UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
CÂMPUS CIDADE UNIVERSITÁRIA
FACULDADE DE COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Matheus Kazumi Silva Miyashiro

Análise do consumo de recursos na execução dos algoritmos Dijkstra e A* em nós simulados de névoa e nuvem para o gerenciamento de tráfego com semáforos inteligentes

Campo Grande - MS
2025

Matheus Kazumi Silva Miyashiro

Análise do consumo de recursos na execução dos algoritmos Dijkstra e A* em nós simulados de névoa e nuvem para o gerenciamento de tráfego com semáforos inteligentes

Trabalho de Conclusão de Curso de Graduação em Engenharia de Computação, da Faculdade de Computação da Universidade Federal de Mato Grosso do Sul, apresentado como requisito para a obtenção do título de Engenheiro de Computação.
Orientador: Prof. Dr. Dionisio Machado Leite Filho.

Campo Grande - MS
2025

Este trabalho é dedicado aos meus pais e à minha irmã,
pelo amor, apoio incondicional e por sempre acreditarem
em mim.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus por permitir meu ingresso em uma universidade federal e por me conceder forças, sabedoria e saúde ao longo dos cinco anos de graduação.

Aos meus pais, Eva e Altamiro, por todo suporte, empatia, paciência e amor. Por acreditarem em mim em todos os momentos e por sempre me incentivarem a buscar voos mais altos. Obrigado por serem meu porto seguro, sem vocês nada disso seria possível.

À minha irmã, Larissa, pela parceria e companheirismo, por acreditar no meu potencial mesmo nas minhas inseguranças. Por cada conselho, palavra de incentivo, demonstração de carinho e apoio constante ao longo da graduação. Sua presença firme e afetuosa fez toda a diferença nessa caminhada.

Aos demais familiares que acompanharam a minha trajetória, em especial aos meus avós e à tia Maria, que acreditaram que essa realização seria possível e por sempre me incentivarem.

Aos amigos que fiz ao longo da graduação, por todos os momentos vividos juntos. A ajuda mútua, as aulas compartilhadas, as conversas e risadas nos corredores e os momentos no LIA tornaram essa jornada mais leve e inesquecível.

Aos demais amigos, pelo apoio em todos os momentos, pelas palavras de incentivo, pela paciência nas horas difíceis e por celebrarem comigo cada pequena conquista.

Ao meu orientador, Prof. Dr. Dionisio Machado Leite Filho, pela orientação dedicada, paciência e sugestões valiosas durante todo o desenvolvimento deste trabalho.

À Prof. Dra. Bianca de Almeida Dantas, coordenadora do curso de Engenharia de Computação, pela organização e suporte. Sua dedicação à qualidade do curso e ao bem-estar dos alunos foi essencial para minha formação.

Aos docentes do curso de Engenharia de Computação, pela seriedade com que conduzem o ensino, pelo apoio e por compartilharem seus conhecimentos com tanto comprometimento. Cada disciplina contribuiu de maneira significativa para minha trajetória acadêmica, pessoal e para a construção da minha formação profissional.

À Universidade Federal de Mato Grosso do Sul, pela excelência no ensino, pesquisa e extensão. Por proporcionar um ambiente que integra ciência e sociedade, promovendo o pensamento crítico, a inovação e o desenvolvimento integral de seus alunos, tanto no aspecto acadêmico quanto humano.

Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a se fazer.

Alan Turing

RESUMO

Este trabalho tem como objetivo analisar o desempenho e o consumo de recursos de nós simulados de computação em névoa (*fog*) e em nuvem (*cloud*), responsáveis pelo gerenciamento do tráfego de veículos em ambientes urbanos com semáforos inteligentes, por meio do cálculo de rotas de menor caminho. A metodologia empregada envolveu o uso de contêineres Docker para representar os ambientes de névoa, com recursos de hardware limitados, e de nuvem, com capacidade computacional ampliada. Os algoritmos Dijkstra e A* foram utilizados como carga de trabalho nesses cenários para determinar o menor caminho entre pontos previamente definidos, além de avaliar seu desempenho e o consumo de recursos computacionais durante a execução. Foram coletados dados referentes ao tempo de execução dos algoritmos, consumo de CPU e memória RAM, além do tráfego de entrada e saída de dados na interface de rede dos contêineres, com o intuito de verificar qual algoritmo é mais adequado para cada contexto com base nos critérios definidos. Os resultados indicam que o algoritmo A* é mais adequado ao contexto de névoa, devido à sua constância na execução, e também ao contexto de nuvem, por proporcionar maior economia de recursos.

Palavras-chave: computação em névoa; computação em nuvem; gerenciamento de tráfego; semáforos inteligentes; contêineres Docker.

ABSTRACT

This study aims to analyze the performance and resource consumption of simulated fog and cloud computing nodes responsible for managing vehicle traffic in urban environments with smart traffic lights, through the calculation of shortest paths. The methodology involved the use of Docker containers to represent the fog environment, with limited hardware resources, and the cloud environment, with expanded computational capacity. The Dijkstra and A* algorithms were used as workloads in these scenarios to determine the shortest path between predefined points, as well as to evaluate their performance and computational resource usage during execution. Data were collected regarding the algorithms' execution time, CPU and RAM usage, and the inbound and outbound traffic on the containers' network interfaces, in order to identify which algorithm is more suitable for each context based on defined criteria. The results indicate that the A* algorithm is more suitable for the fog computing context due to its consistent performance, and also for the cloud context, for offering greater resource efficiency.

Keywords: fog computing; cloud computing; traffic management; smart traffic lights; Docker containers.

LISTA DE FIGURAS

Figura 1 – Infraestrutura inteligente em perspectiva 17
Figura 2 – Comunicação entre dispositivos IoT no setor de transportes 18
Figura 3 – Organização estrutural da computação em nuvem 19
Figura 4 – Arquitetura em camadas da computação em névoa 21
Figura 5 – Função para a geração de grafos com arestas e pesos aleatórios 28
Figura 6 – Função de execução dos algoritmos de caminho mínimo 30
Figura 7 – Função de controle de execução do experimento 31
Figura 8 – Função principal do experimento 32
Figura 9 – Função de seleção das sementes válidas para as simulações 33
Figura 10 – Script de captura das métricas de desempenho via docker stats 36
Figura 11 – Tempo de execução por número de arestas na topologia 2000 - Névoa . 39
Figura 12 – Tempo de execução por número de arestas na topologia 3000 - Névoa . 40
Figura 13 – Tempo de execução por número de arestas na topologia 4000 - Névoa . 41
Figura 14 – Tempo de execução por número de arestas na topologia 5000 - Névoa . 42
Figura 15 – Tempo de execução por número de arestas na topologia 6000 - Névoa . 44
Figura 16 – Tempo de execução por número de arestas na topologia 2000 - Nuvem . 45
Figura 17 – Tempo de execução por número de arestas na topologia 3000 - Nuvem . 46
Figura 18 – Tempo de execução por número de arestas na topologia 4000 - Nuvem . 47
Figura 19 – Tempo de execução por número de arestas na topologia 5000 - Nuvem . 48
Figura 20 – Tempo de execução por número de arestas na topologia 6000 - Nuvem . 49
Figura 21 – Tempo de execução médio por topologia - Névoa 50
Figura 22 – Tempo de execução médio por topologia - Nuvem 51
Figura 23 – Consumo médio de CPU por topologia - Névoa 68
Figura 24 – Consumo médio de memória RAM por topologia - Névoa 68
Figura 25 – Dados recebidos pela interface de rede do contêiner por topologia - Névoa 69
Figura 26 – Dados enviados pela interface de rede do contêiner por topologia - Névoa 69
Figura 27 – Consumo médio de CPU por topologia - Nuvem 70
Figura 28 – Consumo médio de memória RAM por topologia - Nuvem 71
Figura 29 – Dados recebidos pela interface de rede do contêiner por topologia - Nuvem 71
Figura 30 – Dados enviados pela interface de rede do contêiner por topologia - Nuvem 72

LISTA DE TABELAS

Tabela 1 – Síntese dos trabalhos relacionados 26
Tabela 2 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 2000 nós no contêiner de névoa 38
Tabela 3 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 3000 nós no contêiner de névoa 40
Tabela 4 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 4000 nós no contêiner de névoa 41
Tabela 5 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 5000 nós no contêiner de névoa 42
Tabela 6 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 6000 nós no contêiner de névoa 43
Tabela 7 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 2000 nós no contêiner de nuvem 45
Tabela 8 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 3000 nós no contêiner de nuvem 46
Tabela 9 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 4000 nós no contêiner de nuvem 47
Tabela 10 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 5000 nós no contêiner de nuvem 48
Tabela 11 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 6000 nós no contêiner de nuvem 49
Tabela 12 – Tempos de execução médios dos algoritmos 50
Tabela 13 – Consumo médio por semente na topologia 2000 (Dijkstra/Névoa) . . . 52
Tabela 14 – Consumo médio por semente na topologia 2000 (A*/Névoa) 52
Tabela 15 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 2000 - Névoa 53
Tabela 16 – Coeficientes de variação das métricas de desempenho na topologia 2000 - Névoa 53
Tabela 17 – Consumo médio por semente na topologia 3000 (Dijkstra/Névoa) . . . 54
Tabela 18 – Consumo médio por semente na topologia 3000 (A*/Névoa) 54
Tabela 19 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 3000 - Névoa 54
Tabela 20 – Coeficientes de variação das métricas de desempenho na topologia 3000 - Névoa 55
Tabela 21 – Consumo médio por semente na topologia 4000 (Dijkstra/Névoa) . . . 55
Tabela 22 – Consumo médio por semente na topologia 4000 (A*/Névoa) 56

Tabela 23 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 4000 - Névoa 56
Tabela 24 – Coeficientes de variação das métricas de desempenho na topologia 4000 - Névoa 56
Tabela 25 – Consumo médio por semente na topologia 5000 (Dijkstra/Névoa) . . . 57
Tabela 26 – Consumo médio por semente na topologia 5000 (A*/Névoa) 57
Tabela 27 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 5000 - Névoa 58
Tabela 28 – Coeficientes de variação das métricas de desempenho na topologia 5000 - Névoa 58
Tabela 29 – Consumo médio por semente na topologia 6000 (Dijkstra/Névoa) . . . 59
Tabela 30 – Consumo médio por semente na topologia 6000 (A*/Névoa) 59
Tabela 31 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 6000 - Névoa 59
Tabela 32 – Coeficientes de variação das métricas de desempenho na topologia 6000 - Névoa 60
Tabela 33 – Consumo médio por semente na topologia 2000 (Dijkstra/Nuvem) . . . 61
Tabela 34 – Consumo médio por semente na topologia 2000 (A*/Nuvem) 61
Tabela 35 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 2000 - Nuvem 61
Tabela 36 – Consumo médio por semente na topologia 3000 (Dijkstra/Nuvem) . . . 62
Tabela 37 – Consumo médio por semente na topologia 3000 (A*/Nuvem) 62
Tabela 38 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 3000 - Nuvem 63
Tabela 39 – Consumo médio por semente na topologia 4000 (Dijkstra/Nuvem) . . . 63
Tabela 40 – Consumo médio por semente na topologia 4000 (A*/Nuvem) 64
Tabela 41 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 4000 - Nuvem 64
Tabela 42 – Consumo médio por semente na topologia 5000 (Dijkstra/Nuvem) . . . 65
Tabela 43 – Consumo médio por semente na topologia 5000 (A*/Nuvem) 65
Tabela 44 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 5000 - Nuvem 65
Tabela 45 – Consumo médio por semente na topologia 6000 (Dijkstra/Nuvem) . . . 66
Tabela 46 – Consumo médio por semente na topologia 6000 (A*/Nuvem) 66
Tabela 47 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 6000 - Nuvem 67
Tabela 48 – Consumo médio de recursos por topologia no ambiente de névoa 67
Tabela 49 – Consumo médio de recursos por topologia no ambiente de nuvem . . . 70

LISTA DE ABREVIATURAS E SIGLAS

BS	Base Stations
CPU	Central Processing Unit
EDTLCM	Efficient Dynamic Traffic Light Control algorithm for Multiple
FOX	Fast Offset Xpath
IoT	Internet of Things
IoV	Internet of Vehicles
ITCMS	Intelligent Traffic Congestion Management System
KNN	K-Nearest Neighbors
LED	Light Emitting Diode
LTS	Long Term Support
RAM	Random Access Memory
RSU	Roadside Units
SDN	Software Defined Networking
STLS	Smart Traffic Lights System
SUMO	Simulation of Urban Mobility
TIC	Tecnologia da Informação e Comunicação
VFC	Vehicular Fog Computing

SUMÁRIO

1 INTRODUÇÃO 14
1.1 OBJETIVOS 15
1.1.1 Objetivo Geral 15
1.1.2 Objetivos Específicos 16
2 REFERENCIAL TEÓRICO 17
2.1 CIDADES INTELIGENTES 17
2.2 INTERNET DAS COISAS 18
2.3 COMPUTAÇÃO EM NUVEM 19
2.4 COMPUTAÇÃO EM NÉVOA 20
2.5 DOCKER 21
3 TRABALHOS RELACIONADOS 23
4 MATERIAIS E MÉTODOS 28
4.1 PROGRAMAÇÃO 28
4.2 PROCESSO DE OBTENÇÃO DAS SEMENTES DE SIMULAÇÃO . 32
4.3 PROCEDIMENTO DE EXECUÇÃO DAS SIMULAÇÕES NO DOCKER 33
4.3.1 Execução da simulação em ambiente de névoa 34
4.3.2 Execução da simulação em ambiente de nuvem 35
4.3.3 Scripts para a coleta das métricas de desempenho dos contêiners via docker stats 35
4.3.4 Fluxo de execução das simulações e análise dos dados coletados 36
5 RESULTADOS 38
5.1 TEMPO DE EXECUÇÃO DOS ALGORITMOS DIJKSTRA E A* NO AMBIENTE DE NÉVOA 38
5.1.1 Topologia de 2000 semáforos inteligentes processada por computação em névoa 38
5.1.2 Topologia de 3000 semáforos inteligentes processada por computação em névoa 39
5.1.3 Topologia de 4000 semáforos inteligentes processada por computação em névoa 41
5.1.4 Topologia de 5000 semáforos inteligentes processada por computação em névoa 42
5.1.5 Topologia de 6000 semáforos inteligentes processada por computação em névoa 43
5.2 TEMPO DE EXECUÇÃO DOS ALGORITMOS DIJKSTRA E A* NO AMBIENTE DE NUVEM 44
5.2.1 Topologia de 2000 semáforos inteligentes processada por computação em nuvem 44

5.2.2 Topologia de 3000 semáforos inteligentes processada por computação em nuvem . . . 46
5.2.3 Topologia de 4000 semáforos inteligentes processada por computação em nuvem . . . 47
5.2.4 Topologia de 5000 semáforos inteligentes processada por computação em nuvem . . . 48
5.2.5 Topologia de 6000 semáforos inteligentes processada por computação em nuvem . . . 49
5.3 TEMPOS DE EXECUÇÃO MÉDIOS DOS ALGORITMOS DE ROTEAMENTO NOS AMBIENTES DE NÉVOA E NUVEM . . . 50
5.4 RECURSOS COMPUTACIONAIS CONSUMIDOS A PARTIR DA EXECUÇÃO DOS ALGORITMOS DE ROTEAMENTO NO CONTEXTO DE NÉVOA POR TOPOLOGIA . . . 51
5.4.1 Consumo médio no processamento de 2000 semáforos inteligentes em névoa . . . 52
5.4.2 Consumo médio no processamento de 3000 semáforos inteligentes em névoa . . . 53
5.4.3 Consumo médio no processamento de 4000 semáforos inteligentes em névoa . . . 55
5.4.4 Consumo médio no processamento de 5000 semáforos inteligentes em névoa . . . 57
5.4.5 Consumo médio no processamento de 6000 semáforos inteligentes em névoa . . . 58
5.5 RECURSOS COMPUTACIONAIS CONSUMIDOS A PARTIR DA EXECUÇÃO DOS ALGORITMOS DE ROTEAMENTO NO CONTEXTO DE NUVEM POR TOPOLOGIA . . . 60
5.5.1 Consumo médio no processamento de 2000 semáforos inteligentes em nuvem . . . 60
5.5.2 Consumo médio no processamento de 3000 semáforos inteligentes em nuvem . . . 62
5.5.3 Consumo médio no processamento de 4000 semáforos inteligentes em nuvem . . . 63
5.5.4 Consumo médio no processamento de 5000 semáforos inteligentes em nuvem . . . 64
5.5.5 Consumo médio no processamento de 6000 semáforos inteligentes em nuvem . . . 66
5.6 ANÁLISE DO CONSUMO MÉDIO DE RECURSOS POR TOPOLOGIA NOS AMBIENTES DE NÉVOA E NUVEM . . . 67
5.6.1 Ambiente de névoa . . . 67

5.6.2 Ambiente de nuvem 70
6 CONCLUSÃO 73
REFERÊNCIAS 74

1 INTRODUÇÃO

O trânsito pode ser definido como a utilização das vias por pessoas, veículos e animais, isolados ou em grupos, conduzidos ou não, para fins de circulação, parada, estacionamento e operação de carga ou descarga (CTB, 2017).

Nesse sentido, cerca de 36% dos brasileiros que residem em grandes centros urbanos comprometem mais de uma hora por dia no trânsito (IPRI, 2023). Logo, mostra-se imprescindível a criação de uma estratégia para o gerenciamento do fluxo de veículos nas vias, de maneira a optar por caminhos de menor distância com a finalidade de obter uma melhor otimização do tempo.

A partir disso, o conceito de cidades inteligentes, também denominado *smart cities*, é proposto para auxiliar na resolução de problemas urbanos, visto que faz uso de dispositivos de TIC (Tecnologia da Informação e Comunicação) para promover melhorias na qualidade de vida das pessoas e nos serviços urbanos, de modo a aplicar a conectividade dos dispositivos para alinhar cenários urbanos considerados complexos e atender as necessidades dos indivíduos (BOUSKELA *et al.*, 2016).

Nesse contexto, para interceder de forma positiva no controle de tráfego de veículos, aplicam-se os dispositivos de IoT (Internet of Things), ou *Internet of Things*, os quais são descritos como uma rede de equipamentos físicos incorporados a sensores, softwares e outras tecnologias com o intuito de estabelecer conexões e promover a troca de dados entre os dispositivos (ORACLE, 2025).

Desse modo, uma das estratégias a ser adotada para intervir no impasse é a instalação de semáforos inteligentes nas vias públicas urbanas, os quais permitem que o fluxo de trânsito seja controlado conforme a ocorrência de eventos. Eles possuem componentes de hardware que auxiliam na identificação de diversos cenários a qualquer tempo. Assim, as luzes de sinal e o tempo entre elas são ajustados conforme a obtenção dos dados do trânsito em tempo real, considerando os pedestres, os ciclistas e os veículos (CALDAS, 2024).

A partir da necessidade de processar os dados obtidos, a computação em névoa, também conhecida como *fog computing*, é uma abordagem que visa receber os dados brutos coletados por sensores e atuadores, processá-los localmente e encaminhá-los para a nuvem. Essa técnica tem como vantagem proporcionar maior rapidez na tomada de decisões, dado que promove a redução da latência e melhorias na qualidade de serviço (ABDELSHKOUR, 2015). Ainda, segundo Bonomi *et al.* (2012), ela permite estender o paradigma da computação em nuvem até a borda da rede, de maneira a fornecer uma interface intermediária entre os dispositivos finais e os *data centers* da nuvem, com serviços de computação, de armazenamento e de rede. Portanto, é adequada para cenários em que são empregados dispositivos de baixo poder computacional, uma vez que o processamento pode ser realizado de forma local, sem a dependência de grandes servidores.

Além disso, outra estratégia a ser considerada é a computação em nuvem, também denominada *cloud computing*. De acordo com Susnjara e Smalley (2024a), pode ser definida como um acesso por demanda a elementos computacionais como servidores físicos ou virtuais, recursos de redes e ferramentas de aplicações e de desenvolvimento de software, de modo a oferecer melhor escalabilidade e flexibilidade. É vantajosa por eliminar a necessidade de gerenciar recursos físicos próprios, de forma que seu custo é definido pela quantidade de recursos necessitados (GOOGLE, 2025). Então, essa tecnologia mostra-se adequada em ambientes com alto fluxo de processamento de dados, nos quais é necessário o uso de *data centers*.

Então, surge a demanda de analisar o comportamento do processamento de dados e do controle de tráfego com semáforos inteligentes nos dois cenários: na nuvem, com maior poder computacional, e na névoa, mais próxima dos dispositivos, mas com recursos limitados.

Sob essa ótica, neste estudo são propostas a simulação e a análise de desempenho, via contêineres Docker, de ambientes computacionais de nós de computação em névoa e em nuvem encarregados de realizar o gerenciamento de tráfego, com semáforos inteligentes, em infraestruturas urbanas ideais de *smart cities*.

A abordagem faz uso dos algoritmos de roteamento Dijkstra e A* para identificar o menor caminho em uma rota previamente definida e, então, verificar e comparar o desempenho e o consumo de recursos em diferentes topologias de grafos - utilizados como abstrações de malhas urbanas - e em distintos ambientes computacionais.

Ao final deste trabalho, com base nos dados experimentais, é possível estabelecer qual dos algoritmos possui maior estabilidade quando executado no ambiente de névoa - caracterizado por limitações de processamento, armazenamento e energia -, e qual se mostra mais eficiente no uso de recursos computacionais no ambiente de nuvem, em que os recursos são provisionados sob demanda e o custo é baseado no consumo.

1.1 OBJETIVOS

Nas seções a seguir estão descritos o objetivo geral e os objetivos específicos desta monografia.

1.1.1 Objetivo Geral

Este trabalho tem como objetivo analisar o desempenho, em termos de consumo de recursos computacionais, dos ambientes simulados de computação em névoa e em nuvem, os quais atuam como nós de gerenciamento de tráfego com semáforos inteligentes, durante a execução dos algoritmos de roteamento Dijkstra e A*, visando identificar quais algoritmos são mais apropriados para cada contexto e critério estabelecidos.

1.1.2 Objetivos Específicos

- Realizar uma revisão bibliográfica abrangente sobre a computação em névoa e em nuvem, a fim de compreender as aplicações práticas das tecnologias em cidades inteligentes, além de explorar novas possibilidades de estudo.
- Modelar um sistema de semáforos inteligentes por meio de grafos direcionados, em que os nós representam os semáforos e as arestas representam as vias de tráfego, a fim de reproduzir as malhas urbanas de cidades inteligentes ideais.
- Simular os ambientes de nuvem e de névoa por meio de contêineres Docker e diferenciá-los a partir dos contrastes na disponibilidade de recursos.
- Aplicar os algoritmos de roteamento Dijkstra e A* como carga de trabalho, a fim de avaliar o seu desempenho e o uso de recursos computacionais nos contêineres que simulam nós de computação em névoa e em nuvem, os quais são responsáveis por gerenciar o tráfego com semáforos inteligentes.
- Verificar, com base na análise dos dados experimentais obtidos, o algoritmo mais conveniente para cada um dos ambientes simulados.

2 REFERENCIAL TEÓRICO

2.1 CIDADES INTELIGENTES

A crescente migração e o aumento do número de indivíduos presentes nas cidades têm contribuído para o surgimento de diversos desafios no contexto urbano, dado que a população mundial será 68% urbana até 2050 (SARAIVA, 2022).

Nesse sentido, o conceito de cidades inteligentes tem se mostrado essencial para o desenvolvimento e para a melhoria da qualidade de vida das pessoas no momento presente. Existem várias definições para cidades inteligentes, uma delas estabelece que são cidades comprometidas com o desenvolvimento urbano e a transformação digital sustentáveis, em seus aspectos econômico, ambiental e sociocultural, que atuam de forma planejada, inovadora, inclusiva e em rede (SOUSA JÚNIOR *et al.*, 2021).

A seguir, observa-se, na Figura 1, as aplicações que podem ser inseridas em cidades inteligentes de acordo com os eixos urbanos.

