



UNIVERSIDADE FEDERAL DE LAVRAS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Graduação em Sistemas de Informação / Ciência da Computação

Relatório técnico do trabalho prático

GCC262 - Grafos e suas Aplicações

Professor: Mayron César de Oliveira Moreira

Equipe:

Gustavo Ribeiro de Figueiredo - 202210846

Cesar Augusto Pires - 202210116

Lavras, 01 de julho de 2025

1. Introdução

A disciplina GCC262 - Grafos e suas Aplicações busca trazer uma visão prática da aplicação de algoritmos em grafos para o curso de Sistemas de Informação da Universidade Federal de Lavras.

O trabalho prático desenvolvido se trata do estudo e desenvolvimento de uma solução para o problema de logística descrito a seguir:

1.1 Motivação

Estudar problemas de logística é crucial para otimizar o fluxo de bens e serviços, resultando em maior eficiência e redução de custos para empresas e consumidores. A análise detalhada de processos logísticos permite identificar gargalos, melhorar o planejamento de rotas, gerenciar estoques de forma mais eficaz e implementar tecnologias que aprimoram a tomada de decisões.

A logística desempenha um papel fundamental na competitividade das empresas, influenciando diretamente a satisfação do cliente e a sustentabilidade ambiental. Ao compreender os desafios logísticos, é possível desenvolver soluções inovadoras que impulsionam o crescimento econômico e promovem um futuro mais eficiente e responsável.

1.2 Definição Formal

O problema base pode ser definido em um grafo conexo $G = (V, E)$, onde V é o conjunto de nós e E o conjunto de arestas. Os nós representam intersecções (ou esquinas) em uma região (urbana ou rural), enquanto as arestas são as vias de acesso (ruas, avenidas, etc). Um subconjunto $ER \subseteq E$ dessas arestas deve ser atendido. Seja $n = |ER|$ o número de serviços. Uma aresta $(i, j) \in E$ pode ser percorrida qualquer número de vezes com um custo de c_{ij} cada vez, e uma demanda de q_{ij} está associada a qualquer aresta $(i, j) \in ER$. O problema visa encontrar um conjunto de viagens de veículos com custo mínimo, tal que cada viagem comece e termine em um nó depósito $v_0 \in V$, cada aresta requerida seja atendida por uma única viagem, e a demanda total para qualquer veículo não exceda uma capacidade Q .

A variação estudada no trabalho prático redefine G , em particular, como um multigrafo conectado $G = (V, E, A)$, onde V é o conjunto de nós, E o conjunto de arestas e A o conjunto de arcos (vias de mão única). Serviços são requeridos para um subconjunto de nós $VR \subseteq V$, arestas $ER \subseteq E$ e arcos $AR \subseteq A$, tal que $n = |VR| + |ER| + |AR|$.

1.3 Etapas

1.3.1 Pré-processamento dos dados

A primeira etapa consiste em representar a modelagem de grafos por meio de estruturas de dados, realizar a leitura das instâncias que representam os grafos e calcular estatísticas a respeito dessas instâncias disponíveis.

1.3.2 Solução inicial

A segunda etapa consiste em desenvolver um algoritmo construtivo para o problema

estudado na fase 1. Por construtivo, entende-se como um algoritmo que inicia com uma solução vazia e ao final de suas iterações, constrói uma solução que atende a todas as restrições do problema.

Para isso, é preciso uma solução que atenda aos seguintes critérios:

- Não ultrapassar a capacidade dos veículos em cada rota;
- Cada serviço deve ser executado por exatamente 1 rota;
- Caso uma rota passe mais de uma vez por um vértice, ou uma aresta, ou um arco requerido, o valor de demanda do serviço e seu custo de serviço devem ser contados apenas 1 vez.

1.3.3 Métodos de melhoria

Na terceira etapa, pretende-se aprimorar o algoritmo construtivo da fase 2 através de um novo algoritmo construtivo ou um algoritmo de busca local. As mesmas restrições e instâncias de teste foram mantidas nessa etapa.

2. Objetivos

O principal objetivo do trabalho prático é alcançar uma solução para os problemas de logística propostos em cada instância de teste.

2.1 Objetivo principal

Cada instância representa um grafo particular, com arestas, arcos, vértices e serviços únicos. A proposta é realizar a leitura dos dados de entrada, modelar o grafo em memória e oferecer como saída uma solução viável para o problema.

