UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Matheus Pereira Leal

Teoria dos Jogos em redes IoV para Otimização da Alocação de Recursos em Computação de Borda

São José do Rio Preto
2023

Matheus Pereira Leal

Teoria dos Jogos em redes IoV para Otimização da Alocação de Recursos em Computação de Borda

Dissertação apresentada para o Exame de Defesa, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação (PPGCC), do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Orientador:

Prof. Dr. Rodolfo Ipolito Meneguette

São José do Rio Preto
2023

L435t

Leal, Matheus Pereira

Teoria dos jogos em redes IoV para otimização da alocação de recursos em computação de borda / Matheus Pereira Leal. -- São José do Rio Preto, 2023

137 f.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientador: Rodolfo Ipolito Meneguette

1. Ciência da computação. 2. Teoria dos jogos. 3. Internet das coisas. 4. Alocação de recursos. 5. Computação em nuvem. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Matheus Pereira Leal

Teoria dos Jogos em redes IoV para Otimização da Alocação de Recursos em Computação de Borda

Dissertação apresentada para o Exame de Defesa, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação (PPGCC), do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Comissão Examinadora

Prof. Dr. Rodolfo Ipolito Meneguetti
USP - Câmpus São Carlos
Orientador

Prof. Dr. Roger Kreutz Immich
UFRN - Câmpus Natal

Prof. Dr. José Remo Ferreira Brega
UNESP - Câmpus Bauru

São José do Rio Preto
17 de Outubro de 2023

Agradecimentos

Primeiramente gostaria de agradecer a Deus por me guiar até essa oportunidade e proporcionar essa experiência incrível!

Em segundo a minha família, em especial a minha mãe Angela Maria de Paula Leal, ao meu pai Angelo Pereira Leal, e ao meu irmão Marcio Arlei Lamera Junior, que sempre acreditaram e apoiaram a realização deste sonho.

Ao meu orientador, Professor Doutor Rodolfo Ipolito Meneguette, que contribuiu em diferentes aspectos para que este projeto se tornasse realidade.

Ao meu amigo Matheus Sanches Quessada, que colaborou diversas vezes, auxiliando na resolução das minhas dúvidas e contribuindo cientificamente nas publicações que foram geradas à partir deste trabalho.

Por último a Universidade Estadual Paulista (UNESP), por fornecer um time de professores extremamente qualificados, uma ótima infraestrutura, além de todo suporte em relação a dúvidas, *softwares* e outros benefícios.

"*Deus nos concede, a cada dia, uma página de vida nova no livro do tempo. Aquilo que colocarmos nela, corre por nossa conta.*"

(Xavier, [s.d.])

Resumo

O advento das tecnologias de redes móveis 5G e 6G contribuíram para o surgimento e a evolução de diversos campos de pesquisas, como é o caso da Internet das Coisas. Em paralelo a isso, outras ramificações emergentes que utilizam esses mesmos conceitos também começaram a ser desenvolvidas, sendo a Internet dos Veículos um dos mais notáveis exemplos desse contexto. Entretanto, diversos desafios ainda devem ser superados, principalmente para atender as solicitações dos aplicativos que são sensíveis a atrasos em ambientes de alta mobilidade. Esses fatores são essenciais para alcançar os requisitos esperados de qualidade de serviço e experiência do usuário e, por isso, faz-se necessário recorrer a outras alternativas em frente a tradicional Computação em Nuvem, como é o caso da Computação em Borda. Dessa forma, o objetivo deste trabalho é apresentar um algoritmo intitulado METRO, baseado em conceitos da Teoria dos Jogos, um estudo matemático que busca maximizar os resultados através de modelos de jogos que representam um conjunto de ações, jogadores e regras, que se desenvolvem em um determinado contexto. Isto quer dizer que os jogos são capazes de simular os cenários de disputa de recursos nas infraestruturas de rede, com a finalidade de maximizar a decisão de alocação desses recursos, à partir do descarregamento de computação, realizado pelos veículos na borda da rede. Tudo isso é desenvolvido por meio dos conceitos de jogos cooperativos, como o Equilíbrio de Nash. Por fim, para trazer a maior realidade possível ao processo de simulação, o ambiente considerado está repleto de interações V2X existentes na Internet dos Veículos, que descreve a troca de informações entre veículos e qualquer outro dispositivo inteligente.

Palavras-chave: Alocação de Recursos, Teoria dos Jogos, Computação em Borda Veicular, Computação em Nuvem, Internet dos Veículos.

Abstract

The advent of 5G and 6G mobile network technologies contributed to the emergence and evolution of several research fields, such as the Internet of Things. In parallel to this, other emerging branches that use these same concepts also began to be developed, with the Internet of Vehicles being one of the most notable examples of this context. However, several challenges still need to be overcome, mainly to meet requests from applications that are sensitive to delays in highly mobile environments. These factors are essential to achieve the expected requirements of quality of service and user experience and, therefore, it is necessary to resort to other alternatives in front of traditional Cloud Computing, as is the case of Edge Computing. Thus, the objective of this work is to present an algorithm called METRO, based on concepts of Game Theory, a mathematical study that seeks to maximize results through game models that represent a set of actions, players and rules, which are developed in a certain context. This means that the games are capable of simulating resource dispute scenarios in network infrastructures, with the aim of maximizing the decision to allocate these resources, based on the offloading of computation, carried out by vehicles at the edge of the network. All this is developed through the concepts of cooperative games, such as the Nash Equilibrium. Finally, to bring as much reality as possible to the simulation process, the considered environment is full of existing V2X interactions in the Internet of Vehicles, which describes the exchange of information between vehicles and any other smart device.

Keywords: *Resource Allocation, Game Theory, Vehicular Edge Computing, Cloud Computing, Internet of Vehicles*

Lista de Figuras

Figura 1 Os principais tipos de interações em redes IoV 26
Figura 2 Rede IoV: Seus agentes e interações 28
Figura 3 Arquitetura usual das redes IoV 29
Figura 4 Nova Arquitetura para redes IoV 30
Figura 5 Proximidade dos servidores VEC em relação aos servidores CC 34
Figura 6 Interação entre VEC e CC 37
Figura 7 Comparação da latência em áreas com e sem servidores VEC . 38
Figura 8 Comparação da latência entre servidores VEC e VCC 39
Figura 9 Função de densidade da probabilidade do atraso E2E entre VEC e VCC 40
Figura 10 Matriz de resultados: Dilema do Prisioneiro 44
Figura 11 Fluxo de Decisão com Algoritmo Aleatório 46
Figura 12 Hierarquia dos lobos cinzentos 47
Figura 13 Representação do padrão de caça dos lobos cinzentos 49
Figura 14 Vetor 2D com as possibilidades das próximas posições 49
Figura 15 Efeito "sonar"dos micro-morcegos 50
Figura 16 Protocolo de Revisão Sistemática 54
Figura 17 Condução de Seleção dos Estudos 58
Figura 18 Fluxograma do Algoritmo METRO 70
Figura 19 Cenário de Simulação 75

Figura 20 Mapa de *Grid* 5x5 Utilizado nas Simulações 81
Figura 21 Percentual do Total de Tarefas Alocadas por Execução 89
Figura 22 Percentual de Tarefas Alocadas em VEC por Execução 90
Figura 23 Percentual de Tarefas Alocadas em CC por Execução 91
Figura 24 Média do Total de Tarefas Alocas em VEC e CC 92
Figura 25 Média do Total de Tarefas Alocas em VEC e CC 93

Lista de Tabelas

Tabela 1 Comparação entre VEC e VCC 38
Tabela 2 Critérios de Inclusão e Exclusão 57
Tabela 3 Número de Estudos Selecionados na Última Etapa 59
Tabela 4 Abstração do Estado da Arte 63
Tabela 5 Exemplo de Algumas Tarefas com Valores Aleatórios 80
Tabela 6 Configurações da Simulação 82
Tabela 7 Total de Entradas em Cada Execução à Partir da *Seed* 85
Tabela 8 Média das Execuções do Algoritmo METRO 86
Tabela 9 Média das Execuções do Algoritmo SRD 87
Tabela 10 Média das Execuções do Algoritmo GWO 88

Lista de Abreviações

5G *Fifth Generation*

6G *Sixth Generation*

ACO *Ant Colony Optimization*

AI *Artificial Intelligence*

BAT *Bat Algorithm*

CC *Cloud Computing*

CO *Computing Offloading*

D2D *Device-to-Device*

DT *Decision Tree*

E2E *End-to-End*

EC *Edge Computing*

ECAS *Eletric, Connected, Autonomous and Shared*

GA *Genetic Algorithm*

GT *Game Theory*

GWO *Grey Wolf Optmizer*

IoT *Internet of Things*

IoV *Internet of Vehicles*

ITS *Intelligent Transportation System*

MEC *Mobile Edge Computing*

MG *Minority Game*

ML *Machine Learning*

NE *Nash Equilibrium*

OBCs *On-bord Computers*

PD *Prisioner's Dilemma*

PO *Pareto Optimality*

QoE *Quality of Experience*

QoS *Quality of Service*

RF *Random Forest*

RQs *Research Questions*

RSUs *Road-side Units*

RT *Real Time*

SCs *Smart Cities*

SDN *Software-defined Networking*

SLR *Systematic Literature Review*

SRD *Simple Random Decision*

V2D *Vehicle-to-Device*

V2G *Vehicle-to-Grid*

V2H *Vehicle-to-Home*

V2I *Vehicle-to-Infrastructure*

V2P *Vehicle-to-Pedestrian*

V2R *Vehicle-to-Road*

V2S *Vehicle-to-Sensors*

V2V *Vehicle-to-Vehicle*

V2X *Vehicle-to Everything*

VANETs *Vehicle Ad hoc Networks*

VC *Vehicular Cloud*

VCC *Vehicular Cloud Computing*

VEC *Vehicular Edge Computing*

VECNs *Vehicular Edge Computing Networks*

VMs *Virtual Machines*

Sumário

1 Introdução **17**

1.1 Motivação e Justificativas **19**

1.2 Objetivos **20**

1.3 Organização do Trabalho **21**

2 Fundamentação Teórica **23**

2.1 Internet dos Veículos (*Internet of Vehicles* - IoV) **23**

2.1.1 Componentes e Tipos de Interações 24

2.1.2 Evolução da Arquitetura 28

2.1.3 Futuro da Mobilidade Veicular 31

2.2 Computação em Borda Veicular (*Vehicular Edge Computing* - VEC) . . . **32**

2.2.1 Benefícios da Utilização de VEC em Redes IoV 35

2.2.2 Comparativo entre VEC e VCC. 36

2.2.3 Estratégias de Alocação e Gerenciamento de Recursos em VEC 40

2.3 Métodos de Decisão para Alocação de Recursos **42**

2.3.1 Teoria dos Jogos (*Game Theory* - GT) 42

2.3.2 Decisão Aleatória Simples (*Simple Random Decision* - SRD) 45

2.4 Métodos Meta-heurísticos de Decisão para Alocação de Recursos 46

2.4.1 Método de Otimização do Lobo Cinzento (*Grey Wolf Optimizer* - GWO) . 47

2.4.2 Algoritmo do Morcego (*Bat Algorithm* - BAT) 48

2.5 Considerações Parciais 51

3 Revisão Sistemática da Literatura 53

3.1 Protocolo da Pesquisa 53

3.1.1 Questões de Pesquisa (*Research Questions* - RQs) 54

3.1.2 Estratégias de Busca 55

3.1.3 Processo de Seleção 57

3.1.4 Extração de Dados 59

3.1.5 Considerações Parciais 60

3.2 Estado da Arte 60

3.3 Ameaças e Limitações 64

4 Teoria dos Jogos em Redes IoV 67

4.1 Algoritmo METRO 68

4.2 Cenário de Execução do Algoritmo 74

4.3 Ferramentas de Simulação 76

4.3.1 *Simulation of Urban Mobility* - SUMO 76

4.3.2 *Objective Modular Network Testbed* - OMNeT++ 76

4.3.3 *Artery* 77

5 Resultados 79

5.1 Parâmetros de Simulação 79

5.1.1 Tarefas (*Tasks*) 79

5.1.2 Mapa 81

5.1.3 Configurações 82

5.2 Conjunto de Tarefas. 83

5.3 Média das Execuções 84

5.4 Métricas das Avaliações 87

5.5 Performance dos Algoritmos 89

5.6 Considerações Parciais 93

6 Conclusão 95

6.1 Principais Contribuições 96

6.2 Trabalhos Futuros 96

Referências 97

Apêndice A Código da Estrutura de Alocação de Recursos em C++ 104

Apêndice B Código do Algoritmo METRO em C++ 112

Apêndice C Código do Algoritmo SRD em C++ 116

Apêndice D Código do Algoritmo GWO em C++ 119

Anexo A Código do Artery: Arquivo *Middleware*.h 124

Anexo B Código do Artery: Arquivo *Middleware*.cpp 128

Capítulo 1

Introdução

Os avanços tecnológicos relacionados a conexão, processamento e armazenamento dos dispositivos, trouxeram novas possibilidades para o paradigma de Internet das Coisas (*Internet of Things* - IoT). Uma prova disso são os inúmeros dispositivos inteligentes presentes no dia a dia das pessoas, os quais possuem capacidade de comunicação sem fio ou estão conectados a internet para o compartilhamento de dados, como é o caso de *smartphones*, *smartwatchs*, geladeiras, câmeras, sensores, entre outros diversos dispositivos (Kavre; Gadekar; Gadhade, 2019).

Em paralelo a isso, os modelos de comunicação e infraestrutura também evoluíram, as redes móveis de Quinta Geração (*Fifth Generation* - 5G) já são uma realidade e as redes de Sexta Geração (*Sixth Generation* - 6G) prometem recursos ainda mais poderosos. Sem dúvidas, esses foram os principais fatores que contribuíram para que novas ramificações relacionadas ao campo de IoT surgissem, como a Internet dos Veículos (*Internet of Vehicles* - IoV), que basicamente herda os conceitos de IoT em um contexto de rede veicular (Zhang, 2022; Markakis, 2021; Ji *et al.*, 2020; Hammoud *et al.*, 2020; Pham *et al.*, 2020).

Uma das abstrações mais utilizadas e aceitas na literatura são as que tratam de Computação em Nuvem Veicular (*Vehicular Cloud Computing* - VCC). Esse tipo de solução é bastante eficiente, já que possui uma capacidade computacional e de armazenamento extremamente robusta, a qual basicamente cria um ambiente de colaboração veicular, que contribui para o processamento e propagação de dados, através do compartilhamento de recursos (Meneguette; De Grande; Loureiro, 2018; Meneguette; Boukerche, 2020).

Entretanto, a quantidade de veículos conectados e os seus mais variados tipos de comunicação, como a introdução do conceito de Veículo para Tudo (*Vehicle-to-Everything* - V2X) em redes IoV resultaram em diversos problemas, que estão rela-

cionados a necessidade de uma resposta menor latência, além de um controle mais eficiente para solucionar outros desafios, como o de congestionamento existente na rede (Hou *et al.*, 2020; Meneguette; Boukerche, 2017; Kazmi *et al.*, 2020).

Dessa forma, as aplicações que fazem parte do contexto de Sistema de Transporte Inteligente (*Intelligence Transportation System* - ITS) que são sensíveis a atrasos, como direção automática, direção assistida e aplicativos relacionados a emergência, necessitaram de um paradigma diferente da VCC, com o objetivo de garantir fatores de Qualidade do Serviço (*Quality of Service* - QoS) e a Qualidade da Experiência (*Quality of Experience* - QoE) em redes IoV (Wang *et al.*, 2020; Meneguette; Madeira; Bittencourt, 2012).

Nesse sentido, a Computação em Borda Veicular (*Vehicular Edge Computing* - VEC), localizada na borda da rede, que herda os conceitos de mobilidade e baixa latência da Computação em Borda Móvel (*Mobile Edge Computing* - MEC), foi proposta para solucionar os desafios exigidos por aplicativos de Tempo Real (Real Time - RT), por exemplo. Além disso, o objetivo principal dessa proximidade dos servidores VEC em relação aos usuários, é a contribuição que essa tecnologia possui para resolução de problemas relacionados ao congestionamento da rede (Zhang *et al.*, 2019; Xiao *et al.*, 2020; Meneguette *et al.*, 2021).

Porém, os conceitos ainda são considerados bastante emergentes, e diversas novas propostas de arquiteturas estão sendo disseminadas no universo científico. Assim sendo, algumas arquiteturas existentes na literatura estão apresentadas, mais especificamente as arquiteturas que fazem o uso de duas, três e quatro camadas, em que cada camada possui uma responsabilidade específica, as quais serão definidas ao longo desta monografia. Este trabalho foca na arquitetura de três camadas, que é bastante aceita na academia, devido a sua eficiência na divisão de responsabilidades e solução de tarefas (Meneguette *et al.*, 2021; Zhang, 2022).

Além disso, quando se trata do contexto veicular, a mobilidade é um fator extremamente desafiador, já que os veículos precisam fazer o Descarregamento de Computação (*Computing Offloading* - CO) nos servidores VEC, que irão realizar o processo de decisão de alocação de recursos. Isto quer dizer que, por se tratar de um sistema distribuído, altamente dinâmico, diversos recursos são disputados, necessitando de um controle eficiente para garantir performance, segurança e confiabilidade, por exemplo (Dai *et al.*, 2018).

Sendo assim, o objetivo deste trabalho é apresentar um algoritmo baseado na Teoria dos Jogos (*Game Theory* - GT) com o propósito de maximizar a decisão de alocação de recursos em servidores VEC no contexto de redes IoV. Esse algoritmo é aplicado

em uma arquitetura VEC específica de três camadas, que deixa a responsabilidade de CO para os veículos em servidores VEC, os quais devem ser redirecionados ou não as camadas superiores com base em fatores de sensibilidade a atraso e ou congestionamento de rede, por exemplo.

Para concluir, todo o contexto necessário para o entendimento do trabalho está descrito ao longo desta monografia, com base em uma coleção de trabalhos obtidos na literatura por meio de uma extensa Revisão Sistemática da Literatura (SLR). Além disso, há também informações à respeito da metodologia definida e os resultados que foram obtidos por meio das simulações que foram realizadas. Por fim, uma discussão acerca dos dados gerados, demonstrando a eficiência do algoritmo para aplicações sensíveis a atrasos e sendo extremamente aplicável para colaborar no desenvolvimento no contexto das redes IoV.

1.1 Motivação e Justificativas

Os automóveis possuem diversos dispositivos e sensores embarcados que são utilizados para resolver algumas questões objetivas. Entretanto há uma dicotomia em relação a esses dispositivos, já que em alguns casos há uma grande falta de processamento para executar determinadas tarefas e em outros casos há uma ociosidade no uso dos recursos desses dispositivos (Wang *et al.*, 2020).

Atualmente, a estratégia mais difundida é o paradigma VCC, que é basicamente uma solução propícia para explorar os recursos ociosos dos veículos, com o objetivo de atender as solicitações dos aplicativos das tradicionais Redes Veiculares Ad Hoc (Vehicular Ad Hoc Networks - VANETs) (Meneguette; De Grande; Loureiro, 2018).

No entanto esses aplicativos podem ser classificados em diversas categorias, como em soluções de entretenimento que não são sensíveis a atrasos, ou em serviços de emergência que necessitam de uma latência mínima, sendo esse último praticamente impossível ser executado em um ambiente VCC tradicional, por conta da sua arquitetura baseada em nuvem (Wang *et al.*, 2020).

Além disso, um outro fator que deve ser levado em consideração é o surgimento das redes 5G e a promissora tecnologia 6G. Essas redes trazem novas possibilidades para serem exploradas, e uma das áreas que se beneficiou dessa evolução foi a IoT, que serviu como modelo para que seus conceitos fossem compartilhados em outros campos de pesquisas, como é o caso da IoV, que explora diversos tipos de comunicação com foco no contexto veicular (Zhang, 2022; Markakis, 2021).

Nas redes IoV, esses veículos podem se comunicar de diversas maneiras, como é

o caso da comunicação V2X que foi introduzida nesse tipo de infraestrutura, que em resumo representa a comunicação do veículo com qualquer outro tipo de dispositivo inteligente integrado a rede, por isso a utilização da letra X (Meneguette; De Grande; Loureiro, 2018).

Enquanto aplicativos de entretenimento são bem atendidos em contextos VCC, os serviços de emergência não conseguem fazer o uso de recursos em RT nesses ambientes, já que a latência é relativamente alta para esses sistemas, não garantindo dessa forma os fatores necessários de de QoS e QoE (Zhang *et al.*, 2019; Zhang, 2022).

Dessa forma, faz-se necessário utilizar outros paradigmas, com o objetivo de alcançar uma melhor eficiência em redes IoV, como é o caso da MEC, que foi proposta em um contexto veicular como VEC. Entretanto, é essencial desenvolver algoritmos precisos para orquestrar todo esse ambiente altamente complexo de alocação de recursos. (Zhang *et al.*, 2019; Zhang, 2022).

1.2 Objetivos

Essa monografia resolve os desafios relacionados ao CO e a decisão de alocação e gerenciamento de recursos, através do uso de algoritmos baseados em GT, em um contexto que envolve uma arquitetura VEC de três camadas, que mescla os conceitos de MEC com CC, sendo especialmente proposta para redes IoV.

Os objetivos específicos deste trabalho são:

- Implementar algoritmos baseado em GT para montagem de jogos entre os recursos disputados, visando a maximização da CO e decisão de alocação e gerenciamento de recursos em redes IoV;
- Utilizar uma arquitetura VEC de três camadas, que abstrai os paradigmas de MEC e CC para o contexto veicular com o objetivo de atender todos os tipos de solicitações (sensíveis ao atraso ou não) em redes IoV;
- Resolver os problemas de alta latência dos aplicativos ITS sensíveis a atrasos, além dos problemas de congestionamento das redes IoV;
- Modelar cenários urbanos compostos por redes IoV que utilizam a arquitetura proposta no trabalho, com o intuito de obter dados em ambientes de simulação; e
- Comparar os resultados com os trabalhos obtidos na literatura à partir da revisão sistemática, como forma de avaliar a eficiência e validar a aplicabilidade dos

algoritmos no ambiente proposto.

1.3 Organização do Trabalho

O presente trabalho está organizado em seis capítulos:

- **Capítulo 1:** Este primeiro capítulo aborda o conteúdo relacionado a introdução deste trabalho, contendo uma apresentação do contexto, argumentação da motivação, detalhes dos objetivos, além desta própria seção, que descreve como este trabalho está dividido.
- **Capítulo 2:** Expõe toda a fundamentação teórica, contextualizando os conceitos que fazem parte do universo do trabalho, por meio de pesquisas sobre IoV e VEC, além das principais estratégias de algoritmos para a alocação de recursos e das considerações parciais obtidas ao final do capítulo.
- **Capítulo 3:** Apresenta uma extensa pesquisa que contém um compilado de trabalhos existentes na literatura, elegidos à partir de uma *string* de busca, que foi utilizada em bibliotecas digitais e outros acervos científicos, desenvolvidas por meio da metodologia de SLR que contém um protocolo bem definido.
- **Capítulo 4:** Detalha a metodologia desta monografia, por meio da explicação do funcionamento do algoritmo proposto, que foi intitulado de METRO, além do contexto e dos parâmetros que foram utilizados para o ambiente de simulação.
- **Capítulo 5:** Apresenta todos os parâmetros utilizados no processo de simulação, os resultados da média das execuções, além da análise da performance de cada um dos algoritmos.
- **Capítulo 6:** Sintetiza todo o trabalho, as principais contribuições, as produções científicas proporcionadas para a academia, e por fim, a possibilidade de trabalhos futuros que podem ser devolvidos à partir deste.

Capítulo 2

Fundamentação Teórica

Este capítulo possui o objetivo de delinear os conceitos utilizados neste trabalho, além de contextualizar todo o cenário em que o mesmo está inserido. Nesse sentido, o capítulo está dividido em cinco seções, sendo que na Seção 2.1 são abordados os conceitos relacionados a Internet dos Veículos *(Internet of Vehicles - IoV)*, seu funcionamento, sua importância para o Sistema de Transporte Inteligente (*Intelligence Transportation System* - ITS), sua arquitetura atual, um tipo de arquitetura emergentes, e por último uma previsão do que é esperado para o futuro no contexto de IoV; Na Seção 2.2 é apresentado um conteúdo aprofundado sobre Computação em Borda Veicular *(Vehicular Edge Computing - VEC)*, tratando de pontos que estão relacionados ao funcionamento, arquitetura, benefícios da sua utilização em redes IoV, um comparativo entre VEC e Computação em Nuvem Veicular (*Vehicular Cloud Computing* - VCC) e ao final uma introdução sobre o gerenciamento de recursos nesses ambientes; Na Seção 2.3 estão descritos de forma detalhada todo o fluxo de funcionamento de dois métodos existentes para decisão de alocação de recursos: Teoria dos Jogos (*Game Theory* - GT) e Árvore de Decisão (*Decision Tree* - DT); Na Seção 2.4 estão descritos de forma detalhada todo o fluxo de funcionamento de dois métodos metaheurísticos existentes para decisão de alocação de recursos: Algoritmo do Morcego (*Bat Algorithm* - BAT) e Método de Otimização do Lobo Cinzento (*Grey Wolf Optmizer* - GWO). Por fim, na Seção 2.5 contém algumas considerações baseadas no que foi apresentado durante o capítulo.

2.1 Internet dos Veículos (*Internet of Vehicles* - IoV)

O campo de pesquisa de IoT é considerado um agente essencial para a evolução de diversas áreas. Uma das áreas que testemunha essa contribuição é a IoV, que basi-

camente permite a troca de informação entre veículos e infraestruturas. Nesse sentido, fica evidente que toda essa arquitetura é complexa, devido a alta mobilidade existente nesse contexto, fator responsável pela causa de diversos problemas, os quais estão relacionados a alocação de recursos, segurança, privacidade, entre outros, por exemplo. Dessa forma, torna-se indispensável o desenvolvimento de novas soluções que resolvam esses obstáculos desafiadores, referentes ao regimento desse tipo de tecnologia (Hammoud *et al.*, 2020; Garg *et al.*, 2020).

Em outras palavras, IoV pode ser descrita como uma rede veicular distribuída, que está qualificada para gerenciar todos os usuários que estão conectados a ela, administrando recursos de comunicação e armazenamento, além de ser capaz de utilizar Aprendizagem de Máquina (*Machine Learning* - ML), antecipando de forma inteligente as intenções dos usuários envolvidos nesse cenário. Tudo isso só é possível graças aos vários componentes presentes em sua arquitetura, que podem ser classificados como componentes físicos e lógicos, sendo respectivamente veículos ou infraestrutura e aprendizagem ou algoritmos, por exemplo. (Markakis, 2021).

Ainda em (Markakis, 2021), IoV é retratada como uma plataforma de agregação, a qual permite a comunicação entre diversos componentes, desde de veículos inteligentes, sensores, câmeras, pedestres e dispositivos de infraestrutura, como as Unidades de Beira de Estrada *(Road-side Units* - RSUs). Os RSUs são dispositivos de comunicação sem fio que ficam distribuídos na infraestrutura a beira da estrada, os quis são responsáveis pelo fornecimento de conectividade e suporte aos veículos que passam, por meio do compartilhamento de informações.

Ademais, IoV integra redes veiculares heterogêneas, como é o caso das tradicionais Redes Veiculares *Ad Hoc* (*Vehicular Ad Hoc Networks* - VANETs) e Redes de Computação em Borda Veicular (*Vehicular Edge Computing Networks* - VECNs).

Em conclusão, os trabalhos de (Mollah *et al.*, 2021; Ang *et al.*, 2019; Meneguette *et al.*, 2021) explicam que embora IoV ainda seja considerado um conceito emergente, ele pode ser capaz de auxiliar a concretização de ITS em um futuro próximo, assim que alguns desafios relacionados a infraestrutura, CO (processo de transferência de dados dos dispositivos para ambientes robustos de processamento), além da decisão de alocação de recursos sejam solucionados.

2.1.1 Componentes e Tipos de Interações

A tecnologia atual já proporciona a criação de dispositivos extremamente compactos, que estão preparados para execução de determinadas tarefas, possuindo grande eficiência energética, integrando funcionalidades de conectividade e de compartilha-

mento de dados, por um preço extraordinariamente baixo, resultando em um processo que elevou a experiência dos usuários para um próximo nível (Shah; Yaqoob, 2016).

Naturalmente, houve uma expansão na utilização desses dispositivos, que começaram a ser empregues em larga escala, além da evolução tecnológica da comunicação e infraestrutura, fatores que contribuíram para o advento dos conceitos conhecidos atualmente no campo de pesquisa IoT. Não demorou muito para que essa abstração surgisse em outras áreas, como foi o caso de IoV, que utiliza IoT juntamente com as redes móveis em um contexto onde os veículos funcionam em uma rede como nós ou objetos inteligentes (Ang *et al.*, 2019).

Por conta disso, diversos conceitos, nomenclaturas e funcionalidades são compartilhados do universo IoT para o universo IoV. Um exemplo que pode ser ilustrado são os componentes, que no cenário IoV também são classificados em objetos físicos, os quais representam os objetos inteligentes, que podem ser sensores ou veículos, por exemplo. Outro classificação existente são as dos componentes lógicos, compostos por algoritmos, por exemplo. Esses componentes lógicos possuem diversas responsabilidades, como alocação e gerenciamento de recursos. Além disso, o uso de Inteligência Artificial (*Artificial Intelligence* - AI) também é um exemplo de componente lógico (Markakis, 2021; Ang *et al.*, 2019).

