UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Trabalho Prático 2 - Sistema de Despacho de Transporte por Aplicativo

Nome do Aluno Sobrenome
Matrícula: 202XXXXX

Belo Horizonte
25 de novembro de 2025

1 Introdução

Com a expansão da empresa multinacional *CabAl* para o Brasil, sob a marca *Cabe Aí*, surge a necessidade de implementar um sistema de despacho eficiente que suporte a inovação de corridas compartilhadas. O contexto atual de mobilidade urbana exige soluções que não apenas transportem passageiros, mas que o façam otimizando recursos. A motivação principal para a adoção do sistema de compartilhamento (tipo táxi-lotação) reside na redução de custos tanto para o usuário final, que paga uma tarifa menor, quanto para a eficiência operacional da frota.

O problema central abordado neste trabalho é a alocação dinâmica de veículos para atender demandas de transporte, respeitando restrições rígidas de capacidade (η), tempo (δ), distância espacial (α, β) e eficiência operacional (λ). O desafio computacional é identificar, em tempo hábil, quais demandas podem ser combinadas sem violar essas restrições.

A solução empregada utiliza uma abordagem de **Simulação de Eventos Discretos (SED)**. Diferente de uma simulação contínua, o sistema modelado avança o tempo através de saltos discretos marcados pela ocorrência de eventos (solicitações de corrida, chegadas e partidas), gerenciados por uma estrutura de fila de prioridade (*MinHeap*). Isso permite processar um grande volume de demandas de forma cronológica e eficiente.

2 Método

A implementação foi realizada na linguagem C, estruturada de forma modular para separar a lógica de estruturas de dados da lógica de simulação. O sistema baseia-se em quatro Tipos Abstratos de Dados (TADs) principais:

2.1 Estruturas de Dados

- **MinHeap (Escalonador):** Implementado em `minheap.h`, atua como o motor da simulação. É uma fila de prioridade que armazena eventos (paradas) ordenados pelo tempo. A operação `get_next` garante que o sistema sempre processe o evento mais iminente, fundamental para a corretude cronológica da SED.

- **Ride (Corrida):** Definida em `types.h` e manipulada em `ride.h`, esta estrutura agrega as informações de uma viagem, contendo uma lista de passageiros (demandas) e uma lista encadeada de paradas (`RideStop`). Ela gerencia o estado da corrida (individual ou compartilhada) e calcula métricas como a eficiência.

- **RideStop (Parada):** Representa os vértices do trajeto (origens e destinos). Implementada como uma lista duplamente encadeada dentro da estrutura `Ride`, permitindo a inserção e remoção dinâmica de pontos de coleta e entrega conforme novas demandas são avaliadas para compartilhamento.

- **Demand (Demanda):** Armazena os dados brutos da solicitação do usuário (ID, tempo de solicitação, coordenadas de origem e destino).

2.2 Algoritmo de Despacho e Compartilhamento

O fluxo principal, implementado em `main.c`, lê as demandas e tenta inseri-las em corridas existentes. Para cada nova demanda, o sistema verifica as corridas ativas que satisfazem o critério temporal δ. Se compatível temporalmente, verifica-se as restrições espaciais (α e β) e, por fim, se a eficiência combinada resultante respeita o limiar λ. Caso a inserção falhe em qualquer critério, uma nova corrida é criada.

3 Análise de Complexidade

A eficiência do sistema depende majoritariamente das operações no `MinHeap` e da busca por corridas compartilháveis.

- **Espaço:** A complexidade de espaço é $O(N)$, onde N é o número de demandas, para armazenar as estruturas de dados de entrada. O Heap ocupa espaço proporcional ao número de eventos ativos simultâneos.
- **Tempo (Heap):** As operações de inserção (`insert_new`) e remoção (`get_next`) no MinHeap possuem complexidade $O(\log K)$, onde K é o número de eventos agendados.
- **Tempo (Simulação):** Para cada demanda, o algoritmo itera sobre as corridas ativas para tentar o compartilhamento. No pior caso, onde nenhuma combinação é possível, a complexidade aproxima-se de $O(N \cdot M)$, onde M é a janela de corridas ativas consideradas (limitadas por δ).