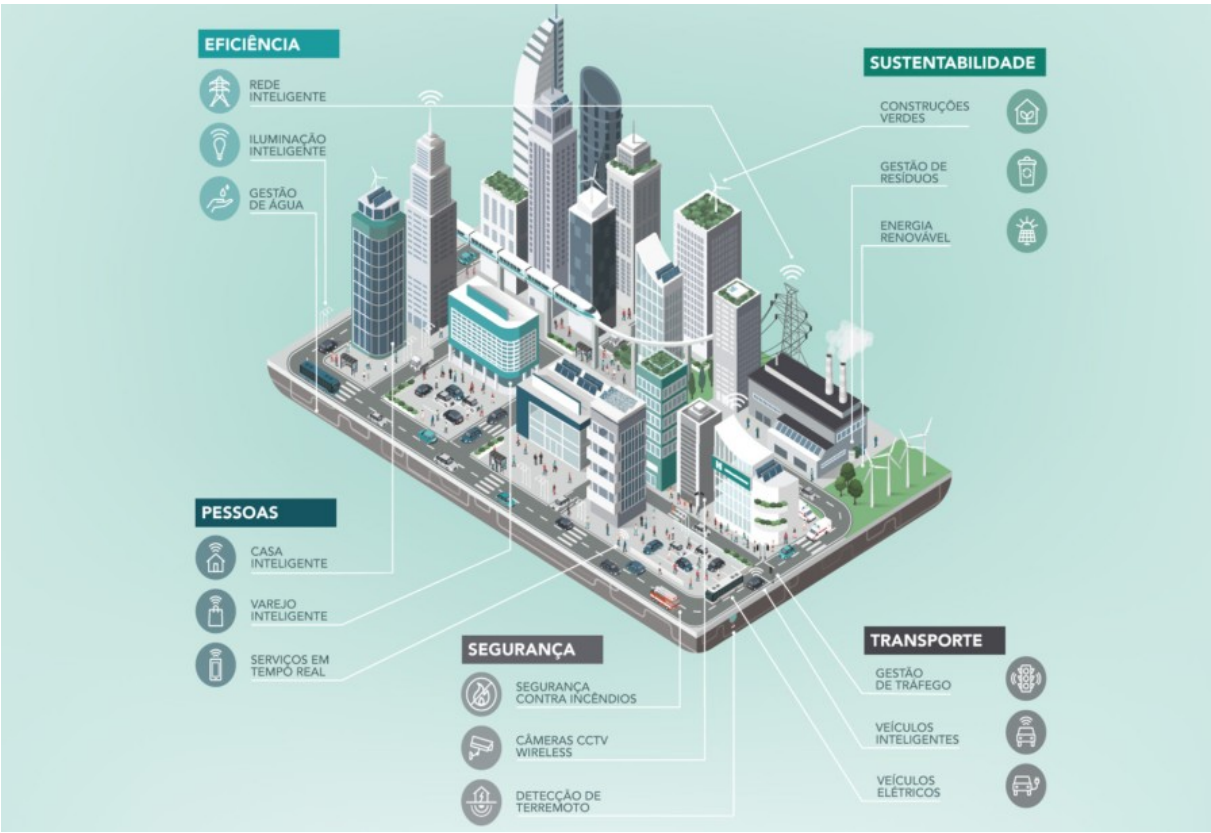

Figura 1 – Infraestrutura inteligente em perspectiva

Fonte: (CARVALHO, 2021).

De acordo com a Figura 1, considera-se a cidade, sua inteligência e os processos que a transformam para torná-la mais inteligente, a partir de ferramentas tecnológicas, caracterizadas por aspectos políticos, de governança e econômicos, os quais englobam as camadas pública, privada, individual, coletiva, de infraestrutura e a intangível (MATOS

et al., 2017).

A partir disso, são consideradas cidades tradicionais e, com o fenômeno da urbanização, se faz necessário gerenciar os dados provenientes de dispositivos IoT de forma adequada para que possam ser utilizados em aplicações diversas nas cidades (SHALINI; ROOPA; DEVI, 2019).

2.2 INTERNET DAS COISAS

A Internet das Coisas tem sido observada com maior frequência no cotidiano das pessoas pelas suas funcionalidades, melhorias e transformações a partir da digitalização de áreas como: economia, agricultura, saúde, transporte, logística, entre outras áreas (SOUSA; FREITAS, 2022).

Na atualidade, com a globalização e os avanços tecnológicos, tem-se um aumento expressivo no número de dispositivos conectados à rede mundial de computadores. O número de dispositivos IoT conectados estabeleceu-se em 18,8 bilhões no ano de 2024, com previsão de aumento para cerca de 41,1 bilhões para o ano de 2030 (SINHA, 2024).

Essa tecnologia define um contexto em que os objetos conectados são capazes de se comunicar mutuamente e compartilhar informações, de modo que a coleta de dados é realizada pelos dispositivos em intervalos de tempo regulares e as informações obtidas são analisadas e utilizadas para a realização de determinadas ações, de forma a estabelecer uma rede inteligente capaz de fazer a análise, o planejamento e a tomada de decisão (DIVYA *et al.*, 2017).

Na Figura 2, é possível verificar a aplicação da IoT no setor de transportes de uma cidade inteligente, com a integração entre pedestres, veículos e semáforos.

Figura 2 – Comunicação entre dispositivos IoT no setor de transportes

Fonte: (GREGÓRIO, 2019).

Em geral, entende-se que a IoT é comumente utilizada para conectar os dispositivos e controlá-los, de maneira presencial ou remota, contudo, define-se como uma rede que pode ser sentida, controlada e programada (DIVYA *et al.*, 2017).

Por fim, considera-se que essa tecnologia é a "espinha dorsal" que realiza a conexão entre os sensores e os dispositivos, com o intuito de promover a conectividade entre eles e, como conseguinte, a tomada de decisão precisa e baseada em dados (MOTA *et al.*, 2023).

2.3 COMPUTAÇÃO EM NUVEM

De acordo com Mell e Grance (2011), a computação em nuvem possibilita o acesso onipresente, conveniente e sob demanda a uma rede de recursos computacionais configuráveis e compartilhados (redes, servidores, armazenamento, aplicativos e serviços), os quais podem ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação com o provedor de serviços.

Com essa tecnologia, não se faz necessário armazenar e processar os dados na máquina local e as empresas podem acessar esses recursos a partir de provedores, oferecendo aos usuários um acesso rápido a recursos escaláveis (FREITAS, 2023).

Ainda, França *et al.* (2023) afirmam que a computação em nuvem apresenta crescente utilização ao passar dos anos devido às suas vantagens como flexibilidade e escalabilidade. A Figura 3 demonstra a arquitetura da computação em nuvem, com os dispositivos atuantes de cada camada.

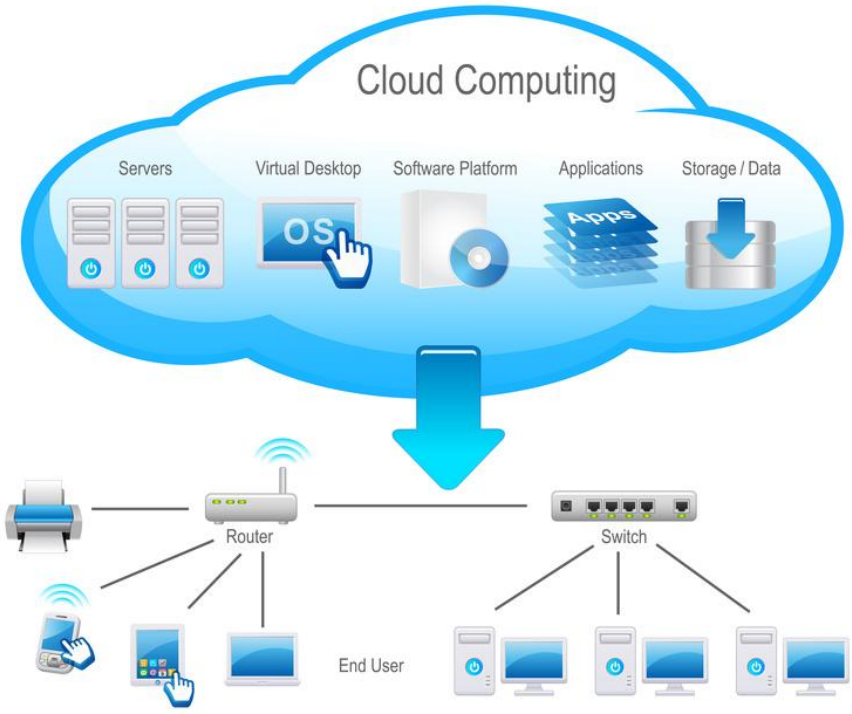

Figura 3 – Organização estrutural da computação em nuvem

Fonte: (SHRIVASTAVA; GANGADHAR; SHUKLA, 2015).

Alguns benefícios dessa técnica são a agilidade, pela possibilidade de gerar recursos computacionais de vários tipos de serviço de forma rápida, e a elasticidade, pois permite provisionar a quantidade de recursos que é realmente necessária (AMAZON AWS, 2025a). Ainda, fornece maior confiabilidade, com sistemas tolerantes a falhas, maior segurança e produtividade (ORACLE, 2020).

2.4 COMPUTAÇÃO EM NÉVOA

A computação em névoa foi introduzida pela Cisco como um serviço que estende a computação em nuvem e os seus serviços até a borda da rede, de modo a fornecer os dados, a computação, o armazenamento e os serviços das aplicações para os usuários finais, tendo como destaque nas suas características o suporte a mobilidade, a distribuição geográfica densa e a sua proximidade com os usuários finais (ABDELSHKOUR, 2015).

Para Jamil e Khan (2019), essa solução tecnológica tem ganhado cada vez mais popularidade pelo fato de conseguir obter um tempo de resposta rápido e reduzir a latência em aplicações consideradas críticas, visto que é normalmente utilizada como um intermediário entre a nuvem e os sensores e atuadores.

Abdelshkour (2015) apresenta os benefícios associados à adoção dessa tecnologia, destacando entre eles os seguintes aspectos:

- Diminuição do consumo da largura de banda;
- Altos níveis de escalabilidade, confiabilidade e tolerância a falhas;
- Redução de congestionamento, custo e latência;
- Reduz a carga de processamento centralizado;
- Eliminação de gargalos provenientes de sistemas de computação centralizados.

Além disso, Delfin *et al.* (2019) expõem as principais características desse tipo de plataforma de processamento localizada próxima à fonte de dados, como:

- Capacidade de oferecer suporte a aplicações sensíveis à latência, que necessitem do processamento de dados em tempo real;
- Os nós de névoa são distribuídos geograficamente, o que fornece maior suporte à mobilidade;
- A troca de informações entre névoa e dispositivos é feita em tempo real;
- Possibilita a análise de dados confidenciais localmente, sem a necessidade de enviá-los a nuvem;
- O constante envio de dados a nuvem pode ocasionar a ocorrência de gargalos, então, a computação em névoa alivia a carga do processamento centralizado.

Ainda, Delfin *et al.* (2019) afirmam que a computação em névoa possui uma estrutura hierárquica, sendo a base da hierarquia composta por dispositivos de borda,

a camada de névoa como intermediária e a camada de nuvem ao topo. Nesse sentido, a camada de borda contém os dispositivos responsáveis pela coleta de dados e transmissão para a camada superior, como sensores, atuadores ou *gadgets* IoT. Já a camada de névoa é responsável por armazenar, pré-processar e realizar a tomada de decisão. Por fim, a camada de nuvem recebe os dados essenciais processados.

A Figura 4 ilustra a organização em camadas da computação em névoa, com os níveis de IoT, névoa e nuvem.

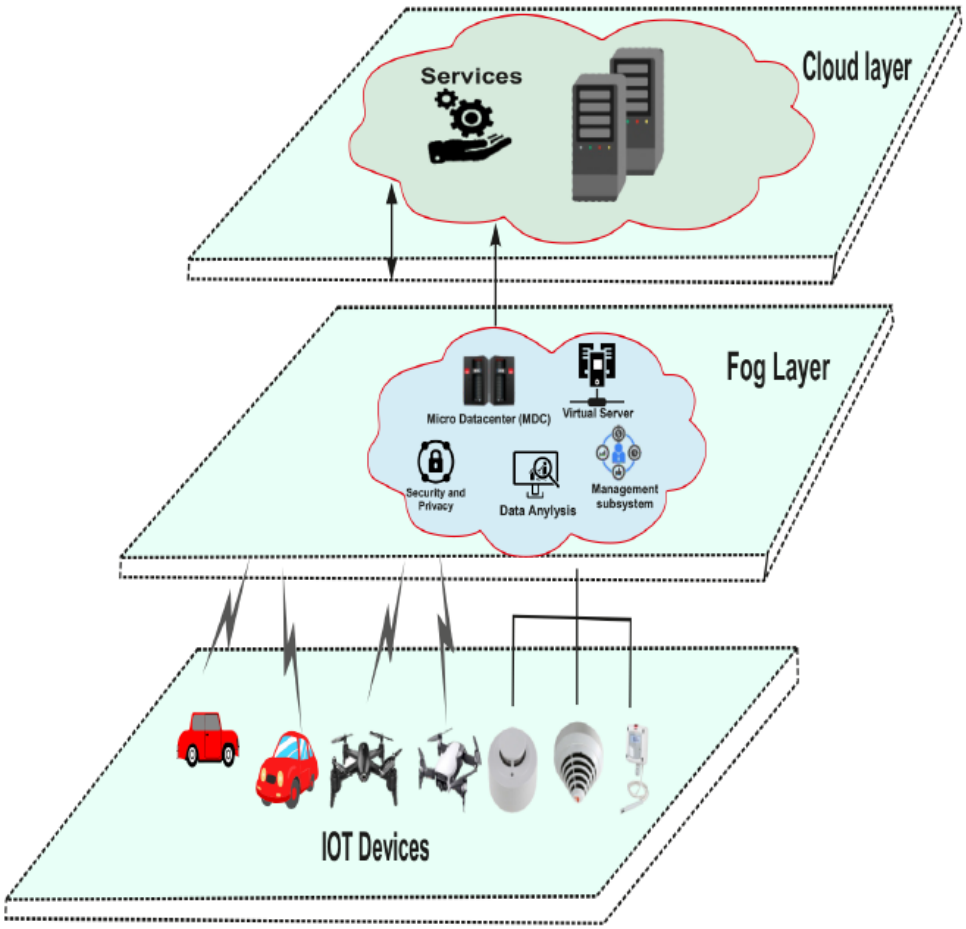

Figura 4 – Arquitetura em camadas da computação em névoa

Fonte: (JAMIL; KHAN, 2019).

Na Figura 4, evidencia-se o funcionamento da troca de informações entre os dispositivos, considerando as particularidades de cada camada. Então, nota-se que cada nível se comunica apenas com os adjacentes.

2.5 DOCKER

Susnjara e Smalley (2024b) definem o Docker como uma plataforma de código aberto que permite aos desenvolvedores as seguintes ações: construir, implementar, executar, atualizar e gerenciar contêineres.

A tecnologia de contêiner Docker foi lançada em 2013 e pode ser descrita como uma unidade padrão de software que tem a capacidade de empacotar o código e todas as suas dependências, de modo que a aplicação seja executada rapidamente e de forma confiável de um ambiente computacional para outro (DOCKER, 2025).

Complementarmente, o Docker faz a simplificação do ciclo de vida do desenvolvimento ao permitir que os desenvolvedores trabalhem em ambientes padronizados, utilizando os contêineres locais, que fornecem suas aplicações e serviços (AMAZON AWS, 2025b).

Os contêineres têm como principais características serem portáteis, leves e isolados. São leves pelo fato de compartilharem o *kernel* do sistema *host*, portáteis pois podem ser usados em várias máquinas quando são construídos e isolados porque são executados em processos separados (MAKODE, 2024).

Essa tecnologia ocupa menos espaço de armazenamento, pode lidar com outras aplicações do sistema e exige menos no que se refere às máquinas virtuais e sistemas operacionais. Em contrapartida, as máquinas virtuais funcionam como uma abstração do hardware físico, que inclui em seu funcionamento uma cópia do sistema operacional, das aplicações e dos componentes necessários, ocupando maior espaço em disco (DOCKER, 2025).

Outro conceito importante é o de imagem Docker. A imagem, feita a partir da escrita de um conjunto de instruções em um arquivo `Dockerfile`, é um modelo do contêiner que contém todos os elementos necessários para a execução da aplicação, como código, bibliotecas e dependências (MAKODE, 2024).

Nesse sentido, um contêiner corresponde a instância de uma imagem Docker que está sendo executada. Ao ser criado, o Docker inicializa essa imagem em um ambiente separado, no qual a aplicação pode ser executada de forma autônoma em conjunto com todas as suas dependências e configurações necessárias (MAKODE, 2024).

3 TRABALHOS RELACIONADOS

Jang e Lin (2018) propuseram uma estrutura de controle de semáforos baseada em Redes Definidas por Software (*Software Defined Networking* - SDN), computação em névoa e computação em nuvem para otimizar a coleta e transmissão de dados de tráfego. O uso de SDN permitiu a alocação flexível de largura de banda da rede conforme a demanda, enquanto a computação em névoa foi empregada para reduzir a carga da nuvem e proporcionar respostas mais rápidas. A computação em nuvem, por sua vez, organizou os recursos globalmente, realocando-os conforme as necessidades de tráfego. A simulação, utilizando Mininet, Ryu Controller, Vanet Simulator e sFlow, avaliou o desempenho do sistema, demonstrando que o algoritmo de controle de semáforos proposto otimizou o tempo de condução, reduzindo-o em até 55,6% para cenários com mais de 300 veículos.

Jamil e Khan (2019) desenvolveram um sistema de gestão de acidentes baseado em computação em névoa para melhorar a resposta emergencial e minimizar os congestionamentos. A arquitetura foi dividida em três camadas: dispositivos inteligentes, névoa e nuvem. A primeira camada, que engloba os sensores e atuadores que realizam a comunicação sem fio, foi designada para realizar a coleta dos dados localmente por meio dos equipamentos veiculares e enviá-los para a camada superior. Além disso, a camada de névoa foi responsável pela detecção de acidentes e comunicação em tempo real com os serviços de emergência, enquanto a camada de nuvem armazenou e analisou os dados. A simulação foi realizada no iFogSim com diferentes configurações, comparando a latência e o uso da rede entre a computação em névoa e em nuvem. Os resultados mostraram que a abordagem em névoa reduziu significativamente a carga da nuvem e melhorou a eficiência da resposta a acidentes.

Qin e Zhang (2020) apresentaram um sistema inteligente de controle de semáforos baseado em computação em névoa e aprendizado por reforço, utilizando o algoritmo *Q-learning* para ajustar dinamicamente a temporização dos semáforos. Servidores de névoa foram distribuídos em cruzamentos para coletar e processar os dados do trânsito, enquanto a camada de nuvem organizou o tráfego em nível urbano. A simulação, realizada com VISSIM, Excel e VBA-MATLAB, comparou o sistema proposto a métodos tradicionais, demonstrando sua eficácia na redução do congestionamento e na melhoria do fluxo viário. No entanto, a segurança do método foi avaliada apenas por simulação, necessitando de validação prática.

Ning, Huang e Wang (2019) propuseram um sistema de gerenciamento de tráfego em tempo real baseado em Computação em Névoa Veicular (do inglês *Vehicular Fog Computing* - VFC) para reduzir tempos de resposta a eventos viários. A arquitetura em três camadas (nuvem, *cloudlets* e névoa) permitiu a coleta descentralizada de dados por veículos e Unidades de Beira de Estrada (*Roadside Units* - RSU), com processamento local em nós de névoa e *cloudlet* antes do envio à nuvem. A validação, realizada em distritos

de Xangai com trajetórias de táxis, indicou que a abordagem reduz o atraso de resposta e alivia a carga da rede, melhorando a eficiência do gerenciamento de tráfego.

Brennand *et al.* (2016) desenvolveram o sistema *Fast Offset Xpath* (FOX), um modelo baseado em computação em névoa para o gerenciamento de tráfego em tempo real. Utilizando RSUs e sensores, cada nó de névoa monitorou os congestionamentos localmente e trocou informações com outras unidades para otimizar o tráfego. O modelo utilizou um grafo dinâmico para redirecionar os veículos de maneira probabilística por meio da Distribuição de Boltzmann, evitando novos congestionamentos. A simulação, realizada com OMNet++, *Simulation of Urban Mobility* (SUMO) e o modelo EMIT, mostrou que o FOX reduz o tempo de viagem, as emissões de CO_2 e o consumo de combustível, validando sua eficácia para cidades inteligentes.

Luo, Chen e Wu (2021) apresentaram um Sistema de Semáforos Inteligentes (*Smart Traffic Lights System* - STLS) baseado em computação em névoa e em nuvem para analisar a eficiência energética de ambas as tecnologias quando aplicadas a um ambiente de tráfego urbano. Os semáforos inteligentes foram usados com o propósito de reduzir a ocorrência de congestionamentos nos cruzamentos. Desse modo, utilizou-se de câmeras e RSUs como nós de névoa em cada cruzamento. Para fins de estudo, a nuvem foi inserida próxima das ruas do centro de Houston, as quais foram consideradas na simulação do estudo. A simulação, realizada com o software iFogSim e feita com quatro parâmetros de configuração, sendo eles o número de ruas, a proporção da potência ociosa para a potência de utilização total, a especificação da Unidade Central de Processamento (*Central Processing Unit* - CPU) e a capacidade de processamento do microprocessador, evidenciou que a computação em névoa é sempre mais eficiente energeticamente do que a computação em nuvem nessa aplicação.

Islam e Hassan (2023) desenvolveram um Sistema Inteligente de Gerenciamento de Congestionamento de Tráfego (*Intelligent Traffic Congestion Management System* - ITCMS) com a utilização do paradigma da Internet de Veículos (*Internet of Vehicles* - IoV) e computação em nuvem e em névoa. A camada de nuvem foi empregada com o intuito de ofertar maior capacidade de armazenamento e processamento ao sistema. A camada de névoa, formada por Estações Base (*Base Stations* - BS), foi utilizada para validar os dados obtidos pelos sensores antes de retransmitir as informações. A camada de dispositivos inteligentes, composta por sensores e atuadores, foi designada para reunir as informações e os detalhes presentes nos veículos e enviá-las para avaliação e processamento. O objetivo da pesquisa foi comparar o desempenho da computação em névoa com o da computação em nuvem seguindo os parâmetros de uso da rede, latência e custo financeiro de execução. A partir da simulação realizada no iFogSim, constatou-se que a névoa reduziu de forma significativa o uso da rede, a latência e o custo de execução e implementação.

Matange *et al.* (2023) propuseram um sistema de controle de tráfego de veículos, adaptável e resiliente, pensado idealmente para a Índia. A computação em névoa e em

nuvem foram empregadas em conjunto visando o melhor desempenho do sistema. Dessa forma, os nós de névoa são responsáveis por coletar as imagens de câmeras instaladas em cruzamentos e, a partir das gravações, processar e inferir o número total de veículos presentes nas vias e encaminhar para a nuvem, a qual ajusta o tempo dos semáforos a partir da manipulação dos dados por um código de ajuste de tempo de ciclo. Além disso, é proposta uma alternativa para o caso de falha em nós de névoa e então, por meio de inteligência artificial, infere-se o tempo dos semáforos com base nos dados coletados anteriormente. A simulação, realizada com SUMO, OpenCV, YOLO e o modelo de regressão K-Vizinhos Mais Próximos (*K-Nearest Neighbors* - KNN), evidenciou a redução do tempo médio de espera dos veículos e de perda de tempo tanto no funcionamento pleno do sistema quanto na situação em que alguns nós de névoa falharam.

Al-Hakimi e Subbiah (2024) apresentaram uma abordagem teórica a respeito de um sistema de semáforos autônomos com névoa. O objetivo da pesquisa foi proporcionar, a partir de situações de emergência, o controle dos semáforos pelos motoristas dos veículos de emergência da origem até o destino com o intuito de evitar a perda de tempo em congestionamentos. Desse modo, ao receber a notificação de emergência e estabelecer a rota, os semáforos são identificados e o motorista pode alterar o funcionamento destes abrindo-os antes de iniciar a viagem e, ao passar por eles, estes retornam para o sinal vermelho. A névoa é responsável por armazenar o status dos semáforos e por definir a melhor rota a ser percorrida. São propostos 4 pseudocódigos, sendo eles para controlar o Diodo Emissor de Luz (*Light Emitting Diode* - LED) dos semáforos, para o funcionamento dos semáforos sem interrupções, para ativar o modo de espera para o estado de emergência e para manter a verificação de emergências, que possibilitam melhor experiência de direção.

Gu, Hu e Jia (2023) propuseram um sistema de controle e gerenciamento de congestionamentos de trânsito baseado em áreas adaptáveis com a utilização da Internet de Veículos (*Internet of Vehicles* - IoV) e de computação em névoa. O objetivo é particionar uma região urbana em áreas de gerenciamento de tráfego e, assim, implementar as orientações de rotas e o controle dos semáforos para diminuir os congestionamentos. A névoa é aplicada por meio de servidores e RSUs, os quais são responsáveis por coletar as informações do fluxo de veículos e armazená-las para uso, e tem o propósito de estabelecer o caminho mais curto nas rotas e informar o grau do fluxo do tráfego, em tempo real, nas áreas divididas. A validação, realizada por meio de três algoritmos (geração de sub-rede rodoviária, ajuste de semáforo adaptativo local e controle de semáforo coordenado regional) e pela simulação com o uso do SUMO, evidenciou que o sistema proposto é eficaz para diminuir ou cessar a ocorrência de congestionamentos.