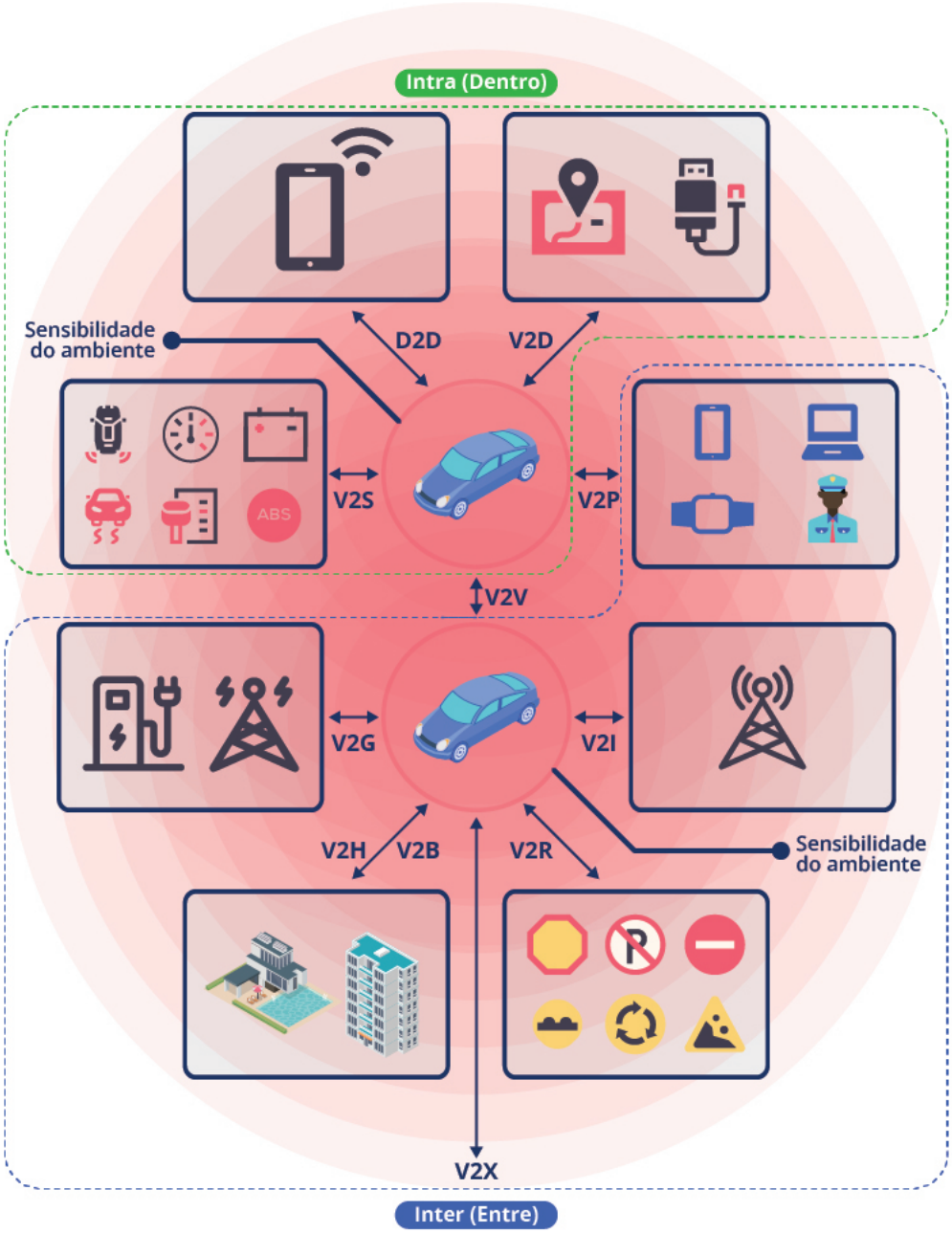
Nesse sentido, os trabalhos de (Ang *et al.*, 2019; Meneguette; De Grande; Loureiro, 2018) explicam que no contexto de IoV, esses objetos inteligentes são representados por sistemas, que também são conhecidos como modelos dinâmicos de comunicação móvel. Esses sistemas constituem-se basicamente a organização da comunicação entre veículos e outros objetos, as quais podem acontecer de diversas formas, como as apresentadas na Figura 1

Em suma, a Figura 1 pode ser explicada da seguinte maneira: Há duas possibilidades de interações, sendo dentro do veículo (*intra-vehicle*) ou entre veículos (*inter-vehicle*). No primeiro caso, temos a interação *intra-vehicle*, que pode ser exemplificada com as comunicações de (Ang *et al.*, 2019; Ji *et al.*, 2020; Meneguette; De Grande; Loureiro, 2018):

- **Dispositivo para Dispositivo (*Device-to-Device* - D2D):** Conexão direta entre um dispositivo e o veículo, ou seja, um *smartphone* conectado via *bluetooth*, por exemplo.

- **Veículo para Dispositivo (Vehicle-to-Device - V2D):** Dispositivos conectados diretamente com o veículo, como um *smartphone* espelhando a tela no aparelho multimídia do carro.

Figura 1: Os principais tipos de interações em redes IoV



Fonte: Adaptado de (Ang *et al.*, 2019; Ji *et al.*, 2020) .

- **Veículo para Sensores (*Vehicle-to-Sensors* - V2S):** Interação entre os sensores com o veículo, como indicador de bateria, óleo, controle de tração e estabilidade, entre outros componentes do carro.
- **Veículo para Pedestre (*Vehicle-to-Pedestrian* - V2P):** Comunicação entre pedestres (ou dispositivos utilizados pelos mesmos) e veículos, pode ser utilizada em aplicativos ITS que visam a segurança, por exemplo.

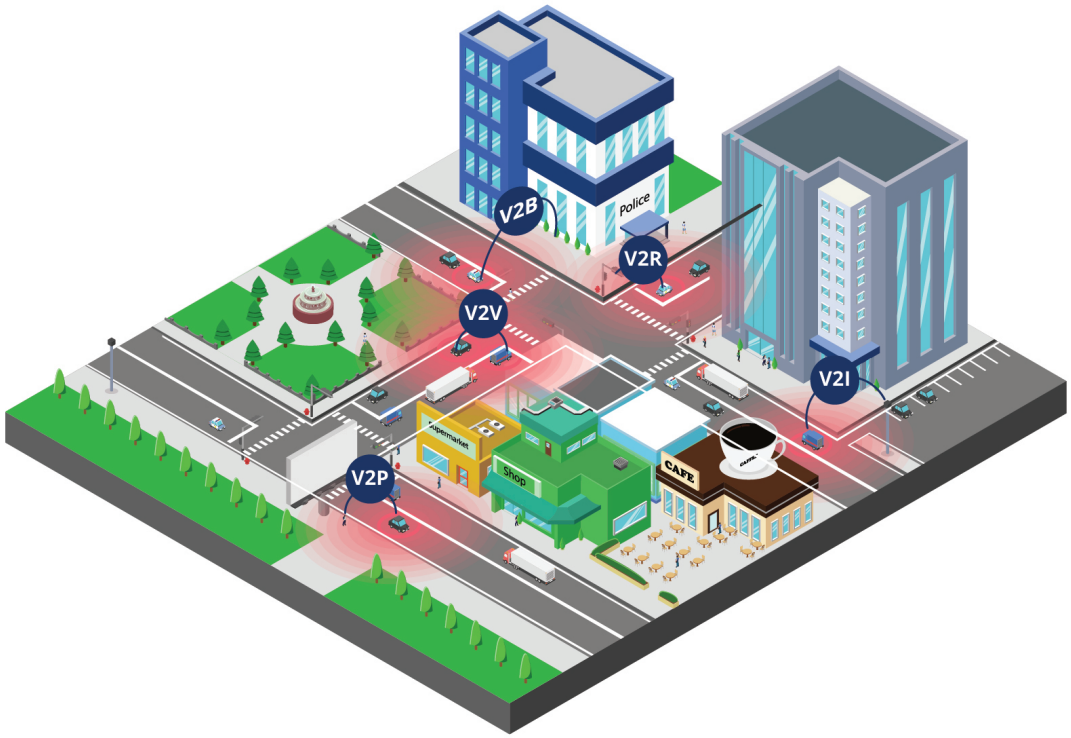
Ainda sobre a Figura 1 existe o segundo caso de interação que é chamada de interação *inter-vehicle*, que pode ser ilustrada nas comunicações de (Ang *et al.*, 2019; Ji *et al.*, 2020; Meneguette; De Grande; Loureiro, 2018):

- **Veículo para Veículo (*Vehicle-to-Vehicle* - V2V):** Troca de informações entre veículos, interação que possui inúmeras funcionalidades.
- **Veículo para Rede (*Vehicle-to-Grid* - V2G):** Interação entre rede elétricas ou estações de carregamento e veículos.
- **Veículo para Infraestrutura. (*Vehicle-to-Infrastructure* - V2I):** Comunicação existente os veículos e infraestrutura, como dispositivos de Unidades de Beira de Estrada *(Roadside Units) - RSUs* ou servidores VEC.
- **Veículo para Casa (*Vehicle-to-Home* - V2H)/Veículo para Edifício (*Vehicle-to-Building* - V2B):** Interação entre casas e edifícios com veículos, por exemplo: Garagem abre o portão automaticamente quando o veículo se aproxima ou um veículo que troca informações com um edifício, como uma delegacia de polícia.
- **Veículo para Estrada (*Vehicle-to-Road* - V2R):** Tipo de troca de dados feita entre veículos e placas, semáforos, entre outros componentes de sinalização existentes nas rodovias.
- **Veículo para Tudo (*Vehicle-to-Everything* - V2X):** Conceito que foi introduzido em redes IoV, basicamente traz a ideia um conceito que torna o veículo interativo, ou seja, o qual possui a capacidade de "sentir"o ambiente e se comunicar com qualquer outro objetivo "inteligente".

A fim de elucidar melhor esse cenário, a Figura 2 traz a demonstração dos exemplos de funcionamento dos diversos tipos de interações que foram citados na Figura 1

Através da Figura 2, fica perceptível visualizar os diferentes mecanismos de troca de informações existentes dentro do conceito de IoV. A principal diferença é que IoV implementa o conceito de V2X, ou seja, a comunicação entre veículos e qualquer outro tipo de dispositivo "inteligente". É fortemente importante salientar que esses processos de comunicação ocorrem de forma extremamente rápida, e também podem acontecer de forma simultânea, devido ao fator que está relacionado a alta mobilidade existente nesse contexto. Além disso, esses processos compreendem diferentes tipos de arquiteturas de redes, graças a abstração que são herdadas no campo de IoT.

Figura 2: Rede IoV: Seus agentes e interações



Fonte: Elaborado pelo autor, 2021.

2.1.2 Evolução da Arquitetura

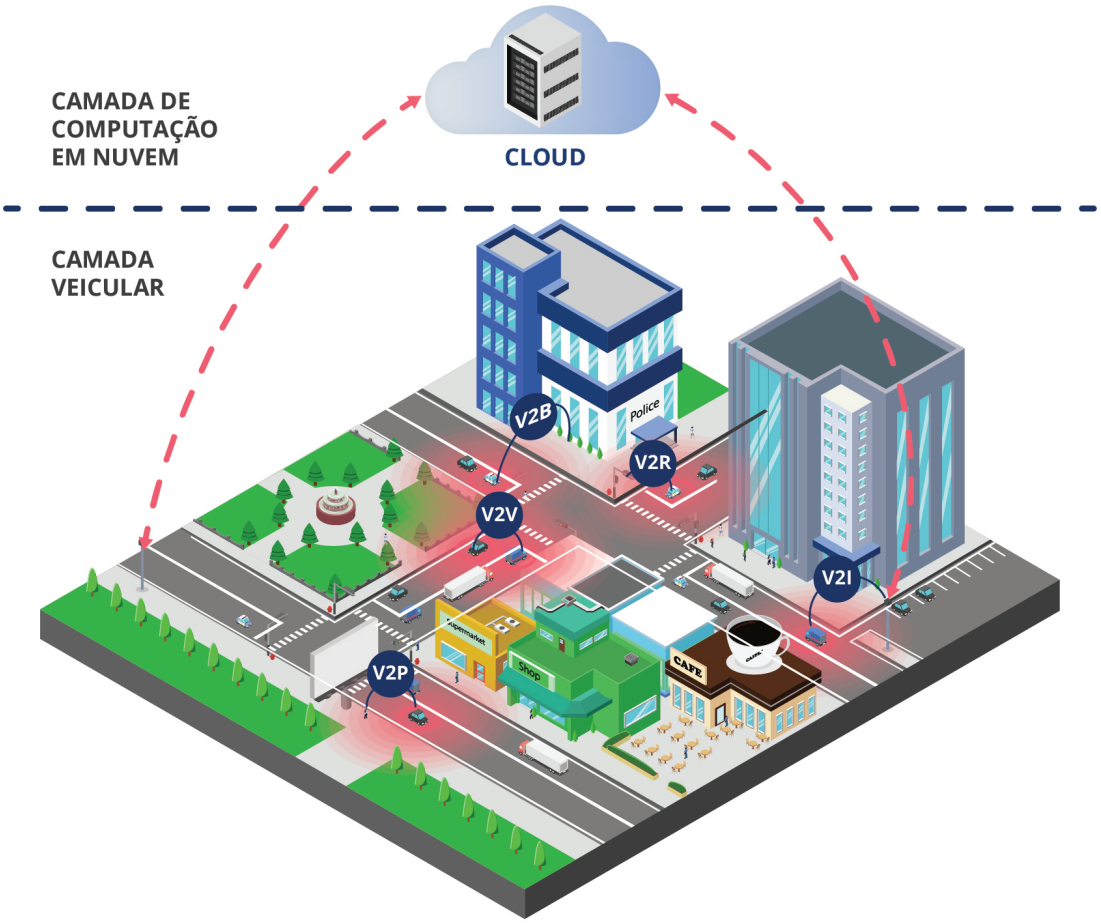
A literatura vêm apresentado diversos trabalhos com propostas de novos tipos de arquiteturas para a tecnologia IoV. Em suma, grande parte dessas teorias convergem para um objetivo específico, que é dividir responsabilidades para a resolução de grandes desafios, que estão relacionados a alocação de recursos, segurança, privacidade, ML, entre outros. São propostas relacionadas a parte física, divisão da arquitetura em camadas ou desenvolvimento de algoritmos que gerenciam de forma mais eficiente todo o funcionamento do sistema, como está descrito nos trabalhos (Ang *et al.*, 2019; Hou *et al.*, 2020; Ji *et al.*, 2020; Zhang *et al.*, 2021; Markakis, 2021).

Dessa forma, o objetivo principal deste item é apresentar um comparativo entre a arquitetura já existente, utilizada no contexto de IoV, além de uma arquitetura emergente, que visa explorar recursos da Quinta Geração (*Fifth Generation* - 5G) e principalmente da futura Sexta Geração (*Sixth Generation* - 6G) das redes móveis (Ji *et al.*, 2020).

Atualmente, a arquitetura padrão de IoV pode ser ilustrada através da Figura 3

Conforme pode ser observado na Figura 3, a arquitetura atual não está preparada

Figura 3: Arquitetura usual das redes IoV



Fonte: Adaptado de (Ang *et al.*, 2019; Ji *et al.*, 2020).

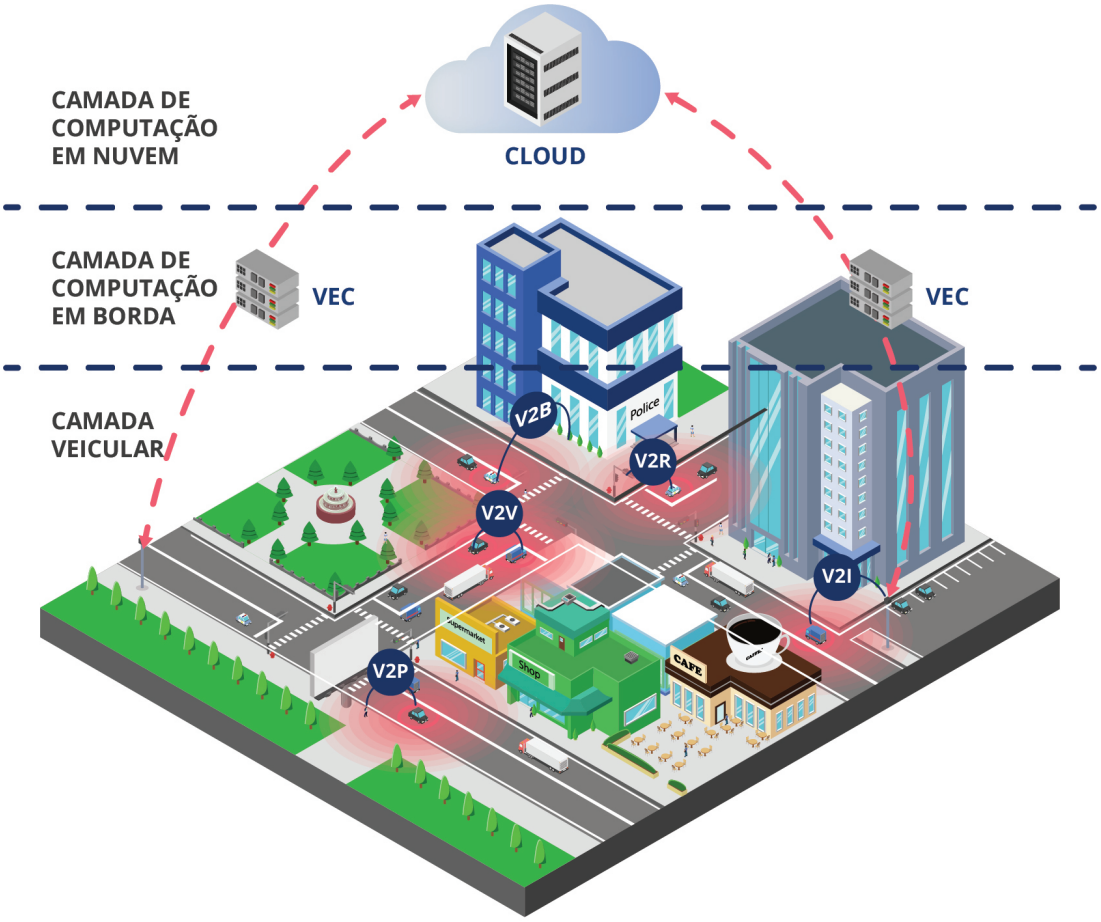
para as novas tecnologias de conexão 5G e 6G, o que impossibilita uma série de recursos relacionados a Tempo Real *(Real Time - RT)*. (Meneguette; De Grande; Loureiro, 2018) descreve os desafios de se criar um sistema seguro, robusto e que funciona em RT para Cidades Inteligentes (*Smart Cities* - SCs). O mesmo vale para o conceito de IoV, já que a sua arquitetura de infraestrutura possui inúmeras semelhanças em relação a IoT, como fazer o uso da técnica VCC, que compõe a atual arquitetura de rede IoV (Ji *et al.*, 2020; Meneguette; De Grande; Loureiro, 2018).

Entretanto, diversos sistemas críticos são sensíveis a latência avançada, como é o caso de aplicativos de direção assistida, direção automática e gerenciamento de falha de emergência, que necessitam de uma estratégia diferente da VCC, técnica que funciona muito bem para sistemas que são tolerantes a falha, como é o caso de aplicativos de entretenimento. Nesse sentido, a VEC surge como uma solução extremamente in-

teressante para o contexto IoV, já que a mesma estende recursos de processamento e armazenamento para a borda da rede, fornecendo serviços de computação de baixa latência (Hou *et al.*, 2020).

Por isso, novas arquiteturas vêm sendo apresentadas na literatura, principalmente pelo fato de que a tecnologia 5G já é uma realidade em diversas localidades, além da futura tecnologia 6G, que já é visada como a solução de grandes desafios enfrentados hoje nas redes IoV. A Figura 4 contém uma ilustração de um novo conceito de arquitetura para IoV, que é extremamente interessante e funcional (Meneguette *et al.*, 2021; Zhang *et al.*, 2021):

Figura 4: Nova Arquitetura para redes IoV



Fonte: Adaptado de (Meneguette *et al.*, 2021; Liu *et al.*, 2021; Zhang, 2022)

Diferente da Figura 3 que demonstra uma arquitetura baseada no uso de Computação em Nuvem (*Cloud Computing* - CC), a Figura 4 apresenta uma outra abordagem, a qual traz os recursos de VEC com o intuito de melhorar o atendimento as aplicativos

sensíveis ao atraso, fornecendo o atendimento RT. É importante salientar que a CC ainda tem um papel fundamental na resolução de processamento pesado, e pode ser utilizada para serviços que não necessitam de baixas latências, como aplicativos de entretenimento.

Mediante ao que foi exposto neste item, fica perceptível visualizar que a abstração IoV realmente funciona como uma plataforma de agregação, já que o conceito faz o uso dos veículos como objeto "inteligente"central, o qual é capaz de se comunicar com diversos outros dispositivos em contextos de infraestrutura de redes diferentes (Markakis, 2021).

2.1.3 Futuro da Mobilidade Veicular

O cenário tradicional de transporte está passando por um processo evolutivo, devido a diversos fatores, sendo eles: novos comportamentos e modelos de mobilidade, utilização de energia elétrica para locomoção, aplicabilidade da tecnologia de direção autônoma, preocupação com questões de privacidade, segurança e proteção, além da empregabilidade de técnicas de AI para aprendizagem e tomada de decisão. Uma das contribuições para a integração desses fatores é o IoV, que resultam em uma mobilidade mais conectada, segura e também mais verde (Markakis, 2021).

Dessa forma, diversos conceitos emergentes estão sendo apresentados como potenciais soluções futuras, para os mais variáveis problemas. Os trabalhos (Markakis, 2021). (Zhang *et al.*, 2021) e (Kombate; Wanglina, 2016; Wang; Zhu; Hossain, 2021) apresentam respectivamente estudos que detalham os conceitos: Elétrico, Conectado, Autônomo e Compartilhado (*Eletric, Connected, Autonomous and Shared* - ECAS) nos cenários de mobilidade veícular, Computação de Borda Móvel (*Mobile Edge Computing* - MEC) e as possibilidades existentes e futuras com as novas gerações de redes móveis 5G e 6G.

O conceito ECAS possui uma abordagem que converge a inteligência da computação de borda existentes nos veículos baseados em tecnologias de AI, resultando em uma inteligência colaborativa, distribuída, a qual engloba todos os dispositivos IoT, que faz o uso de uma infraestrutura de computação de ponta, com processos cognitivos analíticos que são organizados hierarquicamente em espaços físicos, de infraestrutura de rede e dados. (Markakis, 2021).

Seguindo um raciocínio semelhante em termos de utilização de computação de borda, (Zhang *et al.*, 2021) apresenta uma pesquisa que descreve o MEC sendo integrado as redes IoV como uma consequência da tecnologia 5G. Essa integração garante uma melhora na performance da rede, resultando em uma menor sobrecarga e

uma baixa latência. Entretanto a alta mobilidade do cenário ainda assombra em questões que estão relacionadas a estratégias de descarregamento de tarefas de computação consideradas mais complexas, exigindo assim soluções que resolvam o problema de alocação de recursos em ambientes de computação de ponta para redes IoV.

Em paralelo a isso, as redes móveis 5G e 6G trazem novas possibilidades pertinentes a IoV. O 5G por sua vez, possui a capacidade de auxiliar na expansão da infraestrutura dessas redes, oferecendo uma maior largura de banda e confiabilidade, que por resultado traz um maior número de terminais conectados e um aumento no número dados trocados, além da computação com uma latência ainda menor, requisito exigido por alguns aplicativos ITS (Kombate; Wanglina, 2016; Wang; Zhu; Hossain, 2021).

Já o emergente 6G caminha para um desenvolvimento verde do IoV, visando questões que possui relação com mecanismos computacionais, comunicação, tráfego, entre outros recursos que serão elevados a sua carga máxima de trabalho, devido ao grande número de dispositivos na rede. Por isso, que os estudos de 6G frente a IoV focam na otimização de energia, mais precisamente em aspectos relacionados a carga de trabalho, roteamento, alocação de recursos, entre outros (Kombate; Wanglina, 2016; Wang; Zhu; Hossain, 2021).

Enfim, ainda há um longo caminho a ser percorrido para integrar todos esses veículos, objetos inteligentes e infraestruturas de rede que fazem parte do cenário IoV. Além disso, é extremamente importante que os fatores de Qualidade do Serviço (*Quality of Service* - QoS) e a Qualidade da Experiência (*Quality of Experience* - QoE) sejam atingidos com eficiência, e para isso acontecer, é preciso um conjunto de contribuições de diversos setores, que parte das empresas de telecomunicações, as quais devem fornecer infraestrutura adequada e da indústria automotiva, que necessita investir em engenharia para equipar seus veículos com dispositivos capazes de servir toda a tecnologia IoV (Dureja; Sangwan, 2021).

2.2 Computação em Borda Veicular (*Vehicular Edge Computing* - VEC)

Durante muitos anos, pesquisadores vêm apresentando diversos estudos relacionados ao uso do paradigma de CC no contexto veicular. O resultado disso foi o surgimento de um novo conceito, muito conhecido na literatura, que foi intitulado de VCC. Em resumo, uma Nuvem Veicular (*Vehicular Cloud* - VC) nada mais é do que um conjunto de veículos que compartilham seus recursos de computação, criando uma

interação entre os próprios veículos e os RSUs; (Meneguette; De Grande; Loureiro, 2018).

Este tipo de tecnologia é bastante utilizado para servir diversos contextos que necessitam de uma carga pesada de processamento, o que acontece em redes IoT e IoV, por exemplo. Nesse sentido, a alta capacidade de computação e recursos de armazenamento existentes no conceito de nuvem, tornam-se soluções extremamente poderosas para inúmeros tipos de aplicações (Liu *et al.*, 2021). No geral, até então apenas duas possibilidades diferentes eram exploradas e utilizadas como alternativas para o processamento de aplicativos em redes IoV. De acordo (Wang *et al.*, 2020), a primeira opção descreve a utilização de Computadores de Bordo (*On-bord Computers* - OBCs) de alto desempenho sendo integrado aos veículos, já a segunda, explora o uso de CC, conceito que foi abordado no início desta seção.

Entretanto para que todo esse ecossistema de rede IoV funcione corretamente, é necessário um outro tipo de tecnologia envolvida na infraestrutura, que garanta uma latência menor, além de uma maior confiabilidade para este cenário. Essa tecnologia deve servir todos os sistemas e aplicativos (em especial os sensíveis a atrasos) em termos de processamento e armazenamento de dados, com algoritmos que sejam capazes de alocar esses recursos da maneira mais eficiente possível, garantindo assim fatores relacionados a QoS e a QoE nesses ambientes de extrema mobilidade (Gu; Zhang; Cao, 2021; Raza *et al.*, 2019; Nguyen *et al.*, 2018).

Segundo (Wang *et al.*, 2020), há um grande problema com as duas alternativas citadas anteriormente, já que a primeira proposta de integração de OBCs de alto desempenho nos veículos resultariam em uma elevação altíssima no custo de fabricação do mesmo, e consequentemente iria inviabilizar o preço final que seria ofertado ao usuário. Tudo isso somado ao fato de que esse tipo de arquitetura isolaria os veículos uns dos outros, diferindo-se assim dos conceitos de conectividade e comunicação das redes IoV, razões que tornam essa solução impraticável.

Ainda de acordo com (Wang *et al.*, 2020), a segunda alternativa foca em distribuir essa carga de processamentos e armazenamento através do uso de CCs, no entanto essa solução garante atendimento a apenas uma parcela dos aplicativos, já que a distância da nuvem do usuário resulta em uma grande latência, impossibilitando assim a implantação do conceito de RT, existente em ITS e outros aplicativos que são sensíveis a atrasos, envolvidos no cenário de redes IoV, por exemplo.

Por isso, novas alternativas foram exploradas ao longo do tempo, como é o caso da VEC, tecnologia promissora que herda praticamente todos conceitos de MEC em um contexto preparado para veículos. Em resumo, uma MEC é baseada nos funda-

mentos da Computação de Borda (*Edge Computing* - EC), que visa transferir tarefas que exigem alto processamento para servidores VEC, os quais estão localizados mais próximos dos usuários em relação aos servidores CCs, garantindo assim uma latência muito menor, possibilitando o uso de aplicativos sensíveis a atrasos, como direção assistida ou direção automática, como pode ser ilustrado na Figura 5 (Zhang, 2022; Mehrabi *et al.*, 2019; Dai *et al.*, 2018)

Figura 5: Proximidade dos servidores VEC em relação aos servidores CC

Fonte: Elaborado pelo autor, 2021.

A Figura 5 ilustra como a VEC está localizada mais próxima dos usuários em relação a CC, isso pode ser usado de uma forma extremamente eficiente para resolver os desafios de latência existentes no contexto de IoV. No entanto, esses recursos disponíveis nos servidores VEC são limitados, o que novamente traz o desafio de encontrar uma melhor maneira para alocar e gerenciar recursos nesse tipo de ambiente que ne-

cessita ser bastante flexível, por conta da alta mobilidade existente, principalmente para atender aplicações que são extremamente sensíveis a atrasos (Dai *et al.*, 2018).

Um outro exemplo pode ser encontrado no trabalho de (El-Sayed; Chaqfeh, 2019), o qual tem a proposta de utilizar MEC como solução para a nova geração de ITS SCs. Para comprovar o resultado, são apresentados dados de uma simulação, os quais mostram que o protocolo proposto foi o responsável por trazer uma melhora significativa em relação ao desempenho, comunicação e atraso nesses cenários.

Em conclusão é perceptível que, o uso de VEC em redes IoV traz diversas melhorias, além de novas possibilidades, principalmente para as aplicações que sofrem com os problemas relacionados a latência dos servidores baseado em nuvem. Entretanto, como citado anteriormente é preciso de um ambiente controlado, já que esses recursos são limitados, ou seja, é faz-se necessário alocar e gerenciar da forma mais eficiente possível os recursos, para extrair ao máximo o potencial existente nessa tecnologia (Zhang, 2022; Meneguette *et al.*, 2021).

2.2.1 Benefícios da Utilização de VEC em Redes IoV

De acordo com (Borcoci, 2021) inúmeros benefícios são proporcionados com o uso de VEC aos serviços e as gerações atuais de redes móveis, como o 5G e as novas, como 6G. Dentre eles, os mais importantes que podem ser citados são uma melhor flexibilidade em todo o gerenciamento, aumento expressivo na largura de banda, melhor capacidade de suporte a tráfego, tempo de resposta mais eficiente, maior número de terminais disponíveis, mais eficiência energética, entre outros.