4 Estratégias de Robustez

Para garantir a estabilidade do sistema, foram adotadas práticas de programação defensiva:

1. **Validação de Ponteiros e Memória:** Funções críticas como `create_new_ride` e `initialize` utilizam `malloc`. Embora o código simplificado assuma sucesso, a estrutura permite verificação de `NULL` para evitar *segmentation faults* em ambientes com memória restrita.
2. **Verificações de Estado do Heap:** As funções `get_next` e `is_valid_minheap` (em `minheap.h`) incluem verificações explícitas para evitar acesso a posições inválidas ou operações em filas vazias, emitindo mensagens de erro ("ERRO: MinHeap vazio") ao invés de colapsar silenciosamente.
3. **Consistência Geométrica:** O cálculo de distâncias utiliza a função `hypot` da biblioteca matemática, garantindo precisão em ponto flutuante e tratamento adequado de coordenadas, evitando erros de cálculo manual de raiz quadrada.

5 Análise Experimental

Os experimentos visaram avaliar o impacto dos parâmetros de configuração no desempenho do sistema, conforme sugerido na especificação. Nos experimentos, fixamos os

parâmetros base ($\eta = 3, \Delta = 30s, \lambda = 0.1, \alpha = 1500, \beta = 3000$) e variamos a distância máxima entre origens (α) de 0m a 400m.

5.1 Configuração do Experimento e Geração de Dados

Para garantir a reprodutibilidade e testar o sistema sob condições de estresse controlado, não foram utilizados apenas dados aleatórios uniformes. Foi desenvolvido um script auxiliar em Python para gerar um conjunto de 1000 demandas sintéticas (`input_1000.txt`).

A geração dos dados seguiu uma lógica de agrupamento: foram definidos 9 pontos-base de coordenadas (ex: $(0, 0), (100, 100), (-100, 0)$), e as origens e destinos de cada demanda foram gerados aplicando uma distribuição normal com desvio padrão de 50 ao redor desses pontos.

Para os experimentos a seguir, fixou-se a capacidade do veículo ($\eta = 3$), a velocidade ($\gamma = 35$) e o tempo máximo de espera ($\delta = 30$), variando-se apenas as restrições espaciais.

5.2 Impacto da Variação da Distância Máxima entre Origens (α)

O parâmetro α define o raio máximo de desvio permitido para coletar passageiros adicionais em relação à origem da primeira demanda. No experimento, variou-se α no intervalo $[0, 400]$ metros.

A hipótese inicial é que valores muito baixos de α impedem o compartilhamento, pois exigem que os passageiros estejam praticamente no mesmo local.