2.2 Objetivos secundários

Além de apresentar uma solução ao problema, outros objetivos também podem ser mencionados, como por exemplo:

- Compreender as estruturas de dados auxiliares que ajudam a modelar o grafo em memória, como matrizes e listas de adjacências.
- Compreender algoritmos relacionados à Teoria de Grafos, como Floyd-Warshall, Busca em Largura, Busca em Profundidade, etc.
- Desenvolver adaptações nos algoritmos clássicos para resolução do TSP, como Caminho mais Curto e 2-opt.
- Aplicar programação orientada a objetos para encapsular responsabilidades e desenhar um software legível e de fácil manutenção.
- Aplicar conceitos de otimização e aprimorar a solução inicial.
- Trabalhar com a linguagem de programação C++ e utilizar de recursos avançados da linguagem.

Todos esses itens compreendem objetivos secundários que foram atingidos ao longo do desenvolvimento do trabalho.

3. Metodologia

Vamos apresentar toda a metodologia aplicada no trabalho prático que utilizamos

para alcançar a solução final.

3.1 Solução inicial

Na etapa 2, aproveitamos da implementação da leitura dos dados de entrada (fileReader.h) e da modelagem do grafo em memória (grafo.h) da etapa 1. A partir disso, implementamos a classe Solucao (solucao.h).

Nossa implementação para a etapa 2 utiliza de uma adaptação do algoritmo de Caminho mais Curto, usado em problemas como o TSP (Traveling Salesman Problem (Problema do Caixeiro Viajante)).

Partindo do vértice inicial (depósito), sempre procuramos o serviço mais próximo para ser atendido primeiro, atendemos esse serviço, nos movemos para sua posição, e novamente buscamos o serviço mais próximo da posição atual, reiniciando o ciclo até que não haja mais capacidade ou todos os serviços já tenham sido atendidos.

O problema dessa abordagem é o aspecto míope que ela traz consigo. O algoritmo sempre busca o primeiro serviço com menor custo de deslocamento, mas ele não consegue identificar serviços que, embora inicialmente estejam um pouco mais distantes, possam produzir resultados melhores posteriormente.

Uma tentativa para solucionar o problema será abordada no tópico 3.2.

O pseudocódigo para os principais métodos da etapa 2 pode ser encontrado a seguir:

Classe: Solucao

Entrada: Grafo, CapacidadeVeiculo, VerticeDeposito

Inicialização da classe: Identificar todos os serviços requeridos no Grafo

Principais métodos:

1. void encontrarRotas()
 - 1.1. enquanto houver serviços pendentes
 - 1.1.1. criar uma RotaAtual vazia
 - 1.1.2. iniciar rotaAtual com o VerticeDeposito
 - 1.1.3. definir cargaRestante = CapacidadeVeiculo
 - 1.1.4. definir localizacaoAtual = VerticeDeposito
 - 1.1.5. enquanto ainda puder atender serviços
 - 1.1.5.1. identificar serviço mais próximo para ser atendido
 - 1.1.5.2. se não houver serviço para ser atendido
 - 1.1.5.2.1. deslocar até VerticeDeposito
 - 1.1.5.2.2. adicionar o custo de deslocamento de localizacaoAtual até VerticeDeposito
 - 1.1.5.2.3. adicionar rotaAtual à lista de rotas da solução
 - 1.1.5.3. se houver serviço para ser atendido
 - 1.1.5.3.1. atender serviço
 - 1.1.5.3.2. adicionar o custo de deslocamento de localizacaoAtual até VerticeDeposito
 - 1.1.5.3.3. adicionar o custo de atendimento ao serviço
 - 1.1.5.3.4. atualizar localizacaoAtual

1.1.5.3.5. atualizar cargaRestante

2. tuple<int, int> encontrarMelhorServico(int verticeAtual, int cargaRestante)
 - 2.1. para cada serviço pendente
 - 2.1.1. identificar serviço pendente mais próximo a verticeAtual
 - 2.2. priorizar serviço em nó de origem de outro serviço em arco/aresta
 - 2.3. retornar serviço a ser atendido e o custo de deslocamento

3.2 Otimização

Na etapa 3, o objetivo era otimizar os resultados encontrados na etapa anterior.

