Problema do Caixeiro Viajante

Modelos com Penalização de Atraso

Allan dos Santos Gustavo Santana Maycon Silva
Guilherme Sestari Caio Lima Mariana Rodrigues

1 de dezembro de 2025

Sumário

1 Introdução

Este documento apresenta a formulação matemática e implementação computacional de variantes do Problema do Caixeiro Viajante (TSP) que consideram penalizações por atraso no atendimento aos clientes. O problema consiste em determinar uma rota ótima que visita todos os clientes exatamente uma vez, retornando ao depósito inicial, minimizando diferentes métricas relacionadas ao atraso no atendimento.

1.1 Contexto e Motivação

O Problema do Caixeiro Viajante é um dos problemas mais estudados em otimização combinatória e tem aplicações diretas em diversas áreas como logística, distribuição, manufatura e prestação de serviços. Na sua formulação clássica, o TSP busca minimizar a distância total percorrida ou o tempo total de viagem. Entretanto, em aplicações práticas modernas, especialmente em contextos de e-commerce, entrega expressa e prestação de serviços com agendamento, a satisfação do cliente está fortemente ligada ao cumprimento de prazos estabelecidos.

Em cenários reais de logística, cada cliente pode ter uma janela de tempo preferencial ou um prazo máximo (deadline) para receber o atendimento. O não cumprimento desses prazos pode resultar em diversos tipos de penalidades: custos contratuais, multas, perda de reputação, insatisfação do cliente, ou até mesmo a perda de contratos futuros. Portanto, minimizar os atrasos torna-se tão importante quanto minimizar as distâncias percorridas.

Este trabalho aborda o problema sob duas perspectivas complementares que refletem diferentes estratégias de gerenciamento e políticas de atendimento ao cliente. A primeira busca minimizar o impacto agregado dos atrasos sobre toda a base de clientes, enquanto a segunda prioriza a equidade no atendimento, garantindo que nenhum cliente seja excessivamente prejudicado.

1.2 Variantes Consideradas

1.2.1 Problema A: Minimização do Atraso Total

Nesta formulação, o objetivo é minimizar a soma de todos os atrasos individuais. Esta abordagem é adequada quando:

- Existe um custo proporcional ao atraso (por exemplo, multas contratuais baseadas em tempo de atraso);
- O objetivo é minimizar o custo total de penalidades assumindo custos uniformes por unidade de tempo;
- A empresa busca otimizar o desempenho agregado do sistema de entregas;
- Existe flexibilidade para que alguns clientes tenham atrasos maiores, desde que o total seja minimizado.

Esta formulação tende a produzir soluções que podem aceitar atrasos significativos em alguns clientes se isso permitir reduzir substancialmente o atraso total. Por exemplo, pode ser preferível ter dois clientes com 10 minutos de atraso cada do que um cliente com 25 minutos de atraso, mesmo que isso pareça menos justo individualmente.

1.2.2 Problema B: Minimização do Atraso Máximo (Minimax)

Nesta formulação, o objetivo é minimizar o maior atraso individual entre todos os clientes. Esta abordagem é adequada quando:

- Existe preocupação com equidade no atendimento aos clientes;
- O custo de atrasos extremos é desproporcionalmente alto (por exemplo, perda de cliente);
- Existem acordos de nível de serviço (SLA) que estabelecem limites máximos de atraso;
- A reputação da empresa depende de evitar casos extremos de mau atendimento;
- Regulamentações ou políticas internas limitam o atraso máximo permitido.

Esta formulação frequentemente resulta em um atraso total maior do que o Problema A, mas garante que nenhum cliente seja excessivamente prejudicado. É uma abordagem mais conservadora que prioriza o pior caso (worst-case optimization), comum em contextos onde a satisfação do cliente menos satisfeito é crítica para o sucesso do negócio.

2 Modelo Matemático

Esta seção apresenta a formulação completa do modelo de programação linear inteira mista (MILP) para ambas as variantes do problema. A formulação baseia-se no TSP clássico com a adição de variáveis e restrições para modelar a propagação temporal ao longo da rota e o cálculo dos atrasos em cada cliente.

2.1 Notação e Definições

2.1.1 Conjuntos e Índices

- $N = \{0, 1, 2, \ldots, n\}$: Conjunto de todos os nós, onde 0 representa o depósito
- $V = \{1, 2, \ldots, n\}$: Conjunto de clientes (todos os nós exceto o depósito)
- $i, j \in N$: Índices utilizados para representar nós

O depósito (nó 0) é o ponto de partida e chegada do veículo. Todos os clientes devem ser visitados exatamente uma vez, formando um ciclo hamiltoniano que inicia e termina no depósito.

2.1.2 Parâmetros

- (x_i, y_i): Coordenadas cartesianas do nó $i \in N$ no plano euclidiano
- $s_i \geq 0$: Tempo de serviço (atendimento) no nó $i \in N$, com $s_0 = 0$ para o depósito
- $d_i \geq 0$: Prazo (deadline) para atendimento do cliente $i \in V$, com $d_0 = 0$ para o depósito

- c_{ij}: Tempo de viagem (distância euclidiana) entre os nós i e j, calculado como:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad \forall i, j \in N,\ i \neq j \tag{1}$$

- $M > 0$: Constante suficientemente grande (Big-M) utilizada nas restrições de propagação temporal

A matriz de distâncias $C = [c_{ij}]$ é simétrica ($c_{ij} = c_{ji}$) e possui diagonal nula ($c_{ii} = 0$). O tempo de viagem é assumido proporcional à distância euclidiana, o que é adequado para contextos urbanos sem grandes variações de velocidade.