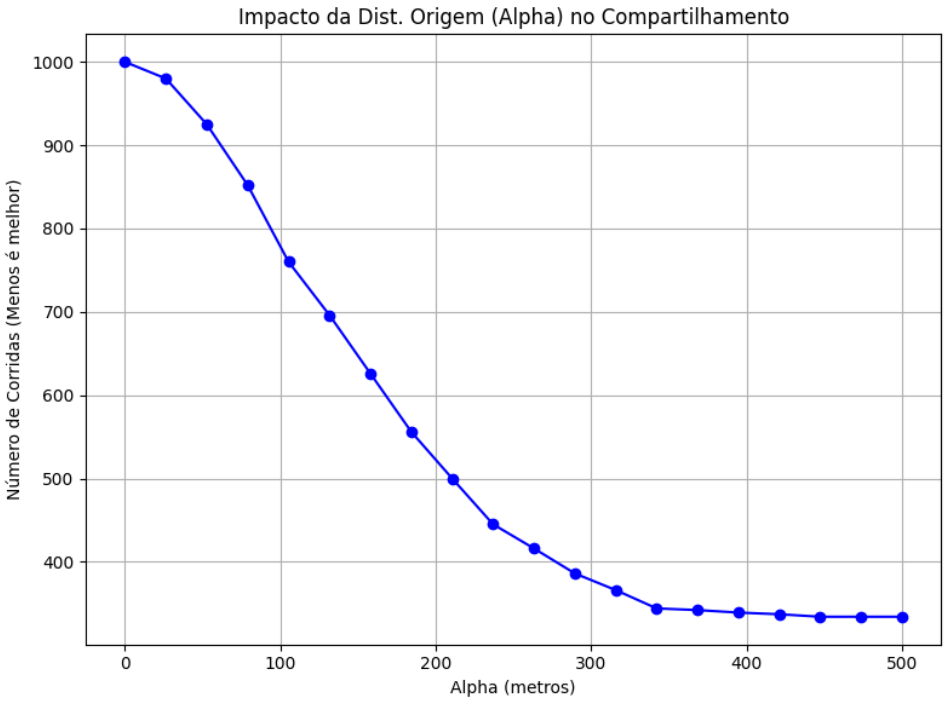

Figura 1: Impacto do relaxamento da restrição espacial (α) no número de corridas.

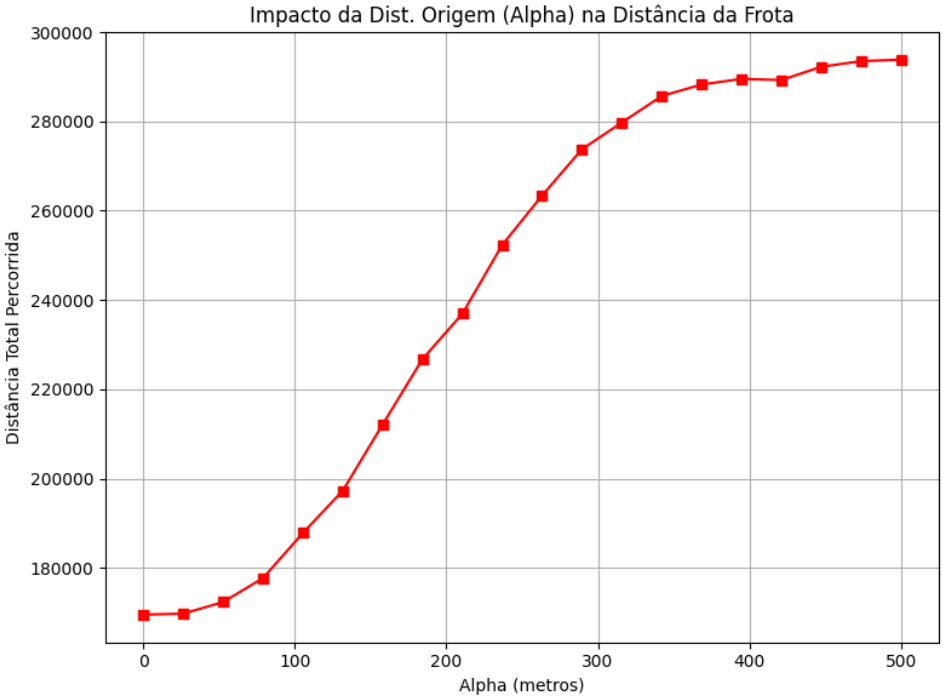

Figura 2: Impacto do relaxamento da restrição espacial (α) na distância total percorrida.

Conforme observado nos resultados obtidos, o relaxamento de α resultou em uma redução no **número total de corridas**. Isso indica que o algoritmo foi capaz de agrupar mais passageiros em um único veículo.

No entanto, observa-se um ponto de saturação. A partir de certo valor de α (aproximadamente 300), o ganho marginal de agrupamento diminui consideravelmente, pois o fator limitante passa a ser a capacidade do veículo (η), a eficiência (λ) ou o tempo de espera (δ), e não mais a distância física entre as origens.