A princípio, tentamos utilizar de uma heurística que consistia em direcionar o veículo ao depósito quando a carga restante estivesse acabando (carga restante menor que 10, 15, ou 20% da capacidade total). A ideia era evitar que o veículo se afastasse ainda mais do depósito buscando o serviço mais próximo, e tendo que fazer uma longa viagem de volta vazio, sem atender novos serviços. O resultado encontrado foi decepcionante e o custo das soluções aumentou para praticamente todas as instâncias de teste. Esse é o fardo de uma heurística. Há cenários que funcionam bem, outros nem tanto. No nosso contexto, a heurística falhou.

Ainda em busca de melhorar os resultados, tentamos aplicar o método 2-opt. Esse algoritmo parte de uma solução já existente e busca inverter segmentos procurando por soluções mais baratas. Para o nosso contexto era ótimo, uma vez que os resultados da etapa anterior estavam disponíveis. Basicamente, comparamos os custos das rotas obtidas nas etapas anteriores com os custos das novas rotas aplicando o 2-opt. Se o resultado com 2-opt for mais barato, acatamos esse resultado. Caso contrário, mantemos a rota original.

Os resultados foram animadores, e dessa vez o custo para praticamente todas as instâncias foi melhorado.

O pseudocódigo para os principais métodos da etapa 3 pode ser encontrado a seguir:

Classe: Solucao

Entrada: Grafo, CapacidadeVeiculo, VerticeDeposito

Inicialização da classe: Identificar todos os serviços requeridos no Grafo

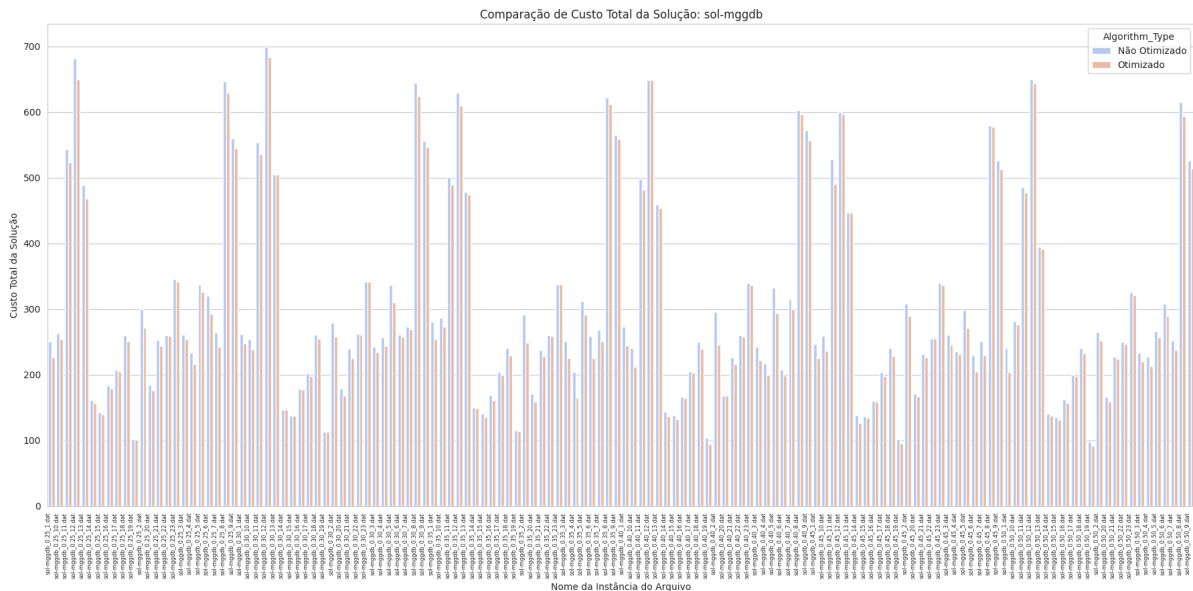
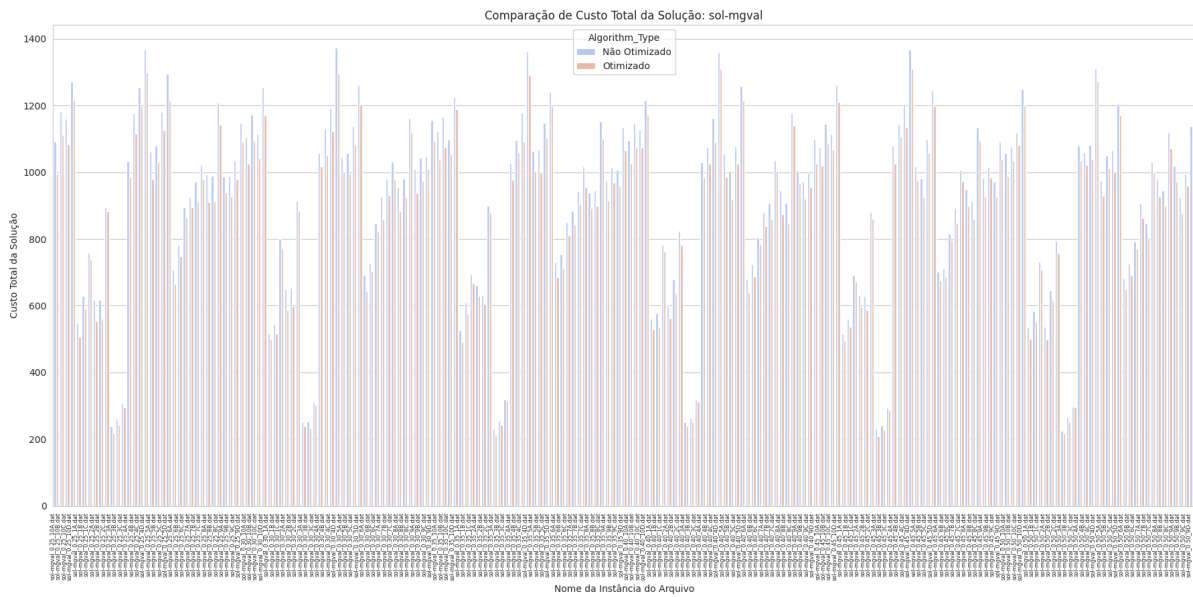
Principais métodos:

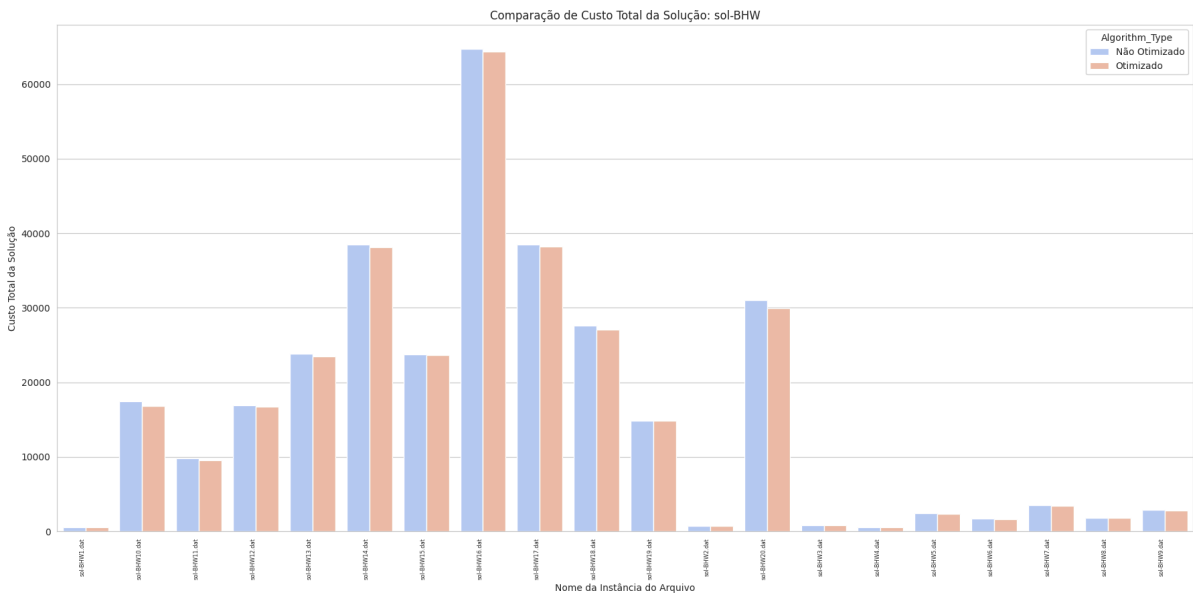
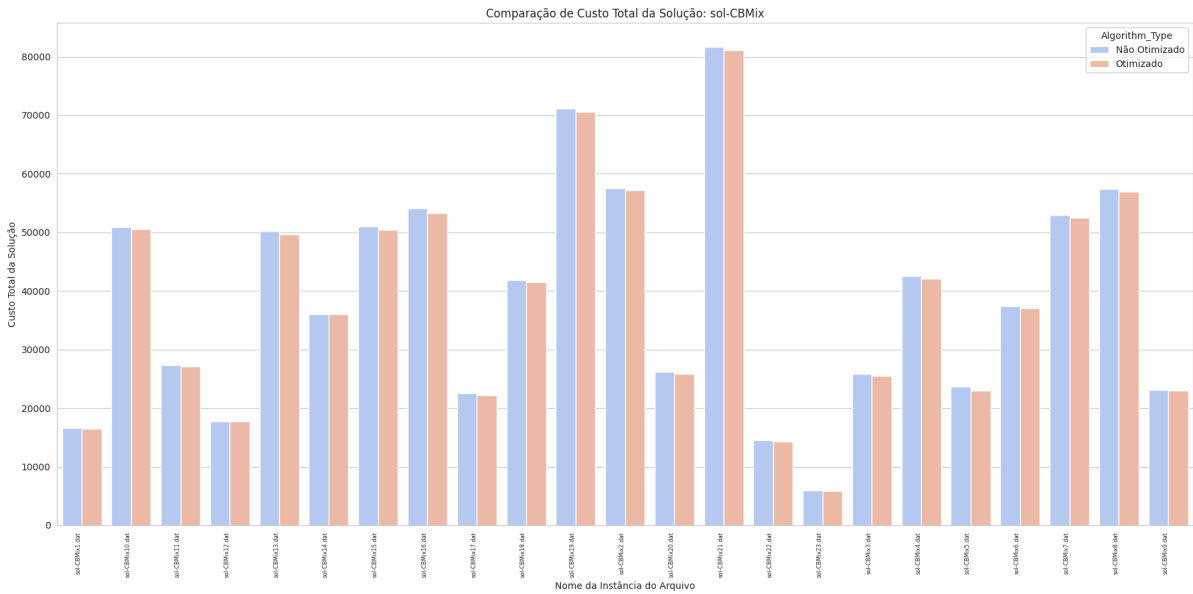
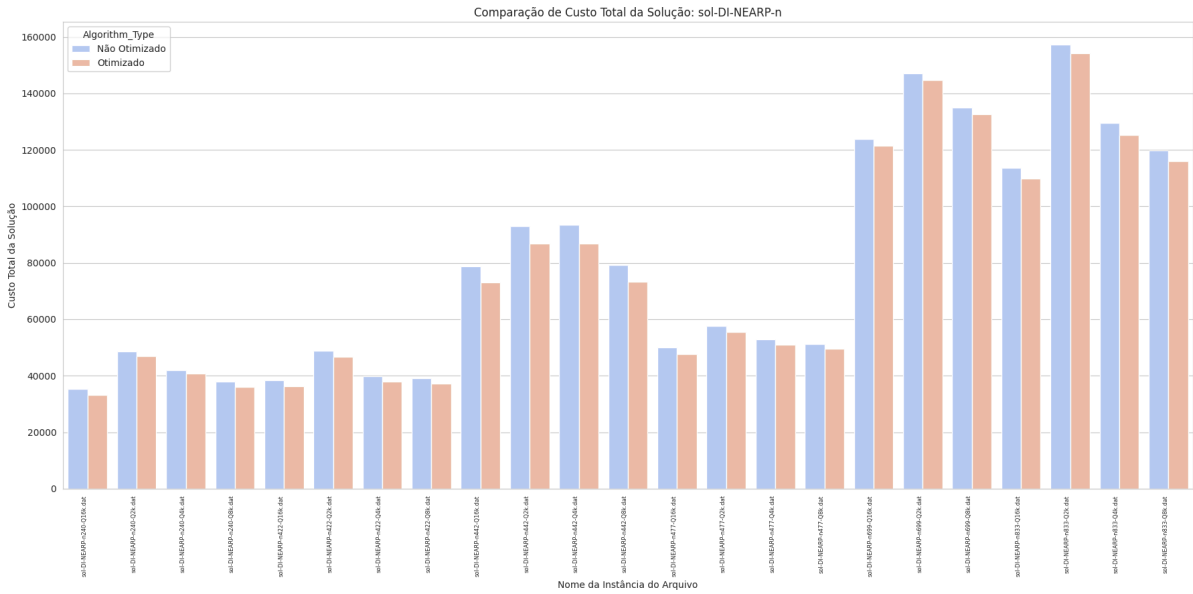
1. bool aplicar2optEmRota(Rota& rota)
 - o Entrada: Rota
 - o Saída: Verdadeiro se a rota foi melhorada, Falso caso contrário
1. Se a rota possui menos de 2 serviços, retornar falso
2. Loop
 - 2.1. inicializar melhorCusto = Custo atual da rota
 - 2.2. para i de 0 até tamanho da rota - 1
 - 2.3. para j de i+1 até tamanho da rota
 - 2.3.1. gerar uma novaSequencia de serviços invertendo o trecho entre as posições i+1 e j na sequência atual
 - 2.3.2. se o trecho invertido contém um serviço em Arco Direcionado, descartar essa sequência