Hossan e Nower (2020) desenvolveram um sistema de controle dinâmico de semáforos baseado em computação em névoa para melhorar o transporte público. Os autores propuseram a aplicação do Algoritmo de Controle Dinâmico Eficiente do Semáforo para Múltiplos (*Efficient Dynamic Traffic Light Control algorithm for Multiple* - EDTLCM), o

qual tem por objetivo estabelecer a duração ideal dos sinais verdes visando diminuir o uso de combustíveis, o tempo médio de espera e a maximização do rendimento. Considerou-se no estudo veículos comuns e de transporte público. Os nós de névoa são aplicados para fornecer baixa latência, dados em tempo real e capacidade de processamento de dados a partir de muitos dispositivos. Assim, cada nó de névoa é conectado com seus nós adjacentes e todos eles são vinculados à nuvem com o intuito de obter os dados da própria intersecção e das suas vizinhas visando cobrir toda a área metropolitana. A simulação, feita com SUMO, Docker, MySQL e Python no desenvolvimento do EDTLCM, mostrou que o sistema reduz o tempo médio de espera, o consumo de combustível e o rendimento quando comparado com sistemas dinâmicos semelhantes e com o sistema de controle de semáforos de tempo físico.

A partir dos trabalhos relacionados acima, a Tabela 1 mostra a síntese das pesquisas realizadas, com ênfase na metodologia, no problema e nas ferramentas utilizadas.

Tabela 1 – Síntese dos trabalhos relacionados

Trabalho	**Metodologia**	**Problema**	**Ferramentas**
(BRENNAND *et al.*, 2016)	FOX baseado em névoa.	Diminuição do tempo de viagem, das emissões de CO_2 e do consumo de combustível.	OMNet++, SUMO e EMIT.
(JANG; LIN, 2018)	Estrutura de controle de semáforos baseada em SDN, névoa e nuvem.	Otimização do tempo de condução de veículos.	Mininet, Ryu Controller, Vanet Simulator e sFlow.
(JAMIL; KHAN, 2019)	Sistema de gestão de acidentes com névoa.	Melhora da resposta emergencial e a minimização de congestionamentos.	iFogSim.
(NING; HUANG; WANG, 2019)	Sistema de gerenciamento de tráfego em tempo real baseado em VFC.	Redução do tempo de resposta a eventos viários.	Algoritmos *Branch-and-Bound*, *Edmonds-Karp* e de *offloading* habilitado para VFC.
(QIN; ZHANG, 2020)	Sistema inteligente de controle de semáforos com névoa e aprendizado por reforço.	Redução de congestionamentos e melhoramento do fluxo viário.	VISSIM, Excel, VBA-MATLAB e *Q-Learning*.

Trabalho	**Metodologia**	**Problema**	**Ferramentas**
(HOSSAN; NOWER, 2020)	Sistema de controle dinâmico de semáforos com névoa para transporte público.	Redução do uso de combustível, do tempo médio de espera e a maximização do rendimento.	SUMO, Docker, MySQL, Python e EDTLCM.
(LUO; CHEN; WU, 2021)	STLS com computação em névoa e em nuvem.	Análise da eficiência energética em ambientes de tráfego urbano.	iFogSim.
(ISLAM; HASSAN, 2023)	ITCMS com uso de IoV, névoa e nuvem.	Comparação do desempenho da computação em névoa e em nuvem no uso da rede, latência e custo de execução e implantação.	iFogSim.
(MATANGE *et al.*, 2023)	Sistema de controle de tráfego de veículos adaptável e resiliente para a Índia.	Redução do tempo médio de espera dos veículos e da perda de tempo.	SUMO, OpenCV, YOLO e KNN.
(GU; HU; JIA, 2023)	Sistema de controle e gerenciamento de congestionamentos de trânsito com IoV e névoa.	Redução de congestionamentos.	SUMO, algoritmo de geração de sub-rede rodoviária, de ajuste de semáforo adaptativo local e de controle coordenado regional de semáforos.
(AL-HAKIMI; SUBBIAH, 2024)	Abordagem teórica de um sistema de semáforos autônomos com computação em névoa.	Necessidade de realizar o controle dos semáforos por motoristas de veículos de emergência em situações emergenciais.	Algoritmo para realizar o controle dos LEDs dos semáforos, para o funcionamento dos semáforos sem interrupções, para ativar o modo de espera e para manter a verificação de emergências.

Fonte: Elaborado pelo autor.

4 MATERIAIS E MÉTODOS

Nesta pesquisa, foi realizada uma análise do comportamento de ambientes computacionais simulados de névoa e nuvem, a nível de contêiner, para comparar o desempenho e o consumo de recursos computacionais quando submetidos a topologias e algoritmos de roteamento distintos, considerando as decisões de rota entre dois semáforos inteligentes para a obtenção do caminho mais curto.

Nesse sentido, as seguintes ferramentas foram adotadas: `Python 3.12.3` para a programação, `Docker 28.1.1` para a criação e execução dos ambientes simulados e os módulos `networkx 3.4.2`, `random`, `time`, `csv` e `argparse`.

Além disso, a máquina em que os experimentos foram realizados contém o processador Intel Core i5-8265U, com 8 núcleos, 8GB de memória RAM e sistema operacional Ubuntu 24.04.2 *Long Term Support* (LTS).

4.1 PROGRAMAÇÃO

A utilização de grafos foi definida a partir da necessidade de se propor uma abstração da infraestrutura viária de cidades inteligentes ideais, compostas por vias e semáforos inteligentes. Então, foi desenvolvida uma função para a geração dos grafos com quantidades aleatórias de arestas e de pesos associados a elas, a qual é mostrada a seguir na Figura 5.

```python
# Função que gera os grafos com arestas aleatórias e pesos aleatórios
def gera_grafos(semaforos, semente, p_mim=10, p_max=50):
    random.seed(semente) # Definição da semente
    ruas = random.randint(semaforos*1, semaforos*2)

    G = nx.DiGraph()
    G.add_nodes_from(range(semaforos))

    # Atribui posições (x, y) aleatórias para cada nó
    for node in G.nodes:
        G.nodes[node]["pos"] = (random.randint(0, 100), random.randint(0, 100))

    arestas_adicionadas = set()

    while len(G.edges) < ruas:
        # Estabelecimento de arestas aleatórias
        origem = random.randint(0, semaforos - 1)
        destino = random.randint(0, semaforos - 1)

        # Verificação se a aresta já existe ou impedimento de origem/destino iguais
        if origem != destino and (origem, destino) not in arestas_adicionadas:
            peso = random.randint(p_mim, p_max)
            G.add_edge(origem, destino, weight=peso)
            arestas_adicionadas.add((origem, destino))
    return G
```

Figura 5 – Função para a geração de grafos com arestas e pesos aleatórios

Fonte: Elaborado pelo autor.

Ela recebe como parâmetros a quantidade de semáforos inteligentes, que equivale a quantidade de nós do grafo, a numeração da semente de simulação e os pesos mínimo e máximo das arestas, os quais foram definidos como 10 e 50 respectivamente. A ideia principal foi estabelecer uma aleatoriedade controlada, com o uso de sementes de simulação.

A aplicação de sementes foi feita para manter as mesmas características estabelecidas nos cenários de simulação, em todas as topologias, a fim de que a experimentação possa ser replicada por terceiros em outras máquinas *host*. Nesse sentido, são preservadas todas as especificações do grafo atribuídas à semente, como o número de arestas, seus pesos e as ligações estabelecidas.

A princípio, na função, foram definidas a semente e o número de arestas do grafo, o qual equivale ao número de ruas no ambiente de simulação. Então, para ambos os casos foi utilizado o módulo `random`, sendo usada a função `seed()` para a estabelecer a semente e a função `randint()` para definir um número aleatório para as arestas. A última recebeu como parâmetros a quantidade de nós e o dobro desse valor como limites mínimo e máximo, respectivamente.

Em seguida, o grafo foi gerado por meio do módulo `networkx` com a função `DiGraph()`, a qual gera grafos direcionados. Em seguida, os nós são adicionados, a partir da quantidade informada de semáforos, pela função `add_nodes_from()`. Também, foram atribuídas posições aleatórias em forma de par ordenado (x, y). Novamente, a função `randint()` foi usada e teve como parâmetros mínimo e máximo 0 e 100, em ordem, tanto para a coordenada x quanto para a y.

Logo após, deu-se início a inserção das arestas no grafo, por meio da ferramenta `set` do Python, a qual estabelece uma coleção não ordenada de elementos. Então, a origem e o destino da aresta são definidos aleatoriamente com `randint()`, de modo que o mínimo é 0 e o máximo é o número de semáforos decrescido de uma unidade.

Posto isso, é verificado se a origem da aresta é diferente de seu destino e se ambos já não estão inclusos no conjunto de arestas. Se a condição for satisfeita, é definido um peso para a aresta, com `randint()` que recebe como parâmetros os valores informados no parâmetro da função de gerar grafos, a aresta é adicionada ao grafo por meio da função `add_edge()` e a origem e o destino são inseridos no conjunto de arestas adicionadas.

A partir disso, a função retorna o grafo G criado e que será utilizado para a experimentação. Como foi definido o uso de sementes, cada grafo G produzido será distinto aos das demais sementes em relação às arestas, pesos e pares ordenados.

Também, foi escrita uma função responsável por executar os algoritmos de roteamento, Dijkstra e A*, e coletar o caminho, o custo e o tempo de execução em milissegundos. Ela recebe como parâmetros o grafo G, o tipo de algoritmo de roteamento a ser executado e os nós de origem e destino. O código correspondente é exposto na Figura 6.

```python
#Função que executa os algoritmos Dijkstra ou A* e coleta o caminho, custo e tempo em milissegundos
def executa_algoritmo(G, algoritmo, origem, destino):
    inicio = time.time()

    if algoritmo.lower() == "dijkstra":
        caminho = nx.dijkstra_path(G, origem, destino, weight='weight')

    elif algoritmo.lower() == "astar":
        # Verifica se as posições existem
        if "pos" not in G.nodes[origem] or "pos" not in G.nodes[destino]:
            raise ValueError("Os nós precisam ter atributo 'pos' com coordenadas (x, y) para usar A*.")

        heuristica = lambda u, v: distancia_manhattan(G.nodes[u]["pos"], G.nodes[v]["pos"])
        caminho = nx.astar_path(G, origem, destino, heuristic=heuristica, weight='weight')

    else:
        raise ValueError("Use 'Dijkstra' ou 'A*'.")

    fim = time.time()
```

Figura 6 – Função de execução dos algoritmos de caminho mínimo

Fonte: Elaborado pelo autor.

O algoritmo de Dijkstra foi escolhido devido ao fato dos grafos gerados terem pesos associados a suas arestas, ou seja, serem grafos ponderados. O peso, no caso deste estudo, representa a distância entre um par de nós ligados por determinada aresta. Nesse sentido, o algoritmo possibilita encontrar o caminho de menor custo entre um nó de origem e os demais nós de um grafo. O seu funcionamento se dá a partir da manutenção das distâncias mais curtas conhecidas entre os nós do grafo e o nó de origem e realiza a atualização dos valores caso encontrar um caminho mais curto. Então, quando o algoritmo encontra um caminho de menor custo entre o nó de origem e um nó vizinho, o nó vizinho é marcado e adicionado ao caminho percorrido. O processo continua até que todos os nós que possuam menor custo com relação ao outro sejam adicionados ao caminho, tendo no final a rota de menor custo possível (NAVONE, 2022).

Já o A* foi selecionado pois é um algoritmo de busca que faz uso de uma heurística para orientar a sua busca com relação ao nó de destino. A heurística é responsável por estimar o custo para alcançar o nó de destino a partir de outro nó, de maneira a considerar rotas mais promissoras e evitar a exploração de todos os caminhos. Nesse caso, a heurística funciona como um palpite de qual caminho percorrer até o destino considerando o nó atual (KUMAR, 2024).

Então, foi definida a Distância de Manhattan como a heurística a ser utilizada, devido a simplicidade do cálculo para se obter o valor da distância. Ela é definida matematicamente como sendo a soma dos módulos das diferenças entre as coordenadas x e y de dois nós, como pode ser visto na Equação (1).

$$D = |x_1 - x_2| + |y_1 - y_2| \tag{1}$$

A função para a heurística foi definida no código como `distancia_manhattan()` e recebe como parâmetros dois atributos de posição de nós distintos. Assim, foram definidas

as variáveis x_1 e y_1 para representar a posição do primeiro nó, e x_2 e y_2 para representar a do segundo nó. Ao final, foi retornado o valor numérico da distância entre os nós.

A partir disso, é usado o módulo `time` para registrar o momento de início da execução, a qual é realizada de acordo com o tipo de algoritmo informado via parâmetro.

Caso seja o Dijkstra, a função `dijkstra_path()` do módulo `networkx` recebe como parâmetros o grafo G, os nós de origem e destino e o peso. A função define que os pesos das arestas devem ser atributos numéricos e as distâncias entre os nós serão calculadas com a soma dos pesos das arestas percorridas, para obter a rota mais curta do nó fonte para o nó alvo no grafo G (NETWORKX, 2024b).

Se o algoritmo a ser executado for o A*, primeiro é verificado se os pares ordenados (x, y) existem no grafo G. Então, com o uso da função `astar_path()` são passados como parâmetros o grafo G, os nós de origem e destino, a heurística e o peso. Ela estabelece que a heurística a ser utilizada deve receber argumentos de posição de dois nós, as coordenadas (x, y), e retornar um número (NETWORKX, 2024a).

Em ambos os casos, após a conclusão, o tempo de fim da execução é armazenado e são realizados os cálculos do custo total e do tempo de execução. São retornados pela função o caminho, o custo total e o tempo de execução em milissegundos.

Por fim, a última função desenvolvida foi a responsável por controlar toda a experimentação. Ela recebe como parâmetros a topologia, que corresponde ao número de nós, a numeração da semente, o algoritmo de roteamento a ser usado e o nome do arquivo em que serão armazenados os resultados no formato `.csv`. A seguir, exibe-se na Figura 7 o código que implementa a lógica da função.

```python
# Função principal para rodar o experimento
def rodar_experimento(topologia, semente, algoritmo, nome_arquivo):
    # Gera o grafo para a topologia e semente
    G = gera_grafos(topologia, semente)

    # Definindo os nós de origem e destino
    origem = 0
    destino = 1754

    # Executa o algoritmo e coleta as métricas
    caminho, custo, tempo = executa_algoritmo(G, algoritmo, origem, destino)

    # Se não houver caminho válido, ignora essa execução
    if caminho is None:
        print(f"Semente {semente}: Não há caminho entre {origem} e {destino}.")
        return

     # Número de arestas
    num_arestas = len(G.edges)

    # Registra os resultados no arquivo CSV
    with open(nome_arquivo, mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['topologia', 'semente', 'num_arestas', 'caminho', 'custo', 'tempo(ms)'])
        writer.writerow([topologia, semente, num_arestas, caminho, custo, tempo])
```

Figura 7 – Função de controle de execução do experimento

Fonte: Elaborado pelo autor.

Primeiramente, o grafo G é definido ao usar a função `gera_grafos()` e os nós de

origem e destino são estabelecidos como 0 e 1754, respectivamente. Em seguida, o caminho, o custo e o tempo de execução são coletados pela função `executa_algoritmo()`. Depois, é verificado se há um caminho existente entre os nós de origem e destino e, caso não houver, o programa encerra a sua execução.

A partir das informações coletadas, é gerado o arquivo de resultados em formato `.csv`, com o uso das funções `writer()` e `writerow()` do módulo `csv`, com as seguintes colunas: topologia, semente, num_arestas, caminho, custo e tempo em milissegundos.

Fora do bloco de funções, foi estabelecida a coleta dos parâmetros informados na linha de execução do código com o uso do módulo `argparse`, como exposto na Figura 8.

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Simulação de algoritmos em topologias de grafos.")
    parser.add_argument("--topologia", type=int, required=True, help="Número de nós na topologia (ex: 20)")
    parser.add_argument("--semente", type=int, required=True, help="Número da semente (ex: 20)")
    parser.add_argument("--algoritmo", choices=["dijkstra", "astar"], required=True, help="Algoritmo a ser utilizado")
    parser.add_argument("--saida", type=str, default="resultados.csv", help="Nome do arquivo de saída CSV")

    args = parser.parse_args()
    rodar_experimento(args.topologia, args.semente, args.algoritmo, args.saida)
```

Figura 8 – Função principal do experimento

Fonte: Elaborado pelo autor.

São eles: o número de nós, a numeração da semente, o algoritmo de roteamento a ser usado e o nome do arquivo de saída `.csv`. Tais informações foram adicionadas ao parser por meio da função `add_argument()` e, posteriormente, transferidas para uma variável com o uso de `parse_args()`. Por fim, a função que executa o experimento é chamada com todos os dados obtidos como parâmetros, com o objetivo de iniciar a execução.

Para a realização das simulações, foram definidas as topologias 2000, 3000, 4000, 5000 e 6000. Para cada uma delas foram obtidas 10 sementes, totalizando 50 sementes na experimentação.

4.2 PROCESSO DE OBTENÇÃO DAS SEMENTES DE SIMULAÇÃO

Para se obter as sementes de simulação, foi desenvolvido um script em Python que recebe, na linha de comando de execução no terminal, a topologia e o número máximo de sementes a serem encontradas. Ao final retorna todas as sementes consideradas válidas. O comando executado, para todas as topologias, foi:

```
python3 encontra_sementes.py -topologia (número da topologia)
                             -max_semente (quantidade de sementes)
```

O algoritmo aproveita a função de geração de grafos mostrada anteriormente e adiciona uma nova função responsável por encontrar as sementes válidas. Nela, uma lista vazia de sementes válidas é inicializada e são informadas a origem e o destino como 0 e 1754, nessa ordem, como pode ser visto na Figura 9.

```python
def encontra_sementes_validas(topologia, max_semente=999):
    sementes_validas = []
    origem = 0
    destino = 1754

    for semente in range(max_semente + 1):
        print(sementes_validas)
        try:
            G = gera_grafos(topologia, semente)
            # Tenta encontrar caminho com Dijkstra
            caminho = nx.dijkstra_path(G, origem, destino, weight='weight')
            if caminho:  # Se encontrar caminho
                sementes_validas.append(semente)
        except (nx.NetworkXNoPath, nx.NodeNotFound):
            # Ignora se não há caminho ou se o nó destino não existe
            continue
    return sementes_validas

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Encontra sementes válidas para uma topologia.")
    parser.add_argument("--topologia", type=int, required=True, help="Número de nós da topologia")
    parser.add_argument("--max_semente", type=int, default=999, help="Valor máximo de semente a ser testado")

    args = parser.parse_args()
    sementes = encontra_sementes_validas(args.topologia, args.max_semente)

    print(f"Sementes válidas para topologia {args.topologia}: {sementes}")
```

Figura 9 – Função de seleção das sementes válidas para as simulações

Fonte: Elaborado pelo autor.

Então, em um laço, é gerado um grafo G para a atual semente e tenta-se encontrar um caminho válido entre a origem e o destino informados. Caso exista, o número da semente é inserido na lista de sementes válidas, caso contrário, ignora e vai para a próxima semente. O processo se repete até que o número máximo de sementes informado na execução seja atingido. No término, é mostrada no terminal a listagem de sementes válidas para a topologia.

É válido destacar que, para a experimentação, foram selecionadas as 10 primeiras sementes válidas de cada topologia a partir dos resultados gerados pelo script.

4.3 PROCEDIMENTO DE EXECUÇÃO DAS SIMULAÇÕES NO DOCKER

A abordagem adotada para a experimentação foi a de utilizar os algoritmos de roteamento Dijkstra e A* como carga de trabalho para a verificação do quanto de recurso será consumido em um ambiente de nuvem (com recursos ilimitados) e de névoa (com recursos limitados) com o uso de contêineres Docker e, assim, definir qual algoritmo é mais adequado em cada contexto.

O intuito é simular, em nível de contêiner, o comportamento de uma estrutura de nó de névoa e nó de nuvem, avaliando a execução dos algoritmos de roteamento na origem e destino estabelecidos. Esse caminho corresponde à rota mais curta entre os semáforos inteligentes definidos previamente.

Para isso, criou-se um arquivo `Dockerfile` a fim de gerar uma imagem personalizada e reutilizável para, então, inicializar os contêineres. Nele, foram definidos:

- A imagem base do Python: `python:3.12.3-slim`
- A definição do diretório de trabalho no contêiner: `WORKDIR /app`

- Foi feita a instalação de dependências: `apt-get update && apt-get install -y python3-dev build-essential && rm -rf /var/lib/apt/lists/*`
- Cópia do arquivo *requirements*: `COPY requirements.txt RUN pip install -no-cache-dir -r requirements.txt`
- Cópia dos demais arquivos: `COPY . .`
- Definido o comando padrão de execução do script: `CMD ["python", "main.py"]`

O arquivo `requirements.txt` também foi criado e possui os seguintes módulos que serão utilizados pelo Python na execução do experimento: `networkx`, `argparse` e `docker-py`. Desse modo, os módulos são instalados de forma correta para uso nos contêineres. Com isso, são garantidas a portabilidade e a reprodutibilidade da imagem Docker em outras máquinas que contenham o software instalado, de maneira a possibilitar a replicação do experimento realizado.

4.3.1 Execução da simulação em ambiente de névoa

A simulação em névoa foi implementada considerando um ambiente com recursos restritos. A limitação de recursos foi estabelecida devido ao fato do nó de névoa atuar como um intermediário entre os dispositivos de borda e a nuvem. Nesse sentido, ele opera como uma unidade de computação que localiza-se próxima aos semáforos inteligentes e auxilia na determinação da rota de menor caminho entre dois pontos, definidos como semáforos de origem e destino. Logo, a fim de reproduzir um comportamento mais fiel à realidade, os recursos computacionais foram definidos manualmente na inicialização do contêiner responsável pela simulação neste tipo de ambiente.

Para realizar a inicialização da imagem Docker, foi utilizado o seguinte comando no terminal:

```
docker build -t simulacao-tcc .
```

Ele é encarregado de criar a imagem Docker personalizada a partir do arquivo `Dockerfile` e nomeá-la como `simulacao-tcc`.

O comando usado para a execução da simulação em névoa, em todas as topologias, sementes e algoritmos, é descrito a seguir:

```
docker run -it -rm -cpus=0.1 -memory=64m -memory-swap=128m -cpu-shares=64
  -device-read-bps=/dev/sda:250k -device-write-bps=/dev/sda:250k -name
   nevoa -v "/home/user/Área de trabalho/simulacao_tcc:/app:rw-u $(id
-u):$(id -g) simulacao-tcc python3 main.py -topologia (nº da topologia)
    -semente (nº da semente) -algoritmo (dijkstra ou astar) -saida
                          (arquivo.csv)
```

Abaixo, é descrita a capacidade de hardware alocada para o contêiner de névoa, o qual representa um nó de névoa.

- `-cpus=0.1`: máximo de 10% de um núcleo.
- `-memory=64m`: limita a memória RAM a ser utilizada em 64 MB.
- `-memory-swap=128m`: limita o uso combinado entre memória RAM e *swap* em 128 MB.
- `-cpu-shares=64`: é responsável por definir a prioridade do contêiner com relação ao uso da CPU.
- `-device-read-bps=/dev/sda:250k`: limita a taxa de leitura em 250 KB/s.
- `-device-write-bps=/dev/sda:250k`: limita a taxa de escrita em 250 KB/s.

4.3.2 Execução da simulação em ambiente de nuvem

Para a simulação em nuvem, não foram estabelecidas limitações de hardware pois teve-se o objetivo de representar um nó da nuvem com os recursos computacionais virtualmente ilimitados, dado que esse tipo de técnica é empregada em ambientes com ampla capacidade computacional de processamento.

Assim como no cenário de névoa, foi necessário realizar a criação da imagem Docker, a partir do arquivo `Dockerfile`, com o mesmo comando informado no tópico anterior. Em seguida, foi realizada a execução da simulação em nuvem, em todas as topologias, sementes e algoritmos, com o comando que segue:

```
docker run -it -rm -name nuvem -v "/home/user/Área de
trabalho/simulacao_tcc:/app:rw-u $(id -u):$(id -g) simulacao-tcc python3
main.py -topologia (nº da topologia) -semente (nº da semente) -algoritmo
(dijkstra ou astar) -saida (arquivo.csv)
```

Os parâmetros utilizados foram iguais aos da simulação em névoa, exceto pela exclusão das *flags* que representavam as limitações de hardware aplicadas no contêiner daquele contexto. Outra alteração se dá no nome do contêiner, que nesta etapa é denominado como 'nuvem'.

4.3.3 Scripts para a coleta das métricas de desempenho dos contêiners via docker stats

Para a coleta das métricas (`CPU %, MEM USAGE, MEM %, NET INPUT e NET OUTPUT`), foi necessário desenvolver um script que fosse capaz de colher, em tempo real e durante a execução do contêiner, as métricas de desempenho de cada semente, considerando todas as topologias, tipos de algoritmo de roteamento e cenários de execução. O script foi desenvolvido na linguagem Bash.

Primeiramente, ele faz a captura do nome do contêiner a ser monitorado, o qual é fornecido na linha de comando ao executá-lo. Depois, é verificado se o nome fornecido pertence a um contêiner existente, caso contrário o script é encerrado com um erro. Então, é feita a criação do arquivo `.csv` em que serão armazenados os dados obtidos.

Além disso, o código contém uma função responsável por verificar se o contêiner está em execução, a qual será utilizada na coleta de métricas, e apresenta a funcionalidade de esperar pelo início do contêiner para que as execuções da simulação e do script sejam feitas uma de cada vez, a iniciar pela do script.

Por fim, existe um laço de repetição que verifica se o contêiner está sendo executado a partir da função responsável por isso. As métricas são coletadas até que o contêiner seja encerrado, momento em que o laço é interrompido. Os resultados armazenados no arquivo `.csv` contém o nome do contêiner, o percentual de CPU consumido, o uso de memória com relação ao limite, o percentual de memória consumida e o envio e recebimento de dados pela rede. Após o contêiner parar a sua execução, o script exibe uma mensagem e é finalizado. A seguir, a Figura 10 contém o código referente ao script.

