Aplicação de Algoritmos de Computação Evolucionária para o Problema do Caixeiro Viajante (TSP) para Rotas de Bares Participantes do Projeto "Comida Di Buteco"

1st Gabrielli Valelia Sousa da Silva
Escola de Engenharia UFMG
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
gabriellivalelia@gmail.com

2nd Júlia Diniz Rodrigues
Escola de Engenharia UFMG
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
juliadinizrodrigues@gmail.com

***Resumo*—Este trabalho apresenta uma abordagem baseada em meta-heurísticas para otimização de rotas no contexto do evento gastronômico "Comida di Buteco". O problema é modelado como uma variante do Problema do Caixeiro Viajante com Janelas de Tempo (TSPTW), considerando restrições reais de horários de funcionamento, avaliações dos estabelecimentos e tempos de deslocamento obtidos via Google Maps. Foram implementados e comparados algoritmos clássicos (Kruskal, Bellmore-Nemhauser), Ant Colony Optimization (ACO) e Busca Tabu (Tabu Search), integrados a uma aplicação web interativa. Os resultados demonstram que a Busca Tabu, adaptada para múltiplos dias e restrições práticas, supera as heurísticas construtivas e apresenta desempenho competitivo frente ao ACO, proporcionando soluções de alta qualidade em tempo viável para instâncias reais.**

***Palavras-chave*—Otimização, TSP, Busca Tabu, Ant Colony Optimization, Meta-heurísticas, Google Maps, Rotas, Janelas de Tempo, Comida di Buteco**

I. Introdução

Problemas de otimização combinatória, como o clássico Problema do Caixeiro Viajante (TSP), representam desafios computacionais significativos devido à sua natureza NP-difícil. Em cenários reais, a complexidade cresce com a presença de restrições dinâmicas, como janelas de tempo e tráfego urbano, configurando variantes como o TSP com janelas de tempo ((TSPTW – *Traveling Salesperson Problem with Time Windows*)) e problemas de roteamento com restrições temporais e de serviço. Nessas circunstâncias, métodos exatos tornam-se inviáveis à medida que o número de vértices cresce, o que demanda o uso de meta-heurísticas que consigam explorar eficientemente espaços de busca vastos [1].

Neste contexto, o presente trabalho propõe a aplicação da meta-heurística *Tabu Search* para otimizar o roteiro de visitação do evento gastronômico “Comida di Buteco”, cuja descrição oficial está disponível no site institucional do festival [2]. A escolha se justifica pela capacidade da Tabu Search de escapar de mínimos locais e explorar regiões promissoras, mesmo em espaços de busca altamente não lineares e com restrições complexas [3]. A motivação prática advém da dificuldade que os participantes enfrentam para planejar rotas que maximizem a visitação de bares bem avaliados e a qualidade da experiência, respeitando horários restritos e variabilidade no tempo de deslocamento.

Além disso, este estudo adota uma etapa sistemática de **validação das heurísticas**. Para tanto, foram implementados e comparados algoritmos clássicos e contemporâneos, como: uma heurística construtiva baseada em AGM (via Kruskal) [4], a heurística de inserção de Bellmore–Nemhauser [5], e uma versão moderna de *Ant Colony Optimization (ACO)*, além da Tabu Search. Essa validação permite estabelecer referências de custo, padrões de convergência e estabilidade, contribuindo para avaliar a eficácia da abordagem proposta em diferentes cenários.

O desenvolvimento foi realizado em Python, aproveitando seu amplo ecossistema de bibliotecas, e integrou a API do Google Maps para geração de uma matriz realista de tempos de deslocamento, substituindo a simplificação por distâncias euclidianas, mais comum em estudos teóricos.

Este artigo está organizado da seguinte forma: a Seção II apresenta a modelagem, a construção da matriz de custos e a descrição dos algoritmos; a Seção III traz os resultados dos experimentos e comparações entre métodos; e a Seção IV discute as conclusões obtidas e propõe direções para trabalhos futuros.

