UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
MESTRADO ACADÊMICO EM COMPUTAÇÃO APLICADA

EDUARDO AUGUSTO KLOSOWSKI

UM PROCESSO PARA OTIMIZAÇÃO DA SUBSTITUIÇÃO DE DISPOSITIVOS DE COMUTAÇÃO DE PACOTES NA MIGRAÇÃO PARA UMA REDE SDN HÍBRIDA

JOINVILLE
2021

EDUARDO AUGUSTO KLOSOWSKI

UM PROCESSO PARA OTIMIZAÇÃO DA SUBSTITUIÇÃO DE DISPOSITIVOS DE COMUTAÇÃO DE PACOTES NA MIGRAÇÃO PARA UMA REDE SDN HÍBRIDA

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada, da Universidade do Estado de Santa Catarina, como requisito parcial para obtenção do grau de Mestre em Computação Aplicada.
Orientador: Prof. Dr. Adriano Fiorese

JOINVILLE
2021

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Setorial do CCT/UDESC,

com os dados fornecidos pelo(a) autor(a)

Klosowski, Eduardo Augusto
Um processo para otimização da substituição de dispositivos de comutação de pacotes na migração para uma rede SDN híbrida / Eduardo Augusto Klosowski. -- 2021.
79 p.

Orientador: Adriano Fiorese
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2021.

1. Migração de rede. 2. Redes definidas por software. 3. Cobertura mínima de vértices. I. Fiorese, Adriano. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

EDUARDO AUGUSTO KLOSOWSKI

UM PROCESSO PARA OTIMIZAÇÃO DA SUBSTITUIÇÃO DE DISPOSITIVOS DE COMUTAÇÃO DE PACOTES NA MIGRAÇÃO PARA UMA REDE SDN HÍBRIDA

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada, da Universidade do Estado de Santa Catarina, como requisito parcial para obtenção do grau de Mestre em Computação Aplicada.

BANCA EXAMINADORA

Prof. Dr. Adriano Fiorese
UDESC

Membros:

Prof. Dr. Rafael Rodrigues Obelheiro
UDESC

Prof. Dr. Luis Carlos Erpen de Bona
UFPR

Joinville, 26 de fevereiro de 2021

AGRADECIMENTOS

Primeiramente agradeço a Deus, e o sábado que Ele deu para a humanidade, o qual ajudou bastante em todo o processo do mestrado, garantindo um descanso semanal onde eu pude me desligar um pouco.

Agradeço aos meus professores Walter e Priscila, que me incentivaram a ingressar no mestrado, e pelas suas orientações que me auxiliaram na entrada deste programa. Assim como aos demais professores do programa que de alguma forma também contribuíram no desenvolvimento deste trabalho.

Eu não poderia deixar de agradecer as várias pessoas que conheci na Twitch durante esse período, que me ajudaram distraindo um pouco, como o pokemaobr, pelo qual pude conhecer muitas outras pessoas. Também as pessoas com quem pude trocar alguma ideia durante as suas lives sobre temas relacionados a minha pesquisa, como o profbrunolopes sobre o 3-opt, e o racerxdl sobre a diferença de rodar código em FPGA ou GPU. Agradeço à theviolinhabit por fazer a música de fundo durante a escrita deste trabalho, assim como meus professores de violino, Antônio (que as aulas muitas vezes eram dois falando das dificuldades do mestrado) e Joan.

E por último a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), pela bolsa de estudos que recebi durante parte do programa (Código de Financiamento 001). Assim como o INF/UFRGS, que me permitiu acesso aos recursos da infraestrutura PCAD para a execução dos meus experimentos.

RESUMO

Este trabalho apresenta um modelo de processo para migrar uma rede tradicional de computadores, permitindo assim, a utilização de funcionalidades SDN. O processo apresentado visa otimizar os dispositivos envolvidos na migração, tendo como resultado uma rede híbrida, com parte dos equipamentos operando em modo SDN, enquanto a outra parte continua operando de forma tradicional. A utilização de SDN é importante por não exigir que cada dispositivo precise ter sua própria implementação das funcionalidades desejadas, o que evita incompatibilidades e a impossibilidade de usar a funcionalidade, caso algum dispositivo não a tenha implementado. A otimização dos dispositivos, quando comparada à migração de toda a infraestrutura existente, reflete na redução do custo da migração, aumentando a viabilidade da implementação das funcionalidades desejadas através da migração. Parte do processo proposto envolve a resolução do problema conhecido como cobertura mínima de vértices (MVC), ou sua versão com pesos (MWVC), que por serem classificados como NP-difícil, apresentam desafios em sua resolução. Neste trabalho é discutida a resolução do MVC e do MWVC através de força bruta sem e com paralelismo em CPU e GPU, assim como o uso da heurística 3-opt para os casos que não puderem ser tratados com força bruta, apontando a técnica e parâmetros mais indicados para cada migração. Experimentos demostram a redução na quantidade de dispositivos envolvidos na migração de 25% a 50%.

Palavras-chave: Migração de rede. Redes definidas por software. Cobertura mínima de vértices.

ABSTRACT

This work presents a process model for migrating a traditional computer network allowing the use of SDN features. The proposed process aims to optimize the number of devices involved in the migration, resulting in a hybrid network, with part of the equipment operating in SDN mode while the other part continues to operate in a traditional way. The use of SDN is important because it does not require each device to have its own implementation of the desired feature, which avoids incompatibilities and the inability to use the functionality in case any device has not implemented it. The devices number optimization, when compared to the migration of the entire existing infrastructure, reflects at the migration cost reduction, increasing the feasibility of the desired functionalities implementation through the migration. Part of the proposed process involves solving the problem known as minimum vertex cover (MVC), or its weighted version (MWVC), which present challenges in their resolution since they are classified as NP-hard problems. This work discusses solving MVC and MWVC through brute force without and with parallelism in CPU and GPU, as well as the use of the 3-opt heuristic for cases that cannot be treated with brute force, pointing out the most suitable technique and parameters for each migration. Experiments show a reduction at the number of devices involved in the migration from 25% to 50%.

Keywords: Network migration. Software-defined networking. Minimum vertex cover.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CAPEX *Capital Expenditure*

CDP *Cisco Discovery Protocol*

COVER *Cover Edges Randomly*

CPU *Central Processing Unit*

DDoS *Distributed Denial-of-Service*

EWLS *Edge Weighting Local Search*

GPU *Graphics Processing Unit*

HTTP *Hypertext Transfer Protocol*

IP *Internet Protocol*

LLDP *Link Layer Discovery Protocol*

MIMD *Multiple Instruction Multiple Data*

MVC *Minimum Vertex Cover*

MWVC *Minimum Weight Vertex Cover*

NAT *Network Address Translation*

NP não determinístico em tempo polinomial

SC *Set Cover*

SDN *Software-Defined Networking*

SIMD *Single Instruction Multiple Data*

SISD *Single Instruction Single Data*

SNMP *Simple Network Management Protocol*

SSH *Secure Shell*

TCP *Transmission Control Protocol*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, os aspectos cotidianos da vida em sociedade estão cada vez mais dependentes de comunicação suportada por redes de computadores. É ela que permite que possamos realizar e delegar atividades, estar em contato com familiares, amigos e colaboradores (mesmo a distância), etc. Embora já tenha existido distinção entre redes de telecomunicações e redes de computadores, com a convergência de serviços de dados (voz, vídeo, dados indistintos, etc.), é possível, com um certo grau de confiança, considerarmos as eventuais diferenças acomodadas no conceito de redes de computadores (TANENBAUM, 2003).

Uma rede de computadores permite que os diferentes dispositivos a ela conectados troquem informações através de pacotes de dados. Para permitir essa comunicação, os dispositivos intermediários tomam decisões a respeito do encaminhamento desses pacotes de forma independente, conforme as regras que receberam e que ditam como deve ser seu comportamento. Tal procedimento visa proporcionar a tolerância a falha da rede, uma vez que ele não cria relações de dependência para a operação de cada dispositivo de rede. Em caso de ocorrência de problemas em algum desses dispositivos, tal procedimento, permite que não seja interrompida a comunicação na parte da rede que não utiliza os dispositivos que apresentaram falhas. Para que isso tudo funcione adequadamente, entre outras coisas, é necessário que cada dispositivo de rede seja configurado individualmente.