Um dos fatores mais importantes da utilização desse esse tipo de tecnologia é que a VEC oferece a possibilidade dos veículos inteligentes realizarem transferências de tarefas pesadas de processamento para seus servidores, devido ao fato de que o OBC presente nos veículos possuem um baixo poder computacional. Além disso, essas informações processadas serão devolvidas de uma forma muito mais rápida, ou seja, com uma menor latência, contribuindo assim para melhorar o congestionamento existente na rede, devido a enorme quantidade de dados dos inúmeros dispositivos inteligentes compartilhados nesse ambiente (Zhang, 2022).

Algumas arquiteturas propõem a união dos paradigmas VEC e CC, como descrevem (Gu; Zhang; Cao, 2021; Liu *et al.*, 2021; Wang *et al.*, 2020; El-Sayed; Chaqfeh, 2019; Meneguette *et al.*, 2021), em seus trabalhos. Basicamente a arquitetura mais comum se divide em 3 camadas, sendo respectivamente a camada dos veículos, a camada da EC, e por fim a camada de CC (Meneguette *et al.*, 2021) ilustra a diferença entre arquiteturas VECs com duas, três e quatro camadas, sendo respectivamente pou-

cas camadas para um funcionamento, nível ideal de camadas para resolver todas as tarefas, excesso de camadas que aumenta a complexidade e congestiona a rede com mensagens de serviços.

A Figura 6 tem o objetivo de ilustrar a integração entre a arquitetura VEC de três camadas com a CC, proposta em diversos trabalhos (Gu; Zhang; Cao, 2021; Liu *et al.*, 2021; Wang *et al.*, 2020; El-Sayed; Chaqfeh, 2019; Meneguette *et al.*, 2021)

Em resumo, é possível identificar na Figura 6 a interação existente entre os servidores VEC e CC, os quais ajudam a equilibrar os recursos disponíveis nas redes, alocando os mesmos com base em suas necessidades, geralmente priorizando fatores que estão relacionados a sensibilidade de atraso, ou seja, privilegiando aplicativos que necessitam de uma latência mínima em servidores VEC e direcionando aplicativos que não necessitam para servidores CC.

Dessa forma, é importante ressaltar como as redes IoV utilizam diversas estratégias relacionadas a AI e *Blockchain* para melhorar as capacidades e segurança desse ambiente. Entretanto, esses recursos colaboram ainda mais para o congestionamento da nuvem, sendo favorecidos em contextos que utilizem servidores VEC, como o que foi exemplificado anteriormente na Figura 6 (Hammoud *et al.*, 2020).

A Figura 7 traz um exemplo da diferença da latência existente quando há e quando não há um servidor VEC disponível na borda da rede (Wang *et al.*, 2020)

2.2.2 Comparativo entre VEC e VCC

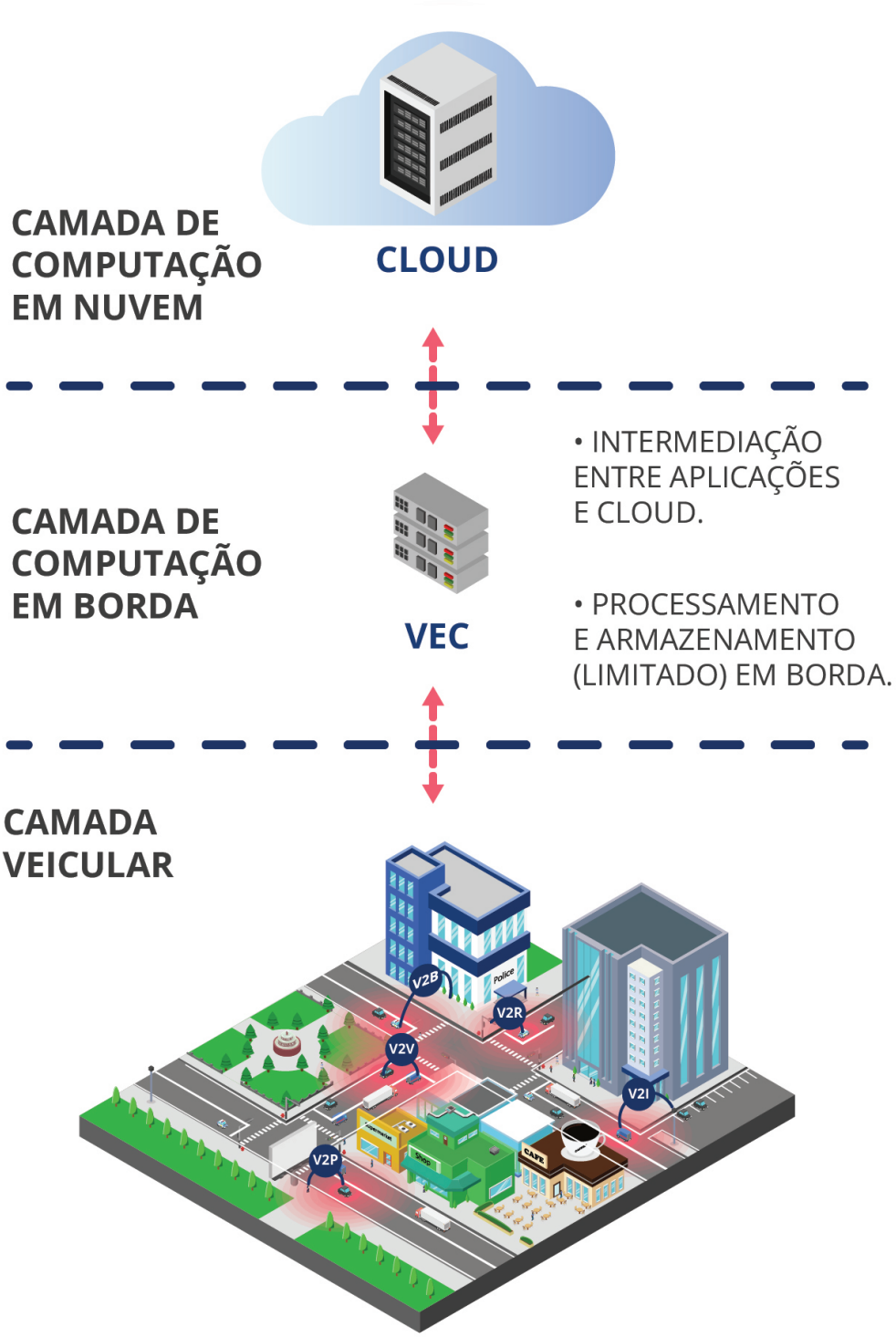
Os paradigmas VEC e VCC são importantes para a resolução de diversos problemas, os quais já foram citados anteriormente. A comparação é importante, mas o que deve ser levado em consideração é o contexto, já que cada um dos paradigmas foi desenvolvido para resolver problemas específicos. Para analisar a paridade da VEC e VCC no contexto dessa monografia, existem três tópicos que devem ser observados com atenção: latência, mobilidade e decisão. (Wang *et al.*, 2020; Meneguette *et al.*, 2021; Raza *et al.*, 2019).

A Tabela 1 apresenta um comparativo entre VEC e VCC, apontando as principais diferenças em relação a diversos tópicos, como localização, latência, mobilidade, decisão, comunicação, armazenamento, consciência, heterogeneidade, processamento e custo:

Em grande parte dos pontos a tecnologia VEC é superior a VCC, perdendo apenas em elementos de armazenamento e processamento.

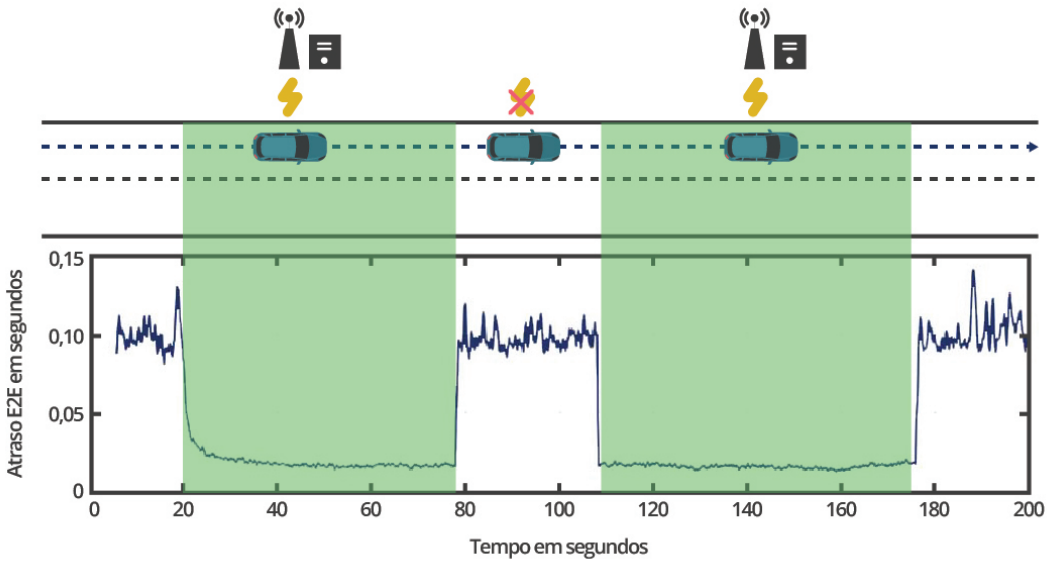
A Figura 8 traz uma comparação entre o *delay*, ou seja, o atraso existente entre as tecnologias VEC e VCC. É importante evidenciar que os dados que são apresentados

Figura 6: Interação entre VEC e CC



Fonte: Elaborado pelo autor, 2021.

Figura 7: Comparação da latência em áreas com e sem servidores VEC



Fonte: Adaptado de (Wang *et al.*, 2020).

Tabela 1: Comparação entre VEC e VCC

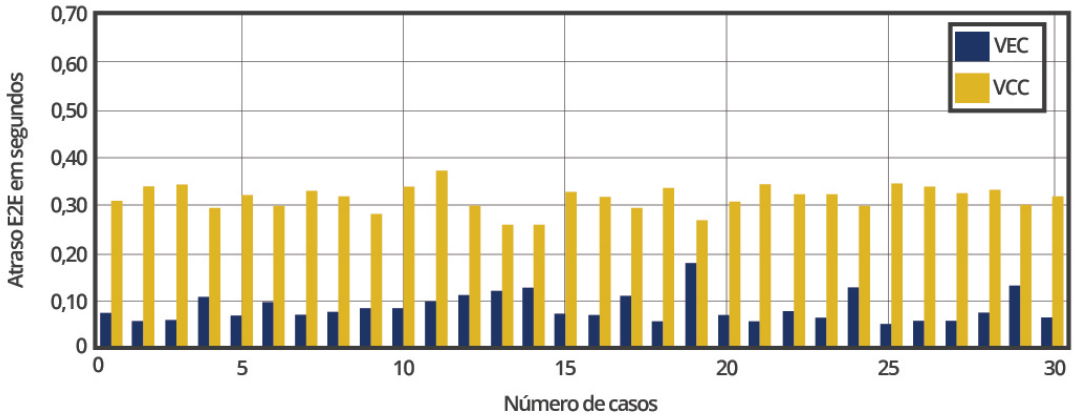
Paradigma:	**VEC**	**VCC**
Localização:	Próximo aos usuários	Remoto aos usuários
Latência:	Baixa	Alta
Mobilidade:	Alto	Limitado
Decisão:	Local	Remoto
Comunicação:	Tempo Real	Restrito a Largura de Banda
Armazenamento:	Limitada	Alta Escalabilidade
Consciência:	Sim	Não
Heterogeneidade:	Suporte Alto	Suporte Limitado
Processamento:	Médio	Alto
Custo:	Baixo	Alto

Fonte: Adaptado de (Raza *et al.*, 2019).

dessa simulação são de fonte de um cenário que possui 30 veículos, diferente da maior parte dos trabalhos que fazem uma simulação individual. Isto quer dizer que o ambiente de simulação está mais próximo da realidade, além disso, a arquitetura de três camadas está relacionada a um conjunto de algoritmos que utilizam diversas abstrações que visam solucionar os desafios de alocação de recursos, como a GT (Wang *et al.*, 2020)

Em suma, a Figura 8 apresenta um esquema gráfico que possui o número de casos,

Figura 8: Comparação da latência entre servidores VEC e VCC



Fonte: Adaptado de (Wang *et al.*, 2020).

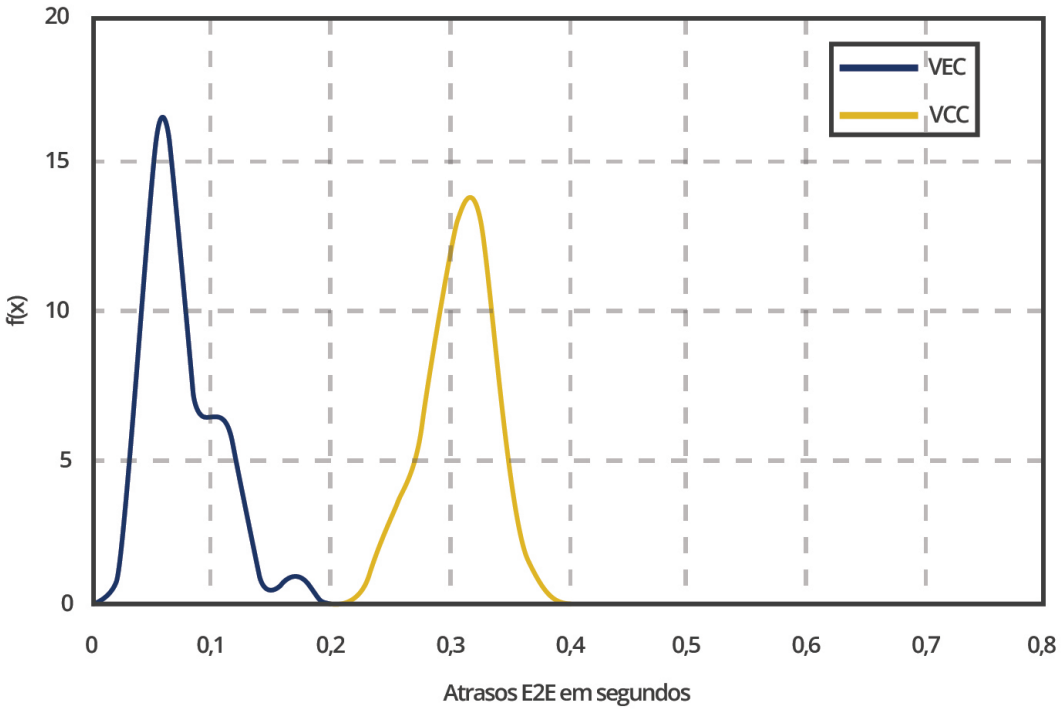
ou seja, a quantidade de veículos que é representado pelo eixo X, além do atraso (*delay*) de ponta-a-ponta (*End-to-End* - E2E) representado pelo eixo Y, descrito em segundos. Ao analisar o gráfico é possível perceber que o caso 19 foi o que teve maior *delay* em relação aos servidores VEC, denotando menos de 0.2 segundos ou mais especificamente 200 milissegundos. Em paralelo a isso, no caso 11 a VCC apresenta um (*delay*) de quase 0.4 segundos ou mais especificamente 400 milissegundos. Dessa forma, os servidores VEC sempre resultaram em um menor atraso em relação aos servidores VCC, que em alguns casos (caso 25, por exemplo) chegam a ser quase oito vezes mais eficiente.

Para detalhar melhor essa eficiência, uma função de densidade de probabilidade de atraso E2E está ilustrada na Figura 9 (Wang *et al.*, 2020)

O eixo X contém o *delay* E2E em segundos, já o eixo Y descreve a $f(x)$. Nesse sentido, como pode ser observado para estratégia VEC há uma variação entre 0 a 0.2 segundos (200 milisegundos), enquanto VCC apresenta uma variação entre 0.2 a 0.4 segundos (400 milisegundos). Dessa forma, através das Figuras 8 e 9 é possível perceber o poder que o paradigma VEC pode oferecer recursos relacionados a mobilidade, garantindo uma entrega de dados em RT e suportando alta estabilidade. Contudo, é importante salientar que ainda há diversos desafios a serem vencidos nesse cenário, e que ainda há uma grande necessidade de um esforço conjunto para a vencê-los (Wang *et al.*, 2020; Meneguette *et al.*, 2021).

Por fim, em busca de uma arquitetura mais robusta, a combinação entre VEC e CC pode ser uma técnica bastante efetiva, devido ao fato de que os servidores VEC podem focar em atender solicitações sensíveis a latência, e os servidores CC podem

Figura 9: Função de densidade da probabilidade do atraso E2E entre VEC e VCC



Fonte: Adaptado de (Wang *et al.*, 2020).

focar em atender solicitações pesadas de aplicativos de entretenimento, que não são sensíveis a atrasos, por exemplo. Esse esquema pode ser feito através de uma Rede Definida por *Software* (*Software-defined Networking* - SDN) centralizada, que possui a capacidade de monitorar toda o estado da rede, possibilitando uma interação entre servidores VEC e CC, através de um *framework* hierárquico de interação. Todo esse conjunto de funcionalidades pode contribuir significativamente para que a abstração de IoV realmente aconteça como planejado (Gu; Zhang; Cao, 2021).

2.2.3 Estratégias de Alocação e Gerenciamento de Recursos em VEC

A tecnologia MEC é extremamente revolucionária, já que ela traz a adaptação do conceito de EC para o contexto mobile, melhorando a latência e habilitando o funcionamento de aplicativos RT. No entanto, gerenciar todas as requisições dos veículos não é uma tarefa muito fácil, pois os servidores devem resolver requisições de processamento, armazenamento, desempenho, entre outros, tudo isso sendo eficiente com o uso de energia. Por isso, alocar recursos em EC é uma questão chave dessa tecnologia, que precisa ser resolvida com urgência, afim de aproveitar as novas possibilidades que

o 5G e o 6G fornecem as redes móveis. (Zhang *et al.*, 2021).

Esse trabalho não é diferente no cenário dos veículos, pois enquanto a função de alocação de recursos é feita pelos servidores VEC, os veículos são os responsáveis pelo CO. São os veículos que vão decidir qual tarefa e qual parte da mesma será transferida para determinado servidor VEC, tudo isso pode ser atestado com base em uma garantia de QoS requisitada pelo veículo, por exemplo.

Por isso, um dos principais desafios existentes é realizar de maneira eficiente a alocação e o gerenciamento de recursos após o CO, em ambientes veiculares, como nas redes IoV, por exemplo. Diversos trabalhos descrevem isso na literatura (Zhang *et al.*, 2021; Meneguette; De Grande; Loureiro, 2018; Liu *et al.*, 2021; Meneguette *et al.*, 2021). Também como já mencionado anteriormente, esse tipo de rede requer estratégias robustas para abstrair todo o tipo de comunicação existente, além de suprir as necessidades dos diversos tipos de aplicativos utilizados, principalmente os que são sensíveis a atrasos (Markakis, 2021).

Entretanto, a arquitetura VEC ainda é considerada bastante emergente, pois necessita de uma boa estratégia para otimização de recursos, já que há o envolvimento de diversos recursos limitados nos servidores VEC em comparação com os servidores VCC, como processamento, armazenamento e comunicação. Por isso, é necessário utilizar algoritmos distribuídos de baixa complexidade para realizar o gerenciamento correto desses recursos, garantindo dessa forma uma entrega de serviço mais eficiente. Dessa forma, os métodos de decisão para alocação de recursos nada mais são do que estratégias compostas de algoritmos, responsáveis por otimizar o gerenciamento de recursos computacionais, como armazenamento e processamento (Zhang, 2022; Dai *et al.*, 2018).

Um exemplo de algoritmo online e distribuído está descrito em (Wang *et al.*, 2020). Em suma, é um algoritmo considerado bastante aplicável para os protocolos de rede que são utilizados atualmente, e o seu funcionamento consiste em adaptar um conceito que faz parte do ramo da matemática, bastante aplicado em modelos econômicos, intitulado de GT. O objetivo principal desse algoritmo é que cada veículo alcance o Equilíbrio de Nash (*Nash Equilibrium* - NE), maximizando assim o resultado para todos os veículos que estão participando da disputa daqueles recursos distribuídos.

Na Seção 2.3 está apresentado um conteúdo mais aprofundado sobre os métodos tradicionais, além dos meta-heurísticos descritos na Seção 2.4.

2.3 Métodos de Decisão para Alocação de Recursos

Nesta seção serão abordados métodos mais tradicionais de alocação de recursos, os quais são intitulados dessa forma devido ao fato de já serem utilizados com frequência na literatura para determinados fins, como no caso de gerenciamento de recursos. Sendo assim, o Item 2.3.1 contém informações detalhadas sobre o funcionamento do método que foi baseado na GT proposta por (Neumann, 1944) no ano de 1944, além do Item 2.3.2 que descreve o funcionamento do método de decisão randômica, (Karp, 1991; S *et al.*, 2021). Os respectivos métodos foram escolhidos de acordo com a relevância que os mesmos possuem na literatura.

2.3.1 Teoria dos Jogos (*Game Theory* - GT)

A GT foi proposta no ano de 1944 por John Von Neumann (Neumann, 1944) em seu trabalho *Theory of Games and Economic Behavior*. Em resumo, um modelo de jogos são um conjunto de ações, jogadores e regras que se desenvolvem em um determinado contexto, em que os jogadores tomam decisões (com base nas regras), as quais podem sofrer influência de uma ação de um jogador anterior ou até mesmo influenciar a ação do próximo jogador. Nesse sentido, a GT é utilizado em contextos em que há necessidade de processos de tomada de decisão (Sajjad; Sharma; Gerdes, 2017).

De acordo com (Bahamou; Ouadghiri; Bonnin, 2016), os componentes principais que podem existir em um jogo completo são:

- **Agentes:** são os jogadores, em um jogo pode é possível ter um jogador ou um grupo de membros, representado por $J = [j_1, j_2, ..., j_n]$.
- ***Payoff*:** É a função útil ou função de recompensa.
- **Equilíbrio:** Estabilização das estratégias.

O *payoff* é o responsável por ditar o nível de satisfação dos agentes, ou seja, a recompensa que o jogador pode obter ao final do jogo. O uso de GT na maior parte dos casos foca na estabilização das estratégias, em outras palavras buscam o equilíbrio, que pode ser garantido através de dois principais elementos, o Equilíbrio de Nash (*Nash Equilibrium* - NE) e o Ótimo de Pareto (*Pareto Optimality* - PO) (Das; Goswami; Konar, 2019).

O NE pode ser explicado da seguinte maneira, cada jogador está fazendo o melhor que pode no contexto de escolha em que o mesmo se encontra, ou seja, nenhum dos

jogadores pode se beneficiar com qualquer tipo de mudança unilateral escolhida. Já o PO, nada mais é do que um resultado em que há tentativa de beneficio de determinado jogador, dessa forma, o resultado será desviado, ocasionando uma perda de satisfação de outro jogador. Nesse sentido, como o próprio nome já diz, o NE é traz um conceito de equilíbrio que é considerado estrategicamente viável, enquanto o PO significa que não existe alguma maneira de beneficiar um jogador sem prejudicar o outro, o que pode ser considerado eficiente de certa forma.

O Dilema dos Prisioneiros (*Prisioner's Dilemma* - PD) é uma representação visual bastante utilizada para explicar os conceitos de GT. Em resumo o dilema pode ser descrito da seguinte forma: dois prisioneiros foram presos, eles devem trair um ao outro ou manter a cooperação? Ambos são colocados em selas separadas, pois não existem provas suficientes para condená-los. Um acordo é oferecido para os dois e traz as seguintes possibilidades:

1. Se um prisioneiro confessar o crime e o outro ficar em silêncio, quem confessa saí e quem fica em silêncio é condenado a dez anos de prisão;

2. Se ambos os prisioneiros ficarem em silêncio, a polícia só poderá condená-los a um ano de prisão;

3. Se ambos os prisioneiros confessarem, cada um ficará cinco anos na prisão.

Basicamente a regra do jogo é: Cada prisioneiro toma a decisão sem saber da escolha do outro. Dessa forma, alguns questionamentos são levantados:

- Como os prisioneiros vão reagir?

- Qual seria a decisão mais racional a ser tomada?

A matriz de resultados ilustrada na Figura 10 apresenta uma forma esquemática, que tem o objetivo de mostrar as interações do jogo

O objetivo dos jogadores é ficar o menor tempo preso possível, isto quer dizer que eles precisam maximizar os seus resultados, tomando uma melhor decisão. A melhor decisão nesse cenário é trair (contrária a intuição). Esta decisão pode ser considerando as duas hipóteses que o prisioneiro A (ou B) deve levar em consideração, por exemplo:

- **Hipótese 1: O prisioneiro B** escolhe **colaborar**:

  A : Se o **prisioneiro A colaborar**, ambos pegam **1 (um) ano** de prisão.

Figura 10: Matriz de resultados: Dilema do Prisioneiro

Fonte: Elaborado pelo autor, 2021.

B : Se o **prisioneiro A trair**, é **solto** e o **prisioneiro B** pega **10 (dez) anos** de prisão.

- **Hipótese 2: O prisioneiro B** escolhe **trair**:

  A : Se o **prisioneiro A colaborar**, pega **10 (dez) anos** e o **prisioneiro B** é **solto**.

  B : Se o **prisioneiro A trair**, ambos pegam **5 (cinco) anos** de prisão.

Dessa forma, a estratégia dominante é "trair", ou seja, a decisão que apresenta o melhor resultado, independentemente da decisão do outro jogador. Isto quer dizer que se em um jogo, for possível escolher uma determinada decisão sem se preocupar com a decisão do outro jogador, a estratégia dominando sempre deverá ser escolhida com o objetivo de maximizar os resultados. Sendo assim, como a estratégia dominante no dilema dos prisioneiros é trair ambos vão pegar 5 (cinco) anos de prisão, ou seja, este é o NE, explicado anteriormente. Nesse caso, o NE não é o melhor resultado para

nenhum dos jogadores, por isso é descrito como um esquema de equilíbrio ineficiente, já que induzem a um resultado pior.

Esse cenário pode ser utilizado em diversas áreas, como no contexto de alocação de recursos em VEC.

Um conjunto de recursos pode ser representado por $R = [sto_i, proc_i, mem_i]$:

- sto_i: Armazenamento
- $proc_i$: Processamento
- mem_i: Memória

Os serviços *S* possuem dois *status*, atendidos e negados e funcionam como jogadores nesse contexto, ou seja, cada serviço é um jogador que foi solicitado pelo usuário. Esses serviços podem ser diversos aplicativos ITS.

Considerando um jogo *J* composto pelos jogadores *A* e *B*, sendo que cada jogador possui um conjunto de estratégias, que pode ser "Atendido"ou "Bloqueado", é possível considerar que o NE nesse caso pode ser encontrado quando ambos os serviços são negados, já quando ambos o serviços são atendidos, é encontrado o PO.

2.3.2 Decisão Aleatória Simples (*Simple Random Decision* - SRD)

De acordo com (Karp, 1991), algoritmos randômicos, ou seja, aleatórios são bastante utilizados na computação, para diversos fins, seja para teoria dos números, álgebra, grafos, seleção, pesquisa, geometria entre outros.

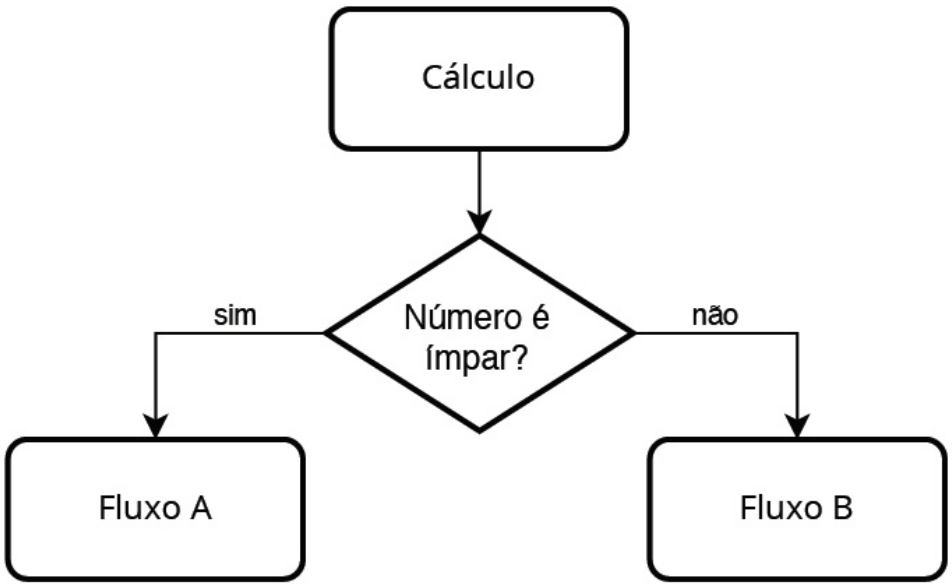
No contexto de alocação de recursos não é diferente, a aleatoriedade pode ser utilizada para a tomada de decisão. Em (S *et al.*, 2021) é citado o algoritmo Floresta Aleatória (*Random Forest* - RF) que utilizada conceitos de aprendizado de máquina, produzindo resultados mais robustos e menos propensos a *overfitting*, que são ajustes excessivos, que acontece por exemplo com o algoritmo de Árvore de Decisão (*Decision Tree* - DT)

Os mesmos também podem receber *input* de dados para ser utilizados como parâmetros em uma operação específica contribuir na geração de um resultado randômico (Karp, 1991).

A Figura 11 demonstra um exemplo de fluxo de tomada de decisão de acordo com o resultado da aleatoriedade

Em resumo, a Figura 11 traz um esquema de fluxo, que tem a sua decisão tomada pela aleatoriedade, que como já foi exemplificado nesta seção, esse cálculo pode ser feito de diversas maneiras, com *input* de dados ou não, de acordo com a abordagem

Figura 11: Fluxo de Decisão com Algoritmo Aleatório



Fonte: Elaborado pelo autor, 2022.

utilizada. Neste exemplo, um algoritmo simples de decisão utiliza uma decisão extremamente simples, por meio do resultado do sorteio de números do intervalo de 0 a 9, se o número for ímpar o fluxo segue em A, se o número for par o fluxo é seguido em B.