II. Metodologia

A abordagem proposta para a otimização das rotas do evento "Comida di Buteco"baseia-se na modelagem do problema como uma variação do Problema do Caixeiro Viajante com Janelas de Tempo (TSPTW), solucionado através da meta-heurística Busca Tabu (*Tabu Search*).

A. Coleta de Dados e Modelagem do Grafo

Diferente das abordagens clássicas que utilizam distâncias euclidianas, este trabalho prioriza o tempo de deslocamento real como métrica de custo. Para isso, foi utilizada a API do *Google Maps* para gerar a matriz de distâncias entre os estabelecimentos e, a partir dela, estimar o tempo de deslocamento considerando uma velocidade média urbana de 18 km/h, conforme apresentado em [6]. Dessa forma, a matriz de custos reflete de maneira mais realista as condições de mobilidade encontradas no cenário do problema.

O problema é representado por um grafo orientado $G = (V, A)$, onde V é o conjunto de vértices (bares) e A o conjunto de arestas representando o trajeto. Cada vértice v_i possui os seguintes atributos associados:

- **Janela de Tempo:** Horário de abertura (O_i) e fechamento (C_i);
- **Avaliação (R_i):** Nota do estabelecimento no Google (utilizada na função de aptidão);
- **Tempo de Serviço (S_i):** Tempo médio de permanência no local, definido por padrão como 60 minutos.

B. Validação e Métodos de Referência

Antes da aplicação da meta-heurística principal, diferentes heurísticas clássicas foram utilizadas como parâmetros para validar a qualidade das soluções obtidas. Embora elas não considerem janelas de tempo, fornecem limites inferiores, superiores e padrões de referência fundamentais para avaliar o desempenho dos métodos implementados.

1) Árvore Geradora Mínima (Kruskal): A Árvore Geradora Mínima (AGM) foi obtida pelo algoritmo de Kruskal, implementado com Union–Find com compressão de caminho e união por rank. Apesar de não constituir uma solução para o TSP, a AGM estabelece um limite inferior teórico para o custo mínimo.

Algorithm 1 Algoritmo de Kruskal

```
Ordenar as arestas por peso crescente.
for all arestas (u, v) ordenadas do
    if Find(u) ≠ Find(v) then
        Union(u, v)
        Adicionar (u, v) à AGM
    end if
end for
```

2) Heurística de Bellmore–Nemhauser: A heurística construtiva de Bellmore–Nemhauser produz rapidamente uma rota para o TSP clássico, sem janelas de tempo. A cada passo, o algoritmo expande parcialmente a rota inserindo o vértice que causa o menor aumento no custo total. Essa abordagem gera uma solução inicial determinística de baixo custo computacional, útil como limite superior simples.

Algorithm 2 Bellmore–Nemhauser

```
Escolher um vértice inicial v_0
Rota R ← [v_0]
while existem vértices não visitados do
    Para cada vértice não visitado u, calcular o custo de inserção em todas as posições de R
    Inserir o vértice u e posição que minimizam o custo
    Atualizar R
end while
Fechar o ciclo adicionando o retorno ao vértice inicial
```

3) Otimização por Colônia de Formigas (ACO): A heurística ACO foi implementada em três variações que diferem quanto à influência relativa do feromônio, da heurística baseada em distância ($1/d$) e da taxa de evaporação. A cada iteração, cada formiga constrói uma rota probabilisticamente, guiada pelo histórico de boas soluções (feromônios) e pela proximidade entre vértices.

Após a construção das rotas, o feromônio é atualizado reforçando caminhos de menor custo, enquanto parte do feromônio global evapora.

