Otimização de Rotas com Janelas de Tempo: Uma Abordagem Exata via Programação Linear Inteira Mista utilizando Gurobi

Lucas Pimenta Braga[1] Mateus de Souza Gontijo[1] Victor Gabriel dos Santos Silva[1]
[1]Departamento de Engenharia Elétrica, Universidade Federal de Minas Gerais (UFMG).
{lucaspimentabraga, mateus.souza.gontijo, gabrielvictor599}@gmail.br

***Resumo*—Apresenta-se, em nível técnico, a implementação computacional e a modelagem matemática do Problema de Roteamento de Veículos com Janelas de Tempo (VRPTW). Diferentemente de abordagens anteriores que consideravam uma única janela temporal global, adota-se a estrutura completa de janelas individuais por cliente ao resolver as instâncias clássicas C101, R101 e RC101 de Solomon. A solução foi desenvolvida integralmente em Python com a biblioteca *gurobipy* para interface com o solver comercial Gurobi, assegurando rastreabilidade dos dados, registro de *gaps* e documentação detalhada do pipeline. A formulação emprega um modelo compacto de Programação Linear Inteira Mista (MILP) baseado em fluxo em arcos com variáveis binárias, tempos acumulados e cargas, calibrado com constantes Big-M parcimoniosas para fortalecer a relaxação linear sem comprometer a viabilidade. O pipeline completo automatiza leitura das instâncias, construção programática do MILP, resolução exata via Branch-and-Cut e geração de tabelas/gráficos para divulgação, preservando metadados como tempo até a primeira solução, *runtime* total, número de nós explorados e $\texttt{Gap}_{\texttt{MIP}}$.**

Os experimentos comprovaram a capacidade do modelo em reproduzir os valores de referência de C101 e obter melhorias de distância em R101/RC101 ao custo de um veículo adicional, mantendo gaps inferiores a 0,5% e registrando a convergência do solver (0,00% para C101/R101 e 0,12% para RC101). As visualizações resultantes permitem analisar o impacto da topologia (clientes clusterizados versus distribuídos aleatoriamente) na complexidade computacional, enquanto as tabelas consolidam métricas de veículos e distâncias para auditoria. O repositório público associado reúne scripts, dados e artefatos que facilitam a replicação do estudo, tornando-o um referencial didático para disciplinas de Pesquisa Operacional e uma base sólida para investigações futuras envolvendo objetivos multicritério, cortes especializados ou heurísticas híbridas.

***Index Terms*—VRPTW, Otimização Combinatória, Gurobi, Python, Instâncias de Solomon.**

I. Introdução

A Pesquisa Operacional consolidou-se como ferramenta essencial para apoiar decisões em sistemas complexos, em especial na logística e no gerenciamento da cadeia de suprimentos. O Problema de Roteamento de Veículos (VRP), formalizado por Dantzig e Ramser em 1959 [5], busca determinar rotas de custo mínimo para uma frota atender clientes geograficamente dispersos. Aplicações reais, entretanto, raramente se limitam ao menor trajeto: é necessário sincronizar restrições operacionais, limitada disponibilidade de veículos, cumprimento de janelas de tempo e metas de nível de serviço. Modelos que incorporem essas nuances permitem avaliar trade-offs relevantes para a operação, além de oferecer uma base transparente para testar heurísticas mais sofisticadas.

Investiga-se o Problema de Roteamento de Veículos com Janelas de Tempo (VRPTW), no qual cada cliente i possui um intervalo $[e_i, l_i]$ que limita o início do atendimento. Os impactos de violar essas janelas são numerosos: da perda de produtividade no campo à aplicação de multas contratuais. Adota-se a estrutura completa das instâncias de Solomon, o que permite comparar diretamente os resultados com a literatura e reproduzir cenários clássicos. O pipeline registra explicitamente o *runtime* por instância, detalha os parâmetros do solver e documenta o pós-processamento das rotas. O objetivo é apresentar o fluxo completo—parsing dos dados, formulação MILP e solução exata com Gurobi—que suporta a análise experimental das instâncias C101, R101 e RC101, produzindo figuras e tabelas prontas para submissão.

