57º SBPO
SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
Gramado • RS
DE 5 A 9 DE OUTUBRO DE 2025

Aplicação de Modelo de Atenção para o Problema do Reparador Viajante *

Wendell Reis Milani Matias
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
R. José Péres, nº 558, Centro, Leopoldina - MG, Brasil
wendell@cefetmg.br

Henrique Barros Lopes
Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (IFMG)
R. Mato Grosso, nº 02, Vista Alegre, Ibirité - MG, Brasil
henrique.lopes@ifmg.edu.br

RESUMO

Neste artigo, abordamos o Problema do Reparador Viajante, cujo objetivo é minimizar o tempo de espera dos clientes ao determinar a melhor sequência de visitas. Trata-se de um problema de roteamento NP-difícil, com grande relevância nas áreas de logística e e-commerce, cuja solução heurística é amplamente utilizada, porém ainda possui um gargalo de tempo para resolver o problema. Soluções baseadas em *machine learning* vêm sendo aplicadas em problemas de roteamento de forma que, uma vez treinadas, o tempo para encontrar uma solução é menor do que em heurísticas. Para resolvê-lo, utilizamos um modelo de rede neural baseado em mecanismos de atenção, capaz de identificar padrões em dados sequenciais e gerar soluções de alta qualidade. Nossos modelos foram avaliados em instâncias padrão da literatura e em casos reais da TSPLIB, demonstrando a capacidade de encontrar soluções eficientes quase instantaneamente e generalizar bem para problemas de diferentes dimensões.

PALAVRAS CHAVE. Problema do Reparador Viajante, Modelo de Atenção, Rede Neural.

Tópicos (Inteligência Computacional, Otimização Combinatória)

ABSTRACT

In this article, we address the Traveling Repairman Problem, whose objective is to minimize customer waiting time by determining the best sequence of visits. It is an NP-hard routing problem, with great relevance in the fields of logistics and e-commerce, for which a heuristic solution is widely used, but still presents a time bottleneck to solve the problem. Machine learning-based solutions have been applied to routing problems so that, once trained, the time to find a solution is shorter than with heuristics. To solve it, we use a neural network model based on attention mechanisms, capable of identifying patterns in sequential data and generating high-quality solutions. Our models were evaluated on standard benchmark instances from the literature and on real cases from TSPLIB, demonstrating the ability to find efficient solutions almost instantly and to generalize well to problems of different sizes.

KEYWORDS. Travelling Repairman Problem, Graph Attention, Neural Network.

Paper topics (Computational Intelligence, Combinatory Optimization)

*Este trabalho teve o apoio institucional do IFMG e do CEFET-MG, com o financiamento do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brasil, por meio do processo nº 158949/2024-2.

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

1. Introdução

Problemas de roteamento de veículos possuem grande importância para a indústria, pois impactam o custo-benefício de sistemas logísticos [Bogyrbayeva et al., 2024]. Um dos problemas mais clássicos é o Problema do Caixeiro Viajante (TSP, do inglês *Travelling Salesman Problem*), cujo objetivo é encontrar o menor caminho que visita todas as cidades uma única vez e retorna à origem. Trata-se de um problema de otimização NP-difícil [Garey e Johnson, 1979], pois não possui algoritmo exato e complexidade de tempo polinomial. Assim, esses problemas têm recebido grande atenção de pesquisadores das mais diversas àreas, como Matemática, Pesquisa Operacional, Física, Biologia e Inteligência Artificial [Reinelt, 2003].

O Problema do Reparador Viajante (TRP, do inglês *Travelling Repairman Problem*), também conhecido como Problema de Mínima Latência (PML), é uma variante do TSP e é descrito por Blum et al. [1994]. No TRP, existe um reparador viajante que deseja visitar todos os seus clientes de forma a minimizar o tempo médio de espera (latência) que cada cliente aguarda antes de ser atendido. Assumimos que cada cliente é atendido de forma instantânea quando visitado. Trata-se de um problema de otimização combinatória que ganhou destaque nos últimos anos devido ao crescimento do e-commerce e principalmente durante a COVID-19, onde a maioria dos consumidores estava em suas casas. Nesse cenário, as empresas precisaram garantir a satisfação de seus clientes para obterem sucesso nos seus negócios, ou seja, minimizar a latência dos consumidores se tornou tão importante quanto minimizar o custo total dos entregadores [Uzun e Kara, 2021].

Mikula e Kulich [2022] destacam que a resolução do TRP segue duas principais abordagens. A primeira consiste em algoritmos exatos, capazes de encontrar soluções ótimas, mas restritos a instâncias pequenas devido ao alto custo computacional. A segunda, e mais utilizada, envolve algoritmos aproximados, incluindo heurísticas e metaheurísticas, que buscam soluções próximas do ótimo por meio de estratégias de busca. Esses métodos são preferidos para instâncias maiores, pois apresentam menor complexidade temporal.

Entre as abordagens exatas, Lucena [1990] e Bianco et al. [1993] utilizam formulações inteiras não lineares, nas quais a relaxação Lagrangiana é empregada para obter limites inferiores. Já entre as abordagens aproximadas, destacam-se a combinação do *Greedy Randomized Adaptive Search Procedure* (GRASP) com *Variable Neighborhood Search* (VNS) e *Variable Neighborhood Descent* (VND), proposta por Salehipour et al. [2011], além da heurística de busca local determinística paralela (PDLSH) de Yelmewad e Talawar [2021].

Por outro lado, Bogyrbayeva et al. [2024] evidencia que a utilização de estratégias de *Machine Learning* (ML) tem impulsionado avanços significativos na resolução de problemas de roteamento. Esses métodos exploram o poder computacional para aprender funções aproximadas que geram soluções próximas do ótimo. Além disso, os modelos baseados em ML possuem a capacidade de generalizar para novas instâncias de problemas com distribuições semelhantes, oferecendo soluções instantâneas a partir dos padrões aprendidos. Esses métodos são destacados por Kool et al. [2019], que introduziram o uso de modelos de *Deep Neural Networks* (DNNs) para parametrizar heurísticas através do aprendizado por reforço. Inspirados pela arquitetura *Transformer* de Vaswani et al. [2017], esses modelos aplicaram mecanismos atenção para capturar relações em dados sequenciais e gerar soluções eficientes para problemas como o TSP, evidenciando o potencial do aprendizado profundo na criação de heurísticas personalizadas.