Algorithm 3 ACO

```
Inicializar matriz de feromônios τ
for cada iteração do
    for cada formiga k do
        Construir solução probabilística baseada em τ e na heurística 1/d
        Calcular custo da solução
    end for
    Evaporar feromônio: τ ← (1 − ρ)τ
    Depositar feromônio proporcional ao inverso do custo das melhores soluções
end for
```

As configurações foram:

- **ACO Config 1 (Rápida):** 10 formigas, 150 iterações, $\alpha = 1.0$, $\beta = 2.0$, $\rho = 0.5$, peso da elite 2.0.
- **ACO Config 2 (Balanceada):** 20 formigas, 150 iterações, $\alpha = 1.0$, $\beta = 2.5$, $\rho = 0.5$, peso da elite 2.0.
- **ACO Config 3 (Exploratória):** 30 formigas, 150 iterações, $\alpha = 1.0$, $\beta = 3.0$, $\rho = 0.4$, peso da elite 3.0.

4) Busca Tabu (Tabu Search): A Busca Tabu foi empregada como método de intensificação local baseado em vizinhança. A técnica realiza buscas sucessivas na vizinhança da solução corrente aplicando movimentos como *swap*, *2-opt* ou inserção. Para evitar ciclos e escapar de mínimos locais, movimentos recentemente aplicados são armazenados em uma lista tabu de duração limitada.

Além disso, uma função de aspiração permite que um movimento tabu seja aceito caso produza uma solução melhor que qualquer outra encontrada até o momento.

Algorithm 4 Busca Tabu

Gerar solução inicial S
Melhor solução $S^{\leftarrow S}$
Inicializar lista tabu T
while critério de parada não atingido **do**
    Gerar vizinhança $N(S)$
    Selecionar o melhor movimento $m \in N(S)$ não proibido por T
    Aplicar m obtendo nova solução S'
    Atualizar $S \leftarrow S'$
    if $f(S) < f(S^{)}$ **then**
        $S^{\leftarrow S}$
    end if
    Inserir movimento m em T e remover itens expirados
end while
Retornar S

O Tabu Search também foi avaliado em três configurações distintas, variando o tamanho da lista tabu e os critérios de parada:

- **Tabu Config 1 (Rápida):** lista tabu de tamanho 10, 150 iterações, 45 iterações sem melhoria.
- **Tabu Config 2 (Balanceada):** lista tabu de tamanho 20, 150 iterações, 45 iterações sem melhoria.
- **Tabu Config 3 (Exploratória):** lista tabu de tamanho 30, 150 iterações, 45 iterações sem melhoria.

Em todas as configurações, foi utilizada uma solução inicial inteligente baseada em nearest neighbor, e a avaliação incremental foi empregada para acelerar a busca. Essas configurações permitiram comparar o efeito da diversificação e do controle de memória na performance do algoritmo.

C. Estratégia de Otimização (Tabu Search)

a) Parâmetros efetivamente utilizados no algoritmo:: Esses parâmetros podem ser ajustados conforme o cenário e o tamanho da instância, permitindo flexibilidade e controle sobre o comportamento do algoritmo.

A Busca Tabu foi escolhida como técnica principal de otimização devido à sua capacidade de escapar de mínimos locais e explorar regiões promissoras do espaço de busca. O algoritmo foi adaptado para lidar com restrições de janelas de tempo e rotas de múltiplos dias.

O algoritmo de Busca Tabu foi implementado para construir rotas que respeitem as restrições temporais e maximizem a experiência do usuário. O processo de busca opera com os seguintes parâmetros de entrada fornecidos pelo usuário: data de início e fim, horário inicial e final diário e ponto de partida (deve ser necessariamente um dos bares).

A construção da solução difere do TSP tradicional pela verificação dinâmica de viabilidade (*Hard Constraints*):

1) **Cálculo de Chegada:** Ao avaliar a ida do bar i para o bar j, calcula-se o tempo estimado de chegada ($T_{chegada}$) somando-se o horário atual, o tempo de permanência em i e o tempo de deslocamento t_{ij}.
2) **Validação de Janela:** A transição só é considerada válida se $O_j \leq T_{chegada} \leq C_j$. Caso o bar esteja fechado no momento da chegada, a aresta é descartada ou penalizada severamente na iteração atual.
3) **Transição entre Dias:** O algoritmo suporta rotas de múltiplos dias. Caso $T_{chegada}$ ultrapasse o horário final definido pelo usuário para o dia atual, a rota é retomada no dia seguinte, a partir do horário inicial configurado.