2.1.3 Variáveis de Decisão

- $x_{ij} \in \{0, 1\}$: Variável binária que indica se o veículo viaja diretamente do nó i para o nó j ($i \neq j$)
- $t_i \geq 0$: Variável contínua que representa o tempo de início do atendimento no nó i, com $t_0 = 0$
- $L_i \geq 0$: Variável contínua que representa o atraso no atendimento do cliente $i \in V$
- $T_{\max} \geq 0$: Variável contínua que representa o atraso máximo (utilizada apenas no Problema B)

As variáveis x_{ij} definem a estrutura da rota, enquanto as variáveis t_i capturam a dimensão temporal do problema. As variáveis L_i quantificam as violações dos prazos estabelecidos.

2.2 Restrições

O modelo é composto por cinco grupos principais de restrições que garantem a viabilidade da solução e a correta propagação temporal.

2.2.1 Restrições de Grau (Conectividade)

Cada nó deve ter exatamente um arco de entrada e um de saída, garantindo que todos os clientes sejam visitados exatamente uma vez:

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

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

Estas restrições garantem que a solução forme um ou mais ciclos que cobrem todos os nós. As restrições de propagação temporal (apresentadas adiante) eliminam implicitamente a possibilidade de subciclos que não incluam o depósito.

2.2.2 Fixação do Tempo Inicial

O veículo inicia sua operação no tempo zero:

$$t_0 = 0 \tag{4}$$

Esta restrição estabelece a origem temporal da rota e serve como referência para todos os demais tempos de atendimento.

2.2.3 Restrições de Propagação Temporal

Para cada arco utilizado na solução, o tempo de início do atendimento no nó destino deve respeitar a conclusão do serviço no nó origem mais o tempo de viagem:

$$t_j \geq t_i + s_i + c_{ij} - M(1 - x_{ij}) \quad \forall i, j \in N,\ i \neq j,\ j \neq 0 \tag{5}$$

Quando $x_{ij} = 1$ (o arco é utilizado), a restrição torna-se $t_j \geq t_i + s_i + c_{ij}$, garantindo a propagação correta do tempo. Quando $x_{ij} = 0$ (o arco não é utilizado), o termo $-M$ torna a restrição redundante.

Note que esta restrição não é aplicada quando $j = 0$ (retorno ao depósito), pois o tempo de chegada ao depósito não afeta a função objetivo. Além disso, estas restrições eliminam subciclos: qualquer subciclo que não inclua o depósito seria forçado a ter tempos crescentes indefinidamente, o que é impossível em uma solução factível.

2.2.4 Definição do Atraso

O atraso em cada cliente é definido como a diferença positiva entre o tempo de início do atendimento e o prazo estabelecido:

$$L_i \geq t_i - d_i \quad \forall i \in V \tag{6}$$

Como $L_i \geq 0$ por definição e o objetivo é minimizar funções dos atrasos, em uma solução ótima teremos $L_i = \max\{0, t_i - d_i\}$. Se o cliente for atendido dentro do prazo ($t_i \leq d_i$), o valor ótimo de L_i será zero. Se houver atraso ($t_i > d_i$), então L_i assumirá o valor exato do atraso.

2.2.5 Restrição do Atraso Máximo (Apenas Problema B)

Para a formulação minimax, uma restrição adicional garante que $T_{\max}$ seja maior ou igual ao atraso de todos os clientes:

$$T_{\max} \geq L_i \quad \forall i \in V \tag{7}$$

Esta restrição força $T_{\max}$ a ser pelo menos igual ao maior atraso individual. Como o objetivo é minimizar $T_{\max}$, na solução ótima teremos $T_{\max} = \max_{i \in V} L_i$.

2.3 Funções Objetivo

2.3.1 Problema A: Minimização do Atraso Total

O objetivo é minimizar a soma de todos os atrasos individuais:

$$\min \sum_{i \in V} L_i \tag{8}$$

Esta função objetivo penaliza cada unidade de tempo de atraso igualmente, independentemente do cliente. Uma solução ótima para este problema pode apresentar atrasos concentrados em poucos clientes se isso resultar em menor atraso total.

2.3.2 Problema B: Minimização do Atraso Máximo

O objetivo é minimizar o maior atraso individual:

$$\min \ T_{\max} \tag{9}$$

Esta função objetivo implementa uma estratégia conservadora, buscando reduzir o pior caso. Uma solução ótima para este problema tende a distribuir os atrasos de forma mais equilibrada entre os clientes, mesmo que isso aumente o atraso total.

2.4 Formulação Completa

Para referência, apresentamos a formulação completa do Problema A:

$$\min \quad \sum_{i \in V} L_i \tag{10}$$

$$\text{s.a.} \quad \sum_{j \in N, j \neq i} x_{ij} = 1 \quad \forall i \in N \tag{11}$$

$$\sum_{j \in N, j \neq i} x_{ji} = 1 \quad \forall i \in N \tag{12}$$

$$t_0 = 0 \tag{13}$$

$$t_j \geq t_i + s_i + c_{ij} - M(1 - x_{ij}) \quad \forall i, j \in N,\ i \neq j,\ j \neq 0 \tag{14}$$

$$L_i \geq t_i - d_i \quad \forall i \in V \tag{15}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in N,\ i \neq j \tag{16}$$

$$t_i \geq 0 \quad \forall i \in N \tag{17}$$

$$L_i \geq 0 \quad \forall i \in V \tag{18}$$

E a formulação completa do Problema B (adiciona-se as restrições $T_{\max} \geq L_i, \forall i \in V$ e substitui-se a função objetivo por $\min T_{\max}$).

