UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Lucas Gabriel Oliveira Sales Lima

Análise de performance de algoritmos estocásticos aplicados ao problema do caixeiro viajante

Recife
2024

Lucas Gabriel Oliveira Sales Lima

Análise de performance de algoritmos estocásticos aplicados ao problema do caixeiro viajante

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Cleviton Vinícius Fonsêca Monteiro

Recife
2024

Agradecimentos

Agradeço à Deus pela vida, por todas as pessoas em meu caminho ao longo desse tempo, por todos os desafios, vitórias e aprendizados, e por todas as bênçãos visíveis e invisíveis para mim.

Aos meus pais, Fábio e Marta, por terem me educado para ser um homem íntegro e correr atrás dos meus objetivos, por terem me mostrado que tudo na vida vem através da dedicação e amor. Agradeço por todo o sacrifício e suporte que fizeram por mim em todos esses anos e ainda hoje mesmo à distância, continuam me guiando.

Agradeço à minha amada esposa Thaís, que sempre me amou e apoiou, minha amiga e conselheira em tempo integral. Obrigado por todo o suporte e paciência. Você foi e ainda é minha maior motivação para me tornar alguém melhor.

Agradeço aos meus amigos e familiares que sempre estiveram participando da minha vida acadêmica e, mesmo indiretamente, me dando dicas e conselhos para enfrentar os desafios da vida; minha tia Marília que sempre me mostrou como as coisas funcionam e sempre sorri comigo; minhas tias Andreia e Alice com inestimáveis conselhos; ao meu tio Marco que sempre alegra a mim e a todos ao seu redor.

Agradeço ao meu orientador, Prof. Dr. Cleviton Vinícius Fonsêca Monteiro, por ter aceitado me orientar neste trabalho de conclusão de curso; por todas as orientações, os ensinamentos, conselhos, que me deu desde o início da graduação e pela paciência e didática para retirar todas as minhas dúvidas.

Agradeço à todos que participam ou participaram do corpo docente, discente e técnico do Curso de Bacharelado em Sistemas de Informação da UFRPE, os quais são inúmeros e não consigo cita-los aqui.

Agradeço a todos os amigos que fiz durante a minha graduação, em especial Eliana França, Everton Veloso, Jadiel Eudes, Evele Lemos, José Augusto e tantos outros que participaram desta caminhada.

Análise de performance de algoritmos estocásticos aplicados ao problema do caixeiro viajante

Lucas Gabriel Oliveira Sales Lima[1], Prof. Cleviton Monteiro[1]

[1]Departamento de Estatistica e informatica - Universidade Federal Rural de Pernambuco, Rua Dom Manuel de Medeiros, s/n, - CEP: 52171-900 - Recife - PE - Brasil

lucas.sales@ufrpe.br, cleviton.monteiro@ufrpe.br

RESUMO: Algoritmos de otimização são ferramentas cada vez mais relevantes nas empresas modernas por serem capazes de otimizar processos e recursos, garantindo resultados mais eficientes e com processamento em tempo hábil para tomada de decisão. A comparação destes algoritmos é o processo comum durante seus estudos de adoção. Entretanto, a utilização de metodologias complexas muitas vezes pode levar à escolha de um algoritmo impreciso, pois seu resultado pode não refletir a realidade de uma empresa que busca implementar aplicações com recursos limitados. Tendo em vista esta problemática, surge a necessidade de avaliar estes algoritmos sob uma nova ótica. O objetivo principal deste trabalho é propor uma reflexão acerca da forma que experimentos em algoritmos são conduzidos. O presente estudo realizou experimentos com algoritmos de otimização utilizando recursos computacionais semelhantes àqueles encontrados na maioria das empresas, comparando com um outro trabalho no qual foram utilizados otimizações e *tunings* nesses mesmos algoritmos. Para o experimento, foi utilizado o problema do caixeiro viajante, através de 15 *benchmarking* dividido em 3 categorias, de acordo com o tamanho de cada artigo. Ao final foram obtidas métricas estatísticas do desempenho de cada algoritmo que, comparadas com o artigo de referência, obtiveram tempos de execução menores, sem comprometer a precisão dos resultados. Algoritmos probabilísticos possuem grande importância financeira para empresas com necessidades de gerir recursos rapidamente, tais como aeroportos e estaleiros. Sendo assim, a escolha adequada de parâmetros fornece uma visão mais acurada da realidade.

PALAVRAS-CHAVES: Algoritmos de otimização. Caixeiro viajante. Parâmetros.

ABSTRACT: Optimization algorithms are increasingly relevant tools in modern companies because they are capable of optimizing processes and resources, ensuring more efficient results and timely processing for decision making. Comparing these algorithms is a common process during their adoption studies. However, the use of complex methodologies can often lead to the choice of an imprecise algorithm, since its result may not reflect the reality of a company seeking to implement applications with limited resources. In view of this problem, the need arises to evaluate these algorithms from a new perspective. The main objective of this work is to propose a reflection on the way experiments in algorithms are conducted. The present study carried out experiments with optimization algorithms using computational resources similar to those found in most companies, comparing them with another work in which optimizations and tunings were used in these same algorithms. For the experiment, the traveling salesman problem was used, through 15 benchmarking divided into 3 categories, according to the size of each article. Finally, statistical metrics were obtained for the performance of each algorithm, which, when compared to the reference article, showed shorter execution times without compromising the accuracy of the results. Probabilistic algorithms are of great financial importance to companies that need to manage resources quickly, such as airports and shipyards. Therefore, the appropriate choice of parameters provides a more accurate view of reality.

KEYWORDS: Optimization algorithm. Traveling salesman. Parameters.

1. Introdução

No mundo moderno, é comum empresas se depararem com problemas de difíceis soluções, onde a resposta correta nem sempre é possível. Um porto, por exemplo, precisa encontrar a melhor forma de organizar e gerir muitos contêineres diariamente [Wang and Zhao, 2022]; entretanto, há uma sequência de ações que maximiza a eficiência desta tarefa, mas não vale a pena descobri-la, o motivo disso é o tempo e esforço por trás da resposta [Wetter and Wright, 2004]. Ao invés disso, empresas podem optar por soluções suficientemente boas que as permitam seguir em funcionamento.

É dentro deste cenário que surgem os algoritmos probabilísticos, os quais fornecem um conjunto de soluções viáveis de forma pseudo-aleatória [Freivalds, 1979]. Muitos são os fatores que influenciam na escolha de um algoritmo, tais como o custo computacional e o tempo de execução [Webb *et al.*, 2019]. Tendo isto em mente é extremamente relevante pautar a adoção de um algoritmo em uma série de experimentos nos principais algoritmos cogitados.

Existe uma aplicação especial para algoritmos probabilísticos chamada roteirização [Wang *et al*., 2008; Husnain and Anwar, 2022], um caso prático pautado no sistema de entregas de mercadorias e atendimento de serviços. Esse problema está recebendo muita atenção por conta de sua complexidade e importância na sociedade. O tema está intimamente relacionado com a *Teoria dos Grafos* [Diestel, 2024], algoritmos estocásticos conseguem desempenhar um papel importante na compreensão e análise desse tipo de situação [Medina-Gonzalez *et al.*, 2020]. Um sistema de entregas precisa de um roteiro para minimizar seu tempo de entrega e custos, de modo que a eficiência desta operação vai ser pautada no resultado destes algoritmos. Por isso é tão importante o seu estudo.

1.1. Problema e motivação

O mundo da computação está repleto de problemas e desafios difíceis de resolver. Estes problemas, muitas vezes, são abstrações do mundo real e são amplamente estudados por matemáticos. Cientistas trabalham e desenvolvem algoritmos para resolvê-los, mas alguns são tão complexos que um algoritmo "tradicional" não é viável [Kumar and Memoria, 2020].

O caixeiro viajante é um destes problemas, uma abstração para o problema de rotas onde o objetivo é encontrar a rota mais eficiente de forma a visitar todas as coordenadas de um conjunto [Abdullah and Razali, 2024]. Conforme o tamanho desse conjunto aumenta, sua complexidade cresce exponencialmente ao ponto de que algoritmos determinísticos não são

capazes de obter uma solução em um tempo hábil. A partir deste cenário, surgem os algoritmos estocásticos (probabilísticos) que são métodos computacionais que utilizam elementos de aleatoriedade em suas operações para resolver problemas. Eles podem incluir decisões aleatórias, geração de números pseudo aleatórios ou outras abordagens baseadas em probabilidade, e frequentemente são empregados para lidar com problemas complexos ou para explorar espaços de solução amplos. [Cheikhrouhou and Khoufi, 2021; Larni-Fooeik *et al.*, 2024].

Ao longo das décadas, muitos algoritmos estocásticos surgiram, inspirados nas mais variadas filosofias e com mecanismos de busca únicos. Embora tenham um mesmo objetivo, a sutileza de seu comportamento os torna singulares e mais adaptados que outros em determinadas circunstâncias. Algoritmos estocásticos são capazes de explorar o espaço de solução de forma mais abrangente do que algoritmos determinísticos. Eles podem evitar ficar presos em mínimos locais e podem encontrar soluções mais diversificadas, o que é especialmente útil em problemas com múltiplos ótimos locais [Hoos and Stützle, 2004; Erten *et al.*, 2020].

Com isso, é importante o estudo comparativo dos algoritmos em problemas específicos para futuras referências. Entender a curva de desempenho ao longo de um mesmo problema com diversos tamanhos pode significar a diferença entre um serviço eficiente ou um serviço que desperdiça recursos [Kaur and Verma, 2012; Si *et al*., 2016, Long *et al*., 2023].

1.2. Objetivo geral

Este trabalho teve por objetivo replicar os experimentos descritos por Halim e Ismail (2017) a fim de comparar os resultados obtidos utilizando, para isso, cinco algoritmos heurísticos dentro do problema do caixeiro viajante.

Os objetivos específicos deste trabalho incluem, primeiramente, a seleção e categorização de problemas de referência do Caixeiro Viajante, que servirão como base para a execução dos testes. Em seguida, esses problemas de referência serão utilizados para conduzir os experimentos necessários. Será feita a seleção de algoritmos candidatos para compor o conjunto de técnicas a serem avaliadas, com a implementação de cinco desses algoritmos escolhidos utilizando a linguagem Python. Além disso, será garantida a uniformidade dos recursos computacionais empregados em cada experimento, assegurando condições consistentes de execução. Os experimentos serão conduzidos de maneira alinhada à abordagem descrita no artigo de referência, com o objetivo de permitir comparações confiáveis. Por fim,

será realizada uma análise e avaliação detalhada do desempenho obtido por cada algoritmo nas execuções realizadas, permitindo a obtenção de conclusões baseadas nos resultados.

1.3. Justificativa

Este trabalho tem como objetivo reunir insumos importantes para as equipes de tecnologia, facilitando a tomada de decisão na adoção de algoritmos estocásticos. Além disso, busca demonstrar de forma prática as vantagens e desvantagens de cada algoritmo abordado. Uma das metas é replicar o estudo "Combinatorial Optimization: Comparison of Heuristic Algorithms in Travelling Salesman Problem", desenvolvido por Halim e Ismail (2017), a fim de fornecer novas evidências sobre o tema. O trabalho também pretende fornecer uma classificação dos algoritmos estocásticos com base em sua performance.

1.4. Abordagem metodológica

A replicação empírica é essencial para a ciência, pois valida, refina e expande nosso conhecimento. Ela assegura que os achados científicos sejam robustos, generalizáveis e úteis para aplicações práticas. Além disso, promove a integridade e a transparência científica, contribuindo para um progresso científico mais confiável e sustentável [Shepperd *et al.*, 2018].