2.4 Métodos Meta-heurísticos de Decisão para Alocação de Recursos

Em paralelo aos métodos mais tradicionais que foram apresentados na Seção 2.3 também existem as soluções de algoritmos denominadas meta-heurísticos. De acordo com (Jha; Jain; Thenmalar, 2018; Katoch; Chauhan; Kumar, 2020; Kumar *et al.*, 2020), os métodos meta-heurísticos podem ser descritos como abordagem baseadas na natureza, ou seja, bio-inspirados e podem ser utilizados em diversos campos de pesquisa, como no campo da computação já que são considerados otimizados para a resolução de problemas específicos.

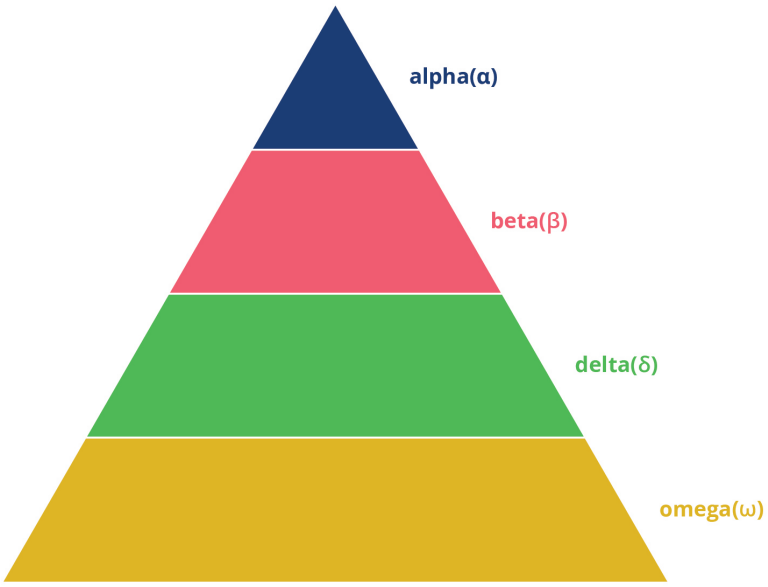
Nesse sentido, está seção apresenta dois diferentes métodos meta-heurísticos que

são utilizados para a resolução de problemas de gerenciamento de recursos, sendo o Método de Otimização do Lobo Cinzento (*Grey Wolf Optmizer* - GWO) no Item 2.4.1 e o Algoritmo do Morcego (*Bat Algorithm* - BAT) no Item 2.4.2. Assim como na Seção 2.3, os respectivos métodos também foram escolhidos de acordo com a relevância que os mesmos possuem na literatura.

2.4.1 Método de Otimização do Lobo Cinzento (*Grey Wolf Optimizer* - GWO)

Os lobos cinzentos são animais extraordinários que costumam viver em grupos, formando uma alcateia que tem em média 5 a 12 animais. Além disso, eles vivem em uma espécie de hierarquia que possui quatro níveis, ou seja, cada um dos animais estão classificados em níveis: *alpha*, *beta*, *delta* e *omega* a qual pode ser ilustrada através da Figura 12 (Mirjalili; Mirjalili; Lewis, 2014)

Figura 12: Hierarquia dos lobos cinzentos



Fonte: Elaborado pelo autor, 2021.

As funções de cada uma das hierarquias estão descritas em ordem decrescente, ou seja, de maior para menor relevância:

- *alpha* (α)**:** Primeiro nível, líder da alcateia, responsável por tomar todas as decisões do grupo, inclusive as de decisões relativas a caça de uma determinada presa;

- *beta* (β)**:** Segundo nível, auxilia o líder nas tomadas de decisões da alcateia;
- *delta* (δ)**:** Terceiro nível, lida com a segurança da alcateia, reportando os dois primeiros níveis e protegendo os lobos de nível inferior (ω);
- *omega* (ω)**:** Todos os demais lobos da alcateia ficam no nível mais inferior da hierarquia.

Ainda além disso, a alcateia possui um comportamento de caça extremamente organizado, que está dividido em 3 etapas:

1ª Etapa: A alcateia começa rastreando, perseguindo e se aproximando da presa;

2ª Etapa: A alcateia cerca a presa, até a mesma ficar imóvel;

3ª Etapa: A alcateia finalmente ataca a presa.

Esses comportamentos são considerados tão interessantes que inspiraram o método GWO, conhecido por compartilhar os conceitos da técnica de otimização intitulada algoritmo lobo cinzento. O funcionamento deste algoritmo pode ser explicado da seguinte maneira: Em um conjunto de tarefas, existe uma hierarquia em que α é a melhor tarefa, ou seja, melhor solução, seguida pela segundo melhor β, seguida pela terceira melhor δ e por fim, os restantes das tarefas são considerados soluções ω. Dessa forma, a caça é guiada pelas soluções α, β e δ (Mirjalili; Mirjalili; Lewis, 2014).

O processo de movimentação dos lobos durante a realização do cerco, pode ser observado na Figura 13

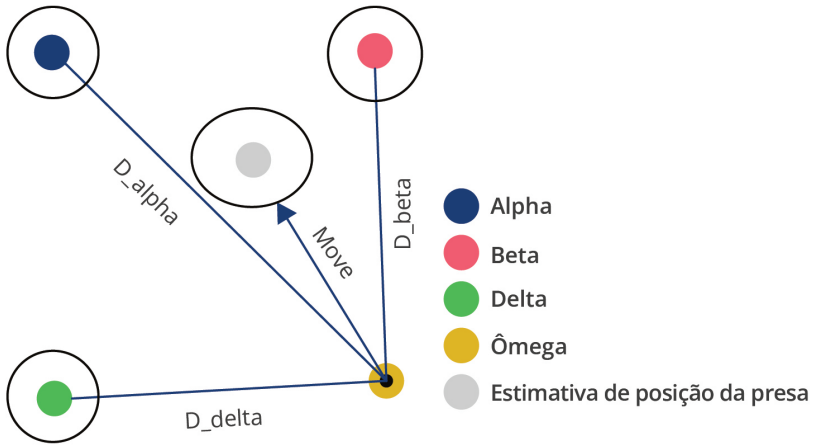
A presa é representada pela posição (X*, Y*), a qual é utilizada como referência para a mudança de posicionamento dos lobo (X, Y) demonstrado na Figura 14. A atualização dos coeficientes são definidos em valores aleatórios de 0 e 1, com o objetivo de alcançar a melhor posição (Mirjalili; Mirjalili; Lewis, 2014)

Após o cerco, há o reposicionamento guiado pelo lobo α sendo que β e δ também participam do processo. Não há como simular matematicamente o comportamento da caça, devido a ao fato de que o posicionamento ótimo da presa normalmente não é informado (Mirjalili; Mirjalili; Lewis, 2014).

2.4.2 Algoritmo do Morcego (*Bat Algorithm* - BAT)

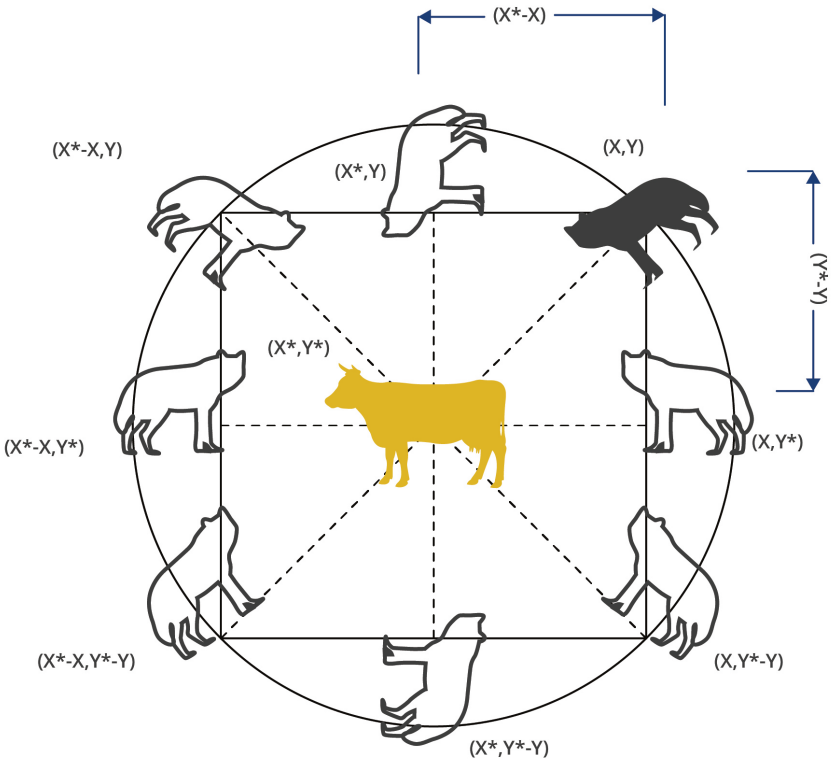
O algoritmo BAT é um tipo de algoritmo meta-heurístico, que se baseia nos padrões de comportamento de caça dos micro-morcegos. Esses animais possuem uma espécie de sonar, mais especificamente denominado eco localização, recurso utilizado para se

Figura 13: Representação do padrão de caça dos lobos cinzentos



Fonte: Adaptado de (Mirjalili; Mirjalili; Lewis, 2014).

Figura 14: Vetor 2D com as possibilidades das próximas posições



Fonte: Adaptado de (Mirjalili; Mirjalili; Lewis, 2014).

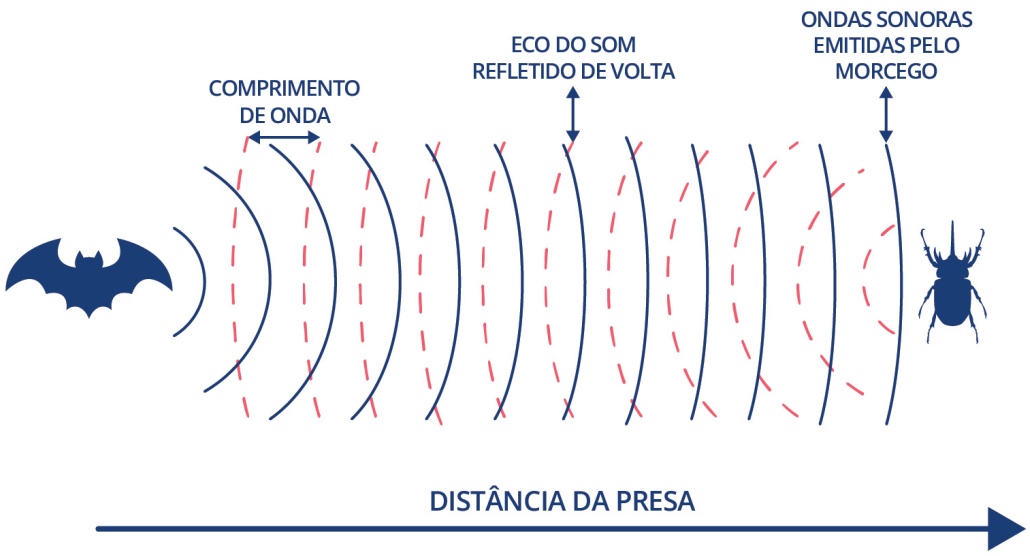
alimentarem, já que são uma espécie insetívora, em outras palavras são animais que se alimentam de insetos. Esse mecanismo não serve somente para encontrar as suas

presas, mas funcionam como localizadores, sendo responsável por guiar os animais em ambientes noturnos (Yang, 2010).

Ainda de acordo com (Yang, 2010), a eco-localização funciona da seguinte maneira, os micro-morcegos emitem ondas de pulsos sonoros extremamente altas, as quais retornam para os mesmos caso haja algum tipo de superfície ou obstáculo, por exemplo. Normalmente a frequência fica em uma média de 10 a 20 pulsos por segundos, mas em momentos de caças podem chegar até 200 pulsos por segundo. Mesmo não seguindo um padrão específico, os micro-morcegos possui dois *modus operandi*, o primeiro é o reconhecimento do ambiente e identificação da presa, já o segundo é o avanço em direção a presa, que significa o ataque.

A Figura 15 apresenta uma simulação de como funciona esse efeito

Figura 15: Efeito "sonar"dos micro-morcegos



Fonte: Adaptado de (Srivastava; Sahana, 2019).

Os fatores mais importantes que precisam ser levados em consideração para encontrar a melhor solução para este algoritmo são (Yang, 2010; Yang, 2013):

- Movimento (m_i): O único padrão que os morcegos seguem é o citado anteriormente, já que diferente de outras espécies eles voam de forma aleatória;
- Velocidade (v_i): A posição ou soluções encontradas definem a velocidade;
- Volume do Pulso (a_i): Assim como a Velocidade, o Volume do Pulso também muda quando a posição ou soluções são alteradas, ou seja, em situação padrão é menor, em situação de caça é maior;

- Emissão Pulso (r_i): A emissão do pulso aumenta quando o morcego se aproxima da presa; e
- Frequência (f_i): A frequência varia de 25kHz a 150kHz.

2.5 Considerações Parciais

Os conceitos de VEC surgiram por diversos motivos, sendo os principais deles em razão das novas gerações de redes móveis 5G e 6G, que respectivamente trouxeram e prometeram inúmeros recursos e funcionalidades. Outro fator importante é a conexão cada vez mais rápida, possibilitando a implementação de funcionalidades extremamente sensíveis a atrasos em aplicativos ITS. Em resumo, isso torna o uso da Computação em Nuvem (*Cloud Computing* - CC) isolada inviável nesses casos.

Entretanto, diferente da CC a VEC não possui recursos computacionais "ilimitados", como processamento, memória e armazenamento. Por isso, é essencial que estratégias relacionadas a arquitetura e gerenciamento de recursos sejam aplicadas nesses contextos, com o objetivo de definir responsabilidades e agregar o que as tecnologias possuem de melhor.

As Seções 2.3 e 2.4 apresentam exemplos de algoritmos de diferentes classificações que podem ser utilizados para o contexto de decisão de alocação de recursos. Contudo, a GT se destaca, por apresentar uma solução mais flexível para este fim, de acordo com as inúmeras variáveis existentes no contexto de alocação de recursos em ambientes de redes IoV.

Nesse sentido, os algoritmos tradicionais que são baseados na GT, podem ser utilizados em contextos de decisão de alocação e gerenciamento de recurso em computação de borda, já que eles visam maximizar os resultados, através de algumas teorias, como NE e PO. Portanto, é extremamente viável utilizar GT para decidir quais são os recursos sensíveis a latência que precisam ser atendidos pela VEC e quais são os recursos não sensíveis a latência que podem ser atendidos em ambientes de computação centralizado, ou seja, nuvem.

Por fim, é possível que o uso de arquitetura VEC de três camadas com algoritmos GT para gerenciamento de recursos seja benéfico para resolver problemas de latência, mobilidade e heterogeneidade, principalmente para o contexto das redes de IoV, que herda os conceitos do paradigma IoT, aplicando-os em ambientes veiculares, trazendo a possibilidade da interação V2X.

Capítulo 3

Revisão Sistemática da Literatura

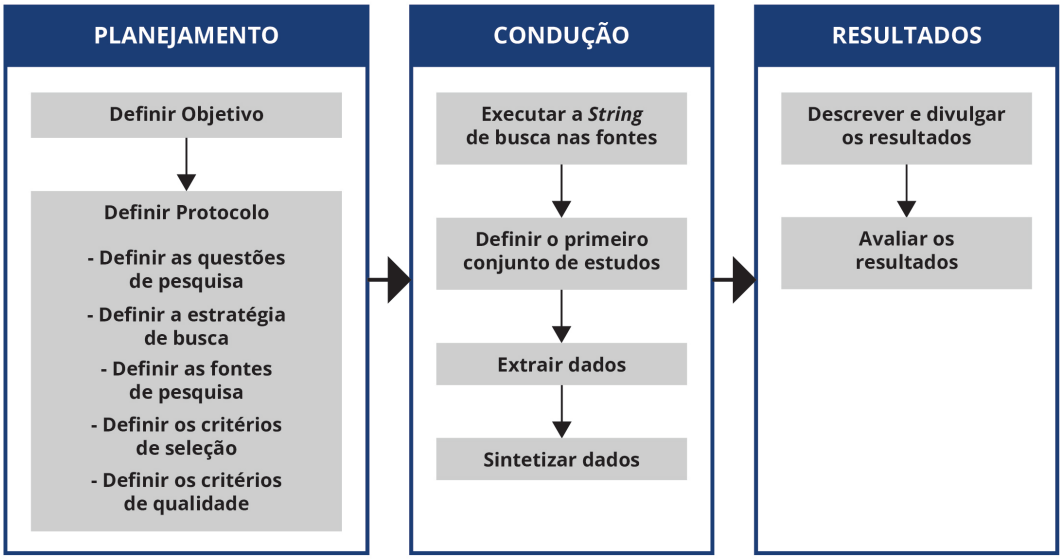
Este capítulo possui o objetivo de documentar todo o protocolo de pesquisa que foi realizado para buscar os trabalhos relacionados na literatura. Um modelo de Revisão Sistemática da Literatura (*Systematic Literature Review* - SLR), elaborado com base nas diretrizes desenvolvidas por Barbara Kitchenham, foi proposto para definir a seleção dos estudos descobertos. Esse processo é feito por meio de um protocolo que contém questionamentos, métricas e métodos pertinentes ao propósito de encontrar uma maneira mais assertiva de definir a qualidade dos trabalhos que foram selecionados, garantindo assim uma maior imparcialidade deste processo, por exemplo.

Nesse sentido, o capítulo está dividido em três Seções, sendo que na Seção 3.2 são apresentados os estudos relacionados disponíveis na literatura, ou seja, há um estrutura detalhada sobre os assuntos referentes ao contexto deste trabalho; Na Seção 3.1 está definido o núcleo do processo de revisão, explorando os questionamentos, estratégias de pesquisa, processos de seleção dos estudos, método de extração dos dados, e por último, as considerações parciais. Por fim, na Seção 3.3 estão descritos as prováveis ameaças existentes na pesquisa e as possíveis limitações que o processo executado nesta SLR não conseguiu abranger, além dos principais desafios que são enfrentados em relação ao tema abordado.

3.1 Protocolo da Pesquisa

Esta seção descreve o protocolo que foi utilizado para conduzir todo processo SRL, totalmente baseado no protocolo Kitchenham (Kitchenham, 2004). As etapas deste componente referem-se aos itens existentes para que a pesquisa seja realizada de forma eficiente. A Figura 16 traz uma visão geral do protocolo que será descrito ao longo deste capítulo.

Figura 16: Protocolo de Revisão Sistemática



Fonte: Elaborado pelo autor, 2022.

Em resumo, esses itens abordam as estratégias de busca que foram utilizadas para a obtenção dos estudos, o processo de seleção que descreve os critérios avaliativos da qualidade dos trabalhos selecionados, além da metodologia de extração dos dados que foi utilizada, e por fim, as considerações parciais obtidas ao final do desenvolvimento do protocolo.

3.1.1 Questões de Pesquisa (*Research Questions* - RQs)

As RQs possuem o objetivo de direcionar a procura das respostas corretas, através de questionamentos que são pertinentes a este trabalho, que consequentemente contribuem para que a investigação seja feita de forma mais assertiva. Dessa forma, as RQs focam principalmente nos problemas, estratégias, soluções existentes, benefícios e desafios, e o cenário futuro dentro do contexto do trabalho, que envolve a proposta que buscar uma solução para maximizar o Descarregamento de Computação (*Computing Offloading* - CO), além da decisão de alocação e gerenciamento de recursos através de algoritmos, em especial Teoria dos Jogos (Game Theory, GT) à partir de uma arquitetura de Computação em Borda Veicular (*Vehicular Edge Computing* - VEC) que seja mais eficiente para redes de Internet dos Veículos (*Internet of Vehicles* - IoV).

As RQs são:

- **RQ1:** Quais os principais problemas de gerenciamento de CO e recursos em ambientes VEC?

- **RQ2:** Quais as principais soluções existentes na literatura em relação a algoritmos de CO e decisão de alocação de recursos em ambientes VEC?
- **RQ3:** Quais os tipos de algoritmos em GT dos jogos se destaca entre essas soluções?
- **RQ4:** Quais os benefícios do uso de VEC em redes IoV?
- **RQ5:** Quais os desafios de integrar VEC em redes IoV?
- **RQ6:** O que esperar no futuro do paradigma VEC e das redes IoV?

Esses questionamentos contribuem principalmente para a etapa inicial da pesquisa, pois auxiliam no processo introdutório da procura de trabalhos na literatura, os quais carregam um forte conteúdo para a contextualização e apresentação das principais tendências relacionados aos assuntos, apresentados no Capítulo 2.

3.1.2 Estratégias de Busca

Esta seção define a etapa de estratégias de buscas do procolo de pesquisa da SLR. Um conjunto de passo a passo foi estabelecido a partir das RQs definidas anteriormente no Item 3.1.1, a qual foi responsável por selecionar um primeiro conjunto de trabalhos base, utilizados no Capítulo 2. Esse passo a passo está dividido em ordem linear, composto por:

- Definição das palavras-chave;
- Definição da *string* de busca;
- Definição das bases para as buscas;

A definição das palavras-chave foi criada à partir dos estudos base, sendo descritas em ordem de relevância:

1. Alocação de Recursos (*Resource Allocation*);
2. Teoria dos Jogos (*Game Theory*);
3. Computação em Borda Veicular (*Vehicular Edge Computing*);
4. Descarregamento de Computação (*Computing Offloading*)
5. Internet dos Veículos (*Internet of Vehicles*);

6. V2X (*Vehicle-to-everything*);

Com as palavras-chave definidas é hora de trabalhar na construção da *string* de busca. Para realizar este processo as palavras-chave foram organizadas com as suas respectivas variações e abreviações:

*((**Resource Allocation**) AND (**Game Theory**) AND (**Vehicular Edge Computing** OR **VEC**) AND (**Computing Offloading** OR **CO**) AND (**Internet of Vehicles** OR **IoV** OR **V2X**))*

A última etapa do processo consiste em definir as bibliotecas digitais, ou seja, as plataformas de busca que foram utilizadas para conduzir a pesquisa. A *string* de busca foi aplicada nos seguintes sistemas:

1. **ACM Digital Library**[1]
2. **IEEExplorer Digital Library**[2]
3. **Scopus**[3]
4. **Springer Link**[4]
5. **Web Of Science**[5]
6. **Outras**[6]

Para ser possível utilizar a *string* como entrada, foi preciso realizar a busca por meio da pesquisa avançada, disponível na maioria das bibliotecas digitais. Além disso, para que o resultado fosse ainda mais preciso, outros parâmetros de filtragem também foram adicionados, como listagem dos trabalhos por ordem de relevância, produções científicas feitas nos últimos cinco anos, resultados encontrados no título e no resumo e até seleções de categorias de pesquisa. É importante salientar que cada biblioteca digital possui a sua particularidade quanto a esses filtros citados anteriormente, e por isso, a *string* de busca foi adaptada com os filtros genéricos, ou seja, que existiam em todas as bases digitais, com o objetivo de manter a maior coerência possível nos resultados gerados. Por fim, para obter alguns outros estudos relevantes que foram publicados em outras bibliotecas digitais, a *string* de busca também foi utilizada no

Google Scholar, por isso, essa fonte foi classificada como "outros", já que engloba uma quantidade pequena de estudos obtidas em diversas bibliotecas digitais.

3.1.3 Processo de Seleção

O processo de seleção é considerado uma etapa extremamente importante, por isso é necessário que essa fase seja feita da forma mais imparcial possível. O processo está dividido em duas fases, além de uma etapa de pré-seleção, a qual advêm dos resultados da *string* de buscas, com todos os parâmetros definidos, ou seja, filtros de intervalo de período, por exemplo. Além disso, alguns critérios avaliativos auxiliam no processo de seleção, sendo estes definidos em conjunto com o orientador.

A Tabela 2 contém a lista de critérios de inclusão e exclusão, que faz parte dessa etapa de pré-seleção:

Tabela 2: Critérios de Inclusão e Exclusão

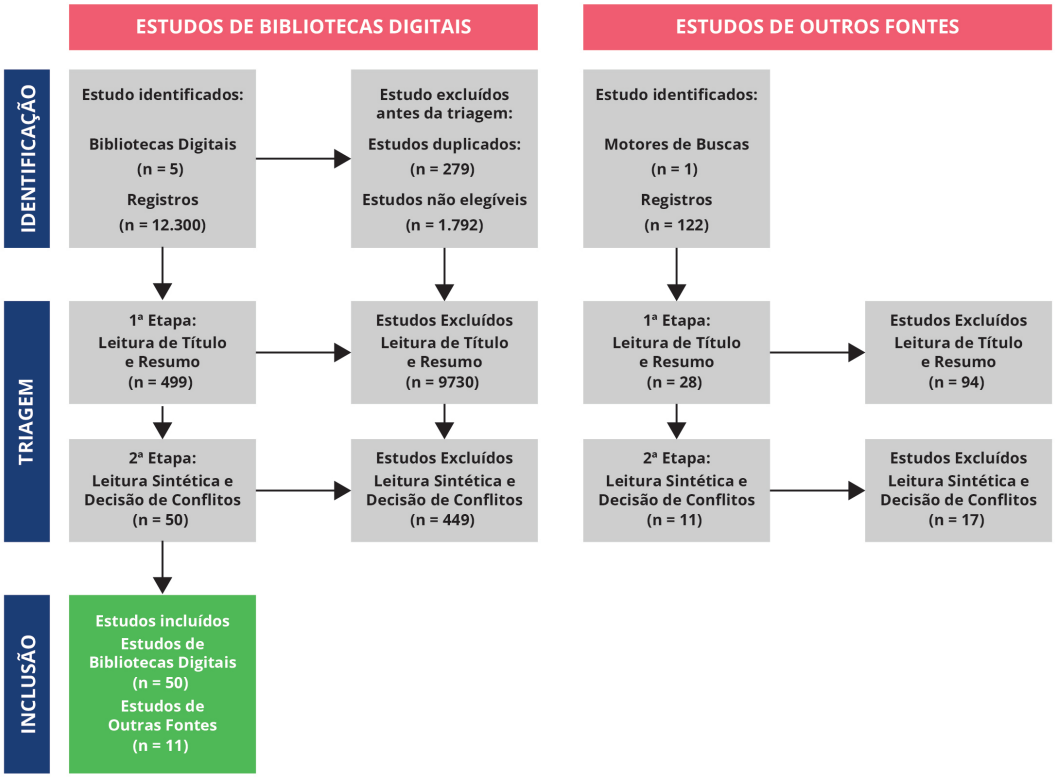
Critérios de Inclusão	**Critérios de Exclusão**
1. O estudo está no contexto das palavras-chave da *string* de busca.	1. O estudo até está no contexto mas foge das questões principais.
2. O estudo fornece resposta a pelo menos uma das perguntas.	2. O estudo não fornece resposta a nenhuma das perguntas.
3. O estudo trata sobre Alocação de Recursos ou CO em servidores MEC.	3. O estudo foca apenas no contexto de CO.
4. O estudo propõem algoritmos que usam a abordagem GT.	4. O estudo não propõe algoritmo ou utiliza métodos meta-heurísticos.
5. O estudo apresenta uma metodologia completa.	5. O estudo não foi escrito em inglês.
6. O estudo apresenta resultados comprovando sua eficiência.	6. O estudo é totalmente teórico ou não apresenta dados.

Fonte: Elaborado pelo autor, 2021.

Após o processo de inclusão que engloba todas as bibliotecas digitais citadas anteriormente, os trabalhos são listados em uma planilha, e posteriormente os trabalhos duplicados são excluídos e o processo de triagem se inicia, como pode ser observado na Figura 17

[1]https://dl.acm.org/
[2]https://ieeexplore.ieee.org/
[3]https://www.scopus.com/
[4]https://link.springer.com/
[5]https://www.webofscience.com/
[6]https://scholar.google.com/

Figura 17: Condução de Seleção dos Estudos



Fonte: Elaborado pelo autor, 2022.

No início do fluxo o objetivo é mostrar a relação entre os estudos encontrados pela *string* de busca pela quantidade dos estudos que foram pré-selecionados para a primeira fase de seleção:

A primeira fase de seleção possui o objetivo de separar os trabalhos que vão contribuir para a pesquisa. A metodologia utilizada para isso foi a leitura do título, já que são dois itens capazes de contextualizar bem o trabalho. O processo também está representado na Figura 17, identificado pelo fluxo da "1ª - Etapa".

Já a segunda e última fase de seleção possui uma maior complexidade, sendo necessário a ajuda de outro pesquisador para avaliar esse conjunto de estudos. Agora, é preciso ler outros capítulos dos estudos, os que são mais pertinentes, como metodologia, resultado, conclusão, entre outros.

Por fim, após a etapa de pré-seleção, a primeira e a segunda etapa de seleção, que são responsáveis por analisar e validar os artigos pertinentes a este trabalho, resultaram em 61 estudos selecionados, sendo que a Tabela 3 auxilia na demonstração da quantidade específica que foi selecionada em cada biblioteca digital:

Esses estudos foram identificados como de extrema importância para conduzir toda

Tabela 3: Número de Estudos Selecionados na Última Etapa

Bibliotecas Digitais	**Documentos Selecionados**
ACM Digital Library	9
IEEE Xplore	26
Springer Link	10
Scopus	3
Web of Science	2
Outras Fontes	11
Total:	**61**

Fonte: Elaborado pelo autor, 2021.

a pesquisa desta monografia, já que os mesmos passaram por todos os critérios de seleção definidos, contendo informações relevantes, relacionadas não só ao foco que é metodologia e resultado, mas também outras abordagens, como a conceituação de diversos itens utilizados no contexto do trabalho.

