perder casos da solução ótima. Por isso, é interessante permitir que defina uma ordem inicial e, a partir dessa ordem, gerar as permutações restantes. Assim, a sequência inicial não será necessariamente $[1, 2, 3, 4]$, mas poderá começar com $[3, 2, 4, 1]$ ou outra combinação.

```
if not perm:
    arr_perm = list(itertools.permutations(np.arange(1,n_bus_init+1)))
else:
    arr_perm = list(itertools.permutations(perm))
```

Aleatoriedade

À medida que o número de viagens numa rota aumenta, o tempo de execução e a complexidade do programa também crescem. Nesse contexto, a introdução da aleatoriedade permite baralhar as permutações das ordens da atribuição. Esse parâmetro é especialmente útil quando o número de veículos é maior. Sem aleatoriedade, mesmo que uma nova combinação das ordens seja definida, as ordens subsequentes tendem a ser muito semelhantes à inicial, o que limita a diversidade das soluções. Com a aleatoriedade, as permutações são baralhadas, aumentando a possibilidade de encontrar uma solução diferente e possivelmente melhor.

```
if randomly:
    arr_perm = sorted(arr_perm, key = lambda x: random.random())
```

Resultados

Considere-se um exemplo maior do que a rota 60_0 para melhor execução do programa: a rota 78_0 do Cais de Sodré ao Cemitério da ajuda. Para os serviços de dias úteis, *Inverno_Util_20240229*, tem 76 viagens por dia das 6 horas de manhã às 8 horas da tarde. A atribuição a partir da primeira viagem disponível é ilustrada na Figura 3.3.

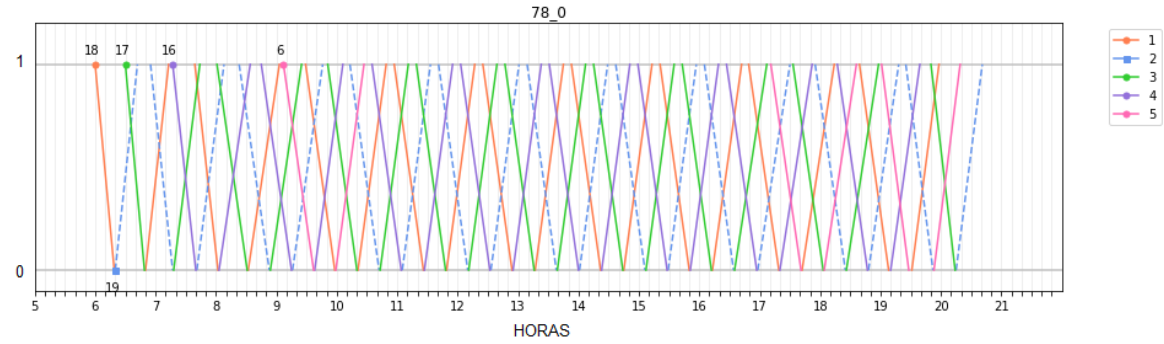


Figura 3.3 Representação gráfica da atribuição das viagens da rota 78_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos.

O número de veículos para efetuar todas as viagens de forma sequencial é de 5, sendo $[0, 1, 2, 5, 15]$ o vetor da posição das primeiras viagens de cada veículo respetivamente.

Portanto para testar se é possível reduzir o número de veículos ou número de viagens não atribuídas, considera-se agora *n_bus_init*$= 4$. Na Figura 3.4, observa-se que foi retirado o veículo '5', as primeiras viagens dos veículos ficam nas posições $[0, 1, 2, 5]$ respetivamente. As linhas pretas a tracejado correspondem a viagens não atribuídas a nenhum veículo.

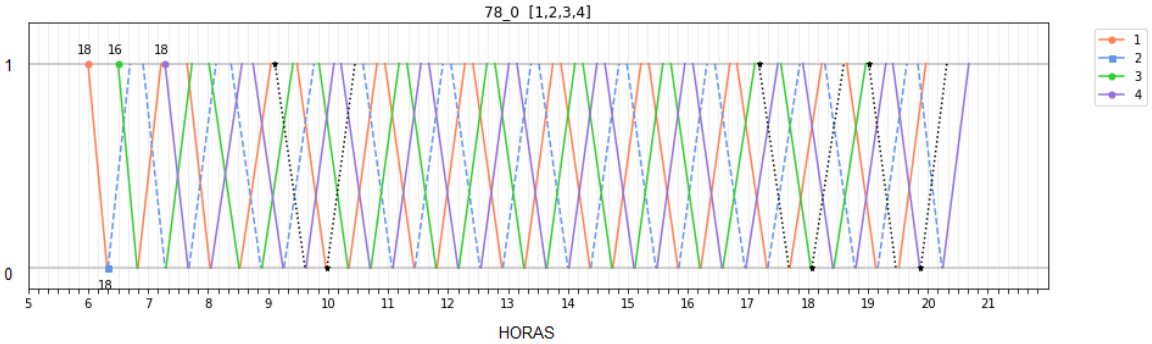


Figura 3.4 Representação gráfica da atribuição das viagens da rota 78_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos e número de veículos 4.

Sendo o número de permutações dado por $n!$ com n a dimensão da ordem, com $n = 4$ teria $4! = 24$ permutações. Para não serem percorridas todas as permutações, usa-se o parâmetro $solucao\_igual\_max$. Considerando-se $solucao\_igual\_max = 5$, o processo termina nas primeiras cinco iterações, visto não se conseguirem atribuir a nenhum veículo as viagens restantes. Portanto, para testar se outras permutações dariam resultados diferentes, introduz-se uma ordem para o parâmetro $perm$. Seja $perm = [4, 3, 1, 2]$, com índice das primeiras viagens $[5, 2, 0, 1]$, e $solucao\_igual\_max = 5$, o processo termina em:

```
(4, 3, 1, 2)
(4, 3, 2, 1)
(4, 1, 3, 2)
(4, 1, 2, 3)
```

(4, 2, 3, 1).

Na permutação $(4, 2, 3, 1)$, a primeira iteração é realizada com o veículo '4'. A atribuição dele começa com a viagem 5 até preencher todas as viagens possíveis (o tempo da partida ser superior ao tempo de pausa mais o tempo da chegada da última viagem, e direções contrárias). De seguida realiza-se o mesmo processo para o veículo '2', que começa com a viagem 1 e assim sucessivamente para os 4 veículos. Como o processo parou em 5 iterações, $solucao\_igual\_max = 5$, as viagens restantes são sempre as mesmas. Comparando-se os resultados das Figuras 3.5 e 3.4 que não há diferença entre elas, conclui-se que esta metodologia não melhora a atribuição para esta rota, isto é, não diminuiu o número de veículos nem a quantidade de viagens não atribuídas, mas por outro lado, equilibrou o número de viagens entre os veículos.

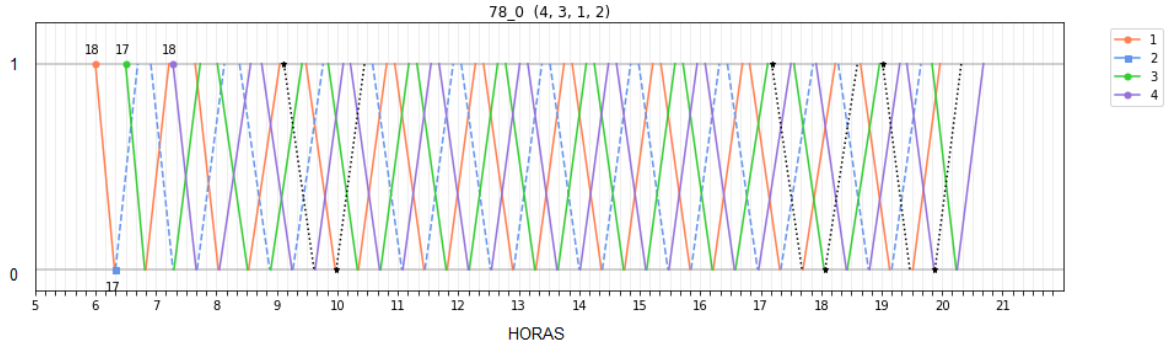


Figura 3.5 Representação gráfica da atribuição das viagens da rota 78_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos e a ordem $[4, 3, 1, 2]$.

Ao se adicionar a aleatoriedade, é mais provável encontrarem-se ordens com resultados diferentes visto que a diversidade é maior. Considere-se $randomly = True$. O processo termina em:

(1, 2, 4, 3)
(3, 1, 4, 2)
(1, 4, 3, 2)
(3, 4, 2, 1)
(2, 1, 3, 4).

Todas estas ordens deram resultados iguais, ou seja, as viagens restantes não diminuíram apesar da introdução de novos parâmetros. No entanto, como o número de viagens é relativamente baixo (76 em vez de mais de 200), qualquer destes processos foi realizado com pouco tempo de execução. Para as ordens pequenas (< 7 veículos), é possível calcular todas as permutações usando este processo e apresentar a melhor solução. Para ordens maiores, por exemplo, para 7 veículos o número de permutações é $7! = 5040$, requer um tempo de execução consideravelmente maior. Na secção seguinte usa-se o método de Recozimento Simulado (*Simulated Annealing*) para se encontrar a melhor permutação.

3.2 Soluções meta-heurísticas

As meta-heurísticas são algoritmos projetados para resolver problemas de otimização complexos, cujo objetivo é encontrar uma solução suficientemente boa num tempo razoável, sem garantir que seja a solução ótima. Ao contrário das heurísticas, que são estratégias específicas para determinados problemas, as meta-heurísticas são métodos mais gerais, aplicáveis a uma vasta gama de problemas.

Exemplos de meta-heurísticas incluem Algoritmos Genéticos, inspirados na evolução biológica; a Pesquisa Tabu, que evita ciclos repetitivos mantendo uma lista de soluções proibidas; o Recozimento Simulado (*Simulated Annealing*), inspirado no processo do arrefecimento de materiais na física; e os Algoritmos de Colónia de Formigas (*Ant Colony Optimization*), baseados no comportamento das formigas na procura do alimento.

Esses métodos combinam heurísticas simples de maneira inteligente, explorando o espaço de soluções e evitando armadilhas como ótimos locais. São particularmente úteis nos problemas da grande escala ou com muitas variáveis, onde os métodos tradicionais da otimização falham em encontrar soluções satisfatórias de forma eficiente.

A principal desvantagem destas abordagens é o elevado custo computacional. Quanto maiores forem os dados envolvidos, maior o tempo da execução que tende a aumentar exponencialmente. Por essa razão, as soluções anteriores são mais adequadas para problemas de menor dimensão (7).

De acordo com Osman e Laporte (1996) (8), as meta-heurísticas foram desenvolvidas a partir dos anos 1980 para resolver problemas de otimização complexos, onde as heurísticas clássicas e os métodos convencionais não são suficientemente eficazes ou eficientes. Além disso, uma meta-heurística é definida como um processo da geração iterativa que guia uma heurística subordinada, combinando de maneira inteligente diversos conceitos para explorar o espaço da pesquisa. As estratégias da aprendizagem são utilizadas para estruturar a informação, visando encontrar soluções quase ótimas.

Desde então, muitos algoritmos meta-heurísticos foram investigados (8), como por exemplo, algoritmos genéticos, redes neuronais, problema e investigação espacial heurística, recozimento simulado e pesquisa tabu. De seguida, é apresentado a técnica utilizada neste capítulo, o recozimento simulado,

3.2.1 *Simulated Annealing*

O recozimento simulado (*SA*) meta-heurístico teve origem na física estatística. O interesse na utilização do *SA* para problemas de otimização combinatória começou no início da década de 80 com o trabalho de (*Kirkpatrick*, *Gelatt* e *Vecchi*, 1983) e (*Cerny*, 1985) (8).

Como o nome sugere, o SA inspira-se no processo do recozimento térmico, onde um material é aquecido a uma temperatura elevada e depois arrefecido lentamente, para gradualmente se aproximar de uma solução ótima.

O funcionamento do SA baseia-se num processo estocástico. Parte-se de uma solução

inicial e, em cada iteração, gera-se uma nova solução. Se a nova solução for melhor que a anterior, é aceite e utilizada na próxima iteração. Se a nova solução for pior, pode ainda ser aceite com uma certa probabilidade, que diminui ao longo do tempo.

Na Figura 3.6 está representado um diagrama do processo do algoritmo.

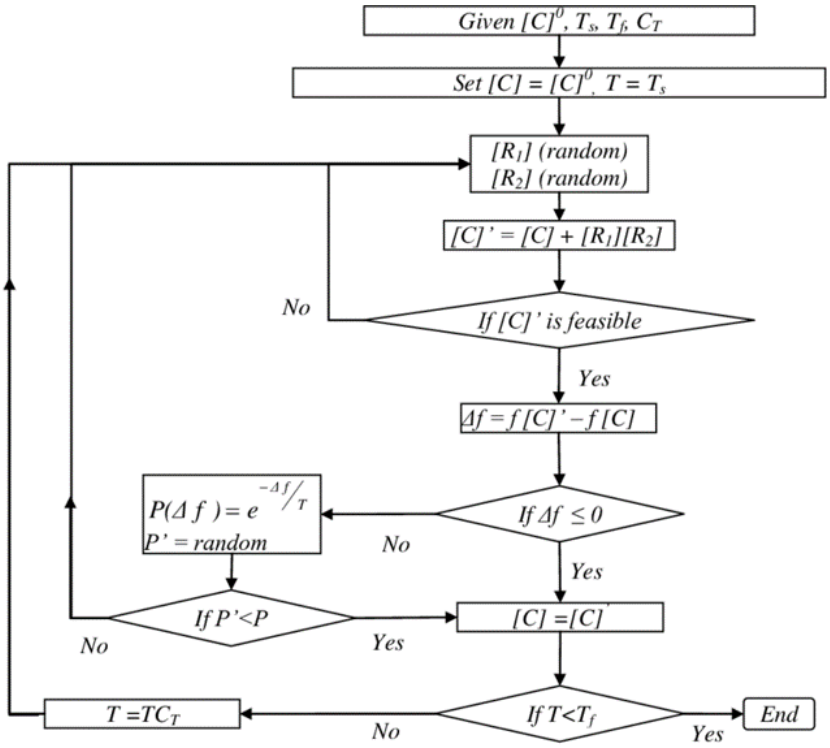


Figura 3.6 Algoritmo do recozimento simulado, adaptada de (9).

Com o objetivo da diminuição do custo computacional e de encontrar uma solução ótima, o algoritmo baseia-se na atribuição em permutações das ordens a partir do número completo de veículos. Então, são considerados 4 parâmetros inicialmente.

$[C]^0$ - a ordem inicial dos veículos

T_s - a temperatura inicial

T_f - a temperatura final para parar as iterações

C_T - o fator da redução da temperatura

Os valores destes parâmetros são determinados pelo esquema do resfriamento do *SA*. Tipicamente, o algoritmo SA começa com uma temperatura elevada, permitindo uma busca ampla no espaço de soluções, e vai reduzindo gradualmente a temperatura para focar numa região específica.

3.2.2 Implementação

O problema que se pretende resolver envolve a atribuição de veículos para operar num circuito urbano entre duas localizações, designadas por *A* e *B*. Ao longo do dia, há várias viagens com partida em *A* e chegada em *B*, e vice-versa. A frequência dessas viagens não é regular, pois é influenciada por fatores como as horas de ponta e outros imprevistos.

Determinação da solução inicial

Para determinar a solução inicial, seguem os seguintes passos:

1. Atribui-se ao veículo '1' a primeira viagem disponível com partida em A e destino em B. Quando o veículo chega a B, faz uma pausa para compensar possíveis atrasos causados por fatores imprevistos, como trânsito. Em seguida, é-lhe atribuído a primeira viagem disponível com partida em B e destino em A.
2. Este processo repete-se, alternando entre partidas em A e B, até que todas as viagens disponíveis sejam atribuídos ao veículo '1'. Uma vez concluída a atribuição, eliminam-se as viagens já utilizados para este veículo.
3. O mesmo procedimento é aplicado ao veículo '2', que recebe a primeira viagem disponível, seja com partida em A ou em B. O processo continua até que todas as viagens disponíveis sejam atribuídos.
4. O processo repete-se para todos os veículos necessários até que todas as viagens no circuito entre A e B sejam preenchidos.
5. No final, obtém-se o número total de veículos necessários, designado por N. Para cada veículo, regista-se a identificação de primeira viagem e a direção respetiva. Esta informação é organizada num dicionário, onde:
    - As chaves contém a ordem dos veículos (variando de 1 a N);
    - Os valores de cada chave indicam um tuplo com identificação de primeira viagem e a sua direção.

Otimização

Com a solução inicial de N veículos diferentes, consegue-se explorar outras combinações alterando a ordem dos veículos. O número de permutações possíveis das ordens de 1 a N permite avaliar todas as alternativas:

- Se N for pequeno (inferior a 7), pode-se calcular todas as permutações possíveis e avaliar cada caso.
- Se N for maior, aplica-se uma técnica da otimização usando SA para encontrar uma solução mais eficiente.

Avaliação da função objetivo

Na avaliação da função objetivo, o nosso objetivo é minimizar o número de veículos necessários para cobrir todas as viagens disponíveis. O método é inspirado no processo do recozimento térmico de materiais, onde estes são aquecidos e arrefecidos gradualmente até atingirem um estado de menor energia. Na otimização, o objetivo é similar: minimizar o número de veículos necessários e equilibrar a distribuição de viagens entre eles.

O SA segue os seguintes passos para otimizar o problema:

1. **Solução inicial:** O algoritmo começa com uma solução inicial obtida pela atribuição de veículos conforme descrito anteriormente. Esta solução serve como ponto de partida.

2. **Geração de uma nova solução:** A cada iteração, o algoritmo gera uma nova solução, alterando ligeiramente a solução atual. Esta alteração é feita aplicando uma pequena permutação aleatória na ordem dos veículos.

3. **Avaliação da nova solução:** A nova solução gerada é avaliada com base em dois critérios: o número de veículos utilizados e a distribuição de viagens.