O trabalho de Da Silva *et al.* (2014) discute a importância de experimentos empíricos realizados na engenharia de software, e por meio de um mapeamento sistemático, analisa a metodologia desses estudos. O trabalho também sumariza, através de seis perguntas, aspectos que se deve levar em consideração ao replicar um trabalho. Uma dessas questões trata sobre o que é considerado um bom relatório de replicação, indicando que o principal motivo para estes relatórios estejam abaixo no critério de qualidade é a restrição do número de páginas necessárias para descrever aspectos do trabalho original.

Sobre o estudo e comparação entre algoritmos, é possível utilizar muitos métodos comparativos distintos, a depender do aspecto que se deseja avaliar. No entanto, uma série de requisitos e uniformidades precisam ser cumpridos para que a comparação seja justa e assertiva. Brownlee *et al*. (2007) apresentam o processo de seleção de diretrizes e parâmetros a fim de alcançar um experimento consistente e imparcial, comparando algoritmos de otimização.

O artigo de Hassam, Cohanim and Weck (2005) compara dois dos principais algoritmos utilizados para problemas de otimização combinatória. O trabalho estabelece métricas para a condução dos experimentos, através do emprego de um conjunto de problemas de referência. Cada teste evidencia um aspecto diferente dos algoritmos testados.

Esses três últimos trabalhos citados deixam clara a relevância e a necessidade de se haver um controle de métricas ao conduzir um trabalho de replicação de estudos em algoritmos. Inspirado neles e na motivação do cenário atual de algoritmos probabilísticos, este experimento foi idealizado para revalidar o desempenho descrito pelo trabalho modelo, bem como discutir a hipótese de parâmetros proporcionais à instância do experimento.

1.5. Organização do trabalho

Este trabalho está organizado em 6 seções principais. Na seção 2 é feita uma revisão teórica de todos os conceitos, termos e tecnologias utilizadas por este estudo. Ao longo da seção 3 é feita uma breve análise de trabalhos correlatos. Na seção 4, a abordagem da proposta é descrita mais detalhadamente. Na seção 5, é apresentada toda a metodologia que guiou este trabalho, juntamente aos experimentos, processos e ferramentas. A seção 6 apresenta as conclusões obtidas e as possibilidades para o desenvolvimento de trabalhos futuros, por fim, a seção 7 introduz os próximos trabalhos.

2. Referencial teórico

Nesta seção, os conceitos abordados por este trabalho serão explicados através de um resumo, juntamente com os termos técnicos, tecnologias e algoritmos utilizados no decorrer deste estudo. Os conceitos teóricos abordados foram divididos nos seguintes grupos: Conceitos gerais sobre aprendizado de máquina, algoritmos estocásticos, complexidade computacional e otimização;

2.1. Algoritmos estocásticos

Algoritmos desta categoria são definidos por sua metodologia probabilística ao solucionarem problemas que os algoritmos determinísticos não conseguem [Spall, 2003]. A Figura 1 ilustra alguns dos principais algoritmos estocásticos e como estão classificados.

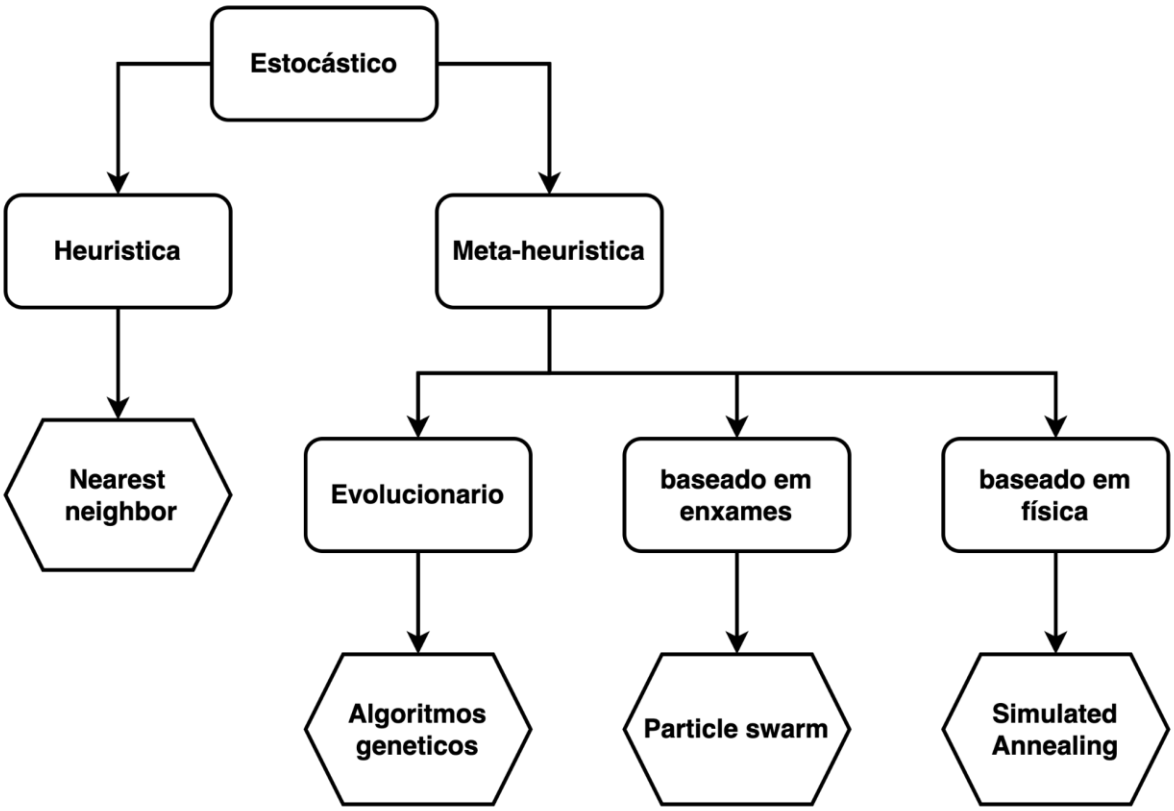

Figura 1. Classificação dos algoritmos estocásticos

2.1.1. Heurística

Um algoritmo heurístico é uma abordagem que utiliza regras práticas e/ou metodologias probabilísticas em vez de tentar encontrar a melhor solução possível. Esses algoritmos são projetados para encontrar soluções boas o suficiente em um tempo razoável, especialmente em situações em que a busca exaustiva por uma solução ótima seria computacionalmente inviável [Pearl, 1984; Jahwar and Abdulazeez, 2020].

2.1.2. Meta-heurística

Algoritmos meta-heurísticos são estratégias de otimização que podem ser utilizadas para resolver vários problemas de otimização sem dependerem de alguma especificidade do problema. O termo "meta" vem justamente dessa capacidade de abstração destes algoritmos,

que fornecem diretrizes adaptáveis [Balamurugan, Natarajan and Premalatha, 2015; Kaur *et al*., 2023].

2.1.3. *Nearest Neighbor Algorithm*

Este é o primeiro algoritmo utilizado neste experimento para resolver o problema. O *Nearest Neighbor Algorithm (NN)* é um algoritmo do tipo "aprendizado supervisionado" e consiste numa busca pelo ponto similar mais próximo, dado um conjunto de pontos, seja por regressão ou classificação. Foi desenvolvido por Evelyn Fix e Joseph Hodges em 1951. O NN possui uma implementação simples e executa rapidamente pequenos conjuntos de dados de treinamento. Também não precisa de conhecimento prévio sobre a estrutura dos dados de treinamento [Sahalot, Antima and Shrimali, Sapna 2014]. Quando o conjunto de treinamento é grande, pode-se ocupar muito espaço e o tempo de teste pode se tornar muito grande.

2.1.4. *Simulated Annealing*

Simulated Annealing (SA) é o algoritmo meta-heurístico que utiliza uma técnica probabilística para encontrar o valor ótimo global de uma função [Das and Chakrabart, 2005]. O SA foi baseado no processo metalúrgico de aquecer e resfriar o metal de forma controlada com o objetivo de alterar algumas propriedades do material. Para problemas onde o espaço de busca é discreto, o SA é muito utilizado [Ghannadi and Kourehli, 2021]. A Figura 2 representa de forma esquemática o fluxo de um (SA).

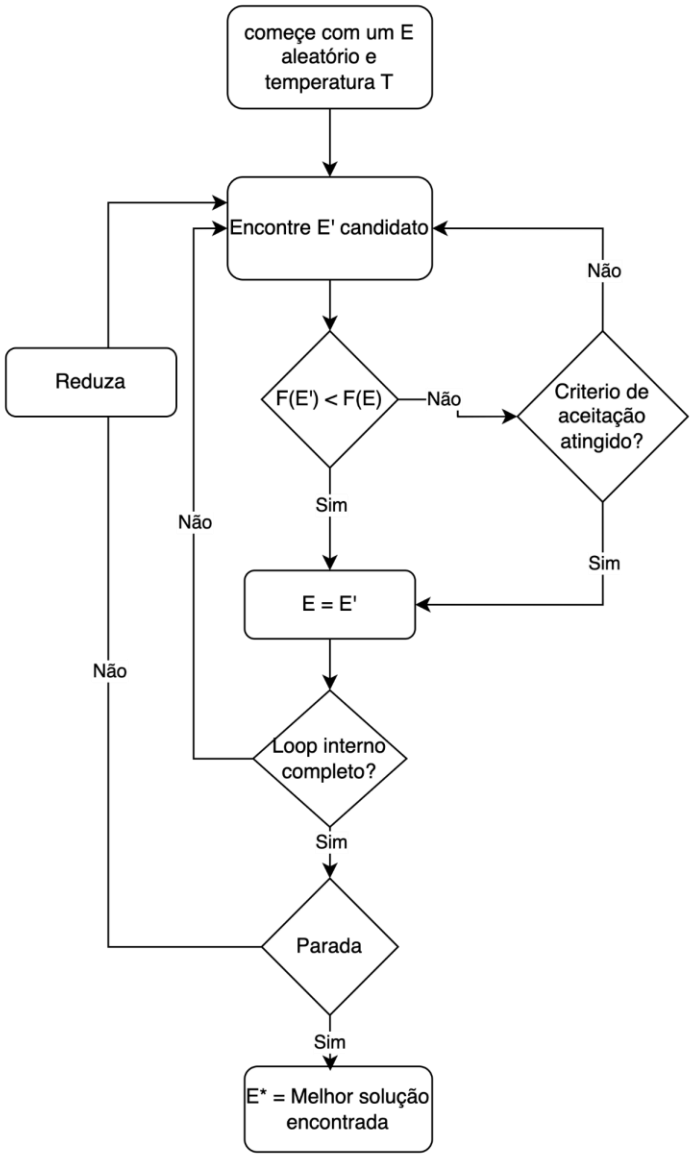

Figura 2. Fluxo do Simulated Annealing

2.1.5. *Genetic Algorithm*

Algoritmos genéticos, ou Genetic Algorithm (GA), são uma classe particular de algoritmos evolutivos, caracterizados por serem probabilísticos e fornecer um mecanismo de busca paralela e adaptativa, fundamentado no princípio da sobrevivência dos mais aptos e na reprodução, conforme descrito por [Kramer, 2017]. Eles se distinguem dos algoritmos tradicionais de otimização em quatro aspectos principais. Primeiramente, baseiam-se em uma codificação do conjunto das soluções possíveis, ao invés de trabalharem diretamente com os parâmetros de otimização. Além disso, os resultados são apresentados como um conjunto de

soluções, chamado de população, e não como uma única solução. Outra característica importante é que esses algoritmos não necessitam de conhecimento específico do problema em questão, exigindo apenas uma forma de avaliar os resultados. Por fim, os algoritmos genéticos utilizam transições probabilísticas em vez de regras determinísticas.