O paradigma *Software-Defined Networking* (SDN) visa retirar a camada de configuração de cada dispositivo, passando-a para um servidor externo, padronizando as diferentes interfaces e comandos de configuração de cada dispositivo. Isso permite centralizar a configuração de todos os dispositivos da rede em um mesmo servidor, facilitando a administração da rede. Outra vantagem é a possibilidade de adicionar funcionalidades para a rede como um todo, onde esse serviço centralizador pode processar a funcionalidade requerida, aplicando as configurações necessárias nos dispositivos. De outra forma, sem essa característica SDN, cada dispositivo precisaria ter sua implementação das funcionalidades ou protocolos desejados, o que poderia gerar incompatibilidades entre as implementações. Porém, com SDN é possível delegar a lógica de controle para esse serviço, não necessitando que o dispositivo tenha uma implementação local de tais características.

Entretanto, existem redes que ainda não fizeram a migração ou adotaram do paradigma SDN. Tendo em vista as vantagens operacionais da utilização de SDN nas redes, a migração das atuais infraestruturas para infraestruturas totalmente, ou mesmo parcialmente SDN, apresenta-se como uma alternativa potencialmente atrativa. Porém, a migração para SDN apresenta desafios em aberto (KREUTZ et al., 2015; ANDRIOLI; RIGHI; AUBIN, 2017; SELVARAJ; NAGARAJAN, 2017). Não são todos

os equipamentos de rede que conseguem operar em modo SDN. Muitas vezes é necessária a substituição dos mesmos. A boa notícia, em termos de custos é que também é possível utilizar uma rede com parte dos equipamentos operando de modo SDN e outra parte operando ainda em modo tradicional (KREUTZ et al., 2015).

Desta forma, este trabalho visa apresentar um modelo de processo para auxiliar na migração de infraestruturas de rede tradicionais (legadas) para SDN, centrando-se no estudo de quais dispositivos de rede devem ser substituídos para que possam ser utilizadas funcionalidades na rede por meio de SDN. Esse modelo de processo tem como foco otimizar a quantidade de dispositivos a serem substituídos, aumentando assim a viabilidade financeira da migração, e reduzindo seu impacto no *Capital Expenditure* (CAPEX) da empresa proprietária da rede. Para tanto, vale-se do estudo das funcionalidades desejadas que podem ser implementadas via SDN cruzando-se informações da topologia atual com tais funcionalidades após a migração. O modelo resultante do cruzamento dessas informações resulta em um problema conhecido como *Minimum Vertex Cover* (MVC), ou *Minimum Weight Vertex Cover* (MWVC), que são problemas NP-difíceis (GAREY; JOHNSON, 1979). Logo, existem dificuldades em otimizar o conjunto de dispositivos que devem ser substituídos. No presente trabalho será apresentado um modelo de processo para que essas migrações possam ser efetuadas, assim como resolver o MVC e o MWVC oriundo de redes de computadores, apontando as vantagens e limitações das escolhas possíveis nas etapas desse modelo de processo.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Modelar um processo para apontar quais os dispositivos de uma rede de computadores devem ser migrados, de forma que a rede permita utilizar funcionalidades por meio de SDN, visando otimizar os dispositivos a serem migrados ou substituídos.

1.1.2 Objetivos específicos

- Estudar formas de apontar quais dispositivos de uma rede legada de computadores devem ser migrados;
- Estudar técnicas de resolução do MVC e do MWVC oriundos das topologias de rede analisadas, identificando seus melhores parâmetros e limitações;
- Analisar os resultados da aplicação do processo em redes de exemplo.

1.2 ESCOPO

Este trabalho visa auxiliar na migração de uma rede Ethernet tradicional para SDN, identificando quais dispositivos de rede é necessário substituir para que as funcionalidades de rede desejadas possam ser utilizadas por meio de SDN. Assim, como resultado, gera-se uma rede híbrida, com parte dos dispositivos operando em modo tradicional e parte em modo SDN. Porém, neste trabalho não será tratada a escolha do modelo do novo dispositivo, nem como efetuar a substituição do mesmo, ou qualquer outra questão administrativa decorrente da migração. Em relação à configuração do novo dispositivo, este trabalho se limitará a indicar em qual dispositivo as regras que possibilitam as funcionalidades desejadas deverão ser aplicadas. Ele também não abordará como fazer a configuração dessas regras nos dispositivos através do controlador SDN.

Para que uma funcionalidade de rede possa ser aplicada através deste trabalho, a mesma precisa ser modelada em uma das seguintes formas: funcionalidades atribuídas diretamente a um dispositivo, onde um dispositivo recebe as regras do controlador SDN e é o responsável por aplicar a funcionalidade localmente; e funcionalidades atribuídas a um enlace, onde o controlador SDN poderia atribuir as regras a qualquer um dos dispositivos do enlace. Esta última é a forma preferível por possibilitar mais otimizações. Funcionalidades que não puderem ser modeladas desta forma não serão atendidas por este trabalho.

Uma característica da rede a ser migrada é a segmentação dos domínios de colisão por *switches* e roteadores, onde cada domínio de colisão possua apenas dois dispositivos. Não serão abordadas topologias com meios compartilhados contendo três ou mais dispositivos no mesmo domínio de colisão, como barramento, ou estrelas com *hub*. Neste último caso é possível a substituição dos *hubs* por *switches*, criando um domínio de colisão diferente entre o *switch* e o outro dispositivo.

O modelo de processo apresentado neste trabalho envolverá apenas a substituição de dispositivos por outros com suporte a SDN, não havendo alteração na quantidade de dispositivos, ou local dos mesmos dentro da rede. Qualquer mudança na rede que envolva essas características deve ser feita antes de se aplicar este processo. Se assim for feito, o resultado da aplicação do processo proposto contemplará essas mudanças na topologia, uma vez que aplicar essas alterações posteriormente pode levar a resultados menos otimizados da migração.

1.3 CARACTERIZAÇÃO DA PESQUISA

Utilizando os níveis de maturidade acerca de uma pesquisa, de 1 até 5, definidos por Wazlawick (2009), este trabalho se caracteriza no nível 3, uma vez que apresenta algo presumidamente melhor. Na literatura existem trabalhos que tratam de problemas

NP-difícil no contexto de rede de computadores (SU et al., 2014; BONFOH; MEDJIAH; CHASSOT, 2018; BARDALAI; MEDHI; CHAKRABORTY, 2019), porém eles adotam uma solução gulosa, sem otimização, ou sem discutir o que poderia ser feito para tentar reduzir o tamanho do problema para facilitar o seu cálculo. Como não existe algum *benchmark* padronizado de MVC ou MWVC voltado para grafos oriundos de redes de computadores ou sobre migração de redes, não é possível afirmar que o trabalho proposto é reconhecidamente melhor para atingir o nível 4 de maturidade.

Observando-se os três paradigmas descritos por Eden (2007), este trabalho apresenta características racionalistas na modelagem do problema e no estudo de suas propriedades que permitem diferentes formas para encontrar uma solução. Assim como, ao realizar comparação entre essas diferentes formas de encontrar a solução, utilizando para tal o desempenho de diferentes implementações, também apresenta características tecnocráticas. Apresentando assim, aspectos de dois dos três paradigmas descritos.

Segundo a classificação do raciocínio lógico proposto por Marconi e Lakatos (2003), este trabalho se classifica em hipotético-dedutivo, visto que parte do conhecimento geral aplicando-o a um problema específico, utilizando-se de experimentos para validar as questões em relação ao desempenho das implementações que buscam a resposta.