- 2.3.3. calcular o novoCusto da rota com a novaSequencia
- 2.3.4. se novoCusto < melhorCusto
 - 2.3.4.1. atualizar melhorCusto
 - 2.3.4.2. atualizar novaSequencia como a melhor encontrada
- 3. retornar valor booleano indicando se houve melhoria na rota

4. Resultados

Os gráficos abaixo comparam o custo total da solução obtida sem otimização (Etapa 2) e com otimização (Etapa 3, utilizando 2-opt) para diferentes conjuntos de instâncias.





3.2 Comparação de Custo Total da Solução e Porcentagem de Melhoria por Instância

A seguir, são apresentados os resultados de melhoria obtidos pela aplicação do algoritmo 2-opt para cada tipo de instância:

	Porcentagem_Melhora (%)
sol-mgval	5.04
sol-DI-NEARP-n	4.29
sol-mggdb	4.22
sol-BHW	2.22
sol-CBMix	1.01

Os dados mostram que o algoritmo 2-opt obteve melhorias consistentes em todas as instâncias testadas, com destaque para a instância sol-mgval, que apresentou a maior redução de custo (5.04%). A menor melhoria foi observada na instância sol-CBMix (1.01%), ainda assim indicando um ganho de eficiência.

Este resultado reforça a eficácia do método 2-opt como uma heurística de otimização para o problema de roteamento de veículos com demanda e capacidade, especialmente em comparação com a solução inicial míope. A capacidade de reordenar segmentos da rota para encontrar arranjos mais eficientes de atendimento aos serviços é crucial para a redução do custo total da logística.

5. Conclusões

O desenvolvimento deste trabalho prático foi fundamental para compreender a aplicação de algoritmos em grafos na otimização de problemas de logística. A implementação de uma solução construtiva e, posteriormente, de uma heurística de otimização como o 2-opt, demonstrou claramente a importância da busca local para refinar soluções iniciais. As melhorias consistentes observadas em todas as instâncias testadas, que variaram de 1.01% a 5.04% de redução de custo, validam a eficácia do 2-opt em encontrar rotas mais eficientes e evidenciam que mesmo abordagens simples de otimização podem gerar ganhos significativos em problemas complexos como o VRP.

A experiência também ressaltou os desafios inerentes à modelagem de problemas de roteamento com serviços em diferentes tipos de elementos do grafo (nós, arestas e arcos), além das restrições de capacidade e atendimento único. A necessidade de uma representação de dados robusta e algoritmos que considerem todas essas particularidades é crucial para a viabilidade da solução. Adicionalmente, a tentativa de heurísticas menos sofisticadas que não geraram melhorias sublinhou a importância de escolher abordagens de otimização adequadas e testá-las rigorosamente.

Para futuras melhorias e aprofundamento, o grupo sugere a exploração de meta-heurísticas mais avançadas como Simulated Annealing ou Algoritmos Genéticos, que poderiam otimizar ainda mais os resultados ao explorar o espaço de soluções de forma mais

global. A concepção do problema também poderia ser expandida para incluir aspectos mais realistas, como janelas de tempo para atendimento, frotas de veículos heterogêneas, consideração de múltiplos depósitos, ou até mesmo fatores dinâmicos como condições de tráfego em tempo real. Tais aprimoramentos levariam a soluções ainda mais robustas e aplicáveis a cenários logísticos do mundo real.