Neste trabalho, abordamos o problema do TRP por meio de uma adaptação do modelo de redes neurais profundas (DNNs) proposto por Kool et al. [2019], preservando a metodologia de treinamento, o número de avaliações e os parâmetros dos modelos. A validação baseia-se na avaliação em instâncias de Salehipour et al. [2011] e da TSPLIB [Reinelt, 1991]. Por fim, apresentamos os resultados computacionais obtidos, que contribuem para o avanço da literatura na área. Até onde

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

sabemos, esta é a primeira aplicação de *Machine Learning* para resolver o TRP.

2. Descrição do Problema

Neste estudo, abordamos o TRP, onde um conjunto de clientes deve ser atendido por um reparador viajante. O objetivo é minimizar o tempo total de espera dos clientes, determinando um percurso que visite cada cliente exatamente uma vez [Salehipour et al., 2011]. Assume-se que o reparador inicia sua rota no vértice de índice zero e que o tempo de reparo em cada local é instantâneo após a visita.

Formalmente, temos um grafo completo e não orientado $G = (V, E)$, com $n+1$ vértices. O vértice v_0 representa a posição inicial do reparador, e os vértices $v_1, \ldots, v_n$ correspondem aos clientes. Cada aresta $(v_i, v_j) \in E$ possui um custo simétrico $w(v_i, v_j) = w(v_j, v_i)$, que representa o tempo de viagem entre os vértices. Além disso, cada aresta do percurso está associada a uma latência, a qual representa o tempo de espera dos clientes ainda não atendidos no momento em que a aresta é percorrida. Para modelar esse efeito, o custo da aresta é multiplicado pelo número de clientes restantes a serem atendidos, incorporando assim a latência ao custo total da viagem.

A solução (*tour*) $\pi = (\pi_0, \pi_1, \ldots, \pi_n)$ consiste em uma sequência de visita aos vértices, iniciando em $\pi_0 = v_0$, seguida por uma permutação dos n vértices clientes. Ou seja, os vértices $\pi_1, \ldots, \pi_n$ formam uma permutação dos clientes, satisfazendo $\pi_t \neq \pi_{t'}$ para todo $t \neq t'$, com $t, t' \in \{1, \ldots, n\}$.

A função objetivo do TRP, que corresponde à soma ponderada dos tempos de chegada com base na ordem de visita, é dada por:

$$f(\pi) = \sum_{t=1}^{n} (n - t + 1) \cdot w(\pi_{t-1}, \pi_t), \tag{1}$$

onde $w(\pi_{t-1}, \pi_t)$ representa o tempo de viagem entre dois vértices consecutivos.

3. Modelo de Atenção

A partir da definição formal do TRP, adaptamos o modelo de atenção de Kool et al. [2019], com modificações na função objetivo e no cálculo de probabilidades na primeira iteração, garantindo que as soluções sempre iniciem no vértice de índice zero. Essas alterações foram aplicadas sobre a implementação original do problema TSP.

Definimos uma instância s como um grafo completo com $n + 1$ nós, em que cada nó $i \in \{0, ..., n\}$ é representado pelo par ordenado x_i, correspondente à coordenada do vértice i. O vértice x_0 é fixado como ponto de partida para todas as soluções geradas.

O modelo adota uma política de probabilidades $p(\pi|s)$, que seleciona uma solução π para uma instância s. Essa política é parametrizada por θ e definida como:

$$p_\theta(\pi|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \pi_{1:t-1}) \tag{2}$$

Esse modelo é baseado em uma estrutura *encoder-decoder* [Hamilton, 2020] e envolve dois principais algoritmos: codificador (*encoder*) e o decodificador (*decoder*). Primeiramente, o codificador transforma um nó de um grafo em um vetor que codifica sua posição e vizinhança, suas conexões. Esse vetor resultante é chamado de *embedding*. Em seguida, o decodificador processa um *embedding* e reconstrói a estrutura local do grafo que ele representa. Os parâmetros θ correspondem aos pesos e matrizes de transformação aprendíveis envolvidos no *encoder*, *decoder* e na geração dos *embeddings*. Dito isso, o objetivo é encontrar uma política geral que funciona bem em diversas instâncias do problema.

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

3.1. Encoder

O codificador utilizado por Kool et al. [2019] é semelhante ao de Vaswani et al. [2017], porém sem a utilização de codificação posicional. Cada nó *i* é representado por um par ordenado $\mathbf{x}_i$, suas coordenadas espaciais. A partir disso o codificador calcula os *embeddings* iniciais $\mathbf{h}_i^{(0)} = W^{\mathbf{x}}\mathbf{x}_i + b^{\mathbf{x}}$ para cada nó, onde $d_h = 128$. Os parâmetros $W^{\mathbf{x}}$ e $b^{\mathbf{x}}$ correspondem, respectivamente, à uma matriz de projeção e ao viés (*bias*) utilizados para transformar a entrada em um vertor de *embedding*. Tais parâmetros são aprendidos durante o treinamento do codificador.

Esses *embeddings* passam por N camadas de atenção, cada uma composta por uma subcamada *Mult-Head Attention* (MHA) e uma *Feed-Forward* (FF). A cada camada, os *embeddings* são atualizados, gerando valores $\mathbf{h}_i^{(\ell)}$ para $\ell \in \{1, ..., N\}$. Ao final, o codificador calcula a média dos *embeddings* gerados conforme equação (3).

$$\mathbf{h}_i^{(N)} : \bar{\mathbf{h}}^{(N)} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{h}_i^{(N)} \tag{3}$$

De forma geral, a camada MHA realiza a passagem das mensagens entre os nós vizinhos, utilizando $M = 8$ cabeças, cada uma com $\frac{d_h}{M} = 16$ dimensões. Já a camada FF projeta os *embeddings* através de uma função de ativação ReLu e possui 512 dimensões. Além disso, essas camadas realizam um processo de normalização para acelerar o treinamento. As demais configurações da rede são as mesmas utilizadas por Kool et al. [2019].