Em relação as variáveis utilizadas, segue-se uma abordagem quantitativa (GIL, 2002), utilizando valores objetivos, como quantidade de dispositivos, valores de parâmetros e tempo de execução dos algoritmos. Utilizando a classificação de variáveis dependentes e independentes de Cervo, Bervian e Silva (2007), junto a definição dos tipos de variáveis quantitativas de Barbetta, Reis e Bornia (2010), as variáveis podem ser classificadas em:

- **Independentes:**
  - Tipo de rede ou topologia: Qualitativa categórica.
  - Número de dispositivos ou vértices: Quantitativa discreta.
  - Técnica de paralelização: Qualitativa categórica.
  - Parâmetros das técnicas: Qualitativa categórica para escolher uma estratégia de distribuição de carga. Quantitativa discreta para valores utilizados pelas bibliotecas e técnicas.
  - Hardware: Qualitativa categórica.
  - Compilador e Biblioteca: Qualitativa categórica.
- **Dependente:**
  - Tempo de execução: Quantitativa contínua.

Essas variáveis são coletadas de forma direta (MARCONI; LAKATOS, 2003) através dos testes efetuados, que sempre envolvem mais de uma variável, sendo ao menos uma independente e outra dependente, sempre tendo o controle total das variáveis independentes (GIL, 2002).

1.4 CONTRIBUIÇÕES

Durante o desenvolvimento desse trabalho foram feitas as seguintes publicações:

- KLOSOWSKI, Eduardo Augusto; FIORESE, Adriano. Cobertura mínima de vértices na migração para SDN. In: **Anais da XIX Escola Regional de Alto Desempenho da Região Sul**. Porto Alegre, RS, Brasil: SBC, 2019. p. 139–142. ISSN 2595-4164. Disponível em: ⟨https://sol.sbc.org.br/index.php/eradrs/article/view/7073⟩.
- ____. FAEController: A Relative Efficiency Evaluation Framework of SDN Controllers. In: **Proceedings of the XV Brazilian Symposium on Information Systems**. ACM, 2019. p. 75:1–75:8. ISBN 978-1-4503-7237-4. Disponível em: ⟨http://doi.acm.org/10.1145/3330204.3330285⟩.
- ____. FREEController: A Framework for Relative Efficiency Evaluation of Software-Defined Networking Controllers. In: **Proceedings of the 21st International Conference on Enterprise Information Systems**. SciTePress, 2019. v. 1, p. 349–360. ISBN 978-989-758-372-8. Disponível em: ⟨http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0007761503490360⟩.

1.5 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho esta organizado da seguinte forma: O Capítulo 2 apresenta os conceitos aplicados neste trabalho, juntamente com a discussão de trabalhos relacionados, diferenciando este trabalho dos demais. O Capítulo 3 discute como modelar a migração, apresenta o modelo de processo proposto, assim como suas variantes. O Capítulo 4 apresenta e discute as diferentes formas de se resolver os problemas MVC e MWVC utilizando diferentes técnicas (força bruta, gulosa e 3-opt), seus parâmetros, assim como suas limitações. O Capítulo 5 aplica o modelo em algumas redes e discute os resultados obtidos. O Capítulo 6 apresenta as considerações a cerca dos resultados obtidos e aponta oportunidades para novos trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo apresenta os conceitos de rede de computadores, grafos, problema não determinístico em tempo polinomial (NP)-difícil e paralelismo aplicados nesse trabalho. Comparando ao final, as diferenças com outros trabalhos.

2.1 REDE DE COMPUTADORES

Uma rede de computadores é um meio que permite que os diferentes dispositivos a ela conectados realizem comunicações. Nas redes TCP/IP esta comunicação ocorre através de troca de mensagens (tráfego), utilizando uma pilha de protocolos que recebe o nome dos principais protocolos, *Transmission Control Protocol* (TCP) e *Internet Protocol* (IP). Essa pilha funciona encapsulando as mensagens de um protocolo dentro de outro, podendo incluir informações adicionais como endereços para indicar os dispositivos envolvidos na comunicação. Uma rede pode incluir dispositivos especializados que, conforme suas características e configurações, utilizam os endereços de diferentes camadas ou protocolos para tomar decisões sobre qual segmento da rede encaminhar as mensagens (TANENBAUM, 2003). Desta forma, uma rede de computadores é composta de dispositivos e enlaces (segmentos de rede), os quais interconectam esses dispositivos e são utilizados para realizar a troca de mensagens.

Existem diferentes tipos de dispositivos que podem compor uma rede de computadores, os quais oferecem diferentes funcionalidades conforme suas capacidades. Neste trabalho será tratada como uma funcionalidade qualquer comportamento que o dispositivo possa ter, como tomada de decisão de encaminhamento, não encaminhamento, priorização, ou alteração de mensagens, onde essas funcionalidades são configuradas por regras aplicadas nos dispositivos. Estes dispositivos podem possuir um *hardware* específico, ou serem virtuais, através da execução de um *software* em um dispositivo genérico (CSIKOR et al., 2020). Neste trabalho será adotado como comutação o processo de encaminhamento que ocorre utilizando as informações do protocolo Ethernet, ou do protocolo IP, conforme pode ser visto na Figura 1.

Uma rede tradicional é uma rede de computadores cujo plano de controle (*control plane*) encontra-se nos dispositivos da rede (KREUTZ et al., 2015). Desta forma cada dispositivo opera de forma independente dos demais, o que aumenta a tolerância a falhas, porém também exige que a configuração seja feita individualmente em cada dispositivo. Um outro modo de operar surge com SDN, onde o plano de controle é retirado do dispositivo e colocado em um servidor (KREUTZ et al., 2015). Este servidor, que também pode ser chamado de controlador SDN, torna-se o responsável por controlar os dispositivos da rede, disponibilizando uma interface centralizada e padronizada para configurar as funcionalidades da rede.

Figura 1 – Camadas da rede de computadores

Camadas	Unidade de Medida	Protocolo	Dispositivo
Aplicação	Dados	HTTP, SSH, DNS...	
Transporte	Segmento	TCP, UDP	
Internet	Pacote	IP	Roteador
Acesso à Rede	Quadro	Ethernet	Switch

Fonte: Elaborada pelo autor, 2021.

Uma das tecnologias que permitem os dispositivos operaram em modo SDN é o OpenFlow (MCKEOWN et al., 2008). Este protocolo define o formato das mensagens entre os dispositivos e o controlador SDN, assim como regras que podem ser configuradas nas tabelas de fluxos destes dispositivos para que possam tratar o tráfego da rede (SELVARAJ; NAGARAJAN, 2017). Ainda, é possível fazer a configuração nos dispositivos de duas formas: A primeira é a pró-ativa, onde o controlador cadastra, de forma antecipada em relação ao tráfego de produção, as regras na tabela de fluxo dos dispositivos. Estes, ao receberem um pacote, tomarão a ação devida, conforme as regras previamente instaladas. A segunda forma é a reativa onde os dispositivos, ao receberem um pacote que não se encaixa nas regras já cadastradas, consulta o controlador para tomar a ação devida relativa a ele. Nesse caso, de acordo com essa consulta, o controlador pode configurar regras nos dispositivos sob demanda, evitando uma nova consulta.

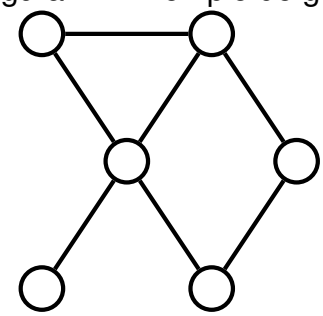
A rede de computadores também pode sofrer mudanças, como a conexão ou desconexão de dispositivos, o que altera a topologia da rede, ou a reconfiguração de seus dispositivos para que passem a apresentar um comportamento diferente, como a inclusão de uma nova funcionalidade. Estas mudanças podem ser tratadas como uma migração, uma vez que altera a rede de um estado antigo (topologia ou configuração) para um novo estado. Quando se deseja adicionar ou implementar funcionalidades em uma rede tradicional com SDN, é necessário que o administrador da mesma faça a mudança na configuração dos dispositivos, de forma que os mesmos passem a operar em modo SDN, caracterizando uma migração. Após a realização dessa mudança nos dispositivos, será possível a implementação das funcionalidades desejadas, concluindo assim a migração da rede.