Função-objetivo: É o objetivo por trás da otimização. Esta função é aplicada aos indivíduos e, também, é usada na comparação de resultados;

Indivíduo: É a representação da busca realizada pelo algoritmo, o portador do *código genético* (espaço de busca);

Seleção: A seleção é o mecanismo de ordenação dos indivíduos de acordo com a função-objetivo, de modo que lhes são atribuídas probabilidades decrescentes de serem escolhidos. Probabilidades, essas, que são proporcionais à razão entre a adequação do indivíduo e a soma das adequações de todos os indivíduos da população. A escolha é feita aleatoriamente de acordo com essas probabilidades. Os mais bem adaptados, então, são escolhidos como pais, sem deixar de lado a diversidade dos menos adaptados. As formas de seleção podem, ainda, ser aplicadas de acordo com a especificidade do problema a ser tratado;

Reprodução*:* A reprodução, tradicionalmente, é dividida em três etapas: acasalamento, recombinação e mutação. O acasalamento é a escolha de dois indivíduos para se reproduzirem (geralmente gerando dois descendentes para manter o tamanho populacional). A recombinação, ou *crossing-over*, é um processo que imita o processo biológico homônimo na reprodução sexuada: os descendentes recebem em seu código genético parte do código genético do pai e parte do código da mãe. Esta recombinação garante que os melhores indivíduos sejam capazes de trocar entre si as informações que os levam a serem mais aptos a sobreviver e, assim, gerarem descendentes ainda mais aptos [Lambora, A. and Gupta, K. 2019]. Por último vem as mutações, que são feitas com probabilidade mais baixa possível. Elas possuem como objetivo permitir maior variabilidade genética na população, impedindo que a busca fique estagnada em um mínimo local.

2.1.6. *Ant Colony Optimization*

Ant Colony Optimization (ACO) foi desenvolvido por Marco Dorigo em 1992 para sua tese de Phd. Este algoritmo se baseia no comportamento real de formigas em busca de alimento, e é empregado na resolução de problemas que envolvem o descobrimento de caminhos em grafos. Uma vez que uma dessas formigas artificiais encontra comida, ela precisa voltar para a colônia deixando um rastro de feromônios; o rastro será seguido por outras formigas que encontrá-lo. É importante observar que rastros mais antigos ou menos usados sofrerão um processo de evaporação, tendendo a ser menos atrativos. As formigas se adaptam em tempo real a um grafo dinâmico, isto é, um grafo que muda sua configuração [Dorigo, Di Cargo and Gambardella, 1999; Kaur, Chaudhary and Singh, 2023].

2.1.7. *Tabu search*

Tabu search (TS) é um algoritmo metaheurístico que realiza buscas locais. Foi desenvolvido por Fred W. Glover e é amplamente utilizado em problemas de optimização. Embora buscas locais possuam a tendência de se estacionarem em regiões subótimas, este algoritmo possui a característica de relaxar sua regra base ganhando mais mobilidade em uma dessas regiões. *Tabu search* possui muitas similaridades com o algoritmo simulated annealing, tendo em vista que ambos possuem movimentos de "*down hill*". Seu funcionamento pode ser resumido em uma busca local que, interativamente, procura soluções melhores que a inicial escolhida aleatoriamente até que algum critério de aceitação seja atingido. Seu diferencial consiste em uma estrutura de memória que gera uma lista tabu onde soluções gravadas não serão revisitadas em curto prazo [Cordeau and Laporte, 2005; Prajapati, Jain, Chouhan, 2020].

2.2. Complexidade computacional

A *Teoria da Complexidade Computacional* é o campo de estudo que classifica problemas computacionais de acordo com sua dificuldade. Neste contexto, um problema computacional é entendido como uma tarefa que é, em princípio, passível de ser resolvida por um computador. Estes problemas são classificados em classes de complexidade, conforme apresentado na Tabela 1:

Tabela 1. Ordem de grandeza.

Designação	Função
Constante	c
Logarítmica	log n
Logarítmica quadrática	log
Linear	n
n log n	n log n
Quadrática	n^2
Cúbica	n^3
Polinomial	nr
Exponencial	2n
Fatorial	n!

Além da classificação de complexidade, existe ainda a classificação de problemas computacionais, demonstrado através de um diagrama teórico na Figura 3.

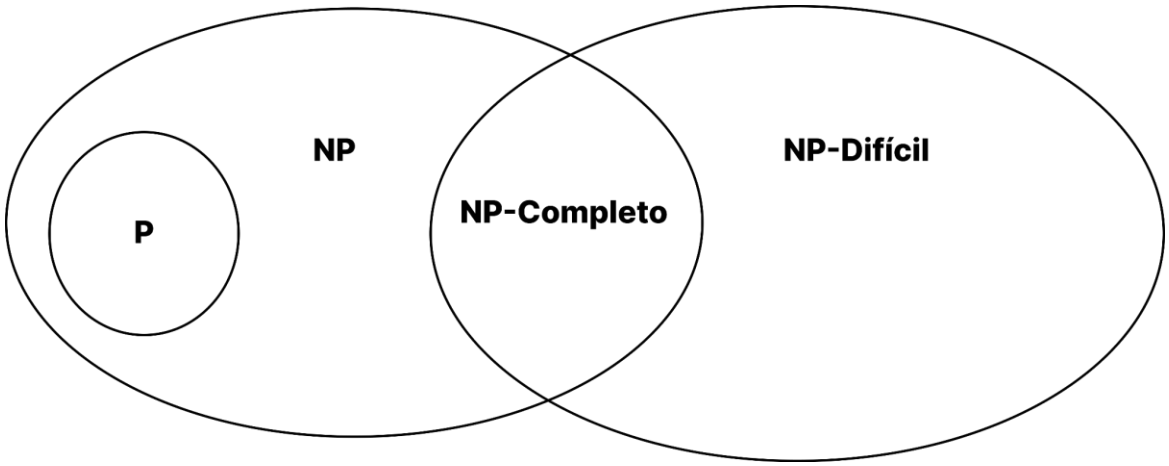

Figura 3. Diagrama de problemas computacionais.

Dentre eles, está a classe de problemas NP-difícil [Jaffe, A. 2006]. Um problema H é dito NP-difícil se, e somente se, existir um problema NP-completo L que é resolvido em tempo polinomial por uma máquina de turing não determinística, sendo esta máquina uma subrotina para resolver H. O problema do caixeiro viajante (TSP) é um problema clássico de otimização NP-difícil, onde o objetivo é tentar determinar a menor rota para percorrer todos os vértices de

um grafo completo e pesado, e retornar ao vértice de origem passando uma única vez por cada aresta (caminho hamiltoniano).

2.3. Problemas de otimização

Todo problema solucionável apresenta pelo menos uma solução ótima. Otimização trata da busca de uma solução que minimize ou maximize uma métrica relacionada ao problema [Figueira et al. 2017]. Algoritmos de otimização, por sua vez, são ferramentas matemáticas que são executadas de forma iterativa, comparando várias soluções possíveis até que uma solução ótima, ou pelo menos satisfatória, seja encontrada.

Os algoritmos de otimização dependem de cinco elementos fundamentais, os quais estão ligados à natureza do problema. Em primeiro lugar, as variáveis de design representam os parâmetros a serem ajustados no processo de busca por soluções, podendo ser contínuas, discretas ou inteiras. Essas variáveis sofrem modificações durante as iterações do algoritmo. Em segundo lugar, os limitantes são restrições específicas que relacionam as variáveis de design com aspectos físicos ou limitações de recursos, delimitando a viabilidade das soluções.

Outro componente importante é o espaço de busca, que corresponde ao conjunto de soluções possíveis, limitado por funções de restrição que definem as fronteiras desse espaço. A função objetivo, por sua vez, é a métrica que o algoritmo busca minimizar ou maximizar, sendo essencial que ela seja expressa matematicamente, uma vez que métodos numéricos são amplamente utilizados na otimização. Por fim, os limites das variáveis restringem mais o espaço de busca, estabelecendo condições que a solução ótima deve necessariamente respeitar.

A otimização pode ser realizada por meio de duas abordagens principais: os métodos determinísticos e os métodos probabilísticos. Os métodos determinísticos geram sequências estruturadas de possíveis soluções, geralmente exigindo o cálculo de derivadas da função objetivo em relação às variáveis de design. Nessa abordagem, tanto a função objetivo quanto as restrições são descritas por relações matemáticas bem definidas, sendo necessário que a função objetivo seja contínua e diferenciável dentro do espaço de busca, conforme apontado por Bastos (2004). Em contrapartida, os métodos probabilísticos utilizam avaliações estocásticas e introduzem elementos aleatórios no processo de busca. Apesar de sua maior flexibilidade e aplicabilidade em problemas complexos ou com funções não diferenciáveis, esses métodos apresentam a desvantagem de um tempo de processamento geralmente superior ao dos métodos determinísticos.

A otimização pode ser realizada por meio de duas abordagens principais, os métodos determinísticos e os métodos probabilísticos, cada um com características específicas que os tornam mais adequados a diferentes tipos de problemas.

Os métodos determinísticos baseiam-se em uma sequência sistemática de soluções potenciais, utilizando frequentemente a primeira derivada da função objetivo em relação às variáveis de projeto. Essa abordagem pressupõe que a função objetivo e as restrições sejam descritas por expressões matemáticas bem definidas, contínuas e diferenciáveis no espaço de busca. Essas condições permitem a aplicação de técnicas analíticas para identificar soluções ótimas locais ou globais, tornando os métodos determinísticos precisos e eficientes em problemas bem comportados. No entanto, eles podem enfrentar limitações em cenários onde a função objetivo apresenta descontinuidades, múltiplos ótimos locais ou outras complexidades matemáticas que dificultam a convergência para uma solução global [Bastos, 2004].

Por outro lado, os métodos probabilísticos destacam-se por sua flexibilidade e capacidade de explorar o espaço de busca sem a necessidade de derivadas ou equações matemáticas explicitamente definidas. Esses algoritmos incorporam elementos estocásticos em seu funcionamento, permitindo que lidem com problemas complexos, não lineares ou com múltiplos ótimos de maneira eficaz. Exemplos de métodos probabilísticos incluem algoritmos genéticos, otimização por enxame de partículas e métodos de simulação. Contudo, essa abordagem apresenta como principal desvantagem o elevado tempo de processamento, pois o uso intensivo de cálculos e iterações aumenta significativamente os requisitos computacionais. Além disso, embora sejam eficazes para explorar o espaço de busca, os métodos probabilísticos não garantem a identificação do ótimo global, dependendo fortemente de ajustes adequados nos parâmetros do algoritmo para alcançar resultados satisfatórios.

A escolha entre métodos determinísticos e probabilísticos deve considerar as características do problema, a disponibilidade de recursos computacionais e a natureza da função objetivo. Em algumas situações, a combinação dessas abordagens pode ser vantajosa, aproveitando a precisão dos métodos determinísticos para refinar soluções identificadas pelos métodos probabilísticos, resultando em um processo de otimização mais robusto e eficaz.