3.2. Decoder

O decodificador acontece de forma sequencial em cada instante $t \in \{1, ..., n\}$ onde um nó π_t é selecionado com base na maior probabilidade. O cálculo das probabilidades de visitação de cada nó é feita com base em um nó de contexto *c*, que consiste em uma tríade com os *embeddings* do grafo ($\bar{\mathbf{h}}^{(N)}$), do primeiro e último nós visitados conforme descrito na equação (4).

$$\mathbf{h}_{(c)}^{(N)} = \begin{cases} \left[\bar{\mathbf{h}}^{(N)}, \mathbf{h}_{\pi_{t-1}}^{(N)}, \mathbf{h}_{\pi_1}^{(N)}\right] & t > 1 \\ \left[\bar{\mathbf{h}}^{(N)}, \mathbf{v}^1, \mathbf{v}^f\right] & t = 1. \end{cases} \tag{4}$$

Aqui, $\mathbf{v}^1$ e $\mathbf{v}^f$ são vetores aprendidos pelo modelo e utilizados como *placeholders* quando $t = 1$, uma vez que ainda não há um nó anterior definido na sequência.

3.3. Mecanismo de Atenção MHA

O mecanismo MHA é responsável por calcular as probabilidades de escolha dos próximos nós e utiliza os vetores da equação (5) para isso.

$$\mathbf{q}_{(c)} = W^Q\mathbf{h}_{(c)}, \quad \mathbf{k}_i = W^K\mathbf{h}_i, \quad \mathbf{v}_i = W^V\mathbf{h}_i. \tag{5}$$

Em que $\mathbf{q}_{(c)}$ representa uma consulta com informações do contexto para selecionar mais nós para o *tour*. Cada nó possui uma chave $\mathbf{k}_i$ e um valor $\mathbf{v}_i$ que são utilizados para medir sua compatibilidade, que indica o quão preferível é um nó para uma consula. Os termos W^Q,W^K e W^V são matrizes de pesos que são aprendidas pelo mecanismo e que transformam *embeddings* em uma consulta, chave ou valor. A partir disso é feito o cálculo da equação (6).

$$u_{(c)j} = \begin{cases} C \cdot \tanh\left(\frac{q_{i0}^T k_j}{\sqrt{d_i}}\right) & \text{se } j \neq \pi_l \quad \forall t' < t \\ -\infty & \text{caso contrário.} \end{cases} \tag{6}$$

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

A equação (6) representa a compatibilidade de um nó candidato *j* para uma determinada consulta. Os valores numéricos são estabilizados pela tangente hiperbólica e são restritos ao intervalo $[-C, C]$, com $C = 10$. Nesse cálculo os nós que já foram visitados são descartados por meio da atribuição de $-\infty$. Por fim, as probabilidades finais são calculadas da conforme equação (7).

$$p_i = p_\theta(\pi_t = i|s, \pi_{1:t-1}) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}} \tag{7}$$

3.4. Reforço e Baseline Rollout

O aprendizado por reforço é um método utilizado para resolver problemas modelados como Processos de Decisão de Markov (MDP), nos quais um agente interage com um ambiente tomando ações para alcançar um objetivo [Bogyrbayeva et al., 2024]. No caso do TRP, o objetivo é aprender os parâmetros θ da política, definida pela equação (2), por meio da otimização da função objetivo utilizando gradiente descendente. Esse aprendizado depende da correta atribuição de pesos às ações a serem tomadas, o que é realizado por meio da avaliação de múltiplas amostras. No entanto, como a recompensa de cada episódio pode variar significativamente devido à aleatoriedade das escolhas e à dinâmica do ambiente, os gradientes estimados apresentam alta variabilidade, o que pode dificultar a convergência do modelo [Wu et al., 2018].

Para mitigar esse problema, Wu et al. [2018] sugerem o uso de *baselines*, que funcionam como referências para estabilizar as atualizações dos parâmetros. A ideia é substituir a recompensa bruta por uma diferença entre a recompensa obtida e um valor esperado fornecido pela *baseline*, o que reduz a variância sem introduzir viés na estimativa do gradiente.

Para este trabalho, foi utilizada a *rollout baseline* proposta por Kool et al. [2019], que avalia as soluções geradas pelo modelo atual comparando-as com aquelas obtidas por uma política de referência, definida em uma versão anterior do modelo. A diferença entre os custos das soluções é utilizada para atualizar os parâmetros da política, tornando o aprendizado mais estável e eficiente. Dessa forma, a função de perda, que é a expectativa do custo $L(\pi)$ de um *tour*, é definida pela equação (8). A atualização dos parâmetros é feita com o estimador de gradiente com *baseline* $b(s)$, dado pela equação (9).

$$\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[L(\pi)] \tag{8}$$

$$\nabla\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}\left[(L(\pi) - b(s))\nabla \log p_\theta(\pi|s)\right] \tag{9}$$

As atualizações de θ são conduzidas utilizando o algoritmo Adam [Kingma e Ba, 2015], que ajusta os pesos com base no gradiente $\nabla\mathcal{L}$ e ocorre apenas quando os novos valores resultam em melhorias estatisticamente significativas em relação aos valores previamente armazenados na *baseline*. Essas melhorias são verificadas com base no desempenho em um conjunto de instâncias de validação. Sempre que uma nova versão de θ apresenta desempenho significativamente superior, ela substitui a versão armazenada. Além disso, realiza-se uma avaliação adicional em instâncias de teste a cada atualização para mitigar o risco de *overfitting*.