2.2 GRAFO

Um grafo é uma estrutura abstrata, composta por elementos denominados nós ou vértices, e as relações entre esses elementos são denominadas arcos ou arestas, podendo ser útil na representação e solução de problemas (GOLDBARG; GOLDBARG, 2012). Considerando uma rede de computadores, composta por dispositivos e enlaces interconectando-os, ela pode ser descrita na forma de um grafo, onde os dispositivos são os vértices, e os enlaces são as arestas, facilitando assim o estudo do problema da migração da rede.

A representação matemática de um grafo pode ser descrita como $G(V, A)$, onde G representa o grafo, V é um conjunto finito não vazio de elementos (vértices), A é um conjunto de pares não ordenados de elementos de V (arestas). Nele, cada aresta $a \in A$ é representado por seu par de vértices $a = (v, u)$ (SZWARCFITER, 2018). A representação gráfica dos grafos ocorre por meio da representação dos vértices como pontos ou figuras geométricas, e das arestas como linhas que conectam esses elementos, conforme o exemplo na Figura 2. Também é possível dizer que um grafo $H(U, B)$ é um subgrafo do grafo $G(V, A)$ quando os vértices e arestas de H são subconjuntos dos vértices e arestas de G, ou seja, $U \subseteq V$ e $B \subseteq A$.

Figura 2 – Exemplo de grafo

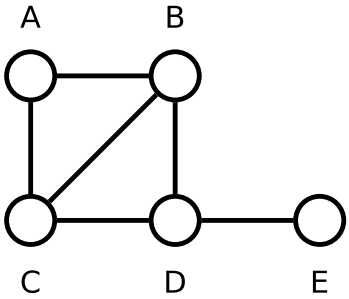


Fonte: Elaborada pelo autor, 2021.

É possível adicionar mais informações em um grafo associando dados a suas arestas ou vértices. Quando isso ocorre esse grafo passa a ser chamado de grafo rotulado (GOLDBARG; GOLDBARG, 2012), conforme o exemplo na Figura 3. Quando essas associações são de valores numéricos, representando o peso das arestas ou vértices, esse grafo também pode ser chamado de grafo ponderado (GOLDBARG; GOLDBARG, 2012), conforme o exemplo na Figura 4. Ao se associar um rótulo identificador a um vértice, permite-se que posteriormente seja identificado a qual dispositivo de rede aquele vértice representa. Também é possível utilizar os pesos associados aos vértices para dar preferência à migração de determinados dispositivos, o que pode ser levado em consideração na hora de resolver o problema da migração.

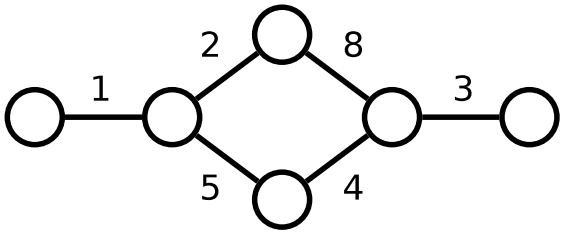
Diz-se que dois vértices (v e u) são adjacentes quando existe uma aresta ligando-os ($a = (v, u)$ e $a \in A$), e para que duas arestas serem adjacentes, é necessário que elas possuam um de seus vértices em comum (SZWARCFITER, 2018). Quando uma

Figura 3 – Exemplo de grafo rotulado



Fonte: Elaborada pelo autor, 2021.

Figura 4 – Exemplo de grafo ponderado



Fonte: Elaborada pelo autor, 2021.

sequência de vértices $v_1, v_2, \ldots, v_k$ respeita a propriedade $(v_j, v_{j+1}) \in A, 1 \leq j < k$, ou seja, é uma sequência de vértices adjacentes, essa sequência representa um caminho entre v_1 e v_k no grafo (SZWARCFITER, 2018). Outra propriedade que pode ser observada é se existe um caminho entre cada par de vértices do grafo. Quando isso ocorre, o grafo é denominado conexo, e quando não ocorre é denominado desconexo, podendo ser verificada uma descontinuidade na sua representação gráfica (SZWARCFITER, 2018). Grafos desconexos podem ser divididos em subgrafos conexos, e resolvidos separadamente quando considerado o problema da migração, trazendo vantagens, conforme será discutido no Capítulo 4.

Uma cobertura de vértices de um grafo $G(V, A)$, é um conjunto de vértices $C \subseteq V$, onde pelo menos um dos vértices de cada aresta esteja em C ($v \in C$ ou $u \in C \mid (v, u) = a, a \in A$) (GOLDBARG; GOLDBARG, 2012). Podem existir diversos conjuntos (C) que cumpram essa propriedade para um mesmo grafo, e o(s) conjunto(s) com a menor quantidade de elementos ($min(|C_1|, |C_2|, ..., |C_n|)$), que é uma otimização, recebe o nome de cobertura mínima de vértices, ou em inglês *Minimum Vertex Cover* (MVC). Quando o grafo ao qual se busca uma cobertura de vértices é ponderado, além da otimização pela menor quantidade de vértices, é possível otimizar pela menor soma de pesos (valores associados aos vértices). Neste caso a condição para um conjunto ser uma cobertura de vértices continua sendo a mesma, porém em vez de eleger o(s) conjunto(s) com a menor quantidade de elementos como o(s) mais otimizado(s), opta-se por aquele que tiver a menor soma de pesos dos vértices presentes no conjunto. Esse problema recebe o nome de peso mínimo da cobertura de vértices, ou em inglês *Minimum Weight Vertex Cover* (MWVC).

Assim como a cobertura de vértices inclui pelo menos um vértice de cada aresta do grafo, espera-se que a migração para SDN inclua pelo menos um dispositivo que possa atuar em cada enlace de rede, aplicando as funcionalidades desejadas. O Capítulo 3 discutirá em mais detalhes a relação entre funcionalidades e enlaces de rede, assim como modelar o grafo que ao ter resolvido a cobertura de vértices permitirá a utilização das funcionalidades desejadas.

2.3 PROBLEMA NP-DIFÍCIL

Os algoritmos de decisão, que são aqueles que apresentam apenas dois valores como resposta (sim ou não), para serem considerados eficientes, devem possibilitar a descrição da sua complexidade computacional (tempo de execução) através de uma expressão polinomial em relação ao tamanho de sua entrada (SZWARCFITER, 2018). Porém, para alguns problemas de decisão, não se conhecem algoritmos determinísticos eficientes (algoritmos que consigam encontrar uma solução em tempo polinomial). Embora se conheçam algoritmos eficientes que consigam validar se uma resposta é uma solução para o problema, esses problemas são classificados como não determinístico em tempo polinomial (NP). Quando é conhecida uma redução em tempo polinomial de um problema da classe NP-completo para outro da classe NP, este segundo problema também é dito pertencente a classe NP-completo, visto que ao conseguir resolver o segundo problema, também será possível resolver o primeiro. Um problema pertencente a classe NP-completo, pertence ao subconjunto dos problemas mais difíceis de se resolver da classe NP (SZWARCFITER, 2018). Ainda, é possível reduzir um problema NP-completo a outro problema que não seja de decisão, ou seja, reduzi-lo a um problema que não pertence a classe NP. Visto que a complexidade de tal problema reduzido será igual ou superior a dificuldade de se resolver um problema NP-completo, este segundo problema é classificado como NP-difícil.

O problema MVC, é conhecido como um problema NP-completo (GAREY; JOHNSON, 1979), onde se busca responder se existe algum conjunto com quantidade k ou menor de vértices que seja uma cobertura. Assim como é possível reduzir o MVC ao MWVC ao atribuir peso 1 para todos os vértices, também se classifica o MWVC com um problema NP-completo. Nesse trabalho será utilizado as versões de otimização do MVC e MWVC, portanto NP-difícil, uma vez que se busca identificar quais os vértices, e não apenas a quantidade de vértices correspondente à cobertura otimizada. Porém, ao se saber quais são os vértices, basta contá-los para conseguir responder à versão NP-completo do problema.