2.3.1. Caixeiro Viajante

O Problema do Caixeiro Viajante (TSP) é um problema específico de otimização NP-difícil, inspirado na problemática de representantes comerciais que precisam percorrer várias cidades em um melhor tempo e rota possíveis. Com isto, o TSP busca determinar uma menor

rota entre um conjunto de cidades, visitando-as apenas uma vez a fim de retornar, ao final, à cidade de origem [Pop *et al*., 2023; Sarikyriakidis, and Goulianas, 2023] .

O problema é definido por TSP(G, c): dados um grafo G e um custo c_e em Q≥ para cada aresta (cidade) para determinar um circuito hamiltoniano (Figura 4) C que minimize c(C). O grafo G é completo e tem um custo c_{ij} associado a cada par (i,j) de vértices. O TSP restrito ao conjunto de instâncias (G, c) em que G é completo e c satisfaz a desigualdade triangular é conhecido como problema do caixeiro viajante simétrico [Hoffman *et al*., 2013; Osaba, Yang and Del Ser, 2020].

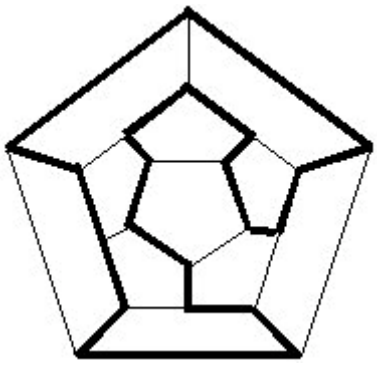

Figura 4. Caminho Hamiltoniano.

Matematicamente, o problema pode ser formulado como um grafo completo G=(V,E) onde:

- V é o conjunto de vértices (cidades).
- E é o conjunto de arestas, que representa as distâncias entre as cidades.

Seja:

- c(i,j) o custo (distância) de viajar da cidade i para a cidade j,

xij uma variável binária, onde xij=1 se o caixeiro viajar diretamente da cidade i para a cidade j, e xij= 0 caso contrário.
O objetivo é minimizar o custo total da viagem, dado por:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c(i,j)\cdot x_{ij}$$

Sujeito às seguintes restrições:

1 - Cada cidade deve ser visitada exatamente uma vez:

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall_i \in V$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall_j \in V$$

2 - O caminho deve formar um ciclo único que percorre todas as cidades (evitando subciclos, uma restrição conhecida como restrição de subtour):

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subset V, \; |S| \geq 2$$

A solução ótima do problema dá o caminho de menor custo para o caixeiro viajante. No entanto, o TSP é um problema NP-difícil, ou seja, não existe um algoritmo eficiente conhecido para resolvê-lo em tempo polinomial para todas as instâncias. O TSP é bastante utilizado como método comparativo na performance de diversos algoritmos, além de ser relevante para o mundo real como logística de estaleiros, serviços de limpeza urbana, projetar dispositivos de hardware e rede de computadores e muitos outros [Sathya and Muthukumaravel, 2015].

Inúmeros algoritmos são avaliados através do TSP. Apesar disso, destaca-se que este trabalho buscou avaliar 5 algoritmos estocásticos: Nearest Neighbor (NN), *Simulated Annealing* (SA), *Genetic Algorithm* (GA), *Ant Colony* (AC) e *Tabu Search* (TS).

3. Trabalhos relacionados

Nesta seção serão descritos os trabalhos correlatos relevantes, comparando seus objetivos com os deste trabalho.

Em Blum and Roli (2003) é descrito um estudo em diversos algoritmos metaheurísticos, comparando-os em termos de intensificação e diversificação. O trabalho analisou os componentes conceituais destes algoritmos, buscando similaridades e diferenças com o objetivo de tentar criar, futuramente, algoritmos metaheurísticos híbridos. No entanto, não foi encontrada uma forma de construir um algoritmo híbrido com performance superior aos descritos na literatura. Por isto, este trabalho apenas compara algoritmos tidos como "puros".

O artigo de Hassam, Cohanim and Weck (2005) compara dois dos principais algoritmos utilizados para problemas de otimização combinatória. Os algoritmos *particle swarm* (PSO) e *genetic algorithm* (GA) são ambos analisados, bem como submetidos a problemas de benchmark, a exemplo da, função banana (Rosenbrock), da função *Eggcrate* e *Golinski's Speed Reducer* que diferem do problema proposto no atual trabalho. O resultado do artigo é que o PSO é computacionalmente mais eficiente que o GA para estes problemas devido ao número de funções utilizadas. Este resultado é relevante para o atual experimento que usará um outro tipo de problema (caixeiro viajante).

Além dos algoritmos já citados anteriormente, existem muitos outros que se baseiam nas mais diversas filosofias e que podem, assim, serem empregados para resolver problemas de otimização combinatória. O trabalho de Karaboga and Akay (2009) se propôs a estudar e avaliar o algoritmo *"Artificial Bee Colony"* (ABC) que é baseado em enxames. O trabalho comparou os resultados obtidos em diversos testes numéricos com os resultados de algoritmos mais antigos, tais como: *Genetic algorithm, particle swarm* e *differential evolution algorithm*. Embora o número de parâmetros de controle seja menor no ABC, sua performance foi similar e um pouco melhor do que aquela oriunda dos outros algoritmos onde há muito mais parâmetros que influenciam na performance.

Assim como este trabalho, o artigo de Halim and Ismail (2017) estudou a performance de algoritmos metaheurísticos na resolução do problema do caixeiro viajante. O trabalho utilizou problemas de benchmark em três escalas: simples, moderada e complexa. Estas escalas dizem respeito ao volume de arestas (cidades) a ser testado. A instância simples indica um volume de até 99 nós; a moderada é um intervalo entre 100 e 190; por fim, a instância complexa apresenta o volume acima dos 190 nós. O resultado descoberto vem através da análise do tempo computacional, dos valores estatísticos e da convergência. Para o tempo computacional, o *Nearest Neighbor* (NN) foi o melhor algoritmo, enquanto para a métrica de valores estatísticos foram os algoritmos *tabu search* (TS) e *genetic algorithm* (GA) que lideraram, embora a precisão do TS tenha tendência a cair conforme o tamanho do problema. Por fim, os algoritmos NN e TS tiveram a melhor convergência dentro de 300 iterações. Também é importante

observar que, embora a maioria dos algoritmos estudados sejam "puros", ou seja, sem alterações significativas em seus mecanismos, o algoritmo *Tree physiology optimization* foi modificado especificamente para atender o problema do caixeiro viajante. Não sendo possível replicar as alterações neste último algoritmo, ele foi deixado de fora do presente estudo.

4. Desenho do experimento

Nas seções anteriores foram abordadas as ferramentas e técnicas que serão utilizadas neste estudo, bem como os trabalhos relacionados com a temática em questão. Estes trabalhos demonstram como esta pesquisa está inserida no contexto dos algoritmos probabilísticos, e em temas relacionados à performance em otimização. Desta forma, os resultados obtidos neste trabalho podem ser usados para guiar empresas e organizações na tomada de decisões, e na contínua melhoria de suas operações.

4.1. Ambiente experimental

Nesta subseção será apresentado o ferramental técnico utilizado para executar os experimentos. A fim de tornar todas as comparações entre algoritmos a mais justa possível, todos os experimentos foram executados no mesmo ambiente controlado e de forma sequencial, com um intervalo de 1 min entre os algoritmos para que os recursos iniciais fossem restabelecidos. No momento da execução, todos os aplicativos não relacionados estavam fechados e qualquer conexão com a internet estava desabilitada. O experimento foi executado três vezes em cada instância (pequeno, médio e grande), sendo que, ao final de cada uma, os resultados foram anotados em uma planilha e a máquina foi reiniciada (software rejuvenation). A máquina responsável é um *MacBook Pro*. Segue abaixo sua configuração:

- Processador 2,6 GHz Intel Core i7 6-core;
- Memória 16 GB 2667 MHz DDR4;
- OS Ventura 13.5.1.

Além da máquina e do sistema operacional, é importante padronizar a forma de implementação dos algoritmos. Todos foram implementados em *Python* 3.9, usando bibliotecas de *profiling* e executados através de um terminal, conforme ilustrado na Tabela 2:

Tabela 2. Bibliotecas Python utilizadas no experimento.

Biblioteca	Objetivo
Numpy v1.26.0	Implementação de matrizes
Tsplib95 v0.7.1	Implementação de alias para arquivos contendo o TSP
Memory-Profiler v0.61.0	Observador utilizado na coleta de memória de uma função
time	Observador utilizado para medir o tempo de execução
Matplotlib 3.8.2	Gerador de gráficos

4.1.1. Pré-processamento do experimento

Para a execução dos experimentos foi preciso realizar um pré-processamento dos arquivos utilizados. Para que o TSP esteja condizente com o formato proposto por este trabalho, foi necessário remover o cabeçalho dos arquivos e avaliar a existência de ao menos um caminho Hamiltoniano. Isto significa que, dado um grafo, é possível percorrer todos os vértices uma única vez retornando ao vértice de origem [Rahman and Kaykobad, 2005]. Todos os arquivos são convertidos para o formato de uma lista de tuplas, onde cada tupla é coordenada (x,y) no plano cartesiano.

4.2. Replicação de Experimentos de Referência

Primeiramente foi realizada a replicação dos experimentos do trabalho de referência. Todos os dados utilizados foram coletados de 15 problemas de referência encontrados no repositório, criado em 2018 e acessado em 21/02/2024, da universidade alemã *Uni-heidelberg Teaching*. Apenas duas bases de dados estavam indisponíveis (Eil51 e Eil76) e foram substituídas por "att48" e "ulysses22", enquanto que as demais são similares ao trabalho de referência. As instâncias testadas estão divididas em arquivos que contêm um grafo completo e pesado, classificados em três grupos: Pequeno ($n < 100$), Médio ($100 \leq n < 190$) e Grande ($n \geq 190$), conforme mostrado pela Tabela 3. Dentro de cada arquivo, há um cabeçalho seguido de linhas contendo 3 números, separados por espaço, significando o nó e a coordenada (x,y),

respectivamente. Os algoritmos são executados até que seja encontrado o valor mínimo global ou até a última iteração definida por hiperparâmetro.

Tabela 3. Classificação dos arquivos que contém o TSP.

Index	Nome	Menor distância	Tamanho
1	Ulysses22	75	Pequeno
2	Att48	33523	Pequeno
3	Berlin52	7542	Pequeno
4	St70	675	Pequeno
5	Pr76	108159	Pequeno
6	Rat99	1211	Pequeno
7	KroA100	21282	Médio
8	Eil101	629	Médio
9	Ch130	6110	Médio
10	Ch150	6528	Médio
11	Rat195	2323	Grande
12	D198	15780	Grande
13	A280	2579	Grande
14	Rd400	15281	Grande
15	Pcb442	50778	Grande

4.2.1. Parâmetros utilizados

Esta subseção apresentará os parâmetros de cada algoritmo utilizado durante os experimentos. É importante notar que todos os algoritmos foram implementados sem maiores melhorias, ou ainda, sem o uso de bibliotecas externas. O repositório com os códigos utilizados estão disponíveis em https://github.com/lucas-sales/traveler-salesman.

A escolha de parâmetros influencia diretamente na performance e qualidade das soluções encontradas por um algoritmo heurístico, no entanto, foi preciso fazer uma concessão nestes parâmetros. A máquina utilizada possui menos recursos computacionais que a utilizada

pelo trabalho de referência, o número de iterações e o tamanho populacional foram divididos por 10 para que a máquina fosse capaz de finalizar a tarefa em um tempo hábil.