4. Resultados Computacionais

O modelo de atenção para o TRP é uma adaptação do modelo proposto por Kool et al. [2019], implementado em Python com a biblioteca PyTorch 2.5.1 e plataforma CUDA 12.1. As principais modificações incluem a alteração da função objetivo e a obrigatoriedade de selecionar o nó de índice zero como posição inicial. Os experimentos foram realizados em um computador com processador Intel Core i5-13450HX (4.60GHz), GeForce RTX 3050 Mobile com 6GB de VRAM, 16 GB de RAM DDR5 (4800MHz), rodando Windows 11.

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

O treinamento dos modelos foi conduzido com base em diversos parâmetros, sendo um dos principais o tamanho das instâncias de treinamento, que é o número de nós presentes em cada grafo. Com base nos tamanhos utilizados por Salehipour et al. [2011], foram gerados modelos com 11, 21, 51 e 101 nós, que são os vértices dos clientes mais o vértice do reparador. A configuração do treinamento seguiu os mesmos parâmetros adotados por Kool et al. [2019], com taxa de aprendizado $\eta = 10^{-4}$ e $N = 3$ camadas no encoder.

Devido a limitações de memória da GPU utilizada, o processo foi particionado em várias sessões de treinamento consecutivas, em que o modelo final de cada sessão serviu como ponto de partida para a seguinte. Ao final do processo, todos os modelos completaram um total de 128.000.000 avaliações, conforme estabelecido em Kool et al. [2019].

Outros parâmetros utilizados nos modelos e o tempo médio gasto no treinamento de cada *epoch* estão apresentados na Tabela 1.

Modelo	Tamanho	Epochs	Quantidade de Epochs	Tamanho do Dataset de Validação	Tempo Médio por Epoch (s)
TRP11	11	128,000	1,000	10,000	48
TRP21	21	128,000	1,000	10,000	58
TRP51	51	128,000	1,000	10,000	156
TRP101	101	64,000	2,000	10,000	408

Tabela 1: Parâmetros Gerais dos Modelos Treinados

O processo de avaliação de uma instância foi feito com dois métodos de decodificação: *greedy* e *sampling*. O método *greedy* seleciona o nó com maior probabilidade, resultando em uma solução gulosa. O método *sampling* utiliza uma amostragem estocástica para gerar 1280 soluções, quantidade estabelecida em Kool et al. [2019], podendo o número de amostras ser aumentado. Dentre as soluções geradas, a de menor custo é selecionada. Para cada método, são apresentados o tempo médio de avaliação ($\bar{t}$) e o *gap* de otimalidade para cada problema, conforme a equação (10).

$$Gap(\%) = \left| \frac{MelhorValor_{\text{modelo}} - MelhorValor_{\text{literatura}}}{MelhorValor_{\text{literatura}}} \right| \times 100 \tag{10}$$

Nas Tabelas 2, 3, 4, 5 e 6, comparamos os dois métodos de decodificação avaliando o desempenho dos modelos nas instâncias propostas por Salehipour et al. [2011]. Os melhores valores de *gap* de otimalidade para cada instância estão destacados em negrito. Os valores ótimos e melhores conhecidos podem ser encontrados em Silva et al. [2012].

Analisando os resultados instâncias aleatórias de Salehipour et al. [2011], é possível perceber que os modelos obtêm melhor desempenho em problemas cuja dimensão são próximas àquela usada durante o treinamento. Em particular, observa-se que o modelo treinado com instâncias de determinado tamanho obteve os menores *gaps* quando avaliado com ambos os métodos de decodificação nas instâncias do mesmo tamanho. A única exceção ocorre nas instâncias de tamanho 10, em que o modelo TRP11 com decodificação *sampling* apresentou um *gap* 0,12% maior que o modelo TRP21 com o mesmo método.

Além disso, é evidente que a avaliação com decodificação por *sampling* tende a encontrar as melhores soluções, apresentando os menores *gaps* médios para problemas com até 100 nós. Esses resultados podem ser ainda mais aprimorados com o aumento do tamanho da amostragem. No entanto, isso também provoca maior tempo de execução e uso de memória, podendo inviabilizar a avaliação de instâncias maiores devido às limitações da GPU.

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

Instância	Ótimo	TRP11		TRP21		TRP51		TRP101	
		Greedy	Sampling	Greedy	Sampling	Greedy	Sampling	Greedy	Sampling
S10-R1	1,303.00	**24.36**	**24.36**	**24.36**	**24.36**	35.62	35.62	**24.36**	**24.36**
S10-R2	1,517.00	**22.58**	**22.58**	**22.58**	**22.58**	**22.58**	**22.58**	36.71	36.71
S10-R3	1,233.00	**26.94**	**26.94**	27.12	**26.94**	61.51	50.80	27.12	**26.94**
S10-R4	1,386.00	**21.99**	**21.99**	**21.99**	**21.99**	23.74	23.74	39.24	34.14
S10-R5	978.00	**30.30**	**30.30**	**30.30**	**30.30**	46.24	40.27	36.00	36.00
S10-R6	1,477.00	**25.01**	**25.01**	**25.01**	**25.01**	31.69	**25.01**	59.42	40.06
S10-R7	1,163.00	**30.45**	**30.45**	**30.45**	**30.45**	**30.45**	**30.45**	54.56	54.56
S10-R8	1,234.00	**27.55**	**27.55**	30.54	**27.55**	**27.55**	**27.55**	54.00	30.54
S10-R9	1,402.00	24.65	24.65	**24.23**	**24.23**	24.65	24.65	**24.23**	**24.23**
S10-R10	1,388.00	**25.02**	**25.02**	**25.02**	**25.02**	28.68	28.68	**25.02**	**25.02**
S10-R11	1,405.00	25.87	**25.49**	25.80	**25.49**	28.73	28.73	25.87	25.87
S10-R12	1,150.00	**27.48**	**27.48**	**27.48**	**27.48**	**27.48**	**27.48**	34.56	34.56
S10-R13	1,531.00	28.39	28.39	30.33	28.39	36.23	**28.12**	36.76	36.76
S10-R14	1,219.00	**25.48**	**25.48**	**25.48**	**25.48**	36.03	36.03	**25.48**	**25.48**
S10-R15	1,087.00	**30.02**	**30.02**	**30.02**	**30.02**	48.52	48.18	47.54	36.90
S10-R16	1,264.00	36.97	**30.90**	36.97	**30.90**	33.37	**30.90**	50.33	38.66
S10-R17	1,058.00	**27.84**	**27.84**	**27.84**	**27.84**	**27.84**	**27.84**	67.31	67.31
S10-R18	1,083.00	**24.78**	**24.78**	30.75	**24.78**	31.59	29.19	36.63	36.63
S10-R19	1,394.00	28.90	28.90	28.90	**28.23**	28.90	**28.23**	66.70	38.57
S10-R20	951.00	34.14	33.93	33.93	**32.60**	34.14	**32.60**	65.81	46.40
$\overline{Gap}$ (%)	-	27.44	27.10	27.96	**26.98**	33.28	31.33	41.88	35.98
$\bar{t}$ (s)	-	0.00	0.03	0.00	0.04	0.00	0.03	0.00	0.04