3 Resultados Computacionais

Esta seção apresenta os resultados obtidos na resolução de oito instâncias de teste, variando de 10 a 30 clientes. Todas as instâncias foram resolvidas utilizando o solver Gurobi 9.x em Python, com limite de tempo de 1800 segundos (30 minutos). Os experimentos foram executados em um ambiente Google Colab com recursos computacionais padrão.

3.1 Instâncias de Teste

As instâncias foram geradas com coordenadas aleatórias no plano euclidiano, tempos de serviço e prazos (deadlines) variados para cada cliente. A matriz de distâncias foi calculada usando a métrica euclidiana conforme especificado na Seção 2. As instâncias testadas possuem 10, 12, 15, 17, 20, 22, 25 e 30 clientes, além do depósito.

3.2 Resultados Consolidados

A Tabela 1 apresenta um resumo comparativo dos resultados obtidos para as duas formulações em todas as instâncias testadas.

Tabela 1: Comparação de resultados entre Problema A e Problema B

Instância	Nós	Atraso Total A	GAP A (%)	Tempo A (s)	Atraso Máx. B	GAP B (%)	Tempo B (s)
inst_10	10	147.76	0.00	0.37	116.87	0.00	0.23
inst_12	12	0.96	0.00	10.44	0.96	0.00	11.29
inst_15	15	77.28	0.00	160.13	49.01	0.00	104.32
inst_17	17	0.00	0.00	0.90	0.00	0.00	0.94
inst_20	20	0.00	0.00	1.99	0.00	0.00	1.09
inst_22	22	618.06*	83.97	1800.04	253.39*	82.82	1800.02
inst_25	25	90.93*	53.88	1800.03	60.82	0.00	185.93
inst_30	30	0.00	0.00	5.87	0.00	0.00	9.57

Nota: Valores marcados com * indicam soluções que não atingiram a otimalidade comprovada dentro do limite de tempo. O GAP representa a diferença percentual entre a melhor solução encontrada e o melhor limite inferior (lower bound) conhecido.

3.3 Análise Detalhada por Instância

3.3.1 Instâncias Pequenas (10-12 clientes)

Instância inst_10: Esta instância apresentou atrasos em ambas as formulações. No Problema A, a rota ótima resultou em um atraso total de 147.76 unidades de tempo, com o maior atraso individual ocorrendo no nó 7 (116.87 unidades). No Problema B, o objetivo de minimizar o atraso máximo produziu uma rota diferente que manteve o atraso máximo em 116.87, mas o atraso total aumentou para 272.42 unidades. Isto ilustra claramente o trade-off entre as duas formulações: o Problema B aceita aumentar o atraso total para não piorar o pior caso.

A rota do Problema A foi: $0 \to 2 \to 7 \to 5 \to 3 \to 8 \to 1 \to 4 \to 6 \to 0$

A rota do Problema B foi: $0 \to 2 \to 7 \to 4 \to 6 \to 1 \to 8 \to 3 \to 5 \to 0$

Note que ambas as rotas começam com a sequência $0 \to 2 \to 7$, mas divergem posteriormente. Ambos os problemas foram resolvidos rapidamente (menos de 0.5 segundos) com GAP zero.

Instância inst_12: Esta instância apresentou atrasos mínimos. Ambas as formulações convergiram para a mesma rota ótima com atraso total de 0.96 e atraso máximo de 0.96 (ocorrendo no nó 9). Esta convergência ocorre porque existe um único cliente com atraso e, portanto, minimizar o atraso total é equivalente a minimizar o atraso máximo. O tempo de resolução foi similar para ambas as formulações (aproximadamente 10-11 segundos), indicando dificuldade computacional semelhante.

3.3.2 Instâncias Médias (15-20 clientes)

Instância inst_15: Nesta instância, observamos uma diferença significativa entre as formulações. O Problema A resultou em atraso total de 77.28, com o maior atraso individual de 58.60 unidades. O Problema B reduziu o atraso máximo para 49.01, uma redução de aproximadamente 16.4%, mas aumentou ligeiramente o atraso total para 77.58.

O tempo de resolução foi consideravelmente diferente: 160.13 segundos para o Problema A versus 104.32 segundos para o Problema B. Isto sugere que, para esta instância específica, a formulação minimax foi mais fácil de resolver, possivelmente devido à geometria específica do problema e à estrutura dos prazos.

Instância inst_17: Ambas as formulações encontraram soluções sem nenhum atraso (todos os clientes foram atendidos dentro dos prazos). Quando não há atrasos, as duas formulações são equivalentes e convergem para soluções de qualidade similar. As rotas encontradas foram diferentes, mas ambas viáveis e ótimas com relação aos critérios de atraso. A resolução foi extremamente rápida (menos de 1 segundo).

Instância inst_20: Similar à inst_17, esta instância também permitiu soluções sem atrasos para ambas as formulações. Os prazos estabelecidos eram suficientemente flexíveis para permitir rotas que atendem todos os clientes dentro do tempo esperado. Novamente, a resolução foi muito rápida (1-2 segundos).