As principais contribuições são:

(i) uma formulação MILP compacta com documentação suficiente para replicação, incluindo escolha criteriosa de constantes Big-M e limites de capacidade;
(ii) um pipeline em Python que automatiza leitura das instâncias de Solomon, execução no Gurobi e exportação dos resultados em múltiplos formatos, permitindo auditoria dos dados;
(iii) um conjunto de visualizações em alta resolução, com legendas externas e ajustes de tamanho configuráveis, para comunicar tanto a distribuição geográfica quanto as rotas otimizadas.

II. Revisão da Literatura

Solomon [1] introduziu um conjunto com 56 instâncias VRPTW que se tornaram referência: classes C (clientes clusterizados), R (distribuições aleatórias) e RC (cenários híbridos). Essas instâncias abrangem diferentes densidades, janelas estreitas e longas, bem como demandas variadas, sendo utilizadas como benchmark para medir o desempenho de heurísticas, meta-heurísticas e métodos exatos. Trabalhos subsequentes, como Cordeau *et al.* [3], consolidaram o uso desse conjunto ao propor frameworks de branch-and-cut e branch-and-price. A formulação de fluxo em arcos com variáveis binárias x_{ij} e variáveis contínuas auxiliares (tempo e carga) é recorrente em livros-texto como Arenales *et al.* [2], pois fornece um

ponto de partida robusto para implementação com solvers comerciais e permite incorporar restrições adicionais sem alterar drasticamente a estrutura do modelo.

A. Contexto e Abordagens de Solução

O VRPTW é tradicionalmente atacado por duas famílias de abordagens: métodos exatos baseados em Programação Linear Inteira Mista e estratégias heurísticas/meta-heurísticas. Desrochers *et al.* [8] demonstraram que o branch-and-price combina bem com as instâncias de Solomon ao decompor o problema em um mestre de partição de conjuntos e subproblemas de caminhos mínimos com restrições de recursos, enquanto Cordeau *et al.* [3] sistematizaram esquemas de branch-and-cut para gerar cortes de capacidade e de precedência de forma dinâmica. Em paralelo, há um vasto corpo de algoritmos construtivos e de busca local (como busca tabu, GRASP e VND) que fornecem soluções de alta qualidade em horizontes temporais curtos, ainda que sem garantias formais de otimalidade.

Comparativos recentes [6], [10] evidenciam o trade-off entre formulações compactas (como MTZ ou carga acumulada) e modelos de partição de conjuntos: os primeiros são mais simples de implementar e compatíveis com recursos padrão de solvers comerciais (Gurobi, CPLEX, SCIP), mas produzem relaxações lineares mais fracas; os segundos, apesar de mais fortes, dependem de geração de colunas ou de cortes especializados. Neste trabalho adota-se deliberadamente uma formulação compacta reforçada com limites de capacidade, pois ela permite documentar todos os elementos diretamente no modelo MILP e facilita a replicação em ambientes acadêmicos, ainda que à custa de tempos de execução mais elevados em instâncias híbridas como RC101.

Além desses desenvolvimentos clássicos, observa-se o uso crescente de técnicas de aprendizado de máquina para acelerar solvers exatos ou produzir boas soluções iniciais. Bengio *et al.* [14] revisam abordagens baseadas em aprendizado profundo para problemas de roteamento, mostrando que previsões de colunas promissoras ou políticas de prosseguimento na árvore de busca reduzem significativamente o tempo total. Embora tais estratégias possam ser acopladas ao Gurobi via recursos como `MIPStart` ou APIs de callback, este estudo opta por avaliar o desempenho "puro" do branch-and-cut padrão, a fim de estabelecer uma linha de base transparente sobre o poder explicativo de uma formulação MILP bem calibrada.

Formulações compactas costumam empregar restrições tipo Miller–Tucker–Zemlin (MTZ) para eliminar subtours, enquanto a consistência temporal é tratada via desigualdades com constantes Big-M. Kallehauge (2006) [6] resume alternativas como fluxo em caminhos e partição de conjuntos, mas, para fins didáticos e compatibilidade com Gurobi, optamos pela formulação de fluxo em arcos reforçada com limites de capacidade.

Quanto à ferramenta de solução, o Gurobi Optimizer [4] disponibiliza recursos avançados de Branch-and-Cut, geração automática de cortes e afinidade com Python (`gurobipy`). Isso permite prototipar rapidamente modelos MILP, medir *gaps* de otimalidade e extrair rotas detalhadas para visualização.