```bash
#!/bin/bash

CONTAINER_NAME="$1"

if [ -z "$CONTAINER_NAME" ]; then
    echo "Uso: $0 <nome_do_container>"
    exit 1
fi

OUTPUT_FILE="docker_stats_$(date +'%Y%m%d_%H%M%S').csv"

echo "Timestamp,Container,CPU %,Mem Usage,Mem %,Net I/O,Block I/O,PIDs" > "$OUTPUT_FILE"

container_is_running() {
    docker ps --format '{{.Names}}' | grep -q "^${CONTAINER_NAME}$"
}

echo -n "Aguardando container '$CONTAINER_NAME' iniciar..."
while ! container_is_running; do
    sleep 1
    echo -n "."
done
echo -e "\nContainer encontrado! Iniciando captura contínua..."

# Coletar as métricas continuamente enquanto o container estiver em execução
while container_is_running; do
    # Coleta de métricas do container
    docker stats --no-stream --format "{{.Name}},{{.CPUPerc}},{{.MemUsage}},{{.MemPerc}},{{.NetIO}},{{.BlockIO}},{{.PIDs}}" "$CONTAINER_NAME" | while read -r line; do
        echo "$(date +'%Y-%m-%d %H:%M:%S'),$line" >> "$OUTPUT_FILE"
    done
    # Intervalo de 1 segundo entre as coletas de dados
    sleep 1
done

echo -e "\nContainer $CONTAINER_NAME foi encerrado. Dados salvos em $OUTPUT_FILE"
```

Figura 10 – Script de captura das métricas de desempenho via docker stats

Fonte: Elaborado pelo autor.

O comando de execução do script de coleta de métricas do Docker é mostrado a seguir:

```
./monitor_docker.sh <nuvem> ou <nevoa>
```

4.3.4 Fluxo de execução das simulações e análise dos dados coletados

Em todos os experimentos, o primeiro passo consistiu na execução do script responsável pela obtenção das sementes válidas para cada topologia. A partir disso, foram escolhidas as 10 primeiras sementes válidas de cada topologia, totalizando 50 sementes.

Em seguida, foi criada a imagem Docker. Então, foram abertos dois terminais, sendo o primeiro para a execução do script de coleta de métricas do Docker e o segundo para a execução do código principal do experimento.

A ordem de execução adotada foi a inicialização do script de coletas e, em seguida, a execução do código principal, dado que o script de métricas apresenta um mecanismo responsável por aguardar a inicialização do contêiner e, assim, evitar a execução simultânea e possível perda de métricas por *delay*.

Primeiramente, foram executadas as simulações referentes ao ambiente de névoa. Após a conclusão desta etapa e a organização dos arquivos gerados, foram feitas as experimentações no cenário de nuvem, bem como a organização dos dados obtidos.

Para cada ambiente, em cada topologia e suas sementes, foram gerados dois arquivos: um contendo as métricas obtidas pelo Docker e o outro com os resultados da execução da semente, ambos `.csv`.

Diante disso, os dados obtidos foram organizados em tabelas e realizou-se a análise e comparação de desempenho entre os diferentes cenários, topologias e sementes. Para isso, calcularam-se as médias amostrais ($\bar{x}$) dos tempos de execução obtidos dos algoritmos e dos recursos computacionais consumidos nos ambientes de nuvem e névoa com a Equação (2), onde x_i representa os valores obtidos e n representa o número total de valores.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2}$$

Com as médias geradas, calcularam-se os valores dos desvios padrão amostrais (s) para os tempos de execução obtidos dos algoritmos e dos recursos computacionais consumidos com a Equação (3).

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{3}$$

Por fim, para verificar o grau de variabilidade dos tempos de execução dos algoritmos e das métricas de desempenho coletadas no ambiente de névoa, utilizou-se a Equação (4) para obter os coeficientes de variação (CVs). Nesse contexto, observa-se que um CV baixo indica maior estabilidade das informações.

$$CV = \frac{s}{\bar{x}} \tag{4}$$

5 RESULTADOS

Com relação aos resultados obtidos, nesse capítulo são apresentados os dados referentes aos tempos de execução dos algoritmos Dijkstra e A*, os recursos computacionais consumidos nos contêineres durante a execução deles e a comparação dos dados nos ambientes de névoa e de nuvem.

5.1 TEMPO DE EXECUÇÃO DOS ALGORITMOS DIJKSTRA E A* NO AMBIENTE DE NÉVOA

Os dados obtidos a partir das execuções no contêiner de névoa foram inseridos em tabelas, cujas colunas são compostas pela numeração da semente de execução, pela quantidade de arestas, e pelos tempos de execução, em milissegundos, dos algoritmos de roteamento utilizados.

5.1.1 Topologia de 2000 semáforos inteligentes processada por computação em névoa

A Tabela 2 apresenta as informações de tempo de execução dos algoritmos de roteamento por semente.

Tabela 2 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 2000 nós no contêiner de névoa

Semente	**Qtd. de Arestas**	**Tempo Exec. (ms) - Dijkstra**	**Tempo Exec. (ms) - A***
0	3729	3,490	2,791
2	3957	3,826	2,274
9	2948	9,259	1,601
12	2971	2,739	2,633
17	3069	0,969	90,320
20	3854	94,131	2,297
22	3962	92,809	1,873
23	3894	3,485	2,113
26	3530	93,684	2,300
27	3328	2,985	3,754

Fonte: Elaborado pelo autor.

Com os dados apresentados anteriormente, a Figura 11 expõe o gráfico que evidencia a diferença entre os tempos de execução do Dijkstra e do A* com relação ao número de arestas.

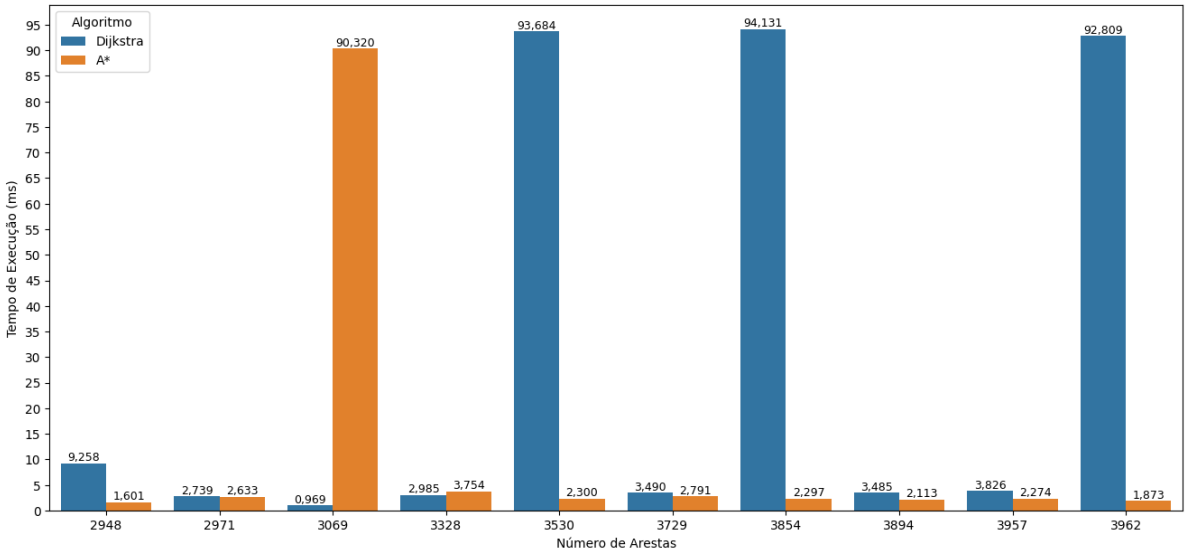

Figura 11 – Tempo de execução por número de arestas na topologia 2000 - Névoa

Fonte: Elaborado pelo autor.

Então, constatou-se que o algoritmo A* apresentou menor tempo de execução em 80% das sementes de simulação. Além disso, foram calculados os tempos de execução médio dos algoritmos Dijkstra e A*, sendo 30,738 ms e 11,196 ms, respectivamente. A partir disso notou-se que, para a topologia de 2000 nós, o A* executa cerca de 2,75 vezes mais rápido do que o Dijkstra, em média.

Também, foram estabelecidos os valores de desvio padrão do tempo de execução médio para cada um dos algoritmos, para o Dijkstra de 43,391 ms e para o A* com 27,808 ms. Ademais, ambos os algoritmos apresentaram altos valores de desvio padrão devido às variações significativas de tempo entre as diferentes execuções. Desse modo, foram obtidos os coeficientes de variação do Dijkstra e do A*, 1,412 e 2,484, respectivamente. Assim, notou-se que o Dijkstra apresentou maior estabilidade mesmo sendo mais lento que o A*.

5.1.2 Topologia de 3000 semáforos inteligentes processada por computação em névoa

A Tabela 3 expõe os dados de tempo de execução dos algoritmos de roteamento aplicados na topologia composta por 3000 semáforos inteligentes.

Tabela 3 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 3000 nós no contêiner de névoa

Semente	**Qtd. de Arestas**	**Tempo Exec. (ms) - Dijkstra**	**Tempo Exec. (ms) - A***
3	3974	2,409	2,130
4	3966	1,635	1,158
6	5350	94,019	93,734
11	4852	2,914	92,849
12	4943	1,941	91,782
17	5138	93,710	93,129
18	3742	1,634	1,296
29	5245	94,361	94,276
30	5208	1,408	1,284
33	5336	0,763	0,416

Fonte: Elaborado pelo autor.

O gráfico observado na Figura 12 demonstra a diferença entre os tempos de execução do Dijkstra e do A* com relação ao número de arestas.

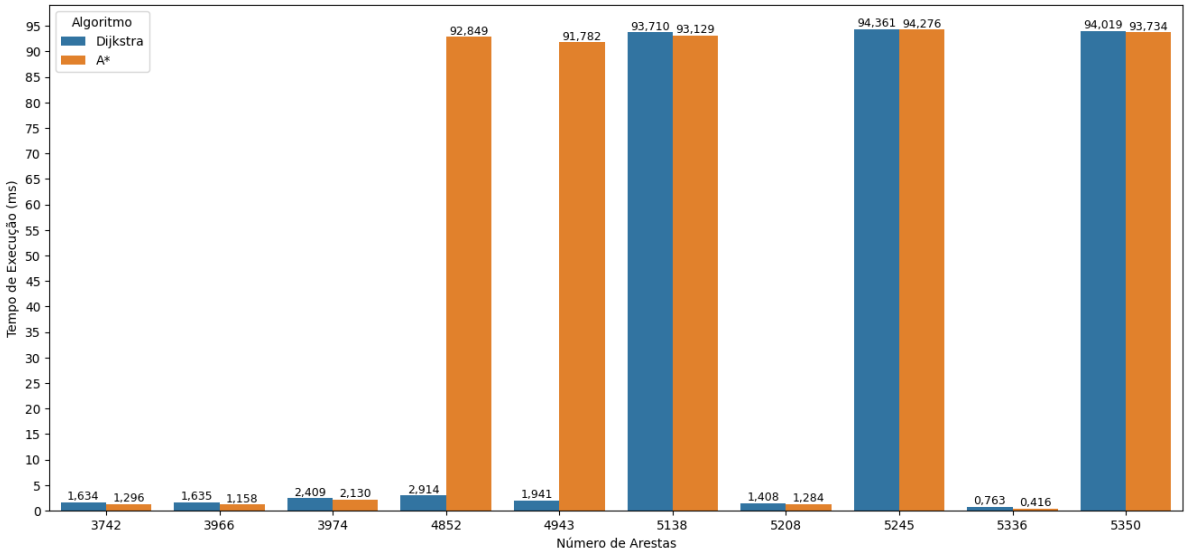

Figura 12 – Tempo de execução por número de arestas na topologia 3000 - Névoa

Fonte: Elaborado pelo autor.

A partir dele, percebeu-se que, assim como para a topologia de 2000 semáforos inteligentes, o A* apontou menores tempos de execução em 80% das execuções. Os tempos de execução médio para o Dijkstra e para o A* também foram calculados, 29,479 ms e 47,205 ms, em ordem. Então, detectou-se um comportamento diferente do que foi visto para a topologia de 2000 nós. Aqui, o Dijkstra foi cerca de 1,6 vezes mais rápido em média do que o A*.

Por fim, os valores de desvio padrão dos tempos de execução médios para os algoritmos são 44,548 ms para o Dijkstra e 48,440 ms para o A*. Além disso, foram determinados os coeficientes de variação, sendo 1,511 para o Dijkstra e 1,026 para o A*. Logo, nesse caso, o algoritmo A* mostrou maior estabilidade.

5.1.3 Topologia de 4000 semáforos inteligentes processada por computação em névoa

A Tabela 4 expõe as informações referentes ao tempo de execução dos algoritmos de roteamento que foram aplicados na topologia composta por 4000 semáforos inteligentes.

Tabela 4 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 4000 nós no contêiner de névoa

Semente	Qtd. de Arestas	Tempo Exec. (ms) - Dijkstra	Tempo Exec. (ms) - A*
2	7915	98,586	98,618
3	4974	1,245	1,075
5	6551	91,197	1,059
6	7249	90,711	0,362
9	5896	5,378	4,901
10	6340	94,369	4,468
19	6773	0,877	0,640
20	7709	97,954	100,598
22	7924	5,617	94,744
23	7788	2,916	3,809

Fonte: Elaborado pelo autor.

A Figura 13 torna evidente a diferença entre os tempos de execução do Dijkstra e do A* com relação ao número de arestas.

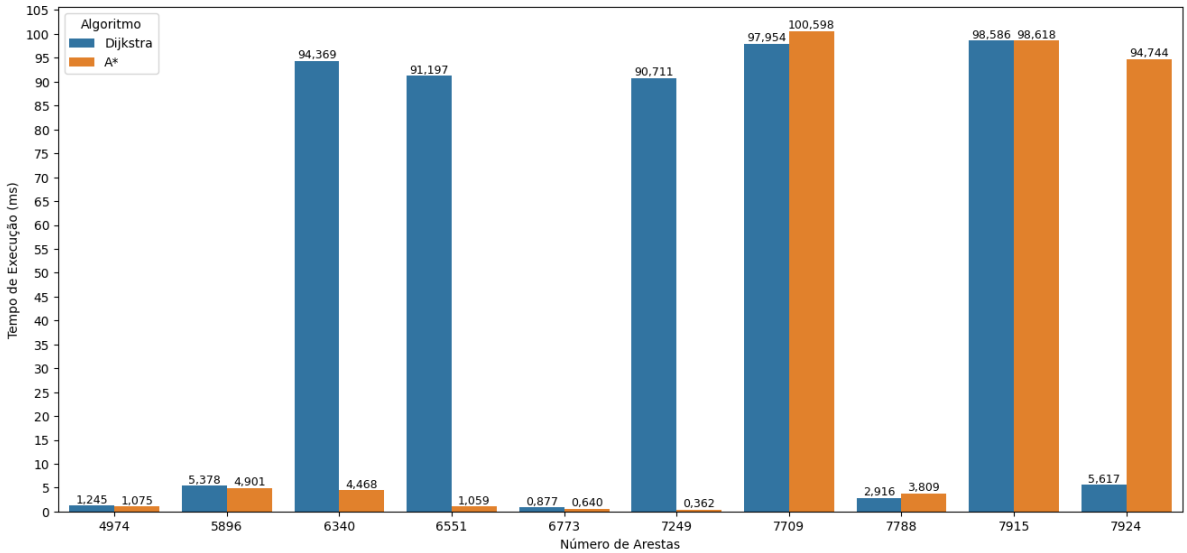

Figura 13 – Tempo de execução por número de arestas na topologia 4000 - Névoa

Fonte: Elaborado pelo autor.

Então, constatou-se que o algoritmo A* teve menor tempo de execução em 60% das simulações realizadas, considerando a topologia de 4000 nós no ambiente de névoa. Os tempos de execução médio do Dijkstra e do A* são, respectivamente, 48,885 ms e 31,028 ms. A partir dos dados, observou-se que o A* foi 1,58 vezes mais rápido do que o Dijkstra, em média.

Os valores de desvio padrão do tempo de execução médio dos algoritmos são, na mesma ordem, 48,234 ms e 46,256 ms. Também, ambos os algoritmos apresentaram altos valores para o desvio padrão e teve-se, como coeficientes de variação, 0,987 para o Dijkstra e 1,491 para o A*. Isso indica que o Dijkstra teve maior estabilidade em relação ao tempo de execução, mesmo sendo mais lento.

5.1.4 Topologia de 5000 semáforos inteligentes processada por computação em névoa

A Tabela 5 apresenta as informações de tempo de execução dos algoritmos de roteamento aplicados na topologia composta por 5000 semáforos inteligentes.

Tabela 5 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 5000 nós no contêiner de névoa

Semente	Qtd. de Arestas	Tempo Exec. (ms) - Dijkstra	Tempo Exec. (ms) - A*
0	8155	92,652	1,256
4	6933	4,802	5,019
6	9700	199,256	6,527
9	8792	96,314	97,803
11	8705	95,976	97,265
12	8887	97,332	8,095
15	6711	4,600	94,406
25	8088	1,065	0,655
27	8931	1,447	0,901
30	9416	98,312	98,821

Fonte: Elaborado pelo autor.

O gráfico presente na Figura 14 expõe a diferença entre os tempos de execução do Dijkstra e do A* com diferentes números de arestas.

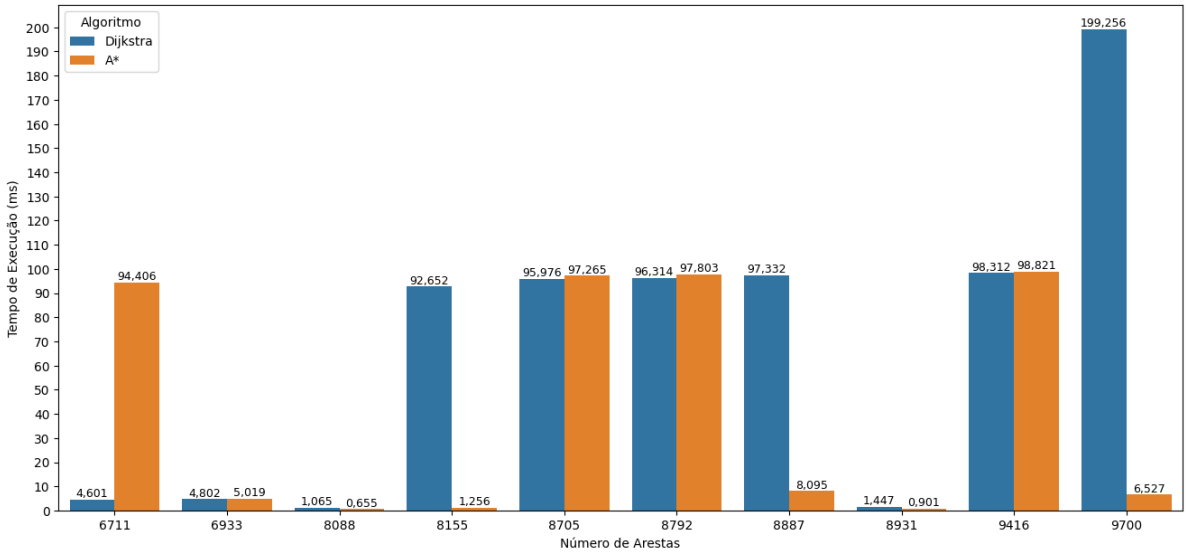

Figura 14 – Tempo de execução por número de arestas na topologia 5000 - Névoa

Fonte: Elaborado pelo autor.

A partir do gráfico, detectou-se que em 50% das execuções o A* obteve um melhor tempo de execução e nos outros 50% o Dijkstra teve o mesmo feito. Foram obtidos os valores médios de tempo de execução do Dijkstra e do A*, sendo 69,176 ms e 41,075 ms, respectivamente. Então, viu-se que o A* é cerca de 1,68 vezes mais rápido do que o Dijkstra no tempo médio.

Também, foram calculados os valores de desvio padrão com relação aos tempos de execução médios do Dijkstra, 65,071 ms, e do A*, 48,269 ms. Ainda, obteve-se como coeficientes de variação 0,941 para o Dijkstra e 1,175 para o A*. Logo, o Dijkstra expôs melhor estabilidade.

5.1.5 Topologia de 6000 semáforos inteligentes processada por computação em névoa

A Tabela 6 expõe os dados de tempo de execução dos algoritmos de roteamento aplicados na topologia composta por 6000 semáforos inteligentes.

Tabela 6 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 6000 nós no contêiner de névoa

Semente	**Qtd. de Arestas**	**Tempo Exec. (ms) - Dijkstra**	**Tempo Exec. (ms) - A***
6	10700	98,420	95,873
8	7857	3,336	2,708
11	9705	90,894	0,839
20	11921	101,516	99,645
22	7149	1,118	0,943
23	8374	196,000	5,053
24	11835	100,747	97,523
29	10490	100,410	101,386
33	10672	2,381	92,108
37	11586	1,536	1,304

Fonte: Elaborado pelo autor.

O gráfico apresentado na Figura 15 demonstra a diferença entre os tempos de execução do Dijkstra e do A* com relação ao número de arestas.

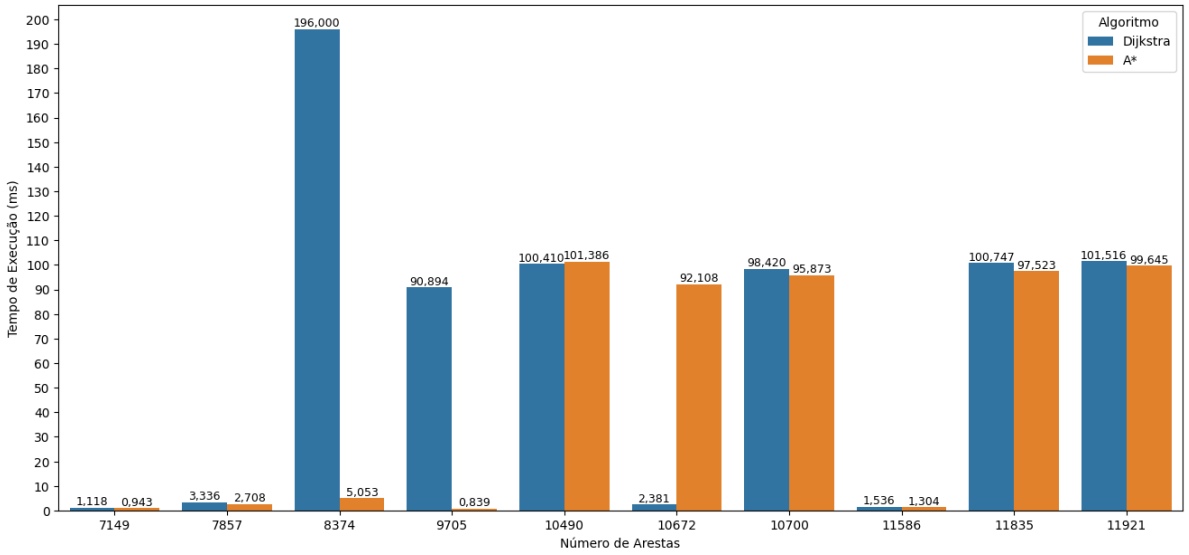

Figura 15 – Tempo de execução por número de arestas na topologia 6000 - Névoa

Fonte: Elaborado pelo autor.

A partir da visualização dos dados comparativos, observou-se que o algoritmo A* apresentou melhor tempo de execução em 80% das sementes de simulação. Ainda, foram obtidos os tempos médio de execução para o Dijkstra e para o A*, sendo 69,636 ms e 49,738 ms, respectivamente. Devido a esses valores, constatou-se que o A* foi 1,40 vezes mais rápido que o Dijkstra, em média.

Com relação ao desvio padrão do tempo de execução médio dos algoritmos, eles obtiveram 65,346 ms e 50,213 ms, na mesma ordem. Além disso, ambos os algoritmos apresentaram alta variabilidade, entretanto, o Dijkstra demonstrou maior estabilidade pois seu coeficiente de variabilidade foi de 0,938, enquanto o do A* de 1,009.

5.2 TEMPO DE EXECUÇÃO DOS ALGORITMOS DIJKSTRA E A* NO AMBIENTE DE NUVEM

As informações coletadas a partir das execuções no contêiner de nuvem foram organizadas em tabelas, cujas colunas indicam a numeração da semente de execução, a quantidade de arestas e os tempos de execução, em milissegundos, dos algoritmos de roteamento utilizados.

5.2.1 Topologia de 2000 semáforos inteligentes processada por computação em nuvem

A Tabela 7 apresenta os tempos de execução obtidos, a partir da execução dos algoritmos de roteamento, para a topologia de 2000 semáforos inteligentes.

Tabela 7 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 2000 nós no contêiner de nuvem

Semente	**Qtd. de Arestas**	**Tempo Exec. (ms) - Dijkstra**	**Tempo Exec. (ms) - A***
0	3729	3,163	2,692
2	3957	3,974	2,276
9	2948	1,981	1,452
12	2971	2,358	2,400
17	3069	0,873	0,582
20	3854	3,891	2,210
22	3962	3,277	1,822
23	3894	3,186	2,083
26	3530	2,771	2,176
27	3328	2,739	2,927

Fonte: Elaborado pelo autor.

O gráfico a seguir, mostrado na Figura 16, realiza um comparativo entre os tempos de execução dos algoritmos de roteamento para cada quantidade de arestas.

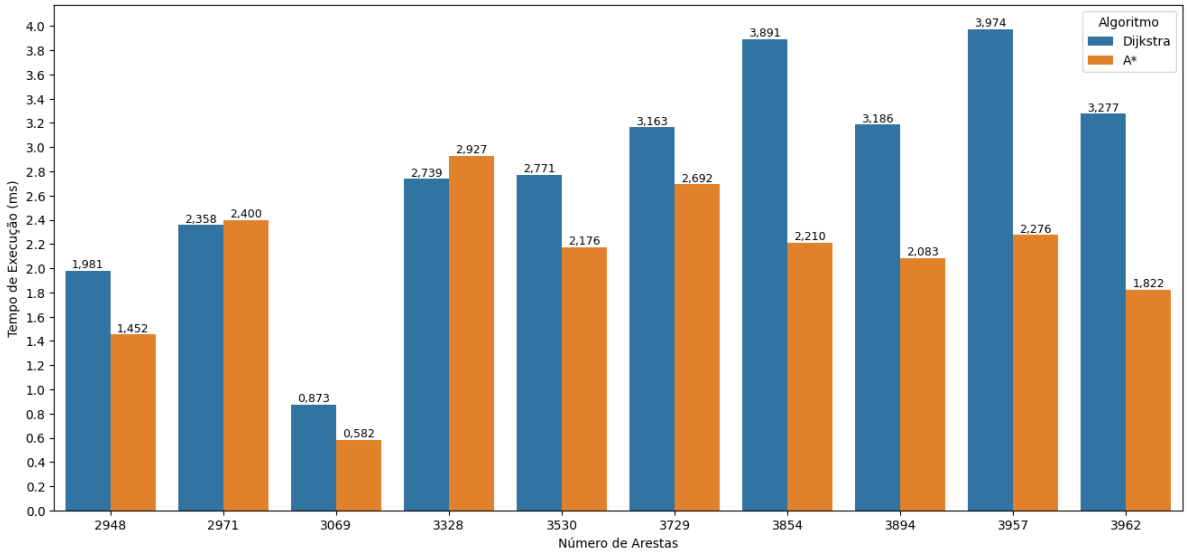

Figura 16 – Tempo de execução por número de arestas na topologia 2000 - Nuvem

Fonte: Elaborado pelo autor.

Então, a partir das informações apresentadas, verificou-se que o A* apresentou melhor tempo de execução em 80% das sementes de simulação. O tempo de execução médio do Dijkstra foi de 2,821 ms, enquanto que o do A* foi de 2,062 ms. Tais valores mostraram que o A* é cerca de 1,37 vezes mais rápido, em média, do que o Dijkstra.

Já o desvio padrão do tempo de execução médio do Dijkstra foi de 0,922 ms e o do A* foi de 0,663 ms. Ainda, foram obtidos os valores dos coeficientes de variação, sendo 0,327 para o Dijkstra e 0,322 para o A*. Logo, o algoritmo A* é ligeiramente mais estável com relação ao tempo de execução.

5.2.2 Topologia de 3000 semáforos inteligentes processada por computação em nuvem

A Tabela 8 evidencia os tempos de execução obtidos, a partir da execução dos algoritmos de roteamento, para a topologia de 3000 semáforos inteligentes.

Tabela 8 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 3000 nós no contêiner de nuvem

Semente	Qtd. de Arestas	Tempo Exec. (ms) - Dijkstra	Tempo Exec. (ms) - A*
3	3974	2,330	2,058
4	3966	1,552	1,108
6	5350	4,290	3,387
11	4852	2,854	2,431
12	4943	1,869	1,781
17	5138	3,827	2,897
18	3742	1,520	1,409
29	5245	4,668	3,916
30	5208	1,418	1,243
33	5336	0,729	0,409

Fonte: Elaborado pelo autor.

O gráfico exposto na Figura 17 faz um comparativo entre os tempos de execução do Dijkstra e do A* para cada quantidade de arestas.

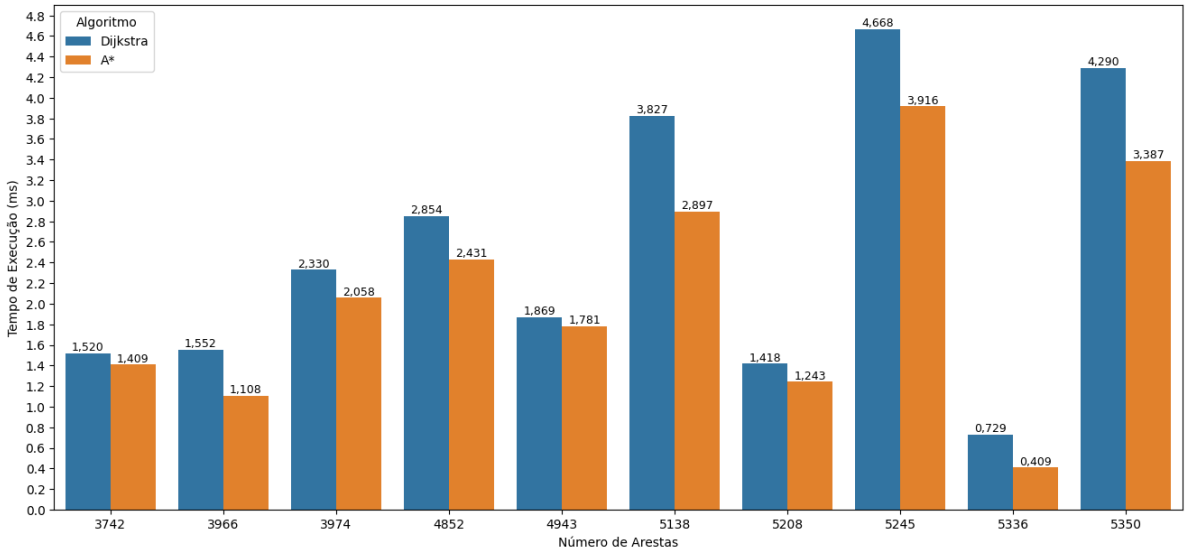

Figura 17 – Tempo de execução por número de arestas na topologia 3000 - Nuvem

Fonte: Elaborado pelo autor.

Logo, constatou-se que o algoritmo A* apresentou o melhor tempo de execução em 100% das sementes de simulação. A partir dos dados obtidos, teve-se que o tempo de execução médio do Dijkstra foi de 2,506 ms e o do A* de 2,064 ms. Com as informações conseguidas, concluiu-se que o A* é mais rápido, em média, cerca de 1,21 vezes quando comparado ao Dijkstra.

Já o desvio padrão do tempo de execução médio do Dijkstra foi de 1,349 ms e o do A* de 1,098 ms. Além disso, calculou-se os valores dos coeficientes de variação de ambos os algoritmos, sendo 0,538 para o Dijkstra e 0,532 para o A*. Portanto, o A* teve estabilidade minimamente maior com relação ao tempo de execução nesse cenário.

5.2.3 Topologia de 4000 semáforos inteligentes processada por computação em nuvem

A seguir, é exposta a Tabela 9 que evidencia os tempos de execução obtidos, a partir da execução dos algoritmos de roteamento, para a topologia de 4000 semáforos inteligentes.

Tabela 9 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 4000 nós no contêiner de nuvem

Semente	Qtd. de Arestas	Tempo Exec. (ms) - Dijkstra	Tempo Exec. (ms) - A*
2	7915	7,754	10,212
3	4974	1,144	0,940
5	6551	1,116	0,984
6	7249	0,757	0,349
9	5896	4,727	4,629
10	6340	4,384	4,654
19	6773	0,850	0,663
20	7709	7,457	9,156
22	7924	5,038	3,445
23	7788	2,823	3,526

Fonte: Elaborado pelo autor.

Na Figura 18, o gráfico evidenciado realiza um comparativo entre os tempos de execução dos algoritmos Dijkstra e do A* em relação a quantidade de arestas.

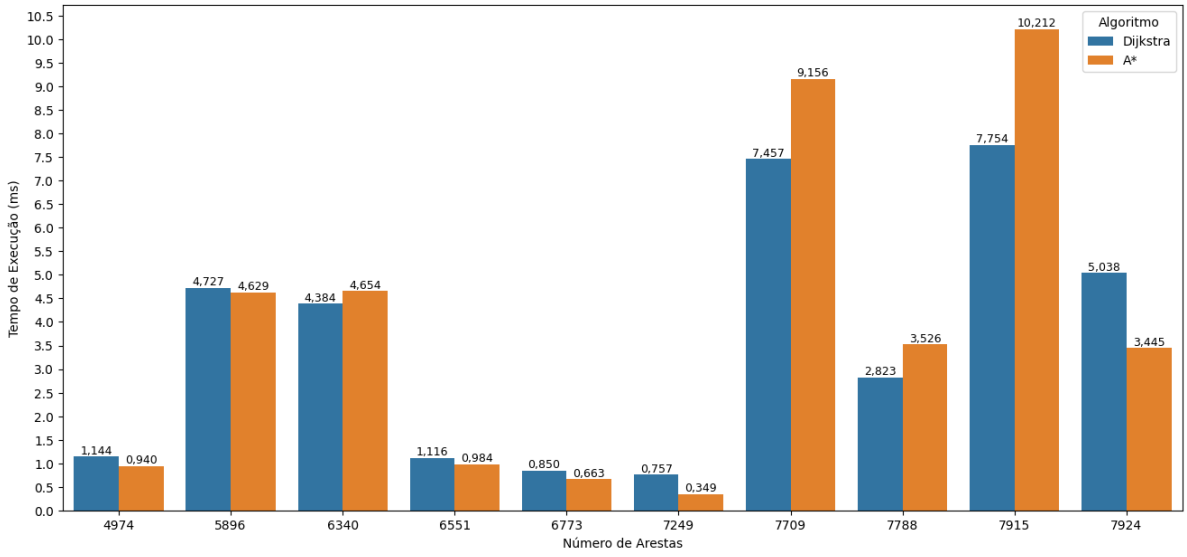

Figura 18 – Tempo de execução por número de arestas na topologia 4000 - Nuvem

Fonte: Elaborado pelo autor.

A partir das informações expostas, verificou-se que o algoritmo A* apresentou melhor tempo de execução em 60% das sementes de simulação. Ainda, teve-se que o tempo de execução médio do Dijkstra foi de 3,605 ms e o do A* de 3,856 ms. Então, constatou-se que o Dijkstra é, em média, cerca de 1,07 vezes mais rápido do que o A*.

Já o desvio padrão do tempo de execução médio do Dijkstra foi de 2,678 ms e o do A* de 3,484 ms. Com relação aos coeficientes de variação, o do Dijkstra foi de 0,743 e o do A* de 0,904. Logo, nesse cenário, o algoritmo Dijkstra demonstrou maior estabilidade.

5.2.4 Topologia de 5000 semáforos inteligentes processada por computação em nuvem

A Tabela 10 expõe os dados de tempo de execução dos algoritmos aplicados na topologia composta por 5000 semáforos inteligentes.

Tabela 10 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 5000 nós no contêiner de nuvem

Semente	Qtd. de Arestas	Tempo Exec. (ms) - Dijkstra	Tempo Exec. (ms) - A*
0	8155	1,745	1,154
4	6933	4,698	4,742
6	9700	18,553	7,017
9	8792	5,370	10,566
11	8705	5,061	6,857
12	8887	6,453	6,718
15	6711	3,717	4,118
25	8088	1,090	0,667
27	8931	1,395	0,873
30	9416	8,215	8,382

Fonte: Elaborado pelo autor.

Na Figura 19 vê-se a diferença entre os tempos de execução do Dijkstra e do A*.

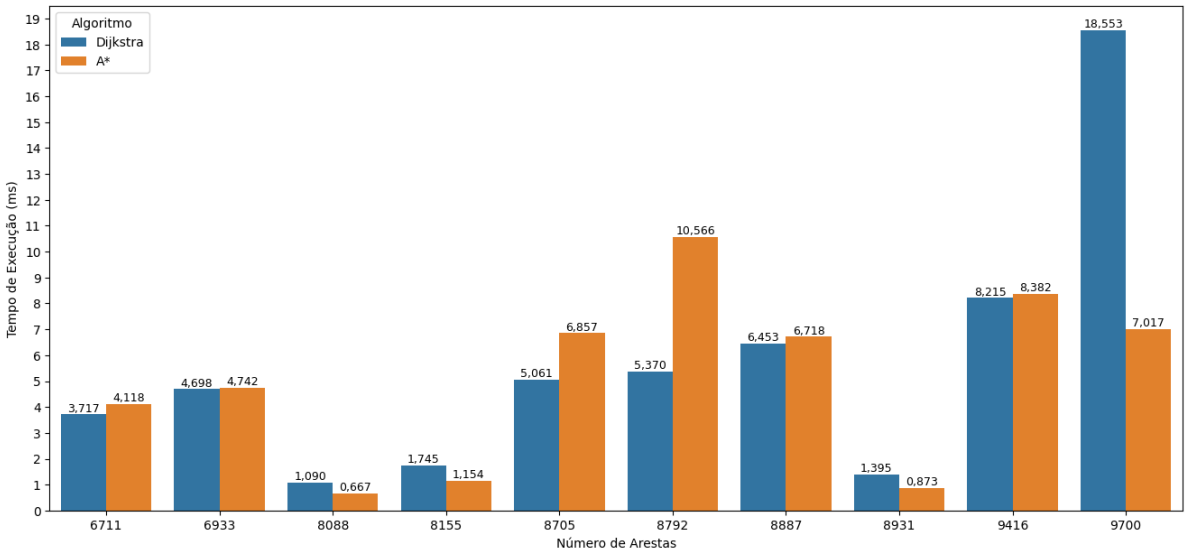

Figura 19 – Tempo de execução por número de arestas na topologia 5000 - Nuvem

Fonte: Elaborado pelo autor.

Então, constatou-se que o algoritmo Dijkstra apresentou melhor tempo de execução em 60% das sementes de simulação. Ademais, teve-se que o tempo de execução médio do Dijkstra foi de 5,630 ms e o do A* de 5,109 ms. Logo, verificou-se que o A* é, em média, cerca de 1,10 vezes mais rápido que o algoritmo de Dijkstra.