Tabela 2: Resultados dos modelos nas instâncias de tamanho 10 de Salehipour et al. [2011].

Os modelos também avaliaram as instâncias da TSPLIB selecionadas por Salehipour et al. [2011], conforme mostrado na Tabela 3. Nesse cenário, os modelos de tamanho 51 e 101 apresentaram melhor desempenho, sendo responsáveis pelos melhores resultados na maioria dos problemas, com exceção da instância pr107. O fato de o modelo TRP101 ter produzido um valor ruim nesse problema com 107 nós sugere que o modelo ainda precisa passar por mais sessões de treinamento. Além disso, a Tabela 3 mostra que, mesmo para instâncias com 532 nós, os modelos conseguem realizar a avaliação de forma instantânea com decodificação gulosa.

Instância	Ótimo	TRP11		TRP21		TRP51		TRP101	
		Greedy	Sampling	Greedy	Sampling	Greedy	Sampling	Greedy	Sampling
st70	19,215.00	30.85	35.32	24.06	21.46	22.47	16.45	16.56	**14.41**
rat99	54,984.00	33.18	62.43	21.07	32.15	22.29	**15.89**	28.52	23.02
kroD100	949,594.00	45.38	63.36	30.93	35.54	13.88	**13.82**	20.31	14.15
lin105	585,823.00	43.35	87.85	33.38	40.79	25.07	22.26	29.34	**19.79**
pr107	1,980,767.00	30.82	70.58	**14.43**	33.02	21.64	24.51	43.80	80.02
rat195	210,191.00	31.67	111.45	32.45	72.08	**20.35**	33.42	26.28	29.45
pr226	7,100,308.00	61.17	202.41	31.50	103.61	23.99	53.55	**15.31**	31.37
lin318	5,560,679.00	65.03	218.02	37.36	132.87	30.42	55.60	**23.66**	34.86
pr439	17,688,561.00	57.79	257.60	50.13	139.44	**38.82**	91.44	59.47	116.97
att532	5,581,240.00	450.12	1,044.90	361.68	818.81	**325.68**	566.95	361.41	485.19
$\overline{Gap}$ (%)	-	84.94	215.39	63.70	142.98	**54.46**	89.39	62.47	84.92
$\bar{t}$ (s)	-	0.00	10.40	0.00	10.08	0.00	10.20	0.00	10.21

Tabela 3: Resultados dos modelos no conjunto de instâncias da TSPLIB de Salehipour et al. [2011].

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

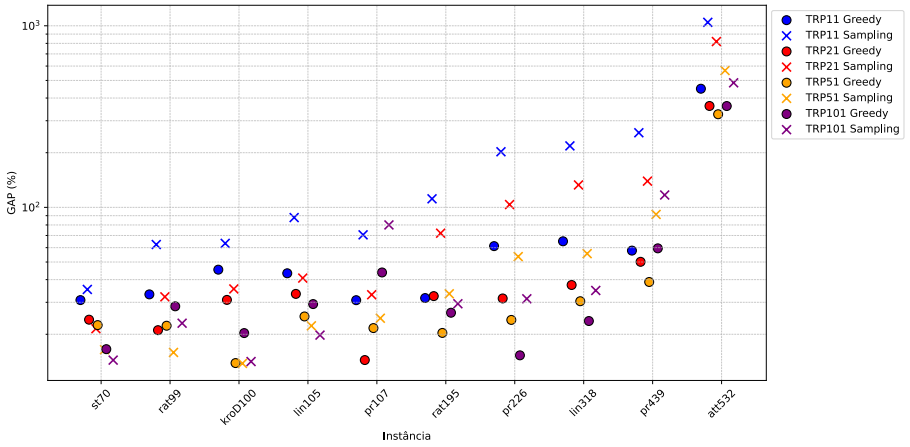


Figura 1: Desempenho dos modelos nas instâncias da TSPLIB

As Figuras 1 e 2 ilustram graficamente os *gaps* dos modelos em todas as instâncias avaliadas. A Figura 1 está representada em escala logarítmica para facilitar a visualização. Observa-se que, conforme o tamanho do problema supera o tamanho do modelo treinado, o desempenho com a decodificação *sampling* tende a ser inferior à solução gulosa. Por outro lado, quando as instâncias possuem tamanho próximo ao do modelo, a decodificação por *sampling* frequentemente supera a abordagem gulosa. Tanto essa degradação quanto a superioridade do *sampling* em instâncias de tamanho semelhante também foram observadas também em Kool et al. [2019].