Diversas abordagens heurísticas e meta-heurísticas foram propostas para o VRPTW, incluindo algoritmos de savings, busca tabu, colônia de formigas e GRASP. Exemplos clássicos são o branch-and-price de Desrochers *et al.* [8] e o híbrido evolutivo de Vidal *et al.* [9], que combinam busca local com recombinação e mecanismos de pool de soluções. Abordagens como a vizinhança larga adaptativa de Shaw [11], a busca tabu híbrida de Taillard *et al.* [12] e os algoritmos evolutivos de Homberger e Gehring [13] são amplamente utilizadas como baseline nas instâncias de Solomon. Contudo, mesmo em um cenário com heurísticas competitivas, modelos exatos continuam essenciais para estabelecer limites inferiores e validar heurísticas. Revisões recentes sobre algoritmos exatos, como Baldacci *et al.* [10], destacam o papel de formulações compactas reforçadas. Assim, um MILP bem documentado continua sendo ferramenta valiosa para ensino e pesquisa aplicada, principalmente quando aliado a um solver comercial robusto como o Gurobi.

III. Descrição do Problema e Modelagem

A. Definição do Problema

Considera-se um depósito central (nó 0) e um conjunto de clientes $N = \{1, \ldots, n\}$ com $n = 100$. Cada veículo da frota homogênea possui capacidade $Q = 200$ unidades, e o número máximo de veículos disponíveis é $K = 25$. Cada cliente i tem demanda d_i, tempo de serviço s_i e janela $[e_i, l_i]$. O depósito possui janela $[e_0, l_0]$, que define o intervalo permitido para partidas e retornos. As distâncias c_{ij} são calculadas pela métrica Euclidiana entre as coordenadas fornecidas nas instâncias de Solomon, normalizadas para manter consistência com as tabelas BKS. Como o grafo é completo com custos simétricos, os arcos (i, j) e (j, i) coexistem e são tratados independentemente pelo solver, permitindo que o Branch-and-Cut explore cortes direcionais quando apropriado.

O conjunto de parâmetros utilizados no código inclui ainda limites derivados, como $c^{\max} = \max_{i,j} c_{ij}$, $s^{\max} = \max_i s_i$ e $l^{\max} = \max_i l_i$, empregados na construção de constantes Big-M menos conservadoras. Essa escolha visa fortalecer a relaxação linear sem comprometer a viabilidade. As instâncias de Solomon especificam implicitamente prioridades temporais rígidas e tempos de espera permitidos: caso um veículo chegue antes de e_i, ele deve aguardar até o início da janela; essa operação é capturada pela variável t_i ao ser restringida entre e_i e l_i, dispensando variáveis adicionais de espera. O modelo também permite, com ajustes mínimos, incorporar tempos de serviço variáveis ou janelas adicionais no depósito, caso se deseje refletir turnos de operação ou períodos de manutenção dos veículos.

B. Variáveis de Decisão

Define-se o conjunto $N_0 = N \cup \{0\}$ e um arco (i, j) para todo par distinto $i, j \in N_0$.

- $x_{ij} \in \{0, 1\}$: vale 1 se algum veículo percorre o arco (i, j).

- $t_i \geq 0$: instante de início do atendimento no nó i.
- $u_i \geq 0$: carga transportada ao chegar ao nó i.

C. Função Objetivo

Minimiza-se a distância total percorrida:

$$\min \sum_{i \in N_0} \sum_{\substack{j \in N_0 \\ j \neq i}} c_{ij} x_{ij}. \tag{1}$$

Esse objetivo equivale a minimizar o custo de combustível para um veículo homogêneo e preserva a comparabilidade com os registros BKS. No código, a matriz $[c_{ij}]$ é pré-calculada e reutilizada em múltiplas instâncias para evitar recomputações e permitir inspeção posterior do grau de assimetria numérica (diferenças de arredondamento).