4. **Decisão de aceitação:** Se a nova solução for melhor (ou seja, utilizar menos veículos ou minimizar número de serviços atribuídos ao veículo com menos viagens), ela é automaticamente aceite. Se for pior, o algoritmo pode ainda assim aceitá-la com uma probabilidade calculada pela fórmula:

   $$P(aceitacao) = \exp(\Delta f / T),$$

   onde Δf é a diferença na qualidade entre a nova solução e a atual, e T é a temperatura atual. Esta probabilidade ajuda o algoritmo a evitar ficar preso em soluções ótimas locais, permitindo a exploração de soluções inicialmente subótimas de forma controlada.

5. **Redução da temperatura:** A cada iteração, a temperatura T é reduzida de acordo com o fator de arrefecimento C_T. Conforme a temperatura diminui, a probabilidade de aceitar soluções piores também diminui. No início, o algoritmo faz uma exploração mais ampla do espaço de soluções; à medida que a temperatura se reduz, o foco passa a ser no refinamento.

Algoritmo 6 Pseudocódigo da atribuição de viagens usando o algoritmo meta-heurística: Recozimento simulado. (Ver código no Anexo B.9)

Entrada: Número de pontos *numPontos*, Temperatura inicial *temperaturaInicial*, Temperatura mínima *temperaturaMinima*, Fator de redução da temperatura *fatorReducaoTemperatura*, Dicionário de viagens *dic_init*, Pausa entre viagens *pausa*

Saída: Ordem final de atribuição *ordemFinal*, Viagens não atribuídas *viagem_restante*

$pontoInicial \leftarrow$ *aleatorio(numPontos)*

$pontoAtual \leftarrow$ *pontoInicial*

$ordemAtual \leftarrow$ a ordenação dos valores de *pontoAtual*

$temperatura \leftarrow$ *temperaturaInicial*

se *numPontos for menor que o tamanho de dic_init* **então**

  para cada *índice i até o tamanho de dic_init* **faça**

    se $i \leq numPontos$ **então**

      Atribuir apenas a primeira viagem ao veículo correspondente no dic_init

    fim

    senão

      Remover o veículo do dic_init

    fim

  fim

fim

$count \leftarrow 0$

enquanto *temperatura for maior que temperaturaMinima* **faça**

  $count \leftarrow count + 1$

  $novoPonto \leftarrow$ *pontoAtual*

  $indiceAleatorio \leftarrow$ *random(numPontos)*

  $novoPonto[indiceAleatorio] \leftarrow$ *aleatorio(numPontos)* $+$ valor aleatório

  $ordemNova \leftarrow$ a ordenação dos valores de *novoPonto*

  enquanto *ordemAtual for igual a ordemNova* **faça**

    $novoPonto[indiceAleatorio] \leftarrow$ *aleatorio(numPontos)* $+$ valor aleatório

    $ordemNova \leftarrow$ a ordenação dos valores de *novoPonto*

  fim

  $delta \leftarrow$ *funcaoObjetivo(ordemNova)-funcaoObjetivo(ordemAtual)*

  se $delta \leq 0$ ***ou*** $exp(-delta/temperatura) \geq nmeroaleatrio$ **então**

    $pontoAtual \leftarrow$ *novoPonto*

    $ordemAtual \leftarrow$ *ordemNova*

  fim

  $temperatura \leftarrow$ *temperatura* $\times$ *fatorReducaoTemperatura*

fim

$ordemFinal \leftarrow$ a ordenação de *pontoAtual*

onde a função *funcaoObjetivo* é dada pelo Algoritmo 7.

Algoritmo 7 Pseudocódigo da atribuição de viagens adaptada ao programa de SA. (Ver código no Anexo B.9.1)

Entrada: Número de pontos *numPontos*, Temperatura inicial *temperaturaInicial*, Ordem de permutação *ordem*, Dicionário de viagens *dic_init*, Pausa entre viagens *pausa*

Saída: Tamanho de *viagem_restante*, Solução *solucao*, Viagens não atribuídas *viagem_restante*

$viagem\_restante \leftarrow$ lista vazia

$solucao, list\_trip\_used \leftarrow$*atrb_perm_one(ordem, dic_init, df, pausa) [Alg.4]*

para cada *b em dic_init* **faça**

  $list\_trip\_used \leftarrow list\_trip\_used + b[0]$

fim

para cada *a em df['Trips_id']* **faça**

  se *a não está em list_trip_used* **então**

    $viagem\_restante \leftarrow$ a

  fim

fim

A função começa verificando se *numPontos* é um número inteiro. Se sim, gera um vetor inicial (*pontoInicial*) aleatório, que servirá de base para o SA. Este vetor é usado para calcular a ordem da atribuição inicial dos veículos. Depois, ajusta-se o número de veículos. Se o número de veículos (*numPontos*) for menor que o número de chaves no *dic_init*, a função ajusta o dicionário *dic_init*, removendo veículos ou mantendo apenas as primeiras viagens. A cada iteração do algoritmo, a função:

- Gera uma nova proposta de solução (*novoPonto*) alterando um ponto da solução atual de forma aleatória.
- Calcula a nova ordem de atribuição baseada nesse ponto.
- Compara a nova solução com a atual, usando a função objetivo.
- Aceita a nova solução se ela for melhor ou, com certa probabilidade, se for pior, a depender da temperatura atual.
- Atualiza a temperatura, esta vai sendo gradualmente reduzida em cada iteração até atingir o limite mínimo.

Quando o processo termina, a função retorna a ordem final dos veículos e as viagens que ficaram sem atribuição.

3.2.3 Resultados

Dado que a rota `60_0` tem pequena dimensão para trabalhar num algoritmo meta-heurístico, considera-se novamente a rota `78_0` (Sec. 3.1.2), onde a atribuição direta realizada para o

serviço *Inverno_Util_20240229* com tempo de pausa de 5 minutos é representada na Figura 2.12.

Considera-se à partida o número de veículos igual a 4 ($numPontos = 4$) para encontrar a possibilidade de reduzir a quantidade de veículos ou de viagens restantes, a temperatura inicial de 30, a temperatura final de $1e-1$ e o fator da redução de 0.9. Quanto maior a diferença entre as temperaturas e maior o fator da redução, maior o espaço de soluções e consequentemente melhor a solução ótima encontrada. A ordem inicial a começar o algoritmo é aleatória. Numa das execuções, a ordem começou com $[4, 2, 1, 3]$, e daí iterando ao mesmo tempo baixar a temperatura até aproximar uma solução melhor. Os resultados de algumas iterações são mostrados na Tabela 3.1.

	Ordem	Temperatura	Viagens restantes
1ª iteração	[4 2 1 3]	30	6
2ª iteração	[2 4 1 3]	27	6
3ª iteração	[2 4 3 1]	24.3	6
4ª iteração	[2 3 4 1]	21.876	6
5ª iteração	[2 4 1 3]	19.68	6
6ª iteração	[2 1 4 3]	17.71	6
7ª iteração	[1 2 4 3]	15.94	6
8ª iteração	[1 2 3 4]	14.35	6
9ª iteração	[1 3 4 2]	12.91	6
...	...	...	...
55ª iteração	[1 4 2 3]	0.1	6

Tabela 3.1 Tabela de representação dos resultados das iterações do algoritmo SA, da rota 78_0 do tipo de serviço $Inverno_Util_$20240229, com tempo de pausa de 5.

O caso exemplo da rota 78_0 não é suficientemente grande para mostrar a funcionalidade da otimização, pois o número de iterações é muito maior que as permutações de ordem 4, logo o algoritmo esteve a repetir entre os vetores de ordens. Foi escolhido então a rota 76_0 do serviço $Inverno_Util_$20240229, da Praça de Figueira a Algés, que começa às 5 horas da manhã e termina à 1 hora do dia a seguir.

Pela atribuição a partir da primeira viagem disponível, o número dos veículos que é necessário para englobar todas as viagens é 14, o que é um valor adequado para testar o algoritmo. A atribuição é ilustrada na Figura 3.7.

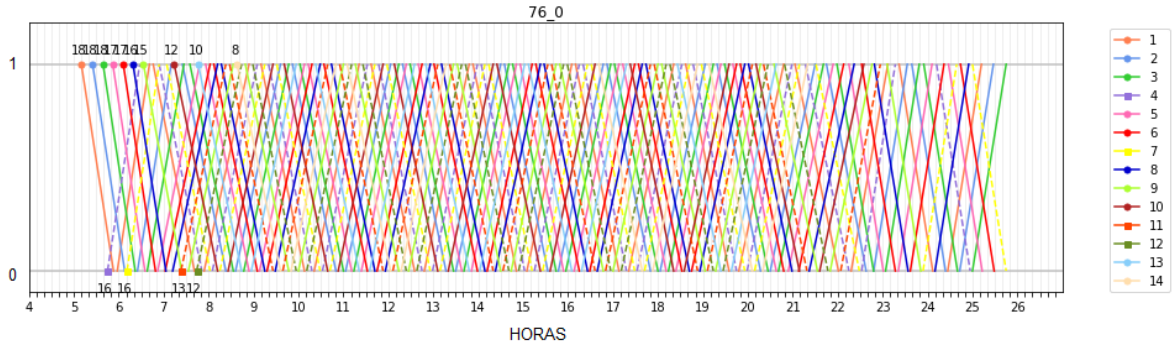


Figura 3.7 Representação gráfica da atribuição das viagens da rota 76_0 do tipo de serviço $Inverno_Util$_20240229, com tempo de pausa de 5 minutos.

O veículo '14' realiza 8 viagens. Então executa-se a otimização de SA, com número de veículos igual a 13 e mantendo todos os outros parâmetros do caso anterior. O objetivo é de encontrar uma possível ordem que diminua a quantidade de viagens restantes.

Após 5 minutos da execução, obteve-se o resultado ilustrado na Tabela 3.2, onde se conclui que não obteve resultados melhores à atribuição da abordagem inicial, a partir da primeira viagem disponível e veículos em execução por ordem crescente, visto que o número de viagens restantes a serem atribuídas são sempre 8.

	Ordem	Temperatura	Viagens restantes
1ª iteração	[12 10 13 1 4 8 9 11 5 6 7 3 2]	30	8
2ª iteração	[12 10 13 1 4 8 11 5 6 7 3 2 9]	27	8
3ª iteração	[12 10 13 4 1 8 11 5 6 7 3 2 9]	24.3	8
4ª iteração	[12 10 13 4 1 8 11 5 6 3 2 7 9]	21.876	8
5ª iteração	[12 10 13 4 1 8 11 7 5 6 3 2 9]	19.68	8
6ª iteração	[12 10 13 1 8 4 11 7 5 6 3 2 9]	17.71	8
7ª iteração	[10 12 13 1 8 4 11 7 5 6 3 2 9]	15.94	8
8ª iteração	[10 12 13 1 8 4 11 7 5 6 2 9 3]	14.35	8
9ª iteração	[10 12 13 1 7 8 4 11 5 6 2 9 3]	12.91	8
...	...	...	...
55ª iteração	[12 2 8 11 3 7 13 4 1 10 6 5 9]	0.1	8

Tabela 3.2 Tabela de representação dos resultados das iterações do algoritmo SA, da rota 76_0 do tipo de serviço $Inverno_Util$_20240229, com tempo de pausa de 5 minutos.

Para confirmar a funcionalidade do algoritmo, extrai-se o veículo '12' dos casos [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] e [12, 2, 8, 11, 3, 7, 13, 4, 1, 10, 6, 5, 9], apresentados nas Figuras 3.8 e 3.9, respetivamente. Nos gráficos destacando o único veículo, facilita a visualização e uma melhor compreensão do resultado. No segundo caso, o veículo '12' está na primeira posição a ser atribuída, logo, a partir da primeira viagem já definida deste veículo, teve oportunidade de realizar as viagens que não atingia no primeiro caso (as últimas quatro), e o conjunto de viagens

foi alterado desde a segunda viagem, que parte das $9:47$, enquanto que no primeiro caso, a segunda viagem partia às $9:58$.

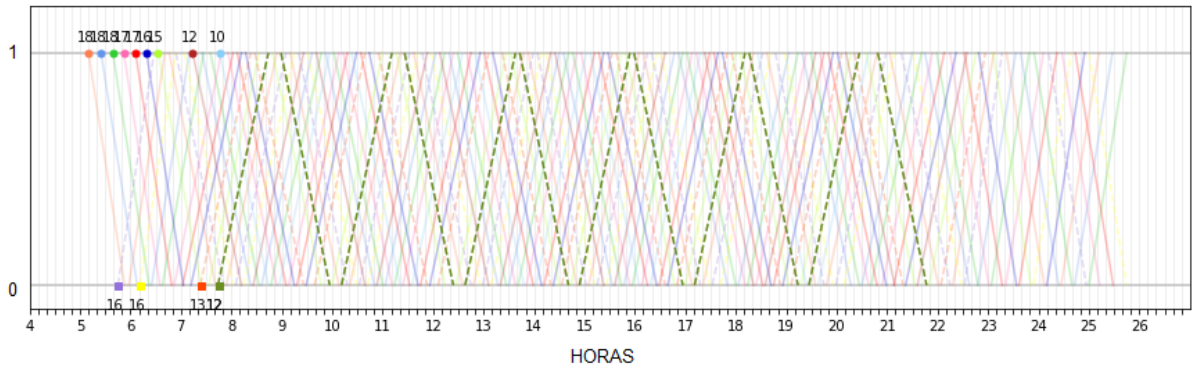


Figura 3.8 Representação gráfica da atribuição das viagens da rota 76_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos, com veículo '12' destacado.

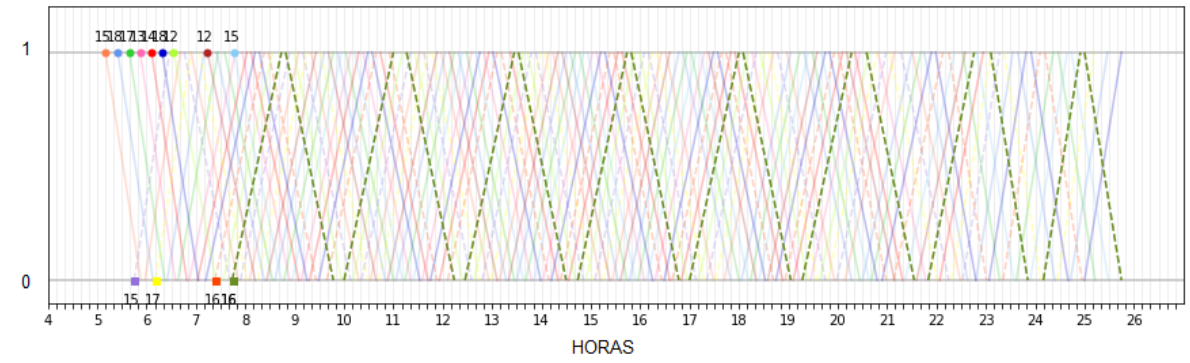


Figura 3.9 Representação gráfica da atribuição das viagens da rota 76_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos, da ordem $[12, 2, 8, 11, 3, 7, 13, 4, 1, 10, 6, 5, 9]$ com veículo '12' destacado.

Este exemplo da rota 76_0 acabou por não se revelar muito significativo. Iterando todas as 175 rotas e os 4 tipos de serviços, com tempo de pausa de 5 e 15 minutos, a abordagem considerada no Recozimento Simulado não melhorou a solução inicial - a solução que considera a primeira viagem disponível, quer no número de veículos quer na quantidade de viagens restantes. Não chegou à expetativa de minimização do número de veículos, no entanto a quantidade de viagens atribuídas aos veículos estão mais equilibrados com a ordem de atribuição diferente. Por exemplo, com a rota 76_0, nas Figuras 3.8 e 3.9, a diferença entre o veículo com mais viagens e o veículo com menos viagens são diferentes. A atribuição sequencial (Figura 3.8) tem uma diferença de $18 - 10 = 8$ viagens, enquanto que a atribuição da ordem $[12, 2, 8, 11, 3, 7, 13, 4, 1, 10, 6, 5, 9]$ tem uma diferença de $18 - 12 = 6$ viagens, apresentando uma atribuição mais equilibrada.

Para obter resultados diferentes, outra abordagem é a atribuição de viagens definidas a cada veículo, como referido na Secção 2.2.2. Ou seja, a cada veículo é lhe atribuído um índice da viagem pretendida e forma-se um vetor que contém todos os veículos, por exemplo,

para os veículos $[1, 2, 3, 4, 5]$ a atribuição a partir da primeira viagem disponível tem como vetor $[0, 0, 0, 0, 0]$, então e se fosse $[1, 1, 1, 1, 1]$? Poderá dar resultados diferentes. Ao mesmo tempo da nova abordagem, pode-se adicionar as restrições ditas anteriormente (minimização do maior intervalo da inatividade, equilibrar a quantidade de viagens aos veículos e ao mesmo tempo maximizar o tempo de pausa).

Capítulo 4

Otimização Multi-objetivo

A otimização multi-objetivo é uma abordagem fundamental em problemas que envolvem a maximização de várias funções em conflito, onde a melhoria de um objetivo pode prejudicar outro. O foco não está em encontrar uma única solução ótima, mas sim em identificar um conjunto das soluções que representam diferentes compromissos. Neste contexto, o algoritmo NSGA-II (Non-dominated Sorting Genetic Algorithm II) destaca-se como uma estratégia eficaz. Ele ao combinar operadores evolutivos como *crossover* e mutação, resulta em soluções mais equilibradas, otimizando a distribuição e minimizando o número dos veículos.

4.1 Introdução e Conceitos

A otimização multi-objetivo é uma área da otimização matemática que trata da resolução de problemas em que é necessário otimizar simultaneamente duas ou mais funções objetivo. Estes problemas surgem frequentemente em cenários onde existe um conflito entre os diferentes objetivos a serem alcançados. Em vez de um único objetivo, como minimizar o número dos veículos, são consideradas várias metas que podem ser contraditórias, exigindo compromissos entre elas.

Em muitas situações reais, não é possível maximizar ou minimizar simultaneamente todas as funções objetivo, pois a melhoria de uma pode implicar a deterioração de outra. Este fenómeno é conhecido como *trade-off* ou compromisso, e significa que, ao melhorar um dos objetivos, é necessário sacrificar o desempenho do outro (10). Por exemplo, no transporte público, otimizar o número dos veículos pode comprometer a qualidade do serviço (tempo de espera dos passageiros). Assim, em vez de se procurar uma única solução "ótima", o objetivo é identificar um conjunto de soluções que oferecem diferentes equilíbrios entre os objetivos.