Embora o tempo para se computar a solução para um problema NP-difícil cresça de forma exponencial, conforme o crescimento do tamanho de sua entrada, é possível utilizar heurísticas, para se encontrar boas soluções em tempo aceitável. Isso é possível uma vez que a heurística, que considerando as características específicas daquele

problema, toma decisões que podem não levar a melhor solução, podendo excluir a possibilidade de encontrar a resposta ótima. Assim, dependendo do problema, uma heurística pode levar a uma solução válida subótima (menos otimizada). Outra forma de encontrar uma solução para problemas dessa classe é adaptando uma meta-heurística, que é uma heurística genérica, e que pode ser utilizada para encontrar soluções de diversos problemas. Nesse caso, será necessário apenas adaptar certas partes da meta-heurística conforme o problema que se deseja resolver.

Para otimizar a migração para SDN é necessário conseguir tratar os problemas MVC ou MWVC, pertencentes à classe NP-difícil. As opções determinísticas permitem ter a certeza da migração ótima, porém o tempo para fazer o cálculo da resposta torna inviável sua utilização para grandes redes, enquanto as heurísticas permitem lidar com qualquer tamanho de rede, porém sem a certeza da migração ser a mais otimizada possível. O Capítulo 4 abordará essa questão em mais detalhes, considerando as características de cada forma de tratar esses problemas.

2.4 PARALELISMO

Uma forma determinística de se resolver os problemas MVC e MWVC consiste em testar todas as possibilidades de resposta (força bruta). Isso pode ser feito testando cada possibilidade, uma após a outra. Porém essa não é uma forma considerada eficiente para tratar instâncias muito grandes desses problemas, visto o tempo necessário para se calcular a resposta. Entretanto, para esses casos, é possível verificar simultaneamente as possibilidades de resposta, utilizando técnicas de paralelismo, onde se espera reduzir o tempo de calculo da resposta conforme a quantidade de verificações feitas paralelamente, o que possibilitaria expandir um pouco o tamanho dos instâncias que são possíveis de tratar com essa técnica.

Existem diferentes formas de organização da arquitetura de um computador. Flynn (1972) classifica-as conforme a simultaneidade no processamento de instruções e dados. A arquitetura mais simples é a *Single Instruction Single Data* (SISD) que aplica uma única instrução em um único dado por vez, e assim não permite utilizar técnicas de paralelismo. As *Central Processing Unit*s (CPUs) modernas possuem vários núcleos de processamento que operam de forma independente, onde cada núcleo pode aplicar diferentes instruções em diferentes dados simultaneamente (uma instrução e um dado por núcleo). Essa arquitetura recebe o nome de *Multiple Instruction Multiple Data* (MIMD), permitindo o paralelismo em CPU. Os computadores modernos também possuem placas de vídeos com *Graphics Processing Unit* (GPU), que são coprocessadores da arquitetura *Single Instruction Multiple Data* (SIMD), onde vários núcleos executam a mesma instrução em diferentes dados (cada núcleo processa um dado diferente). As GPUs normalmente possuem muito mais núcleos que as CPUs, possibilitando um paralelismo maior quando se deseja executar as mesmas operações

em um grande conjunto de dados.

Uma discussão mais aprofundada da aplicação tanto do paralelismo em CPU, quanto em GPU na resolução dos problemas MVC e MWVC, será feita na seção 4.1. Nela será demonstrado como aplicar cada técnica e qual o ganho de cada uma em relação a não utilização de paralelismo.

2.5 TRABALHOS RELACIONADOS

A literatura aponta SDN como uma tendência, mas a migração de uma rede tradicional para SDN como uma questão em aberto, apresentando desafios ainda não resolvidos, devido aos novos conceitos e a nova forma de funcionamento das redes de computadores (KREUTZ et al., 2015; ANDRIOLI; RIGHI; AUBIN, 2017; SELVARAJ; NAGARAJAN, 2017). Alguns trabalhos relatam como a construção de uma rede SDN pode ocorrer, como Singh et al. (2016) que discorre sobre as diferentes gerações de redes utilizadas dentro de um *data center*, ou Yan, Liu e Jin (2017) que discute como dispor e interligar os equipamentos SDN. Porém, ambos os trabalhos partem da construção de uma nova infraestrutura, o que nem sempre é viável devido aos custos envolvidos. Gilesh, Kumar e Jacob (2017) apresenta uma opção para migrar uma rede tradicional para dentro de um *data center* com rede SDN, onde se mantém a topologia existente de forma virtualizada. Porém, para uma empresa isso envolveria a construção desse *data center*, ou mover sua infraestrutura local para um terceiro que ofereça esse serviço, podendo não estar de acordo com as escolhas da empresa.

Uma opção para a migração é a utilização de alguns dispositivos SDN, tendo parte da infraestrutura a operar em modo SDN e outra parte ainda operando de forma tradicional. Este tipo de rede recebe o nome de rede híbrida (KREUTZ et al., 2015), e permite utilizar algumas das vantagens do SDN enquanto a infraestrutura é migrada aos poucos para SDN. Com essa estratégia, Chang et al. (2015) introduz um dispositivo SDN na rede, utilizando-o para controlar parte do comportamento de roteamento de um roteador tradicional. Csikor et al. (2020) também introduz dispositivos SDN, porém virtuais ao invés de físicos, permitindo controlar um *switch* tradicional com OpenFlow. Porém, para tanto é necessário um servidor para executar dois *switches* virtuais para cada *switch* físico, o que aumenta os pontos de falhas únicos da rede, uma vez que um problema no servidor ou nos *switches* virtuais, deixaria inoperante o *switch* tradicional. Essa solução também limita a vazão total do *switch* à vazão do enlace com o servidor, uma vez que o *switch* precisa encaminhar todos os pacotes para o servidor.

Existem trabalhos que apesar de trabalharem com toda a rede operando em modo SDN, adicionam funcionalidades inserindo regras SDN apenas em determinados dispositivos. Esses trabalhos também fazem um estudo para encontrar o conjunto de dispositivos que receberão as regras, onde tenta-se optimizar esse conjunto, fazendo com que a menor quantidade de dispositivos recebam tais regras SDN. Su et al.

(2014) trabalha tentando encontrar ao menos um dispositivo no caminho de cada fluxo presente na rede para visualizar o consumo de largura de banda da rede, utilizando uma versão com pesos do problema conhecido como *Set Cover* (SC) para identificar os dispositivos que receberão as regras. De forma semelhante, Bonfoh, Medjiah e Chassot (2018) também faz o monitoramento da largura de banda, porém tendo um dispositivo em cada enlace de rede para isso, se diferenciando do anterior que poderia exigir um novo conjunto conforme ocorrem as mudanças dos fluxos, utilizando o problema conhecido como *Minimum Vertex Cover* (MVC). Bardalai, Medhi e Chakraborty (2019) também utiliza o MVC, porém para aplicar regras para mitigar ataques distribuídos de negação de serviço – *Distributed Denial-of-Service* (DDoS). Todos esses trabalhos utilizam soluções gulosas para encontrar uma resposta ao problema que eles tratam, o qual pode não encontrar a resposta mais otimizada.

Outros trabalhos tratam especificamente do problema MVC, tentando melhorar o desempenho na resolução desse problema independente da sua aplicação. Richter, Helmert e Gretton (2007) apresenta a heurística *Cover Edges Randomly* (COVER) para buscar coberturas com k vértices, enquanto Cai, Su e Chen (2010) apresenta a heurística *Edge Weighting Local Search* (EWLS) que busca melhorar a cobertura de vértices utilizando coberturas parciais. Porém esses trabalhos tratam grandes grafos, e não possuem uma aplicação direta em uma rede de computadores, que podem produzir grafos menores.