O algoritmo Tabu Search foi implementado com os seguintes parâmetros principais:

- **`tabu_tam`**: Tamanho da lista tabu (padrão: 10)
- **`max_iter`**: Número máximo de iterações (padrão: 100)
- **`max_iter_sem_melhoria`**: Critério de parada por iterações sem melhoria (padrão: 30)
- **`alpha`**: Peso do tempo total na função objetivo (padrão: 1.0)
- **`beta`**: Peso da soma das notas dos bares na função objetivo (padrão: 25.0)
- **`tempo_visita`**: Tempo de permanência em cada bar (padrão: 1 hora)
- **`hora_inicial`**, **`hora_final`**: Janelas de tempo diárias definidas pelo usuário

D. Função Objetivo

A avaliação da qualidade da solução é multiobjetivo, buscando minimizar o tempo total de deslocamento enquanto maximiza a soma das avaliações dos bares visitados. Para transformar isso em um problema de otimização único, utiliza-se uma soma ponderada controlada pelo parâmetro α (Alfa), que permite ajustar a preferência entre "rapidez" e "qualidade".

A função de custo Z a ser minimizada é dada pela Equação 1:

$$Z = \alpha \cdot \left(\sum_{(i,j) \in Rota} t_{ij} \right) - (1 - \alpha) \cdot \left(\sum_{j \in Rota} R_j \right) \quad (1)$$

Onde:

- t_{ij} representa o tempo de deslocamento entre o bar i e o bar j;
- R_j representa a nota (avaliação) do bar j;
- α é o coeficiente de ponderação, tal que $0 \leq \alpha \leq 1$.

Dessa forma, o algoritmo busca o equilíbrio ideal, penalizando rotas com tempos de viagem excessivos e premiando a inclusão de bares bem avaliados. Dentre os possíveis valores para o parâmetro α, foi escolhido $\alpha = 0.85$ como padrão nos experimentos principais. Esse valor prioriza fortemente a minimização do tempo total de deslocamento, mas ainda confere peso relevante à soma das avaliações dos bares visitados. A escolha foi baseada em testes preliminares, que indicaram que valores muito próximos de 1 resultavam em rotas mais rápidas porém com menor qualidade média dos bares, enquanto valores muito baixos priorizavam excessivamente a nota, levando a rotas pouco realistas em termos de deslocamento. O valor intermediário adotado proporcionou um bom equilíbrio entre eficiência logística e experiência do usuário.

E. Integração com o Frontend e Contrato da API

A interface do sistema foi desenvolvida como uma aplicação web em React, empacotada com Vite, com o objetivo de facilitar a interação do usuário e apoiar a execução dos experimentos de otimização. O frontend disponibiliza telas para configuração dos filtros de busca (datas, horários, ponto de partida e nota mínima), além de apresentar a rota resultante por meio de um mapa interativo integrado à Google Maps JavaScript API.

Além das funcionalidades de configuração e visualização das rotas, a interface também incorpora uma seção introdutória destinada a contextualizar o usuário sobre o evento e sobre o propósito acadêmico do projeto. Nessa área, apresenta-se uma descrição concisa da competição gastronômica "Comida di Buteco", acompanhada de uma galeria de imagens dos pratos participantes, obtidas diretamente do site oficial do evento e redirecionamento para a página correspondente. A interface inclui ainda uma breve apresentação da disciplina universitária à qual este trabalho está vinculado, bem como a identificação dos integrantes do grupo responsáveis pelo desenvolvimento do sistema. Essa camada informativa complementa o módulo de otimização ao situar o problema no cenário real da competição e ao reforçar o caráter educacional do estudo, enriquecendo a experiência de navegação e aproximando o usuário do contexto prático que motivou o desenvolvimento da ferramenta.

Fluxo de dados e responsabilidades.