Um conceito fundamental em otimização multi-objetivo é a frente de pareto. A frente de pareto é composta por todas as soluções que são não dominadas — ou seja, as soluções para quais não existe outra que melhore um objetivo sem piorar o outro . As soluções pertencentes à frente de pareto são consideradas ótimas no sentido de pareto, pois representam diferentes compromissos entre os objetivos em conflito. Estas soluções são preferíveis porque nenhuma pode ser considerada estritamente melhor sem avaliar a importância relativa de cada objetivo..

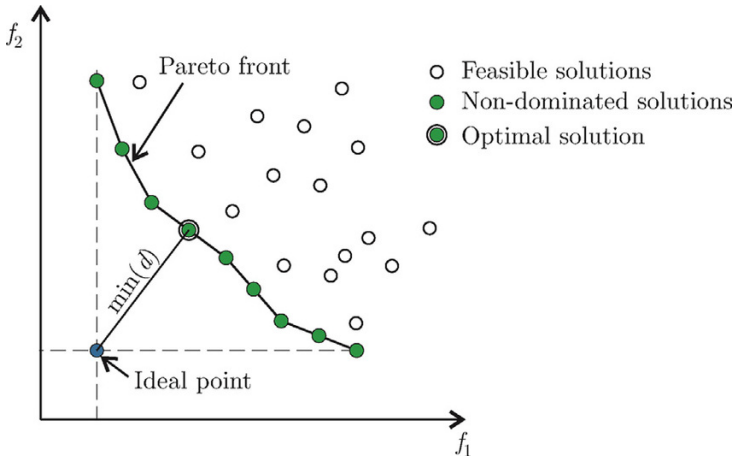


Figura 4.1 Esquema da frente de pareto, pontos ideal e ótimo, fonte: www.researchgate.net.

A escolha final de uma solução entre as que estão na frente de pareto depende das preferências ou prioridades do decisor, que pode valorizar mais um objetivo em prejuízo do outro, dependendo do contexto específico. Por exemplo, um gestor de um sistema do transporte público pode escolher uma solução que reduza o número dos veículos ao máximo possível, aceitando algum aumento no tempo de espera dos passageiros, ou pode priorizar a qualidade do serviço com mais veículos em operação, equilibrando esses dois fatores conforme necessário.

4.2 Princípio básico

A otimização multi-objetivo é uma abordagem inteligente utilizada para otimizar processos que envolvem múltiplos objetivos contraditórios. O princípio básico desta técnica começa com a inicialização de *N* soluções ou indivíduos, que representam diferentes combinações das variáveis de decisão possíveis para o problema em questão. Cada uma dessas soluções é avaliada através do cálculo do valor da função objetivo, que reflete a qualidade da solução em relação aos objetivos estabelecidos.

Diversas estratégias podem ser aplicadas para criar novas soluções, como os algoritmos genéticos (*GA*), otimização por enxame de partículas (*PSO*), e a Otimização de Lobo Cinzento (*Gray Wolf Optimization*), entre outras. Estas técnicas seguem o princípio de melhoria progressiva, gerando soluções mais adaptadas ao problema através de iterações sucessivas. A cada nova ronda, o objetivo é obter um novo conjunto de soluções que possa competir com as anteriores.

Num cenário da otimização de objetivo único, o critério da seleção é simples: as soluções são avaliadas com base no valor da função objetivo, e aquelas com o valor mais baixo, num problema da minimização, são consideradas as melhores. No entanto, a situação torna-se mais complexa quando se lida com múltiplos objetivos, pois o desafio é como classificar as soluções de forma adequada. A dificuldade da otimização multi-objetivo reside na avaliação da qualidade de cada solução, uma vez que não é possível utilizar um único critério para classifi-

car todas as soluções. Neste caso, as soluções são comparadas em termos do compromisso entre os diferentes objetivos, e utiliza-se frequentemente o conceito da dominância de pareto para identificar as soluções que oferecem um equilíbrio mais satisfatório entre os objetivos, ou seja, aquelas que pertencem à frente de pareto.

Definição 4.2.1 *Uma solução X diz-se dominante da solução Y se e só se:*

- *A solução X não é pior que solução Y em todas as funções objetivo;*
- *A solução X é melhor do que solução Y em pelo menos uma função objetivo.*

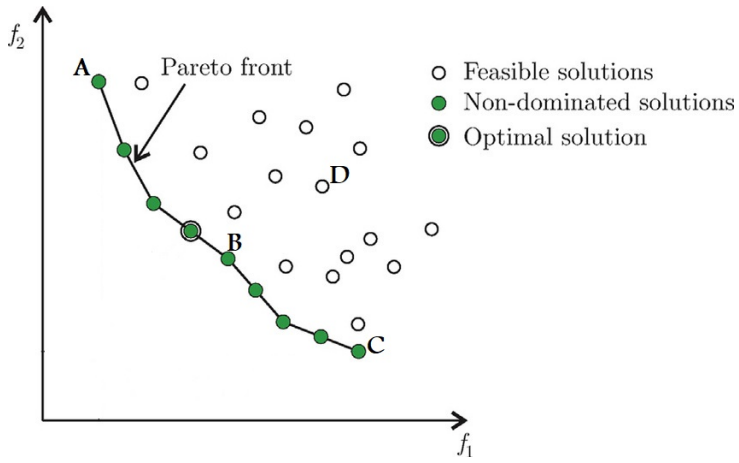


Figura 4.2 Esquema com pontos de exemplo, fonte: https://www.researchgate.net, pontos criados por autor.

Toma-se a Figura 4.2 como exemplo, observa-se que:

- $B \prec D$, o ponto B domina o ponto D, visto que é o minimizado;
- $A \prec D$, o ponto A é estritamente melhor que o ponto D, pois é melhor na função objetivo f_1;
- A e B não se podem dominar.

4.3 Descrição da Estratégia

Esta abordagem da otimização multi-objetivo distingue-se da primeira estratégia. A primeira abordagem não considerava de forma adequada o equilíbrio necessário entre os vários fatores em jogo, nem a redução do tempo de pausa, o que resultava na impossibilidade de encontrar uma distribuição adequada das tarefas aos veículos. Por esse motivo, é necessário adotar uma estratégia mais avançada e robusta, baseada na otimização multi-objetivo.

A estratégia proposta começa por definir um vetor em que cada elemento assume um valor representando a ordem inicial de atribuição das viagens aos veículos. Cada número no vetor corresponde a um veículo específico, e a sua posição indica a sequência de viagens que aquele veículo deve realizar.

Inicialmente, parte-se de um vetor simples, como $[0, 0, 0, 0, 0, 0]$, em que todos os elementos assumem o valor 0. O comprimento do vetor é suficientemente grande para cobrir todos as viagens predefinidos. Este vetor inicial é apenas uma representação básica e será otimizado para melhorar a distribuição das viagens entre os veículos.

Relativamente aos algoritmos, anteriormente o problema foi tratado com uma abordagem da otimização uni-objetivo, utilizando o SA, com o objetivo de minimizar o número de autocarros necessários para cobrir todos os horários disponíveis. A função da avaliação focava-se exclusivamente neste critério, procurando soluções que utilizassem o menor número possível de autocarros.

No entanto, ao focar apenas na minimização do número de autocarros, outro aspeto importante como a minimização do tempo de inatividade entre os serviços, foram desconsiderados. Isso pode levar a soluções desequilibradas, onde alguns autocarros ficam sobrecarregados e outros subutilizados.

Agora, com a transição para a otimização multi-objetivo, utilizando o algoritmo NSGA-II, é mantido o objetivo inicial de minimizar o número de autocarros, mas ainda adicionam-se novos objetivos, como equilibrar o número de viagens atribuídas a cada veículo e minimizar o tempo de inatividade entre os serviços. A função de avaliação foi expandida para refletir estes múltiplos critérios, permitindo encontrar soluções que oferecem um melhor compromisso entre os vários objetivos da empresa. O NSGA-II irá identificar um conjunto de soluções eficientes, representando diferentes compromissos (*trade-offs*) entre os objetivos em conflito.

4.4 *NSGA-II*

Nesta secção, será apresentada a implementação do algoritmo NSGA-II (*Non-dominated Sorting Genetic Algorithm II*), que foi escolhido como a abordagem de otimização multi-objetivo para resolver o problema em estudo. O NSGA-II, proposto por *Goldberg* (1998), é amplamente reconhecido pela sua eficiência e capacidade de identificar um conjunto de soluções ótimas, permitindo explorar diferentes compromissos entre os múltiplos objetivos (13).

Serão explicados dois conceitos fundamentais que sustentam o funcionamento do NSGA-II (10): o Distância de aglomeração (*Crowding Distance*), que ajuda a selecionar as soluções ao longo da frente de pareto, e o Classificação não-dominada (*Non-dominated Sorting*), que organiza as soluções com base no princípio da dominância de pareto. Além disso, será descrito detalhadamente o processo de execução do NSGA-II, desde a geração inicial de soluções até à sua evolução ao longo das iterações.

O NSGA-II é um dos algoritmos mais populares para a resolução de problemas de otimização multi-objetivo. Baseado em conceitos de algoritmos genéticos, ele utiliza uma abordagem evolucionária para explorar o espaço das soluções, com o objetivo de encontrar um conjunto de soluções ótimas.

O NSGA-II distingue-se pelo uso de dois mecanismos fundamentais:

- Classificação não-dominada: as soluções são classificadas com base no conceito da

dominância de pareto, onde uma solução é melhor do que outra se melhorar pelo menos um objetivo sem piorar os restantes.

- Distância de aglomeração: Um método que mede a densidade das soluções em torno de uma solução, permitindo preservar a diversidade ao longo da fronteira de pareto, evitando a concentração de soluções num único ponto.

Classificação não-dominada

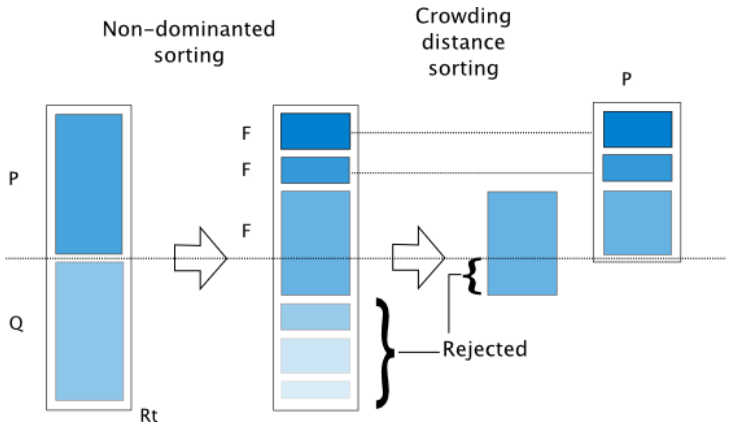


Figura 4.3 Esquema de Classificação não-dominada, fonte: `https://pymoo.org/algorithms/moo/nsga2.html`, visitado em 30 de junho de 2024.

A Classificação não-dominada é um dos principais componentes do algoritmo NSGA-II e refere-se ao processo de organização de soluções (ou indivíduos) com base no conceito da dominância de pareto. Este método classifica as soluções em diferentes frentes de pareto, permitindo identificar aquelas que oferecem um bom compromisso entre múltiplos objetivos (14).

Ela organiza todas as soluções de uma população em várias frentes, onde a primeira frente é composta pelas soluções não dominadas, a segunda frente contém soluções dominadas apenas pelas da primeira frente, e assim sucessivamente.

O processo de Classificação não-dominada funciona da seguinte forma:

1. Para cada indivíduo, verifica-se se ele é dominado por outro da população. Se não for dominado por nenhum, é classificado na primeira frente.

2. Repetir o processo para identificar as soluções dominadas apenas pelas da primeira frente, e serão classificadas na segunda frente.

3. O processo continua até que todas as soluções sejam classificadas em frentes sucessivas, com as soluções nas primeiras frentes sendo as mais desejáveis.

É essencial para a seleção de soluções numa população, pois ajuda a priorizar as que estão mais próximas da fronteira de pareto. Cada indivíduo é comparado pela classificação, sendo que as soluções nas frentes mais baixas (como a primeira frente) têm prioridade. Se

os dois indivíduos estiverem na mesma frente, aí a distância de aglomeração é utilizada como critério secundário para selecionar aqueles que estão mais afastados das outras soluções.

Distância de aglomeração

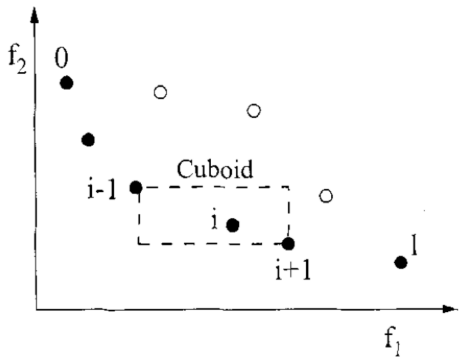


Figura 4.4 Esquema da Distância de Aglomeração, fonte: `https://pymoo.org/algorithms/moo/nsga2.html`, visitado em 30 de junho de 2024.

No NSGA-II, os indivíduos são inicialmente classificados por frentes de pareto, ou seja, com base no nível de dominância. No entanto, pode surgir uma situação em que, dentro de uma dada frente, seja necessário dividir os indivíduos, pois nem todos poderão sobreviver para a próxima geração. Nesse caso, as soluções são selecionadas com base na distância de aglomeração.

A distância de aglomeração é medida utilizando a distância de *Manhattan* no espaço objetivo, e serve para preservar a diversidade das soluções. A fórmula da distância é dada pelas diferenças normalizadas para duas funções f_1 e f_2 (12):

$$d_i^1 = \frac{f_1(x_{i+1}) - f_1(x_{i-1})}{f_1^{max} - f_1^{min}} \tag{4.1}$$

$$d_i^2 = \frac{f_2(x_{i+1}) - f_2(x_{i-1})}{f_2^{max} - f_2^{min}} \tag{4.2}$$

$$d_i = d_i^1 + d_i^2 \tag{4.3}$$

A fórmula 4.1 mede a variação de f_1 ao redor de um ponto x_i, normalizada pelo intervalo total de f_1 e a fórmula 4.2 mede a variação de f_2 ao redor do mesmo ponto x_i, também normalizada pelo intervalo total de f_2. Por fim, a equação 4.3 combina as contribuições das duas diferenças normalizadas, representando uma métrica geral que incorpora variações relativas de ambas as funções ao redor de x_i.

As soluções que se encontram mais afastadas de outras têm maior probabilidade de serem selecionadas, evitando a concentração de soluções em determinadas áreas do espaço objetivo, pois significa uma solução relativamente única. Além disso, os pontos extremos da frente de pareto, que representam os limites dos objetivos, são preservados em cada geração, recebendo uma distância de aglomeração infinita para garantir a sua sobrevivência.

Processo do Algoritmo

O código (Anexo B.11) implementa uma solução baseada no algoritmo NSGA-II para otimização multi-objetivo no problema de atribuição de tarefas para veículos de transporte público. O processo envolve atribuir viagens a veículos enquanto se considera múltiplos objetivos.

1. Definição do Problema $MyProblem$

- A classe $MyProblem$ herda de $ElementwiseProblem$, é uma classe usada para definir problemas de otimização elementar (avaliação de um indivíduo por vez).

- **Parâmetros de entrada:** $diretorio$, $rota$, $service\_id$. Estes parâmetros são usados para carregar os dados das rotas que serão otimizadas.

- As **variáveis de decisão** são armazenadas num dicionário x. São 19 variáveis:

- $x1$: minutos de pausa (varia de 1 a 20 minutos).
- $x2$ a $x4$: índice de viagem a atribuir ao veículo correspondido, sendo a primeira viagem disponível de 0.
- $x5$ a $x19$: fixadas em 0, correspondem à atribuição da primeira viagem disponível a estes veículos.

- O problema tem 4 objetivos ($n\_obj = 4$) e nenhuma restrição às variáveis.

2. Função de Avaliação $\_evaluate$

- Esta função recebe um vetor de variáveis de decisão e retorna o valor dos 4 objetivos.

- É utilizada a função $funcFF$ para minimizar os 4 objetivos, que são baseados em:

- $A1$: a maior diferença de horários entre as viagens atribuídas aos veículos.
- $A2$: a diferença no número de tarefas atribuídas aos veículos.
- $A5$: o número de veículos utilizados.
- $-minutos$: o negativo do tempo de pausa (para maximizar as pausas).

3. Carregamento dos Dados: $carregar\_tabela$

- Esta função carrega a tabela de informações de viagens a partir de um arquivo CSV.

- A pausa entre viagens é convertida de minutos para horas e adicionada ao tempo do início das viagens.

4. Atribuição de Tarefas: $atribuir\_tarefas$

- Esta função é responsável por atribuir as viagens aos veículos de acordo com a ordem de execução fornecida.

- A função interna $atrb\_bus\_j$ verifica se uma viagem pode ser atribuída a um veículo com base no tempo de pausa e na direção da viagem.

- O dicionário $equipas\_tarefas$ armazena quais viagens são atribuídas a cada veículo.

5. Funções Objetivo

- $encontrar\_maior\_diferenca$: Esta função calcula a maior diferença de tempo entre viagens consecutivas para cada veículo.

- $encontrar\_diferenca\_tarefas$: Calcula a diferença no número de viagens atribuídas a diferentes veículos.

Algoritmo NSGA-II

1. População Inicial: O número de soluções geradas usando o operador de amostragem aleatória de valores inteiros ($IntegerRandomSampling$).

2. Operadores Evolutivos:

   - Crossover: *Simulated Binary Crossover* (SBX) para combinar duas soluções e criar novas soluções.

   - Mutação: A mutação polinomial (PM) que introduz pequenas mudanças nas soluções.

   - Reparação: O operador de reparação ($RoundingRepair$) garante que as variáveis de decisão sejam arredondadas.

3. Critério de Paragem: O algoritmo é configurado para parar após n_gen gerações.

Fluxo do Algoritmo

1. Inicialização:

   - O algoritmo começa definindo o problema e a função de avaliação.