Na Tabela 4 pode-se observar parâmetros como, a taxa de mutação em algoritmos genéticos e a taxa de evaporação de feromônio em colônias de formigas, ambos ajudam a equilibrar a exploração (busca de novas soluções) e o aprimoramento das soluções atuais. O ajuste desses parâmetros é crucial para evitar uma possível convergência indesejada, garantindo que o algoritmo explore boa parte do espaço amostral. O ajuste de parâmetros em algoritmos é chamado de *tuning*, podendo ser realizada de forma automática ou manual (empírica). Neste trabalho foi realizada a metodologia automática, uma vez que os parâmetros foram escolhidos baseados no trabalho modelo.

Tabela 4. Configuração de parâmetros por algoritmo.

Algoritmos	**Parâmetros**
Nearest Neighbor	Iteração = 1000 Rota visitada = tamanho do problema
Ant Colony Optimization	Número de iterações = 1000 Número de formigas = 20 α (feromônio) = 1.0 β (heurística) = 1.0 P (taxa de evaporação) = 0.95
Tabu Search	iteração = 1000 tamanho da lista tabu = 30
Genetic Algoritm	número de gerações = 1000 população = 30 mutação = *Swap, flip, slide* seleção = *Roulette-wheel*
Simulated Annealing	iteração = 1000 temperatura inicial = 0.025 α (taxa de resfriamento) = 0.99

4.3. Métricas

Algumas métricas foram observadas ao longo do experimento, como tempo de execução, distância percorrida, uso de memória e complexidade computacional.

O tempo de execução por vezes se torna a métrica mais importante em problemas complexos, uma vez, algumas soluções de problemas difíceis levariam tempos acima do polinomial para serem obtidos, tornando o processo inviável.

A distância percorrida total é o somatório das distâncias entre cidades visitadas na rota escolhida pelo algoritmo, essa métrica indica o quão próximo um algoritmo chegou do menor percurso possível. O uso de memória aponta o grau de eficiência de um algoritmo e indica a sua capacidade de escalabilidade, algoritmos que consomem menos memória podem lidar com instâncias maiores do problema. A complexidade computacional demonstra uma previsão do comportamento dos algoritmos, indicando a progressão do tempo de resolução e desempenho destes, à medida em que as instâncias do problema crescem. Essas métricas foram utilizadas como parâmetros comparativos para a avaliação da performance dos algoritmos [Halim and Ismail, 2021].

4.4. Limitações e Ameaças

Nesta seção, são apresentadas as principais limitações e ameaças que podem impactar os resultados e a validade deste trabalho. As limitações referem-se a aspectos intrínsecos ao estudo, como restrições metodológicas, lacunas de recursos ou escopo delimitado, que podem ter influenciado as análises e conclusões. Já as ameaças dizem respeito a fatores externos e potenciais riscos que poderiam comprometer a aplicação prática ou a generalização dos resultados. Reconhecer esses elementos é essencial para contextualizar as descobertas e apontar direções para estudos futuros, assegurando maior transparência e rigor acadêmico.

A primeira limitação deste trabalho refere-se ao volume de experimentos realizados. Para obter insumos comparativos suficientes, seria necessário realizar 30 repetições para cada experimento. No entanto, devido à longa duração de alguns experimentos, que se estendem por várias horas, não foi possível atingir esse número ideal. Outra limitação identificada foi a ausência de técnicas mais formais para a definição dos hiper parâmetros utilizados nos experimentos, o que pode impactar a reprodutibilidade e a eficiência dos resultados.

A principal ameaça à validade deste estudo está relacionada à implementação manual dos algoritmos, sem o uso de bibliotecas ou códigos previamente otimizados. Isso pode resultar na

ausência de otimizações específicas e no uso de estruturas de dados diferentes das descritas na literatura, o que pode influenciar a performance, especialmente em algoritmos como o *Ant Colony* que apresenta a maior sensibilidade à essas otimizações.

5. Resultados

Nesta seção, é apresentada uma análise abrangente das métricas de desempenho dos algoritmos heurísticos aplicados ao Problema do Caixeiro Viajante (TSP), comparando-as ao trabalho de referência. Foram avaliados cinco algoritmos: *Simulated Annealing* (SA), *Nearest Neighbor* (NN), *Tabu Search* (TS), *Ant Colony Optimization* (AC) e *Genetic Algorithm* (GA).

5.1. Tempo de execução

O gráfico exibido na Figura 5, mostra o tempo necessário que cada algoritmo precisou para processar o número de nós. Avaliando sob a ótica de velocidade, os algoritmos TS e SA foram os que apresentaram o melhor desempenho. Contudo, estes dois algoritmos obtiveram as piores médias de precisão do grupo.

Embora mais robusto, o AC foi o que apresentou o pior desempenho de tempo em todas as instâncias, tendo que lidar com uma matriz n^2 que é acessada constantemente por cada formiga, outro fator determinante foi o volume de operações, em um cenário otimizado em CPUs modernas (com vários núcleos) ou em GPUs, pode-se esperar que a taxa de operações possa variar numa faixa de 1 a 100 milhões de operações por segundo.

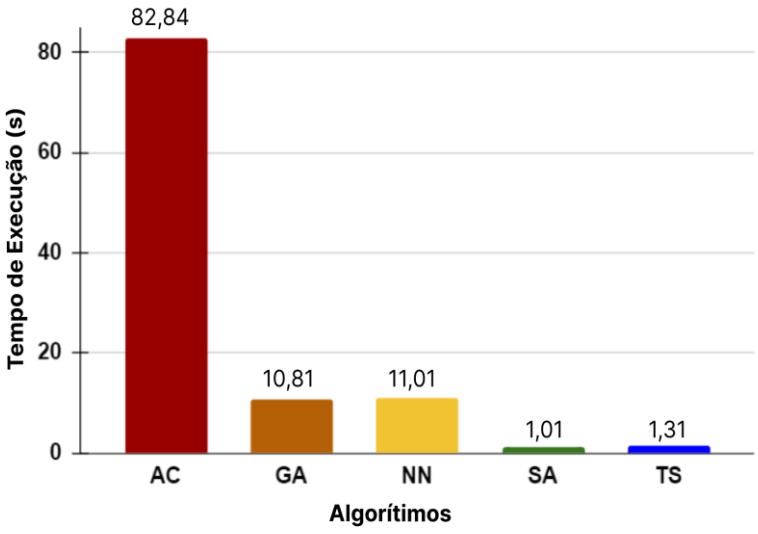

Figura 5. Gráfico comparativo do tempo de execução médio.

Os resultados obtidos diferem parcialmente daqueles encontrados pelo trabalho de referência, tendo como ponto em comum o algoritmo AC apontado como o pior avaliado nesta métrica. A Tabela 5 compara e enumera os algoritmos de acordo com a performance dos dois trabalhos em termos de tempo de execução. O algoritmo TS foi o mais discrepante entre o grupo avaliado, a razão desta diferença se deve a uma mudança no número de iterações feita pelo trabalho de referência, aumentando assim a precisão do algoritmo e o tempo de execução.

Tabela 5. Comparativo de performance dos algoritmos no trabalho atual e Halim and Ismail (2017)

Ranking	Neste trabalho	Halim and Ismail (2017)
1	Simulated annealing	Nearest Neighbor
2	Tabu Search	Genetic Algorithm
3	Genetic algorithm	Simulated Annealing
4	Nearest Neighbor	Tabu Search
5	Ant Colony	Ant Colony

Os gráficos de dispersão demonstrados pelas Figuras 6 e 7 indicam o tempo de execução pela quantidade de nós. Os algoritmos NN e GA demonstraram uma boa capacidade de lidar com instâncias maiores, enquanto que o AC se mostrou o mais sensível a este aumento. A Figura 8 apresenta os resultados obtidos pelo trabalho de referência, apontando o TS e AC como os piores nesta métrica, levando mais de 17 horas para resolver uma instância grande. Também indica o NN como o mais performático ao longo das instâncias, progredindo quase linearmente.

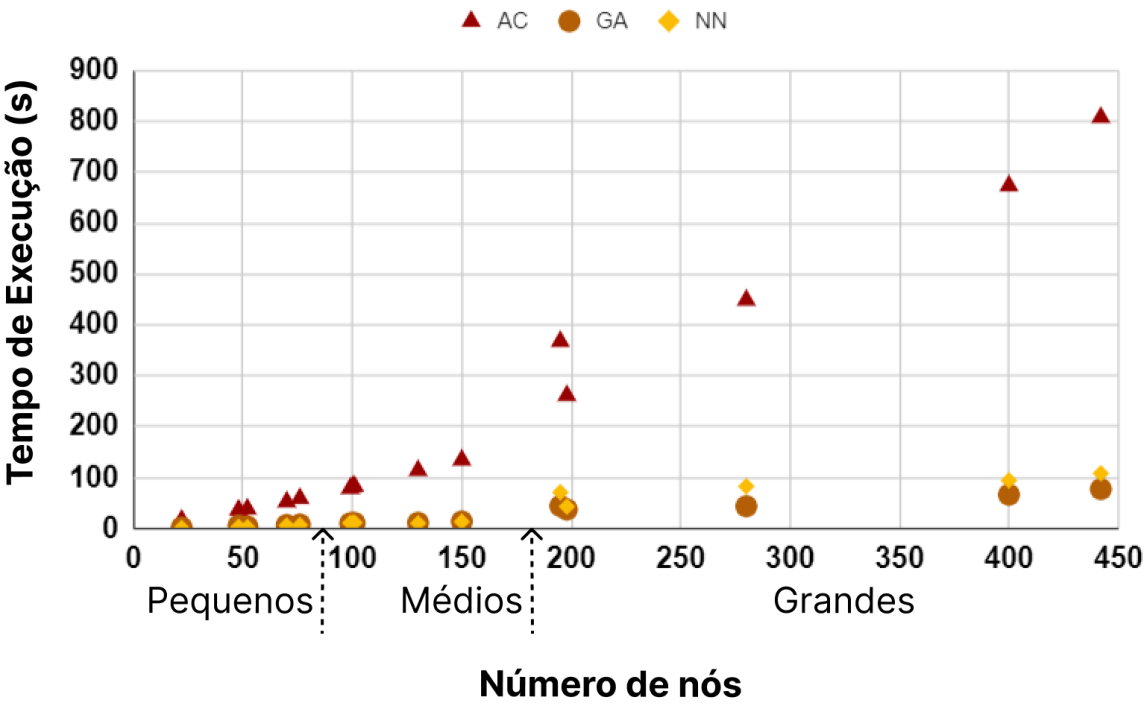

Figura 6. Gráfico de dispersão dos algoritmos NN, GA e AC

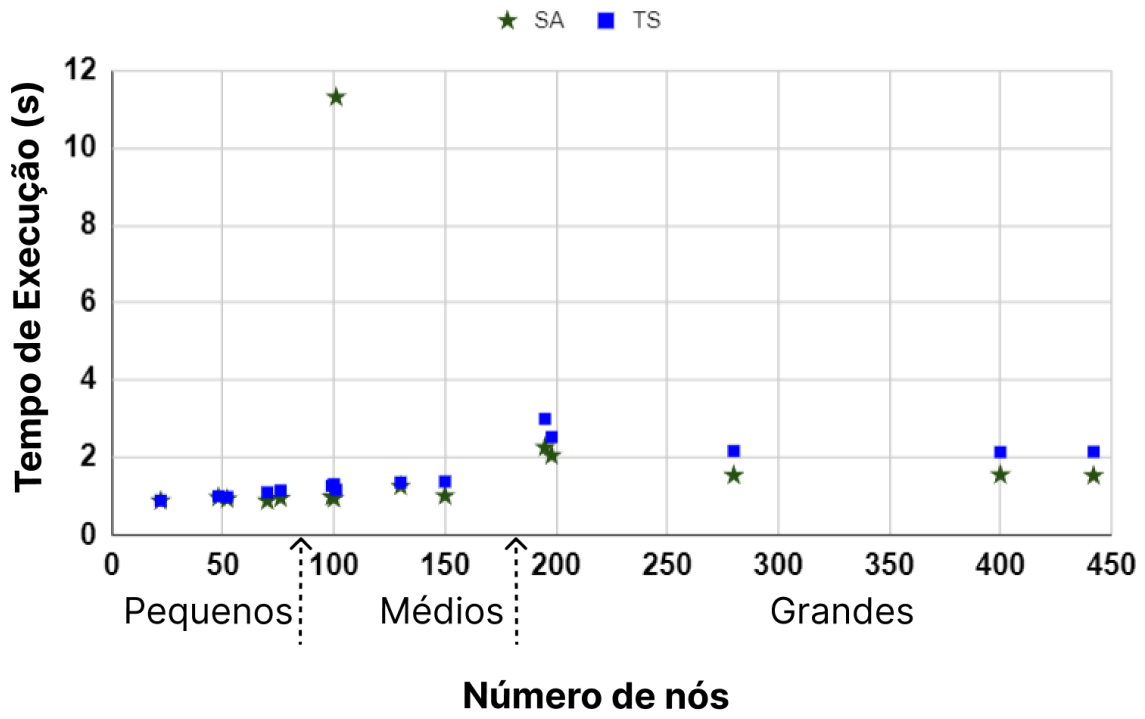

Figura 7. Gráfico de dispersão dos algoritmos TS e SA.