3.3.3 Instâncias Grandes (22-30 clientes)

Instância inst_22: Esta foi a instância mais desafiadora do conjunto de testes. Nenhuma das duas formulações conseguiu comprovar a otimalidade dentro do limite de 1800 segundos.

Para o Problema A, a melhor solução encontrada apresentou atraso total de 618.06, com GAP de 83.97%. O maior atraso individual nesta solução foi de 356.31 unidades (nó 13). Este GAP extremamente alto indica que o solver não conseguiu encontrar bons limites inferiores, possivelmente devido à complexidade da instância e às restrições temporais apertadas.

Para o Problema B, a melhor solução encontrada apresentou atraso máximo de 253.39, com GAP de 82.82%. Comparando com o Problema A, observamos que a estratégia minimax reduziu o pior caso de 356.31 para 253.39 (redução de 28.9%), mas o atraso total aumentou significativamente para 1901.50 unidades. Este é um exemplo claro do trade-off fundamental entre as duas abordagens.

A distribuição dos atrasos no Problema B foi muito mais equilibrada do que no Problema A, com vários clientes apresentando atrasos na faixa de 150-250 unidades, enquanto no Problema A havia grande concentração do atraso em poucos clientes.

Instância inst_25: Esta instância apresentou comportamento interessante. O Problema A não atingiu otimalidade (GAP de 53.88%), encontrando uma solução com atraso total de 90.93. Já o Problema B conseguiu provar otimalidade em 185.93 segundos, com atraso máximo de 60.82.

A solução do Problema A concentrou os atrasos principalmente nos nós 13 (60.82) e 22 (30.11). A solução do Problema B, por sua vez, distribuiu os atrasos de forma que o máximo fosse exatamente 60.82, resultando em atraso total de 97.84. Note que mesmo sem otimalidade comprovada no Problema A, a diferença entre os atrasos totais foi pequena (90.93 vs 97.84).

Instância inst_30: Esta foi a maior instância testada, mas ambas as formulações encontraram soluções sem atrasos e comprovaram otimalidade rapidamente (5.87 e 9.57

segundos). Isto demonstra que o tamanho da instância (número de clientes) não é o único fator determinante da dificuldade computacional. A estrutura dos prazos, a geometria das localizações e a folga temporal disponível são igualmente importantes.

3.4 Análise Comparativa entre Formulações

3.4.1 Trade-off entre Atraso Total e Atraso Máximo

A Tabela 2 apresenta uma análise do trade-off entre as duas formulações, mostrando a relação entre os valores obtidos em cada abordagem.

Tabela 2: Análise do trade-off entre as formulações

Instância	**Atraso Total A**	**Atraso Total B**	**Atraso Máx. A**	**Atraso Máx. B**
inst_10	147.76	272.42	116.87	116.87
inst_12	0.96	1.92	0.96	0.96
inst_15	77.28	77.58	58.60	49.01
inst_17	0.00	0.00	0.00	0.00
inst_20	0.00	0.00	0.00	0.00
inst_22	618.06*	1901.50*	356.31	253.39*
inst_25	90.93*	97.84	60.82	60.82
inst_30	0.00	0.00	0.00	0.00

Nota: Valores marcados com * não são ótimos comprovados. Os valores de "Atraso Máx. A"e "Atraso Total B"são obtidos analisando as soluções encontradas por cada formulação.

Desta tabela, podemos extrair as seguintes observações:

1. **inst_10:** O Problema B manteve o atraso máximo igual ao do Problema A (116.87), mas aumentou o atraso total em 84.4% (de 147.76 para 272.42). Isto ocorreu porque a formulação B redistribuiu os atrasos, criando atrasos em clientes que não tinham atraso na solução A (como o nó 5, que passou de 0 para 94.16).

2. **inst_12:** Ambas encontraram a mesma estrutura de solução, mas devido a diferenças numéricas mínimas nos tempos, o Problema B dobrou o atraso total (de 0.96 para 1.92), distribuindo o atraso entre dois clientes (nós 9 e 7) ao invés de concentrar em um único cliente.

3. **inst_15:** Esta instância mostra o trade-off ideal: o Problema B reduziu o atraso máximo em 16.4% (de 58.60 para 49.01) com aumento quase desprezível no atraso total (apenas 0.39%, de 77.28 para 77.58). Este é um caso onde a formulação minimax oferece clara vantagem.

4. **inst_22:** O trade-off mais extremo observado. O Problema B reduziu o atraso máximo em 28.9%, mas triplicou o atraso total (aumento de 207.6%). Esta é uma instância onde a escolha entre as formulações depende fortemente das prioridades do tomador de decisão.

5. **inst_25:** Apesar do GAP no Problema A, observamos que ambas as formulações convergiram para o mesmo atraso máximo (60.82), mas com pequena diferença no atraso total (90.93 vs 97.84).

3.4.2 Dificuldade Computacional

A Figura 3 analisa a relação entre tempo de resolução e qualidade da solução.

Tabela 3: Análise de desempenho computacional

Instância	Tempo A (s)	Tempo B (s)	GAP A (%)	GAP B (%)
inst_10	0.37	0.23	0.00	0.00
inst_12	10.44	11.29	0.00	0.00
inst_15	160.13	104.32	0.00	0.00
inst_17	0.90	0.94	0.00	0.00
inst_20	1.99	1.09	0.00	0.00
inst_22	1800.04	1800.02	83.97	82.82
inst_25	1800.03	185.93	53.88	0.00
inst_30	5.87	9.57	0.00	0.00

Observações sobre desempenho computacional:

- **Instâncias pequenas (10-12 clientes):** Ambas as formulações resolvem rapidamente, com tempos na ordem de décimos de segundo a alguns segundos. Não há diferença significativa de dificuldade.