   - Em seguida, carrega os dados da rota e das viagens e considera o tempo de pausa entre as viagens.

2. Avaliação de Soluções:

   - A cada geração, o NSGA-II gera novas soluções e as avalia com base nos 4 objetivos.

   - A função $funcFF$ é chamada para cada solução, calculando o número de veículos utilizados, a diferença entre tarefas e a maior diferença de tempo entre viagens.

3. Seleção e *Crossover*:

   - As soluções são selecionadas e recombinadas usando o operador SBX, que gera novas soluções a cada geração.

   - A mutação é aplicada para garantir diversidade na população.

4. Terminando a Execução:

   - Após o número das gerações definidas, o algoritmo retorna as melhores soluções encontrada que equilibram todas as funções objetivo.

4.5 Resultados

Exemplo 1

Considerando a rota 100_0 com o serviço *Inverno_Util*_20240229, entre Martim Moniz e Sacavém, encontram-se 130 viagens distribuídas entre as 5 : 30 da manhã e as 9 : 30 da tarde. O resultado da resolução realizada da atribuição de ordem $[0, 0, 0, 0, ...]$, a partir da primeira viagem disponível em cada iteração, pode ser visualizada pela Figura 4.5.

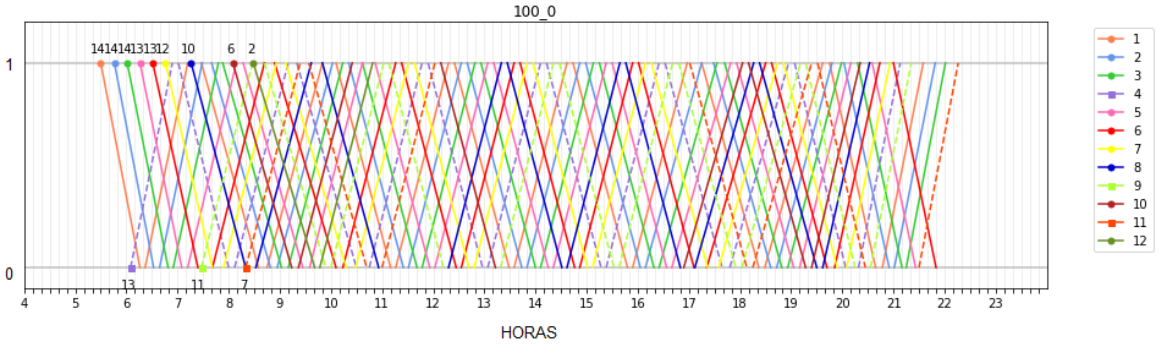


Figura 4.5 Representação gráfica da atribuição das viagens da rota 100_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 5 minutos.

Com a atribuição a partir da primeira viagem disponível, para esta rota são necessários 12 veículos. Executa-se este exemplo com o NSGA-II, com os seguintes valores dos parâmetros:

```
algorithm = NSGA2(
    pop_size = 20,
    n_offsprings = 20,
    sampling = IntegerRandomSampling(),
    crossover = SBX(prob=0.9,eta=20,repair=RoundingRepair()),
    mutation = PM(prob=0.05, eta=20,repair=RoundingRepair()),
    eliminate_duplicates = True
)
```

com `pop_size=20`, o número de soluções seria 20, para `n_offsprings=20`, são criadas 20 descendentes gerados em cada geração, as variáveis de decisão das soluções vão ser arredondas para inteiros. O cruzamento SBX tem `prob=0.9`, ou seja, em 90% dos casos dois indivíduos se cruzarão para gerar descendentes, e a mutação PM tem `prob=0.05`, cada variável tem 5% de sofrer mutação. E, por fim, evitar soluções duplicadas (`eliminate_duplicates = True`).

Após 20 gerações (`termination = get_termination("n_gen",20)`), com semente de aleatoriedade número 42, alguns resultados das quatro funções objetivo são apresentados na Tabela 4.1.

Nºsol	N_vehicle	Max_dif_trf	Maior_inat	Pausa
1	12	12	6.1267	5
2	12	12	6.1433	4
3	12	13	6.4067	2
4	12	12	6.4233	1
5	12	11	7.2767	5
6	13	12	4.1100	6
7	13	8	6.4767	8
8	13	10	7.0033	7
9	14	10	6.2233	13
10	14	10	5.7067	17
11	14	10	5.9533	16
12	14	10	5.9867	14
13	14	10	5.6733	19

Tabela 4.1 Tabela com algumas soluções não-dominadas obtidas com NSGA-II da rota 100_0 do tipo de serviço $Inverno\_Util$ _20240229.

Na Tabela 4.1 são apresentados alguns dos resultados obtidos com o NSGA-II, correspondentes a soluções não dominadas. A primeira linha mostra os valores dos quatro objetivos para a variável de decisão $[5, 17, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$, onde ao primeiro veículo foi atribuído a décima sétima viagem, ao segundo a terceira viagem disponível dos dados após a atribuição do veículo '1', e aos restantes veículos a primeira viagem da cada iteração. Nessa linha, observamos os valores $[12, 12, 6.1267, 5]$, correspondentes aos objetivos 1 a 4, onde o 1.º objetivo refere-se a 12 veículos, o 2.º objetivo apresenta um máximo de 12 viagens de diferença entre os veículos, o 3.º objetivo mostra um máximo de 6.13 horas de inatividade entre as viagens de algum veículo, e o 4.º objetivo corresponde a um tempo de pausa de 5 minutos.

Para o primeiro objetivo — a minimização da quantidade de veículos ($N\_vehicle$) — as soluções variaram entre 12 e 14 veículos. A maior diferença no número de viagens atribuídas a cada veículo (Max_dif_trf) esteve entre 8 e 13. O menor desequilíbrio na distribuição de viagens foi obtido com 13 veículos (linha 7 da tabela), onde a diferença máxima de viagens foi de 8.

O tempo máximo de inatividade ($Maior\_inat$) variou entre 4.11 e 7.28 horas. O menor tempo de inatividade, 4.11 horas, ocorreu também com 13 veículos, com tempo de pausa de 6 minutos e diferença de 12 viagens entre os veículos.

O tempo de pausa variou de 1 a 19 minutos. Verifica-se que, mesmo com pausas mais curtas, como 1 minuto, os tempos de inatividade podem ser elevados, especialmente com 12 veículos. A pausa mais longa, 19 minutos, ocorreu com 14 veículos e não resultou numa redução significativa do tempo de inatividade.

A atribuição de viagens dos veículos para o caso de variável $[17, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ é apresentada na Figura 4.6.

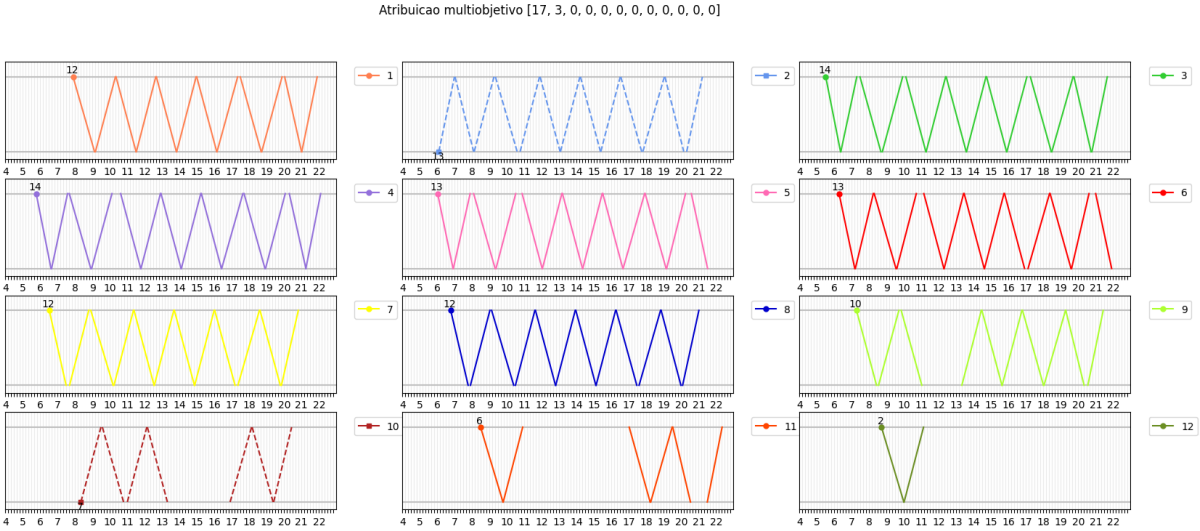


Figura 4.6 Representação gráfica da atribuição de cada veículo da rota 100_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 7 minutos.

Comparando este último caso com o da variável $[14, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ (Figura 4.7), não se consegue determinar qual deles é melhor para o problema, pois ambos apresentam vantagens próprias. Um com menos veículos mas o tempo de pausa é menor, e o outro com maior tempo de pausa mas consequentemente o número de veículos aumenta.

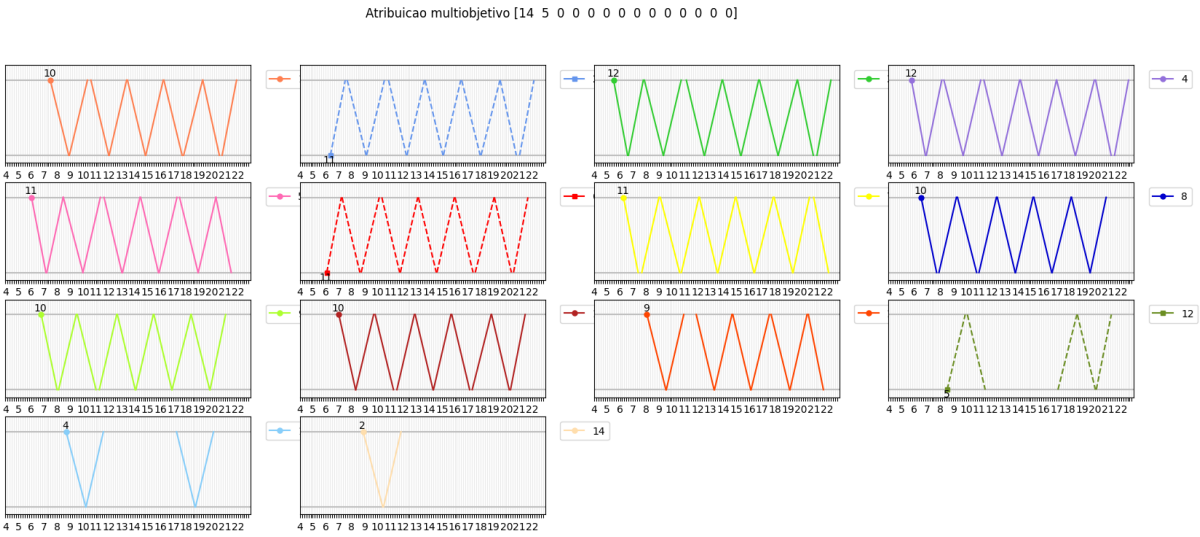


Figura 4.7 Representação gráfica da atribuição de cada veículo da rota 100_0 do tipo de serviço *Inverno_Util*_20240229, com tempo de pausa de 19 minutos.

Exemplo 2

Outro exemplo de rota é a 79_0, entre Corpo Santo e Campo de Ourique. Seja o tipo de serviço *Inverno_Util*_20240229, com 84 viagens. Com tempo de pausa de 5 minutos, o número de veículos determinado pelo vetor da índice de viagem $[0, 0, 0, 0, \ldots]$ é 4. Aplica-se ao algoritmo de NSGA-II os mesmos parâmetros do primeiro exemplo (Código na página 49). Continuando

com 20 gerações e semente 42, alguns resultados das 4 funções objetivo são apresentados na Tabela 4.2.

Nºsol	N_vehicle	Max_dif_trf	Maior_inat	Pausa
1	3	5	0.6834	2
2	4	27	0.2667	3
3	4	21	0.6667	3
4	5	26	0.1667	1
5	5	4	0.2667	20
6	5	4	0.2667	20
7	5	3	0.3667	20
8	5	2	0.5501	20
9	5	1	0.7834	20

Tabela 4.2 Tabela com algumas soluções não-dominadas obtidas com NSGA-II da rota 79_0 do tipo de serviço $Inverno\_Util$ _20240229.

Analisando os as soluções não-dominadas, consegue-se identificar qual é a melhor solução para cada um dos objetivos de forma individual. Nomeadamente, a primeira solução tem o número mínimo de veículos, $N\_vehicle = 3$. A última solução apresenta o melhor equilíbrio entre o número de viagens, com $Max\_dif\_trf = 1$. O menor tempo de inatividade está apresentado na quarta solução, com $Maior\_inat = 0.1666...$ horas. E o maior tempo de pausa, com $Pausa = 20$ minutos, é observado nas soluções com 5 veículos.

Verifica-se que o algoritmo melhorou no sentido do número de veículos. Em comparação com o caso da rota 100_0, na rota 79_0 o número de veículos diminuiu de 4 para 3. Consequentemente, o tempo de pausa também diminui, de 3 minutos a 2 minutos.

A quarta solução, com o menor tempo de inatividade, é apresentada na Figura 4.8. Observa-se que, para atingir o objetivo da minimização do maior tempo de inatividade, o algoritmo atribui apenas uma viagem aos veículos '4' e '5', causando uma grande diferença entre o número de viagens.

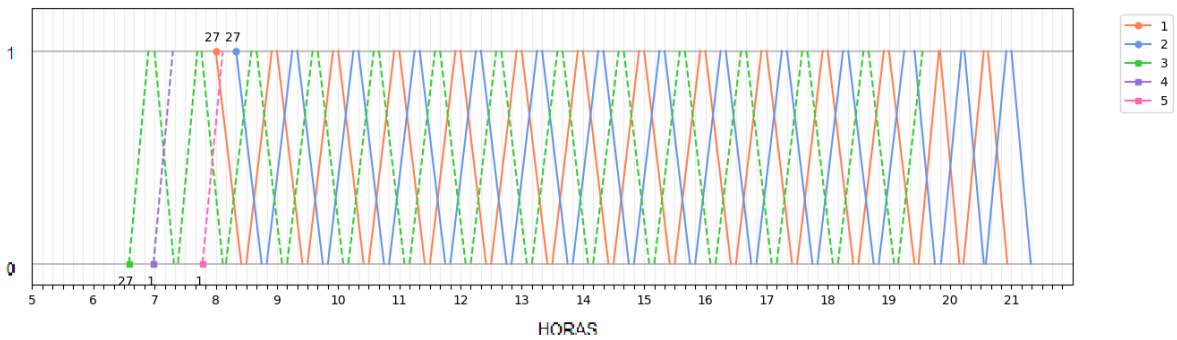


Figura 4.8 Representação gráfica da atribuição de cada veículo da rota 79_0 do tipo de serviço $Inverno_Util$_20240229, com 5 veículos e tempo de pausa de 1 minuto.

A Figura 4.9 apresenta a primeira solução, na qual o número de veículos foi minimizado. Como consequência, o tempo de pausa é quase mínimo, isto leva à preocupação sobre o tempo insuficiente para compensar possíveis atrasos no tráfego da última viagem.

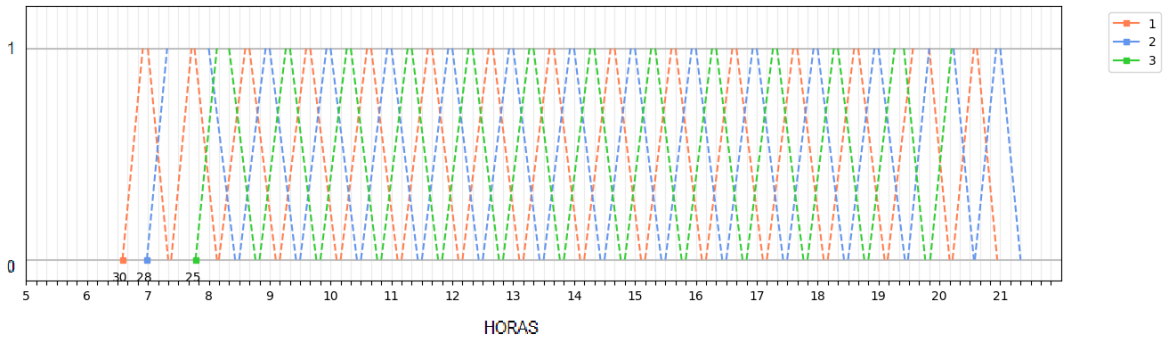


Figura 4.9 Representação gráfica da atribuição de cada veículo da rota 79_0 do tipo de serviço *Inverno_Util*_20240229, com 3 veículos e tempo de pausa de 2 minutos.

As soluções com 5 veículos parecem ser as mais equilibradas entre as funções objetivo. Por exemplo, para a última solução, na Figura 4.10, são atingidos dois objetivos: o tempo de pausa máximo e a diferença mínima entre as viagens. Ou, para a quinta solução, mesmo tendo uma diferença entre as viagens de 4, o tempo de inatividade é o segundo menor das soluções.

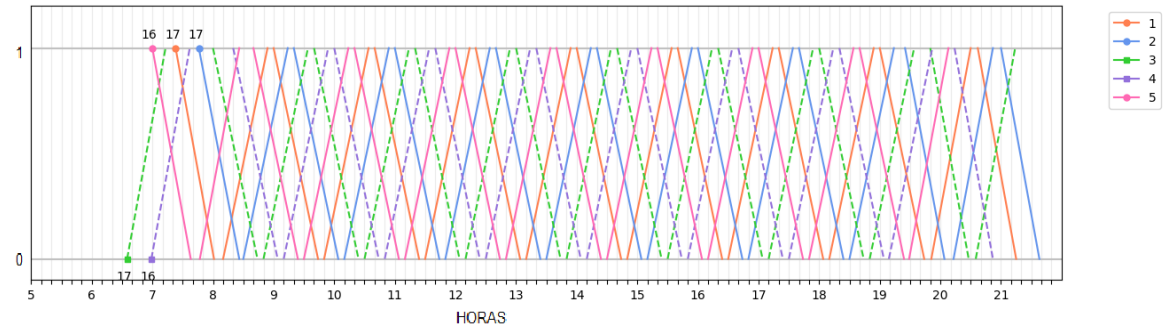


Figura 4.10 Representação gráfica da atribuição de cada veículo da rota 79_0 do tipo de serviço *Inverno_Util*_20240229, com 5 veículos e tempo de pausa de 20 minutos.