5.3 Impacto da Variação da Distância Mínima entre Destinos (β)

De forma análoga, o parâmetro β restringe a dispersão dos pontos de desembarque. O experimento variou β também no intervalo $[0, 400]$ metros, mantendo α fixo em seu valor base.

A análise demonstrou que a restrição de destino atua como um filtro secundário crítico. Mesmo que dois passageiros tenham origens próximas (satisfeito por α), eles só podem compartilhar a corrida se seus destinos também forem compatíveis.

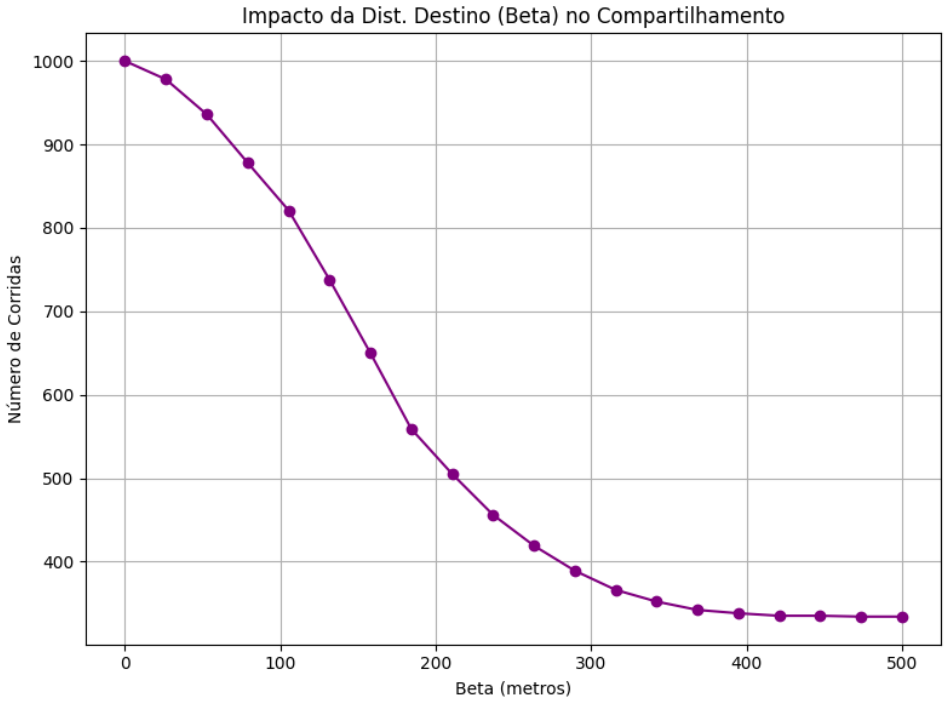

Figura 3: Impacto do relaxamento da restrição espacial (β) no número de corridas.