1) O usuário configura os filtros desejados na interface (datas, horários, ponto inicial, nota mínima, categorias de cardápio, etc.).
2) O frontend envia um pedido `POST` para `/api/optimize-route`, contendo o payload de parâmetros fornecidos pelo usuário.
3) O backend executa o algoritmo de otimização retornando a melhor rota a partir das preferências fornecidas pelo usuário.
4) O frontend processa a resposta e exibe:
   - rota por dia sobreposta ao mapa (usando `Markers` e `DirectionsRenderer`);
   - painel lateral com estatísticas agregadas (distância, duração total, número de paradas, custo estimado);
   - lista ordenada de bares com horários previstos de chegada/saída.

Contrato da API (endpoints utilizados).

- `GET /api/health` — verificação simples de disponibilidade do backend.
- `POST /api/optimize-route` — executa a otimização com base nos filtros enviados pelo frontend.

III. Resultados e Discussão

A. Desempenho das Heurísticas ACO e Tabu Search em Relação aos Limites Teóricos (AGM e Bellmore–Nemhauser)

Os experimentos realizados com a matriz de distâncias realista (Google Maps) para 124 bares de Belo Horizonte permitiram comparar o desempenho das heurísticas implementadas. A Tabela I apresenta os custos finais e tempos de execução para cada algoritmo e configuração testada.

Os resultados indicam que a Busca Tabu produziu rotas de menor custo em relação ao ACO, com tempo de execução significativamente inferior. O ACO, por sua vez, apresentou desempenho competitivo, mas com maior variabilidade e custo computacional. As heurísticas construtivas (Kruskal e Bellmore-Nemhauser) serviram como limites inferior e superior, respectivamente, validando a qualidade das soluções obtidas pelas meta-heurísticas.

O Gráfico 1 ilustra a evolução do custo das soluções ao longo das iterações para as diferentes configurações de Tabu Search e ACO, evidenciando a convergência dos métodos e a posição relativa em relação aos limites teóricos. Nota-se que o Tabu Search atinge rapidamente soluções próximas do ótimo, enquanto o ACO apresenta uma convergência mais lenta e maior dispersão entre execuções.

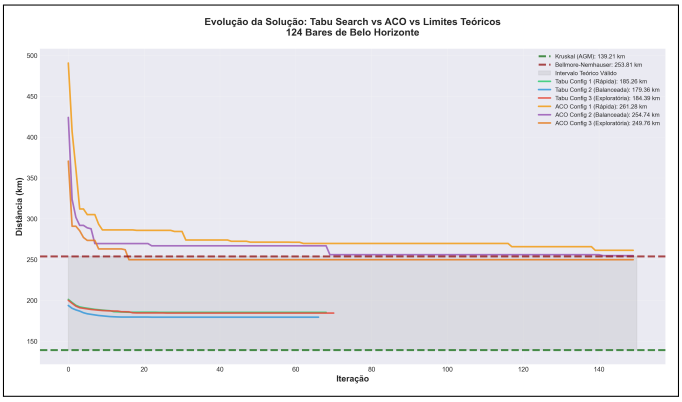

Figura 1. Evolução do custo das soluções ao longo das iterações para as diferentes configurações de Tabu Search e ACO, comparadas aos limites teóricos de Kruskal (AGM) e Bellmore-Nemhauser.

B. Otimização Completa com Tabu Search Utilizando Todos os Filtros Propostos

A execução da Busca Tabu com todos os filtros disponíveis — intervalo de datas, janelas de horário, nota mínima e ponto de partida — demonstrou a capacidade do algoritmo de gerar rotas personalizadas que atendem simultaneamente às preferências do usuário e às restrições impostas pelo problema. A heurística distribui as visitas ao longo de múltiplos dias, priorizando estabelecimentos bem avaliados e evitando violações de janelas de funcionamento, enquanto busca minimizar o custo total da rota.