Portanto, a resolução do problema de otimização multi-objetivo gera um conjunto de soluções não dominadas, ou seja, todas estas soluções têm a mesma relevância matemática. Cabe ao decisor escolher qual a solução que melhor se adequa às suas necessidades específicas e à resolução do seu problema. Conclui-se que a otimização multi-objetivo suporta um equilíbrio entre o número de veículos, o tempo de inatividade, a distribuição de viagens e o tempo de pausa.

Capítulo 5

Conclusão

5.1 Resumo dos Resultados

Este estudo explora a aplicação de técnicas de otimização ao planeamento da afetação dos veículos de transportes públicos, com o objetivo de minimizar o número de veículos necessários e organizar a atribuição dos horários de percursos.

Foram utilizadas duas abordagens: o método de recozimento simulado para o caso de objetivo único, que se centra na eficiência nas permutações das ordens, minimizando do número de veículos; e o algoritmo NSGA-II para o caso de objetivo múltiplo, que considera quatro critérios em simultâneo. No algoritmo de recozimento simulado, as soluções são exploradas através do ajuste de parâmetros (por exemplo, tempo de pausa e ordem de atribuição de tarefas) para chegar a uma solução que minimize o número de autocarros. No NSGA-II, a abordagem multi-objetivo otimiza vários critérios conflituosos, como o número de veículos, a distribuição do tempo de inatividade e a carga de tarefas por veículo.

A solução gerada pelo NSGA-II provou ser bastante eficaz no tratamento do problema multi-objetivo, encontrando um equilíbrio entre diferentes objetivos. Em particular, o número de veículos utilizados é reduzido da atribuição sequencial, as tarefas são distribuídas de forma mais uniforme e o tempo de inatividade é reduzido.

Tanto o recozimento simulado como o NSGA-II são métodos eficazes para otimizar o planeamento de redes de transportes públicos. O Simulated Annealing é eficaz para problemas de objetivo único, enquanto o NSGA-II destaca-se na otimização multi-critério, fornecendo uma solução mais equilibrada e robusta.

5.2 Trabalhos Futuros

Apesar das soluções encontradas serem satisfatórias, há várias oportunidades para melhorar e expandir o presente estudo. Uma das direções possíveis é a tentativa de outras abordagens para o algoritmo SA no sentido de melhorar o único objetivo de minimizar o número de veículos utilizados.

Uma maneira de reduzir o número de veículos seria introduzir mais alternativas, como

aqueles que retornam ao ponto de partida para recolher passageiros para a viagem seguinte.

Outra área a explorar é a aplicação de diferentes técnicas de otimização. Embora o Simulated Annealing e o NSGA-II tenham demonstrado eficácia, a implementação de outros algoritmos, como os baseados em enxame de partículas (PSO), algoritmos evolutivos ou até métodos híbridos, poderia revelar novas soluções ou melhorias adicionais no desempenho.

Adicionalmente, a integração das técnicas de otimização com sistemas de dados em tempo real é uma proposta promissora. Esta integração permitiria que o planeamento se ajustasse de forma dinâmica à medida que surgissem novas informações, como condições de tráfego ou alterações operacionais.

Por fim, poderiam ser considerados novos objetivos no problema de otimização, como a minimização dos custos operacionais, a definição do número de viagens para determinado veículo ou até a maximização do conforto dos passageiros. A inclusão destes fatores traria uma perspetiva mais abrangente e sustentável.

Anexo A

Preparação dos dados

A.1 GTFS

GTFS consiste em duas partes principais: GTFS *Schedule* e GTFS *Realtime*. Neste trabalho apenas foi utilizado o GTFS *Schedule*.

O GTFS *Schedule* contém informações sobre rotas, horários, tarifas e detalhes geográficos de trânsito e é apresentado em ficheiros de texto simples. Este formato simples permite fácil criação e manutenção sem depender de software complexo ou proprietário. O GTFS *Realtime* contém atualizações de viagem, posições de veículos e alertas de serviço.

Os ficheiros que compõem um conjunto de dados GTFS *Schedule* devem ser estruturados e formatados de certa forma. São aplicadas condições de presença aos ficheiros de GTFS, nomeadamente, obrigatório, opcional e condicionalmente necessário. Os ficheiros obrigatórios devem ser incluídos no conjunto de dados e conter um valor válido para cada registo. Os ficheiros opcionais podem ser omitidos, e os ficheiros condicionalmente necessários devem ser incluídos nas condições descritas (4).

Ficheiros obrigatórios	
agency.txt	Agências de trânsito com serviço representado neste conjunto de dados.
stops.txt	Paragens onde os veículos apanham ou deixam os passageiros. Também define as estações e entradas das estações.
routes.txt	Rotas de trânsito. Uma rota é um grupo de viagens que são exibidas aos passageiros como um único serviço.
trips.txt	Viagens para cada rota. Uma viagem é uma sequência de duas ou mais paragens que ocorrem durante um período do tempo específico.
stop_times.txt	Horários em que um veículo chega e sai das paragens para cada viagem.

Tabela A.1 Tabela de ficheiros obrigatórios de GTFS.

Condicionalmente necessários	
calendar.txt	Datas dos serviços usando um esquema semanal com datas de início e fim. É necessário quando as datas dos serviços não estão definidos no *calendar_dates.txt*. Um serviço é o tipo de diário em que as viagens deverão ser realizado por veículos.
calendar_dates.txt	Datas em excessão das que estão definidas no *calendar.txt*. É necessário quando *calendar.txt* é omitido, neste caso *calendar_dates.txt* deve conter todas as datas dos serviços.
levels.txt	Nível de andares das estações.

Tabela A.2 Tabela de ficheiros condicionalmente necessários de GTFS.

Alguns opcionais relevantes	
fare_attributes.txt	Informações dos bilhetes das rotas da agência de trânsito.
shapes.txt	Alinhamento das rotas - a localização geográfica exata das curvas das rotas.
feed_info.txt	Meta dados do conjunto de dados, incluindo informações sobre a edição, a versão e a expiração.

Tabela A.3 Tabela de ficheiros opcionais de GTFS.

Em cada ficheiro, estão contidos os atributos associados. Nestes ficheiros, é fundamental considerar um tipo de atributo específico, chamada a chave primária (CP). Este tipo de atributo tem como caraterística o identificador único dos dados, que facilita a interligação entre os dados dos diferentes ficheiros.

agency.txt	
agency_id (CP)	o código único de identificação da marca da agência de trânsito.
agency_name	o nome completo da agência de trânsito.
agency_url	o endereço da web da agência de trânsito.
agency_timezone	o fuso horário onde a agência se localiza.
agency_lang	a língua principal usada pela agência.

Tabela A.4 Tabela do ficheiro *agency.txt* de GTFS.

stops.txt	
stop_id (CP)	o código único de cada paragem. Várias rotas poderão usar o mesmo *stop_id*.
stop_code	um texto curto ou um número que identifica a paragem para os passageiros.
stop_name	o nome da localização. É o nome indicado no horário impresso na cada paragem.
stop_lat	a latitude da localização.
stop_lon	a longitude da localização.

Tabela A.5 Tabela do ficheiro *stops.txt* de GTFS.

routes.txt	
route_id (CP)	o código único de cada rota.
agency_id	a agência correspondente da rota específica.
route_short_name	o nome curto da rota, para que os passageiros facilmente se identificam as rotas.
route_long_name	o nome longo da rota, geralmente mais descritivo.

Tabela A.6 Tabela do ficheiro *routes.txt* de GTFS.

trips.txt	
route_id	a rota correspondente à viagem.
service_id	o conjunto de datas para quando o serviço é aplicável para a viagem.
trip_id (CP)	o código único de cada viagem.
direction_id	a direção da viagem. 0 - viaje numa direção; 1 - viaje na direção contrária.
shape_id	o alinhamento das rotas correspondente à viagem.

Tabela A.7 Tabela do ficheiro *trips.txt* de GTFS.

stop_times.txt	
trip_id	a viagem correspondente.
arrival_time	o tempo de chegada do veículo à paragem.
departure_time	o tempo de partida do veículo da paragem.
stop_id	a paragem correspondente.
stop_sequence	a sequência por ordem crescente das paragens da viagem.
shape_dist_traveled	distância real viajada ao longo do alinhamento da rota.

Tabela A.8 Tabela do ficheiro *stop_times.txt* de GTFS.

calendar.txt	
service_id (CP)	o código único de cada serviço.
monday	se o serviço funciona todas as segundas-feiras entre *start_date* e *end_date*. 0 - o serviço não está disponível para todas as segundas-feiras dentro do intervalo da data; 1 - o serviço está disponível para todas as segundas-feiras dentro do intervalo da data.
tuesday	funciona da mesma forma que *monday*, exceto só se aplica às terças-feiras.
wednesday	funciona da mesma forma que *monday*, exceto só se aplica às quartas-feiras.
thursday	funciona da mesma forma que *monday*, exceto só se aplica às quintas-feiras.
friday	funciona da mesma forma que *monday*, exceto só se aplica às sextas-feiras.
saturday	funciona da mesma forma que *monday*, exceto só se aplica às sábados.
sunday	funciona da mesma forma que *monday*, exceto só se aplica às domingos.
start_date	a data de início da operação do serviço.
end_date	a data de fim da operação do serviço.

Tabela A.9 Tabela do ficheiro *calendar.txt* de GTFS.

levels.txt	
level_id (CP)	o nível de piso na estação.
level_index	o índice numérico do piso indicando a posição relativa.

Tabela A.10 Tabela do ficheiro *calendar_dates.txt* de GTFS.

fare_attributes.txt	
fare_id (CP)	a classe da tarifa.
price	o preço da tarifa, na unidade específica por *currency_type*.
currency_type	moeda usada para pagar a tarifa.
payment_method	quando a tarifa deve ser pago. 0 - a tarifa é pago no caminho; 1 - a tarifa tem de ser pago antes de partida.
agency_id	a agência relevante para a tarifa.

Tabela A.11 Tabela do ficheiro *fare_attributes.txt* de GTFS.

shapes.txt	
shape_id (CP)	o código único do alinhamento.
shape_pt_lat	a latitude do ponto do alinhamento. Cada dado do ficheiro representa um ponto do alinhamento usado para definir a forma.
shape_pt_lon	a longitude do ponto do alinhamento.
shape_pt_sequence	a sequência por ordem crescente dos pontos de alinhamento da rota.

Tabela A.12 Tabela do ficheiro *shapes.txt* de GTFS

feed_info.txt	
feed_publisher_name	o nome completo da organização que publica os dados.
feed_publisher_url	o endereço da web da organização que publica os dados.
feed_lang	a língua por omissão usada para os textos nos dados.

Tabela A.13 Tabela do ficheiro *feed_info.txt* de GTFS

A figura a seguir, A.1, apresenta um diagrama da entidade do relacionamento (ER) que ilustra a conexão e a dependência entre os atributos dos ficheiros. Como visto nas tabelas anteriores, existem atributos em alguns ficheiros que estão nos outros ficheiros, e daí haver uma ligação entre este ficheiros, e esses atributos são chamados de chave estrangeira. Este diagrama foi elaborado com o intuito de fornecer uma representação visual clara das relações entre eles, permitindo uma melhor compreensão e análise da estrutura e do funcionamento do sistema.

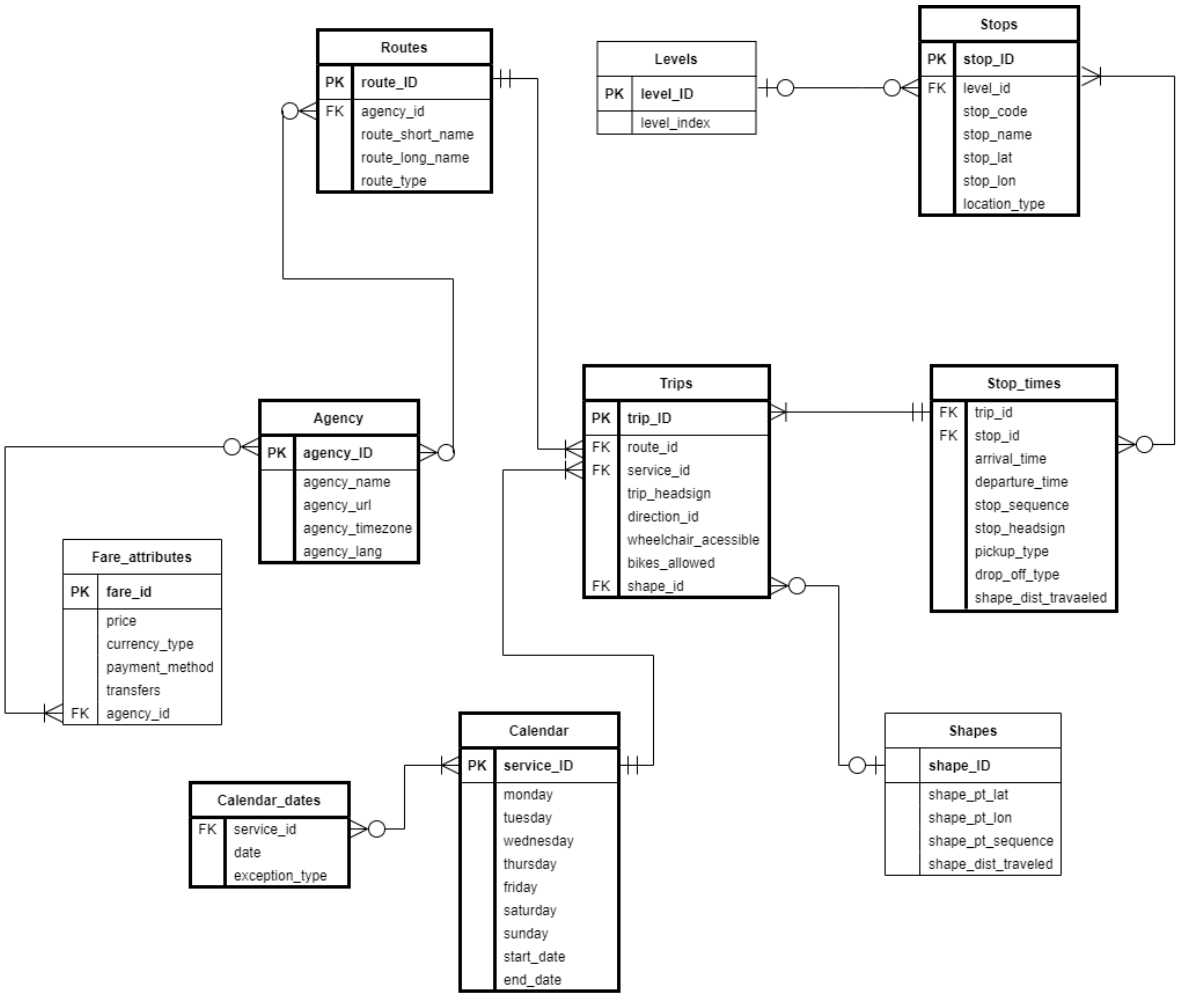
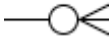


Figura A.1 Diagrama entidade relacionamento dos ficheiros GTFS.

O diagrama mostra as relações entre as sete variáveis principais e três variáveis opcionais: *agency*, *stops*, *routes*, *trips*, *stop_times*, *calendar* e *calendar_dates*, *shapes*, *levels* e *fare_attributes*. As relações entre as variáveis ocorrem devido à inclusão de atributos de outras variáveis. As retas apresentadas no diagrama indicam as relações entre os atributos das variáveis respetivas, com setas de tipos diferentes para diferentes ocasiões:

 multiplicidade de muitos opcional

 multiplicidade de muitos obrigatório

 multiplicidade de um obrigatório

 multiplicidade de um opcional

A variável *agency* do ficheiro *agency.txt* não tem chaves estrangeiras, por isso não existe relacionamento para fora. A variável *stops* do ficheiro *stops.txt* tem uma chave estrangeira *level_id* de *levels*, indica que as paragens têm informação sobre o nível de piso relacionado, em que um nível de piso pode corresponder a várias paragens diferentes opcionalmente, e uma paragem por opção poderá ter um nível de piso correspondente. A variável *routes* do ficheiro *routes.txt* tem uma chave estrangeira *agency_id* de *agency.txt*, indica que a rota se corresponde a uma agência de trânsito. Neste caso, uma agência poderá relacionar com várias rotas e uma rota poderá relacionar com várias agências, sem ser obrigatório. A variável *trips* do ficheiro *trips.txt* tem três chaves estrangeiras: *route_id* de *routes*, *service_id* de *calendar* e *shape_id* de *shapes*, cada valor destes corresponde a entradas com o mesmo valor de identificação. Ambos *route_id* e *service_id* de *trips* são obrigatórios. Uma viagem corresponde a uma única rota e um único serviço, enquanto que cada rota ou cada serviço é obrigatório ter uma ou mais viagens. A conexão entre *trips* e *shapes* já é opcional para os dois lados, mas de mesma forma, uma viagem apenas terá um alinhamento de rota e ele poderá referir a várias viagens. A variável *stop_times* do ficheiro *stop_times.txt* não tem nenhuma chave primária, pelo contrário, contém duas chaves estrangeiras, *trip_id* de *trips* e *stop_id* de *stops*. Cada entrada desta é obrigatório ter conexão com dados de *trips* e *stops*, e cada viagem é indispensável corresponder a um ou mais horários de paragens planeadas. No entanto as paragens não são obrigadas a ter informações de horários. Para a situação da *calendar_dates* do ficheiro *calendar_dates.txt*, como esta apenas existe quando ocorre uma exceção de serviço para além daqueles contidos na *calendar* do ficheiro *calendar.txt*, somente existe uma chave estrangeira correspondida com *calendar*. E devido a especificidade desta variável, nela é obrigatório ter um *service_id* de *calendar* para haver a correspondência, mas é opcional que um *service_id* de calendar tivesse um *service_id* de *calendar_dates*, isto é, poderá existir *service_id* de *calendar* que não se encontram no *calendar_dates*.