3.1.4 Extração de Dados

O processo de extração de dados foi executado por apenas um pesquisador, o qual considerou diversos aspectos considerados relevantes para este trabalho. Basicamente, esta etapa consiste em uma leitura detalhada dos trabalhos que são considerados chave, por exemplo os descritos na Seção 3.2. Isto quer dizer que esses trabalhos de alguma forma possuem tópicos semelhantes em relação aos assuntos desta pesquisa, como teoria dos jogos, alocação de recursos, VEC, CO, IoV, entre outros. A orientação para a condução deste processo está documentada na sequência:

- **Aspectos Bibliométricos:** Respectivamente em ordem de importância o ano, os autores e as instituições.
- **Aspectos Teóricos:** O tipo de estudo, as motivações, justificativas e os objetivos.
- **Aspectos Metodológicos:** Abordagens, ferramentas, tecnologias e os tipos de simulações utilizadas.
- **Questões de Pesquisa:** As 6 Questões de Pesquisa que estão descritas no Item 3.1.1.

Por último, a partir destes aspectos citados acima, os artigos foram analisados para que o processo de extração fosse realizado. Em suma, pode-se considerar que esses

dados foram selecionados e categorizados em planilhas, com o objetivo de centralizar esses materiais para facilitar o acesso, para fins de citação e análise comparativa, por exemplo.

3.1.5 Considerações Parciais

O protocolo de pesquisa definido para a condução desta SLR possui o objetivo de documentar todo este processo burocrático, que se inicia nas questões que são pertinentes a esta pesquisa, e vai para as estratégias de buscas que foram utilizadas, ou seja, a formação da *string* de busca através das definições das palavras-chave, além de toda a abstração da complexidade dos processos de seleção e extração dos dados.

Dessa forma, o protocolo desenvolvido para esta SLR foi extremamente importante em diversos aspectos, desde os que estão intrinsecamente relacionados a esta pesquisa, até mesmo para quesitos que tratam da organização do processo, por exemplo. Esse modelo sistemático aplicado para a realização das pesquisas nas bibliotecas digitais contribuiu fortemente para garantir a qualidade e a importância dos estudos para este trabalho, de acordo com algumas questões e critérios previamente definidos.

3.2 Estado da Arte

Nesta seção, são apresentados trabalhos que foram obtidos na literatura através do protocolo de SRL, descrito na Seção 3.1. Em resumo, estes trabalhos possuem relação ao contexto desta pesquisa, já que exploram diversos assuntos relacionados, como o uso de algoritmos tradicionais baseado na GT para a alocação e gerenciamento de recursos computacionais e CO, além das aplicações de ambientes de Computação em Borda Móvel (*Mobile Edge Computing* - MEC), as interações em redes IoV ou comunicações que fazem parte deste universo como o conceito da interação V2X.

A maior parte dos trabalhos focam em algoritmos baseados em GT que buscam o Equilíbrio de Nash (*Nash Equilibrium* - NE) (Mensah *et al.*, 2021; Wang *et al.*, 2021; Zhang; Wang; Liu, 2020; Hu *et al.*, 2020; Zhou; Jadoon, 2020). Entretanto, (Zafari *et al.*, 2018; Bui; Jung, 2017; Zhao; Lu; Zhiying, 2020) apresentam soluções de GT com jogos cooperativos, e (Shen *et al.*, 2020) mostra o uso do Jogo da Minoria (*Minority Game* - MG). Já (Zeng *et al.*, 2021; Tang *et al.*, 2020) e (Araldo; Stefano, 2020; Avgeris *et al.*, 2019; Meneguette *et al.*, 2021) respectivamente apresentam o modelo de jogo *Stackelberg* e soluções genéricas relacionadas a Computação em Borda (*Edge Computing* - EC).

Em (Mensah *et al.*, 2021) está descrito o uso de GT que busca o NE em interações Dispositivos para Dispositivos (*Device-to-Device* - D2D) com o uso de Computação em Borda Veicular (*Vehicular Edge Computing* - VEC) para aumentar a capacidade da Computação em Nuvem Veicular (*Vehicular Cloud Computing* - VCC). (Wang *et al.*, 2021) traz um algoritmo de NE para CO e alocação de recursos em interações V2X em contextos MEC com o objetivo de reduzir a latência e o consumo de energia. Para amenizar o problema de CO e gerenciamento de recursos em interações V2X, muito comum em IoV, (Zhang; Wang; Liu, 2020) propõe uma arquitetura de rede MEC assistida por Rede Definida por Software (*Software-Defined Networking* - SDN) com algoritmo que busca o NE para melhorar a eficiência e flexibilidade da rede. O NE também é usado no trabalho de (Hu *et al.*, 2020), que abrange o contexto de MEC para resolver problemas de CO e alocação de recursos. Por último, em servidores MEC que possuem acessos múltiplos, um algoritmo que busca NE também pode colaborar em solucionar CO e alocação de recursos (Zhou; Jadoon, 2020).

(Zafari *et al.*, 2018) sugere um algoritmo de complexidade O(N) baseado em GT para CO e decisão de alocação de recursos, que resultam em alocações de ótimas de Pareto em provedores de serviço de EC. A pesquisa de (Bui; Jung, 2017) apresenta soluções de jogos cooperativos que visam colaborar para a redução das interferências existentes nas redes através de alocação distribuída de recursos. Os modelos de jogos coordenados, cooperativos, recompensa e punição baseados em GT são descritos em (Zhao; Lu; Zhiying, 2020) através de um Algoritmo de Otimização da Colônia de Formigas (*Ant Colony Optimization* - ACO) para resolução do problema de alocação de tarefas e detecção de múltiplos sensores. Para enfrentar os desafios de solicitações excessivas de serviços e manter os fatores de Qualidade de Serviço (*Quality of Service* - QoS) e Qualidade de Experiência (*Quality of Experience* - QoE) (Shen *et al.*, 2020) recomenda um modelo de Jogo de Minoria (*Minority Game* - MG) denominado DOM que contribui a EC de dispositivos móveis, sendo benéfico para redes IoV.

Com o objetivo de melhorar a utilidade de servidores VEC, reduzindo o custo e a locação de CO/recursos, um algoritmo de busca rápida para atribuição de tarefas baseado em GT é proposto em (Zeng *et al.*, 2021), o qual usa modelo de jogos *Stackelberg*. Em (Tang *et al.*, 2020) um modelo de jogos *Stackelberg* também é proposto para modelar as interações existentes entre os veículos e os servidores MEC, com o foco em melhorar a eficiência em relação ao custo deste processo.

Por fim, os trabalhos de (Araldo; Stefano, 2020; Avgeris *et al.*, 2019; Meneguette *et al.*, 2021) propõem estratégias mais genéricas para a resolução dos problemas de CO e gerenciamento de recursos em ambientes EC. (Araldo; Stefano, 2020) apresenta

uma solução intitulada MORA, uma estratégia de tempo polinomial para melhoria de diversas métricas de interesse, como QoS e QoE. Já (Avgeris *et al.*, 2019) apresenta uma opção alternativa de execução de tarefas é disponibilizada aos usuários de dispositivos móveis, através de uma alocação de recursos em dois níveis e um mecanismo de controle de admissão, realizados em um *cluster* de servidores EC. Um trabalho mais abrangente que descreve conceitos, tecnologias e uma visão geral das arquiteturas VEC existentes é apresentado por (Meneguette *et al.*, 2021), o qual discute-se também os riscos e as violações, além dos desafios existentes em relação ao uso deste tipo de tecnologia na borda da rede.

Na Tabela 4 estão apresentados todos os trabalhos listados acima (Mensah *et al.*, 2021), (Wang *et al.*, 2021), (Zhang; Wang; Liu, 2020), (Hu *et al.*, 2020), (Zhou; Jadoon, 2020), (Zafari *et al.*, 2018), (Bui; Jung, 2017), (Zhao; Lu; Zhiying, 2020), (Shen *et al.*, 2020), (Zeng *et al.*, 2021), (Tang *et al.*, 2020), (Araldo; Stefano, 2020), (Avgeris *et al.*, 2019) e (Meneguette *et al.*, 2021) em um formato mais sintetizado de dados, classificando os trabalhos através de itens relacionados a metodologia, contexto, objetivo principal e os problemas que foram analisados:

Tabela 4: Abstração do Estado da Arte

Trabalhos	**Metodologia**	**Contexto**	**Objetivo Principal**	**Problemas Analisados**
(Mensah *et al.*, 2021)	Equilíbrio de Nash	D2D/VEC	CO/Alocação de Recursos	Congestionamento/Alta latência em VCC
(Wang *et al.*, 2021)	Equilíbrio de Nash	V2X/MEC	CO/Alocação de Recursos	Eficiência Energética/Alta latência em IoV
(Zhang; Wang; Liu, 2020)	Equilíbrio de Nash	SDN/MEC	CO/Alocação de Recursos	Eficiência/Flexibilidade da rede
(Hu *et al.*, 2020)	Equilíbrio de Nash	MEC	CO/Alocação de Recursos	Compartilhamento de Recursos
(Zhou; Jadoon, 2020)	Equilíbrio de Nash	MEC	CO/Alocação de Recursos	Múltiplos Usuários
(Zafari *et al.*, 2018)	Ótimas de Pareto	EC	CO/Alocação de Recursos	Otimização de Recursos
(Bui; Jung, 2017)	Jogos Cooperativos	Redes	Alocação de Recursos	Interferência da Rede
(Zhao; Lu; Zhiying, 2020)	ACO	Redes	Alocação de Recursos	Gestão de Múltiplos Sensores
(Shen *et al.*, 2020)	Jogo da Minoria	Mobile/EC	Alocação de Recursos	QoS e QoE
(Zeng *et al.*, 2021)	*Stackelberg*	VEC	CO/Alocação de Recursos	Custo e Locação
(Tang *et al.*, 2020)	*Stackelberg*	MEC	CO/Alocação de Recursos	Eficiência de Custo
(Araldo; Stefano, 2020)	MORA	EC	Alocação de Recursos	QoS e QoE
(Avgeris *et al.*, 2019)		MEC	Alocação de Recursos	Alternativas
(Meneguette *et al.*, 2021)		VEC	CO/Alocação de Recursos	*Overview*

Fonte: Elaborado pelo autor, 2021.

3.3 Ameaças e Limitações

O objetivo desta seção é descrever as prováveis ameaças existentes a essa SLR. Além disso, outro agente importante que também será apresentado são as possíveis limitações que este processo sistemático pode conter, ou seja, quais os pontos que não foram abrangidos nessa pesquisa. Esses fatores são fundamentais para a contribuição do mapeamento do estudo, principalmente para auxiliar outros pesquisadores que podem se beneficiar deste trabalho.

As principais ameaças desta SLR são referentes a:

1. A pesquisa foi realizada por apenas um pesquisador, ou seja, grande parte das etapas do protocolo (desde o seu desenvolvimento) foi avaliado por apenas uma pessoa, com exceção da última etapa de decisão de escolha dos estudos, o que pode fazer com que alguns trabalhos ou trechos importantes tenham sido ignorados.

2. O viés do pesquisador também pode ser um fator que pode influenciar na tomada de decisão de seleção dos resultados. Entretanto, os critérios de seleção do trabalho servem justamente para que o mesmo seja feito da forma mais imparcial possível.

As principais limitações desta revisão sistemática são referentes a:

1. O objetivo deste trabalho é encontrar uma solução que consiga maximizar a eficiência dos resultados em aspectos de CO e decisão de alocação e gerenciamento de recursos através de algoritmos tradicionais em um paradigma VEC, com foco em redes IoV. No entanto, diversos aspectos importantes existentes nas redes IoV foram desconsiderados, para manter o foco em VEC, como uso de estratégias das áreas de Inteligência Artificial (*Artificial Inteligence* - AI) e *Blockchain*.

2. O foco do trabalho é buscar uma solução que consiga maximizar a CO e a decisão de alocação e gerenciamento de recursos através de algoritmos tradicionais embasados na GT. Contudo, a limitação de tempo fez com que essa pesquisa fosse realizada apenas de forma manual, ou seja, algumas outras estratégias de busca como *Snowball*, que visa buscar nas referências ou citações dos artigos escolhidos outros trabalhos relevantes que também tratam do mesmo tema. Em consequência disso, algumas pesquisas de relevância podem não ter sido selecionadas para as avaliações, que consiste em extrações de dados e comparação de resultados.

Enfim, essas ameaças e limitações não inviabilizam de forma alguma o mérito do trabalho, apenas reforçam que alguns pontos devem ser analisados com bastante atenção, já que o contexto no qual esta pesquisa está envolvida é bastante específica. Entretanto, pesquisadores que se interessam pelos assuntos de CO, decisão de alocação e gerenciamento de recursos, VEC e redes IoV, podem tirar proveito deste trabalho.

Capítulo 4

Teoria dos Jogos em Redes IoV

Neste capítulo está descrito toda a metodologia utilizada para o desenvolvimento da arquitetura e do algoritmo proposto neste trabalho. A arquitetura de três camadas explicada no Item 2.1.2 será implementada com o uso de algoritmos cooperativos de GT, que utiliza o Equilíbrio de NE como principal estratégia para a otimização da alocação de recursos, exemplificado no Item 2.3.1.

O objetivo principal é utilizar o algoritmo METRO para contribuir na resolução dos problemas de latência, mobilidade e congestionamento de rede existentes no cenário de IoV, alocando os recursos de processamento, memória e armazenamento de maneira mais eficiente. Tudo isso será realizado com base nos dados recebidos no processo de CO, para a orquestração completa de todo o cenário.

Além disso, este capítulo contém informações relevantes acerca dos outros dois algoritmos que também foram implementados e executados neste mesmo cenário, com o objetivo de serem utilizados como parâmetros comparativos da eficiência do algoritmo proposto. Sendo eles o algoritmo SRD e o algoritmo de GWO, os quais são respectivamente embasados nos conceitos que foram detalhados nos Itens 2.3.2 e 2.4.1.

Por fim, este capítulo está organizado da seguinte forma: na Seção 4.1 está descrito o funcionamento do algoritmo METRO para a orquestração da alocação dos recursos, na Seção 4.2 é apresentado todo o cenário para qual o algoritmo será projetado, e por último, na Seção 4.3 está contido os detalhes de todas as ferramentas de simulação

necessárias para este contexto.

4.1 Algoritmo METRO

O algoritmo METRO do acrônimo *ga**ME** **T**heory **R**esource all**O**cation* utiliza os jogos cooperativos da GT como base de otimização da alocação de recursos, em especial o NE, para maximizar o atendimento de todas as requisições que são solicitadas.

Em resumo, o processo é realizado por meio das seguintes etapas:

1. Os dispositivos presentes na Camada Veicular são sensíveis ao ambiente, e por isso, conseguem interagir-se entre si. Dessa forma, quando há necessidade de processar uma determinada tarefa, a solicitação é enviada para a VEC, processo conhecido como CO;

2. Toda a lógica de gerenciamento da decisão de alocação de recursos é efetuada na Camada de Computação em Borda. É nessa camada que é decidido o que será processado em servidores VEC e o que será enviado para servidores CC, com base na prioridade da tarefa, que é recebido juntamente em cada solicitação da Camada Veicular. Nessa etapa que a GT começa a fazer parte do processo, sendo que os servidores VEC e CC são considerados os jogadores (*players*), e nesse momento, há duas possibilidades que são: processar a tarefa em VEC ou processar a tarefa em CC. Essas possibilidades são as estratégias dos jogadores, otimizadas através do NE. Por isso os jogos são denominados cooperativos, já que os jogadores interagem entre si buscando a melhor solução para ambos; e

3. Por fim, a Camada de Computação em Nuvem recebe todas as solicitações compatíveis, de acordo com a prioridade da tarefa. Na grande maioria dos casos, são solicitações que exigem alto poder computacional e não necessitam de atendimento imediato, ou seja, solicitações que não possuem problemas de alta latência.

Nesse contexto, o NE pode ser representado da seguinte maneira: Dois jogadores *i* e *j* recebem uma lista de tarefas com valores de prioridade (*p*), processamento (*P*), memória (*M*) e armazenamento (*S*). Para cada tarefa, a função de utilidade do jogador *i* pode ser representada como:

$$Ui(p, Pi, Mi, Si) = fi(p, Pi, Mi, Si)$$

Assim como a função do de utilizado do jogador *j* pode ser representada como:

$$Uj(p, Pj, Mj, Sj) = fj(p, Pj, Mj, Sj)$$

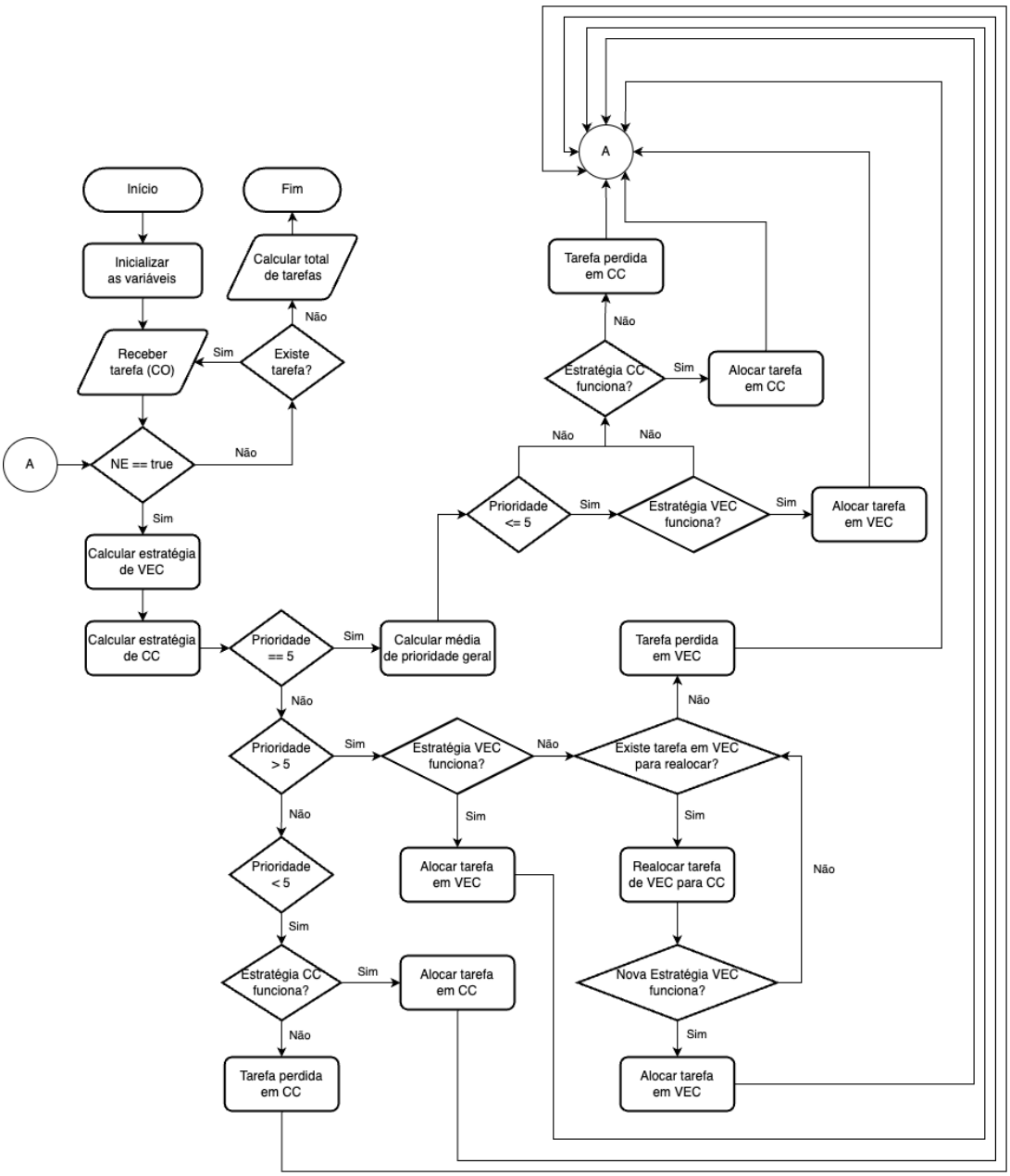
Onde f_i e J_i são as funções responsáveis por modelar a utilidade com base nas prioridades de tarefa e alocação de recursos. Sendo assim, o NE ocorre quando nenhum jogador pode melhorar sua utilidade mudando de forma unilateral a sua estratégia, dadas as estratégias de todos os jogadores envolvidos no contexto.

A Figura 18 apresenta um fluxograma que explica de forma geral o funcionamento do algoritmo METRO

O fluxo é iniciado com o mapeamento da quantidade de recursos da rede, ou seja, a soma da capacidade de processamento, memória e armazenamento de cada nó existentes na rede, separados de acordo com a sua classificação, que pode ser VEC ou CC. Quando um dispositivo na rede como um veículo faz a solicitação de uma tarefa, ocorre o processo de CO. Nesse momento, a Camada de Computação em Borda recebe a solicitação.

Após a solicitação ser recebida, caso a prioridade for maior que 5, obrigatoriamente essa tarefa deve ser alocada em VEC se os recursos existentes na mesma estiverem disponíveis. Mas caso não haja recurso disponível em VEC, existe um mecanismo de cooperação em que a VEC procura transferir as tarefas já alocadas com prioridade igual a 5 para servidores CC. Esse processo é repetido enquanto os recursos ainda não forem suficientes e existirem tarefas com prioridade 5 alocadas em VEC, ou seja, a tarefa destinada para VEC possui uma prioridade alta e a CC coopera o máximo com a VEC para que a solicitação não seja perdida.

Figura 18: Fluxograma do Algoritmo METRO



Fonte: Elaborado pelo autor, 2023.

Se a prioridade for menor que 5, a tarefa é enviada para a próxima camada é que a camada de Computação em Nuvem, que também irá tentar alocar a tarefa se a quantidade de recursos solicitada estiver disponível. Por fim existe o principal momento de decisão do algoritmo, que é quando uma tarefa de prioridade 5 precisa ser alocada, sendo que esse tipo de solicitação pode ser atendida tanto em VEC quanto em CC.

Para realizar esse cálculo há um mecanismo de colaboração entre os jogadores (VEC e CC), que buscam encontrar um equilíbrio para entender a demanda atual de solicitações. Obviamente a preferência de alocação de tarefas com esse tipo de prioridade é a VEC, no entanto caso haja um desequilíbrio entre a relação de tarefas alocadas em VEC/CC, a demanda é imediatamente direcionada para a CC.

O algoritmo 1 METRO está apresentado com as respectivas informações de *input* e *output* de dados, além da declaração das variáveis e todas as operações que são executadas pelo mesmo.

Com uma visão mais ampla do que foi mostrado na Figura 18, o algoritmo 1 tem o seu começo definido pelo *Input* que nada mais é do que a entrada, ou seja, o valor recebido para processamento. No caso do METRO, um conjunto *n* de tarefas é recebido, sendo representado por $T = \{t_1, t_2, ..., t_n\}$. O conjunto de tarefas (T) possui além das informações de processamento, memória e armazenamento a prioridade de execução da tarefa. Já o *output* que é o resultado do processamento do algoritmo, é representado por $A, M,\ \ AV, MV, AC, MC = \{t_1, t_2, \ldots, t_n\}$ que são respectivamente o total de tarefas que foram alocadas, o total de tarefas que não foram alocadas (perdidas), o total de tarefas alocadas em VEC, o total de tarefas não alocadas (perdidas) em VEC por falta de recursos computacionais, o total de tarefas alocadas em CC e por fim, o total de tarefas que não foram alocadas (perdidas) em CC por falta de por falta de recursos computacionais.

Na linha 1 à linha 3 é feita a inicialização das variáveis utilizadas ao longo da execução do algoritmo:

- **VEC:** Total de nós de servidores de Computação em Borda (*Edge Computing - EC*), com capacidades de processamento, memória e armazenamento somados;
- **CC:** Total de nós de servidores de Computação em Nuvem (*Cloud Computing - CC*), com capacidades de processamento, memória e armazenamento somados;
- **NE:** Indica se o NE foi encontrado (verdadeiro ou falso);

Algoritmo 1 METRO - *ga**ME** **T**heory **R**esource all**O**cation*

Input: $T = \{t_1, t_2, ..., t_n\}$
Output: $A, M,\ AV, MV, AC, MC = \{t_1, t_2, \ldots, t_n\}$,

$VEC \leftarrow X$
$CC \leftarrow X$
$NE \leftarrow$ true
while $NE \leftarrow$ true **do**
  $vecStrategyIsWork \leftarrow vecStrategy(T)$
  $ccStrategyIsWork \leftarrow ccStrategy(T)$
  if $T.priority \equiv 5$ **then**
    $taskPriorityAverage \leftarrow calculateTaskPriorityAverage()$
    if $taskPriorityAverage \leq 5$ **and** $vecStrategyIsWork \equiv$ true **then**
      $allocateTaskInVEC(T)$
    else if $ccStrategyIsWork \equiv$ true **then**
      $allocateTaskInCC(T)$
    else
      $missedTaskInCC(T)$
    end if
  else if $T.priority > 5$ **then**
    if $vecStrategyIsWork \equiv$ true **then**
      $allocateTaskInVEC(T)$
    else
      $haveTaskToReallocate \leftarrow$ true
      while $haveTaskToReallocate \leftarrow$ true **do**
        $foundTaskInVECByPriority \leftarrow findTaskInVECByPriority(5)$
        if $foundTaskInVECByPriority \equiv$ true **then**
          $reallocatedTaskToCC(foundTaskInVECByPriority)$
          $newVecStrategyIsWork \leftarrow vecStrategy(T)$
          if $newVecStrategyIsWork \equiv$ true **then**
            $allocateTaskInVEC(T)$
          end if
        else
          $missedTaskInVEC(T)$
        end if
      end while
    end if
  else if $T.priority < 5$ **then**
    if $ccStrategyIsWork \equiv$ true **then**
      $allocateTaskInCC(T)$
    else
      $missedTaskInCC(T)$
    end if
  end if
end while
$A \leftarrow AV + AC$
$M \leftarrow MV + MC$
return A, M, AV, MV, AC, MC

O fluxo de decisão começa à partir da linha 4, que é o momento que ocorre o CO, ou seja, uma tarefa T faz a solicitação dos recursos computacionais necessários para ser executada, que são processamento, memória e armazenamento, além da informação da escala de prioridade. Todos esses valores são representados por números inteiros, no intervalo de 1 até 10. Nessa etapa, as linhas de 4 a 33 são executadas enquanto o NE não for encontrado.

As linhas 4 e 5 são responsáveis por, respectivamente, calcular as estratégias de VEC e CC, ou seja, através da tarefa que é passada como argumento da função, é possível verificar se tanto a VEC quanto a CC conseguem alocar recursos para essa tarefa. Nesse momento, ainda não há uma preocupação em relação a prioridade da tarefa.

Posteriormente, caso a prioridade da tarefa é verifica, da linha 7 à 15 é responsável por alocar tarefas que possuem prioridade igual a 5. Para isso, primeiro uma média de prioridade das tarefas é calculada, esse passo é importante qual das duas infraestruturas de rede estão sendo mais demandadas. Se o resultado dessa operação for menor ou igual a 5 e a VEC possuir recursos suficientes, então a tarefa é alocada na borda da rede, para garantir melhor QoS e QoE. No entanto, caso o resultado seja maior que 5, quer dizer que a borda está tendo uma demanda maior que a nuvem, ou seja, como tarefas de prioridade 5 podem ser executadas tanto em VEC quanto em CC, nesse caso ela é direcionada para CC caso a mesma também possua recursos computacionais suficientes.

Já nas linhas 16 à 34 são responsáveis por realizarem a alocação de tarefas extremamente sensíveis, já que a prioridade é maior que 5. Por isso, há uma cooperação entre a Camada de Borda e a Camada de Nuvem, já que a VEC faz a transferência de tarefas que possuem prioridade 5 quando não há recursos computacionais para alocar a tarefa atual.

Quando a prioridade da tarefa é menor que 5, a mesma deve ser imediatamente

enviada para a CC, que vai alocar o recurso caso haja recursos computacionais no momento. Nesse caso, por se tratar da última camada, não há um mecanismo de transferências de tarefas, porém por se tratar de tarefas não sensíveis a perda dessas tarefas não prejudica o cenário, mas atrapalha a QoE.

O *payoff* que é a recompensa dos jogadores é representado por duas funções $allocateTaskInVEC(T)$ e $allocateTaskInCC(T)$. Isto quer dizer que, de acordo com a estratégia de cada jogador para determinada prioridade, a recompensa é alocar a tarefa em questão. Esses métodos também possuem a responsabilidade de modificar o valor da variável *NE* para *true* (verdadeiro), fazendo com que o algoritmo saia do laço de repetição e aguarde a chegada de uma nova tarefa.

Por último, a implementação completa do Algoritmo na linguagem de programação C++, que foi adicionada ao arquivo de *middleware* do Artery está disponível no Apêndice B.