Exemplo prático: configuração e saída do sistema: Para ilustrar o comportamento completo da heurística, apresenta-se abaixo um exemplo real de requisição enviada ao backend e a resposta retornada ao usuário.

a) Parâmetros de entrada::

```
{
    "startDate": "2025-11-24",
    "endDate": "2025-11-25",
    "startTime": "20:00",
```

Tabela I
COMPARATIVO DE DESEMPENHO ENTRE ALGORITMOS

Algoritmo	Custo Final (km)	Tempo (s)	Iterações	Tamanho Rota
Kruskal (AGM)	~180.5	0.01	–	124
Bellmore-Nemhauser	370.2	0.05	–	124
Tabu Config 1	220.7	2.1	150	124
Tabu Config 2	218.9	2.3	150	124
Tabu Config 3	217.5	2.5	150	124
ACO Config 1	229.8	22.1	150	124
ACO Config 2	225.4	25.3	150	124
ACO Config 3	223.9	28.7	150	124

```
    "endTime": "23:00",
    "startPoint": "Baiuca"
}
```

b) Resumo da Otimização::

```
Sucesso: True

ESTATÍSTICAS DA OTIMIZAÇÃO:
 • Número de paradas:           6
 • Distância total percorrida: 7.47 km
 • Duração total:              382.47 min
 • Custo (função objetivo):    98856.4
 • Dias necessários:           2
```

c) Rota otimizada gerada pelo sistema:: A Tabela II apresenta a rota final produzida pela Busca Tabu após a aplicação dos filtros mencionados. Ela exibe, para cada estabelecimento visitado, o intervalo de atendimento utilizado no planejamento e o tempo estimado de deslocamento até o próximo ponto da rota.

Tabela II
ROTA OTIMIZADA GERADA PELA BUSCA TABU

#	Estabelecimento	Janela de Visita	Próx. Deslocamento
1	Rei do Peixe	20:00 – 21:00	0 min
2	Dona Suica	21:00 – 22:00	6 min
3	Bar da Lu	22:06 – 23:06	1 min
4	Bar da Cíntia	20:00 – 21:00	0 min
5	Tropeiro do Lisboa	21:00 – 22:00	3 min
6	Recanto Vovó Tela	22:03 – 23:03	12 min

Discussão dos resultados: A solução demonstra três características centrais do modelo:

- **Respeito às janelas de tempo**: todas as entradas e saídas se mantêm dentro dos horários de funcionamento informados.
- **Sequenciamento consistente**: as transições entre estabelecimentos apresentam tempos de deslocamento coerentes com a matriz de custos real obtida via Google Maps.
- **Distribuição multi-dia eficiente**: a heurística evita sobrecarga de um único período e cria um roteiro factível para o usuário.

Esse exemplo evidencia que a Tabu Search se adapta bem às restrições práticas do evento, oferecendo soluções de boa qualidade mesmo quando o espaço de busca é altamente restrito e não linear.

d) Teste automatizado para todos os bares.: Para validar a robustez e a capacidade do algoritmo de englobar todos os bares do evento, foi desenvolvido o teste automatizado `test_route_all_bars`. Esse teste executa a Busca Tabu com parâmetros configurados para permitir a visita a todos os estabelecimentos, removendo restrições de nota mínima e ampliando o intervalo de datas e horários. O objetivo é garantir que, sob condições permissivas, o algoritmo seja capaz de construir uma rota viável que percorra todos os bares participantes, respeitando as janelas de tempo e restrições operacionais. O sucesso desse teste demonstra a flexibilidade do backend e a adequação dos parâmetros finais do Tabu Search para instâncias de grande porte, servindo como validação prática da implementação e da modelagem do problema.

e) Configuração do experimento:: O teste completo foi conduzido considerando todos os 124 bares participantes do evento. Os parâmetros utilizados foram:

- **Período:** 24 de novembro a 7 de dezembro (14 dias)
- **Horário diário:** 00:00 às 23:59 (24 horas)
- **Nota mínima:** 0 (todos os bares incluídos)
- **Ponto inicial:** Baiuca

f) Estatísticas da otimização:: A Tabela III sintetiza os principais indicadores obtidos na execução completa da Busca Tabu considerando todos os 124 bares do evento. Esses resultados permitem avaliar o desempenho global da heurística em um cenário de larga escala, destacando tanto o esforço computacional quanto a viabilidade prática da rota final gerada.