A Tabela 1 demonstra um comparativo entre os trabalhos citados, considerando: tipo de rede que o trabalho aborda, como rede híbrida ou inteiramente SDN; qual aplicação na rede o trabalho aborda; qual o problema conhecido abordado, quando se utiliza desses problemas para se encontrar a solução; qual a técnica utilizada para tratar esse problema conhecido.

Este trabalho se diferencia dos demais por adicionar funcionalidades à rede, e não apenas a um dispositivo, através de SDN, mantendo a topologia atual e substituindo dispositivos ao invés de adicionar novos. Assim, torna possível migrar apenas uma parte da infraestrutura para SDN, sem exigir que toda a infraestrutura esteja operando em modo SDN para que a funcionalidade desejada seja utilizada. Ele também utiliza o problema MVC, como Bonfoh, Medjiah e Chassot (2018) e Bardalai, Medhi e Chakraborty (2019), porém é mais generalista ao tipo de funcionalidade adicionada a rede, e discute outras formas além da gulosa para encontrar o conjunto de dispositivos. Tais formas incluem a força bruta para ter garantia da resposta ótima, e uma metaheurística para otimizar a resposta do algoritmo guloso. Este trabalho também aborda o problema MWVC, que permite outro tipo de otimização não discutido pelos demais artigos. Além disso, o trabalho foca-se em grafos provenientes de redes de computadores, modelados de forma a adicionar funcionalidade à rede.

Tabela 1 – Comparação dos trabalhos

Trabalho	**Rede**	**Aplicação na Rede**	**Problema**	**Resolução**
(SINGH et al., 2016)	SDN	Construção de rede SDN	–	–
(YAN; LIU; JIN, 2017)	SDN	Construção de rede SDN	–	–
(GILESH; KUMAR; JACOB, 2017)	SDN	Reconstruir rede dentro do *data center*	–	–
(CHANG et al., 2015)	Híbrida	Controlar roteador tradicional	–	–
(CSIKOR et al., 2020)	Híbrida	Controlar *switch* tradicional	–	–
(SU et al., 2014)	SDN	Monitorar banda	SC	Guloso
(BONFOH; MEDJIAH; CHASSOT, 2018)	SDN	Monitorar enlaces	MVC	Guloso
(BARDALAI; MEDHI; CHAKRABORTY, 2019)	SDN	Mitigar DDoS	MVC	Guloso
(RICHTER; HELMERT; GRETTON, 2007)	–	–	MVC	COVER
(CAI; SU; CHEN, 2010)	–	–	MVC	EWLS

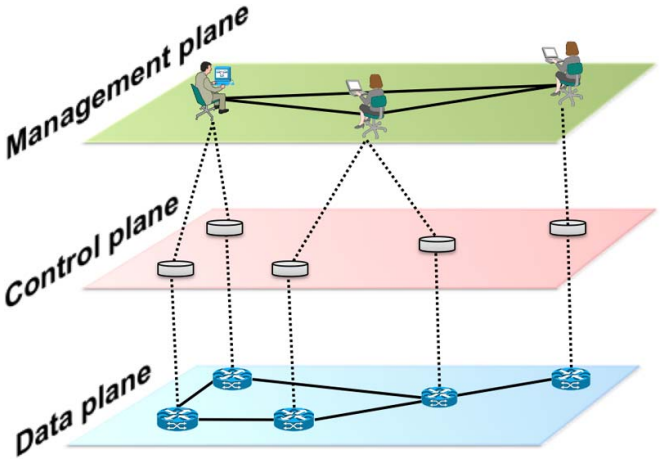
Fonte: Elaborada pelo autor, 2021.

3 DISCUSSÃO DO PROBLEMA

Para utilizar funcionalidades SDN em uma rede legada de computadores (rede que não possui dispositivos operando em modo SDN) é necessário que o modo de operação dos dispositivos seja alterado para SDN. Esta alteração pode incluir desde uma simples configuração, em dispositivos que já suportam SDN, até a substituição do dispositivo quando este não suporta este modo de operação. Contudo, há a possibilidade da utilização de redes híbridas (KREUTZ et al., 2015), onde parte dos dispositivos opera em modo SDN e outra parte continua operando sem utilizar SDN. Nesse caso, é possível utilizar funcionalidades na rede por meio de SDN, tirando proveito de algumas de suas vantagens, porém sem a necessidade de migrar (substituir) todos os dispositivos existentes na rede.

Para definir quais dispositivos devem ser migrados para permitir a utilização de determinada funcionalidade na rede através de SDN, a funcionalidade desejada deve ser analisada em conjunto com a rede onde ela será aplicada. Em uma rede legada, cada dispositivo é configurado diretamente na forma de regras. Essas regras configuradas refletem no comportamento de encaminhamento de pacotes dos dispositivos. A Figura 5 exemplifica este processo, onde funcionalidades são definidas na camada de gerenciamento (*management plane*), que por sua vez configura regras, que representem tais funcionalidades, diretamente nos dispositivos (*control plane*), que as refletem na tomada de decisão durante o encaminhamento de pacotes entre os enlaces disponíveis dos dispositivos até o seu destino.

Figura 5 – Camadas das funcionalidades de rede legada



Fonte: Kreutz et al. (2015).

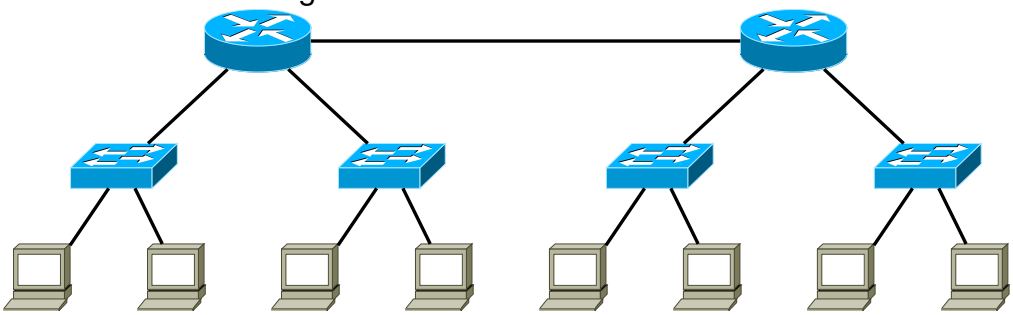
No modo de operação SDN, a camada de controle (*control-plane*) é retirada de dentro do dispositivo e colocada em um servidor externo: o controlador SDN (KREUTZ et al., 2015). Ele pode ser um único servidor para toda a rede, atuando de forma centralizada na gestão do encaminhamento dos diversos fluxos presentes na rede. Os dispositivos operando em modo SDN consultam este controlador, que tomará a

decisão de encaminhamento e poderá instalar regras nas tabelas de fluxos de cada dispositivo, permitindo que este execute o encaminhamento dos pacotes sem precisar consultar o controlador a cada novo pacote recebido do mesmo fluxo de comunicação. Embora no modo SDN a configuração seja feita em um servidor centralizado, as regras continuam sendo aplicadas em determinados dispositivos da rede, uma vez que eles recebem as regras do controlador, ou requisitam a tomada de decisão ao controlador. Logo um estudo de onde a funcionalidade será aplicada contribui para a identificação dos dispositivos a serem migrados, permitindo assim utilizar as funcionalidades de rede através de SDN.

3.1 POSICIONAMENTO DE REGRAS

A Figura 6 apresenta dois roteadores interconectados, onde cada roteador possui duas redes locais distintas, cada uma representada por um *switch* e alguns computadores.

Figura 6 – Redes interconectadas



Fonte: Elaborada pelo autor, 2021.