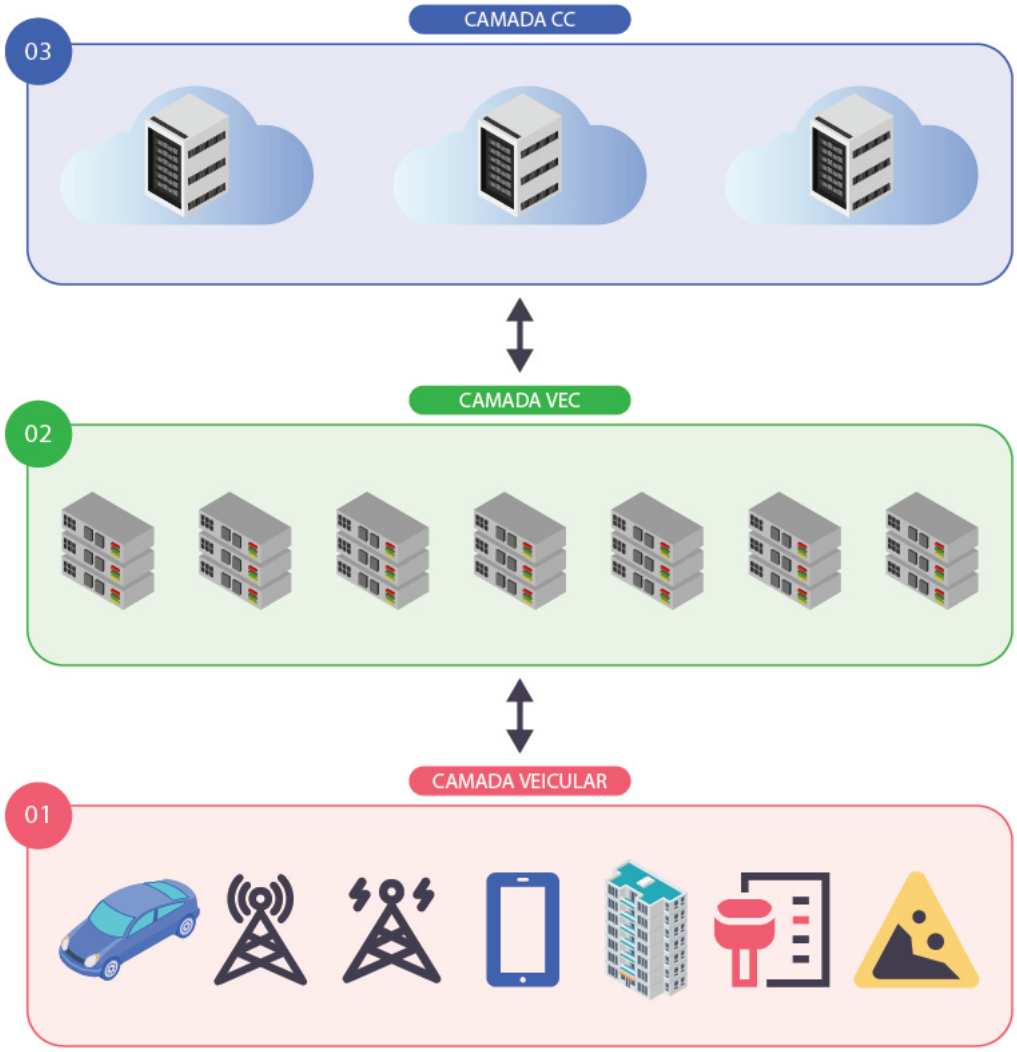
4.2 Cenário de Execução do Algoritmo

O contexto geral da aplicação do algoritmo é extremamente importante para fins de simulação e principalmente para a análise dos resultados obtidos na mesma. Dessa forma, é importante documentar todo o cenário que será utilizado no ambiente de simulação para a execução dos algoritmos. Por isso, a Figura 19, descreve todo arquitetura e as suas divisões de responsabilidades

É possível observar que as três camadas da arquitetura possuem responsabilidades bem definidas, sendo respectivamente descritas as camadas de nível inferior para nível superior: Camada Veicular, Camada de Computação em Borda Veicular (VEC) e Camada de Computação em Nuvem (CC).

A Camada Veicular é a primeira camada da arquitetura, ela basicamente possui a responsabilidade de realizar o CO, ou seja, a operação de transferência dos dados para a próxima camada. Em outras palavras, é nessa camada que as solicitações dos

Figura 19: Cenário de Simulação



Fonte: Elaborado pelo autor, 2022.

serviços são criadas, e posteriormente processadas nas camadas superiores. Isto quer dizer que é nessa camada que acontece as interações existentes entre os dispositivos, por exemplo, uma interação V2X.

No nível intermediário está a Camada de Computação em Borda, é nessa camada que o Algoritmo METRO executa a sua função de gerenciamento de alocação de recursos, processo explicado com detalhes na Seção 4.1. Em resumo, um conjunto de solicitações de serviços são criadas, com base no processo de CO, que traz informa-

ções importantes, principalmente à respeito da prioridade da execução e dos recursos necessários para o processamento da mesma.

Já a camada final da arquitetura possui uma capacidade altamente escalável dos recursos de processamento, memória e armazenamento, porém com uma alta latência em comparação a camada VEC, e consequentemente com diversos problemas gerados no contexto, como os que estão relacionados a falta de mobilidade. No entanto, essa camada é importante para a contribuição no processamento de serviços não sensíveis a atrasos, resolvendo assim solicitações enviadas pela camada intermediária, gerando um processo de distribuição de responsabilidades, contribuindo assim para a mitigação de grandes problemas de congestionamento da rede.

4.3 Ferramentas de Simulação

Esse conjunto de ferramentas foi necessário para desenvolver todo o cenário descrito na Seção 4.2, com a finalidade de executar o Algoritmo METRO, detalhado na Seção 4.1, além dos demais algoritmos SRD e GWO, que estão respectivamente explicados nos Itens 2.3.2 e 2.4.1 com o objetivo de obter os resultados apresentados no Capítulo 5.

4.3.1 *Simulation of Urban Mobility* - SUMO

***S**imulation of **U**rban **MO**bility* - **SUMO**[1] foi projetado para lidar com redes de alta densidade, é um pacote de simulação de tráfego de código aberto que permite a simulação de cenários personalizados através do seu conjunto de ferramentas (SUMO, 2022).

4.3.2 *Objective Modular Network Testbed* - OMNeT++

***O**bjective **M**odular **NE**twork **T**estbed in C++* - **OMNeT++**[2] considerado uma estrutura modular de simulação desenvolvida em C++ baseada em componentes, sendo

utilizada com foco para o desenvolvimento de simuladores de redes (OMNeT++, 2022).

4.3.3 *Artery*

Artery[3] começou como uma extensão, ou seja, um módulo do ***VE****hicles* ***I****n* ***N****etwork* ***S****imulation* - Veins, mas atualmente é um projeto paralelo que pode ser executado de forma isolada. A referência do nome vem do projeto Veins, por isso o uso da palavra artéria, do inglês *artery*, termo que possui conexão com veias, do inglês *veins*. O principal diferencial do Artery em relação ao Veins é que o mesmo possui suporte para simulações V2X e além disso, possui os *middlewares*, que são basicamente interceptadores, que podem ser utilizados para diversos fins, como quando uma troca de mensagens ocorre entre dois dispositivos (Artery, 2023). Os algoritmos serão executados dentro do arquivo de *middleware* do código-fonte do Artery, em que a sua implementação na linguagem de programação C++ está disponível nos Anexos A B.

[1]https://sumo.dlr.de/docs/index.html
[2]https://omnetpp.org/
[3]http://artery.v2x-research.eu/

Capítulo 5

Resultados

Este capítulo apresenta os resultados obtidos por meio da execução dos algoritmos METRO, SRD e GWO em ambiente de simulação, detalhado no Capítulo 4.

O capítulo está organizado da seguinte forma: a Seção 5.1 contém as especificidades dos parâmetros de simulação utilizados, a Seção 5.2 traz o *dataset* de tarefas que foi utilizado em todas as execuções, já a Seção 5.3 apresenta os resultados organizados em tabelas das execuções dos algoritmos, a Seção 5.4 detalha as métricas que foram definidas para a comparação dos resultados, que estão apresentados na Seção 5.5, e por último, a Seção 5.6 traz as considerações parciais com base em tudo o que foi discorrido durante este capítulo.

5.1 Parâmetros de Simulação

Os parâmetros de simulação envolvem diversas variáveis, dentre as principais são as tarefas, o mapa e as demais configurações, relacionadas a mobilidade e transmissão, por exemplo.

5.1.1 Tarefas (*Tasks*)

Uma tarefa é basicamente uma solicitação de um dos nós da rede IoV que precisa ser processada, ou seja, resolvida para ser considerada uma tarefa atendida, e caso não

haja recursos suficientes para atender a essa solicitação, a tarefa é considerada perdida.

No cenário deste trabalho, os veículos possuem uma capacidade mínima de recursos computacionais, e por isso, faz-se necessário que a resolução dessas tarefas sejam realizadas em algum tipo de servidor dedicado para isso, que neste contexto são os servidores presentes na borda da rede, conhecidos como VEC e os servidores em nuvem, conhecidos como CC, cenário que foi apresentado na Seção 4.2.

Para a simulação, quatro variáveis importantes foram levadas em consideração para a execução de uma tarefa, sendo: Prioridade, processamento, memória e armazenamento. Esses valores são representados por números inteiros, que podem ser de no mínimo 1 e no máximo 10. A prioridade indica o quão sensível é determinada solicitação, ou seja, quanto maior a prioridade, mais rápido a tarefa deve ser executada. Se a prioridade estiver no intervalo de 1 a 4, a tarefa é executada em servidores CC, se a prioridade estiver no intervalo de 6 a 10 a tarefa é executada em servidores VEC, por conta da sua sensibilidade a latência. Por último, se a prioridade for 5 a tarefa pode ser executada em VEC ou em CC, sendo este um dos principais pontos de decisão gerenciados pelo algoritmos executados no contexto. As outras variáveis, que são processamento, memória e armazenamento, indicam a quantidade de recursos necessários para que determinada tarefa seja resolvida, em outras palavras quanto maior forem esses valores, mais complexa é considera essa solicitação. A Tabela 5 traz três exemplos de tarefas diferentes:

Tabela 5: Exemplo de Algumas Tarefas com Valores Aleatórios

Prioridade	Processamento	Memória	Armazenamento
2	7	8	2
8	6	9	4
5	3	7	5

Fonte: Elaborado pelo autor, 2023.

Dessa forma, com base nas condições apresentadas, as tarefas presentes na Tabela

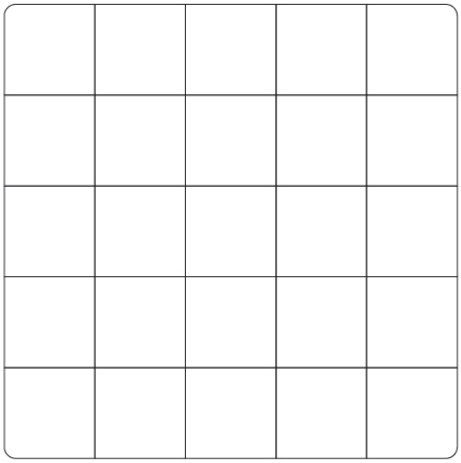
5 devem ser executadas respectivamente em CC, VEC e por último CC ou VEC de acordo com a tomada de decisão presente no algoritmo que está sendo executado no momento, por exemplo, no caso do algoritmo METRO a decisão é tomada através de GT, utilizando NE, detalhado na Seção 4.1.

5.1.2 Mapa

Um dos motivos da escolha é em relação a definição das configurações

Para o ambiente de simulação, foi considerado um mapa no formato de *grid*, que nada mais é do que um mapa quadriculado, como apresentado na Figura 20.

Figura 20: Mapa de *Grid* 5x5 Utilizado nas Simulações



Fonte: Elaborado pelo autor, 2023.

Os principais motivos dessa escolha estão relacionados a questão de que esse mapa em específico já fazia parte de um dos cenários padrões existentes no Artery em diversas infraestruturas de simulação. Portanto, o seu arquivo de configuração é extremamente completo, além disso, há a existência de algumas variações de mobilidades predefinidas nesse mapa.

5.1.3 Configurações

A Tabela 6 é composta pelas colunas de parâmetro e valor, detalhando todo a configuração utilizada no cenário de simulação:

Tabela 6: Configurações da Simulação

Parâmetro	Valor
Geral	
Comunicação	V2X
Cenário	Grid 5x5
Tamanho do Cenário	4000m x 4000m
Tempo de Simulação	60s
Total de Execuções	32x
Mobilidade	
Veículos	De ∼200 até ∼300
RSUs	De ∼100 até ∼200
Outros dispositivos	De ∼100 até ∼200
Rede	
Tecnologias	ITS-G5(IEEE 802.11p) e 5G
Frequência de Transmissão	3.5GHz e 5.9 GHz
Poder de Transmissão	200 mW
Obstáculo na Transmissão	Sim
Dist. Máx de Interferência	1500m
VEC	
Quantidade de Nós	50
Processamento	50
Memória	50
Armazenamento	50
CC	
Quantidade de Nós	3
Processamento	1000
Memória	1000
Armazenamento	1000

Fonte: Elaborado pelo autor, 2023.

A Tabela 6 foi categorizada em: geral, mobilidade, rede, VEC e CC, com o objetivo de descrever melhor esses dados. As categorias apresentam desde os itens mais

importantes citados para a simulação que já foram citados neste capítulo, até outras informações complementares.

Por fim, uma outra variável importante que merece destaque é a mobilidade, fator imprescindível para comprovar a efetividade dos algoritmos executados em um determinado cenário. Além disso, para garantir um resultado de confiança, 32 execuções com diferentes configurações de *seed* (0 a 31) foram realizadas em 3 variações de mobilidade, para cada algoritmo, sendo eles o METRO, SRD e GWO. Em resumo, uma *seed* é uma variável configurável que contribui para o processo randômico do processo de simulação, para diversos fatores, para definir por exemplo a quantidade de tarefas que serão geradas durante todo o processo de simulação.

5.2 Conjunto de Tarefas

O primeiro passo para conseguir obter os resultados é definir a mesma entrada de tarefas para todos os algoritmos. Para isso, um conjunto de dados com um total de cem mil tarefas foi gerado por meio de um código que utiliza métodos de aleatoriedade da linguagem C++. Essas tarefas possuem um identificador denominado *id*, que vai de 1 (um) até 100.000 (cem mil).

Em cada execução da simulação, o total de entrada de tarefas é variável, sendo definido de acordo com o valor da *seed*. Dessa forma, se em um mesmo cenário a simulação for executada *N* vezes, o valor de entrada sempre será o mesmo para cada execução que conter a mesma *seed*.

Por isso, é importante variar a *seed* em cada execução, com o objetivo de modificar o total de entradas das tarefas. Nesse sentido, na execução 0 sempre haverá a quantidade *X* de tarefas e na execução 1 sempre haverá a quantidade *Y* de tarefas, e assim sucessivamente, até a última variação de *seed* que é com o valor 31, ou seja, 32 variações que geram um resultado de aproximadamente 95% de confiança, Dessa forma, independentemente do algoritmo que está sendo testado e da quantidade de ve-

zes que a simulação é executada, a quantidade de tarefas geradas sempre será a mesma para cada execução, fator essencial para não beneficiar ou não prejudicar determinado algoritmo.

5.3 Média das Execuções

Após a execução de cada algoritmo no mesmo cenário com três variações de mobilidade, os resultados extraídos foram separados em tabelas, para formar um outro conjunto de dados que possui o intuito de demonstrar por meio de gráficos, o desempenho de cada algoritmo, de acordo com as métricas definidas na Seção 5.4. É importante salientar que cada tabela contém a média das três variações de mobilidade, explicadas no Seção 5.1.

A Tabela 7 mostra o total de tarefas geradas durante o processo de simulação por execução e valor de *seed*, sendo que a execução 14 com o valor de *seed* 13 gerou 46.352 tarefas, o maior número do intervalo, já a execução 27 com valor de *seed* 26, gerou 36.173 tarafas, o menor número do intervalo.

A Tabela 8 apresenta a média dos resultados da execução do algoritmo METRO, que é o algoritmo proposto deste trabalho. Além da informação do número da execução, há também dados de alocação e perda, tanto em números gerais quanto em números específicos, em relação a VEC e CC, por exemplo. A execução 14 com o valor de *seed* 13, foi a execução com o maior número bruto de tarefas alocadas, um total de 44.291. Já a execução 3 com o valor de *seed* 2, foi a que a execução com o maior número bruto de tarefas perdidas, um total de 3.743. Por fim, é importante destacar a execução 32, com o valor de *seed* 31, que alocou todas as tarefas do cenário, sendo o único algoritmo que conseguiu atingir 100% de aproveitamento de alocação de tarefas nas duas infraestruturas disponíveis, VEC e CC.

Os resultados do algoritmo SRD, estão dispostos na Tabela 9, que detalha a média das execuções das variações de mobilidade. As colunas presentes nesta tabela, são

Tabela 7: Total de Entradas em Cada Execução à Partir da *Seed*

Execução	*Seed*	Total de *tasks*
1	0	42331
2	1	40749
3	2	40788
4	3	40456
5	4	41933
6	5	42666
7	6	40349
8	7	40340
9	8	41346
10	9	40599
11	10	40538
12	11	45154
13	12	39574
14	13	46352
15	14	38772
16	15	38423
17	16	40599
18	17	43486
19	18	39640
20	19	43652
21	20	36456
22	21	42881
23	22	41293
24	23	42212
25	24	38732
26	25	42231
27	26	36173
28	27	41140
29	28	42508
30	29	39325
31	30	41035
32	31	40887

Fonte: Elaborado pelo autor, 2023.

as mesmas da Tabela 8, com informações acerca de total de tarefas alocadas e total de tarefas não alocadas, também apresentadas em números gerais e de acordo com o servidor de alocação.

A execução que obteve a maior quantidade bruta de tarefas alocadas foi a de nú-

Tabela 8: Média das Execuções do Algoritmo METRO

Execução	Total Alocado	Alocado em VEC	Alocado em CC	Total perda	perda em VEC	perda em CC
1	39494	21192	18302	2837	0	2837
2	39366	19030	20336	1383	0	1383
3	37045	20428	16617	3743	0	3743
4	37164	16974	20190	3292	0	3292
5	40673	20527	20146	1260	0	1260
6	40410	20532	19878	2256	0	2256
7	37539	20215	17324	2810	0	2810
8	38206	18077	20129	2134	0	2134
9	39523	20725	18798	1823	0	1823
10	38434	20347	18087	2165	0	2165
11	40400	20319	20081	138	0	138
12	42258	22620	19638	2896	0	2896
13	38651	18900	19751	923	0	923
14	44291	23214	21077	2061	0	2061
15	37615	19420	18195	1157	0	1157
16	36423	19235	17188	2000	0	2000
17	38680	20347	18333	1919	0	1919
18	43115	21761	21354	371	0	371
19	36141	16352	19789	3499	0	3499
20	41001	19198	21803	2651	0	2651
21	35238	18263	16975	1218	0	1218
22	41706	21462	20244	1175	0	1175
23	41278	20697	20581	15	0	15
24	40247	21126	19121	1965	0	1965
25	37223	21135	16088	1509	0	1509
26	39682	21135	18547	2549	0	2549
27	35947	18070	17877	226	0	226
28	39418	20611	18807	1722	0	1722
29	40285	21282	19003	2223	0	2223
30	38617	19701	18916	708	0	708
31	38125	20557	17568	2910	0	2910
32	40887	20481	20406	0	0	0

Fonte: Elaborado pelo autor, 2023.

mero 14, com valor de *seed* 13, que alocou um total de 44.383 tarefas, já a execução 3, com valor de *seed* 2, obteve o maior número bruto de tarefas perdidas, sendo um total de 4.815 tarefas.

Finalmente, a Tabela 10 que contém a mesma estrutura das Tabelas 8 e 9 traz os resultados do algoritmo GWO. A execução de número 14, com valor de *seed* 13, teve um um total bruto de 44.212 tarefas alocadas. Já a execução 29, com valor de *seed* 28, obteve o maior perda bruta de tarefas, totalizando um valor de 3.709.

Tabela 9: Média das Execuções do Algoritmo SRD

Execução	Total Alocado	Alocado em VEC	Alocado em CC	Total perda	perda em VEC	perda em CC
1	39870	22016	17854	2461	1177	1284
2	40267	22427	17840	482	0	482
3	35973	19689	16284	4815	2532	2283
4	38168	19812	18356	2288	2288	0
5	40572	21593	18979	1361	1361	0
6	40094	23280	16814	2572	213	2359
7	38842	21039	17803	1507	1102	405
8	38802	20548	18254	1538	1538	0
9	39563	21080	18483	1783	1559	224
10	36960	18351	18609	3639	3639	0
11	39492	21217	18275	1046	1046	0
12	40579	19894	20685	4575	4575	0
13	39131	21313	17818	443	443	0
14	44383	25597	18786	1969	0	1969
15	37443	20777	16666	1329	499	830
16	36348	18864	17484	2075	2075	0
17	38678	20251	18427	1921	1921	0
18	42118	22405	19713	1368	1368	0
19	35317	17107	18210	4323	4323	0
20	39412	19383	20029	4240	4240	0
21	35969	19538	16431	487	487	0
22	40300	20792	19508	2581	2581	0
23	40957	22360	18597	336	223	113
24	40356	21551	18805	1856	1514	342
25	37285	19707	17578	1447	1447	0
26	40277	21235	19042	1954	1919	35
27	35136	18840	16296	1037	1037	0
28	38520	20982	17538	2620	1537	1083
29	39118	19686	19432	3390	3390	0
30	37007	19121	17886	2318	2318	0
31	38141	22386	15755	2894	227	2667
32	39613	21124	18489	1274	1274	0

Fonte: Elaborado pelo autor, 2023.

5.4 Métricas das Avaliações

As métricas de avaliação são extremamente importantes para a avaliação do desempenho dos algoritmos. É necessário compreender os dados brutos, os quais foram apresentados na Seção 5.3, para extrair o máximo de informações relevantes possíveis.

Dessa forma, no contexto deste trabalho, é muito importante saber o percentual de tarefas que foram alocadas, por exemplo. Isto quer dizer que a quantidade de tarefas que foram alocadas ou perdidas é uma métrica relevante, assim como a mesma informação mas de forma específica em VEC e CC, o que basicamente é uma categorização

Tabela 10: Média das Execuções do Algoritmo GWO

Execução	Total Alocado	Alocado em VEC	Alocado em CC	Total perda	perda em VEC	perda em CC
1	38919	18822	20097	3412	2361	1051
2	38617	18272	20345	2132	2132	0
3	36523	17190	19333	4265	3229	1036
4	36598	16399	20199	3858	3858	0
5	40586	19928	20658	1347	1054	293
6	40495	20377	20118	2171	974	1197
7	37799	17656	20143	2550	2550	0
8	37849	17711	20138	2491	2491	0
9	38694	18246	20448	2652	2470	182
10	37337	17076	20261	3262	3262	0
11	39386	19158	20228	1152	1152	0
12	40590	18047	22543	4564	4564	0
13	38179	18419	19760	1395	1395	0
14	44212	21718	22494	2140	1487	653
15	37689	18830	18859	1083	581	502
16	36438	17241	19197	1985	1985	0
17	37692	17431	20261	2907	2907	0
18	41528	19794	21734	1958	1958	0
19	35810	16012	19798	3830	3830	0
20	40134	18322	21812	3518	3518	0
21	36285	18254	18031	171	0	171
22	38252	16824	21428	4629	4629	0
23	39497	18892	20605	1796	1796	0
24	40319	20330	19989	1893	787	1106
25	35068	15725	19343	3664	3664	0
26	39845	18740	21105	2386	2386	0
27	35380	17301	18079	793	793	0
28	38907	18369	20538	2233	2233	0
29	36799	15564	21235	5709	5709	0
30	36687	17054	19633	2638	2638	0
31	38162	19604	18558	2873	944	1929
32	39070	18655	20415	1817	1817	0

Fonte: Elaborado pelo autor, 2023.

desses dados.

Além disso, existem ainda algumas métricas mais implícitas, como é o caso da relação existente entre as tarefas que foram alocadas em VEC e CC. Esse parâmetro é bastante relevante para a medição do congestionamento de rede, já que a distribuição das responsabilidades ajuda a mitigar gargalos ou ociosidades na rede.

Uma outra métrica importante é a prioridade da tarefa, pois é através dessa informação que os algoritmos tomam a decisão da infraestrutura que será alocada os recursos. É importante ressaltar que dentro do âmbito desta dissertação, tarefas com

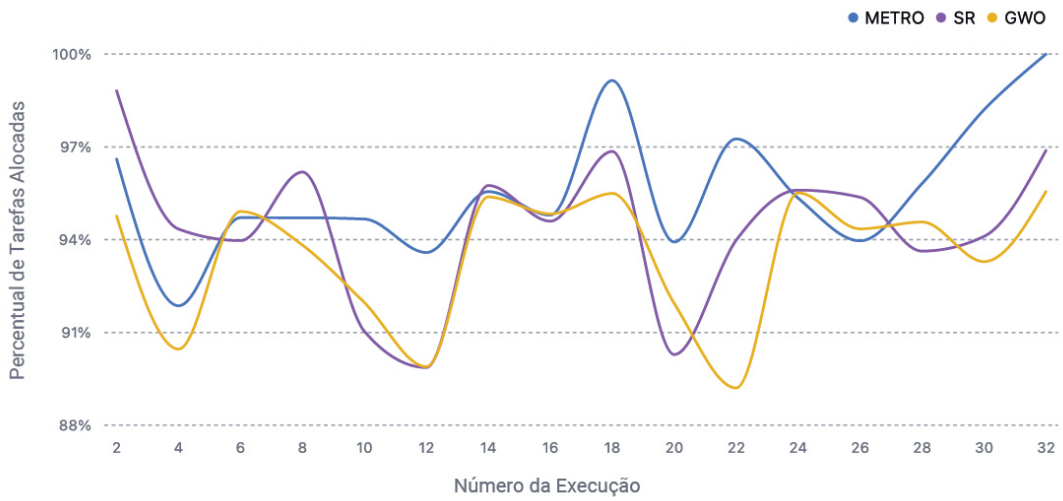
prioridade maior que 5 devem ser executadas em VEC, devido a sua sensibilidade a latência e urgência de resolução.

5.5 Performance dos Algoritmos

À partir dos dados apresentados na Seção 5.3 e das métricas apresentadas na Seção 5.4, está descrito nesta seção a comparação do desempenho dos algoritmos METRO, SRD e GWO.

A Figura 21 traz a performance dos três algoritmos por meio da relação entre a quantidade total de tarefas e a quantidade de tarefas alocadas. De início, é possível observar que a variação do desempenho dos algoritmos SRD e GWO é maior em comparação ao algoritmo METRO. Nesse sentido, fica perceptível que a alocação de recursos é mais previsível e consequentemente mais estável com o algoritmo METRO. Além disso, à partir da execução de número 11 o METRO começa a obter um melhor desempenho, com uma pequena variação no intervalo da execução 24 a 28, mas com um aproveitamento de 100% na última execução, sendo o único algoritmo do cenário a alcançar esse resultado.

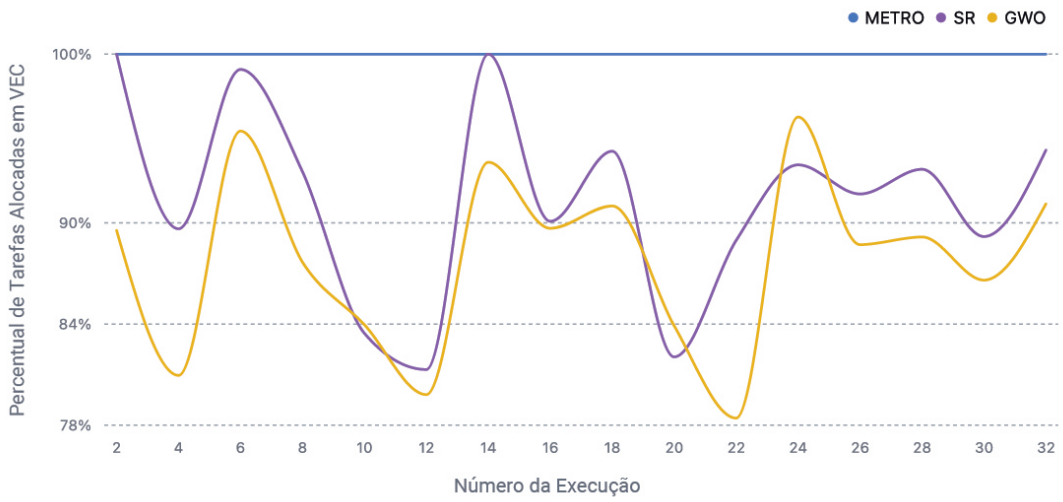
Figura 21: Percentual do Total de Tarefas Alocadas por Execução



Fonte: Elaborado pelo autor, 2023.

O gráfico representado na Figura 22, segue o mesmo princípio do apresentado na Figura 21, em que o eixo X representa o número da execução e o eixo Y o percentual de tarefas que foram alocadas, mas dessa vez, considerando apenas as tarefas alocadas em VEC

Figura 22: Percentual de Tarefas Alocadas em VEC por Execução



Fonte: Elaborado pelo autor, 2023.

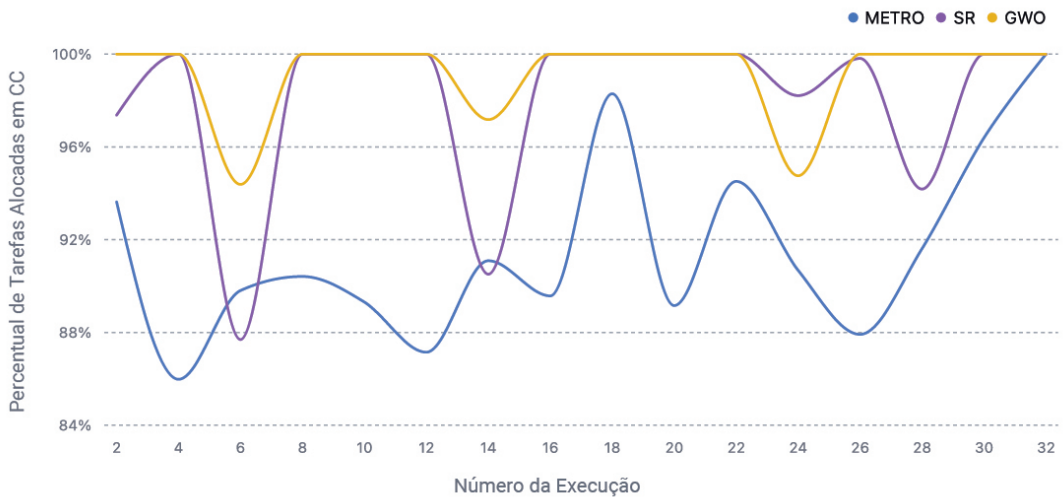
É evidente como o uso de GT, no caso o conceito de NE contribuiu de forma significativa para que os jogadores (VEC e CC) conseguissem cooperar as suas estratégias (alocar ou perder tarefas), de forma a encontrar um ponto de equilíbrio entre eles. Neste caso, a estratégia definida pelo algoritmo METRO é sempre priorizar as tarefas de alta complexidade, ou seja, garantir que tarefas com um nível de prioridade acima de 6 sejam executadas na borda da rede, por meio de mecanismos de realocação explicados na Seção 4.1 e disponíveis no Apêndice B.