- **Instâncias médias (15-20 clientes):** Começamos a observar diferenças. A inst_15 foi resolvida mais rapidamente pelo Problema B (104s vs 160s), enquanto a inst_20 foi mais rápida no Problema B também (1.09s vs 1.99s). As instâncias 17 e 20, por não terem atrasos, foram triviais para ambas.

- **Instâncias grandes (22-30 clientes):** Comportamento heterogêneo. A inst_22 foi extremamente difícil para ambas (ambas atingiram o limite de tempo com GAPs altos). A inst_25 apresentou uma discrepância notável: o Problema A não conseguiu otimalidade em 1800s, enquanto o Problema B provou otimalidade em apenas 186s. Isto sugere que a estrutura específica desta instância favorece a formulação minimax.

  A inst_30, apesar de ser a maior, foi resolvida rapidamente por ambas. Isto confirma que o tamanho não é o único determinante da dificuldade: a inst_22 (menor) foi muito mais difícil que a inst_30 (maior).

3.4.3 Distribuição dos Atrasos

Uma análise qualitativa importante é como os atrasos são distribuídos entre os clientes em cada formulação:

Problema A - Concentração: Tende a concentrar atrasos em poucos clientes. Por exemplo, na inst_10, dos 8 clientes, apenas 2 tiveram atraso (nós 2 e 7), com o nó 7 concentrando 79% do atraso total. Na inst_22, o nó 13 sozinho concentrou 57.6% do atraso total.

Problema B - Distribuição: Tende a distribuir os atrasos de forma mais uniforme. Na inst_10, a solução do Problema B criou atrasos em 3 clientes (nós 2, 5 e 7) ao invés de 2, distribuindo melhor o impacto. Na inst_22, múltiplos clientes apresentaram atrasos na faixa de 150-250 unidades, ao invés de concentração extrema.

Esta diferença fundamental na estrutura das soluções tem implicações práticas importantes:

- O Problema A pode ser preferível quando é aceitável ter alguns clientes com grande atraso, desde que a maioria seja atendida no prazo;
- O Problema B é mais apropriado quando a satisfação do cliente menos satisfeito é crítica, ou quando existem acordos contratuais que penalizam fortemente atrasos individuais extremos.

3.5 Padrões de Rotas

Analisando as rotas obtidas, observamos que:

1. **Rotas diferentes:** Na maioria dos casos com atrasos (inst_10, inst_15, inst_22), as duas formulações produziram rotas completamente diferentes, não apenas variações na ordem de visitação.
2. **Convergência em casos extremos:** Quando não há atrasos (inst_17, inst_20, inst_30) ou quando há apenas um cliente com atraso (inst_12), as rotas tendem a ser similares ou equivalentes.
3. **Início comum:** Em alguns casos (como inst_10), as rotas começam com a mesma sequência inicial mas divergem posteriormente. Isto sugere que decisões iniciais sobre quais clientes visitar primeiro podem ser consensuais, mas a ordenação posterior depende do critério de otimização.
4. **Complexidade da rota:** Não há padrão claro indicando que uma formulação produz rotas mais "simples"ou "complexas"que a outra. A complexidade depende mais da geometria da instância e da distribuição dos prazos.

3.6 Impacto do Parâmetro Big-M

O parâmetro Big-M foi fixado em $M = 10^5$ para todas as instâncias. Este valor mostrou-se adequado para as instâncias testadas, não causando problemas numéricos evidentes. Entretanto, é importante notar que:

- Um valor muito grande pode causar instabilidade numérica e dificultar a resolução;
- Um valor muito pequeno pode tornar as restrições de propagação temporal inválidas;
- Para instâncias maiores ou com características diferentes, pode ser necessário ajustar este parâmetro.

Uma abordagem mais sofisticada seria calcular um valor de M específico para cada instância baseado nos tempos de serviço e distâncias, mas isto não foi explorado neste trabalho.

4 Considerações Finais

4.1 Síntese dos Resultados

Este trabalho apresentou a formulação e implementação computacional de duas variantes do Problema do Caixeiro Viajante com penalizações por atraso: minimização do atraso

total (Problema A) e minimização do atraso máximo (Problema B). Os experimentos computacionais com oito instâncias de teste, variando de 10 a 30 clientes, revelaram características importantes de ambas as formulações.

Os resultados demonstraram que as duas formulações produzem soluções fundamentalmente diferentes quando há atrasos na operação. O Problema A tende a concentrar atrasos em poucos clientes para minimizar o impacto agregado, enquanto o Problema B distribui os atrasos de forma mais equilibrada para evitar casos extremos de mau atendimento. Este trade-off foi particularmente evidente na instância inst_22, onde a redução de 28.9% no atraso máximo (de 356.31 para 253.39) veio ao custo de triplicar o atraso total.

A dificuldade computacional mostrou-se dependente não apenas do tamanho da instância, mas também da estrutura dos prazos e da geometria do problema. A instância inst_22 (22 clientes) foi significativamente mais difícil que a instância inst_30 (30 clientes), com ambas as formulações atingindo o limite de tempo sem comprovar otimalidade. Por outro lado, a instância inst_25 apresentou comportamento assimétrico: o Problema B provou otimalidade em 186 segundos, enquanto o Problema A não conseguiu convergir em 1800 segundos.