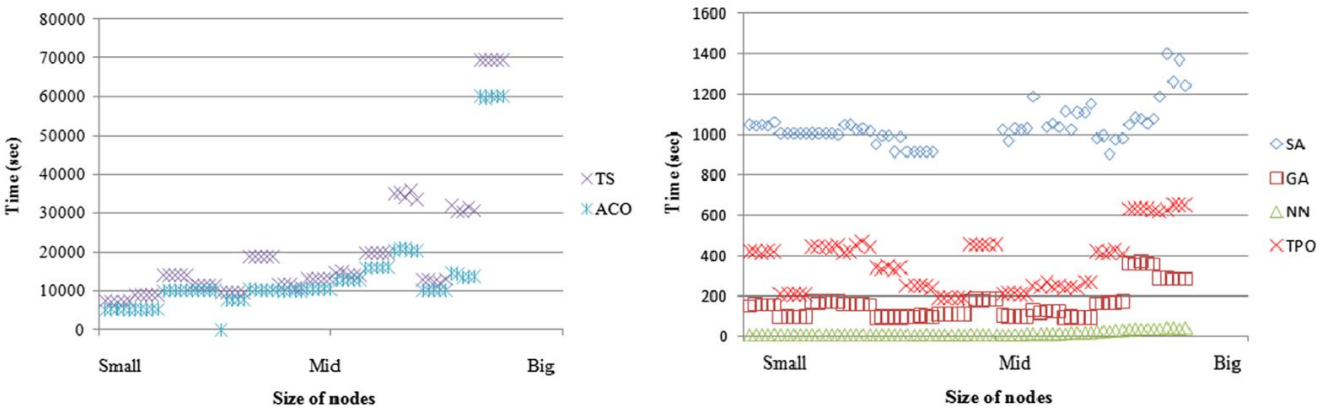

Figura 8. Gráficos de dispersão retirados do trabalho de Halim e Ismail, 2017.

5.2. Distância percorrida

A Tabela 6 apresenta os resultados estatísticos obtidos neste estudo, em relação à distância percorrida, e a Tabela 7 apresenta comparativos dos arquivos testados, neste trabalho com o trabalho de referência. Conforme os resultados expostos nestas 2 tabelas, embora mais simples, o algoritmo NN foi o que apresentou a melhor precisão média total, seguido pelo TS, SA, AC respectivamente. Isto está de acordo com os resultados encontrados pelo trabalho de referência em termos de proporção, os resultados no entanto, estão significativamente inferiores com a exceção do NN. Além disso, todos os valores obtidos pelo NN estão de acordo com o trabalho de referência.

A métrica de erro percentual apresentada na Tabela 6 possui a maior discrepância daquelas apresentadas pelo trabalho de Halim, também observado no comparativo apresentado na Tabela 7, o motivo disso se deve a 3 fatores fundamentais: Implementação, número de iterações e melhorias de performance. Também é possível notar nas tabelas que dentre todos os algoritmos, o GA foi o que apresentou os caminhos mais longos.

Tabela 6. Resultados estatísticos para a distância percorrida.

Arquivo	Código TSP	Algoritmo	Média	Erro Percentual	Desvio Padrão (σ)	Melhor Valor	Pior Valor
ulysses22 (75)	F1	nn	86,91	1,16%	**0,00**	86,91	86,91
		ts	90,90	1,21%	6,36	80,03	91,17
		ga	118,92	1,59%	8,95	108,47	**126,29**

		sa	**80,59**	**1,07%**	8,29	**79,73**	94,50
		ac	82,29	1,10%	0,56	81,81	82,93
att48 (33523)	F2	nn	**39.236,88**	**1,17%**	**0,00**	**39.236,88**	39.236,88
		ts	58.537,79	1,75%	8.613,10	57.022,07	72.640,39
		ga	129.602,56	3,87%	11.746,37	121.524,93	**144.668,6**1
		sa	59.069,75	1,76%	4.357,99	53.244,30	61.771,67
		ac	71.181,59	2,12%	896,28	69.714,14	71.339,42
berlin52 (7542)	F3	nn	**8.182,19**	**1,08%**	**0,00**	**8.182,19**	8.182,19
		ts	13.537,11	1,79%	559,63	12.810,08	13.910,56
		ga	25.306,69	3,36%	909,04	25.064,05	**26.745,79**
		sa	13.535,04	1,79%	1.047,93	12.433,18	14.528,09
		ac	16.210,25	2,15%	90,58	16.117,60	16.298,74
st70 (675)	F4	nn	**761,69**	**1,13%**	**0,00**	**761,69**	761,69
		ts	1.599,40	2,37%	102,22	1.489,98	1.694,24
		ga	3.341,84	4,95%	14,38	3.329,31	**3.357,99**
		sa	1.604,01	2,38%	49,61	1.598,09	1.686,83
		ac	1.764,27	2,61%	95,73	1.751,16	1.923,13
pr76 (108159)	F5	nn	**130.921,00**	**1,21%**	**0,00**	**130.921,00**	130.921,00
		ts	150.779,86	1,39%	**0,00**	150.779,86	150.779,86
		ga	502.821,13	4,65%	19.251,10	486.901,91	**525.221,83**
		sa	249.908,99	2,31%	9.728,04	238.347,84	257.680,50
		ac	312.147,53	2,89%	3.240,35	307.896,64	314.258,48
rat99 (1211)	F6	nn	**1.369,53**	**1,13%**	**0,00**	**1.369,53**	1.369,53
		ts	2.097,92	1,73%	10,16	2.084,06	2.103,86
		ga	7.324,62	6,05%	350,16	7.280,60	**7.907,90**
		sa	3.889,79	3,21%	164,65	3.767,22	4.093,19
		ac	4.221,14	3,49%	134,68	4.163,10	4.419,91

kroA100 (21282)	F7	nn	**24.698,50**	**16,05%**	**0,00**	**24698,5**	24.698,5
		ts	77.128,99	262,41%	9.474,15	70429,75	83.828,22
		ga	151.093,18	609,96%	1.502,76	150030,56	**152.155,7**9
		sa	80.962,56	280,43%	738,20	80440,57	81.484,54
		ac	78.356,28	268,18%	4.764,22	74987,46	81.725,09
eil101 (629)	F8	nn	**736,37**	**17,07%**	**0,00**	**736,37**	736,37
		ts	1.505,71	139,38%	71,18	1455,38	1556,04
		ga	3.095,36	392,11%	76,33	3041,38	**3.149,33**
		sa	1.772,71	181,83%	106,60	1697,33	1.848,09
		ac	2.036,45	223,76%	27,07	2017,31	2.055,59
ch130 (6110)	F9	nn	**7.198,74**	**17,82%**	**0,00**	**7198,74**	7.198,74
		ts	23.447,71	283,76%	65,61	23401,31	23.494,1
		ga	43.062,60	604,79%	726,38	42548,97	**43.576,22**
		sa	23.213,31	279,92%	589,97	22796,13	23.630,48
		ac	25.325,66	314,50%	445,69	25010,51	25.640,81
ch150 (6528)	F10	nn	**7.078,44**	**8,43%**	**0,00**	**7078,44**	7.078,44
		ts	27.568,50	322,31%	1.613,00	26427,94	28.709,06
		ga	49.487,94	658,09%	78,11	49432,71	**49.543,17**
		sa	27.511,77	321,44%	1.265,25	26617,1	28.406,43
		ac	30.074,87	360,71%	203,21	29931,18	30.218,56
rat195 (2323)	F11	nn	**2.628,56**	**13,15%**	**0,00**	**2.628,56**	2.628,56
		ts	4.025,17	73,27%	33,79	3.968,81	4.029,28
		ga	20.828,37	796,62%	189,38	20.734,05	**21.098,90**
		sa	11.868,80	410,93%	162,61	11.715,24	12.040,30
		ac	12.377,89	432,84%	103,11	12.366,67	12.550,61
d198 (15780)	F12	nn	**17.695,84**	**12,14%**	**0,00**	**17.695,84**	17.695,84
		ts	22.583,10	43,11%	172,73	22.349,25	22.686,39

		ga	175.946,29	1015,00%	1.799,04	175.514,50	**178.823,91**
		sa	97.052,99	515,04%	5.057,25	91.598,07	101.701,86
		ac	66.406,67	320,83%	868,16	66.286,92	67.846,92
a280 (2579)	F13	nn	3.094,28	19,98%	10,87	3.094,28	3.113,11
		ts	**2.818,62**	**9,29%**	**0,00**	**2.818,62**	2.818,62
		ga	32.351,55	1154,42%	560,99	31.904,00	**33.018,78**
		sa	19.134,45	641,93%	799,26	18.773,48	20.302,57
		ac	18.674,32	624,09%	189,37	18.393,78	18.754,41
rd400 (15281)	F14	nn	**18.303,30**	**19,78%**	**0,00**	**18.303,30**	18.303,30
		ts	137.276,57	798,35%	1.242,24	136.997,05	139.274,77
		ga	199.368,99	1204,69%	2.499,20	197.730,69	**202.639,4**5
		sa	136.492,34	793,22%	6.162,21	130.350,22	142.674,62
		ac	124.506,25	714,78%	1.356,21	122.858,37	125.548,14
pcb442 (50778)	F15	nn	**58.952,97**	**16,10%**	**0,00**	**58.952,97**	58.952,97
		ts	213.188,63	319,84%	6.049,96	203.147,75	214.015,56
		ga	725.491,21	1328,75%	4.531,35	721.727,50	**730.749,05**
		sa	505.713,11	895,93%	11.750,31	492.530,90	515.970,79
		ac	459.299,79	804,53%	2.425,26	455.870,96	460.556,56

Tabela 7. Comparativo entre os valores de destaque obtidos para cada arquivo, encontrados no trabalho atual, e em Halim e Ismail (2017).