A.2 Exemplos dos dados reais de Carris

agency_id	agency_name	agency_url	agency_timezone	agency_lang
1	Carris	http://www.carris.pt	Europe/Lisbon	pt

Tabela A.14 Tabela de *agency.txt*, fornecido pelo Carris

stop_id	stop_code	stop_name	stop_lat	stop_lon	location_type
50101	14	Externato Champagnat	38.777797	-9.127068	0
50100	13	Externato Champagnat	38.777850	-9.126840	0
50103	30	Alto Chapeleiro	38.787669	-9.163583	0
14723	20	Av. Dr. Arlindo Vicente	38.753624	-9.125934	0
50105	3	Chapeleiro 2	38.792273	-9.157561	0
⋮	⋮	⋮	⋮	⋮	⋮

Tabela A.15 Tabela de exemplo de *stops.txt*, fornecido pelo Carris

route_id	agency_id	route_short_name	route_long_name	route_type
62_0	1	203	Xabregas - Restelo	3
159_0	1	52E	Rua Câmara Pestana - Largo da Anunciada	3
178_0	1	724	Pontinha - Alcântara	3
178_1	1	724	Pontinha - Alcântara - Cç. Tapada	3
178_3	1	724	Pontinha - Estação Sto. Amaro	3
⋮	⋮	⋮	⋮	⋮

Tabela A.16 Tabela de exemplo de *routes.txt*, fornecido pelo Carris

route_id	service_id	trip_id	trip_headsign	direction_id	shape_id
189_0	Inverno_Util_20240229	3566_20240229_189_0_18	NaN	0	189_0_ASC_shp
169_0	Inverno_Util_20240229	6190_20240229_169_0_10	NaN	0	169_0_ASC_shp
76_0	Inverno_DomingoFeriado_20240229	11214_20240229_76_0_15	NaN	1	76_0_DESC_shp
105_0	Inverno_DomingoFeriado_20240229	3934_20240229_105_0_19	NaN	1	105_0_DESC_shp
121_0	Inverno_Sabado_20240229	4141_20240229_121_0_5	NaN	1	121_0_DESC_shp
⋮	⋮	⋮	⋮	⋮	⋮

Tabela A.17 Tabela de exemplo de *trips.txt*, fornecido pelo Carris

trip_id	arrival_time	departure_time	stop_id	stop_sequence	shape_dist_traveled
14946_20240226_75_0_10	17:14:08	17:14:08	10516	19	NaN
1158_20240229_158_0_8	10:24:00	10:24:00	50244	1	NaN
7364_20240229_191_0_11	12:27:43	12:27:43	2713	16	NaN
7416_20240226_124_0_17	13:42:30	13:42:30	9419	23	NaN
3053_20240229_146_0_31	17:03:00	17:03:00	6105	4	NaN
⋮	⋮	⋮	⋮	⋮	⋮

Tabela A.18 Tabela de exemplo de *stop_times.txt*, fornecido pelo Carris

service_id	monday	tuesday	wednesday	thursday	friday	saturday	sunday	start_date	end_date
Inverno_Sabado_20240229	0	0	0	0	0	1	0	20240229	20240324
Inverno_Util_20240226	1	1	1	1	1	0	0	20240226	20240228
Inverno_DomingoFeriado_20240229	0	0	0	0	0	0	1	20240229	20240324
Inverno_Util_20240229	1	1	1	1	1	0	0	20240229	20240324

Tabela A.19 Tabela de *calendar.txt*, fornecido pelo Carris

shape_id	shape_pt_lat	shape_pt_lon	shape_pt_sequence	shape_dist_traveled
62_0_ASC_shp	38.715251	-9.165102	388	NaN
226_0_CIRC_shp	38.725633	-9.112818	190	NaN
226_0_CIRC_shp	38.725693	-9.112760	191	NaN
226_0_CIRC_shp	38.725779	-9.112673	192	NaN
226_0_CIRC_shp	38.725987	-9.112390	193	NaN
⋮	⋮	⋮	⋮	⋮

Tabela A.20 Tabela de exemplo de *shapes.txt*, fornecido pelo Carris

A.3 Pré-processamento dos dados

Após a aquisição dos dados para o planeamento de transportes públicos, surge a necessidade de purificar e preparar esses dados para análise e utilização no futuro. Neste contexto, foi desenvolvido vários códigos em *Python*, abrangendo conceitos fundamentais de base de dados, com o objetivo de extrair informações relevantes para obter as informações para o processo de planeamento.

Depois da extração dos dados, são guardados nos ficheiros criados para ser aplicado na otimização do planeamento. Assim, este capítulo foi explorada a obtenção de dados precisos através do GTFS. O propósito do problema é o planeamento de veículos de transportes públicos, e este planeamento deverá ser feito para veículos que realizariam as viagens no mesmo tipo de dia. Então deverão ser retiradas dos dados de GTFS as viagens pertencentes à rota específica e que realizem o mesmo serviço.

Viagens de cada rota: *Trips - Routes*

Tal como descrito anteriormente, cada rota tem várias viagens por dia. O objetivo neste contexto é então de agrupar as viagens pertencentes à determinada rota.

```
trips[trips['route_id']==route_id]['trip_id']
```

O *route_id* é o identificador único da rota que se pretende visualizar e *trips* a tabela dos dados originais do ficheiro de *trips.txt*. Portanto este programa recebe um *route_id* e vai á coluna *route_id* do *trips* pesquisar aqueles que têm *route_id* iguais. Por fim obtém-se a nova tabela onde a coluna *route_id* apenas terá os valores pretendidos, e retira-se da tabela a coluna de *trip_id* para obter o identificador único de viagens. O código completo da função está no Anexo B.1.

Tomando exemplo a rota com *route_id* de 60_0, as viagens associadas são as seguintes.

```
    route_id    trip_id
0   60_0      12649_20240229_60_0_13
1   60_0      12649_20240226_60_0_13
2   60_0      12651_20240229_60_0_11
3   60_0      12657_20240226_60_0_1
4   60_0      12657_20240226_60_0_2
                ...
65   60_0     12651_20240229_60_0_7
66   60_0     12651_20240229_60_0_4
67   60_0     12651_20240229_60_0_5
68   60_0     12651_20240229_60_0_2
69   60_0     12651_20240229_60_0_3
```

Informações de paragens de cada viagem: *Stop_times - Trips*

Para efetuar o planeamento da afetação de veículos, é essencial o reconhecimento do horário de funcionamento de cada viagem. Os dados fornecidos do formato GTFS oferece bastantes informações, mas a única variável relevante com o horário é *stop_times*. Esta variável tem duas chaves estrangeiras, o horário e a ordem na sequência de caminho. O essencial neste passo é agrupar os horários da chegada e da partida dos veículos a partir de *trip_id* para obter a sequência das paragens, e assim obter as informações de paragem inicial e paragem final.

```
df_trip_stop = stop_times[stop_times['trip_id']==trip_id]
df_trip_stop = df_trip_stop.sort_values('stop_sequence')
df_trip_stop[['stop_sequence','stop_id','arrival_time','departure_time']]
```

onde *trip_id* é o identificador único da viagem que se pretende visualizar e *stop_times* a tabela dos dados originais do ficheiro de *stop_times.txt*. Com o mesmo raciocínio, esta linha de código recebe um *trip_id* e obtém apenas as linhas que correspondem a este *trip_id* de *stop_times* e, em seguida, ordena as entradas da tabela por ordem crescente por parte de *stop_sequence*. Por fim, seleciona-se as colunas *stop_sequence* (sequência da paragem), *stop_id*, *arrival_time* (hora de chegada) e *departure_time* (hora de partida), formando uma nova tabela chamada *df_trip_stop*. Se a reordenação for bem feita, as colunas de *arrival_time* e *departure_time* devem estar ordenadas também. O código completo da função está no Anexo B.1.

Para uma das viagens de 60_0, 12649_20240229_60_0_13 por exemplo, as informações das paragens são representados a seguir.

stop_sequence	stop_id	arrival_time	departure_time
1	3807	05:35:00	05:35:00
2	3802	05:35:38	05:35:38
3	7102	05:36:28	05:36:28
4	3902	05:37:57	05:37:57
5	3906	05:39:30	05:39:30
6	10202	05:40:40	05:40:40
7	4004	05:42:04	05:42:04
8	12121	05:42:55	05:42:55
9	12108	05:44:00	05:44:00
10	9504	05:44:52	05:44:52
11	9506	05:45:44	05:45:44
12	9508	05:46:26	05:46:26
13	4114	05:47:17	05:47:17
14	4112	05:47:48	05:47:48
15	9604	05:49:00	05:49:00
16	9608	05:49:29	05:49:29
17	9614	05:50:21	05:50:21
18	9714	05:51:08	05:51:08
19	9717	05:51:50	05:51:50
20	13211	05:52:33	05:52:33
21	4208	05:53:27	05:53:27
22	4207	05:54:00	05:54:00
23	4202	05:54:43	05:54:43
24	4224	05:55:32	05:55:32
25	18404	05:56:12	05:56:12
26	18402	05:56:54	05:56:54
27	14312	05:58:24	05:58:24
28	14310	05:59:00	05:59:00
29	14308	05:59:17	05:59:17
30	14306	05:59:28	05:59:28
31	50059	05:59:40	05:59:40
32	29368	06:00:00	06:00:00

Começo e Terminal de cada viagem

À base da recolha de informações das paragens, é de imediato reconhecer as informações da partida da primeira paragem e da chegada à última paragem da viagem, e assim atingir o objetivo de agrupar os horários e as viagens de cada rota.

```
Start_time = df_trip_stop['departure_time'].values[0]
```

```
End_time = df_trip_stop['arrival_time'].values[-1]
```

A variável *Start_time* indica o tempo de partida da viagem, logo tende buscar o primeiro valor da coluna *departure_time*, e a variável *End_time* é o caso contrário onde busca o último valor da coluna *arrival_time* para tomar conhecimento do tempo de chegada à paragem final da viagem. O código completo da função está no Anexo B.1.

Então para a viagem 12649_20240229_60_0_13 da rota 60_0, o *stop_id* da paragem inicial, o *stop_id* da paragem final, o tempo da partida da viagem e o tempo do fim da viagem são dados por,

```
(3807, 29368, '05:35:00', '06:00:00')
```

isto é, a viagem começa na paragem com *stop_id* de 3807 às $5:35$ da manhã e no fim chega à paragem com *stop_id* de 29368 às 6 horas da manhã.

Nova tabela como complemento ao GTFS

Para além das informações das paragens, outras também são importantes, tais como: *service_id*, *shape_id* e *direction*. A primeira serve para distinguir as viagens de tipos de dia diferentes, a segunda tem como função a identificação da localização geográfica do alinhamento das viagens, normalmente dependente da direção e a terceira vai ser utilizado para a atribuição do veículo.

Seja *df_info_servico* uma tabela com duas colunas, *route_id* e *trips_id*, com *route_id* a rota pretendida e *trips_id* viagens associadas à rota.

```
df_info_servico['Service_id'] = df_info_servico['Trips_id'].apply(lambda x:
    trips[trips['trip_id']==x]['service_id'].values[0])
```

O código adiciona uma nova coluna chamada *Service_id* à *df_info_servico*. Esta coluna é preenchida aplicando uma função lambda à coluna *Trips_id*, que procura na tabela *trips* o valor da coluna *trip_id* igual ao valor atual *Trips_id*, e uma vez encontrada reserva o valor da coluna *service_id*. Obtendo assim uma relação de associação entre as tabelas *trips* e *df_info_servico*, e também a recolha de dados do tipo de serviços dependente do *trips_id*. O código completo da função está no Anexo B.2.

O mesmo processo é feito com *shape_id* e *direction*.

```
df_info_servico['Shape_id'] = df_info_servico['Trips_id'].apply(lambda x:
    trips[trips['trip_id']==x]['shape_id'].values[0])
df_info_servico['Direction'] = df_info_servico['Trips_id'].apply(lambda x:
    trips[trips['trip_id']==x]['direction_id'].values[0])
```

Primeiro dirige-se à tabela *trips* e associa, de linha a linha, as entradas da *df_info_servico* com as entradas de *trips*. Quando encontrado o *trip_id* coincidente, guarda na *df_info_servico* o valor da coluna *shape_id* ou *direction_id* da tabela *tirps*.

Até este passo, a tabela é constituída de seguinte forma, tomando como exemplo as primeiras linhas da rota 60_0,

```
Route_id Trips_id               Service_id              Shape_id      Direction
60_0     12649_20240229_60_0_13 Inverno_Util_20240229          60_0_ASC_shp  0
60_0     12649_20240226_60_0_13 Inverno_Util_20240226          60_0_ASC_shp  0
60_0     12651_20240229_60_0_11 Inverno_DomingoFeriado_20240229 60_0_ASC_shp  0
60_0     12657_20240226_60_0_1  Inverno_Util_20240226          60_0_DESC_shp 1
60_0     12657_20240226_60_0_2  Inverno_Util_20240226          60_0_ASC_shp  0
60_0     12657_20240226_60_0_3  Inverno_Util_20240226          60_0_DESC_shp 1
60_0     12649_20240226_60_0_6  Inverno_Util_20240226          60_0_DESC_shp 1
60_0     12649_20240226_60_0_7  Inverno_Util_20240226          60_0_ASC_shp  0
60_0     12649_20240226_60_0_4  Inverno_Util_20240226          60_0_DESC_shp 1
```

Para completar a tabela, adicionam-se as informações do começo e terminal de cada viagem. Tal como exemplificado, uma entrada de viagem tem como informação de começo e terminal correspondente de seguinte forma,

```
(3807, 29368, '05:35:00', '06:00:00')
```

Então as novas colunas adicionadas à tabela seriam *Stop_id_start*, *Stop_id_end*, *Start_time* e *End_time*. O atributo *Stop_id_start* corresponde à identificação da paragem do início da viagem, tal como *Stop_id_end* corresponde à identificação da paragem do final da viagem, e também os horários da partida e chegada, *Start_time* e *End_time*, respetivamente. Para a fácil leitura e compreensão das paragens, teve a necessidade de ir buscar à tabela *stop* o atributo *stop_name* associado ao *stop_id*.

```
df_info_servico['Stop_id_start'] = df_info_servico['Trips_id'].apply(lambda x:
    stop_times[stop_times['trip_id']==x]['stop_id'].values[0])
df_info_servico['Stop_id_start'] = df_info_servico['Stop_id_start'].apply(
lambda x: stops[stops['stop_id']==x]['stop_name'].values[0])
df_info_servico['Stop_id_end'] = df_info_servico['Trips_id'].apply(lambda x:
    stop_times[stop_times['trip_id']==x]['stop_id'].values[-1])
df_info_servico['Stop_id_end'] = df_info_servico['Stop_id_end'].apply(lambda
x: stops[stops['stop_id']==x]['stop_name'].values[0])
```

Para a fácil manutenção no programa do planeamento da afetação, é importante recorrer à gestão de variáveis de tempo nas tabelas e à ordenação das entradas de acordo com o horário. Esta utilidade é precisa devido a certas exceções, como por exemplo, para o caso da rota 170_0, de Praça de Chile a Fetais. Esta rota tem serviços desde das cinco horas da manhã até à uma hora do dia a seguir. Pelos ficheiros de GTFS, os serviços realizados à meia-noite, quando se viaje das 23 horas ao dia seguinte, têm o tempo de chegada definido como 24 horas, e não 00 como viagens que partem às zero horas.

```
Route_id Trips_id               Service_id ... Start_time End_time ...
170_0    3910_20240229_170_0_3  Inverno... ... 23:26      24:07    ...
170_0    3866_20240229_170_0_18 Inverno... ... 23:30      24:07    ...
170_0    3822_20240226_170_0_21 Inverno... ... 00:10      00:51    ...
```

Se ordenar a tabela através dos horários pela ordem crescente, as viagens que realizam às 00 horas passariam a estar no primeiro lugar da tabela e ao atribuir a veículos, não seriam considerados viagens subsequentes das viagens das 23 horas. Portanto, para efetuar a ordenação de forma adequada e intuitiva, é crucial identificar corretamente os serviços em causa. Não basta apenas transformar as horas que começam às 24 para 00, pois assim quando se executa o comando de ordenação computacionalmente, os serviços iriam ficar na mesma ao primeiro lugar e não a seguir aos serviços que terminam às 24 horas. Então, o que se pretende fazer em primeiro lugar será determinar o primeiro serviço no dia.

Através de várias verificações, foi descoberto que existem três casos de serviços: a rede da madrugada (ex.: *64_0*) - normalmente funciona da meia-noite às cinco da manhã do dia a seguir, os serviços de horário estendido (ex.: *177_0*) - tem como característica o longo funcionamento do serviço, das seis horas do dia até à uma da madrugada e os serviços ocasionais (ex.: *112_1*) - igual ao tipo anterior, mas têm um grande intervalo entre as viagens de manhã e de tarde, são os serviços acrescentados na hora da ponta para suportar o grande movimento de passageiros.

O critério utilizado para identificar o primeiro serviço a realizar no dia foi o seguinte (B.3): ordenar a lista de horários de partida, definir um parâmetro do tipo intervalo de tempo, o intervalo entre os horários dos serviços consecutivos caso se excede desse parâmetro, então estes serviços poderão ser o primeiro serviço e o último serviço do dia. Por exemplo, se a diferença horária entre duas viagens seguintes, *1* e *2*, for maior que três horas, então o serviço *1* seria o último serviço do dia e o serviço *2* seria o primeiro de todos. Para a rota 717 com *route_id* de 170_0, o primeiro serviço do dia é *3800_20240229_170_0_2* que parte às $4:55$ da manhã e o último é *3897_20240229_170_0_1* que acaba à $1:11$ do dia a seguir.

Para melhorar a clareza e eficiência, reformula-se a forma como as horas estão representadas. Em vez de reiniciar o ciclo de horas após a meia-noite, sugere-se continuar a contagem para além das 24 horas. Em vez de representar uma viagem às $00:30$ como início de um novo dia, poder-se-ia designar esse horário como $24:30$. Esta prática simplifica o tratamento de horários contínuos, evitando ambiguidades e facilitando o cálculo de intervalos de tempo e a otimização da afetação de veículos.