Considerando que a comunicação entre qualquer dispositivo presente nessa rede ocorre sem problemas, e que se deseja criar um bloqueio do serviço *Hypertext Transfer Protocol* (HTTP), impedindo que os computadores das redes locais do roteador da esquerda acessem os serviços disponíveis nas redes locais do roteador da direita, essa funcionalidade seria aplicada em um dos dois roteadores. No modelo legado, a regra é aplicada associada a uma interface de rede e uma direção (entrada ou saída), podendo ser aplicada em qualquer interface dos roteadores para essa topologia, conforme a escolha do administrador da rede. Porém, como se deseja bloquear o acesso de duas redes aos serviços HTTP de outras duas redes, para essa topologia, as interfaces do enlace que ligam os dois roteadores seriam as mais indicadas, uma vez que o bloqueio poderia ser feito com apenas uma regra em qualquer umas das duas interfaces, visto que qualquer comunicação do serviço HTTP que se deseja bloquear passa por esse enlace de rede, embora seja necessário que o pacote trafegue por parte da rede até ser bloqueado. Caso contrário, seria necessário uma regra para cada interface conectada à rede local, duplicando a configuração, e dificultando a manutenção das regras.

Ainda nessa rede, caso deseja-se aplicar também a funcionalidade de bloqueio dos serviços *Secure Shell* (SSH), desta vez dos computadores das redes do roteador da direita para os computadores das redes do roteador da esquerda, a mesma análise se repete. Assim, novamente a regra será melhor aplicada, do ponto de vista da simplificação da configuração, em qualquer uma das interfaces do enlace entre os roteadores.

Desta forma é possível associar essas regras de bloqueio ao enlace de rede entre os roteadores, uma vez que podem ser aplicadas em qualquer uma das interfaces que compõem esse enlace, visto que não existem outros dispositivos no domínio de colisão deste enlace, e qualquer comunicação feita por meio desse enlace passará pelos dois dispositivos. E todas as regras associadas ao enlace podem ser aplicadas de forma centralizada em apenas um dos seus dispositivos. Para o caso de aplicar essas regras por meio de SDN, seria necessário migrar apenas o dispositivo que receberá as regras, tendo como resultado uma rede SDN híbrida, com parte dos equipamentos operando em modo SDN e outra parte em modo legado.

Existem outras funcionalidades que podem ser associadas ao enlace de rede, de forma semelhante à análise feita para o bloqueio de tráfego, que podem ser aplicadas em qualquer um dos dispositivos do enlace. Para o propósito de exemplificação, entre essas funcionalidades estão: i) A limitação da largura de banda, que limita a taxa de transmissão que pode ser utilizada por determinada aplicação ou cliente. ii) Regras de *Network Address Translation* (NAT), que podem ser aplicadas antes de se encaminhar os pacotes por um enlace, ou ao recebê-los na interface de rede do dispositivo seguinte. iii) O monitoramento da rede, onde se deseja saber o estado (operante ou não) e uso da banda (*download* e *upload*) dos enlaces de rede. Para esse caso particular, tira-se proveito de que cada enlace pode ser monitorado por uma única interface, visto que o estado será o mesmo, e o *download* de um lado do enlace será o *upload* do outro e vice-versa. iv) O monitoramento do conteúdo do tráfego, que identifica as comunicações feitas através da rede por cliente ou aplicação. Todas essas funcionalidades, embora possa haver alguma diferença se aplicadas próximas à origem ou ao destino, podem ser configuradas intercambiavelmente entre os dois equipamentos diretamente conectados. Com isso, não é necessário migrar todos os dispositivos para SDN para utilizá-las, funcionando portanto também em uma rede SDN híbrida.

3.2 PROCESSO PROPOSTO

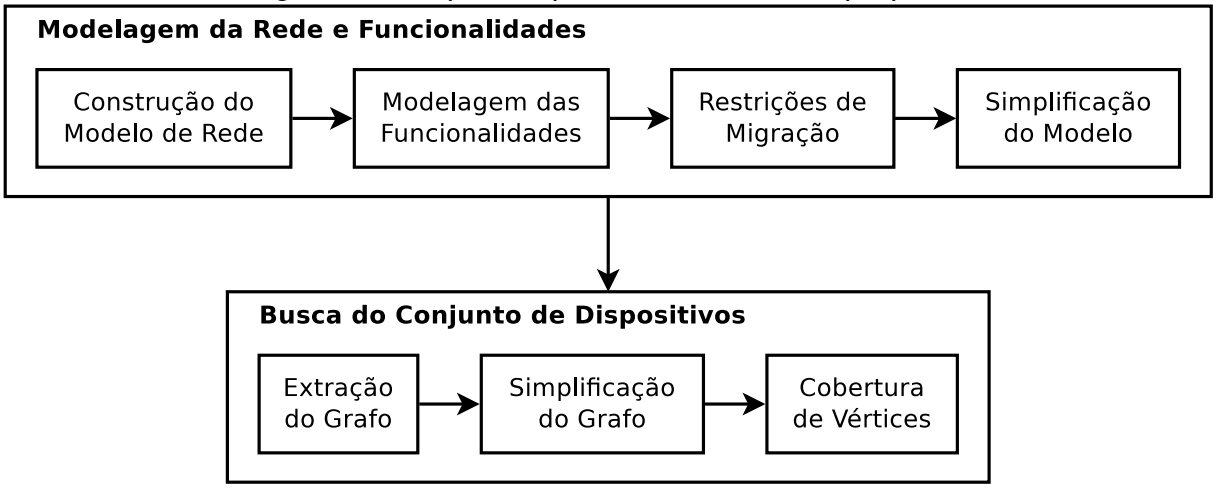
Visando efetuar otimizações que reduzam a quantidade de dispositivos a serem migrados, pode-se generalizar a análise realizada na seção 3.1 em um modelo de processo. A vantagem de um modelo de processo é o conhecimento prévio de todas as etapas e passos a serem executados, assim como a garantia da validade tanto do processo, quando do resultado, além de permitir trabalhar mais facilmente com redes

maiores.

O modelo de processo aqui proposto, permite optar por três tipos de otimização: i) Reduzir o número de dispositivos migrados, onde essa migração pode ir desde a alteração da configuração do dispositivo, até a compra de um outro dispositivo com suporte a SDN para substituí-lo, caso necessário. ii) Reduzir o número de dispositivos substituídos por novos, onde é dada a preferência na aplicação de regras em dispositivos que já possuem suporte a SDN, visando reduzir a compra de novos dispositivos. iii) Reduzir a soma dos pesos dos dispositivos, onde esses pesos podem vir de alguma preferência administrativa, ou representar o custo da migração de cada dispositivo, como por exemplo, o valor de um novo dispositivo somado ao valor da hora técnica para migrá-lo, visando minimizar o custo. As diferenças no processo e como fazer cada um dos tipos de otimização serão apresentadas durante a explicação do modelo de processo.

O modelo de processo proposto nesse trabalho pode ser visto em duas etapas, conforme a Figura 7. A primeira etapa é a modelagem da rede e funcionalidades, onde se constrói um modelo contendo informações da rede e funcionalidades desejadas após a migração. A segunda etapa é a busca do conjunto de dispositivos, onde o modelo da etapa anterior é analisado buscando-se dispor as regras a serem aplicadas nos dispositivos de rede. Uma discussão mais aprofundada de cada etapa será feita nas seções a seguir.

Figura 7 – Etapas do processo de análise proposto



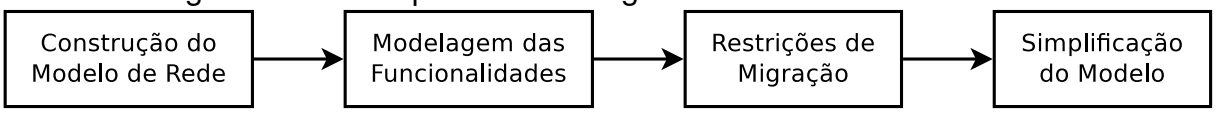
Fonte: Elaborada pelo autor, 2021.

3.2.1 Modelagem da rede e funcionalidades

O objetivo da etapa de modelagem da rede e funcionalidades é a obtenção de informações da rede a ser migrada e das funcionalidades de rede desejadas após a migração. Tal objetivo se materializa em um modelo que agrega essas informações.