Arquivo	Trabalho	Algoritmo (s)	Média	Erro Percentual	Desvio Padrão (σ)	Melhor Valor	Pior Valor
ulysses2*	Atual	sa/nn/ac/ga	80,59 (sa)	1,07% (sa)	0,00 (nn)	79,73 (sa)	126,29 (ga)
	Halim e Ismail (2017)	-	-	-	-	-	-
att48*	Atual	nn/ga	39.236,88 (nn)	1,17% (nn)	0,00 (nn)	39.236,88 (nn)	144.668,61 (ga)

	Halim e Ismail (2017)	-	-	-	-	-	-
berlin52	Atual	nn/ga	8.182,19 (nn)	1,08% (nn)	0,00 (nn)	8.182,19 (nn)	26.745,79 (ga)
	Halim e Ismail (2017)	tpo/nn/ga	7.705,80 (tpo)	2,17% (tpo)	1,66 (nn)	7544,00 (tpo)	8.269,00 (ga)
st70	Atual	nn/ga	761,69 (nn)	1,13% (nn)	0,00 (nn)	761,69 (nn)	3.357,99 (ga)
	Halim e Ismail (2017)	ts/nn	690,27 (ts)	2,26% (ts)	0,91 (nn)	680,99 (ts)	762,99 (nn)
pr76	Atual	nn/ga	130.921,0 (nn)	1,21% (nn)	0,00 (nn)	130.921,0 0 (nn)	525.221,83 (ga)
	Halim e Ismail (2017)	ts/nn	109.930,1 (ts)	1,64 % (ts)	688,21 (ts)	109.046,2 5 (ts)	130.923,18 (nn)
rat99	Atual	nn/ga	1.369,53 (nn)	1,13% (nn)	0,00 (nn)	1.369,53 (nn)	7.907,90 (ga)
	Halim e Ismail (2017)	ts/sa/nn	1.243,47 (ts)	2,68% (ts)	0,85 (sa)	1.233,45 (ts)	1.370,53 (nn)
kroA100	Atual	nn/ga	24.698,50 (nn)	16,05% (nn)	0,00 (nn)	24.698,5 (nn)	24.699,54 (ga)
	Halim e Ismail (2017)	sa/nn/tpo	22.277,50 (sa)	4,68% (sa)	1.65 (nn)	21.795,00 (tpo)	24,699.54 (nn)
eil101	Atual	nn/ga	736,37 (nn)	17,07% (nn)	0,00 (nn)	736,37 (nn)	3.149,33 (ga)
	Halim e Ismail (2017)	ts/nn/tpo	667,61 (ts)	6,14% (ts)	0,40 (nn)	658,66 (tpo)	736,37 (nn)
ch130	Atual	nn/ga	7.198,74 (nn)	17,82% (nn)	0,00 (nn)	7.198,74 (nn)	43.576,22 (ga)
	Halim e	tpo/nn/ts	6515,28	6,63%	1,95 (nn)	6.214,81	7.334,39

	Ismail (2017)		(tpo)	(tpo)		(ts)	(tpo)
ch150	Atual	nn/ga	7.078,44 (nn)	8,43% (nn)	0,00 (nn)	7078,44 (nn)	49.543,17 (ga)
	Halim e Ismail (2017)	ts/nn/aco	6.862,34 (ts)	5,12% (ts)	1,09 (nn)	6.616,01 (ts)	7.370,45 (aco)
rat195	Atual	nn/ga	2.628,56 (nn)	13,15% (nn)	0,00 (nn)	2.628,56 (nn)	21.098,90 (ga)
	Halim e Ismail (2017)	ts/nn	2.373,94 (ts)	2,19% (ts)	1,57 (nn)	2.359,36 (ts)	2.656,84 (tpo)
d198	Atual	nn/ga	17.695,84 (nn)	12,14% (nn)	0,00 (nn)	17.695,84 (nn)	178.823,9 (ga)
	Halim e Ismail (2017)	ts/nn/sa/ aco	16.083,48 (ts)	1,92% (ts)	0,84 (nn)	16.035,16 (sa)	18.206,47 (aco)
a280	Atual	ts/ga	2.818,62 (ts)	9,29% (ts)	0,00 (ts)	2.818,62 (ts)	33.018,78 (ga)
	Halim e Ismail (2017)	ga/nn/aco/ sa	2.789,83 (ga)	8,82% (ga)	0,43 (nn)	2.733,74 (aco)	2.976,77 (sa)
rd400	Atual	nn/ga	18.303,30 (nn)	19,78% (nn)	0,00 (nn)	18.303,30 (nn)	202.639,45 (ga)
	Halim e Ismail (2017)	ga/nn/ts	16.567,29 (ga)	8,42% (ga)	1,96 (nn)	16.346,38 (nn)	20.739,47 (ts)
pcb442	Atual	nn/ga	58.952,97 (nn)	16,10% (nn)	0,00 (nn)	58.952,97 (nn)	730.749,05 (ga)
	Halim e Ismail (2017)	ga/nn/ts	55.718,90 (ga)	9,73% (ga)	2,18 (nn)	54.424,78 (ga)	83,172.00 (ts)
***Arquivos não utilizados por Halim e Ismail (2017)**							

5.3. Uso de memória

A Figura 9 mostra o consumo de memória exigido por cada algoritmo durante o aumento do número de nós. Em termos de eficiência, o SA e TS foram os melhores ao apresentar um leve aumento no consumo de memória conforme o tamanho do problema aumenta. O algoritmo NN e GA mostraram um aumento suave no uso de memória para problemas maiores, enquanto que o AC apresentaram um consumo severo de memória, exibindo um aumento mais acentuado com problemas grandes.

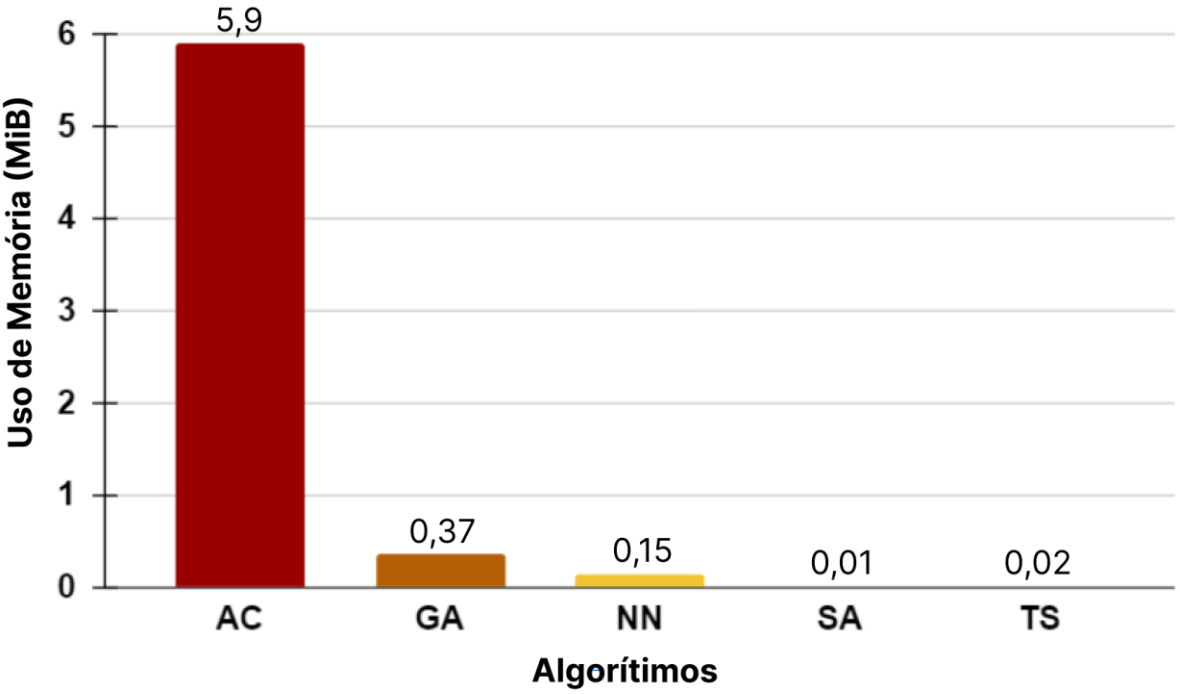

Figura 9. Gráfico comparativo do uso de memória médio.

5.4. Complexidade computacional

Os resultados da métrica de complexidade computacional é uma estimativa com base em um modelo de curva polinomial do tipo $T(n) = a \cdot n^b$, onde $T(n)$ é o tempo de execução, n é o número de nós, a é o coeficiente e b é o expoente que indica a taxa de crescimento do tempo de execução, em relação ao número de nós.

A Tabela 8 apresenta o resultado obtido por cada algoritmo, indicando que SA e TS são mais eficientes em termos de tempo de execução para grandes conjuntos de cidades, enquanto AC tem uma variação maior, sendo altamente eficiente em instâncias médias, mas piora em instâncias grandes. Por fim, NN e GA mostram piora no desempenho para grandes instâncias.

A análise da complexidade assintótica média revela que, em geral, os algoritmos possuem um expoente b negativo, o que indica um comportamento incomum no desempenho. Esse comportamento deve-se ao número limitado de iterações em cada algoritmo, o que impacta a precisão da análise para instâncias grandes. Observou-se também que algoritmos como NN e GA mostram um aumento no tempo de execução conforme o tamanho das instâncias aumenta, refletindo uma tendência menos eficiente para escalas maiores.

Tabela 8. Complexidade assintótica

Algoritmo	**Complexidade assintótica - instância pequena**	**Complexidade assintótica - instância média**	**Complexidade assintótica - instância grande**	**Complexidade assintótica média**
Simulated Annealing	$O(n^{-0,05})$	$O(n^{-0,41})$	$O(n^{0,06})$	$O(n^{-0,13})$
Tabu Search	$O(n^{0,02})$	$O(n^{-0,22})$	$O(n^{0,09})$	$O(n^{-0,04})$
Nearest Neighbor	$O(n^{0,12})$	$O(n^{-0,16})$	$O(n^{0,3})$	$O(n^{0,09})$
Ant Colony	$O(n^{0,02})$	$O(n^{-1,19})$	$O(n^{0,4})$	$O(n^{-0,26})$
Genetic Algorithm	$O(n^{-0,08})$	$O(n^{-0,46})$	$O(n^{0,34})$	$O(n^{-0,07})$

6. Conclusões

Este trabalho consistiu em comparar cinco dos principais algoritmos probabilísticos encontrados na literatura para o problema do caixeiro viajante. São eles: *Nearest Neighbor* (NN), *Genetic Algorithm* (GA), *Ant Colony* (AC), *Tabu Search* (TS) e *Simulated Annealing* (SA). Estes mesmos algoritmos foram usados por Halim and Ismail (2017), trabalho no qual foi utilizado como comparativo no atual estudo.

O trabalho atual obteve resultados que diferem em parte daqueles encontrados no de referência. Como exemplo, os algoritmos TS e AC foram descritos como os mais lentos, enquanto neste estudo, o TS foi o segundo mais rápido embora sua taxa de precisão seja inferior aos demais. Os resultados apontaram que instâncias menores foram resolvidas com mais agilidade por algoritmos mais simples que os robustos (NN), enquanto instâncias médias e

grandes foram melhor resolvidas (custo benefício) por algoritmos como SA e TS. Desta forma, é possível notar que algoritmos probabilísticos possuem uma filosofia própria e dificilmente podem ser comparados utilizando os mesmos parâmetros. Este experimento mostrou que um mínimo ajuste gera impactos significativos na escolha do resultado.