Por isso, o algoritmo METRO conseguiu um resultado superior para o contexto de alocação de recursos de tarefas consideradas sensíveis, já que em todas as execuções o mesmo obteve um aproveitamento de 100% para a alocação de tarefas em VEC, como pode ser visto na Figura 22.

A Figura 23 também possui a mesma estrutura da Figura 21, mas considerando

apenas a relação entre o total de tarefas que deveriam ter sido alocadas em CC e o total que de fato que foi alocado, com o valor também representado em percentual.

Figura 23: Percentual de Tarefas Alocadas em CC por Execução



Fonte: Elaborado pelo autor, 2023.

Um dos *trade-offs* de priorizar as tarefas mais sensíveis em VEC é uma maior perda de tarefas em CC. Por mais que os jogos sejam classificados como cooperativos, há uma tendência em beneficiar a alocação mais prioritária, ou seja, a tarefa que deve ser alocada em VEC. O NE foi o responsável para esse acontecimento, já que a estratégia adotada foi encontrar a melhor solução, isto significa o equilíbrio para os jogadores do cenário em um contexto em que o mínimo de tarefas sensíveis fossem perdidas.

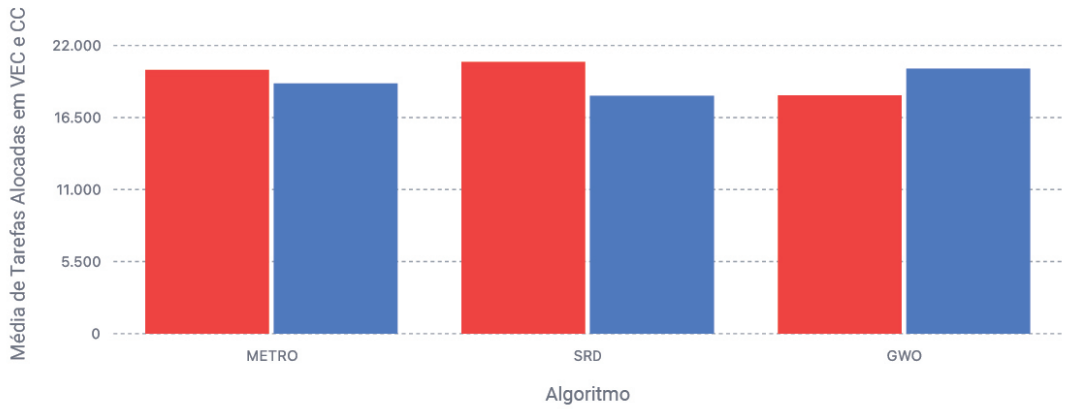
No pior desempenho do algoritmo METRO, demonstrado na execução de número 4, houve uma performance 16% menos eficiente em relação aos outros dois algoritmos, sendo importante ressaltar que as tarefas que são alocadas em CC são de baixa prioridade. Entretanto, ainda é um desempenho melhor quando comparado a Figura 22, mais precisamente na execução de número 22, em que o algoritmo GWO teve um desempenho de aproximadamente 22% menor que o algoritmo METRO, em um cenário de tarefas sensíveis.

Por fim, com o objetivo de demonstrar uma melhor distribuição das tarefas nas

execuções do algoritmo METRO, entre os servidores VEC e CC, foram elaboradas as Figuras 24 e 25.

A Figura 24 traz a relação entre a média da quantidade total de tarefas que foram alocados em VEC e CC. Quanto maior a diferença, mais prejudicial é para o cenário, já que um dos tipos de servidores da rede, VEC ou CC, pode ser sobrecarregado ou até mesmo subutilizado. Nesse aspecto o algoritmo METRO obteve um melhor desempenho, pois CC obteve em média 1024 tarefas a menos que a VEC, isto quer dizer que além da diferença ser menor em relação aos outros dois algoritmos, a tendência de alocação da tarefa é maior em VEC. O algoritmo SRD também obteve uma tendência em alocar as tarefas em VEC, com aproximadamente 2601 tarefas a mais em relação a CC, e por último, o algoritmo GWO foi o único que alocou mais tarefas em CC do que em VEC, com uma diferença média de aproximadamente 2044 tarefas.

Figura 24: Média do Total de Tarefas Alocas em VEC e CC

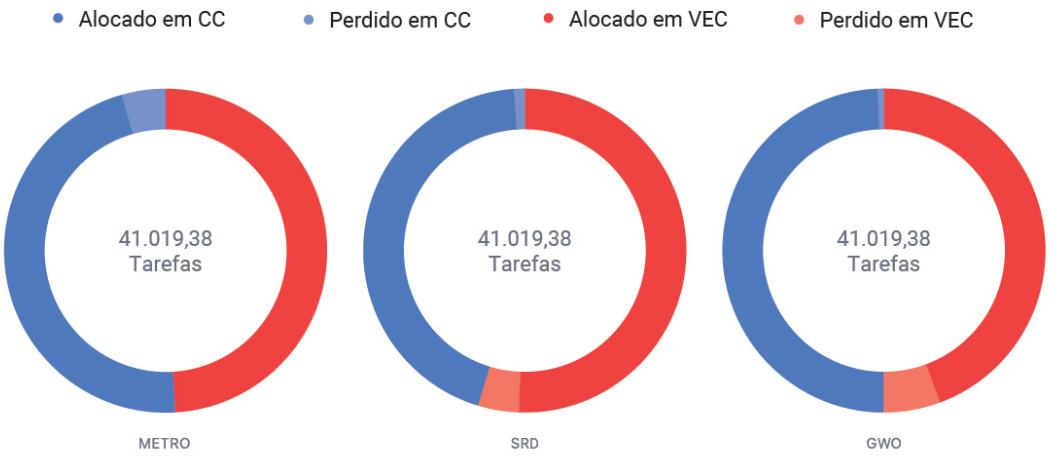


Fonte: Elaborado pelo autor, 2023.

Para concluir, a Figura 25 ilustra a distribuição das tarefas que foram alocadas e perdidas nas duas tecnologias existentes, VEC e CC à partir da média geral de tarefas geradas durante as execuções, que é de aproximadamente 41.019 tarefas. As cores mais escuras indicam as tarefas alocadas e as mais claras são as tarefas perdidas. A perda média do algoritmo METRO em relação a CC foi de aproximadamente 1798 tarefas e nenhuma perda em relação a VEC. Já o algoritmo SRD perdeu em média

1682 tarefas em VEC, um número alta considerando a criticidade dessas solicitações, enquanto em CC foi de aproximadamente 440 tarefas. Por fim, o algoritmo GWO foi o que mais teve perda em relação a VEC, sendo aproximadamente 2349 tarefas, enquanto em CC a perda foi de aproximadamente 254 tarefas.

Figura 25: Média do Total de Tarefas Alocas em VEC e CC



Fonte: Elaborado pelo autor, 2023.

5.6 Considerações Parciais

À partir das métricas de avaliações definidas na Seção 5.4 e dos resultados apresentados na Seção 5.5, é possível observar que, para está arquitetura e cenário em específico, o algoritmo METRO conseguiu um melhor desempenho em relação ao SRD e GWO, principalmente para a alocação de tarefas de maior prioridade, em que o mesmo obteve 100% de aproveitamento nesses casos, já que a média das execuções trouxeram um total de 0 perdas de tarefas em VEC. Dessa forma, foi possível comprovar a efetividade da GT, mais especificamente do uso de NE, que trabalha com o conceitos de jogos cooperativos nesses cenários de criticidade.

Outro fator relevante a se considerar é que, por mais que o desempenho dos demais algoritmos foi superior na questão de alocação de recursos em CC, há uma grande

disparidade na relação alocação VEC x CC por parte dos algoritmos SRD e GWO, gerando assim um maior congestionamento da rede, já que aumenta a demanda em determinada infraestrutura e gera ociosidade na outra. Além disso, o número de perdas média das tarefas em VEC dos algoritmos SRD e GWO foram de respectivamente, 1682 e 2349 tarefas, um número extremamente alto quando se trata de tarefas complexas e de alta criticidade.

Enfim, resumidamente é possível afirmar através dos resultados apresentados que no geral e principalmente levando em consideração ao atendimento de solicitações sensíveis, o gerenciamento da decisão de alocação dos recursos por parte do algoritmo METRO foi superior em relação aos demais algoritmos.

Capítulo 6

Conclusão

A partir da SLR, é perceptível a necessidade de novas propostas para a resolução dos problemas relacionados ao CO e o processo de decisão de alocação de recursos em cenários de redes IoV. Para isso, é extremamente importante explorar as soluções já existentes, pois resolvem diversos problemas, entretanto, faz-se necessário acompanhar as soluções emergentes, como é o caso respectivamente das redes móveis 5G e 6G que trazem propostas inovadoras para solucionar os diversos desafios descritos ao longo deste trabalho e possibilitam a utilização de novas tecnologias. Uma dessas inovações é a VEC, que é responsável por melhorar as interações existentes nesses cenários, principalmente na comunicação V2X, introduzida em redes IoV. No entanto além de uma arquitetura eficiente, como a arquitetura de três camadas, é necessário uma orquestração decisiva para gerenciar toda a alocação de recursos desse cenário.

Nesse sentido, o Algoritmo METRO presente neste trabalho, faz o uso da GT, com a utilização de conceitos de NE, uma espécie de jogo cooperativo, em que os jogadores se interagem entre si, com o uso de estratégias, resultando assim na otimização da alocação das tarefas. O algoritmo METRO foi aplicado em um cenário com 3 variações diferentes de mobilidade, a média desses resultados foi exposta em gráficos que revelaram um melhor desempenho comparado aos outros algoritmos executados no mesmo cenário com as mesmas condições, SRD e GWO, principalmente para alocação de solicitações críticas, ou seja, tarefas sensíveis de alta complexidade que precisam ser

processadas com o mínimo de latência possível.

6.1 Principais Contribuições

As principais contribuições obtidas por meio do desenvolvimento deste projeto de mestrado que resultou a elaboração do algoritmo METRO foram:

- Resolução dos problemas de alta latência em aplicativos que são extremamente sensíveis a atrasos;
- Contribuição para o processo de mobilidade, possibilitando a comunicação de dispositivos heterogêneos;
- Redução significativa dos problemas de congestionamento de rede;
- Otimização do processo de orquestração de alocação de recursos em comunicação V2X em Redes IoV; e
- Obtenção de um resultado satisfatório em relação a outras técnicas de alocação de recursos tradicionais e meta-heurísticas.

6.2 Trabalhos Futuros

Para trabalhos futuros, pretende-se utilizar mais métricas de avaliações, além de outros algoritmos tradicionais e meta-heurísticos, com o intuito de obter mais resultados para a comparação de desempenho. Dispor de outros mapas para o processo de simulação, com a utilização de trechos de cidades reais, por exemplo, com maior número de densidade de nós envolvidos na execução das simulações. Para concluir, espera-se melhorar o processo de decisão para otimizar ainda mais a alocação de recursos em CC.

Referências

ANG, L.-M.; SENG, K. P.; IJEMARU, G. K.; ZUNGERU, A. M. Deployment of iov for smart cities: Applications, architecture, and challenges. *IEEE Access*, v. 7, p. 6473–6492, 2019.

ARALDO, A.; STEFANO, A. D. Resource allocation for edge computing with multiple tenant configurations. In: . New York, NY, USA: Association for Computing Machinery, 2020. (SAC '20), p. 1190–1199. ISBN 9781450368667. Disponível em: https://doi.org/10.1145/3341105.3374026.

ARTERY. *Artery*. 2023. Disponível em: http://artery.v2x-research.eu/.

AVGERIS, M.; DECHOUNIOTIS, D.; ATHANASOPOULOS, N.; PAPAVASSILIOU, S. Adaptive resource allocation for computation offloading: A control-theoretic approach. Association for Computing Machinery, New York, NY, USA, v. 19, n. 2, apr 2019. ISSN 1533-5399. Disponível em: https://doi.org/10.1145/3284553.

BAHAMOU, S.; OUADGHIRI, M. D. E.; BONNIN, J.-M. When game theory meets vanet's security and privacy. In: *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media*. New York, NY, USA: Association for Computing Machinery, 2016. (MoMM '16), p. 292–297. ISBN 9781450348065. Disponível em: https://doi.org/10.1145/3007120.3007168.

BORCOCI, E. *Edge Computing Support for V2X Communications on 5G Technology*. 2021. Disponível em: https://www.iaria.org/conferences2021/filesAICT21/EugenBorcoci_Keynote_EdgeComputingSuport.pdf.

BUI, K.-H. N.; JUNG, J. J. Cooperative game theoretic approach for distributed resource allocation in heterogeneous network. In: *2017 International Conference on Intelligent Environments (IE)*. [S.l.: s.n.], 2017. p. 168–171.

DAI, Y.; XU, D.; MAHARJAN, S.; ZHANG, Y. Joint offloading and resource allocation in vehicular edge computing and networks. In: *2018 IEEE Global Communications Conference (GLOBECOM)*. [S.l.: s.n.], 2018. p. 1–7.

DAS, R.; GOSWAMI, S.; KONAR, A. Relationship between nash equilibria and pareto optimal solutions for games of pure coordination. In: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. [S.l.: s.n.], 2019. p. 1–7.

DUREJA, A.; SANGWAN, S. A review: Efficient transportation future aspects of iov. In: SINGH, P. K.; NOOR, A.; KOLEKAR, M. H.; TANWAR, S.; BHATNAGAR, R. K.; KHANNA, S. (Ed.). *Evolving Technologies for Computing, Communication and Smart World*. Singapore: Springer Singapore, 2021. p. 97–108. ISBN 978-981-15-7804-5.

EL-SAYED, H.; CHAQFEH, M. Exploiting mobile edge computing for enhancing vehicular applications in smart cities. *Sensors*, v. 19, n. 5, 2019. ISSN 1424-8220. Disponível em: https://www.mdpi.com/1424-8220/19/5/1073.

GARG, T.; KAGALWALLA, N.; CHURI, P.; PAWAR, D. A.; DESHMUKH, S. A survey on security and privacy issues in iov. *International Journal of Electrical and Computer Engineering (IJECE)*, v. 10, p. 5409, 10 2020.

GU, X.; ZHANG, G.; CAO, Y. Cooperative mobile edge computing-cloud computing in internet of vehicle: Architecture and energy-efficient workload allocation. *Transactions on Emerging Telecommunications Technologies*, v. 32, p. 1–21, 2021. ISSN 21613915.

HAMMOUD, A.; SAMI, H.; MOURAD, A.; OTROK, H.; MIZOUNI, R.; BENTAHAR, J. Ai, blockchain, and vehicular edge computing for smart and secure iov: Challenges and directions. *IEEE Internet of Things Magazine*, v. 3, n. 2, p. 68–73, 2020.

HOU, X.; REN, Z.; WANG, J.; CHENG, W.; REN, Y.; CHEN, K.-C.; ZHANG, H. Reliable computation offloading for edge computing-enabled software-defined iov. *IEEE Internet of Things Journal*, 03 2020.

HU, J.; LI, K.; LIU, C.; LI, K. Game-based task offloading of multiple mobile devices with qos in mobile edge computing systems of limited computation capacity. *ACM Trans. Embed. Comput. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 19, n. 4, jul 2020. ISSN 1539-9087. Disponível em: https://doi.org/10.1145/3398038.

JHA, A. A.; JAIN, S.; THENMALAR, S. Survey on grey wolf algorithm in resource allocation. *International Journal of Pure and Applied Mathematics - Special Issue*, v. 118, p. 201 – 206, 2018. ISSN 1314-3395 (on-line version).

JI, B.; ZHANG, X.; MUMTAZ, S.; HAN, C.; LI, C.; WEN, H.; WANG, D. Survey on the internet of vehicles: Network architectures and applications. *IEEE Communications Standards Magazine*, v. 4, p. 34–41, 03 2020.

KARP, R. M. An introduction to randomized algorithms. *Discrete Applied Mathematics*, v. 34, n. 1, p. 165–201, 1991. ISSN 0166-218X. Disponível em: https://www.sciencedirect.com/science/article/pii/0166218X9190086C.

KATOCH, S.; CHAUHAN, S. S.; KUMAR, V. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, Springer Science and Business

Media LLC, v. 80, n. 5, p. 8091–8126, oct 2020. Disponível em: https://doi.org/10.1007/s11042-020-10139-6.

KAVRE, M.; GADEKAR, A.; GADHADE, Y. Internet of things (iot): A survey. In: *2019 IEEE Pune Section International Conference (PuneCon)*. [S.l.: s.n.], 2019. p. 1–6.

KAZMI, S. M. A.; DANG, T. N.; YAQOOB, I.; MANZOOR, A.; HUSSAIN, R.; KHAN, A.; HONG, C. S.; SALAH, K. A novel contract theory-based incentive mechanism for cooperative task-offloading in electrical vehicular networks. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, 2020. ISSN 1524-9050.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele Univ.*, v. 33, 08 2004.

KOMBATE, D.; WANGLINA. The internet of vehicles based on 5g communications. In: *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. [S.l.: s.n.], 2016. p. 445–448.

KUMAR, P.; ARTI, S.; KOLEKAR, M. H.; TANWAR, S.; BHATNAGAR, R. K. *Evolving Technologies for Computing , Communication and Smart World*. [S.l.: s.n.], 2020. v. 1. 538 p. ISBN 9789811578038.

LIU, L.; CHEN, C.; PEI, Q.; MAHARJAN, S.; ZHANG, Y. Vehicular edge computing and networking: A survey. *Mobile Networks and Applications*, Mobile Networks and Applications, v. 26, n. 3, p. 1145–1168, 2021. ISSN 15728153.

MARKAKIS, N. M. D. G. M. D. C. M. E. P. K. *Intelligent Technologies for Internet of Vehicles*. [S.l.]: Springer, 2021. v. 1. 511 p. ISBN 978-3-030-76492-0.

MEHRABI, M.; YOU, D.; LATZKO, V.; SALAH, H.; REISSLEIN, M.; FITZEK, F. H. P. Device-enhanced mec: Multi-access edge computing (mec) aided by end device computation and caching: A survey. *IEEE Access*, v. 7, p. 166079 – 166108, 2019. Cited by: 77; All Open Access, Gold Open Access, Green Open Access. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077549605&doi=10.1109%2fACCESS.2019.2953172&partnerID=40&md5=766060482b3f5e0af656b35141a5eadd.

MENEGUETTE, R.; GRANDE, R. D.; UEYAMA, J.; FILHO, G. P. R.; MADEIRA, E. Vehicular edge computing: Architecture, resource management, security, and challenges. *ACM Comput. Surv.*, Association for Computing Machinery, v. 55, n. 1, 2021. ISSN 0360-0300.

MENEGUETTE, R. I.; BOUKERCHE, A. A cooperative and adaptive resource scheduling for vehicular cloud. In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.: s.n.], 2017. p. 398–403.

MENEGUETTE, R. I.; BOUKERCHE, A. Vehicular clouds leveraging mobile urban computing through resource discovery. *IEEE Transactions on Intelligent Transportation Systems*, v. 21, n. 6, p. 2640–2647, 2020.

Meneguette, R. I.; De Grande, R.; Loureiro, A. A. *Intelligent Transport System in Smart Cities*. [S.l.]: Springer, 2018. v. 1. 182 p. ISBN 978-3-319-93331-3.

MENEGUETTE, R. I.; MADEIRA, E. R. M.; BITTENCOURT, L. F. Multi-network packet scheduling based on vehicular ad hoc network applications. In: *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*. [S.l.: s.n.], 2012. p. 214–218.

MENSAH, R. N.; ZHIYUAN, L.; OKINE, A. A.; ADEKE, J. M. A game-theoretic approach to computation offloading in software-defined d2d-enabled vehicular networks. In: *2021 2nd Information Communication Technologies Conference (ICTC)*. [S.l.: s.n.], 2021. p. 34–38.

MIRJALILI, S.; MIRJALILI, S. M.; LEWIS, A. Grey wolf optimizer. *Advances in Engineering Software*, v. 69, p. 46–61, 2014. ISSN 0965-9978. Disponível em: https://www.sciencedirect.com/science/article/pii/S0965997813001853.

MOLLAH, M. B.; ZHAO, J.; NIYATO, D.; GUAN, Y. L.; YUEN, C.; SUN, S.; LAM, K.-Y.; KOH, L. H. Blockchain for the internet of vehicles towards intelligent transportation systems: A survey. *IEEE Internet of Things Journal*, v. 8, n. 6, p. 4157–4185, 2021.

NEUMANN, J. V. *Theory of Games and Economic Behavior*. [S.l.]: Princeton University Press, 1944. 776 p.

NGUYEN, T. D.; NGUYEN, T.-D.; NGUYEN, V. D.; PHAM, X.-Q.; HUH, E.-N. Cost-effective resource sharing in an internet of vehicles-employed mobile edge computing environment. *Symmetry*, v. 10, n. 11, 2018. Cited by: 16; All Open Access, Gold Open Access, Green Open Access. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85057813767&doi=10.3390%2fsym10110594&partnerID=40&md5=7d45a5e5b1d72653a18915e66d8f5c08.

OMNET++. *Objective Modular Network Testbed in C++ - OMNeT++*. 2022. Disponível em: https://omnetpp.org/.

PHAM, Q.-V.; FANG, F.; HA, V. N.; PIRAN, M. J.; LE, M.; LE, L. B.; HWANG, W.-J.; DING, Z. A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access*, v. 8, p. 116974 – 117017, 2020. Cited by: 204; All Open Access, Gold Open Access, Green Open Access. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087817962&doi=10.1109%2fACCESS.2020.3001277&partnerID=40&md5=28faffc3e6ce85dced46219ac7486e7b.

RAZA, S.; WANG, S.; AHMED, M.; ANWAR, M. R. Corrigendum: A survey on vehicular edge computing: Architecture, applications, technical issues, and future directions (wireless communications and mobile computing (2019) 2019 (19) doi: 10.1155/2019/3159762). *Wireless Communications and Mobile Computing*, v. 2019, 2019. ISSN 15308677.

S, M. H.; T, S. K.; MUSTAPHA, S. D. S.; GUPTA, P.; TRIPATHI, R. P. Hybrid approach for resource allocation in cloud infrastructure using random forest and genetic algorithm. *Scientific Programming*, Hindawi, v. 2021, p. 4924708, Oct 2021. ISSN 1058-9244. Disponível em: https://doi.org/10.1155/2021/4924708.

SAJJAD, I.; SHARMA, R.; GERDES, R. A game-theoretic approach and evaluation of adversarial vehicular platooning. In: . New York, NY, USA: Association for Computing Machinery, 2017. (SCAV'17), p. 35–41. ISBN 9781450349765. Disponível em: https://doi.org/10.1145/3055378.3055383.

SHAH, S. H.; YAQOOB, I. A survey: Internet of things (iot) technologies, applications and challenges. In: *2016 IEEE Smart Energy Grid Engineering (SEGE)*. [S.l.: s.n.], 2016. p. 381–385.

SHEN, B.; XU, X.; DAI, F.; QI, L.; ZHANG, X.; DOU, W. Dynamic task offloading with minority game for internet of vehicles in cloud-edge computing. In: *2020 IEEE International Conference on Web Services (ICWS)*. [S.l.: s.n.], 2020. p. 372–379.

SRIVASTAVA, S.; SAHANA, S. K. Application of bat algorithm for transport network design problem. *Applied Computational Intelligence and Soft Computing*, Hindawi, v. 2019, p. 9864090, Jan 2019. ISSN 1687-9724. Disponível em: https://doi.org/10.1155/2019/9864090.

SUMO. *Simulation of Urban Mobility - SUMO*. 2022. Disponível em: https://sumo.dlr.de/docs/index.html.

TANG, C.; ZHU, C.; WU, H.; WEI, X.; LI, Q.; RODRIGUES, J. J. P. C. A game theoretical pricing scheme for vehicles in vehicular edge computing. In: *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*. [S.l.: s.n.], 2020. p. 17–22.

WANG, H.; LIN, Z.; GUO, K.; LV, T. Computation offloading based on game theory in mec-assisted v2x networks. In: *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. [S.l.: s.n.], 2021. p. 1–6.

WANG, J.; ZHU, K.; HOSSAIN, E. Green internet of vehicles (iov) in the 6g era: Toward sustainable vehicular communications and networking. *IEEE Transactions on Green Communications and Networking*, p. 1–1, 2021.

WANG, Z.; ZHENG, S.; GE, Q.; LI, K. Online offloading scheduling and resource allocation algorithms for vehicular edge computing system. *IEEE Access*, v. 8, p. 52428–52442, 2020.

XAVIER, F. C. [s.d.].

XIAO, Z.; DAI, X.; JIANG, H.; WANG, D.; CHEN, H.; YANG, L.; ZENG, F. Vehicular task offloading via heat-aware mec cooperation using game-theoretic method. *IEEE INTERNET OF THINGS JOURNAL*, v. 7, n. 3, p. 2038–2052, MAR 2020. ISSN 2327-4662.

YANG, X.-S. A new metaheuristic bat-inspired algorithm. In: ____. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 65–74. ISBN 978-3-642-12538-6. Disponível em: https://doi.org/10.1007/978-3-642-12538-6_6.

YANG, X.-S. Bat algorithm: Literature review and applications. *International Journal of Bio-Inspired Computation*, v. 5, 08 2013.

ZAFARI, F.; LI, J.; LEUNG, K. K.; TOWSLEY, D.; SWAMI, A. A game-theoretic approach to multi-objective resource sharing and allocation in mobile edge. In: *Proceedings of the 2018 on Technologies for the Wireless Edge Workshop*. New York, NY, USA: Association for Computing Machinery, 2018. (WirelessEdge '18), p. 9–13. ISBN 9781450359313. Disponível em: [https://doi.org/10.1145/3266276.3266277] (https://doi.org/10.1145/3266276.3266277).

ZENG, F.; CHEN, Q.; MENG, L.; WU, J. Volunteer assisted collaborative offloading and resource allocation in vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems*, v. 22, n. 6, p. 3247–3257, 2021.

ZHANG, H.; WANG, Z.; LIU, K. V2x offloading and resource allocation in sdn-assisted mec-based vehicular networks. *China Communications*, v. 17, n. 5, p. 266–283, 2020.

ZHANG, K.; MAO, Y.; LENG, S.; MAHARJAN, S.; VINEL, A.; ZHANG, Y. Contract-theoretic approach for delay constrained offloading in vehicular edge computing networks. *Mobile Networks and Applications*, Mobile Networks and Applications, v. 24, n. 3, p. 1003–1014, 2019. ISSN 15728153.

ZHANG, Y. Mobile edge computing for the internet of vehicles. In: ____. *Mobile Edge Computing*. Cham: Springer International Publishing, 2022. p. 47–64. ISBN 978-3-030-83944-4. Disponível em: https://doi.org/10.1007/978-3-030-83944-4_5.

ZHANG, Y.; ZHANG, M.; FAN, C.; LI, F.; LI, B. Computing resource allocation scheme of iov using deep reinforcement learning in edge computing environment. *EURASIP Journal on Advances in Signal Processing*, v. 2021, n. 1, p. 33, 2021.

ZHAO, C.; LU, X.; ZHIYING, M. A multi-target detection task allocation algorithm based on game theory. In: . New York, NY, USA: Association for Computing Machinery, 2020. (ICASIT 2020), p. 470–475. ISBN 9781450375764. Disponível em: https://doi.org/10.1145/3434581.3434640.

ZHOU, S.; JADOON, W. The partial computation offloading strategy based on game theory for multi-user in mobile edge computing environment. *Comput. Netw.*, Elsevier North-Holland, Inc., USA, v. 178, n. C, sep 2020. ISSN 1389-1286. Disponível em: https://doi.org/10.1016/j.comnet.2020.107334.