Já o desvio padrão do tempo de execução médio do Dijkstra foi de 5,087 ms e o do A* de 3,403 ms. Além disso, foram obtidos os valores dos coeficientes de variação do Dijkstra e do A*, 0,904 e 0,666 respectivamente. Então, o A* mostrou maior estabilidade.

5.2.5 Topologia de 6000 semáforos inteligentes processada por computação em nuvem

A Tabela 11 apresenta os tempos de execução obtidos, a partir da execução dos algoritmos de roteamento, para a topologia de 6000 semáforos inteligentes.

Tabela 11 – Tempo de execução dos algoritmos Dijkstra e A* na topologia de 6000 nós no contêiner de nuvem

Semente	Qtd. de Arestas	Tempo Exec. (ms) - Dijkstra	Tempo Exec. (ms) - A*
6	10700	7,890	5,338
8	7857	3,045	2,699
11	9705	0,925	0,823
20	11921	9,815	8,796
22	7149	1,100	0,976
23	8374	15,418	4,506
24	11835	9,728	7,277
29	10490	9,154	10,596
33	10672	2,313	2,212
37	11586	1,535	1,299

Fonte: Elaborado pelo autor.

A Figura 20 faz um comparativo entre os tempos de execução do Dijkstra e do A*.

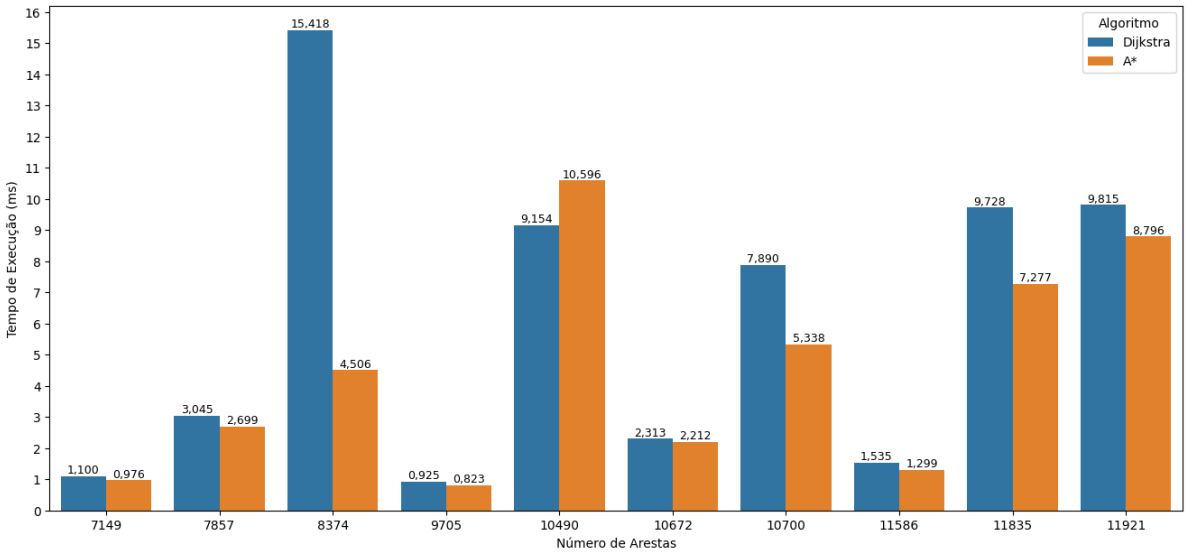

Figura 20 – Tempo de execução por número de arestas na topologia 6000 - Nuvem

Fonte: Elaborado pelo autor.

A partir dos dados expostos, viu-se que o algoritmo A* apresentou melhor tempo de execução em 90% das sementes de simulação. Também, teve-se que o tempo de execução médio do Dijkstra foi de 6,092 ms e o do A* de 4,523 ms. Com isso, inferiu-se que o A* é, em média, cerca de 1,35 vezes mais rápido que o Dijkstra.

Já o desvio padrão do tempo de execução médio do Dijkstra foi de 4,973 ms e o do A* de 3,475 ms. Outrossim, foram estabelecidos os coeficientes de variação dos algoritmos, que foram 0,816 para o Dijkstra e 0,768 para o A*. A partir disso, foi evidenciado que o A* apresentou maior estabilidade.

5.3 TEMPOS DE EXECUÇÃO MÉDIOS DOS ALGORITMOS DE ROTEAMENTO NOS AMBIENTES DE NÉVOA E NUVEM

Os dados de tempo de execução médio foram organizados na Tabela 12.

Tabela 12 – Tempos de execução médios dos algoritmos

Topologia	**Dijkstra - Névoa (ms)**	**A* - Névoa (ms)**	**Dijkstra - Nuvem (ms)**	**A* - Nuvem (ms)**
2000	30,738	11,196	2,821	2,062
3000	29,479	47,205	2,506	2,064
4000	48,885	31,028	3,605	3,856
5000	69,176	41,075	5,630	5,109
6000	69,636	49,738	6,092	4,523

Fonte: Elaborado pelo autor.

A partir do que foi mostrado pela Figura 21, percebeu-se que, no contexto de névoa, o algoritmo A* apresentou melhor tempo de execução médio em 4 das 5 topologias adotadas para simulação. A única discrepância deu-se na topologia de 3000 nós, em que o tempo de execução médio do Dijkstra foi menor.

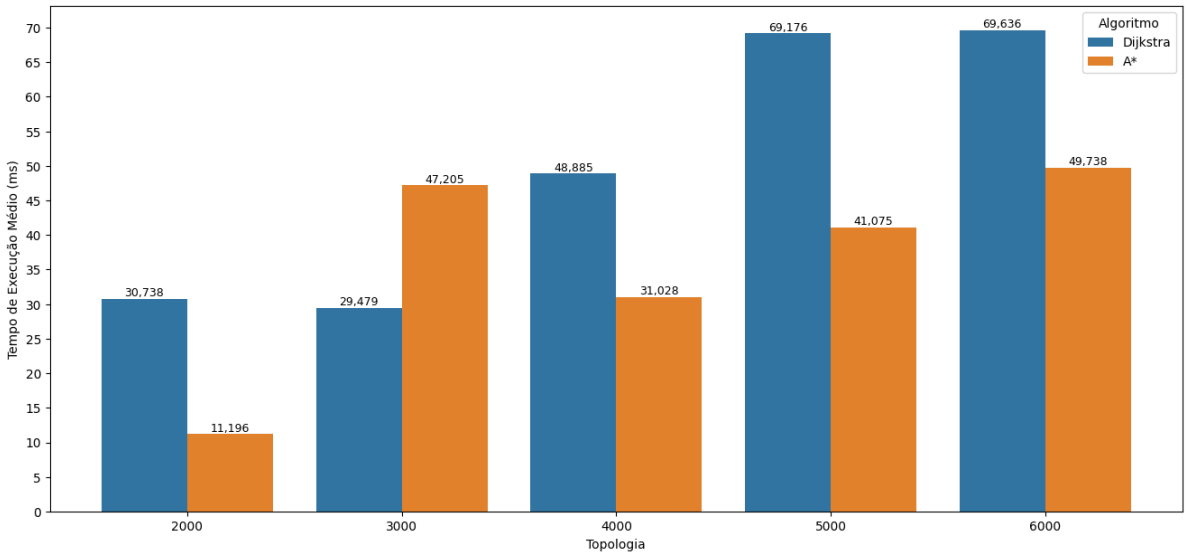

Figura 21 – Tempo de execução médio por topologia - Névoa

Fonte: Elaborado pelo autor.

Já a Figura 22 expõe que, no contexto de nuvem, o algoritmo A* teve melhor tempo de execução médio em 4 das 5 topologias. Porém, a exceção se deu na topologia de 4000 nós, em que o Dijkstra teve um tempo de execução médio sutilmente menor.

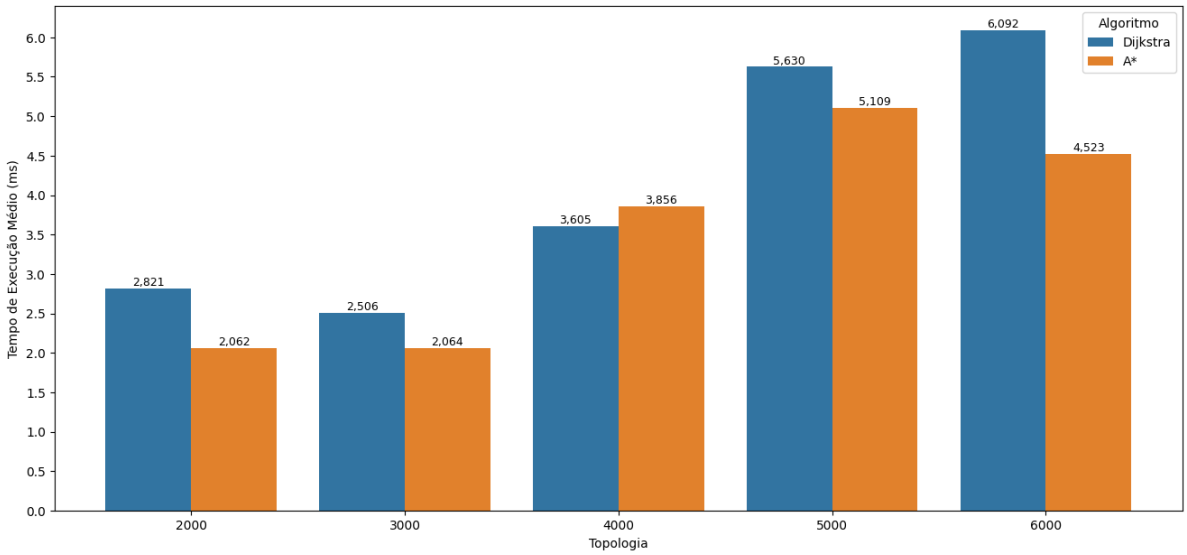

Figura 22 – Tempo de execução médio por topologia - Nuvem

Fonte: Elaborado pelo autor.

5.4 RECURSOS COMPUTACIONAIS CONSUMIDOS A PARTIR DA EXECUÇÃO DOS ALGORITMOS DE ROTEAMENTO NO CONTEXTO DE NÉVOA POR TOPOLOGIA

Para a simulação no ambiente de névoa, foram estabelecidas limitações na configuração do contêiner com relação a quantidade de CPU disponível para uso, a quantia de memória RAM acessível e as taxas de leitura e escrita.

Referente a CPU, foi designado o limite máximo de uso de 10% de um núcleo, logo, os valores percentuais apresentados nas tabelas têm como referência o máximo de 10%. Já com relação à memória RAM, definiu-se o limite máximo de consumo como 64 mebibytes (MiB).

Para cada topologia, o parâmetro de escolha entre os algoritmos foi a melhor estabilidade na sua execução visto que, no contexto de névoa, os nós de névoa costumam apresentar alta limitação de recursos. Então, nesse caso, prevaleceu-se o objetivo de obter uma execução consistente a fim de manter os serviços em pleno funcionamento considerando as restrições de hardware existentes. Posto isso, as análises das métricas foram feitas com base, principalmente, nos valores de coeficiente de variação obtidos.

5.4.1 Consumo médio no processamento de 2000 semáforos inteligentes em névoa

A Tabela 13 mostra o consumo médio de recursos computacionais, de cada semente de simulação, a partir da execução do algoritmo de Dijkstra.

Tabela 13 – Consumo médio por semente na topologia 2000 (Dijkstra/Névoa)

Semente	**CPU %**	**Mem Usage (MiB)**	**Mem %**	**Net Input (B)**	**Net Output (B)**
0	9,91	23,630	36,92	2125,76	112,00
2	9,88	33,150	51,80	4096,00	126,00
9	9,88	25,045	39,13	1537,88	105,00
12	9,89	22,975	35,90	1894,40	105,00
17	9,86	23,100	36,10	1662,88	105,00
20	10,00	25,486	39,82	2307,41	126,00
22	9,91	25,473	39,80	2461,01	126,00
23	9,88	25,080	39,19	2125,76	112,00
26	9,92	25,020	39,09	1945,60	126,00
27	9,96	24,915	38,93	1935,36	105,00

Fonte: Elaborado pelo autor.

Já a Tabela 14 expõe o consumo médio de recursos computacionais a partir da execução do algoritmo A*.

Tabela 14 – Consumo médio por semente na topologia 2000 (A*/Névoa)

Semente	**CPU %**	**Mem Usage (MiB)**	**Mem %**	**Net Input (B)**	**Net Output (B)**
0	10,00	25,005	39,07	2211,84	126,00
2	9,88	25,380	39,65	2461,01	112,00
9	9,89	25,005	39,07	1894,40	126,00
12	9,87	24,860	38,84	2283,52	105,00
17	10,00	23,250	36,33	1764,38	105,00
20	9,87	25,350	39,61	2461,01	126,00
22	9,90	23,710	37,04	2125,76	112,00
23	9,81	25,483	39,82	2461,01	126,00
26	9,84	25,140	39,29	2329,60	126,00
27	9,86	25,180	39,35	1945,60	126,00

Fonte: Elaborado pelo autor.

A partir dos dados das tabelas, foram calculadas as médias do percentual de CPU e de memória RAM consumidos e do envio e recebimento de dados, além do desvio padrão para tais métricas. As informações são expostas na Tabela 15.

Tabela 15 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 2000 - Névoa

Algoritmo	**Dijkstra**	**A***
Média - CPU (%)	9,909	9,892
Desvio padrão - CPU (%)	0,043	0,062
Média - Memória (%)	39,668	38,807
Desvio padrão - Memória (%)	4,511	1,169
Média - Net Input (B)	2209,206	2193,813
Desvio padrão - Net Input (B)	717,999	254,142
Média - Net Output (B)	114,800	119,000
Desvio padrão - Net Output (B)	10,009	9,333

Fonte: Elaborado pelo autor.

Além disso, foram calculados os coeficientes de variação (CV) das métricas, como apresenta a Tabela 16.

Tabela 16 – Coeficientes de variação das métricas de desempenho na topologia 2000 - Névoa

Métrica	**CV Dijkstra**	**CV A***
CPU (%)	0,00434	0,00627
Memória (%)	0,11372	0,03012
Net Input (B)	0,32500	0,11584
Net Output (B)	0,08719	0,07843

Fonte: Elaborado pelo autor.

Tomando como referência os dados das tabelas, viu-se que o algoritmo Dijkstra deteve melhor constância durante a sua execução com relação a CPU, porém, um consumo médio discretamente maior desse recurso. Com relação a memória RAM e a entrada de dados, o A* apresentou melhor estabilidade e menor consumo médio em ambos os casos. No tráfego da saída de dados na interface de rede do contêiner, o Dijkstra apresentou melhor uso médio, contudo, o A* teve um maior grau de estabilidade. A partir disso, considera-se que o algoritmo A* é a melhor opção para a topologia de 2000 nós no contexto de computação em névoa.

5.4.2 Consumo médio no processamento de 3000 semáforos inteligentes em névoa

A Tabela 17 evidencia o consumo médio de recursos computacionais pelo algoritmo de Dijkstra quando este foi executado.

Tabela 17 – Consumo médio por semente na topologia 3000 (Dijkstra/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
3	9,79	25,130	39,27	2345,90	115,50
4	9,91	26,150	40,86	2544,64	126,00
6	9,87	26,564	41,51	2803,71	126,00
11	9,94	25,120	39,25	2577,98	117,60
12	9,86	26,240	41,00	2608,64	126,00
17	9,90	26,462	41,35	2887,68	126,00
18	9,95	24,738	38,65	2549,76	115,50
29	9,88	25,684	40,13	2738,18	117,60
30	9,91	26,504	41,41	2887,68	126,00
33	9,86	25,530	39,89	2779,14	117,60

Fonte: Elaborado pelo autor.

Referente ao algoritmo A*, a Tabela 18 apresenta dados relativos ao consumo de recursos durante a sua execução.

Tabela 18 – Consumo médio por semente na topologia 3000 (A*/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
3	9,91	26,070	40,74	2493,44	126,00
4	9,90	26,172	40,90	2603,52	115,50
6	9,92	26,496	41,40	2887,68	126,00
11	9,91	25,105	39,22	2345,90	115,50
12	9,91	26,262	41,04	2608,64	126,00
17	9,96	25,514	39,87	2670,59	117,60
18	9,85	24,617	38,46	2524,16	115,50
29	9,89	26,544	41,47	2887,68	126,00
30	9,92	25,546	39,92	2715,65	117,60
33	9,90	26,484	41,38	2715,65	126,00

Fonte: Elaborado pelo autor.

Tendo como ponto de partida as informações das tabelas, foram determinadas as médias do percentual de CPU e de memória RAM consumidos e do envio e recebimento de dados, além do desvio padrão para tais métricas. Os dados são mostrados na Tabela 19.

Tabela 19 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 3000 - Névoa

Métrica	**Dijkstra**	**A***
Média - CPU (%)	9,887	9,907
Desvio padrão - CPU (%)	0,046	0,028
Média - Memória (%)	40,332	40,440
Desvio padrão - Memória (%)	1,037	1,029
Média - Net Input (B)	2672,331	2645,291
Desvio padrão - Net Input (B)	175,309	169,476
Média - Net Output (B)	121,380	121,170
Desvio padrão - Net Output (B)	4,930	5,149

Fonte: Elaborado pelo autor.

Também, os coeficientes de variação foram determinados, como mostra a Tabela 20.

Tabela 20 – Coeficientes de variação das métricas de desempenho na topologia 3000 - Névoa

Métrica	**CV Dijkstra**	**CV A***
CPU (%)	0,00465	0,00283
Memória (%)	0,02571	0,02545
Net Input (B)	0,06560	0,06407
Net Output (B)	0,04062	0,04249

Fonte: Elaborado pelo autor.

Segundo os dados expostos nas tabelas, o algoritmo A* teve maior estabilidade no uso da CPU, contudo, apresentou maior consumo médio. Com relação a memória RAM, o algoritmo Dijkstra possuiu consumo médio um pouco melhor, porém, o A* apresentou um grau de constância ligeiramente maior. No tráfego de entrada de dados, o A* deteve melhor uso médio e maior estabilidade. No tráfego de saída de dados, os algoritmos tiveram um consumo médio semelhante, mas o Dijkstra alcançou melhor estabilidade. Logo, infere-se que o algoritmo A* foi a melhor opção para a topologia de 3000 nós.

5.4.3 Consumo médio no processamento de 4000 semáforos inteligentes em névoa

Os dados referentes ao consumo médio de recursos computacionais pelo algoritmo de Dijkstra durante a sua execução na topologia de 4000 nós são expostos na Tabela 21.

Tabela 21 – Consumo médio por semente na topologia 4000 (Dijkstra/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
2	9,90	27,791	43,42	3091,34	121,33
3	9,88	25,793	40,30	2589,64	119,00
5	9,87	27,312	42,67	3012,02	126,00
6	9,88	26,708	41,73	2844,63	120,75
9	9,92	26,264	41,04	2756,56	120,00
10	9,90	27,268	42,61	3042,74	126,00
19	9,89	27,257	42,59	3042,74	126,00
20	9,91	27,608	43,14	3022,08	126,00
22	9,90	28,033	43,80	3809,28	126,00
23	9,91	27,583	43,10	3077,12	126,00

Fonte: Elaborado pelo autor.

As informações relativas aos recursos computacionais que foram consumidos durante a execução do algoritmo A* são mostradas na Tabela 22.

Tabela 22 – Consumo médio por semente na topologia 4000 (A*/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
2	9,87	26,921	42,06	2896,06	121,33
3	9,89	27,836	43,49	3284,99	126,00
5	9,88	27,485	42,95	3592,78	126,00
6	9,91	27,482	42,94	3104,00	126,00
9	9,93	29,411	45,95	3440,64	126,00
10	9,87	27,355	42,74	3360,43	126,00
19	9,91	27,574	43,08	3464,04	126,00
20	9,75	27,552	43,05	3516,16	126,00
22	9,90	27,791	43,42	3412,48	126,00
23	9,92	27,278	42,62	2994,18	126,00

Fonte: Elaborado pelo autor.

Com base nos dados das tabelas, as médias do percentual de CPU e de memória RAM consumidos e do envio e recebimento de dados foram determinadas, além dos valores de desvio padrão. As informações são apresentadas na Tabela 23.

Tabela 23 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 4000 - Névoa

Métrica	**Dijkstra**	**A***
Média - CPU (%)	9,896	9,883
Desvio padrão - CPU (%)	0,016	0,051
Média - Memória (%)	42,440	43,230
Desvio padrão - Memória (%)	1,100	1,038
Média - Net Input (B)	3028,815	3306,576
Desvio padrão - Net Input (B)	319,201	233,414
Média - Net Output (B)	123,708	125,533
Desvio padrão - Net Output (B)	3,015	1,477

Fonte: Elaborado pelo autor.

Ademais, determinou-se os valores dos coeficientes de variação das métricas, como exposto na Tabela 24.

Tabela 24 – Coeficientes de variação das métricas de desempenho na topologia 4000 - Névoa

Métrica	**Dijkstra**	**A***
CPU (%)	0,00162	0,00516
Memória (%)	0,02592	0,02401
Net Input (B)	0,10539	0,07059
Net Output (B)	0,02437	0,01177

Fonte: Elaborado pelo autor.

Com fundamento nos dados das tabelas, notou-se que o algoritmo Dijkstra possuiu maior estabilidade com relação a CPU, porém, teve um consumo médio ligeiramente maior quando comparado ao A*. Com relação a memória RAM, o A* apresentou um grau de constância maior, entretanto, dispôs de maior consumo médio do recurso. Com relação ao tráfego de dados na interface de rede do contêiner, o A* contou com maior estabilidade

em ambos os casos, mas o Dijkstra apresentou consumo médio ligeiramente melhor nos dois cenários. Então, ao considerar a estabilidade como fator preponderante no contexto de névoa, o algoritmo A* é mais recomendado para a topologia de 4000 nós.

5.4.4 Consumo médio no processamento de 5000 semáforos inteligentes em névoa

A Tabela 25 contém os dados referentes ao consumo médio de recursos computacionais do algoritmo Dijkstra na topologia de 5000 nós.

Tabela 25 – Consumo médio por semente na topologia 5000 (Dijkstra/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
0	9,90	42,209	65,95	4103,16	126,00
4	9,94	27,321	42,69	3084,29	121,80
6	9,91	28,250	44,14	3250,80	122,77
9	9,89	28,385	44,35	3123,86	126,00
11	9,89	28,354	44,30	3252,90	126,00
12	9,92	28,343	44,29	3264,00	126,00
15	9,91	27,803	43,44	3127,75	126,00
25	9,94	28,243	44,13	3191,15	126,00
27	9,92	28,388	44,36	3173,54	126,00
30	9,94	28,416	44,40	3223,04	126,00

Fonte: Elaborado pelo autor.

As informações referentes ao consumo do algoritmo A* são detalhadas na Tabela 26.

Tabela 26 – Consumo médio por semente na topologia 5000 (A*/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
0	9,92	27,505	42,97	3115,75	122,18
4	9,91	27,463	42,91	3055,21	121,80
6	9,90	28,091	43,89	3109,31	122,77
9	9,91	28,336	44,28	3176,96	126,00
11	9,93	28,261	44,16	3234,99	126,00
12	9,88	28,480	44,50	3252,91	126,00
15	9,89	27,789	43,42	3151,64	126,00
25	9,87	28,223	44,10	3171,61	126,00
27	9,92	27,806	43,45	3289,06	122,50
30	9,87	28,138	43,96	3196,46	122,77

Fonte: Elaborado pelo autor.

A partir das tabelas, foram calculados os valores das médias e dos desvios padrão dos percentuais de consumo de CPU e memória RAM, bem como do envio e recebimento de dados. Os dados são mostrados na Tabela 27.

Tabela 27 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 5000 - Névoa

Métrica	**Dijkstra**	**A***
Média - CPU (%)	9,916	9,900
Desvio padrão - CPU (%)	0,020	0,022
Média - Memória (%)	46,205	43,764
Desvio padrão - Memória (%)	6,959	0,549
Média - Net Input (B)	3279,449	3175,390
Desvio padrão - Net Input (B)	295,854	71,394
Média - Net Output (B)	125,257	124,202
Desvio padrão - Net Output (B)	1,583	1,915

Fonte: Elaborado pelo autor.

Além disso, foram calculados os coeficientes de variação das métricas, como mostra a Tabela 28.

Tabela 28 – Coeficientes de variação das métricas de desempenho na topologia 5000 - Névoa

Métrica	**Dijkstra**	**A***
CPU (%)	0,00202	0,00222
Memória (%)	0,15061	0,01254
Net Input (B)	0,09021	0,02248
Net Output (B)	0,01264	0,01542

Fonte: Elaborado pelo autor.

De acordo com os dados das tabelas, percebeu-se que os algoritmos Dijkstra e A* expuseram um desempenho semelhante com relação ao consumo médio de recursos e estabilidade com relação a CPU. No que diz respeito a memória RAM e a entrada de dados pela interface de rede, o A* apresentou melhor desempenho no consumo médio e na constância durante a sua execução. Já o Dijkstra, apresentou maior estabilidade no tráfego de saída de dados. Portanto, considerando tais fatores, a melhor escolha nesse cenário é o algoritmo A* devido ao seu consumo equilibrado de recursos no contexto de névoa.

5.4.5 Consumo médio no processamento de 6000 semáforos inteligentes em névoa

As informações de consumo médio de recursos computacionais pelo algoritmo Dijkstra na topologia de 6000 nós são demonstradas na Tabela 29.

Tabela 29 – Consumo médio por semente na topologia 6000 (Dijkstra/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
6	9,90	29,907	46,73	3432,81	126,00
8	9,88	28,660	44,78	3176,45	122,77
11	9,89	29,117	45,50	3253,99	123,37
20	9,92	30,180	47,16	3418,54	126,00
22	9,87	28,482	44,51	3140,78	122,50
23	9,88	29,313	45,80	3170,46	126,00
24	9,89	29,692	46,40	3330,89	123,79
29	9,90	29,411	45,96	3274,01	123,53
33	9,92	29,888	46,70	3406,91	126,00
37	9,91	29,661	46,35	3381,01	123,79

Fonte: Elaborado pelo autor.

Com relação ao algoritmo A*, os dados são expostos na Tabela 30.

Tabela 30 – Consumo médio por semente na topologia 6000 (A*/Névoa)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
6	9,91	29,458	46,03	3359,32	123,53
8	9,91	29,162	45,57	3162,45	126,00
11	9,96	29,110	45,48	3249,52	123,38
20	9,84	30,214	47,21	3461,66	126,00
22	9,90	28,409	44,39	3074,02	122,50
23	9,94	28,877	45,12	3153,46	123,00
24	9,90	29,844	46,63	3403,99	123,79
29	9,91	29,320	45,81	3340,65	123,53
33	9,91	29,413	45,95	3252,93	123,53
37	9,91	30,082	47,00	3422,44	126,00