7. Trabalhos futuros

Os próximos passos deste estudo é investigar uma hipótese sobre o uso de parâmetros proporcionais à instância a ser testada. Para isso, serão fornecidos insumos e reflexões sobre o comportamento dos algoritmos estudados, comparando-os de acordo com a complexidade dos testes. Em boa parte dos trabalhos que realizaram este tipo de comparação, os autores optaram por padronizar os parâmetros de todos os algoritmos de forma semelhante. Dar condições semelhantes a um experimento é o procedimento científico padrão. No entanto, analisar algoritmos sem parâmetros padronizados pode ser também uma prática interessante em alguns casos [Sudholt, 2009]. A quantidade de iterações, pode não ser tão bem aproveitada por um algoritmo, como o *Nearest Neighbor*. Por isso, regulando o volume de iterações pode-se tornar o algoritmo mais rápido sem comprometer o índice de precisão.

Valores de parâmetros podem afetar negativamente a performance de um algoritmo, enquanto beneficiam outro [Yang et al. 2013]. Por se tratar de mecanismos de busca distintos, é necessário respeitar a dinâmica de um algoritmo e ter ciência que, do ponto de vista prático, nem sempre valerá a pena o uso do mesmo parâmetro.

Referências

ABDULLAH, Nor Ain Shafiqah; RAZALI, Siti Noor Asyikin Mohd. Optimal Parameter Value of Genetic Algorithms for Different Size Instances in Solving Traveling Salesman Problem. **Enhanced Knowledge in Sciences and Technology**, v. 4, n. 1, p. 171-182, 2024.

BALAMURUGAN, R.; NATARAJAN, A. M.; PREMALATHA, K. Stellar-mass black hole optimization for biclustering microarray gene expression data. **Applied Artificial Intelligence**, v. 29, n. 4, p. 353-381, 2015.

BLUM, Christian; ROLI, Andrea. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM computing surveys (CSUR)**, v. 35, n. 3, p. 268-308, 2003.

BROWNLEE, Jason *et al*. **A note on research methodology and benchmarking optimization algorithms. Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID**, v. 70125, 2007.

CHEIKHROUHOU, Omar; KHOUFI, Ines. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy. **Computer Science Review**, v. 40, p. 100369, 2021.

CORDEAU, Jean-Françcois; LAPORTE, Gilbert; ROPKE, Stefan. **Recent models and algorithms for one-to-one pickup and delivery problems**. Boston: Springer US, 2008.

DA SILVA, Fabio QB *et al*. Replication of empirical studies in software engineering research: a systematic mapping study. **Empirical Software Engineering**, v. 19, p. 501-557, 2014.

DAS, Arnab; CHAKRABARTI, Bikas K. (Ed.). **Quantum annealing and related optimization methods**. Berlin: Springer Science & Business Media, 2005.

DIESTEL, Reinhard. **Graph theory**. Springer (print edition); Reinhard Diestel (eBooks), 2024.

DORIGO, Marco; DI CARO, Gianni; GAMBARDELLA, Luca M. Ant algorithms for discrete optimization. **Artificial life**, v. 5, n. 2, p. 137-172, 1999.

ERTEN, H. Irem; DEVECI, H. Arda; ARTEM, H. Seçil. **Stochastic optimization methods. In: Designing engineering structures using stochastic optimization methods**. CRC Press, 2020.

FIGUEIRA, José Rui et al. Easy to say they are hard, but hard to see they are easy—towards a categorization of tractable multiobjective combinatorial optimization problems. **Journal of Multi-Criteria Decision Analysis**, v. 24, n. 1-2, p. 82-98, 2017.

FREIVALDS, Rūsiņš. Fast probabilistic algorithms. In: **International Symposium on Mathematical Foundations of Computer Science**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979. p. 57-6.

GHANNADI, Parsa; KOUREHLI, Seyed Sina; MIRJALILI, Seyedali. A review of the application of the simulated annealing algorithm in structural health monitoring (1995-2021). **Frattura ed Integrità Strutturale**, v. 17, n. 64, p. 51-76, 2023.

HALIM, A. H.; ISMAIL, I. Combinatorial optimization: comparison of heuristic algorithms in travelling salesman problem. **Archives of Computational Methods in Engineering**, v. 26, p. 367-380, 2019.

HALIM, A. Hanif; ISMAIL, Idris; DAS, Swagatam. Performance assessment of the metaheuristic optimization algorithms: an exhaustive review. Artificial Intelligence Review, v. 54, n. 3, p. 2323-2409, 2021.

HASSAN, Rania et al. A comparison of particle swarm optimization and the genetic algorithm. In: **46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference**. 2005. p. 1897.

HOFFMAN, Karla L. et al. Traveling salesman problem. Encyclopedia of operations research and management science, v. 1, p. 1573-1578, 2013.

HOOS, Holger H.; STÜTZLE, Thomas. **Stochastic local search: Foundations and applications.** Elsevier, 2004.

HUSNAIN, Ghassan; ANWAR, Shahzad. An intelligent probabilistic whale optimization algorithm (i-WOA) for clustering in vehicular ad hoc networks. **International Journal of Wireless Information Networks**, v. 29, n. 2, p. 143-156, 2022.

JAFFE, Arthur M. The millennium grand challenge in mathematics. **Notices of the AMS**, v. 53, n. 6, p. 652-660, 2006.

JAHWAR, Alan Fuad; ABDULAZEEZ, Adnan Mohsin. Meta-heuristic algorithms for K-means clustering: A review. **PalArch's Journal of Archaeology of Egypt/Egyptology**, v. 17, n. 7, p. 12002-12020, 2020.

KARABOGA, Dervis; AKAY, Bahriye. A comparative study of artificial bee colony algorithm. **Applied mathematics and computation**, v. 214, n. 1, p. 108-132, 2009.

KAUR, Navjot; CHAUDHARY, Deepika; SINGH, Jaiteg. Optimization of neural networks through classical metaheuristic algorithms: A review of past decade. In: **AIP Conference Proceedings**. AIP Publishing, 2023.

KAUR, Shaminder; VERMA, Amandeep. An efficient approach to genetic algorithm for task scheduling in cloud computing environment. **International Journal of Information Technology and Computer Science (IJITCS)**, v. 4, n. 10, p. 74-79, 2012.

KAUR, Sukhpreet et al. A systematic review on metaheuristic optimization techniques for feature selections in disease diagnosis: open issues and challenges. **Archives of Computational Methods in Engineering**, v. 30, n. 3, p. 1863-1895, 2023.

KRAMER, Oliver. **Genetic algorithms,** vol 679. Springer International Publishing, 2017.

KUMAR, Rajiv; MEMORIA, Minakshi. A review of memetic algorithm and its application in traveling salesman problem. **Int. J. Emerg. Technol**., v. 11, n. 2, p. 1110-1115, 2020.

LAMBORA, Annu; GUPTA, Kunal; CHOPRA, Kriti. Genetic algorithm-A literature review. In: 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon). **IEEE**, 2019. p. 380-384.

LARNI-FOOEIK, A.; GHASEMI, N.; MOHAMMADI, E. Insights into the application of the traveling salesman problem to logistics without considering financial risk: A bibliometric study. **Management Science Letters**, v. 14, n. 3, p. 189-200, 2024.

Long, Ngoc Nguyen et al. Performance evaluation of the artificial hummingbird algorithm in the problem of structural damage identification. **Tạp chí Khoa học Giao thông vận tải**, v. 74, n. 4, p. 413-427, 2023. https://doi.org/10.47869/tcsj.74.4.3

MEDINA-GONZALEZ, Sergio et al. A graph theory approach for scenario aggregation for stochastic optimisation. **Computers & Chemical Engineering**, v. 137, p. 106810, 2020.

OSABA, Eneko; YANG, Xin-She; DEL SER, Javier. Traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics. **Nature-inspired computation and swarm intelligence**, p. 135-164, 2020.

PEARL, Judea. **Heuristics: intelligent search strategies for computer problem solving**. Addison-Wesley Longman Publishing Co., Inc., 1984.

POP, Petrică C. et al. A comprehensive survey on the generalized traveling salesman problem. **European Journal of Operational Research**, v. 314, n. 3, 2023.

PRAJAPATI, Vishnu Kumar; JAIN, Mayank; CHOUHAN, Lokesh. Tabu search algorithm (TSA): A comprehensive survey. In: **2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE).** IEEE, 2020. p. 1-8.

RAHMAN, M. Sohel; KAYKOBAD, Mohammad. On Hamiltonian cycles and Hamiltonian paths. **Information Processing Letters**, v. 94, n. 1, p. 37-41, 2005.

SAHALOT, Antima; SHRIMALI, Sapna. A comparative study of brute force method, nearest neighbour and greedy algorithms to solve the travelling salesman problem. International **Journal of Research in Engineering & Technology**, v. 2, n. 6, p. 59-72, 2014.

SARIKYRIAKIDIS, Stavros; GOULIANAS, Konstantinos; MARGARIS, Athanasios I. Using Self-organizing Maps to Solve the Travelling Salesman Problem: A Review. **WSEAS Transactions on Systems**, v. 22, p. 131-159, 2023.

SATHYA, N.; MUTHUKUMARAVEL, A. A review of the optimization algorithms on traveling salesman problem. **Indian Journal of Science and Technology**, v. 8, n. 29, p. 1-4, 2015.

SHEPPERD, Martin. Replication studies considered harmful. In: **Proceedings of the 30th international conference on software engineering.** 2008. p. 411-421.

SI, Binghui et al. Performance indices and evaluation of algorithms in building energy efficient design optimization. **Energy**, v. 114, p. 100-112, 2016.https://doi.org/10.1016/j.energy.2016.07.114

SPALL, James C. **Introduction to stochastic search and optimization: estimation, simulation, and control**. New Jersey: John Wiley & Sons, 2005.

SUDHOLT, Dirk. The impact of parametrization in memetic evolutionary algorithms. **Theoretical Computer Science**, v. 410, n. 26, p. 2511-2528, 2009.

TARIM, S. Armagan. Benders decomposition for stochastic constraint programming. **Annals of Operations Research**, v. 177, n. 1, p. 253-267, 2010.

Universität Heidelberg. **List**, 2018. disponível em: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/. Acessado em 21/02/024.

VAZIRI, Seyed Mohammad et al. Whale optimization algorithm based on Lévy flight and proposed clustering approach for large-scale service composition problems in cloud computing. **Future Generation Computer Systems**, v. 102, p. 238-252, 2020.

YANG, Xin-She et al. A framework for self-tuning optimization algorithm. **Neural Computing and Applications**, v. 23, p. 2051-2057, 2013.

WANG, Ning et al. An overview of routing optimization for internet traffic engineering. **IEEE Communications Surveys & Tutorials**, v. 10, n. 1, p. 36-56, 2008.http://doi.org/10.1109/COMST.2008.4483669

WANG, Shijin; ZHAO, Qianyang. Probabilistic tabu search algorithm for container liner shipping problem with speed optimisation. **International Journal of Production Research**, v. 60, n. 12, p. 3651-3668, 2022. https://doi.org/10.1080/00207543.2021.1930236

WEBB, Helena et al. "It would be pretty immoral to choose a random algorithm" Opening up algorithmic interpretability and transparency. **Journal of Information, Communication and Ethics in Society**, v. 17, n. 2, p. 210-228, 2019. https://doi.org/10.1108/JICES-11-2018-0092

WETTER, Michael; WRIGHT, Jonathan. A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization. **Building and environment,** v. 39, n. 8, p. 989-999, 2004. https://doi.org/10.1016/j.buildenv.2004.01.022