São retirados todos os dados necessários para a composição das tabelas para cada rota definida pelo *Carris*, o passo seguinte é então exportar as tabelas para futuro trabalho. Por exemplo, para a rota *60_0*, o ficheiro nomeado de *rota_servico_60_0.csv* terá as primeiras linhas de seguinte forma,

Trips_id	Service_id	Stop_id_start	Stop_id_end	Start_time	End_time	Shape_id	Direc.
12649...	Inverno...	Linda-a-Velha	Cais Sodré	00:00	00:25	60_0_...	1
12650...	Inverno...	Linda-a-Velha	Cais Sodré	00:00	00:25	60_0_...	1
12649...	Inverno...	Linda-a-Velha	Cais Sodré	00:00	00:25	60_0_...	1
12651...	Inverno...	Linda-a-Velha	Cais Sodré	00:00	00:25	60_0_...	1
12651...	Inverno...	Cais Sodré	Linda-a-Velha	00:30	00:55	60_0_...	0
12659...	Inverno...	Linda-a-Velha	Cais Sodré	00:30	00:55	60_0_...	1

```
12658... Inverno... Linda-a-Velha Cais Sodré    00:30      00:55    60_0_... 1
12650... Inverno... Cais Sodré    Linda-a-Velha 00:30      00:55    60_0_... 0
12649... Inverno... Cais Sodré    Linda-a-Velha 00:30      00:55    60_0_... 0
```

Agrupamento por atributos

Após a reformulação dos ficheiros do tipo GTFS, obtiveram-se todos os dados necessários para o planeamento da atribuição de transportes públicos. Para otimizar a utilização dos dados e facilitar a análise, realiza-se uma nova organização dos ficheiros. Esta reorganização foi feita com o agrupamento dos dados de várias formas: pelo atributo *service_id*, e torna-se em quatro novos ficheiros com dados distintos; pelo atributo *direction*, e torna-se em dois novos ficheiros distintos; por estes dois atributos, tornam em oito ficheiros. Essa abordagem permite um acesso mais eficiente e direciona às informações críticas, que simplifica o processo do planeamento e a alocação dos veículos de transporte público. Na secção 2.2, após a resolução prática do planeamento, conclui-se que apenas os ficheiros agrupados de *service_id* são aplicáveis, pois o planeamento deverá ser feito para viagens do mesmo tipo de dia.

No ficheiro *rota_servico_60_0_Inverno_Util_20240229.csv*, as primeiras linhas seriam as seguintes,

```
Route_id Trips_id Service_id            ... Start_time End_time Shape_id Direc.
60_0     12649... Inverno_Util_20240229 ... 00:00      00:25    60_0_... 1
60_0     12657... Inverno_Util_20240229 ... 00:30      00:55    60_0_... 1
60_0     12649... Inverno_Util_20240229 ... 00:30      00:55    60_0_... 0
60_0     12657... Inverno_Util_20240229 ... 01:00      01:25    60_0_... 0
60_0     12657... Inverno_Util_20240229 ... 01:30      01:55    60_0_... 1
```

A transformação horária das horas das viagens é realizada através da função *to* (Anexo B.4), que converte o tempo no formato $HH:MM:SS$ para um valor numérico representando o número total de horas como uma fração decimal. A função foi projetada para lidar com dados temporais em formato de string e converter essas strings em números inteiros para poder representar graficamente.

Anexo B

Códigos e Resultados de Implementação

B.1 Funções de agrupamento dos atributos

```
def route_trip(self):
    return self.trips[
        self.trips['route_id']==self.rota
        ]['trip_id'].reset_index(drop=True)

def trip_stop(self,trip_idx,trips_input):
    trip_id = trips_input.values[trip_idx]
    df_trip_stop = self.stop_times[self.stop_times['trip_id']==trip_id]
    df_trip_stop = df_trip_stop.sort_values('stop_sequence')
    return df_trip_stop[['stop_sequence','stop_id','arrival_time',
                         'departure_time']]

def arrival_depart_trip(self,trip_idx,trips_input):
    arrival = self.trip_stop(trip_idx,trips_input)['arrival_time'].values[-1]
    departure = self.trip_stop(trip_idx,trips_input)['departure_time'].values[0]
    stop_id_sequence = self.trip_stop(trip_idx,trips_input)['stop_id'].values
    return stop_id_sequence[0],stop_id_sequence[-1],departure,arrival
```

B.2 Extração da tabela dos dados

```
def df_info(self):
    df = pd.DataFrame()
    try:
        service_idx = int(self.service_id)
```

```
    service_id = self.calendar['service_id'][service_idx]
except:
    service_id = self.service_id

trips_da_rota = self.trips[
    (self.trips['service_id'] == service_id) &
    (self.trips['route_id'] == self.rota)]['trip_id'].values

df['Route_id'] = [self.rota] * len(trips_da_rota)
df['Trips_id'] = trips_da_rota
df['Service_id'] = [service_id] * len(trips_da_rota)

l1 = []
l2 = []
l3 = []
l4 = []

for i in range(len(df['Trips_id'])):
    v1, v2, v3, v4 = self.arrival_depart_trip(i, df['Trips_id'])
    l1.append(v1)
    l2.append(v2)
    l3.append(v3)
    l4.append(v4)

df['Stop_id_start'] = l1
df['Stop_id_start'] = df['Stop_id_start'].apply(
    lambda x: self.stops[
        self.stops['stop_id'] == x
        ]['stop_name'].values[0])

df['Stop_id_end'] = l2
df['Stop_id_end'] = df['Stop_id_end'].apply(
    lambda x: self.stops[
        self.stops['stop_id'] == x
        ]['stop_name'].values[0])

df['Start_time'] = self.inicio_fim_dia(l3)
df['End_time'] = self.inicio_fim_dia(l4)

df = df.sort_values(by='Start_time')
```

```
    df['Shape_id'] = df['Trips_id'].apply(lambda x: self.trips[
        self.trips['trip_id'] == x]['shape_id'].values[0])
    df['Direction'] = df['Trips_id'].apply(lambda x: self.trips[
        self.trips['trip_id'] == x]['direction_id'].values[0])

    return df.reset_index(drop=True)
```

B.3 Reordenação da tabela

```
def inicio_fim_dia(self,list_hora):
    list_aux = list_hora * 1
    idx_list = np.arange(len(list_aux))
    df_hora = pd.DataFrame({'Hora':list_aux,'Idx':idx_list})
    df_hora = df_hora.sort_values('Hora')
    df_hora = df_hora.reset_index(drop=True)

    if any(i.startswith('23') for i in list_aux) and \
       any(i.startswith('00') for i in list_aux):
        for i in range(len(list_aux)-1):
            if int(df_hora['Hora'].values[i][:2]) < \
               int(df_hora['Hora'].values[i+1][:2])-3:
                idx_aux = i+1
                idx_sec_day = df_hora['Idx'].values[:idx_aux]
                for j in idx_sec_day:
                    list_aux[j] = list_aux[j].replace(
                        list_aux[j][:2],str(int(list_aux[j][:2])+24),1
                    )
                return list_aux
    else:
        return list_hora
```

B.4 *to_float(str)*

```
def to_float(time):
    if isinstance(time, str):
        return int(time[:2]) + round(int(time[3:5]) * (1/60), 4) + \
               round(int(time[6:8]) * (1/3600), 4)
    else:
        print('Valor não permitido.')
```

B.5 Funções Auxiliares

B.5.1 *viagem_circular(df_circular, trips, direction)*

```
def viagem_circular(df_circular, trips, direction):
    df_circular.loc[
        df_circular['Trips_id'] == trips, 'Direction'
        ] = int(bin(direction + 1)[-1])
    return df_circular
```

B.5.2 *sub_hour((hour, time)*

```
def sub_hour(hour, time):
    return time - hour
```

B.5.3 *add_minute(minute, time)*

```
def add_minute(minute, time):
    return time + round(minute * (1 / 60), 4)
```

B.6 *atrb_direto(df,init_viagem,init_direction)*

```
    def atrb_direto(self, df, init_viagem, init_direction):
        init = 1
        dic_bus = {}
        if df.empty:
            return dic_bus

        df_aux = df.copy()

        if str(init_direction):
            dic_bus[init] = [(
                        df_aux[
                            df_aux['Direction'] == int(init_direction)
                            ].iloc[int(init_viagem)]['Trips_id'],
                        int(init_direction)
                        )]
        else:
            dic_bus[init] = [(
                        df_aux['Trips_id'].values[int(init_viagem)],
                        df_aux['Direction'].values[int(init_viagem)]
                        )]
```

```
init_hour = np.floor(df_aux['Start_time'].iloc[0])
df_aux['Start_time'] = df_aux['Start_time'].apply(
            lambda x: self.sub_hour(init_hour, x)
            )
df_aux['End_time'] = df_aux['End_time'].apply(
            lambda x: self.sub_hour(init_hour, x)
            )
direction_list = df_aux['Direction'] * 1

def atrb_bus_j(df_, dic_bus_j):
    for i in df_.values:
        trip_id = i[1]
        start_time = i[5]
        direction = i[-1]
        df_ = self.viagem_circular(
          df_, trip_id, direction
          ) if i[3] == i[4] else df_

        ultima_viagem = dic_bus_j[-1][0]
        depois_pausa = self.add_minute(
          self.pausa,
          df_[df_['Trips_id'] == ultima_viagem]['End_time'].values[0]
          )
        if depois_pausa <= start_time and \
           df_[df_['Trips_id']==trip_id]['Direction'].values[0] != \
           dic_bus_j[-1][1]:
            dic_bus_j.append((trip_id, direction))

    df_['Direction'] = direction_list

    return dic_bus_j

dic_bus[init] = [(df_aux['Trips_id'].values[0],
          df_aux['Direction'].values[0])]
dic_bus[init] = atrb_bus_j(df_aux, dic_bus[init])

while bool(list(df_aux['Route_id'].values)):
    df_aux.reset_index(drop=True)
    dic_bus[init] = [(df_aux['Trips_id'].values[0],
```

```
                df_aux['Direction'].values[0])]
        dic_bus[init] = atrb_bus_j(df_aux, dic_bus[init])
        for i in dic_bus[init]:
            df_aux = df_aux.drop(df_aux[df_aux['Trips_id'] == i[0]].index)
        init += 1

    return dic_bus
```

B.7 Solução da atribuição sequencial da rota 60_0

B.7.1 Veículo '1' - tempo pausa de 5 minutos

Trip_id	Start_time	End_time	Direction
12649_20240229_60_0_2	00:00:00	00:25:00	1
12649_20240229_60_0_3	00:30:00	00:55:00	0
12649_20240229_60_0_4	01:00:00	01:25:00	1
12649_20240229_60_0_5	01:30:00	01:55:00	0
12649_20240229_60_0_6	02:00:00	02:25:00	1
12649_20240229_60_0_7	02:30:00	02:55:00	0
12649_20240229_60_0_8	03:00:00	03:25:00	1
12649_20240229_60_0_9	03:30:00	03:55:00	0
12649_20240229_60_0_10	04:00:00	04:25:00	1
12649_20240229_60_0_11	04:30:00	04:55:00	0
12649_20240229_60_0_12	05:00:00	05:25:00	1
12649_20240229_60_0_13	05:35:00	06:00:00	0

B.7.2 Veículo '1' - tempo pausa de 10 minutos

Trip_id	Start_time	End_time	Direction
12649_20240229_60_0_2	00:00:00	00:25:00	1
12657_20240229_60_0_2	01:00:00	01:25:00	0
12649_20240229_60_0_6	02:00:00	02:25:00	1
12649_20240229_60_0_9	03:30:00	03:55:00	0
12671_20240229_60_0_8	04:30:00	04:55:00	1
12649_20240229_60_0_13	05:35:00	06:00:00	0

B.7.3 Veículo '2' - tempo pausa de 5 minutos

Trip_id	Start_time	End_time	Direction
12657_20240229_60_0_1	00:30:00	00:55:00	1
12657_20240229_60_0_2	01:00:00	01:25:00	0

```
12657_20240229_60_0_3   01:30:00    01:55:00    1
12671_20240229_60_0_8   04:30:00    04:55:00    1
12671_20240229_60_0_9   05:00:00    05:25:00    0
```

B.8 Atribuição pelas permutações das ordens

B.8.1 Número de Veículos Iniciais

```
def n_min_afetacao(df_,pausa):
  n_bus = 1
  dic_bus_init = {}
  dir_init = df_.iloc[0]['Direction']
  dic_bus_init[1] = [(df_['Trips_id'].iloc[0],dir_init)]

  ultima_viagem_1 = dic_bus_init[1][0][0]
  depois_pausa_1 = add_time(
      pausa,
      df_[df_['Trips_id']==ultima_viagem_1]['End_time'].values[0]
      )

  if df_['Stop_id_start'].values[0] != df_['Stop_id_end'].values[0]:
    for i in df_.values:
      if i[5] >= depois_pausa_1 and i[-1]!=dir_init:
        segunda_viagem_1 = i[1]
        hora_segunda_viagem = i[5]
        indice = df_[df_['Trips_id'] == segunda_viagem_1].index[0]
        indice2 = df_[df_['Start_time'] == hora_segunda_viagem].index[-1]
        viagens_anteriores = df_.iloc[1:indice2]
        break
      else:
        viagens_anteriores = df_.iloc[1:
                                      df_[df_['Trips_id'] == i[1]].index[0]]
        continue

    while not viagens_anteriores.empty:
      n_bus += 1

      dic_bus_init[n_bus] = [(viagens_anteriores['Trips_id'].iloc[0],
                              viagens_anteriores['Direction'].iloc[0])]
      viagens_anteriores = viagens_anteriores.drop(
          viagens_anteriores[
```

```
                viagens_anteriores['Trips_id'] == dic_bus_init[n_bus][0][0]
                ].index[0]
            )

        for i in viagens_anteriores.values:
          ultima_viagem_2 = dic_bus_init[n_bus][0][0]
          depois_pausa_2 = add_time(
              pausa,
              df_[df_['Trips_id']==ultima_viagem_2]['End_time'].values[0]
              )

          if i[5] >= add_time(pausa,depois_pausa_2) and \
             i[-1]!=dic_bus_init[n_bus][-1][1]:
            viagens_anteriores = viagens_anteriores.drop(
                viagens_anteriores[
                    viagens_anteriores['Trips_id'] == i[1]
                    ].index[0]
                )
          else:
            continue

    else:
      for i in df_.values[1:]:
        if i[5] >= depois_pausa_1:
          break
        n_bus += 1
        dic_bus_init[n_bus] = [(i[1],i[-1])]

    return n_bus,dic_bus_init
```

B.8.2 Atribuição usando ordem dos veículos

```
def atrb_perm_one(self, arr_perm, dic_init):
    df_copy = self.df.copy()

    if df_copy.empty:
        return {}

    dic_solucao = {}
    for i in np.arange(1, len(dic_init) + 1):
        if i in arr_perm:
```

```
        dic_solucao[i] = [dic_init[i][0]]

list_trip = []

list_init = [dic_solucao[k][0][0] for k in dic_solucao]
list_trip += list_init

init_hour = np.floor(df_copy['Start_time'][0])

df_copy['Start_time'] = df_copy['Start_time'].apply(
    lambda x: self.sub_hour(init_hour, x))
df_copy['End_time'] = df_copy['End_time'].apply(
    lambda x: self.sub_hour(init_hour, x))

direction_list = df_copy['Direction'] * 1

def atrb_bus_j(df_copy, dic_bus_j):
    list_trip_j = []
    for i in df_copy.values:
        trip_id = i[1]
        start_time = i[5]
        direction = i[-1]

        if trip_id in list_init:
            continue

        df_copy = self.viagem_circular(
            df_copy, trip_id, direction
            ) if i[3] == i[4] else df_copy

        ultima_viagem = dic_bus_j[-1][0]
        depois_pausa = self.add_minute(
            self.pausa,
            df_copy[
                df_copy['Trips_id'] == ultima_viagem
                ]['End_time'].values[0]
            )

        if depois_pausa <= start_time and df_copy[
                df_copy['Trips_id'] == trip_id
```

```
                ]['Direction'].values[0] != dic_bus_j[-1][1]:
            dic_bus_j.append((trip_id, direction))
            list_trip_j.append(trip_id)

        df_copy['Direction'] = direction_list

    return dic_bus_j,list_trip_j

init = 0

while init <= len(arr_perm)-1:
    df_copy.reset_index(drop=True)
    dic_solucao[arr_perm[init]],list_trip_j = atrb_bus_j(
        df_copy, dic_solucao[arr_perm[init]])
    list_trip += list_trip_j

    for i in dic_solucao[arr_perm[init]]:
        df_copy = df_copy.drop(df_copy[df_copy['Trips_id'] == i[0]].index)

    init += 1

return dic_solucao,list_trip
```

B.8.3 Atribuição com todas as permutações de ordens

```
def atrb_perm_all_comp(self,n_bus_init):
    solucao_otima = {}
    df_copy = self.df.copy()

    if df_copy['Trips_id'].values[0] == '0':
        return {}

    n_bus,dic_bus_init = n_min_afetacao(df_copy,self.pausa)

    arr_perm = list(itertools.permutations(np.arange(1,n_bus+1)))

    otima_length = 999
    n_bus = 999
    count = 0

    for i in arr_perm:
```

```
            count += 1
            print(i)
            df_copy2 = self.df.copy()
            solucao,list_trip_used = self.atrb_perm_one(i,dic_bus_init)

            for b in range(1,len(dic_bus_init)+1):
                list_trip_used.append(dic_bus_init[b][0][0])

            viagem_restante = len([
                a for a in df_copy2['Trips_id'].values if a not in list_trip_used ]
                )
            df_trip_unused = df_copy2[
                df_copy2['Trips_id'].isin(list_trip_used)==False
                ]
            df_trip_unused.reset_index(drop=True)

            dic_bus_new = self.atrb_direto(df_trip_unused,0, '')

            if viagem_restante < otima_length and len(dic_bus_new) <= n_bus:
                otima_length = viagem_restante * 1
                solucao_otima = solucao.copy()
                n_bus = len(dic_bus_new) * 1
                dic_bus = dic_bus_new.copy()
                ordem = i


    return solucao_otima,ordem,otima_length,dic_bus
```