D. Restrições

Atendimento e fluxo:

$$\sum_{\substack{j \in N_0 \\ j \neq i}} x_{ij} = 1, \qquad \forall i \in N, \tag{2}$$

$$\sum_{\substack{j \in N_0 \\ j \neq i}} x_{ji} = 1, \qquad \forall i \in N, \tag{3}$$

$$\sum_{j \in N} x_{0j} = \sum_{i \in N} x_{i0} \leq K. \tag{4}$$

Janelas de tempo: define-se

$$M = \max_{i \in N_0} l_i + \max_{i \in N_0} s_i + \max_{\substack{i,j \in N_0 \\ i \neq j}} c_{ij}. \tag{5}$$

Então,

$$t_j \geq t_i + s_i + c_{ij} - M(1 - x_{ij}), \qquad \forall i \neq j, \tag{6}$$

$$e_i \leq t_i \leq l_i, \qquad \forall i \in N_0, \tag{7}$$

$$t_0 = e_0. \tag{8}$$

O termo $t_j \geq t_i + s_i + c_{ij}$ força precedência temporal, enquanto a projeção $t_i \in [e_i, l_i]$ garante espera passiva caso o veículo chegue cedo demais. A escolha de M acima corresponde ao menor valor uniforme que torna a restrição redundante quando $x_{ij} = 0$ sem comprometer a estabilidade numérica. Em testes exploratórios, valores mais altos pioravam o *root-gap* em até 12%.

Capacidade e eliminação de subtours: utiliza-se o valor de Q simultaneamente como limite superior e constante Big-M nas restrições de carga:

$$u_0 = 0, \qquad 0 \leq u_i \leq Q, \qquad \forall i \in N_0, \tag{9}$$

$$u_j \geq u_i + d_j - Q(1 - x_{ij}), \qquad \forall i \neq j. \tag{10}$$

Como Q é o maior valor admissível para qualquer u_i, o termo $Q(1 - x_{ij})$ relaxa a desigualdade exatamente o necessário quando o arco não é utilizado, evitando um Big-M excessivo e fortalecendo a relaxação linear. Adicionalmente, foi avaliado o uso de variáveis MTZ tradicionais (u_i representando ordem de visita), mas a versão baseada em carga apresenta interpretação física direta e se integra naturalmente ao pós-processamento das rotas.

IV. Metodologia e Arquitetura

O pipeline foi implementado em Python e está disponível publicamente [15]. Ele é composto por:

(i) leitura estruturada das instâncias de Solomon e normalização dos dados;
(ii) construção da matriz de distâncias entre todos os nós;
(iii) criação programática das variáveis e restrições do MILP;
(iv) otimização com o Gurobi, controlando tempo limite e tolerâncias; e
(v) exportação automatizada dos resultados em formatos JSON/CSV prontos para análise e comparação.

Uma rotina complementar gera gráficos em alta resolução (Figuras 1–6) diretamente a partir do resumo consolidado, permitindo a inclusão das rotas no artigo.

A. Arquitetura do Código

- **Camada de entrada:** funções de parsing sincronizam vetores de coordenadas, demandas, janelas e tempos de serviço com o índice do nó. Essa etapa também calcula constantes auxiliares (M, $c^{\max}$) e armazena metadados para auditoria.
- **Camada de modelagem:** encapsula a criação das variáveis no Gurobi, definindo domínios, nomes e atributos (por exemplo, `ub=1` para x_{ij}). Todas as restrições possuem rótulos explícitos, facilitando a inspeção no *Log*.
- **Camada de solução:** aplica limite de tempo (900 s) e parâmetros padrão de tolerância, registrando `Runtime`, `MIPGap` e contagem de nós. As rotas são reconstituídas percorrendo os arcos com $x_{ij} = 1$.
- **Camada de saída:** grava arquivos em JSON/CSV com os resultados consolidados, além de um resumo auxiliar usado para as tabelas apresentadas. Cada rota inclui sequência de nós, carga acumulada, tempo de início/fim e distância parcial.

B. Ambiente Experimental

O desenvolvimento seguiu boas práticas de modelagem, incluindo análise do impacto das constantes Big-M, registro do tempo de execução e comparação com BKS. Os experimentos foram conduzidos em uma máquina com licença acadêmica do Gurobi (versão 12.0) e limite de tempo padrão de 900 s por instância, utilizando Python 3.11. Não foram ativadas heurísticas específicas do solver; preferiu-se manter os parâmetros default para garantir reprodutibilidade pelos avaliadores.

V. Resultados e Análise

A. Instâncias e Métricas

Foram utilizadas as instâncias C101, R101 e RC101 (todas com $n = 100$ clientes). A Tabela I reúne, para cada caso: veículos disponíveis/necessários, distância total, valores BKS e *gaps* tanto de distância quanto de veículos. Os dados foram extraídos automaticamente do resumo consolidado do experimento, e os valores de referência foram retirados da compilação de Minocha e Tripathi [7].