Das oito instâncias testadas, três (inst_17, inst_20, inst_30) permitiram soluções sem atrasos, demonstrando que nem sempre há conflito entre os objetivos de roteamento e os prazos estabelecidos. Nestas situações, ambas as formulações convergem para soluções de qualidade equivalente.

4.2 Implicações Práticas

Os resultados obtidos têm implicações diretas para a tomada de decisão em contextos logísticos e de prestação de serviços:

Escolha da formulação apropriada:

- O Problema A é adequado quando existe um custo linear proporcional ao atraso e quando é aceitável que alguns clientes experimentem atrasos maiores, desde que o custo total seja minimizado. Este é o caso, por exemplo, de operações de entrega onde multas são proporcionais ao atraso agregado.

- O Problema B é apropriado quando acordos de nível de serviço (SLA) estabelecem limites máximos de atraso, quando a reputação da empresa depende de evitar casos extremos de insatisfação, ou quando existe preocupação com equidade no atendimento. Este seria o caso de serviços de emergência médica, onde o pior caso é crítico.

4.3 Limitações do Trabalho

Este trabalho possui algumas limitações que devem ser consideradas:

4.3.1 Limitações Metodológicas

1. **Método de resolução:** Utilizou-se apenas o solver exato Gurobi sem explorar métodos heurísticos ou metaheurísticos. Para instâncias muito grandes ou com prazos muito apertados, abordagens alternativas como algoritmos genéticos, simulated annealing ou busca tabu poderiam ser mais eficientes, especialmente quando soluções de boa qualidade (mas não necessariamente ótimas) são aceitáveis.

2. **Eliminação de subciclos:** O modelo não implementa restrições explícitas de eliminação de subciclos (como restrições de subtour elimination ou MTZ constraints). A eliminação de subciclos é garantida implicitamente pelas restrições de propagação temporal. Embora este método seja elegante e economize restrições, pode ser menos eficiente computacionalmente em algumas instâncias, especialmente quando os prazos são muito relaxados.

3. **Parâmetro Big-M:** O uso de constantes Big-M (fixadas em 10^5) pode causar problemas numéricos em instâncias com características diferentes. Métodos alternativos que evitam Big-M, como formulações baseadas em indicadores (indicator constraints), poderiam melhorar a estabilidade numérica e o desempenho computacional.

4. **Simetria:** O modelo não explora a simetria inerente ao TSP (uma rota e sua reversa têm o mesmo custo). Quebras de simetria poderiam reduzir o espaço de busca e acelerar a resolução.

4.3.2 Limitações das Instâncias

1. **Tamanho limitado:** As instâncias testadas variam de 10 a 30 clientes. Aplicações reais de logística urbana frequentemente envolvem 50-200 clientes por veículo. O comportamento do modelo em instâncias maiores não foi explorado.

2. **Veículo único:** O modelo considera apenas um veículo. Problemas reais frequentemente envolvem frotas de múltiplos veículos (Vehicle Routing Problem - VRP), o que adicionaria complexidade significativa.

3. **Prazos artificiais:** Os prazos foram gerados de forma sintética e podem não refletir padrões realistas de demanda. Em aplicações reais, os prazos frequentemente seguem padrões (por exemplo, todos os clientes em uma região podem ter prazos similares).

4. **Ausência de janelas de tempo:** O modelo considera apenas prazos finais (deadlines), sem janelas de tempo (time windows) que especifiquem tanto um tempo mínimo quanto máximo para atendimento. Muitas aplicações reais têm janelas de tempo estritas.

5. **Tempo de viagem determinístico:** Assumiu-se que os tempos de viagem são determinísticos e proporcionais à distância euclidiana. Na prática, tempos de viagem são estocásticos e afetados por tráfego, condições climáticas e outros fatores.

4.3.3 Simplificações do Modelo

1. **Custos uniformes:** O modelo assume que cada unidade de tempo de atraso tem o mesmo custo para todos os clientes. Na prática, diferentes clientes podem ter diferentes penalidades por atraso.

2. **Sem custos de roteamento:** As funções objetivo consideram apenas atrasos, ignorando o custo de roteamento (distância percorrida, combustível, tempo do motorista). Um modelo mais realista incluiria uma combinação ponderada de custos de roteamento e penalidades por atraso.

3. **Capacidade ilimitada:** Não há restrição de capacidade do veículo. Em aplicações de entrega de mercadorias, a capacidade do veículo é uma restrição crítica.

4. **Todos os clientes devem ser atendidos:** O modelo não permite recusar clientes. Em algumas aplicações, pode ser preferível não atender um cliente (ou adiá-lo para outro dia) ao invés de atendê-lo com grande atraso.

O código implementado está disponível no Apêndice A e pode ser adaptado para diferentes contextos de aplicação, servindo como base para as extensões sugeridas acima.

A Descrição do Código e Implementação dos Modelos

Este apêndice apresenta o código-fonte utilizado para a leitura das instâncias e para a modelagem dos dois problemas estudados: (i) minimização do atraso total e (ii) minimização do atraso máximo.

O objetivo deste apêndice é permitir a reprodutibilidade dos experimentos, bem como esclarecer a estrutura das funções empregadas.