Fonte: Elaborado pelo autor.

Com o uso das informações oriundas das tabelas, foram obtidos os valores das médias e dos desvios padrão dos percentuais de consumo de CPU e memória RAM, assim como do envio e recebimento de dados. As informações são expostas na Tabela 31.

Tabela 31 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 6000 - Névoa

Métrica	**Dijkstra**	**A***
Média - CPU (%)	9,896	9,909
Desvio padrão - CPU (%)	0,017	0,031
Média - Memória (%)	45,989	45,919
Desvio padrão - Memória (%)	0,860	0,860
Média - Net Input (B)	3298,585	3288,044
Desvio padrão - Net Input (B)	110,958	129,917
Média - Net Output (B)	124,375	124,126
Desvio padrão - Net Output (B)	1,455	1,341

Fonte: Elaborado pelo autor.

Também, foram determinados os coeficientes de variação das métricas conforme a Tabela 32.

Tabela 32 – Coeficientes de variação das métricas de desempenho na topologia 6000 - Névoa

Métrica	**Dijkstra**	**A***
CPU (%)	0,00172	0,00313
Memória (%)	0,01870	0,01873
Net Input (B)	0,03364	0,03951
Net Output (B)	0,01170	0,01080

Fonte: Elaborado pelo autor.

Segundo os dados das tabelas, verificou-se que os algoritmos apresentaram desempenho médio muito semelhante durante as suas execuções. Entretanto, o algoritmo Dijkstra teve valores menores de coeficiente de variabilidade na maioria das métricas, exceto no tráfego de dados de saída na interface de rede do contêiner, portanto, mostrou maior estabilidade. Nesse caso, tornou-se preferido pois no contexto de névoa é importante a consistência do desempenho devido a limitação de recursos.

5.5 RECURSOS COMPUTACIONAIS CONSUMIDOS A PARTIR DA EXECUÇÃO DOS ALGORITMOS DE ROTEAMENTO NO CONTEXTO DE NUVEM POR TOPOLOGIA

Para a simulação no ambiente de nuvem, não foram estabelecidas limitações na configuração do contêiner com relação a quantidade de CPU disponível para uso, a quantia de memória RAM acessível e as taxas de leitura e escrita. Tal escolha foi realizada devido ao fato do ambiente de névoa ser usualmente empregado em dispositivos com alto poder computacional.

Em cada topologia, o critério de escolha entre os dois algoritmos se deu pela maior capacidade de economia de recursos já que, no contexto de nuvem, os provedores realizam a cobrança a partir dos recursos computacionais que são demandados. Então, nesse caso, prevaleceu o objetivo de deter uma maior economia de custos sem comprometer a performance, já que os recursos podem ser alocados conforme a demanda. Dessa forma, a análise foi centrada nas médias das métricas, dado que este ambiente prioriza a otimização do uso, e não foram calculados os coeficientes de variação.

5.5.1 Consumo médio no processamento de 2000 semáforos inteligentes em nuvem

A Tabela 33 mostra o consumo médio de recursos computacionais, de cada semente de simulação, a partir da execução do algoritmo de Dijkstra.

Tabela 33 – Consumo médio por semente na topologia 2000 (Dijkstra/Nuvem)

Semente	CPU (%)	Mem Usage (MiB)	Mem (%)	Net Input (B)	Net Output (B)
0	32,37	26,380	0,3368	526,00	84,00
2	32,79	26,430	0,3375	1576,96	84,00
9	25,43	25,980	0,3317	526,00	84,00
12	24,59	26,060	0,3328	979,00	84,00
17	11,27	25,930	0,3311	526,00	84,00
20	33,89	26,460	0,3379	979,00	84,00
22	20,94	26,350	0,3365	1320,96	84,00
23	30,30	26,400	0,3371	526,00	84,00
26	26,49	26,220	0,3348	1320,96	84,00
27	25,81	26,160	0,3340	1320,96	84,00

Fonte: Elaborado pelo autor.

A Tabela 34 expõe o consumo médio de recursos computacionais a partir da execução do algoritmo A*.

Tabela 34 – Consumo médio por semente na topologia 2000 (A*/Nuvem)

Semente	CPU (%)	Mem Usage (MiB)	Mem (%)	Net Input (B)	Net Output (B)
0	16,37	26,210	0,3347	1556,48	84,00
2	24,93	26,300	0,3358	526,00	84,00
9	15,40	25,910	0,3308	526,00	84,00
12	41,23	25,920	0,3310	526,00	84,00
17	13,21	25,910	0,3308	1239,04	84,00
20	34,03	26,230	0,3349	526,00	84,00
22	27,18	26,290	0,3357	1341,44	84,00
23	7,47	26,290	0,3357	1443,84	84,00
26	8,43	26,150	0,3339	526,00	84,00
27	6,94	26,050	0,3326	1024,00	84,00

Fonte: Elaborado pelo autor.

A partir dos dados das tabelas, foram calculadas as médias do percentual de CPU e de memória RAM consumidos e do envio e recebimento de dados, além do desvio padrão para tais métricas. Os dados são demonstrados na Tabela 35.

Tabela 35 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 2000 - Nuvem

Métrica	**Dijkstra**	**A***
Média - CPU (%)	26,388	19,519
Desvio padrão - CPU (%)	6,758	11,846
Média - Memória (%)	0,335	0,334
Desvio padrão - Memória (%)	0,003	0,002
Média - Net Input (B)	960,184	923,480
Desvio padrão - Net Input (B)	411,726	440,428
Média - Net Output (B)	84,000	84,000
Desvio padrão - Net Output (B)	0,000	0,000

Fonte: Elaborado pelo autor.

Com fundamento nos dados da tabela, foi possível inferir que o algoritmo A* é o mais recomendado nesse contexto devido o seu menor uso de recursos, considerando

a economia de capacidade computacional como critério. Ele apresentou menor consumo médio em quase todas as métricas, exceto pelo tráfego de saída de dados em que houve um empate entre os dois algoritmos.

5.5.2 Consumo médio no processamento de 3000 semáforos inteligentes em nuvem

A Tabela 36 evidencia o consumo médio de recursos computacionais pelo algoritmo Dijkstra em sua execução.

Tabela 36 – Consumo médio por semente na topologia 3000 (Dijkstra/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
3	69,78	27,230	0,3477	526,00	84,00
4	55,75	27,180	0,3471	526,00	84,00
6	63,28	27,860	0,3557	526,00	84,00
11	76,78	27,580	0,3522	1239,04	84,00
12	62,73	27,540	0,3517	776,00	84,00
17	81,50	27,800	0,3590	526,00	84,00
18	37,33	27,080	0,3458	776,00	84,00
29	55,48	27,840	0,3555	1464,32	84,00
30	75,95	27,570	0,3520	1341,44	84,00
33	99,74	27,590	0,3523	526,00	84,00

Fonte: Elaborado pelo autor.

Referente ao algoritmo A*, a Tabela 37 apresenta dados referente ao consumo de recursos durante a sua execução.

Tabela 37 – Consumo médio por semente na topologia 3000 (A*/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
3	49,17	27,120	0,3463	526,00	84,00
4	36,59	27,100	0,3460	526,00	84,00
6	91,69	27,650	0,3531	526,00	84,00
11	53,77	27,460	0,3506	1341,44	84,00
12	54,38	27,480	0,3509	776,00	84,00
17	28,96	27,600	0,3524	1556,48	126,00
18	52,03	27,000	0,3448	526,00	84,00
29	80,68	27,650	0,3531	526,00	84,00
30	86,42	27,560	0,3519	526,00	84,00
33	64,68	27,590	0,3523	866,00	84,00

Fonte: Elaborado pelo autor.

Tendo como ponto de partida as informações das tabelas, foram determinadas as médias do percentual de CPU e de memória RAM consumidos e do envio e recebimento de dados, além do desvio padrão para tais métricas. Os dados são apresentados na Tabela 38.

Tabela 38 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 3000 - Nuvem

Métrica	**Dijkstra**	**A***
Média - CPU (%)	67,832	59,837
Desvio padrão - CPU (%)	17,096	20,855
Média - Memória (%)	0,352	0,350
Desvio padrão - Memória (%)	0,004	0,003
Média - Net Input (B)	822,680	769,592
Desvio padrão - Net Input (B)	379,857	381,745
Média - Net Output (B)	84,000	88,200
Desvio padrão - Net Output (B)	0,000	13,282

Fonte: Elaborado pelo autor.

Tomando como referência os dados da tabela, notou-se que o A* apresentou melhor economia de recursos em termos de CPU, memória RAM e tráfego de entrada de dados. O Dijkstra, por sua vez, deteve melhor consumo médio com relação a saída de dados. Logo, o algoritmo mais adequado para o contexto de nuvem com a topologia de 3000 nós é o A*, devido a sua menor utilização de recursos computacionais em quase todas as métricas.

5.5.3 Consumo médio no processamento de 4000 semáforos inteligentes em nuvem

Os dados referentes ao consumo médio de recursos computacionais pelo algoritmo de Dijkstra durante a sua execução na topologia de 4000 nós são vistos na Tabela 39.

Tabela 39 – Consumo médio por semente na topologia 4000 (Dijkstra/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
2	74,54	29,455	0,3761	1950,72	105,00
3	54,04	27,980	0,3573	1720,32	105,00
5	61,43	28,670	0,3661	1651,74	105,00
6	68,70	28,810	0,3679	1781,76	105,00
9	57,06	28,735	0,3669	1832,96	105,00
10	75,32	28,265	0,3609	984,92	105,00
19	61,08	28,810	0,3679	1447,84	105,00
20	72,65	29,455	0,3761	2068,48	105,00
22	72,53	29,185	0,3727	1950,72	105,00
23	93,56	28,485	0,3637	882,52	105,00

Fonte: Elaborado pelo autor.

As informações referentes aos recursos computacionais que foram consumidos durante a execução do algoritmo A* são expostas na Tabela 40.

Tabela 40 – Consumo médio por semente na topologia 4000 (A*/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
2	76,58	29,160	0,3723	2058,24	105,00
3	62,23	27,840	0,3555	1832,96	105,00
5	68,40	28,615	0,3654	1894,40	105,00
6	88,38	28,220	0,3603	1215,32	105,00
9	96,96	28,410	0,3628	776,00	84,00
10	58,74	28,740	0,3670	1781,76	105,00
19	74,88	28,365	0,3622	1832,96	105,00
20	92,73	28,595	0,3651	1245,32	105,00
22	84,78	28,945	0,3696	1447,84	105,00
23	82,51	28,875	0,3687	1651,74	105,00

Fonte: Elaborado pelo autor.

Com base nos dados das tabelas, as médias do percentual de CPU e de memória RAM consumidos e do envio e recebimento de dados foram determinadas, além dos valores de desvio padrão. As informações são demonstradas na Tabela 41.

Tabela 41 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 4000 - Nuvem

Métrica	**Dijkstra**	**A***
Média - CPU (%)	69,091	78,619
Desvio padrão - CPU (%)	11,480	12,781
Média - Memória (%)	0,368	0,365
Desvio padrão - Memória (%)	0,006	0,005
Média - Net Input (B)	1627,198	1573,654
Desvio padrão - Net Input (B)	405,353	395,794
Média - Net Output (B)	105,000	102,900
Desvio padrão - Net Output (B)	0,000	6,641

Fonte: Elaborado pelo autor.

Considerando a eficiência no uso de recursos computacionais como critério de avaliação no cenário de computação em nuvem, na topologia de 4000 nós o A* apresentou maior consumo médio de CPU e um grau maior de instabilidade nessa métrica, contudo, tal fato foi compensado considerando as demais métricas dado que o algoritmo demonstrou menor consumo médio de memória RAM e menor tráfego de dados de entrada e de saída na interface de rede do contêiner. Portanto, a partir da economia de recursos globais, o A* oferece um equilíbrio de consumo médio ligeiramente maior quando comparado ao Dijkstra.

5.5.4 Consumo médio no processamento de 5000 semáforos inteligentes em nuvem

A Tabela 42 contém os dados referentes ao consumo médio de recursos computacionais do algoritmo Dijkstra na topologia de 5000 nós.

Tabela 42 – Consumo médio por semente na topologia 5000 (Dijkstra/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
0	81,72	29,120	0,3718	1390,48	112,00
4	83,76	29,065	0,3711	1832,96	105,00
6	85,27	29,900	0,3818	2037,76	112,00
9	87,97	29,307	0,3742	1951,08	112,00
11	82,15	29,440	0,3759	2112,85	112,00
12	81,78	29,653	0,3786	2112,85	112,00
15	80,04	28,955	0,3697	1215,32	105,00
25	79,87	29,137	0,3720	1815,15	112,00
27	88,42	29,207	0,3729	1585,04	112,00
30	91,24	29,720	0,3795	2065,07	112,00

Fonte: Elaborado pelo autor.

As informações referentes ao consumo do algoritmo A* são detalhadas na Tabela 43.

Tabela 43 – Consumo médio por semente na topologia 5000 (A*/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
0	73,46	29,413	0,3756	1619,17	112,00
4	81,47	28,915	0,3692	1832,96	105,00
6	84,66	29,660	0,3787	1815,15	112,00
9	85,53	29,397	0,3754	1585,04	112,00
11	80,54	29,410	0,3755	1585,04	112,00
12	82,11	29,493	0,3766	1731,81	112,00
15	97,52	28,585	0,3650	1832,96	105,00
25	81,73	29,033	0,3707	2065,07	112,00
27	79,21	29,510	0,3768	2037,76	112,00
30	90,76	29,513	0,3768	1438,27	112,00

Fonte: Elaborado pelo autor.

A partir das tabelas, foram calculados os valores das médias e dos desvios padrão dos percentuais de consumo de CPU e memória RAM, bem como do envio e recebimento de dados. Os dados são indicados na Tabela 44.

Tabela 44 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 5000 - Nuvem

Métrica	**Dijkstra**	**A***
Média - CPU (%)	84,222	83,699
Desvio padrão - CPU (%)	3,876	6,595
Média - Memória (%)	0,375	0,374
Desvio padrão - Memória (%)	0,004	0,004
Média - Net Input (B)	1811,856	1754,323
Desvio padrão - Net Input (B)	316,411	202,469
Média - Net Output (B)	110,600	110,600
Desvio padrão - Net Output (B)	2,951	2,951

Fonte: Elaborado pelo autor.

Segundo os dados mostrados na tabela, o A* teve um leve consumo médio menor de CPU e exigiu menos tráfego de entrada de dados, o que o torna ligeiramente mais

eficiente considerando a economia de recursos. Em relação a memória e o tráfego de saída de dados, ambos os algoritmos mostraram praticamente o mesmo desempenho, tanto em uso médio quanto em variação. Portanto, considerando o contexto de execução, o A* é mais adequado para a topologia de 5000 nós pelo fato de apresentar menor consumo de recursos em mais métricas.

5.5.5 Consumo médio no processamento de 6000 semáforos inteligentes em nuvem

As informações do consumo médio de recursos computacionais pelo algoritmo Dijkstra na topologia de 6000 nós são expostas na Tabela 45.

Tabela 45 – Consumo médio por semente na topologia 6000 (Dijkstra/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
6	95,31	30,750	0,3926	2216,96	115,50
8	85,75	30,310	0,3870	1475,81	112,00
11	86,13	30,488	0,3893	1767,34	115,50
20	97,31	31,235	0,3988	2060,80	115,50
22	74,91	30,137	0,3848	2143,57	112,00
23	87,46	30,380	0,3879	2181,12	112,00
24	85,54	31,220	0,3986	1956,59	117,60
29	90,02	30,898	0,3945	2383,36	115,50
33	90,00	30,890	0,3944	2324,48	115,50
37	94,81	31,010	0,3960	2301,44	115,50

Fonte: Elaborado pelo autor.

Com relação ao algoritmo A*, os dados são indicados na Tabela 46.

Tabela 46 – Consumo médio por semente na topologia 6000 (A*/Nuvem)

Semente	**CPU (%)**	**Mem Usage (MiB)**	**Mem (%)**	**Net Input (B)**	**Net Output (B)**
6	84,64	31,038	0,3963	2273,28	115,50
8	91,12	30,090	0,3842	2071,89	112,00
11	83,45	30,623	0,3910	2216,96	115,50
20	84,87	31,140	0,3976	1946,35	117,60
22	75,82	30,040	0,3836	1731,81	112,00
23	97,96	30,130	0,3847	2065,07	112,00
24	84,69	31,138	0,3976	1923,82	117,60
29	83,05	31,048	0,3964	2268,16	115,50
33	91,53	30,848	0,3939	1879,98	115,50
37	99,34	30,838	0,3938	2216,96	115,50

Fonte: Elaborado pelo autor.

Com o uso das informações oriundas das tabelas, foram obtidos os valores das médias e dos desvios padrão dos percentuais de consumo de CPU e memória RAM, assim como do envio e recebimento de dados. As informações são mostradas na Tabela 47.

Tabela 47 – Valores de média e desvio padrão dos indicadores de desempenho por algoritmo na topologia 6000 - Nuvem

Métrica	**Dijkstra**	**A***
Média - CPU (%)	88,724	87,647
Desvio padrão - CPU (%)	6,460	7,260
Média - Memória (%)	0,392	0,392
Desvio padrão - Memória (%)	0,005	0,006
Média - Net Input (B)	2081,147	2059,428
Desvio padrão - Net Input (B)	281,315	185,542
Média - Net Output (B)	114,660	114,870
Desvio padrão - Net Output (B)	1,947	2,150

Fonte: Elaborado pelo autor.

A partir dos dados da tabela, observou-se que ambos os algoritmos apresentaram um desempenho semelhante considerando o uso médio de recursos computacionais. O algoritmo A* teve pequenas vantagens em eficiência, com menor utilização de CPU e tráfego de entrada. Com relação a memória RAM, houve um empate entre os dois algoritmos e o Dijkstra apresentou melhor consumo médio na saída de dados. Logo, considerando a eficiência global, o algoritmo A* é mais recomendado para a topologia de 6000 nós.

5.6 ANÁLISE DO CONSUMO MÉDIO DE RECURSOS POR TOPOLOGIA NOS AMBIENTES DE NÉVOA E NUVEM

Com base nos dados experimentais obtidos, a seguir é exposta a análise do consumo médio dos recursos computacionais a partir da execução dos algoritmos de roteamento, de acordo com cada topologia, nos ambientes de computação em névoa e em nuvem.

5.6.1 Ambiente de névoa

A Tabela 48 mostra o consumo médio de recursos, por topologia, no ambiente de névoa. As métricas analisadas foram o percentual de CPU e de memória consumidos durante a execução do Dijkstra e do A* e a quantidade de dados enviados e recebidos na interface de rede do contêiner.

Tabela 48 – Consumo médio de recursos por topologia no ambiente de névoa

Topologia	**CPU (%) Dijk.**	**CPU (%) A***	**MEM (%) Dijk.**	**MEM (%) A***	**NET I (B) Dijk.**	**NET I (B) A***	**NET O (B) Dijk.**	**NET O (B) A***
2000	9,909	9,892	39,668	38,807	2209,206	2193,813	114,800	119,000
3000	9,887	9,907	40,332	40,440	2672,331	2645,291	121,380	121,170
4000	9,896	9,883	42,440	43,230	3028,815	3306,576	123,708	125,533
5000	9,916	9,900	46,205	43,764	3279,449	3175,390	125,257	124,202
6000	9,896	9,909	45,989	45,919	3298,585	3288,044	124,375	124,126

Fonte: Elaborado pelo autor.

A Figura 23 demonstra o percentual médio consumido da CPU durante a execução dos algoritmos Dijkstra e A* em cada topologia. A escala vertical do gráfico foi manipulada

para destacar visualmente pequenas diferenças que seriam pouco perceptíveis em uma escala completa.

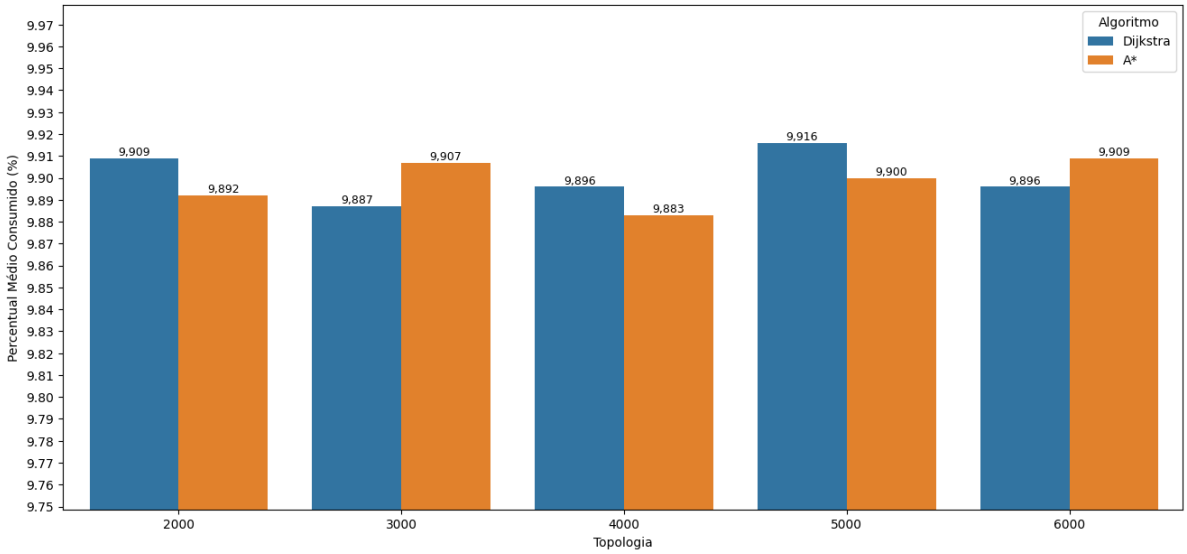

Figura 23 – Consumo médio de CPU por topologia - Névoa

Fonte: Elaborado pelo autor.

Então, notou-se que o algoritmo A* apresentou menor consumo de CPU, em média, nas topologias de 2000, 4000 e 5000 nós, enquanto que o Dijkstra teve tal feito nas topologias de 3000 e 6000 nós.

Com relação ao percentual médio de consumo de memória RAM, o gráfico da Figura 24 expõe tais dados para as cinco topologias utilizadas.

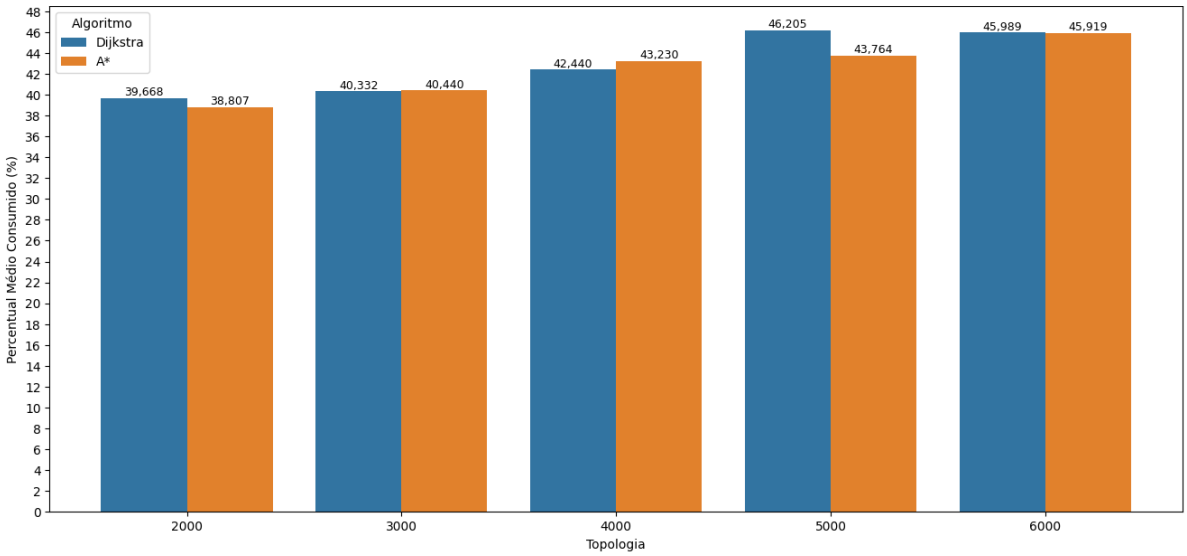

Figura 24 – Consumo médio de memória RAM por topologia - Névoa

Fonte: Elaborado pelo autor.

A partir disso, percebeu-se que o A* teve menor percentual médio de consumo de RAM para as topologias de 2000, 5000 e 6000 nós. Em contrapartida, o Dijkstra teve menor nas topologias de 3000 e 4000 nós.