B.8.4 Atribuição com todas as permutações e número completo de veículos

```
def atrb_perm_all_comp(self,n_bus_init,solucao_igual_max=0,perm=[],randomly=False):
    solucao_otima = {}
    df_copy = self.df.copy()

    if df_copy['Trips_id'].values[0] == '0':
        return {}

    dic_bus_init = self.atrb_direto(self.df,0,'')

    if n_bus_init:
        for i in np.arange(1,len(dic_bus_init)+1):
```

```
        if i < n_bus_init+1:
            dic_bus_init[i] = [dic_bus_init[i][0]]
        else:
            del dic_bus_init[i]
else:
    for i in dic_bus_init.keys():
        dic_bus_init[i] = [dic_bus_init[i][0]]
    n_bus_init = len(dic_bus_init)


if not perm:
    arr_perm = list(itertools.permutations(np.arange(1,n_bus_init+1)))
else:
    arr_perm = list(itertools.permutations(perm))

if randomly:
    arr_perm = sorted(arr_perm, key = lambda x: random.random())
else:
    arr_perm = arr_perm

otima_length = 999
n_bus_falta = 999
count = 0
anterior_length = 0
n_bus_anterior = 0
solucao_igual = 0

for i in arr_perm:
    count += 1
    print(i)
    df_copy2 = self.df.copy()
    solucao,list_trip_used = self.atrb_perm_one(i,dic_bus_init)

    for b in range(1,len(dic_bus_init)+1):
        list_trip_used.append(dic_bus_init[b][0][0])

    viagem_restante = len([
        a for a in df_copy2['Trips_id'].values if a not in list_trip_used ]
        )
    df_trip_unused = df_copy2[
```

```
            df_copy2['Trips_id'].isin(list_trip_used)==False
            ]
        df_trip_unused.reset_index(drop=True)

        dic_bus_new = self.atrb_direto(df_trip_unused,0, '')

        if viagem_restante == 0 and len(dic_bus_new) == 0:
            return solucao,i,viagem_restante,dic_bus_new
        elif viagem_restante < otima_length and len(dic_bus_new) <= n_bus_falta:
            otima_length = viagem_restante * 1
            solucao_otima = solucao.copy()
            n_bus_falta = len(dic_bus_new) * 1
            dic_bus = dic_bus_new.copy()
            ordem = i
        elif viagem_restante == anterior_length and
           len(dic_bus_new) == n_bus_anterior:
            solucao_igual += 1
            if solucao_igual == solucao_igual_max-1:
                return solucao,i,viagem_restante,dic_bus_new
        else:
            solucao_igual = 0

        anterior_length = viagem_restante
        n_bus_anterior = len(dic_bus_new)

    return solucao_otima,ordem,otima_length,dic_bus
```

B.9 *Simulated Annealing*

```
def SimulatedAnnealing(self,numPontos,temperaturaInicial,
                       temperaturaMinima,fatorReducaoTemperatura,dic_init):
    if isinstance(numPontos,int):
        pontoInicial = np.random.rand(numPontos)
        pontoAtual = pontoInicial
        ordemAtual = np.argsort(pontoAtual)+1
    elif isinstance(numPontos,list):
        ordemAtual = np.array(numPontos)

    else:
        raise ValueError("numPontos deve ser um inteiro ou uma lista de inteiros.")
```

```
temperatura = temperaturaInicial

if numPontos < len(dic_init):
    for i in np.arange(1,len(dic_init)+1):
        if i < numPontos+1:
            dic_init[i] = [dic_init[i][0]]
        else:
            del dic_init[i]
else:
    print('Quantidade de veículos impossível.')
    return

count = 0

while temperatura > temperaturaMinima:
    count += 1
    novoPonto = np.copy(pontoAtual)
    indiceAleatorio = np.random.randint(numPontos)
    novoPonto[indiceAleatorio] += np.random.randn() / 4
    ordemNova = np.argsort(novoPonto)+1

    while np.array_equal(ordemAtual, ordemNova):
        novoPonto[indiceAleatorio] += np.random.randn() / 4
        ordemNova = np.argsort(novoPonto)+1

    delta = self.funcaoObjetivo(ordemNova,dic_init)[0] -
            self.funcaoObjetivo(ordemAtual,dic_init)[0]

    if delta <= 0 or np.exp(-delta / temperatura) >= np.random.rand():
        pontoAtual = novoPonto
        ordemAtual = ordemNova

    print(ordemAtual)
    print(self.funcaoObjetivo(ordemAtual,dic_init)[0])

    temperatura *= fatorReducaoTemperatura

ordemFinal = np.argsort(pontoAtual)+1
print('Ordem final:', ordemFinal)
```

```
    print('Função objetivo na ordem final:', self.funcaoObjetivo(ordemFinal,
                                                                  dic_init)[0])

    return ordemFinal,self.funcaoObjetivo(ordemFinal,dic_init)[2]
```

B.9.1 Função objetivo

```
def funcaoObjetivo(self,ordem,dic_init):
    viagem_restante = []
    solucao,list_trip_used = self.atrb_perm_one(ordem,dic_init)
    for b in range(1,len(dic_init)+1):
        list_trip_used.append(dic_init[b][0][0])

    for a in self.df['Trips_id'].values:
        if a not in list_trip_used:
            viagem_restante.append((a,self.df[
                self.df['Trips_id']==a
                ]['Direction'].values[0]))

    return len(viagem_restante),solucao,viagem_restante
```

B.10 Representação Gráfica

```
def draw_atrb(self,list_atrb,list_unused,algor,title_name,bus_visible=''):
    plt.figure(figsize=(15,4))
    plt.axhline(y=1, color='silver', linestyle='-')
    plt.axhline(y=0, color='silver', linestyle='-')
    plt.ylim(-0.1,1.2)
    start_ = np.floor(self.df['Start_time'].values[0])-1
    end_ = np.ceil(self.df['End_time'].values[-1])+1
    tick_positions = np.arange(start_, end_, 1/6)
    tick_labels = [str(round(i)) if round(i,2).is_integer() else ''
                   for i in tick_positions]
    plt.xticks(tick_positions, tick_labels)
    plt.xlim((start_,end_))
    ax = plt.gca()
    ax.get_yaxis().set_visible(False)

    df_copy = self.df.copy()

    def draw_one(str_i,list_autocarro_i,df,clr,dir_init,visibility=1):
```

```
df_copy = df.copy()

init_hour = round(df_copy['Start_time'].values[0],0)

df_copy['Start_time'] = df_copy['Start_time'].apply(
    lambda x: self.sub_hour(init_hour,x))
df_copy['End_time'] = df_copy['End_time'].apply(
    lambda x: self.sub_hour(init_hour,x))

for i in list_autocarro_i:
    coordenadas_x = []
    coordenadas_y = []
    coorx = df_copy[df_copy['Trips_id']==i[0]]['Start_time'].values[0]
    coory = i[1]
    coordenadas_x.append(coorx+init_hour)
    coordenadas_y.append(coory)
    if i == list_autocarro_i[0]:
        if dir_init == 0:
            plt.plot(coordenadas_x[0],coordenadas_y[0],
                     color=clr,marker='s',markersize=5,label=str_i)
            plt.text(coordenadas_x[0]-0.05,coordenadas_y[0]-0.1,
                     len(list_autocarro_i),ha='center')
        elif dir_init == 1:
            plt.plot(coordenadas_x[0],coordenadas_y[0],color=clr,
                     marker='o',markersize=5,label=str_i)
            plt.text(coordenadas_x[0]-0.05,coordenadas_y[0]+0.05,
                     len(list_autocarro_i),ha='center')
    coorx = df_copy[df_copy['Trips_id']==i[0]]['End_time'].values[0]
    coory = int(bin(i[1]+1)[-1])
    coordenadas_x.append(coorx+init_hour)
    coordenadas_y.append(coory)
    if dir_init == 0:
        plt.plot([coordenadas_x[0],coordenadas_x[1]],
                 [coordenadas_y[0],coordenadas_y[1]],'--',
                 color=clr,alpha=visibility)
    elif dir_init == 1:
        plt.plot([coordenadas_x[0],coordenadas_x[1]],
                 [coordenadas_y[0],coordenadas_y[1]],'-',
                 color=clr,alpha=visibility)
    elif dir_init == 2:
```

```
            plt.plot(coordenadas_x[0],coordenadas_y[0],color=clr,
                     marker='*',markersize=5)
            plt.plot([coordenadas_x[0],coordenadas_x[1]],
                     [coordenadas_y[0],coordenadas_y[1]],':',
                     color=clr,alpha=visibility)

if bool(list_atrb):
    list_color = ['coral','cornflowerblue','limegreen','mediumpurple',
                  'hotpink','red','yellow','mediumblue','greenyellow',
                  'firebrick','orangered','olivedrab','lightskyblue',
                  'navajowhite','springgreen','lightpink']

    if bus_visible:
        for i in list_atrb:
            draw_one(str(i),list_atrb[i],df_copy,list_color[i-1],
                     list_atrb[i][0][1],0.25)

        draw_one(str(i),list_atrb[bus_visible],df_copy,
                 list_color[bus_visible-1],list_atrb[bus_visible][0][1],1)

        if bool(list_unused):
            if type(list_unused)==list:
                draw_one('0',list_unused,df_copy,'black',2,0.25)
            elif type(list_unused)==dict:
                for i in list_unused:
                    draw_one('0',list_unused,df_copy,'black',2,0.25)

    else:
        for i in list_atrb:
            draw_one(str(i),list_atrb[i],df_copy,list_color[i-1],
                     list_atrb[i][0][1])

        if bool(list_unused):
            if type(list_unused)==list:
                draw_one('0',list_unused,df_copy,'black',2)
            elif type(list_unused)==dict:
                for i in list_unused:
                    draw_one('0',list_unused[i],df_copy,'black',2)

        plt.title(str(title_name))
```

```
            plt.legend(bbox_to_anchor=(1.04, 1), loc="upper left")

        plt.grid(alpha=0.25)
    plt.show()
```

B.11 Otimização Multi-objetivo

```
from pymoo.core.problem import ElementwiseProblem
import numpy as np
import os
import pandas as pd
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.operators.sampling.rnd import IntegerRandomSampling
from pymoo.operators.crossover.sbx import SBX
from pymoo.operators.mutation.pm import PM
from pymoo.operators.repair.rounding import RoundingRepair
from pymoo.termination import get_termination
from pymoo.optimize import minimize

class MyProblem(ElementwiseProblem):
    def __init__(self, diretorio, rota, service_id):
        ficheiro = os.path.join(diretorio, rota)
        ficheiro = os.path.join(ficheiro, 'rota_servico_' + rota + '_' + \
                                + service_id + '.csv')

        x = {}
        x['x1'] = np.arange(1, 21)
        x['x2'] = np.arange(0, 20)
        x['x3'] = np.arange(0, 10)
        x['x4'] = np.arange(0, 2)
        x['x5'] = np.array([0])
        x.update({f'x{i}': [0] for i in range(6, 20)})

        super().__init__(n_var=19, n_obj=4, n_constr=0,
                         xl=np.array([min(value_list)
                                for value_list in x.values()]),
                         xu=np.array([max(value_list)
                                for value_list in x.values()]))

    def _evaluate(self, x, out, *args, **kwargs):
```

```
        f_values = self.funcFF(x)
        f1, f2, f3, f4 = f_values
        out["F"] = [f1, f2, f3, f4]

    def funcFF(self, y):
        minutos = y[0]
        ordem_equipas = y[1:]
        TT = self.carregar_tabela(minutos)
        equipas_tarefas = self.atribuir_tarefas(TT, ordem_equipas)
        A1 = self.encontrar_maior_diferenca(equipas_tarefas)
        A2 = self.encontrar_diferenca_tarefas(equipas_tarefas)
        A5 = len(equipas_tarefas)
        return np.array([A5, A2, A1, -minutos])

    def carregar_tabela(self, minutos):
        T = pd.read_csv(self.ficheiro)
        TA = np.array([T.iloc[:, 5], T.iloc[:, 6], T.iloc[:, -1]]).T
        TT = pd.DataFrame(TA)
        TT[0] = TT[0].astype("string")
        TT[1] = TT[1].astype("string")

        if any(int(i) == 0 for i in TT[0].values):
            zero_index = [i for i in range(len(list(TT[0].values)))
                          if list(TT[0].values)[i] < 1][0]
            start_time_list = list(TT[0].values)
            for i in range(zero_index, len(start_time_list)):
                start_time_list[i] += 24
            TT[0] = start_time_list

        descanso = round(minutos / 60, 4)
        TT.iloc[:, 1] += descanso
        TT[3] = np.arange(1, TT.shape[0] + 1).reshape(-1, 1)
        return TT

    def atribuir_tarefas(self, TT, ordem_equipas):
        equipas_tarefas = {}
        init = 1

        if TT.empty:
            return equipas_tarefas
```

```
    def atrb_bus_j(df_copy, dic_bus_j):
        for i in df_copy.values:
            start_time = i[0]
            direction = i[2]
            depois_pausa = dic_bus_j[-1][0] + dic_bus_j[-1][1]
            if depois_pausa <= start_time and direction != dic_bus_j[-1][2]:
                dic_bus_j.append(list(i))
        return dic_bus_j

    while bool(list(TT.values)):
        TT.reset_index(drop=True)
        equipas_tarefas[init] = [list(TT.iloc[int(ordem_equipas[init - 1])])]
        equipas_tarefas[init] = atrb_bus_j(TT, equipas_tarefas[init])

        for i in equipas_tarefas[init]:
            TT = TT.drop(TT[TT[0] == i[0]].index)
        init += 1

    return equipas_tarefas

def encontrar_maior_diferenca(self, equipas_tarefas):
    maior_diferenca = 0
    equipa_maior_diferenca = 0

    for i, tarefas in equipas_tarefas.items():
        diferencas = []
        if len(tarefas) <= 1:
            max_diff = 0
        else:
            for l in range(1, len(tarefas)):
                diferencas.append(
                        tarefas[l][0] - (tarefas[l - 1][0] + tarefas[l - 1][1])
                        )
            max_diff = np.max(diferencas)

        if max_diff > maior_diferenca:
            maior_diferenca = max_diff
            equipa_maior_diferenca = i
```

```
    return maior_diferenca

def encontrar_diferenca_tarefas(self, equipas_tarefas):
    num_tarefas = [len(tarefas) for tarefas in equipas_tarefas.values()]
    max_dif_tarefas = max(num_tarefas) - min(num_tarefas)
    return max_dif_tarefas

def calcular_tempo_inatividade(self, equipas_tarefas):
    tempos_inatividade = []
    for i, tarefas in equipas_tarefas.items():
        if len(tarefas) == 0 or len(tarefas) == 1:
            tempos_inatividade.append(0)
        else:
            diferencas = np.diff([tarefa[0] for tarefa in tarefas])
            tempo_inatividade = sum(diferencas) - \
                                - sum([tarefa[1] for tarefa in tarefas])
        tempos_inatividade.append(tempo_inatividade)
    return tempos_inatividade
```

Bibliografia

[1] Tecmic. Disponível em https://www.tecmic.com/. Visitado em 20 de janeiro de 2024.

[2] Rob Story. *Folium: Python Data, GIS, and Mapping*. Disponível em: https://python-visualization.github.io/folium/latest/index.html. Acesso em: 25 Jul. 2024.

[3] *GTFS: Making public transit data universally accessible¶, General Transit Feed Specifications*. Available at: https://gtfs.org/ (Acesso em: 24 de abril de 2024).

[4] *GTFS Schedule Reference*. Available at: https://gtfs.org/schedule/reference/ (Acesso em: 24 de abril de 2024).

[5] João Rolo Santos, *Otimização da alocação de serviços de transporte*, Relatório Final do Estágio da Licenciatura em Matemática Aplicada à Tecnologia e à Empresa (2021/22)

[6] Margarida de Oliveira Costa, *Otimização da alocação de serviços de transporte e logística*, Relatório Final do Estágio da Licenciatura em Matemática Aplicada à Tecnologia e à Empresa (2022/23)

[7] João Afonso Vieira Casal, *Investigação e construção de um Sistema de Informação Geográfica*, Dissertação de Mestrado Mestrado em Engenharia de Sistemas. Universidade do Minho, Escola de Engenharia, Outubro de 2012.

[8] Ibrahim Osman and Gilbert Laporte, "Metaheuristics: A Bibliography", *Annals of Operational Research*, vol. 63, pp. 513-628, Oct. 1996. doi: 10.1007/BF02125421

[9] Abhay Sharma, Fu Zhao, John W. Sutherland, "Econological Scheduling of a Manufacturing Enterprise Operating under a Time-of-Use Electricity Tariff- Scientific Figure on ResearchGate. Disponível em: https://www.researchgate.net/figure/Simulated-annealing (Acesso em: 17 Jul 2024).

[10] Chenhui, "Algoritmo de fácil compreensão, otimização multiobjetivo, NSGA-II (com explicação do código)". Disponível em: https://www.bilibili.com/video/BV1w84y1X7hh (Acesso em: 17 Ago 2024).

[11] Julian Blank, "Multi-objective Optimization in python PYMOO. NSGA-II: Non-dominated Sorting Genetic Algorithm". Disponível em: https://pymoo.org/algorithms/moo/nsga2.html (Acesso em: 30 Jun 2024).

[12] MAHMOOD, S. F.; HUSSAIN, M. T.; MUNIR, A.; MAHMOOD, Z. *Multi-objective Optimization to Increase Nusselt Number and Reduce Friction Coefficient of Water/Carbon Nanotubes via NSGA-II using Response Surface Methodology*. Disponível em: https://www.researchgate.net/publication/339727912. Acesso em: 15 Jul. 2024.

[13] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II- in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017. Disponível em: https://ieeexplore.ieee.org/stamp. (Acesso em 30 Jun 2024).

[14] Carlos A. Coello Coello, Gary B. Lamont, David A. Van Veldhuisen (2007), "Evolutionary Algorithms for Solving Multi-Objective Problems. Springer". ISBN 978-0-387-36797-2.