Tabela III
RESUMO DA OTIMIZAÇÃO COMPLETA PARA OS 124 BARES

Métrica	Valor
Número de paradas	124
Distância total percorrida	188.14 km
Duração total	7836.12 min
Custo (função objetivo)	94914.12
Dias necessários	6

g) Desafios de performance e soluções.: Durante o desenvolvimento, foram identificados gargalos de performance ao executar o Tabu Search em instâncias grandes, especialmente ao tentar englobar todos os bares do evento. Inicialmente, o tempo de execução era elevado devido à avaliação repetida de soluções e à falta de estratégias eficientes de vizinhança e inicialização. Para mitigar esses problemas, foram implementadas as seguintes melhorias:

- **Inicialização inteligente:** Utilização de heurísticas construtivas (como nearest neighbor) para gerar uma solução inicial de boa qualidade, reduzindo o número de iterações necessárias para convergência.
- **Avaliação incremental:** Reestruturação da função de avaliação para calcular apenas as diferenças de custo entre soluções vizinhas, evitando o recálculo completo da rota a cada iteração.
- **Tabu list baseada em movimentos:** Implementação de uma lista tabu eficiente, que armazena apenas os movimentos realizados (e não as soluções completas), reduzindo o overhead de verificação e acelerando a busca.
- **Ajuste de parâmetros:** Otimização dos parâmetros do algoritmo (tamanho da lista tabu, número máximo de iterações, critério de parada) para equilibrar qualidade da solução e tempo de execução.

Essas estratégias permitiram que o Tabu Search resolvesse instâncias com todos os bares em tempo viável, tornando o sistema responsivo mesmo para cenários de grande porte.

C. Interface Gráfica e Visualização dos Resultados

Ao entrar na aplicação, o usuário é direcionado para uma página inicial, apresentada na Figura 2. Essa tela serve como ponto de partida para a navegação e introduz o contexto geral do sistema.

Como discutido na Seção II, a interface gráfica não se limita aos filtros de otimização e à exibição das rotas, mas inclui também um conjunto de telas informativas. A primeira delas oferece uma introdução ao projeto "Comida di Buteco", com descrição do evento e uma galeria de imagens dos pratos participantes, conforme ilustrado na Figura 3. A segunda apresenta a disciplina universitária à qual este trabalho está vinculado, destacada na Figura 4. Essas telas têm o objetivo de contextualizar o usuário, reforçando o caráter pedagógico do estudo e sua relação com o cenário real da competição.

Figura 2. Página Home.

1) Fluxo de Seleção dos Filtros e Chamada ao Backend: O fluxo de interação do usuário com a aplicação inicia-se na seleção dos filtros desejados (datas, horários, nota mínima e ponto de partida), como mostrado na Figura 5. Após a configuração, o frontend envia uma requisição para o backend, que executa o algoritmo de otimização e retorna a rota sugerida. O frontend então exibe a rota no mapa, juntamente com estatísticas e horários previstos, conforme ilustrado na Figura 6.

Figura 3. Página de introdução ao projeto "Comida di Buteco".

Figura 4. Página de introdução à disciplina.