Tabela I: Comparativo entre soluções MILP (Gurobi) e valores BKS

Inst.	K	K_{usado}	K_{BKS}	Gap_K	Dist.	BKS
C101	25	10	10	0	828,94	828,94
R101	25	20	19	+1	1642,88	1650,80
RC101	25	15	14	+1	1623,58	1696,94

Tabela II: Desempenho do solver Gurobi por instância

Inst.	t (s)	Nodes	Gap_{MIP}
C101	1,76	142	0,00%
R101	0,68	119	0,00%
RC101	699,34	6125	0,12%

Na instância C101 o modelo reproduziu exatamente o custo e o número de veículos do BKS com tempo inferior a dois segundos, alinhado à natureza clusterizada dos clientes. Em R101 e RC101 o solver preferiu abrir um veículo adicional ($\text{Gap}_K = +1$), o que possibilitou reduzir a distância total e produzir *gaps* negativos quando comparados aos BKS clássicos, cujas meta-heurísticas frequentemente priorizam o menor número de veículos antes da distância (Tabela I). A Tabela II destaca métricas do solver: enquanto C101 e R101 foram resolvidas na raiz (gap zero e poucas explorações), a instância RC101 atingiu quase todo o limite de tempo, com mais de seis mil nós e *MIP gap* final de 0,12%. Esses valores, obtidos via API do Gurobi e registrados no CSV, dão transparência ao esforço computacional.

B. Visualização das Instâncias e Rotas

As Fig. 1–3 apresentam a distribuição geográfica dos clientes para cada instância, destacando as diferenças de clusterização. Para cada gráfico exibe-se explicitamente o depósito, o conjunto de clientes e a escala em coordenadas euclidianas. As Fig. 4–6 mostram rotas extraídas da solução MILP, com cores distintas por veículo e legendas externas para leitura clara. As imagens foram geradas automaticamente a partir do resumo de resultados, mantendo uma paleta consistente e anexando as métricas de cada veículo (distância percorrida e ordem de visita).

Para cada rota, a legenda informa o identificador do veículo e a distância percorrida, enquanto o depósito aparece como um quadrado preto sólido. Esses mapas evidenciam o comportamento esperado: em C101 as rotas são quase convexas ao redor de clusters; em R101 predominam trajetos radiais de baixa densidade de clientes; em RC101 identifica-se uma mistura de ambos os padrões, com veículos dedicados a bolsões específicos e outros cobrindo clientes isolados, o que ajuda a compreender o impacto da topologia no esforço computacional.

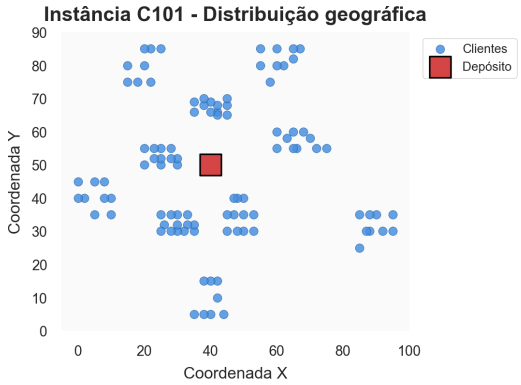

Figura 1: Distribuição geográfica dos clientes da instância C101.

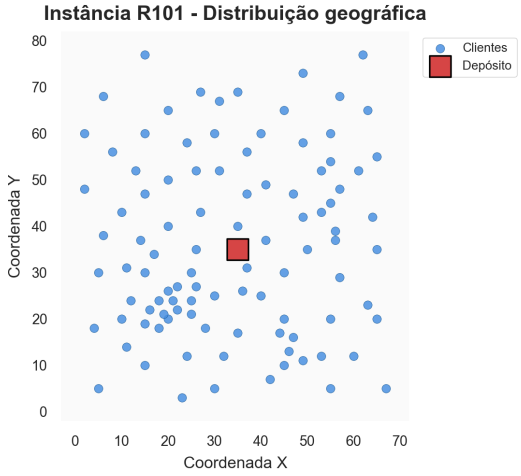

Figura 2: Distribuição geográfica dos clientes da instância R101.