Instância	**Ótimo**	**TRP11**		**TRP21**		**TRP51**		**TRP101**	
		Greedy	**Sampling**	**Greedy**	**Sampling**	**Greedy**	**Sampling**	**Greedy**	**Sampling**
S20-R1	3,175.00	15.34	**13.91**	14.00	**13.91**	15.74	14.00	34.93	27.26
S20-R2	3,248.00	16.97	**15.60**	20.49	**15.60**	27.87	16.40	19.15	17.70
S20-R3	3,570.00	16.92	**13.76**	**13.76**	**13.76**	30.66	18.86	14.75	**13.76**
S20-R4	2,983.00	23.91	**17.03**	**17.03**	**17.03**	18.59	**17.03**	32.65	25.13
S20-R5	3,248.00	15.74	**14.79**	18.65	15.48	24.02	19.31	26.48	15.61
S20-R6	3,328.00	17.68	**16.40**	18.24	16.52	18.32	16.94	18.74	18.74
S20-R7	2,809.00	24.10	20.17	20.85	**19.14**	23.93	22.44	20.38	20.17
S20-R8	3,461.00	15.88	**15.39**	16.98	15.94	19.99	18.52	21.11	17.41
S20-R9	3,475.00	19.86	19.39	19.39	**17.47**	36.10	19.70	21.35	20.95
S20-R10	3,359.00	28.17	20.94	23.51	20.43	21.12	20.94	45.47	**19.93**
S20-R11	2,916.00	16.54	14.86	16.09	**14.66**	16.49	**14.66**	39.28	30.53
S20-R12	3,314.00	**14.21**	**14.21**	**14.21**	**14.21**	18.92	18.92	36.85	32.26
S20-R13	3,412.00	16.43	**14.67**	16.43	**14.67**	34.13	23.63	25.28	19.68
S20-R14	3,297.00	15.97	**15.95**	19.12	**15.95**	24.68	20.75	28.41	28.41
S20-R15	2,862.00	16.50	14.49	15.86	**13.94**	16.03	**13.94**	24.41	23.89
S20-R16	3,433.00	18.67	18.10	18.17	**15.69**	**15.69**	**15.69**	33.81	27.77
S20-R17	2,913.00	21.07	17.37	22.61	**17.12**	31.60	24.61	24.25	23.63
S20-R18	3,124.00	32.21	**15.58**	19.68	**15.58**	19.68	19.68	30.98	**15.58**
S20-R19	3,299.00	16.21	**15.63**	25.36	**15.63**	**15.63**	**15.63**	27.33	25.94
S20-R20	2,796.00	**17.32**	**17.32**	17.32	**17.32**	33.64	33.64	**17.32**	**17.32**
$\overline{Gap}$ **(%)**	-	18.99	16.28	18.39	**16.00**	23.14	19.27	27.15	22.08
$\bar{t}$ **(s)**	-	0.00	0.06	0.00	0.06	0.00	0.06	0.00	0.06

Tabela 4: Resultados dos modelos nas instâncias de tamanho 20 de Salehipour et al. [2011].

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

Instância	Ótimo	TRP11		TRP21		TRP51		TRP101	
		Greedy	Sampling	Greedy	Sampling	Greedy	Sampling	Greedy	Sampling
S50-R1	12,198.00	28.72	34.61	19.07	15.37	16.74	14.88	13.58	**11.08**
S50-R2	11,621.00	42.17	29.17	19.21	**12.38**	17.74	13.27	19.42	16.66
S50-R3	12,139.00	29.28	31.72	18.30	13.77	17.38	13.20	17.07	**10.18**
S50-R4	13,071.00	24.40	26.29	23.23	18.25	14.36	13.08	14.81	**12.83**
S50-R5	12,126.00	33.63	34.65	16.21	14.68	16.08	13.61	11.90	**9.40**
S50-R6	12,684.00	26.27	28.10	21.41	14.85	13.86	12.26	22.54	**11.15**
S50-R7	11,176.00	26.08	31.05	14.19	**11.64**	19.27	14.42	19.68	12.40
S50-R8	12,910.00	23.36	30.51	14.89	13.80	14.26	13.18	12.43	**11.21**
S50-R9	13,149.00	25.47	25.56	17.53	12.28	10.48	**9.62**	15.82	14.23
S50-R10	12,892.00	28.19	28.86	21.06	14.70	13.10	**10.78**	13.87	12.35
S50-R11	12,103.00	24.80	29.40	14.83	13.45	12.91	**10.89**	14.14	11.89
S50-R12	10,633.00	48.63	37.85	24.55	18.97	11.74	**11.10**	22.33	17.23
S50-R13	12,115.00	31.31	25.92	22.18	15.09	14.01	**11.89**	16.79	14.72
S50-R14	13,117.00	21.35	21.14	12.01	11.08	12.09	**7.90**	17.52	10.97
S50-R15	11,986.00	22.40	34.42	16.13	15.06	14.25	**9.82**	20.82	14.73
S50-R16	12,138.00	24.93	27.37	14.37	11.45	15.57	**10.54**	14.21	13.10
S50-R17	12,176.00	26.79	26.79	15.71	14.37	22.59	16.12	14.48	**12.18**
S50-R18	13,357.00	31.96	32.79	18.19	13.18	15.82	**11.13**	22.48	15.79
S50-R19	11,430.00	32.97	31.28	19.54	15.37	14.98	13.72	14.27	**11.08**
S50-R20	11,935.00	33.34	36.32	23.89	17.06	13.46	**10.98**	13.18	11.84
$\overline{Gap}$ **(%)**	-	29.30	30.19	18.33	14.34	15.03	**12.12**	16.57	12.75
$\bar{t}$ **(s)**	-	0.00	0.17	0.00	0.16	0.00	0.15	0.00	0.17

Tabela 5: Resultados dos modelos nas instâncias de tamanho 50 de Salehipour et al. [2011].

Ao comparar os resultados obtidos neste trabalho com os relatados por Kool et al. [2019], observa-se que os modelos propostos por esses autores alcançaram *gaps* de 0,08% para instâncias do TSP com 20 nós e até 2,26% para aquelas com 100 nós. Em instâncias de mesmo tamanho, nossos modelos obtiveram médias de até 16% (TRP21 com *sampling*) e 12,52% (TRP101 com *sampling*) respectivamente.

Apesar de ambos os trabalhos utilizarem as mesmas estratégias baseadas em aprendizado, a diferença significativa observada pode ser atribuída à natureza mais complexa do TRP. Segundo Salehipour et al. [2011], o TRP é consideravelmente mais difícil de resolver ou aproximar em comparação ao TSP. Nessa mesma linha, Blum et al. [1994] ressalta que pequenas alterações na estrutura do grafo podem levar a grandes mudanças nas decisões a serem tomadas. Como consequência, abordagens comuns, como a decomposição do grafo em componentes biconexas para dividir um problema grande em subproblemas menores, não são eficazes no TRP.