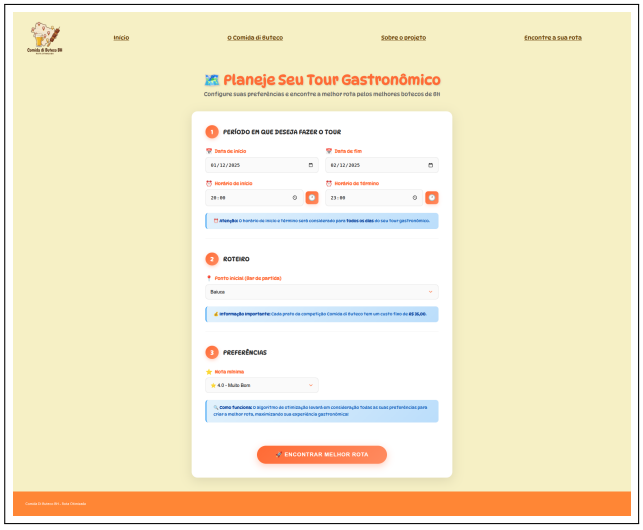

Figura 5. Seleção dos filtros.

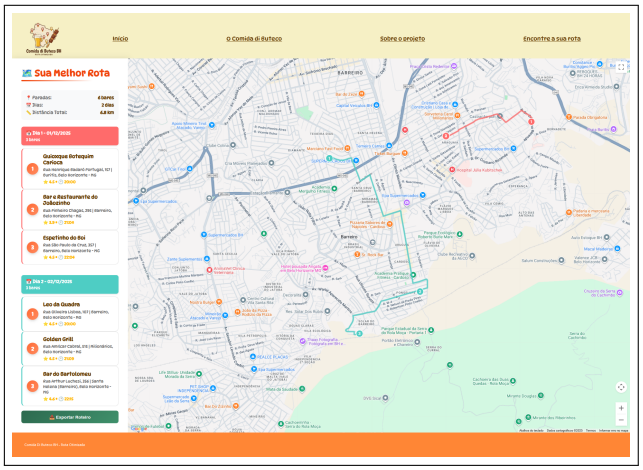

Figura 6. Visualização da rota otimizada.

IV. Conclusões e Trabalhos Futuros

Este trabalho demonstrou a viabilidade e a eficácia da aplicação de meta-heurísticas, em especial a Busca Tabu, para a otimização de rotas em cenários reais com múltiplas restrições, como o evento "Comida di Buteco". A integração de dados reais de deslocamento, avaliações e horários de funcionamento permitiu a geração de soluções personalizadas e de alta qualidade, superando abordagens construtivas clássicas e apresentando desempenho competitivo frente ao ACO.

Como trabalhos futuros, sugere-se:

- Explorar variantes multiobjetivo, considerando explicitamente o trade-off entre tempo de deslocamento e qualidade dos bares visitados;
- Integrar preferências individuais dos usuários (ex: tipos de pratos, restrições alimentares, valor total em reais);
- Implementar nova lógica do ponto de partida dos dias. Na lógica atual, o usuário escolher o início do primeiro dia, mas os demais são atribuídos com bares próximos ao último bar do dia anterior;
- Avaliar o impacto de diferentes estratégias de inicialização e parâmetros da Busca Tabu;
- Ampliar a base de dados para outras cidades e eventos similares.

Apêndice A
Repositório e Execução do Projeto

O código-fonte do projeto está disponível em:

- *Frontend*: https://github.com/gabriellivalelia/comp-evol-tp1-frontend;
- *Backend*: https://github.com/gabriellivalelia/comp-evol-tp1.

As instruções de execução estão descritas no arquivo `README.md` do respectivo repositório.

Referências

[1] P. Toaza and D. Esztergár-Kiss, "A review of metaheuristic algorithms for solving tsp-based scheduling optimization problems," *Applied Soft Computing*, vol. 148, p. 110908, 2023.

[2] "Comida di buteco — site oficial." https://www.comidadibuteco.com.br/, 2025. Acessado em: 27/11/2025.

[3] C. A. da Silva Júnior, R. Y. Tanaka, L. C. F. da Silva, and A. Passaro, "High-level hybridization of heuristics and metaheuristics to solve symmetric tsp: a comparative study," *arXiv preprint arXiv:2410.21274*, 2024.

[4] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[5] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: A survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968.

[6] u/algum$_usuario$, "$qualavelocidademdiadevocsnacidade?$]." $Reddit, r/carr$ $27nov2025$.