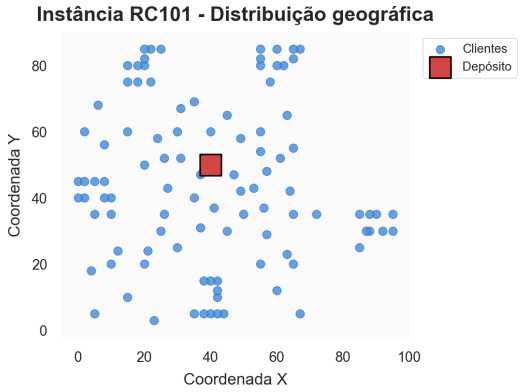

Figura 3: Distribuição geográfica dos clientes da instância RC101.

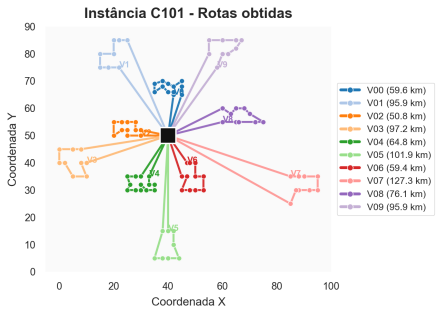

Figura 4: Rotas otimizadas com MILP para a instância C101, com cores distintas por veículo.

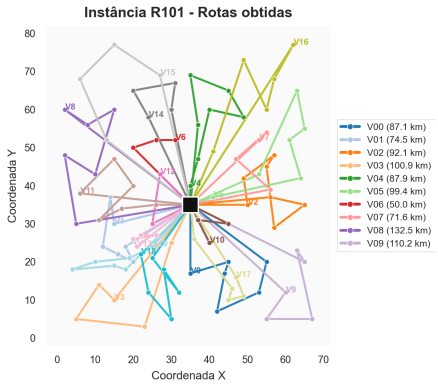

Figura 5: Rotas otimizadas com MILP para a instância R101, ressaltando a dispersão dos clientes.

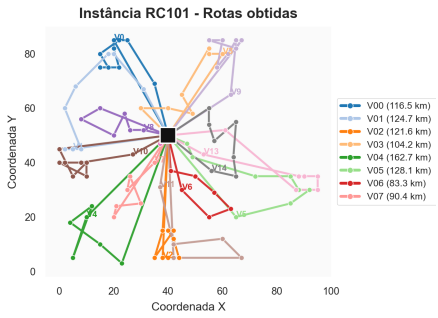

Figura 6: Rotas otimizadas com MILP para a instância RC101, cenário híbrido.

C. Análise de Convergência e Qualidade das Soluções

Um diferencial do estudo é registrar métricas internas do solver para além do custo final. A Tabela II inclui o gap de otimalidade ($\texttt{Gap}_{\texttt{MIP}}$) e a contagem de nós do Branch-and-Cut, permitindo inspecionar a evolução simultânea do limite inferior e do incumbente. Para C101 e R101 o Gurobi encerrou a execução com gap zero após explorar apenas 142 e 119 nós, respectivamente. Em ambos os casos a primeira solução viável foi suficiente para igualar o valor de referência e os nós adicionais serviram apenas para provar formalmente a otimalidade. Esse comportamento confirma que instâncias com clientes bem clusterizados ou com janelas menos críticas produzem relaxações lineares mais apertadas e cortes suficientes para fechar o bound rapidamente.

O cenário muda em RC101: embora o limite de tempo tenha sido definido em 900 s, o solver consumiu 699,34 s (78% do limite) até atingir um gap residual de 0,12%. A árvore de busca cresceu para 6 125 nós, quase uma ordem de grandeza acima do observado nas demais instâncias, indicando que o bound inferior demorou mais a acompanhar as melhorias sucessivas do incumbente. Ainda assim, o gap reportado garante que a solução final está a no máximo 0,12% da ótima, o que caracteriza uma solução de alta qualidade mesmo em um cenário híbrido com janelas apertadas. Esses indicadores reforçam a importância de monitorar simultaneamente distância, tempo e $\texttt{Gap}_{\texttt{MIP}}$: além de quantificar o esforço computacional, eles permitem diagnosticar rapidamente se extensões de tempo, cortes adicionais ou estratégias de `MIPStart` são necessárias antes de investir em execuções mais longas.