Essas características sugerem que os padrões presentes no TRP são mais complexos do que os do TSP, exigindo que os modelos para TRP passem por mais sessões de treinamento. Esse fator pode justificar a diferença de desempenho observada entre os resultados obtidos neste trabalho e os de Kool et al. [2019].

A Tabela 7 apresenta uma comparação dos tempos médios de execução dos modelos propostos com quatro das meta-heurísticas reconhecidamente eficientes (GILS-RVND e MS-GVNS em Mikula e Kulich [2022], bem como DHLSH e PDLSH em Yelmewad e Talawar [2021]), responsáveis por resultados de alta qualidade em diversos *benchmarks* do TRP. Para as instâncias geradas por Salehipour et al. [2011], comparamos os tempos médios desses algoritmos e a média de execução de nossos modelos. Todos os tempos estão expressos em segundos e os melhores valores para cada instância aparecem em negrito.

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

Instância	Ótimo	TRP11		TRP21		TRP51		TRP101	
		Greedy	Sampling	Greedy	Sampling	Greedy	Sampling	Greedy	Sampling
S100-R1	32,779.00	42.40	67.12	36.61	38.77	17.17	**13.93**	17.12	14.90
S100-R2	33,435.00	55.40	71.12	27.50	37.64	15.53	14.03	13.68	**10.72**
S100-R3	32,390.00	33.45	70.58	32.78	37.90	13.05	12.72	11.26	**10.02**
S100-R4	34,733.00	38.19	63.71	19.41	36.38	12.91	**10.88**	16.20	12.60
S100-R5	32,598.00	41.42	62.76	30.67	39.10	16.21	14.23	14.19	**10.52**
S100-R6	34,159.00	24.23	63.69	31.88	37.65	14.48	13.53	16.61	**12.81**
S100-R7	33,375.00	37.85	71.04	24.98	37.23	20.55	16.44	14.51	**11.89**
S100-R8	31,780.00	53.55	75.04	34.88	39.98	25.68	**14.95**	18.40	14.95
S100-R9	34,167.00	37.54	66.12	26.06	40.44	12.69	12.29	12.76	**11.06**
S100-R10	31,605.00	28.54	70.77	22.33	38.94	16.19	**12.57**	14.62	13.38
S100-R11	34,188.00	37.24	60.78	26.78	37.07	15.56	**10.73**	17.80	12.75
S100-R12	32,146.00	35.18	59.00	26.32	32.71	20.17	15.47	15.09	**12.34**
S100-R13	32,604.00	52.29	76.16	30.45	34.79	14.61	13.11	14.92	**12.25**
S100-R14	32,433.00	52.04	66.13	42.89	40.14	14.62	**13.35**	25.92	17.98
S100-R15	32,574.00	43.36	65.39	27.39	38.23	12.51	**11.62**	17.69	11.66
S100-R16	33,566.00	42.60	60.52	30.02	36.62	**10.89**	11.21	18.20	13.59
S100-R17	34,198.00	43.83	55.62	30.30	37.29	20.88	16.28	9.80	**8.74**
S100-R18	31,929.00	46.46	66.05	26.07	38.92	18.56	**13.87**	18.61	14.27
S100-R19	33,463.00	47.92	71.42	24.71	38.21	19.79	15.14	13.46	**10.96**
S100-R20	33,632.00	33.40	71.45	27.21	37.17	18.28	13.13	18.12	**12.94**
$\overline{Gap}$ **(%)**	-	41.34	66.72	28.96	37.76	16.52	13.47	15.95	**12.52**
$\bar{t}$ **(s)**	-	0.00	0.53	0.00	0.53	0.00	0.48	0.00	0.50

Tabela 6: Resultados dos modelos nas instâncias de tamanho 100 de Salehipour et al. [2011].

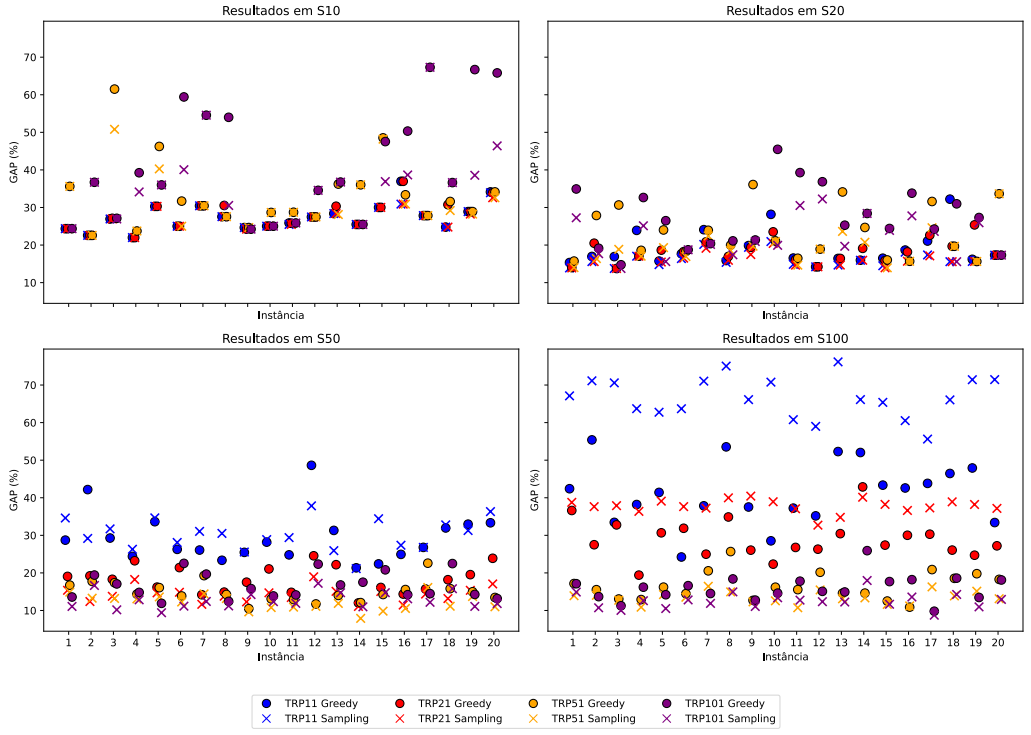


Figura 2: Desempenho dos Modelos nas instâncias de geradas por Salehipour et al. [2011].