A Figura 25 apresenta a quantidade média de dados recebidos pela interface de rede no contêiner em cada topologia.

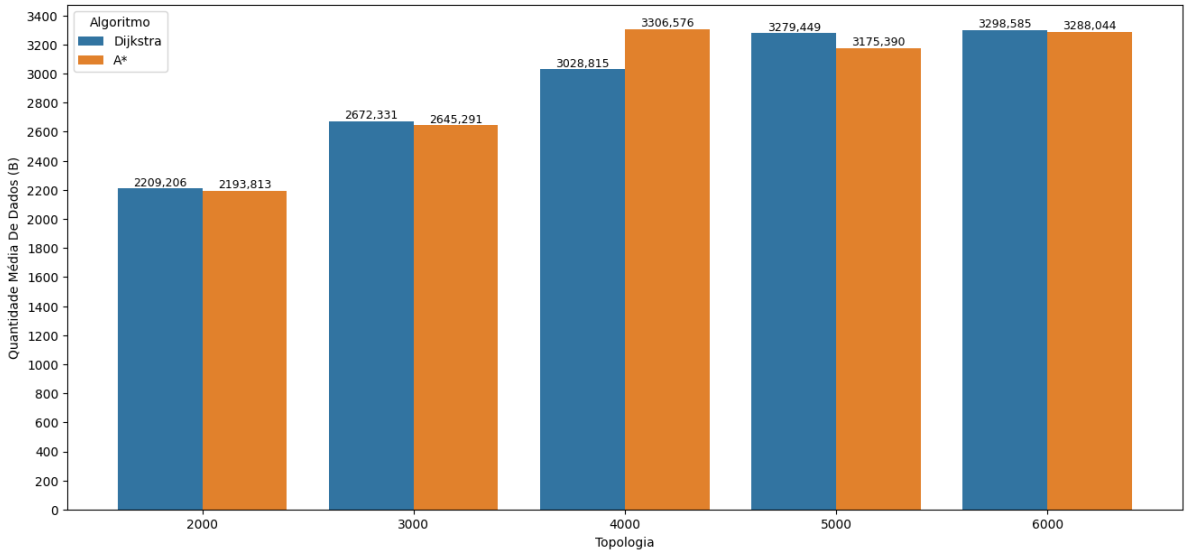

Figura 25 – Dados recebidos pela interface de rede do contêiner por topologia - Névoa

Fonte: Elaborado pelo autor.

Com base nos dados expostos, o Dijkstra recebeu mais dados, em média, nas topologias de 2000, 3000, 5000 e 6000 nós. Por outro lado, o A* recebeu mais informações apenas na topologia de 4000 nós.

Por fim, a Figura 26 demonstra a quantidade média de dados enviados pela interface de rede do contêiner em cada topologia.

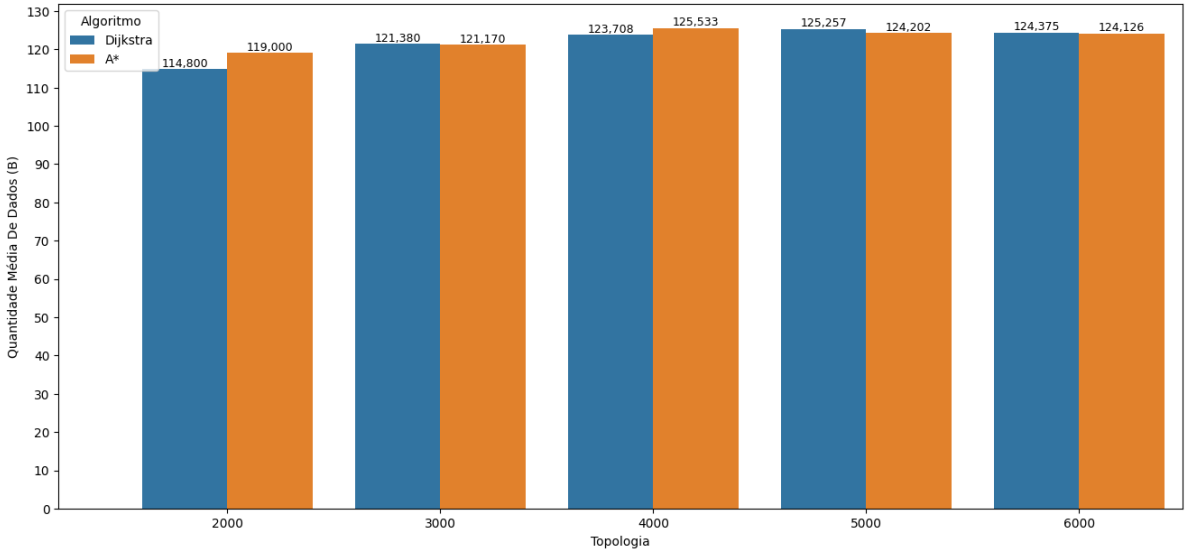

Figura 26 – Dados enviados pela interface de rede do contêiner por topologia - Névoa

Fonte: Elaborado pelo autor.

A partir dessas informações, viu-se que o Dijkstra enviou mais dados, em média, nas topologias de 3000, 5000 e 6000 nós. Já o A*, teve o mesmo comportamento nas topologias de 2000 e 4000 nós.

5.6.2 Ambiente de nuvem

A Tabela 49 evidencia o consumo médio de recursos, por topologia, no ambiente de nuvem. Foram avaliados o percentual de CPU e de memória consumidos durante a execução do Dijkstra e do A* e a quantidade de dados enviados e recebidos na interface de rede do contêiner.

Tabela 49 – Consumo médio de recursos por topologia no ambiente de nuvem

Topologia	**CPU (%) Dijk.**	**CPU (%) A***	**MEM (%) Dijk.**	**MEM (%) A***	**NET I (B) Dijk.**	**NET I (B) A***	**NET O (B) Dijk.**	**NET O (B) A***
2000	26,388	19,519	0,335	0,333	960,184	923,480	84,000	84,000
3000	67,832	59,837	0,352	0,350	822,680	769,592	84,000	88,200
4000	69,091	78,619	0,368	0,365	1627,198	1573,654	105,000	102,900
5000	84,222	83,699	0,375	0,374	1811,856	1754,323	110,600	110,600
6000	88,724	87,647	0,392	0,392	2081,147	2059,428	114,660	114,870

Fonte: Elaborado pelo autor.

A Figura 27 indica o valor médio percentual consumido da CPU durante a execução dos algoritmos Dijkstra e A*.

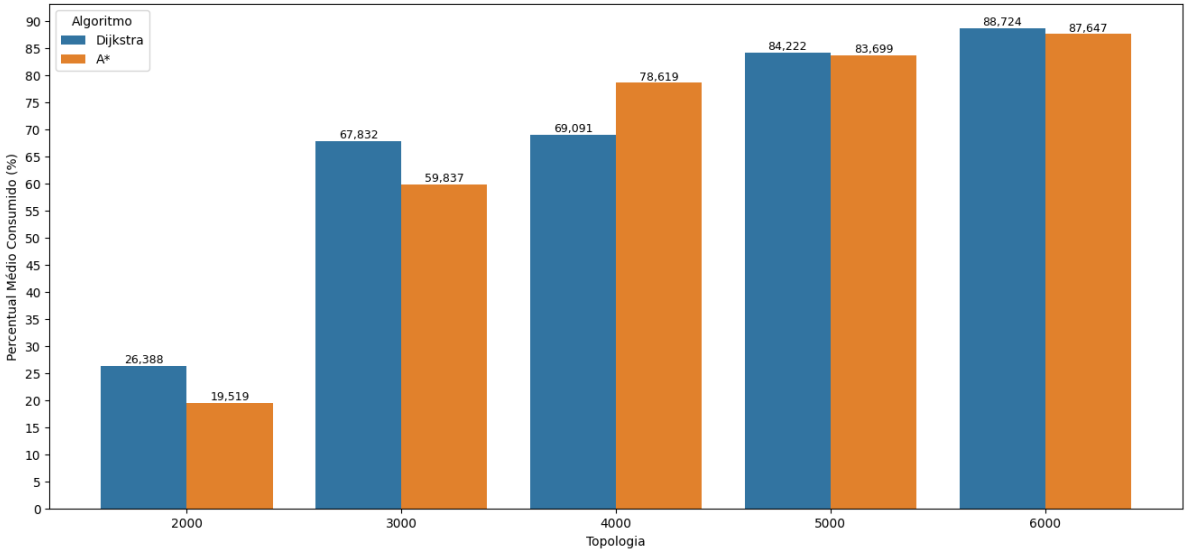

Figura 27 – Consumo médio de CPU por topologia - Nuvem

Fonte: Elaborado pelo autor.

Em decorrência disso, verificou-se que o A* apresentou menor consumo médio de CPU nas topologias de 2000, 3000, 5000 e 6000 nós, enquanto o Dijkstra atingiu tal feito na topologia de 4000 nós.

A Figura 28 expõe o percentual médio de consumo de memória RAM para as cinco topologias utilizadas.

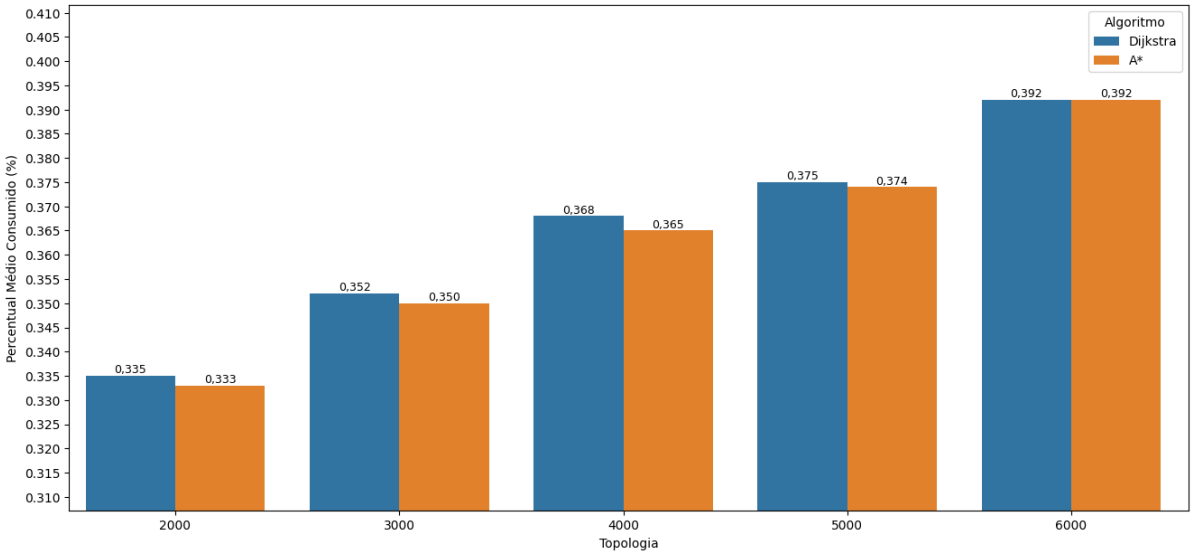

Figura 28 – Consumo médio de memória RAM por topologia - Nuvem

Fonte: Elaborado pelo autor.

Nessa situação, o A* apresentou menor percentual médio de consumo de memória em todas as topologias, exceto pela de 6000 nós, em que se teve os mesmos percentuais de consumo para ambos os algoritmos.

A Figura 29 mostra a quantidade média de dados recebidos pela interface de rede do contêiner.

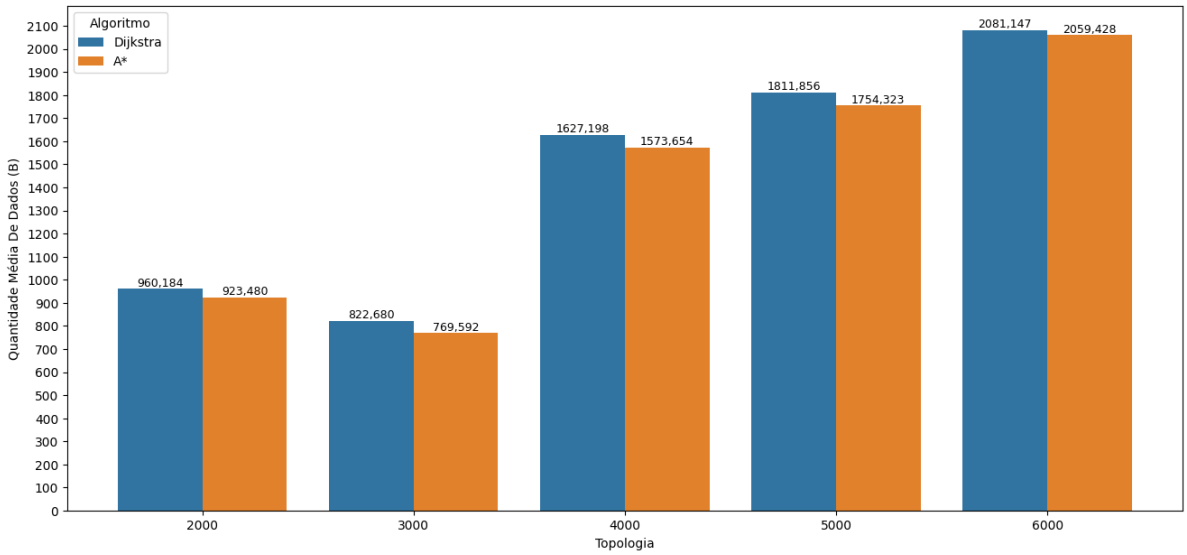

Figura 29 – Dados recebidos pela interface de rede do contêiner por topologia - Nuvem

Fonte: Elaborado pelo autor.

Com base nas informações, constatou-se que o algoritmo Dijkstra recebe mais dados em todas as topologias.

Por fim, a Figura 30 indica a quantidade média de dados enviados pela interface de rede do contêiner.

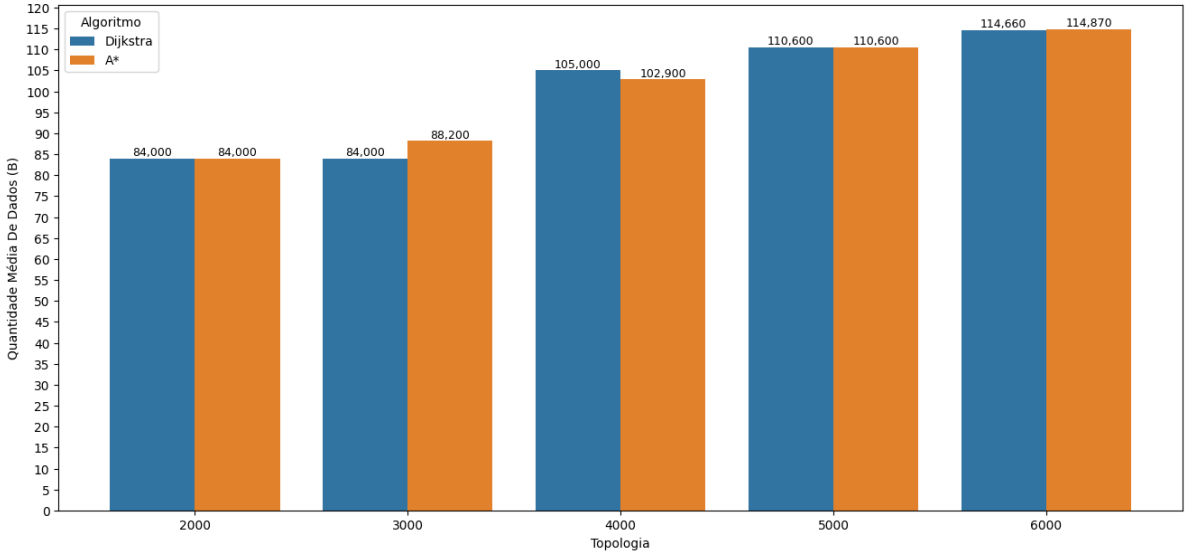

Figura 30 – Dados enviados pela interface de rede do contêiner por topologia - Nuvem

Fonte: Elaborado pelo autor.

Considerando isso, observou-se a mesma quantidade média de dados enviados por ambos os algoritmos nas topologias de 2000 e 5000 nós. Dijkstra teve média superior na topologia de 4000 nós e A* nas topologias de 3000 e 6000 nós.

6 CONCLUSÃO

Esta pesquisa teve como intuito principal analisar comparativamente o desempenho e o uso dos recursos computacionais de ambientes simulados de computação em névoa e em nuvem os quais realizam o gerenciamento de tráfego com semáforos inteligentes. Para isso, foram utilizados os algoritmos Dijkstra e A* como cargas de trabalho, a fim de definir o menor caminho em uma rota predefinida a partir de topologias que representam infraestruturas urbanas ideais de cidades inteligentes. Com isso, verificou-se para cada contexto quais dos algoritmos são mais apropriados considerando os dados obtidos experimentalmente e os critérios definidos.

Com relação a rapidez na execução dos algoritmos, o algoritmo A* apresentou melhor tempo de execução médio em 80% das topologias utilizadas, demonstrando a sua agilidade no conjunto de cenários produzidos.

Além disso, obtiveram-se os dados referentes ao consumo médio com base na topologia e no tipo de ambiente utilizado. Para isso, foram considerados os critérios de estabilidade de desempenho para o ambiente de névoa e a economia no uso de recursos computacionais para o ambiente de nuvem.

Nesse sentido, no ambiente de névoa notou-se que o algoritmo A* apresentou maior estabilidade em 80% das topologias, sendo o mais adequado para uso no contexto de limitações de hardware. Por outro lado, percebeu-se que, no ambiente de nuvem, o algoritmo A* teve melhor economia no uso médio de recursos em todas as topologias, o que impacta diretamente na redução de custos devido ao impedimento de alocação desnecessária de infraestrutura computacional virtual.

A aplicação da temática exposta contribui diretamente para o aperfeiçoamento de tecnologias aplicadas no trânsito de cidades inteligentes pois sugere o uso de algoritmos de roteamento consolidados da ciência da computação e verifica o comportamento em dois cenários distintos, névoa e nuvem, os quais podem ser utilizados na infraestrutura de projetos viários de *smart cities* como nós computacionais distribuídos de otimização do tráfego, com semáforos inteligentes, para a avaliação de menores caminhos.

A limitação deste trabalho refere-se, principalmente, à carga de trabalho utilizada de forma estática, sem considerar variações dinâmicas no tráfego de veículos ao longo do tempo, o que poderia refletir comportamentos mais realistas no contexto urbano de cidades inteligentes. Além disso, os experimentos foram realizados com um conjunto restrito de topologias e algoritmos de roteamento, o que limita a generalização dos resultados.

Por fim, sugere-se como possibilidade de trabalhos futuros a aplicação dos algoritmos utilizados em infraestruturas reais de hardware, sob o mesmo contexto, a fim de verificar e validar os resultados obtidos nesta pesquisa. Além disso, indica-se a verificação do comportamento da experimentação em topologias maiores e com o uso de outros algoritmos.

REFERÊNCIAS

ABDELSHKOUR, Maher. **Perspectives: IoT, from Cloud to Fog Computing**. 2015. Disponível em: https://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing. Acesso em: 9 jan. 2025.

AL-HAKIMI, Asmaa Mahfoud; SUBBIAH, Ahgalya. Autonomous Traffic Light: Emergency Vehicles Take Control of Traffic Lights to Eliminate Traffic Jam Via Fog Computing. *In*: 2024 International Conference on Intelligent Computing and Next Generation Networks (ICNGN). [*S.l.*]: IEEE, 2024. P. 01–06. Disponível em: https://doi.org/10.1109/ICNGN63705.2024.10871590.

AMAZON AWS. **O que é a computação em nuvem?** 2025. Disponível em: https://aws.amazon.com/pt/what-is-cloud-computing/. Acesso em: 8 abr. 2025.

AMAZON AWS. **Qual é a diferença entre o Kubernetes e o Docker?** 2025. Disponível em: https://aws.amazon.com/pt/compare/the-difference-between-kubernetes-and-docker/. Acesso em: 9 mar. 2025.

BONOMI, Flavio; MILITO, Rodolfo; ZHU, Jiang; ADDEPALLI, Sateesh. Fog Computing and Its Role in the Internet of Things. *In*: PROCEEDINGS of the First Edition of the MCC Workshop on Mobile Cloud Computing. [*S.l.*]: ACM, 2012. (MCC '12), p. 13–16. Disponível em: https://doi.org/10.1145/2342509.2342513.

BOUSKELA, Maurício; CASSEB, Márcia; BASSI, Silvia; LUCA, Cristina De; FACCHINA, Marcelo. **The Road toward Smart Cities: Migrating from Traditional City Management to the Smart City**. [*S.l.*], 2016. Disponível em: http://dx.doi.org/10.18235/0012831.

BRENNAND, Celso A. R. L.; CUNHA, Felipe Domingos da; MAIA, Guilherme; CERQUEIRA, Eduardo; LOUREIRO, Antonio A. F.; VILLAS, Leandro A. FOX: A traffic management system of computer-based vehicles FOG. *In*: 2016 IEEE Symposium on Computers and Communication (ISCC). [*S.l.*]: IEEE, 2016. P. 982–987. Disponível em: https://doi.org/10.1109/ISCC.2016.7543864.

CALDAS, Davi. **Semáforos inteligentes podem proporcionar um trânsito mais fluido e seguro**. 2024. Disponível em: https://jornal.usp.br/?p=826774. Acesso em: 8 jan. 2025.

CARVALHO, Grazielle. **Cidades inteligentes e sustentáveis: Quando teremos aqui no Brasil?** 2021. Disponível em: https://aetec.org.br/cidades-inteligentes-e-sustentaveis-quando-teremos-aqui-no-brasil/. Acesso em: 29 jan. 2025.

CTB. **CTB Digital**. 2017. Disponível em: https://www.ctbdigital.com.br/artigo/art1. Acesso em: 5 jan. 2025.

DELFIN, S.; SIVASANKER, N.P; RAJ, Nishant; ANAND, Ashish. Fog Computing: A New Era of Cloud Computing. *In*: 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC). [*S.l.*]: IEEE, 2019. P. 1106–1111. Disponível em: https://doi.org/10.1109/ICCMC.2019.8819633.

DIVYA, A.; KEERTHANA, K.; KIRUTHIKANJALI, N.; NANDHINI, G.; YUVARAJ, G. Secured Smart Healthcare Monitoring System Based on IOT. **Asian Journal of Applied Science and Technology (AJAST)**, AJAST, v. 1, n. 2, p. 53–56, 2017. Disponível em: https://dx.doi.org/10.2139/ssrn.2941100.

DOCKER. **What is a container?** 2025. Disponível em: https://www.docker.com/resources/what-container/. Acesso em: 8 mar. 2025.

FRANÇA, Marlon Tavares; SANTOS, Audrey Teles dos; JESUS, Igor Dias Costa de; TEIXEIRA, Samuel Molendolff; ARAÚJO, Wagner Azis Garcia de; LIMA PEREIRA, Luan Diego de. A utilização da computação em nuvem como auxílio à escalabilidade e disponibilidade de serviços online. **Brazilian Journal of Production Engineering**, Universidade Federal do Espirito Santo, v. 9, n. 2, p. 79–87, 2023. Disponível em: https://doi.org/10.47456/bjpe.v9i2.40518.

FREITAS, Leandson de Oliveira de. **Computação em nuvem: uma breve revisão bibliográfica**. 2023. Trabalho de Conclusão de Curso (Graduação) – Universidade Federal Rural do Semi-Árido (UFERSA). Disponível em: https://repositorio.ufersa.edu.br/server/api/core/bitstreams/d1f7ad01-4c9c-43fc-b5ed-044288d0d866/content. Acesso em: 19 fev. 2025.

GOOGLE. **O que é computação na nuvem?** 2025. Disponível em: https://cloud.google.com/learn/what-is-cloud-computing?hl=pt-BR. Acesso em: 4 mai. 2025.

GREGÓRIO, Jorge Luís. **Os impactos da Internet das Coisas no setor de transportes**. 2019. Disponível em: https://www.fatecjales.edu.br/index.php/component/content/article/49-noticias/557-08-de-marco-fatec-comemora-o-dia-internacional-da-mulher.html. Acesso em: 27 mai. 2025.

GU, Ke; HU, Jieyu; JIA, Weijia. Adaptive Area-Based Traffic Congestion Control and Management Scheme Based on Fog Computing. **IEEE Transactions on Intelligent Transportation Systems**, Institute of Electrical e Electronics Engineers (IEEE), v. 24, n. 1, p. 1359–1373, 2023. Disponível em: https://doi.org/10.1109/TITS.2022.3183687.

HOSSAN, Sakhawat; NOWER, Naushin. Fog-based dynamic traffic light control system for improving public transport. **Public Transport**, Springer Science e Business Media LLC, v. 12, n. 2, p. 431–454, 2020. Disponível em: https://doi.org/10.1007/s12469-020-00235-z.

IPRI. **Pesquisa Mobilidade Urbana & Trabalho**. 2023. Disponível em: https://static.portaldaindustria.com.br/portaldaindustria/noticias/media/filer_public/91/c6/91c6a394-ee8b-4772-8f19-27fa401a295e/pesquisa_cni_mobilidade_urbana_populacao_-_fase_2.pdf. Acesso em: 7 jan. 2025.

ISLAM, Lamya; HASSAN, Md. Tarek. Performance Evaluation of Vehicle-Centered Traffic Management Using Fog Computing-based Wireless Network. *In*: 2023 26th International Conference on Computer and Information Technology (ICCIT). [*S.l.*]: IEEE, 2023. P. 1–6. Disponível em: https://doi.org/10.1109/ICCIT60459.2023.10441628.

JAMIL, Syed Usman; KHAN, M. Arif. Accident Management System using Fog Computing. *In*: 2019 13th International Conference on Open Source Systems and Technologies (ICOSST). [*S.l.*]: IEEE, 2019. P. 1–6. Disponível em: https://doi.org/10.1109/ICOSST48232.2019.9043923.

JANG, Hung-Chin; LIN, Ting-Kuan. Traffic-Aware Traffic Signal Control Framework Based on SDN and Cloud-Fog Computing. *In*: 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall). [*S.l.*]: IEEE, 2018. P. 1–5. Disponível em: https://doi.org/10.1109/VTCFall.2018.8690602.

KUMAR, Rajesh. **O algoritmo A*: Um guia completo**. 2024. Disponível em: https://www.datacamp.com/pt/tutorial/a-star-algorithm. Acesso em: 28 abr. 2025.

LUO, Yawen; CHEN, Yuhua; WU, Junchao. Energy Efficient Fog Computing with Architecture of Smart Traffic Lights System. *In*: 2021 3rd East Indonesia Conference on Computer and Information Technology (EIConCIT). [*S.l.*]: IEEE, 2021. P. 248–254. Disponível em: https://doi.org/10.1109/EIConCIT50028.2021.9431873.

MAKODE, Kedar. **An Introduction to Docker and Containers for Beginners**. 2024. Disponível em: https://www.freecodecamp.org/news/an-introduction-to-docker-and-containers-for-beginners/. Acesso em: 10 mar. 2025.

MATANGE, Ashwini; TANEJA, Varun; CHAUMAL, Aarya; BUWA, Pallavi; ABRAHAM, Jibi. Resilient Fog-Based Adaptive Traffic Control System. *In*: 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT). [*S.l.*]: IEEE, 2023. P. 1–7. Disponível em: https://doi.org/10.1109/ICCCNT56998.2023.10306713.

MATOS, Florinda; VAIRINHOS, Valter Martins; DAMERI, Renata Paola; DURST, Susanne. Increasing smart city competitiveness and sustainability through managing structural capital. **Journal of Intellectual Capital**, Emerald, v. 18, n. 3, p. 693–707, 2017. Disponível em: https://doi.org/10.1108/JIC-12-2016-0141.

MELL, Peter; GRANCE, Timothy. **The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology**. 2011. Disponível em: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf. Acesso em: 15 fev. 2025.

MOTA, Gustavo de Araújo *et al.* Smart sensors and Internet of Things (IoT) for sustainable environmental and agricultural management. **Caderno Pedagógico**, Brazilian Journals, v. 20, n. 7, p. 2692–2714, 2023. Disponível em: https://doi.org/10.54033/cadpedv20n7-014.

NAVONE, Estefania Cassingena. **Algoritmo de caminho de custo mínimo de Dijkstra - uma introdução detalhada e visual**. Tradução: Cayo Dias. 2022. Disponível em: https://www.freecodecamp.org/portuguese/news/algoritmo-de-caminho-de-custo-minimo-de-dijkstra-uma-introducao-detalhada-e-visual/. Acesso em: 28 abr. 2025.

NETWORKX. **astar_path**. 2024. Disponível em: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.astar.astar_path.html. Acesso em: 28 mar. 2025.

NETWORKX. **dijkstra_path**. 2024. Disponível em: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.dijkstra_path.html. Acesso em: 28 mar. 2025.

NING, Zhaolong; HUANG, Jun; WANG, Xiaojie. Vehicular Fog Computing: Enabling Real-Time Traffic Management for Smart Cities. **IEEE Wireless Communications**, Institute of Electrical e Electronics Engineers (IEEE), v. 26, n. 1, p. 87–93, 2019. Disponível em: https://doi.org/10.1109/MWC.2019.1700441.

ORACLE. **O que é Cloud Computing?** 2020. Disponível em: https://www.oracle.com/br/cloud/what-is-cloud-computing/. Acesso em: 8 abr. 2025.

ORACLE. **O que é IoT?** 2025. Disponível em: https://www.oracle.com/br/internet-of-things/. Acesso em: 8 jan. 2025.

QIN, Haoshu; ZHANG, Huimei. Intelligent traffic light under fog computing platform in data control of real-time traffic flow. **The Journal of Supercomputing**, Springer Science e Business Media LLC, v. 77, n. 5, p. 4461–4483, 2020. Disponível em: https://doi.org/10.1007/s11227-020-03443-3.

SARAIVA, Alexia. **ONU-Habitat: população mundial será 68% urbana até 2050**. 2022. Disponível em: https://brasil.un.org/pt-br/188520-onu-habitat-popula%C3%A7%C3%A3o-mundial-ser%C3%A1-68-urbana-at%C3%A9-2050. Acesso em: 27 jan. 2025.

SHALINI, Chintam Muni Kanaka Sri; ROOPA, Y. Mohana; DEVI, J. Sirisha. Fog Computing for Smart Cities. *In*: 2019 International Conference on Communication and Electronics Systems (ICCES). [*S.l.*]: IEEE, 2019. P. 912–916. Disponível em: https://doi.org/10.1109/ICCES45898.2019.9002050.

SHRIVASTAVA, Ajeet Kumar; GANGADHAR, Pvss; SHUKLA, Ragini. To Implement Cloud Computing by using Agile Methodology in Indian E-Governance. **International Research Journal of Engineering and Technology (IRJET)**, v. 2, n. 1, p. 424–428, 2015. Disponível em: https://www.irjet.net/archives/V2/i1/Irjet-v2i176.pdf.

SINHA, Satyajit. **State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally**. 2024. Disponível em: https://iot-analytics.com/number-connected-iot-devices/. Acesso em: 31 jan. 2025.

SOUSA, Devilson da Rocha; FREITAS, Cinthia Obladen de Almendra. Os desafios e as perspectivas para a regulamentação da Internet das Coisas no Brasil. **International Journal of Digital Law**, International Journal of Digital Law, v. 3, n. 2, p. 51–68, 2022. Disponível em: http://dx.doi.org/10.47975/IJDL.sousa.v.3.n.2.

SOUSA JÚNIOR, Almir Mariano de; PRZEYBILOVICZ, Erico; LACERDA, Hiatiane Cunha de; COSTA, Lauren Cavalheiro da. **Carta brasileira para cidades inteligentes: Versão resumida**. 2021. Disponível em: https://cartacidadesinteligentes.org.br/files/carta_resumida_ptbr.pdf. Acesso em: 28 jan. 2025.

SUSNJARA, Stephanie; SMALLEY, Ian. **O que é computação em nuvem?** 2024. Disponível em: https://www.ibm.com/br-pt/topics/cloud-computing. Acesso em: 4 mai. 2025.

SUSNJARA, Stephanie; SMALLEY, Ian. **O que é Docker?** 2024. Disponível em: https://www.ibm.com/br-pt/think/topics/docker. Acesso em: 8 mar. 2025.