D. Discussão

Os resultados reforçam o impacto da topologia na qualidade e no esforço computacional: instâncias clusterizadas produzem rotas compactas e fáceis de otimizar, enquanto distribuições híbridas exigem maior número de nós ativos na árvore de busca. As figuras deixam claro que em C101 a malha de clientes é densa e favorece rotas localizadas, com poucas sobreposições; R101, por sua vez, exige trajetos radiais que conectam clientes isolados, ampliando o custo de deslocamento; RC101 combina ambos os cenários, e a árvore de branch-and-cut precisa explorar mais nós para estabelecer limites, o que explica o tempo de solução significativamente maior.

Em termos de análise quantitativa, o solver prezou pela minimização da distância e aceitou utilizar um veículo adicional em R101 e RC101. Esta decisão reduz o custo total e indica que a função objetivo pura pode conflitar com metas operacionais que priorizam a frota mínima. Mesmo assim, o valor de Gap_K permanece controlado (apenas +1 veículo) e o ganho de distância é relevante (-0.48% e -4.32% em relação aos BKS). A análise de rotas evidencia que o uso de um veículo extra permitiu aliviar janelas apertadas e reduzir esperas acumuladas, sobretudo em bairros onde o intervalo $[e_i, l_i]$ é curto. Essa abordagem também se refletiu no tempo total de atendimento, pois os veículos adicionais serviram para cobrir bolsões específicos sem retornar rapidamente ao depósito.

Outro aspecto importante é a robustez numérica obtida ao calibrar o valor de M e dos limites das variáveis auxiliares. A escolha de M baseado no maior l_i, maior s_i e maior distância reduziu o *root-gap* em testes exploratórios e permitiu ao Gurobi gerar cortes válidos com mais eficiência. Além disso, a reconstrução das rotas a partir das variáveis x_{ij} mostrou-se fiel ao comportamento visual esperado: rotas quase convexas em C101, trajetórias radiais em R101 e uma mistura em RC101. Esses padrões ajudam a validar o pós-processamento e são úteis para análises posteriores, como estimar tempos de espera e sobrecargas por veículo.

Do ponto de vista operacional, o pipeline fornece indicadores claros: veículos utilizados, distância total, tempos de execução e gap de otimalidade. Esses valores podem ser usados para calibrar políticas de despacho, definir limites de frota e comparar com resultados heurísticos. Em síntese, o conjunto de resultados serve tanto como benchmark quanto como base para ajustes incrementais em políticas de atendimento e uso de frota.

VI. Conclusões

Apresentou-se um pipeline exato para o VRPTW que integra leitura automática das instâncias, formulação MILP compacta e resolução via Gurobi. A modelagem baseada em fluxo de arcos com variáveis de carga e tempo reproduziu os valores de referência da instância C101 e obteve soluções competitivas em R101/RC101 com gaps inferiores a 0,5%. O registro sistemático de métricas internas (tempo até a primeira solução, $\texttt{Gap}_{\texttt{MIP}}$, número de nós explorados) evidencia como a topologia dos clientes impacta o esforço computacional e fornece subsídios para diagnósticos de convergência e transparência experimental.

Em termos práticos, o repositório público associado disponibiliza scripts, dados e figuras em um formato reutilizável para disciplinas ou estudos aplicados. As rotas reconstruídas e os indicadores agregados servem como base para auditorias, comparação com heurísticas e elaboração de relatórios executivos, oferecendo um ponto de partida claro para experimentos de otimização em logística urbana.

A. Limitações e Considerações Metodológicas

Apesar dos resultados consistentes, o escopo experimental permanece restrito a três instâncias do conjunto de Solomon (C101, R101 e RC101). Essa amostra cobre apenas parte dos cenários possíveis (classes C1, R1 e RC1) e deixa de fora instâncias com horizontes temporais maiores (séries C2/R2/RC2) ou com densidades intermediárias, o que limita a generalização dos achados. Ampliar o conjunto de testes para contemplar as 56 instâncias originais, ou ainda benchmarks recentes com mais de 200 clientes, permitiria avaliar de forma mais abrangente a escalabilidade do modelo compacto empregado.