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

Ainda, a Tabela 7 evidencia que mesmo em instâncias maiores, a decodificação gulosa apresenta tempo de execução de 0,00 s, evidenciando a capacidade dos modelos de generalizar e responder em tempo real. Esses resultados confirmam que nossos modelos aprenderam, ainda que parcialmente, a resolver o TRP e se mostram promissores para aplicações em que a latência de resposta é um fator crítico.

Instância	GILS-RVND	MS-GVNS	DLSH	PDLSH	Modelos TRP Sampling	Modelos TRP Greedy
S10	**0.00**	**0.00**	-	-	0.04	**0.00**
S20	0.01	**0.00**	-	-	0.06	**0.00**
S50	0.26	0.01	-	-	0.16	**0.00**
S100	3.30	0.05	-	-	0.51	**0.00**
st70	0.80	0.30	0.02	0.05	0.56	**0.00**
rat99	3.90	1.50	0.06	0.05	0.47	**0.00**
kroD100	2.90	1.10	0.06	0.05	0.51	**0.00**
lin105	2.70	1.10	0.07	0.05	0.54	**0.00**
pr107	3.30	1.40	0.03	0.05	0.55	**0.00**
rat195	28.90	12.50	0.24	0.06	1.60	**0.00**
pr226	23.70	26.60	2.19	0.09	2.07	**0.00**
lin318	95.20	96.20	4.15	0.11	3.96	**0.00**
pr439	245.80	301.50	7.90	0.14	7.99	**0.00**
att532	707.20	1,162.60	20.77	0.38	83.97	**0.00**

Tabela 7: Tabela comparativa de tempos médios de execução entre algoritmos.

5. Conclusão

Neste trabalho, propomos modelos estocásticos baseados em mecanismos de atenção, capazes de aprender padrões estruturais em grafos e gerar soluções de alta qualidade para o Problema do Reparador Viajante (TRP). A arquitetura de atenção e o procedimento de treinamento adotados seguem a abordagem proposta por Kool et al. [2019].

Os resultados empíricos indicam que os modelos desenvolvidos obtêm soluções com até 7,9% de diferença em relação ao ótimo, com tempos computacionais inferiores aos reportados por algoritmos tradicionais da literatura, o que evidencia sua capacidade de generalização. Além disso, os treinamentos apresentaram boa convergência, revelando que, à medida que o processo de aprendizado avança, as soluções tornam-se mais próximas do ótimo. Entretanto, mostra-se importante realizar novos treinamentos a partir dos modelos obtidos, bem como avaliá-los em instâncias utilizadas por outros trabalhos, como as apresentadas em Abeledo et al. [2010].

Por fim, pretendemos aprofundar nossas pesquisas em métodos para acelerar o aprendizado para gerar modelos capazes de encontrar melhores rotas em tamanhos de grafo mais variados. Além disso, visamos integrar algoritmos de *Branch and Bound* com as políticas probabilísticas aprendidas pelos modelos para explorar suas estimativas na construção de soluções ótimas.

Referências

Abeledo, H., Fukasawa, R., Pessoa, A., e Uchoa, E. (2010). The time dependent traveling salesman problem: Polyhedra and branch-cut-and-price algorithm. In *Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings 9*, p. 202–213. Springer.

Bianco, L., Mingozzi, A., e Ricciardelli, S. (1993). The traveling salesman problem with cumulative costs. *Networks*, 23(2):81–91.

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br

Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., e Sudan, M. (1994). The minimum latency problem. In *Proceedings of the 26th ACM Symposium on the Theory of Computing (STOC)*, p. 163–171, Montreal, Canada. ACM.

Bogyrbayeva, A., Meraliyev, M., Mustakhov, T., e Dauletbayev, B. (2024). Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W.H. Freeman and Company.

Hamilton, W. L. (2020). *Graph Representation Learning*. Morgan & Claypool Publishers.

Kingma, D. P. e Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. URL https://arxiv.org/abs/1412.6980.

Kool, W., van Hoof, H., e Welling, M. (2019). Attention, learn to solve routing problems! In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=ByxBFsRqYm.

Lucena, A. (1990). Time-dependent traveling salesman problemâthe deliveryman case. *Networks*, 20(6):753–763.

Mikula, J. e Kulich, M. (2022). Solving the traveling delivery person problem with limited computational time. *Central European Journal of Operations Research*, 30:1451–1481.

Reinelt, G. (1991). Tsplib - a traveling salesman problem library. *ORSA journal on computing*, 3 (4):376–384.

Reinelt, G. (2003). *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer.

Salehipour, A. et al. (2011). Efficient grasp+vnd and grasp+vns metaheuristics for the traveling repairman problem. *4OR*, 9:189–209.

Silva, M. M., Subramanian, A., Vidal, T., e Ochi, L. S. (2012). A simple and effective metaheuristic for the minimum latency problem. *European Journal of Operational Research*, 221(3):513–520.

Uzun, G. e Kara, I. (2021). New formulations for the traveling repairman problem with time windows. *Expert Systems with Applications*, 176:114863.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Åukasz Kaiser, e Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, p. 5998–6008.

Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., e Abbeel, P. (2018). Variance reduction for policy gradient with action-dependent factorized baselines. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=H1tSsb-AW.

Yelmewad, P. e Talawar, B. (2021). Parallel deterministic local search heuristic for minimum latency problem. *Cluster Computing*, 24:969–995.

DOI: 10.59254/sbpo-2025-212556
https://proceedings.science/p/212556?lang=pt-br