Apêndice A

Código da Estrutura de Alocação de Recursos em C++

```
1  #include <iostream>
2  #include <fstream>
3  #include <random>
4  #include <string>
5  #include <vector>
6  #include <sstream>
7
8  #define ONE 1
9  #define TEEN 10
10 #define FIFTY 50
11 #define HUNDRED 100
12 #define THOUSAND 1000
13
14 enum COMPUTING_TYPE { VEC, CLOUD };
15
16 struct TaskInput {
17     int id;
18     std::string nodeId;
19     int process;
20     int memory;
21     int storage;
22     int priority;
23 };
```

```
24
25 struct TaskAllocated {
26     int id;
27     std::string nodeId;
28     COMPUTING_TYPE compType;
29     int priority;
30 };
31
32 struct TaskMissed {
33     int id;
34     std::string nodeId;
35     COMPUTING_TYPE compType;
36     int priority;
37 };
38
39 struct VecResource {
40     int totalProcess;
41     int totalMemory;
42     int totalStorage;
43     int nodes;
44 };
45
46 struct CloudResource {
47     int totalProcess;
48     int totalMemory;
49     int totalStorage;
50     int nodes;
51 };
52
53 int VEC_NODES = FIFTY;
54 int CLOUD_NODES = (ONE * 3);
55 struct VecResource vecResource = {(FIFTY * VEC_NODES), (
      FIFTY * VEC_NODES), (FIFTY * VEC_NODES), FIFTY};
56 struct CloudResource cloudResource = {(THOUSAND *
      CLOUD_NODES), (THOUSAND * CLOUD_NODES), (THOUSAND *
      CLOUD_NODES), (ONE * 3)};
```

```
57 vector<TaskInput> taskInputArray;
58 vector<TaskAllocated> taskAllocatedArray;
59 vector<TaskMissed> taskMissedArray;
60 int taskId = 0;
61 int rowCounter = 0;
62
63 TaskInput generateCSVTask(std::string nodeId)
64 {
65     rowCounter++;
66     std::string filePath = "tasks_input.csv";
67     std::ifstream file(filePath);
68     if (file.is_open()) {
69         std::string line;
70         std::getline(file, line);
71         for (int i = 1; i < rowCounter; ++i) {
72             std::getline(file, line);
73         }
74         std::getline(file, line);
75         std::istringstream ss(line);
76         std::string token;
77         int id, process, memory, storage, priority;
78         std::getline(ss, token, ','); id = std::stoi(
              token);
79         std::getline(ss, token, ','); process = std::stoi
              (token);
80         std::getline(ss, token, ','); memory = std::stoi(
              token);
81         std::getline(ss, token, ','); storage = std::stoi
              (token);
82         std::getline(ss, token, ','); priority = std::
              stoi(token);
83         file.close();
84         struct TaskInput randomTask = {id, nodeId,
              process, memory, storage, priority};
85         return randomTask;
86     } else {
```

```
87          std::cerr << "Unable to open file." << std::endl;
88      }
89  }
90
91  TaskInput* findTaskInputByNodeId(const std::string&
        nodeId)
92  {
93      for (auto& task : taskInputArray) {
94          if (task.nodeId == nodeId) {
95              return &task;
96          }
97      }
98      return nullptr;
99  }
100
101 TaskAllocated* findTaskAllocatedByTaskInput(TaskInput*
        taskInput)
102 {
103     for (auto& task : taskAllocatedArray) {
104         if (task.id == taskInput->id && task.nodeId ==
                taskInput->nodeId) {
105             return &task;
106         }
107     }
108     return nullptr;
109 }
110
111 TaskAllocated* findTaskAllocatedInVECByPriority(int
        priority)
112 {
113     for (auto& task : taskAllocatedArray) {
114         if (task.priority == priority && task.compType ==
                COMPUTING_TYPE::VEC) {
115             return &task;
116         }
117     }
```

```
118     return nullptr;
119 }
120
121 float calculateTasksPriorityAverage()
122 {
123     if(taskAllocatedArray.size() <= 0) {
124         return 0;
125     }
126     int sum = 0;
127     for (auto& task : taskAllocatedArray) {
128         sum += task.priority;
129     }
130     return (sum / taskAllocatedArray.size());
131 }
132
133 void releaseResources(TaskInput* taskInput)
134 {
135     if(taskInput) {
136         TaskAllocated* foundTaskAllocated =
              findTaskAllocatedByTaskInput(taskInput);
137         if(foundTaskAllocated) {
138             if (foundTaskAllocated->compType ==
                  COMPUTING_TYPE::VEC) {
139                 vecResource.totalProcess += taskInput->
                      process;
140                 vecResource.totalMemory += taskInput->
                      memory;
141                 vecResource.totalStorage += taskInput->
                      storage;
142             } else {
143                 cloudResource.totalProcess += taskInput->
                      process;
144                 cloudResource.totalMemory += taskInput->
                      memory;
145                 cloudResource.totalStorage += taskInput->
                      storage;
```

```
146             }
147         }
148     }
149 }
150
151 void consumeResources(COMPUTING_TYPE compType, TaskInput*
        taskInput)
152 {
153     if(taskInput) {
154         if (compType == COMPUTING_TYPE::VEC) {
155             vecResource.totalProcess -= taskInput->
                process;
156             vecResource.totalMemory -= taskInput->memory;
157             vecResource.totalStorage -= taskInput->
                storage;
158         } else {
159             cloudResource.totalProcess -= taskInput->
                process;
160             cloudResource.totalMemory -= taskInput->
                memory;
161             cloudResource.totalStorage -= taskInput->
                storage;
162         }
163     }
164 }
165
166 bool haveResourcesToAllocate(COMPUTING_TYPE compType,
       TaskInput* taskInput)
167 {
168     if (compType == COMPUTING_TYPE::VEC) {
169         if (taskInput->process <= vecResource.
            totalProcess && taskInput->memory <=
            vecResource.totalMemory && taskInput->storage
            <= vecResource.totalStorage) {
170             return true;
171         } else {
```

```
172             return false;
173         }
174     } else {
175         if (taskInput->process <= cloudResource.
               totalProcess && taskInput->memory <=
               cloudResource.totalMemory && taskInput->
               storage <= cloudResource.totalStorage) {
176             return true;
177         } else {
178             return false;
179         }
180     }
181     return false;
182 }
183
184 void allocateTask(COMPUTING_TYPE compType, std::string
       nodeId, TaskInput* taskInput)
185 {
186     consumeResources(compType, taskInput);
187     taskAllocatedArray.push_back({taskInput->id, nodeId,
           compType, taskInput->priority});
188 }
189
190 void reallocatedTaskFromVECToCLOUD(TaskAllocated*
       taskAllocated)
191 {
192     TaskInput* taskInput = findTaskInputByNodeId(
           taskAllocated->nodeId);
193     releaseResources(taskInput);
194     if(haveResourcesToAllocate(COMPUTING_TYPE::CLOUD,
           taskInput)) {
195         consumeResources(COMPUTING_TYPE::CLOUD, taskInput
               );
196     } else {
197         taskMissedArray.push_back({taskInput->id,
               taskAllocated->nodeId, COMPUTING_TYPE::CLOUD,
```

```
            taskInput->priority});
198     }
199 }
200
201 void deallocateTask(TaskInput* taskInput)
202 {
203     if(taskInput) {
204         releaseResources(taskInput);
205     }
206 }
```

Apêndice B

Código do Algoritmo METRO em C++

```
1  bool metroVECStrategyIsWork(TaskInput* taskInput)
2  {
3      if(haveResourcesToAllocate(COMPUTING_TYPE::VEC,
          taskInput)) {
4          return true;
5      } else {
6          return false;
7      }
8  }
9
10 bool metroCLOUDStrategyIsWork(TaskInput* taskInput)
11 {
12     if(haveResourcesToAllocate(COMPUTING_TYPE::CLOUD,
          taskInput)) {
13         return true;
14     } else {
15         return false;
16     }
17 }
18
19 void metroCalculateNashEquilibrium(std::string nodeId,
      TaskInput* taskInput)
20 {
```

```
21      bool vecStrategyIsWork = metroVECStrategyIsWork(
           taskInput);
22      bool cloudStrategyIsWork = metroCLOUDStrategyIsWork(
           taskInput);
23      if(taskInput->priority == 5) {
24          float tasksPriorityAverage =
              calculateTasksPriorityAverage();
25          if(tasksPriorityAverage < 5 && vecStrategyIsWork)
               {
26              allocateTask(COMPUTING_TYPE::VEC, nodeId,
                  taskInput);
27          } else {
28              if(cloudStrategyIsWork) {
29                  allocateTask(COMPUTING_TYPE::CLOUD,
                      nodeId, taskInput);
30              } else {
31                  taskMissedArray.push_back({taskInput->id,
                       nodeId, COMPUTING_TYPE::CLOUD,
                      taskInput->priority});
32              }
33          }
34      } else if(taskInput->priority > 5) {
35          if(vecStrategyIsWork) {
36              allocateTask(COMPUTING_TYPE::VEC, nodeId,
                  taskInput);
37          } else {
38              int priorityToSearch = 5;
39              bool isAllocated = false;
40              while(!isAllocated) {
41                  TaskAllocated*
                      foundTaskAllocatedByPriority =
                      findTaskAllocatedInVECByPriority(
                      priorityToSearch);
42                  if(foundTaskAllocatedByPriority && !
                      isAllocated) {
43                      reallocatedTaskFromVECToCLOUD(
```

```
                        foundTaskAllocatedByPriority);
44                      bool newVecStrategyIsWork =
                        metroVECStrategyIsWork(taskInput);
45                      if(newVecStrategyIsWork && !
                        isAllocated) {
46                          allocateTask(COMPUTING_TYPE::VEC,
                                nodeId, taskInput);
47                          isAllocated = true;
48                      }
49                  }
50                  else {
51                      taskMissedArray.push_back({taskInput
                        ->id, nodeId, COMPUTING_TYPE::VEC,
                         taskInput->priority});
52                      break;
53                  }
54              }
55          }
56      } else if(taskInput->priority < 5) {
57          if(cloudStrategyIsWork) {
58              allocateTask(COMPUTING_TYPE::CLOUD, nodeId,
                    taskInput);
59          } else {
60              taskMissedArray.push_back({taskInput->id,
                    nodeId, COMPUTING_TYPE::CLOUD, taskInput->
                    priority});
61          }
62      }
63  }
64
65  void metroAlgorithm(std::string nodeId, TaskInput*
        taskInput)
66  {
67      metroCalculateNashEquilibrium(nodeId, taskInput);
68  }
69
```

```
70 void resourceAllocation(std::string nodeId, TaskInput*
      taskInput)
71 {
72     metroAlgorithm(nodeId, taskInput);
73 }
```

Apêndice C

Código do Algoritmo SRD em C++

```
1  std::random_device randomDevice;
2  std::mt19937 generator(randomDevice());
3  std::uniform_int_distribution<> disBetweenZeroAndOne(0,
       1);
4
5  int randomAllocatedInVEC()
6  {
7      return disBetweenZeroAndOne(generator) == 1;
8  }
9
10 void simplePriorityDecision(std::string nodeId, TaskInput
       * taskInput)
11 {
12     bool haveResourceInVEC = haveResourcesToAllocate(
           COMPUTING_TYPE::VEC, taskInput);
13     bool haveResourceInCLOUD = haveResourcesToAllocate(
           COMPUTING_TYPE::CLOUD, taskInput);
14     if(taskInput->priority == 5) {
15         bool allocateInVEC = randomAllocatedInVEC();
16         if(allocateInVEC && haveResourceInVEC) {
17             allocateTask(COMPUTING_TYPE::VEC, nodeId,
                   taskInput);
18         } else {
19             if(haveResourceInCLOUD) {
20                 allocateTask(COMPUTING_TYPE::CLOUD,
```

```
                    nodeId, taskInput);
21              } else {
22                  taskMissedArray.push_back({taskInput->id,
                        nodeId, COMPUTING_TYPE::CLOUD,
                        taskInput->priority});
23              }
24          }
25      } else if(taskInput->priority > 5) {
26          if(haveResourceInVEC) {
27              allocateTask(COMPUTING_TYPE::VEC, nodeId,
                    taskInput);
28          } else {
29              taskMissedArray.push_back({taskInput->id,
                    nodeId, COMPUTING_TYPE::VEC, taskInput->
                    priority});
30          }
31      } else if(taskInput->priority < 5) {
32          if(haveResourceInCLOUD) {
33              allocateTask(COMPUTING_TYPE::CLOUD, nodeId,
                    taskInput);
34          } else {
35              taskMissedArray.push_back({taskInput->id,
                    nodeId, COMPUTING_TYPE::CLOUD, taskInput->
                    priority});
36          }
37      }
38  }
39
40  void simplePriorityAlgorithm(std::string nodeId,
        TaskInput* taskInput)
41  {
42      simplePriorityDecision(nodeId, taskInput);
43  }
44
45  void resourceAllocation(std::string nodeId, TaskInput*
        taskInput)
```

```
46 {
47     simplePriorityAlgorithm(nodeId, taskInput);
48 }
```

Apêndice D

Código do Algoritmo GWO em C++

```
1  enum WOLF_TYPE { ALPHA, BETA, DELTA, OMEGA };
2
3  struct Wolf {
4      WOLF_TYPE wolfType;
5      int position;
6  };
7
8  Wolf alphaWolf = { WOLF_TYPE::ALPHA, 10 };
9  Wolf betaWolf = { WOLF_TYPE::BETA, 5};
10 Wolf deltaWolf = { WOLF_TYPE::DELTA, 4};
11 Wolf omegaWolf = { WOLF_TYPE::OMEGA, 0};
12
13 int objectiveFunction()
14 {
15     int fitness = 0;
16     deltaWolf.position, omegaWolf.position;
17     if(
18         alphaWolf.position > betaWolf.position &&
19         alphaWolf.position > deltaWolf.position &&
20         alphaWolf.position > omegaWolf.position
21     ) {
22         fitness = 10;
23     } else if(
24         betaWolf.position > alphaWolf.position &&
25         betaWolf.position > deltaWolf.position &&
```

```
26          betaWolf.position > omegaWolf.position
27      ) {
28          fitness = 7;
29      } else if(
30          deltaWolf.position > alphaWolf.position &&
31          deltaWolf.position > betaWolf.position &&
32          deltaWolf.position > omegaWolf.position
33      ) {
34          fitness = 4;
35      } else if(
36          omegaWolf.position > alphaWolf.position &&
37          omegaWolf.position > betaWolf.position &&
38          omegaWolf.position > deltaWolf.position
39      ) {
40          fitness = 1;
41      }
42      return fitness;
43  }
44
45  double calculateRandom(int taskPriority)
46  {
47      return (rand() * taskPriority) / (double)RAND_MAX;
48  }
49
50  double calculateA(double a, double r1)
51  {
52      return 2.0 * a * r1 - a;
53  }
54
55  double calculateC(double r2)
56  {
57      return 2.0 * r2;
58  }
59
60  bool gwoDecision(TaskInput* taskInput)
61  {
```

```
62      int fitness;
63      int MAX_ITERATIONS = taskAllocatedArray.size();
64      for(int i = 0; i < MAX_ITERATIONS; i++) {
65          double a = 2.0 - i * (2.0 / MAX_ITERATIONS);
66          double r1 = calculateRandom(taskInput->priority);
67          double r2 = calculateRandom(taskInput->priority);
68          double A1 = calculateA(a, r1);
69          double C1 = calculateC(r2);
70          double D_alpha = std::abs(C1 * alphaWolf.position
                - omegaWolf.position);
71          double X1 = alphaWolf.position - A1 * D_alpha;
72          double A2 = calculateA(a, r1);
73          double C2 = calculateC(r2);
74          double D_beta = std::abs(C2 * betaWolf.position -
                omegaWolf.position);
75          double X2 = betaWolf.position - A2 * D_beta;
76          double A3 = calculateA(a, r1);
77          double C3 = calculateC(r2);
78          double D_delta = std::abs(C3 * deltaWolf.position
                - omegaWolf.position);
79          double X3 = deltaWolf.position - A3 * D_delta;
80          omegaWolf.position = (X1 + X2 + X3) / 3.0;
81          fitness = objectiveFunction();
82      }
83      if(fitness > 5 && taskInput->priority > 5) {
84          return true;
85      } else {
86          return false;
87      }
88  }
89
90  void gwoAlgorithm(std::string nodeId, TaskInput*
        taskInput)
91  {
92      bool haveResourceInVEC = haveResourcesToAllocate(
            COMPUTING_TYPE::VEC, taskInput);
```

```
93      bool haveResourceInCLOUD = haveResourcesToAllocate(
           COMPUTING_TYPE::CLOUD, taskInput);
94      if(taskInput->priority == 5) {
95          bool isWolfAlpha = gwoDecision(taskInput);
96          if(isWolfAlpha && haveResourceInVEC) {
97              allocateTask(COMPUTING_TYPE::VEC, nodeId,
                   taskInput);
98          } else {
99              if(haveResourceInCLOUD) {
100                 allocateTask(COMPUTING_TYPE::CLOUD,
                       nodeId, taskInput);
101             } else {
102                 taskMissedArray.push_back({taskInput->id,
                        nodeId, COMPUTING_TYPE::CLOUD,
                       taskInput->priority});
103             }
104         }
105     } else if(taskInput->priority > 5) {
106         if(haveResourceInVEC) {
107             allocateTask(COMPUTING_TYPE::VEC, nodeId,
                   taskInput);
108         } else {
109             bool isWolfAlpha = gwoDecision(taskInput);
110             if(isWolfAlpha) {
111                 int priorityToSearch = 5;
112                 TaskAllocated*
                       foundTaskAllocatedByPriority =
                       findTaskAllocatedInVECByPriority(
                       priorityToSearch);
113                 if(foundTaskAllocatedByPriority) {
114                     reallocatedTaskFromVECToCLOUD(
                           foundTaskAllocatedByPriority);
115                     haveResourceInVEC =
                           haveResourcesToAllocate(
                           COMPUTING_TYPE::VEC, taskInput);
116                     if(haveResourceInVEC) {
```

```
117                         allocateTask(COMPUTING_TYPE::VEC,
                                nodeId, taskInput);
118                     } else {
119                         taskMissedArray.push_back({
                               taskInput->id, nodeId,
                               COMPUTING_TYPE::VEC, taskInput
                               ->priority});
120                     }
121                 } else {
122                     taskMissedArray.push_back({taskInput
                           ->id, nodeId, COMPUTING_TYPE::VEC,
                            taskInput->priority});
123                 }
124             } else {
125                 taskMissedArray.push_back({taskInput->id,
                        nodeId, COMPUTING_TYPE::VEC,
                       taskInput->priority});
126             }
127         }
128     } else if(taskInput->priority < 5) {
129         if(haveResourceInCLOUD) {
130             allocateTask(COMPUTING_TYPE::CLOUD, nodeId,
                   taskInput);
131         } else {
132             taskMissedArray.push_back({taskInput->id,
                   nodeId, COMPUTING_TYPE::CLOUD, taskInput->
                   priority});
133         }
134     }
135 }
136
137 void resourceAllocation(std::string nodeId, TaskInput*
      taskInput)
138 {
139     gwoAlgorithm(nodeId, taskInput);
140 }
```

Anexo A

Código do Artery: Arquivo *Middleware*.h

```
1  /*
2  * Artery V2X Simulation Framework
3  * Copyright 2014-2019 Raphael Riebl et al.
4  * Licensed under GPLv2, see COPYING file for detailed
      license and warranty terms.
5  */
6
7  #ifndef ARTERY_MIDDLEWARE_H_
8  #define ARTERY_MIDDLEWARE_H_
9
10 #include "artery/application/Facilities.h"
11 #include "artery/application/LocalDynamicMap.h"
12 #include "artery/application/MultiChannelPolicy.h"
13 #include "artery/application/NetworkInterface.h"
14 #include "artery/application/NetworkInterfaceTable.h"
15 #include "artery/application/StationType.h"
16 #include "artery/application/Timer.h"
17 #include "artery/application/TransportDispatcher.h"
18 #include "artery/utility/Identity.h"
19 #include <omnetpp/clistener.h>
20 #include <omnetpp/csimplemodule.h>
21 #include <omnetpp/simtime.h>
22 #include <vanetza/btp/data_request.hpp>
```

```
23 #include <vanetza/btp/port_dispatcher.hpp>
24 #include <memory>
25 #include <set>
26
27 namespace artery
28 {
29
30 // forward declarations
31 class ItsG5BaseService;
32 class Router;
33
34 /**
35  * Middleware providing a runtime context for services.
36  */
37 class Middleware : public omnetpp::cSimpleModule, public
      omnetpp::cListener
38 {
39     public:
40         Middleware();
41         ~Middleware();
42
43         Facilities& getFacilities() { return mFacilities;
              }
44         const Facilities& getFacilities() const { return
             mFacilities; }
45
46         const Identity& getIdentity() const { return
             mIdentity; }
47         const StationType& getStationType() const {
             return mStationType; }
48         const TransportDispatcher& getTransportDispatcher
             () const { return mTransportDispatcher; }
49
50         /**
51          * Register a network interface at middleware's
              network interface table.
```

```
52          * Only previously registered interfaces are
               recognised by services.
53          *
54          * \param ifc network interface
55          */
56         void registerNetworkInterface(std::shared_ptr<
              NetworkInterface>);
57
58         void requestTransmission(const vanetza::btp::
              DataRequestB&, std::unique_ptr<vanetza::
              DownPacket>);
59         void requestTransmission(const vanetza::btp::
              DataRequestB&, std::unique_ptr<vanetza::
              DownPacket>, const NetworkInterface&);
60
61     protected:
62         // cSimpleModule
63         int numInitStages() const override;
64         void initialize(int stage) override;
65         void finish() override;
66         void handleMessage(omnetpp::cMessage* msg)
              override;
67
68         // cListener
69         void receiveSignal(omnetpp::cComponent*, omnetpp
              ::simsignal_t, long, omnetpp::cObject*)
              override;
70
71         omnetpp::cModule* findHost();
72         void setStationType(const StationType&);
73
74     private:
75         void updateServices();
76         void initializeServices(int stage);
77
78         omnetpp::SimTime mUpdateInterval;
```

```
79          omnetpp::cMessage* mUpdateMessage = nullptr;
80          Timer mTimer;
81          Identity mIdentity;
82          LocalDynamicMap mLocalDynamicMap;
83          Facilities mFacilities;
84          StationType mStationType;
85
86          NetworkInterfaceTable mNetworkInterfaceTable;
87          TransportDispatcher mTransportDispatcher;
88          std::unique_ptr<MultiChannelPolicy>
               mMultiChannelPolicy;
89          std::set<ItsG5BaseService*> mServices;
90  };
91
92  } // namespace artery
93
94  #endif
```

Anexo B

Código do Artery: Arquivo *Middleware*.cpp

```
1  /*
2  * Artery V2X Simulation Framework
3  * Copyright 2014-2019 Raphael Riebl et al.
4  * Licensed under GPLv2, see COPYING file for detailed
      license and warranty terms.
5  */
6
7  #include "artery/application/Middleware.h"
8  #include "artery/application/ItsG5PromiscuousService.h"
9  #include "artery/application/ItsG5Service.h"
10 #include "artery/application/XmlMultiChannelPolicy.h"
11 #include "artery/networking/PositionProvider.h"
12 #include "artery/networking/Router.h"
13 #include "artery/utility/Channel.h"
14 #include "artery/utility/PointerCheck.h"
15 #include "artery/utility/IdentityRegistry.h"
16 #include "artery/utility/InitStages.h"
17 #include "artery/utility/FilterRules.h"
18 #include "inet/common/ModuleAccess.h"
19
20 using namespace omnetpp;
21
22 namespace artery
```

```
23 {
24
25 Define_Module(Middleware)
26
27 namespace
28 {
29
30 ChannelNumber getChannel(const omnetpp::cXMLElement* cfg)
31 {
32     ChannelNumber channel = 0;
33
34     const char* ch_attr = cfg->getAttribute("channel");
35     if (ch_attr) {
36         channel = parseChannelNumber(ch_attr);
37     }
38
39     // fall back to control channel for backward-
       compatibility
40     if (channel == 0) {
41         channel = channel::CCH;
42     }
43
44     return channel;
45 }
46
47 } // namespace
48
49 Middleware::Middleware() : mLocalDynamicMap(mTimer)
50 {
51 }
52
53 Middleware::~Middleware()
54 {
55     cancelAndDelete(mUpdateMessage);
56 }
57
```

```
58 int Middleware::numInitStages() const
59 {
60     return InitStages::Total;
61 }
62
63 void Middleware::initialize(int stage)
64 {
65     if (stage == InitStages::Prepare) {
66         mTimer.setTimebase(par("datetime"));
67         mUpdateInterval = par("updateInterval");
68         mUpdateMessage = new cMessage("middleware update"
               );
69         mIdentity.host = findHost();
70         mIdentity.host->subscribe(Identity::changeSignal,
               this);
71         mMultiChannelPolicy.reset(new
               XmlMultiChannelPolicy(par("mcoPolicy").
               xmlValue()));
72     } else if (stage == InitStages::Self) {
73         mFacilities.register_const(&mTimer);
74         mFacilities.register_mutable(&mLocalDynamicMap);
75         mFacilities.register_const(&mIdentity);
76         mFacilities.register_const(&mStationType);
77         mFacilities.register_const(mMultiChannelPolicy.
               get());
78         mFacilities.register_const(&
               mNetworkInterfaceTable);
79         mFacilities.register_const(inet::getModuleFromPar
               <PositionProvider>(par("positionProviderModule
               "), findHost()));
80
81         initializeServices(InitStages::Self);
82
83         // start update cycle with random jitter to avoid
               unrealistic node synchronization
84         const auto jitter = uniform(SimTime(0, SIMTIME_MS
```

```
            ), mUpdateInterval);
85          scheduleAt(simTime() + jitter + mUpdateInterval,
               mUpdateMessage);
86      } else if (stage == InitStages::Propagate) {
87          emit(artery::IdentityRegistry::updateSignal, &
               mIdentity);
88      }
89  }
90
91  void Middleware::initializeServices(int stage)
92  {
93      cXMLElement* config = par("services").xmlValue();
94      for (cXMLElement* service_cfg : config->
           getChildrenByTagName("service")) {
95          cModuleType* module_type = cModuleType::get(
               service_cfg->getAttribute("type"));
96
97          cXMLElement* service_filters = service_cfg->
               getFirstChildWithTag("filters");
98          bool service_applicable = true;
99          if (service_filters) {
100             artery::FilterRules rules(getRNG(0),
                   mIdentity);
101             service_applicable = rules.applyFilterConfig
                   (*service_filters);
102         }
103
104         if (service_applicable) {
105             const char* service_name = service_cfg->
                   getAttribute("name") ?
106                 service_cfg->getAttribute("name") :
                       module_type->getName();
107             cModule* module = module_type->create(
                   service_name, this);
108             module->finalizeParameters();
109             module->buildInside();
```

```
110             module->scheduleStart(simTime());
111             // defer final module initialisation until
                   service is attached to middleware
112
113             ItsG5BaseService* service = nullptr;
114             if (module_type->isSimple()) {
115                 service = dynamic_cast<ItsG5BaseService
                       *>(module);
116
117                 if (!service) {
118                     error("%s is not of type
                           ItsG5BaseService", module_type->
                           getFullName());
119                 }
120             } else {
121                 // search for first ItsG5BaseService
                       submodule in case of a compound module
122                 for (cModule::SubmoduleIterator it(module
                       ); !it.end(); ++it) {
123                     service = dynamic_cast<
                           ItsG5BaseService*>(*it);
124                     if (service) {
125                         break;
126                     }
127                 }
128
129                 if (!service) {
130                     error("compound module %s has no
                           ItsG5BaseService submodule",
                           module_type->getFullName());
131                 }
132             }
133
134             unsigned ports = 0;
135             unsigned channels = 0;
136             auto promiscuous = dynamic_cast<
```

```
                ItsG5PromiscuousService*>(service);
137
138             for (const cXMLElement* listener :
                    service_cfg->getChildrenByTagName("
                    listener")) {
139                 if (listener->getAttribute("port")) {
140                     auto port = boost::lexical_cast<
                            PortNumber>(listener->getAttribute
                            ("port"));
141                     TransportDescriptor td = std::
                            forward_as_tuple(getChannel(
                            listener), port);
142                     mTransportDispatcher.addListener(
                            service, td);
143                     service->addTransportDescriptor(td);
144                     ++ports;
145                 } else if (promiscuous && listener->
                        getAttribute("channel")) {
146                     ChannelNumber channel = getChannel(
                            listener);
147                     mTransportDispatcher.
                            addPromiscuousListener(promiscuous
                            , channel);
148                     ++channels;
149                 }
150             }
151
152             // ensure that ordinary ITS-G5 services are
                    listening to at least port
153             if (ports == 0 && !promiscuous && service->
                    requiresListener()) {
154                 error("Listening ports are required for %
                        s but none have been specified",
                        module_type->getFullName());
155             }
156
```

```
157             // promiscuous ITS-G5 services grab packets
                   from CCH by default if not specified
                   otherwise
158             if (promiscuous && channels == 0) {
159                 mTransportDispatcher.
                       addPromiscuousListener(promiscuous,
                       channel::CCH);
160             }
161
162             mServices.emplace(service);
163
164             // finalize module initialization now
165             for (int i = 0; i < stage; ++i) {
166                 if (!module->callInitialize(i)) break;
167             }
168         }
169     }
170 }
171
172 void Middleware::finish()
173 {
174     emit(artery::IdentityRegistry::removeSignal, &
           mIdentity);
175 }
176
177 void Middleware::handleMessage(cMessage *msg)
178 {
179     if (msg == mUpdateMessage) {
180         updateServices();
181     } else {
182         error("Middleware cannot handle message '%s'",
               msg->getFullName());
183     }
184 }
185
186 void Middleware::receiveSignal(omnetpp::cComponent*,
```

```
    omnetpp::simsignal_t signal, long changes, omnetpp::
    cObject* obj)
187 {
188     if (signal == Identity::changeSignal) {
189         auto identity = check_and_cast<Identity*>(obj);
190         if (mIdentity.update(*identity, changes)) {
191             emit(artery::IdentityRegistry::updateSignal,
                    &mIdentity);
192         }
193     }
194 }
195
196 cModule* Middleware::findHost()
197 {
198     return inet::getContainingNode(this);
199 }
200
201 void Middleware::setStationType(const StationType& type)
202 {
203     mStationType = type;
204 }
205
206 void Middleware::registerNetworkInterface(std::shared_ptr
    <NetworkInterface> ifc)
207 {
208     mNetworkInterfaceTable.insert(ifc);
209 }
210
211 void Middleware::updateServices()
212 {
213     mLocalDynamicMap.dropExpired();
214     for (auto& service : mServices) {
215         service->trigger();
216     }
217     scheduleAt(simTime() + mUpdateInterval,
        mUpdateMessage);
```

```
218 }
219
220 void Middleware::requestTransmission(const vanetza::btp::
       DataRequestB& request,
221          std::unique_ptr<vanetza::DownPacket> packet,
                const NetworkInterface& netifc)
222 {
223     Enter_Method("requestTransmission");
224     netifc.getRouter().request(request, std::move(packet)
           );
225 }
226
227 void Middleware::requestTransmission(const vanetza::btp::
       DataRequestB& request, std::unique_ptr<vanetza::
       DownPacket> packet)
228 {
229     Enter_Method("requestTransmission");
230
231     auto channels = mMultiChannelPolicy->allChannels(
           request.gn.its_aid);
232     if (channels.empty()) {
233         EV_WARN << "No channel found for ITS-AID " <<
                request.gn.its_aid << "\n";
234     }
235
236     unsigned pass = 0;
237     for (ChannelNumber channel : channels) {
238         auto netifc = mNetworkInterfaceTable.select(
                channel);
239         if (netifc) {
240             ++pass;
241             if (channels.size() > pass) {
242                 // duplicate packet for all but last
                       network interface
243                 netifc->getRouter().request(request,
                       vanetza::duplicate(*packet));
```

```
244                 } else {
245                     // last network interface -> pass "
                           original" packet
246                     netifc->getRouter().request(request, std
                           ::move(packet));
247                 }
248             } else {
249                 EV_ERROR << "No network interface operating
                       on channel " <<  channel << "\n";
250             }
251         }
252
253         if (pass == 0) {
254             EV_ERROR << "ITS-AID " << request.gn.its_aid << "
                    packet lost in Middleware\n";
255         } else {
256             EV_DETAIL << "ITS-AID " << request.gn.its_aid <<
                   " packet passed to " << pass << " network
                   interfaces\n";
257         }
258 }
259
260 } // namespace artery
```