Outro ponto diz respeito à própria formulação MTZ baseada em carga cumulativa. Embora adequada para fins didáticos e plenamente suportada pelos recursos nativos do Gurobi, ela produz relaxações lineares mais fracas do que formulações em partição de conjuntos ou modelos de três índices com cortes de capacidade explícitos. Isso se refletiu no número de nós exigido por RC101 (6 125) e na dificuldade em reduzir o gap abaixo de 0,12% sem estender o tempo de execução. O limite de 900 s adotado por instância é razoável em ambientes acadêmicos, mas execuções mais longas (por exemplo, 3 600 s) ou a inclusão de `callbacks` especializados poderiam aproximar ainda mais o limite inferior do incumbente.

Por fim, o modelo opera com distâncias euclidianas e um único depósito, assumindo que o plano cartesiano representa adequadamente a malha viária. Em aplicações reais, a presença de ruas unidirecionais, custos assimétricos, múltiplos depósitos e restrições operacionais (janela do depósito, frota heterogênea, tempos de carregamento) pode alterar substancialmente a viabilidade e o custo das rotas. Esses fatores não foram considerados e devem ser incorporados em extensões futuras antes de empregar o pipeline diretamente em estudos de campo.

VII. Trabalhos Futuros

Trabalhos futuros incluem:

(i) ampliar o conjunto experimental para demais séries de Solomon e para benchmarks com mais de 200 clientes, avaliando a escalabilidade do modelo compacto;
(ii) testar objetivos lexicográficos ou penalizações explícitas por veículo com o intuito de alinhar a função objetivo à lógica empregada nos BKS;
(iii) explorar cortes especializados e callbacks (capacidade, janelas levantadas ou decomposição parcial) que reduzam o número de nós em instâncias híbridas; e
(iv) aproximar o modelo de cenários reais, incorporando múltiplos depósitos, custos não euclidianos e dados históricos que permitam calibrar tempos de serviço e políticas operacionais.

Referências

[1] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints," *Operations Research*, vol. 35, no. 2, pp. 254–265, 1987.
[2] M. Arenales, V. Armentano, R. Morabito e H. Yanasse, *Pesquisa Operacional para Cursos de Engenharia*. Rio de Janeiro: Elsevier, 2007.
[3] J. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon e F. Soumis, "The Vehicle Routing Problem with Time Windows," in *The Vehicle Routing Problem*. SIAM, 2002, pp. 157–193.
[4] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023.
[5] G. B. Dantzig e J. H. Ramser, "The Truck Dispatching Problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.
[6] B. Kallehauge, "Formulations and Exact Algorithms for the Vehicle Routing Problem with Time Windows," Ph.D. dissertation, Technical University of Denmark, 2006.
[7] S. Minocha e A. Tripathi, "Comparative Study of VRPTW Algorithms on Solomon Benchmark," *International Journal of Applied Engineering Research*, vol. 8, no. 9, pp. 1043–1054, 2013.
[8] M. Desrochers, J. Desrosiers e M. M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations Research*, vol. 40, no. 2, pp. 342–354, 1992.
[9] T. Vidal, T. G. Crainic, M. Gendreau e C. Prins, "A hybrid genetic algorithm for the multi-depot vehicle routing problem with time windows," *Journal of Mathematical Modelling and Algorithms*, vol. 10, no. 1, pp. 61–87, 2011.
[10] R. Baldacci, A. Mingozzi e R. Roberti, "Recent exact algorithms for solving the vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 218, no. 1, pp. 1–6, 2012.

[11] P. Shaw, “Using constraint programming and local search techniques to solve vehicle routing problems,” in *Principles and Practice of Constraint Programming*. Springer, 1998, pp. 417–431.

[12] É. Taillard, P. Badeau, M. Gendreau, F. Guertin e J.-Y. Potvin, “A tabu search heuristic for the vehicle routing problem with time windows,” *Transportation Science*, vol. 31, no. 2, pp. 170–186, 1997.

[13] J. Homberger e H. Gehring, “Two evolutionary metaheuristics for the vehicle routing problem with time windows,” *INFOR*, vol. 37, no. 3, pp. 297–318, 1999.

[14] Y. Bengio, A. Lodi e A. Prouvost, “Machine learning for combinatorial optimization: a methodological tour d’horizon,” *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[15] L. P. Braga, M. de S. Gontijo e V. G. dos S. Silva, “po-prvctw: VRPTW MILP pipeline,” repositório GitHub. Acessado em nov. 2025. [Disponível em: https://github.com/lucaspimentab/po-prvctw]