A.1 Carregamento das Instâncias

A função `carregar_instancia` realiza a leitura dos arquivos de entrada no formato especificado: a primeira linha contém o número total de nós, a segunda contém as coordenadas do depósito, e as demais linhas descrevem cada cliente com suas coordenadas, tempo de serviço e prazo (deadline).

O retorno consiste em três listas: as coordenadas dos nós, tempos de serviço, e deadlines correspondentes.

```
# ========================
# Função para carregar instância de arquivo .txt
# ========================
def carregar_instancia(path):
    with open(path, "r") as f:
        linhas = [l.strip() for l in f if l.strip()]
    n = int(linhas[0])  # número de nós (depósito + clientes)

    # depósito
    x0, y0 = map(int, linhas[1].split())
    coords = [(x0, y0)]
    service = [0]
    deadline = [0]

    # clientes
    for linha in linhas[2:]:
        x, y, s, d = map(int, linha.split())
        coords.append((x, y))
        service.append(s)
        deadline.append(d)

    return coords, service, deadline


# Executar para cada arquivo
paths = [
    "/content/drive/MyDrive/Problema_1/inst_10.txt",
    "/content/drive/MyDrive/Problema_1/inst_12.txt",
    "/content/drive/MyDrive/Problema_1/inst_15.txt",
    "/content/drive/MyDrive/Problema_1/inst_17.txt",
    "/content/drive/MyDrive/Problema_1/inst_20.txt",
    "/content/drive/MyDrive/Problema_1/inst_22.txt",
    "/content/drive/MyDrive/Problema_1/inst_25.txt",
```

```
    "/content/drive/MyDrive/Problema_1/inst_30.txt",
]

for arquivo in paths:
    print(f"\n>>> Rodando instância: {arquivo}")
    coords, service, deadline = carregar_instancia(arquivo)
    mA = resolver_A(coords, service, deadline)
    mB = resolver_B(coords, service, deadline)
```

A.2 Modelo A — Minimização do Atraso Total

O modelo `resolver_A` implementa o problema de roteamento visando minimizar a soma dos atrasos. A estrutura do modelo inclui:

- variáveis binárias de decisão x_{ij}, indicando a visita do nó i ao nó j; - variáveis contínuas t_i representando o tempo de chegada em cada nó; - variáveis L_i para medir o atraso individual de cada cliente; - restrições de grau garantindo que cada nó possui exatamente uma entrada e uma saída; - restrições de propagação de tempo via Big-M; - função objetivo dada pela soma dos atrasos individuais.

```
import gurobipy as gp
from gurobipy import GRB
import math
import time

# =========================
# Letra A | Minimizar atraso total
# =========================
def resolver_A(coords, service, deadline):
    inicio = time.time()  # marca o início da execução

    n = len(coords)
    # Construção da matriz de distâncias euclidianas entre todos os pares de
    ↪  nós
    D = [[0.0]*n for _ in range(n)]
    for i in range(n):
        xi, yi = coords[i]
        for j in range(n):
            if i != j:
                xj, yj = coords[j]
                D[i][j] = math.sqrt((xi - xj)*2 + (yi - yj)*2)

    # Impressão da matriz de distâncias
    def imprimir_matriz(D):
      n = len(D)
      print("\nMatriz de distâncias letra A:")
      header = "     " + " ".join([f"{j:8d}" for j in range(n)])
      print(header)
      for i in range(n):
          linha = f"{i:3d} " + " ".join([f"{D[i][j]:8.2f}" for j in
          ↪  range(n)])
          print(linha)
```

```
imprimir_matriz(D)

# Criação do modelo de otimização
model = gp.Model("TSP_Base")
model.Params.OutputFlag = 0
model.Params.LogToConsole = 0
model.Params.TimeLimit = 1800   # limite de tempo de 30 minutos

# Variáveis de decisão:
x = model.addVars(n, n, vtype=GRB.BINARY, name="x")
t = model.addVars(n, vtype=GRB.CONTINUOUS, name="t")
L = model.addVars(range(1, n), vtype=GRB.CONTINUOUS, lb=0.0, name="L")
M = 1e5

# Restrições de grau (cada nó tem uma entrada e uma saída)
for i in range(n):
    model.addConstr(x[i, i] == 0)
    model.addConstr(gp.quicksum(x[i, j] for j in range(n) if j != i) == 1)
    model.addConstr(gp.quicksum(x[j, i] for j in range(n) if j != i) == 1)
model.addConstr(t[0] == 0)

# Propagação de tempo usando Big-M
for i in range(n):
    for j in range(n):
        if i != j and j != 0:
            model.addConstr(t[j] >= t[i] + service[i] + D[i][j] - M*(1 -
            ↪  x[i, j]))

# Definição dos atrasos: L[i] >= t[i] - deadline[i]
for i in range(1, n):
    model.addConstr(L[i] >= t[i] - deadline[i])

# Objetivo: minimizar soma dos atrasos
model.setObjective(gp.quicksum(L[i] for i in range(1, n)), GRB.MINIMIZE)
model.optimize()

# Função auxiliar para reconstruir a rota a partir das variáveis x[i,j]
def reconstruir():
    rota = [0]
    atual = 0
    visitados = {0}
    while True:
        proximo = None
        for j in range(n):
            if j != atual and x[atual, j].X > 0.5:
                proximo = j
                break
        if proximo is None: break
        rota.append(proximo)
        if proximo == 0: break
        if proximo in visitados: break
        visitados.add(proximo)
```

```
            atual = proximo
        return rota

    fim = time.time()  # marca fim da execução

    # Impressão dos resultados
    if model.Status in (GRB.OPTIMAL, GRB.TIME_LIMIT):
        rota = reconstruir()
        valor = model.ObjVal
        atrasos = {i: L[i].X for i in range(1, n)}
        maior_atraso = max(atrasos.values()) if atrasos else 0.0
        print("\n=== Letra A ===")
        print("Rota:", " → ".join(map(str, rota)))
        print("Tempos de chegada t[i]:", [t[i].X for i in range(n)])
        print("Atrasos L[i]:", atrasos)
        print(f"Valor (soma dos atrasos): {valor:.2f}")
        print(f"Maior atraso individual: {maior_atraso:.2f}")
        print(f"Tempo de execução: {fim - inicio:.2f} segundos")
        print(f"GAP: {model.MIPGap:.4f}")
    else:
        print("\n=== Letra A ===\nInviável.")

    return model
```