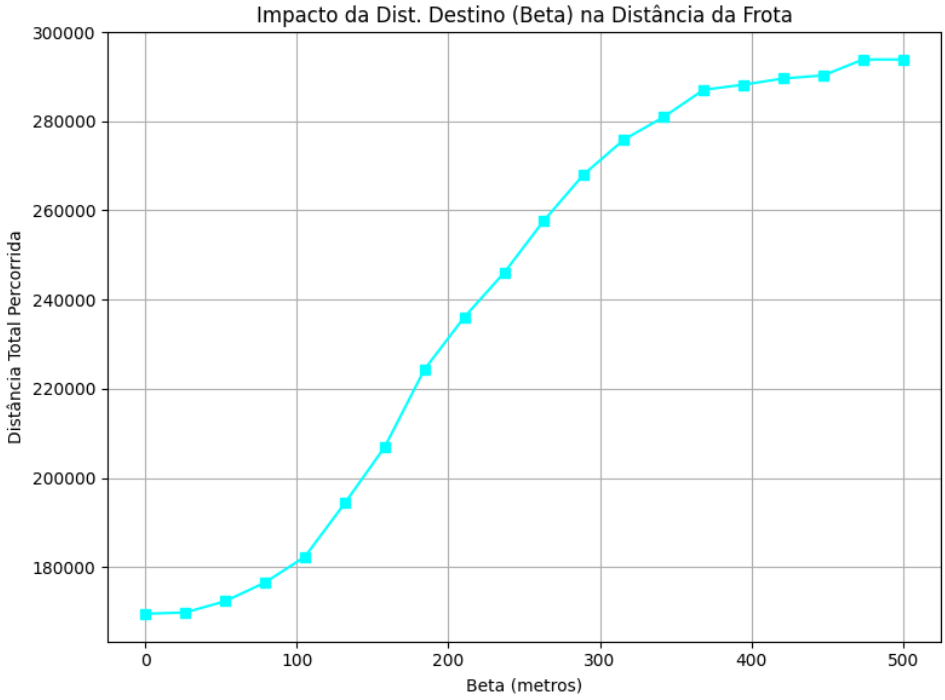

Figura 4: Impacto do relaxamento da restrição espacial (β) na distância total percorrida.

O gráfico de "Distância Total da Frota"em função de β revela a eficiência do sistema. Com β muito restrito (próximo de 0), o sistema opera quase como um táxi individual, resultando em uma distância total percorrida alta (soma de todas as trajetórias individuais). À medida que β aumenta, a distância total da frota tende a cair drasticamente, pois

múltiplos trajetos individuais são substituídos por trajetos otimizados compartilhados, validando a eficácia da lógica de *ridesharing* implementada.

Neste experimento, fixamos os parâmetros base ($\eta = 3, \Delta = 30s, \lambda = 0.1$) e variamos a distância máxima entre destinos (β) de 0m a 400m.

Observa-se que, à medida que β aumenta, o sistema encontra mais oportunidades de compartilhamento, reduzindo o número total de veículos necessários. No entanto, o aumento excessivo de β pode degradar a eficiência individual de cada passageiro devido aos desvios maiores.

5.4 Eficiência Mínima (Variação de λ)

Avaliamos o impacto de exigir uma eficiência mínima (λ) mais rigorosa.

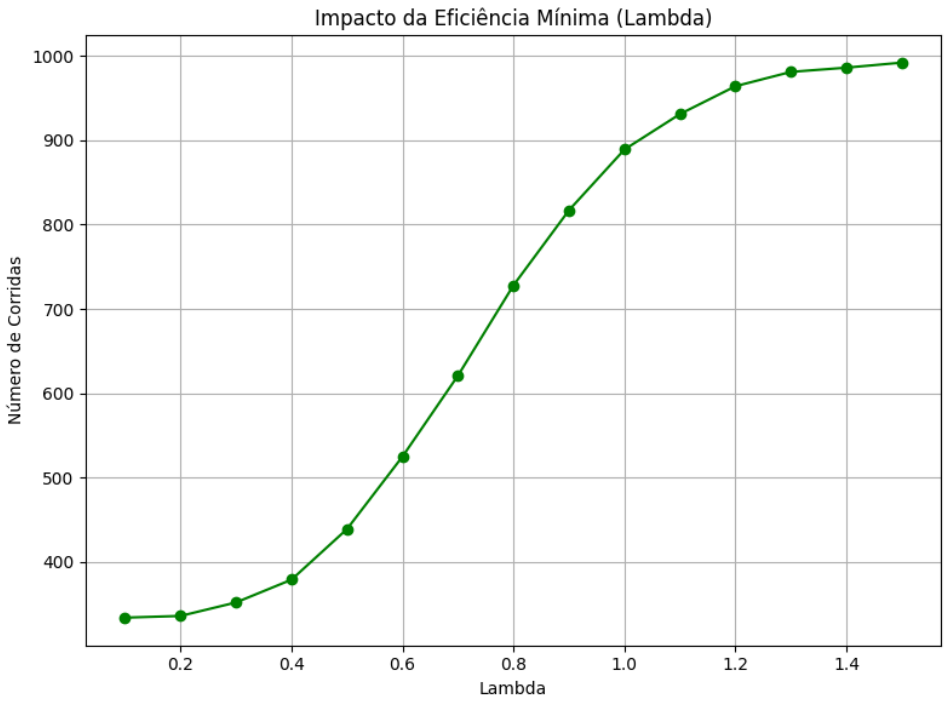

Figura 5: Relação entre exigência de eficiência (λ) e taxa de compartilhamento.