A.3 Modelo B — Minimização do Atraso Máximo

O segundo modelo, `resolver_B`, é semelhante ao anterior, porém introduz uma variável adicional $T_{\max}$, destinada a representar o maior atraso entre todos os clientes.

A função objetivo consiste em minimizar $T_{\max}$, enquanto as restrições asseguram que esta variável seja maior ou igual ao atraso individual de cada cliente.

```
# ========================
# Letra B | Minimizar máximo atraso
# ========================
def resolver_B(coords, service, deadline):
    inicio = time.time()  # marca início da execução

    n = len(coords)
    # Construção da matriz de distâncias euclidianas entre todos os pares de
    ↪ nós
    D = [[0.0]*n for _ in range(n)]
    for i in range(n):
        xi, yi = coords[i]
        for j in range(n):
            if i != j:
                xj, yj = coords[j]
                D[i][j] = math.sqrt((xi - xj)*2 + (yi - yj)*2)

    # Impressão da matriz de distâncias
    def imprimir_matriz(D):
      n = len(D)
      print("\nMatriz de distâncias letra B:")
```

```
    header = "     " + " ".join([f"{j:8d}" for j in range(n)])
    print(header)
    for i in range(n):
        linha = f"{i:3d} " + " ".join([f"{D[i][j]:8.2f}" for j in
        ↪  range(n)])
        print(linha)

imprimir_matriz(D)

# Criação do modelo de otimização
model = gp.Model("TSP_Base")
model.Params.OutputFlag = 0
model.Params.LogToConsole = 0
model.Params.TimeLimit = 1800   # limite de tempo de 30 minutos

# Variáveis de decisão
x = model.addVars(n, n, vtype=GRB.BINARY, name="x")
t = model.addVars(n, vtype=GRB.CONTINUOUS, name="t")
L = model.addVars(range(1, n), vtype=GRB.CONTINUOUS, lb=0.0, name="L")
M = 1e5
Tmax = model.addVar(vtype=GRB.CONTINUOUS, lb=0.0, name="Tmax")  # atraso
↪  máximo

# Restrições de grau (cada nó tem uma entrada e uma saída)
for i in range(n):
    model.addConstr(x[i, i] == 0)
    model.addConstr(gp.quicksum(x[i, j] for j in range(n) if j != i) == 1)
    model.addConstr(gp.quicksum(x[j, i] for j in range(n) if j != i) == 1)
model.addConstr(t[0] == 0)

# Propagação de tempo usando Big-M
for i in range(n):
    for j in range(n):
        if i != j and j != 0:
            model.addConstr(t[j] >= t[i] + service[i] + D[i][j] - M*(1 -
            ↪  x[i, j]))

# Definição dos atrasos: L[i] >= t[i] - deadline[i]
for i in range(1, n):
    model.addConstr(L[i] >= t[i] - deadline[i])

# Restrição para atraso máximo
for i in range(1, n):
    model.addConstr(Tmax >= L[i])

# Objetivo: minimizar atraso máximo
model.setObjective(Tmax, GRB.MINIMIZE)
model.optimize()

# Função auxiliar para reconstruir a rota a partir das variáveis x[i,j]
def reconstruir():
    rota = [0]
```

```
        atual = 0
        visitados = {0}
        while True:
            proximo = None
            for j in range(n):
                if j != atual and x[atual, j].X > 0.5:
                    proximo = j
                    break
            if proximo is None: break
            rota.append(proximo)
            if proximo == 0: break
            if proximo in visitados: break
            visitados.add(proximo)
            atual = proximo
        return rota

    fim = time.time()  # marca fim da execução

    # Impressão dos resultados
    if model.Status in (GRB.OPTIMAL, GRB.TIME_LIMIT):
        rota = reconstruir()
        atrasos = {i: t[i].X - deadline[i] for i in range(1, n)}
        soma_atrasos_pos = sum(v for v in atrasos.values() if v > 0)
        print("\n=== Letra B ===")
        print("Rota:", " → ".join(map(str, rota)))
        print("Tempos de chegada t[i]:", [t[i].X for i in range(n)])
        print("Atrasos reais t[i]-deadline[i]:", atrasos)
        print(f"Valor (atraso máximo): {Tmax.X:.2f}")
        print(f"Soma dos atrasos: {soma_atrasos_pos:.2f}")
        print(f"Tempo de execução: {fim - inicio:.2f} segundos")
        print(f"GAP: {model.MIPGap:.4f}")
    else:
        print("\n=== Letra B ===\nInviável.")

    return model
```