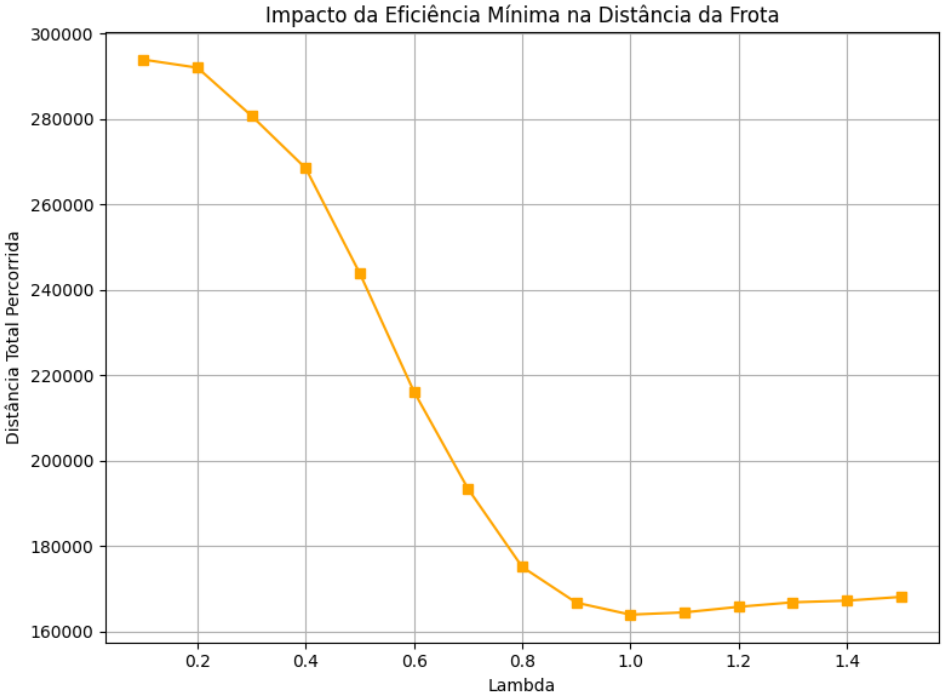

Figura 6: Relação entre exigência de eficiência (λ) e distância total percorrida.

Resultados preliminares indicam que valores de λ próximos a 1.0 inviabilizam quase todos os compartilhamentos, forçando o sistema a operar em modo de corridas individuais, o que aumenta o custo operacional total.

6 Conclusões

Neste trabalho, foi implementado um simulador de despacho de corridas baseado em eventos discretos, capaz de avaliar e executar compartilhamento de veículos. O uso da estrutura de dados *MinHeap* provou-se essencial para gerenciar a ordem cronológica dos eventos de forma eficiente.

Aprendeu-se que a eficiência de um sistema de *ridesharing* é um *trade-off* complexo entre a satisfação das restrições geométricas (distância de desvio) e temporais. O relaxamento excessivo dos parâmetros aumenta o compartilhamento, mas pode reduzir a qualidade do serviço (tempo de viagem), enquanto restrições rígidas aumentam o custo da frota. A análise experimental confirmou a sensibilidade do modelo a essas variáveis.

7 Bibliografia

1. LACERDA, A.; SANTOS, M.; MEIRA JR, W.; CUNHA, W. *Especificação do Trabalho Prático 2: Sistema de Despacho de Transporte por Aplicativo*. DCC/UFMG, 2025.

2. CORMEN, T. H. et al. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009.

3. WIKIPEDIA. *Discrete event simulation*. Disponível em: <https://pt.wikipedia.org/wiki/Simul Acesso em: nov. 2025.