Esta etapa é composta de quatro subetapas, conforme a Figura 8. De forma resumida, a primeira subetapa é a construção do modelo de rede. Nela se constrói o modelo da rede já existente através de um diagrama. A segunda subetapa é a modelagem das funcionalidades. Nela são adicionadas as funcionalidades desejadas ao modelo da rede produzido na subetapa anterior. Na terceira subetapa são verificadas as restrições de migração. Nela analisa-se os dispositivos que podem ou não ser migrados ou substituídos para operar em modo SDN, de forma que as funcionalidades desejadas possam ser, de fato, implementadas. A quarta subetapa é uma simplificação do modelo, feita a partir das informações já identificadas nas subetapas anteriores.

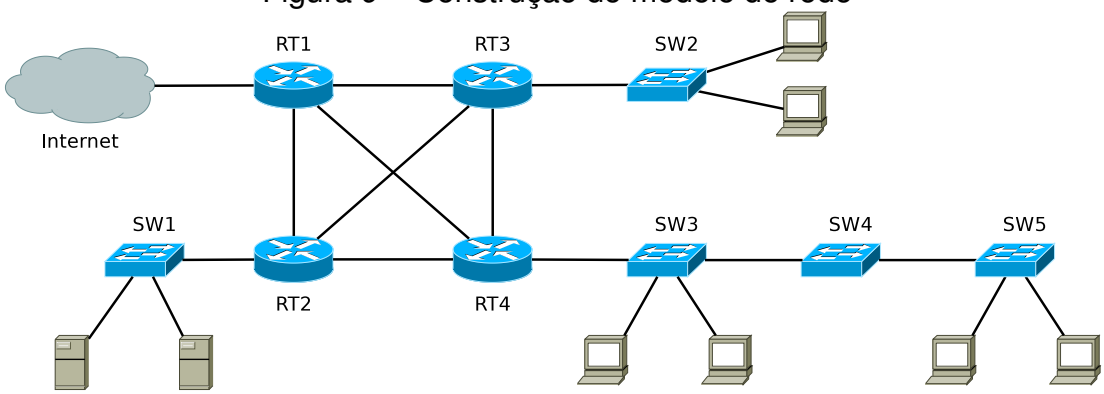
Figura 8 – Subetapas da modelagem da rede e funcionalidades



Fonte: Elaborada pelo autor, 2021.

A primeira subetapa do processo a ser executada é a construção do modelo de rede. O objetivo desta subetapa é descrever os dispositivos presentes na infraestrutura e os enlaces de rede que os conectam, podendo ser representada em um diagrama, como na Figura 9. Esta subetapa pode ser executada de forma inteiramente manual, ou com o auxílio de protocolos de rede, como *Simple Network Management Protocol* (SNMP), *Link Layer Discovery Protocol* (LLDP) e *Cisco Discovery Protocol* (CDP) que apresentam informações sobre os dispositivos e suas interfaces de rede. Para os casos onde deseja-se fazer a otimização do tipo iii (reduzir a soma dos pesos), também é necessário atribuir um peso para cada dispositivo, onde quanto menor o peso, maior a preferência por migrá-lo, e quanto maior o peso, maior a preferência por não migrá-lo.

Figura 9 – Construção do modelo de rede

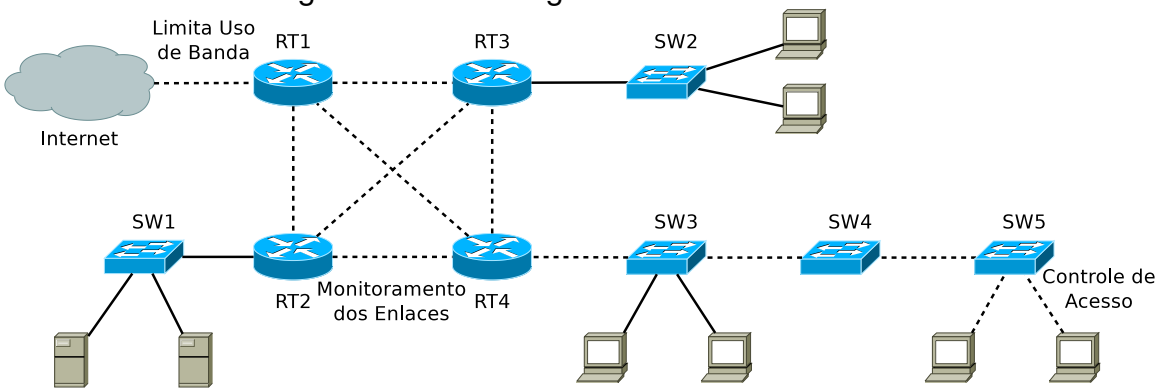


Fonte: Elaborada pelo autor, 2021.

A segunda subetapa do processo a ser executada é a modelagem das funcionalidades. O objetivo desta subetapa é associar as funcionalidades de rede disponibilizadas por meio de SDN desejadas após a migração com o local da rede onde ela será aplicada. Para isso, cada funcionalidade desejada deve ser associada a um

dispositivo ou enlace de rede onde será aplicada. Para as funcionalidades que podem ser associadas a um enlace de rede, como descrito na seção 3.1, deve-se optar por fazê-lo, visto que isso permitirá um estudo posterior a respeito de qual dispositivo daquele enlace receberá a regra, o que permite otimizar os dispositivos na migração. Para as demais funcionalidades, deve-se associá-la a um dispositivo específico. Um exemplo pode ser visto na Figura 10. Nela, em linhas pontilhadas, foram modeladas algumas funcionalidades nos enlaces de rede, como a limitação do uso de banda no enlace entre o roteador `RT1` e a Internet, controle de acesso dos dispositivos conectados no *switch* `SW5`, e monitoramento do estado dos enlaces entre os roteadores `RT1`, `RT2`, `RT3`, `RT4` e *switches* `SW3`, `SW4` e `SW5`.

Figura 10 – Modelagem das funcionalidades

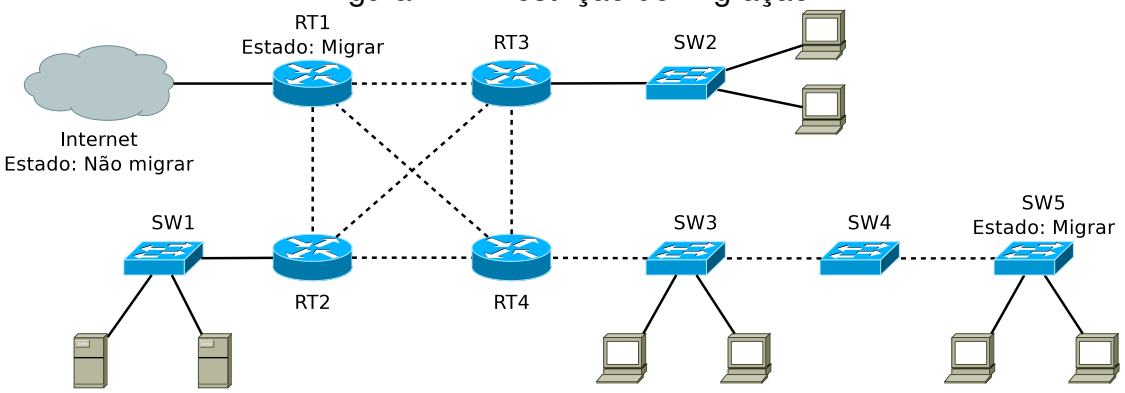


Fonte: Elaborada pelo autor, 2021.

A terceira subetapa do processo a ser executada é a restrição de migração. O objetivo desta subetapa é identificar dispositivos que não podem ser migrados para operar em modo SDN, ou que por decisão administrativa, opte-se por não fazê-lo. Além disso, tem como objetivo identificar dispositivos que serão migrados obrigatoriamente, ou por decisão administrativa, opte-se por fazê-lo. Caso a opção de otimização for a do tipo ii, ou seja, reduzir o número de dispositivos substituídos, evitando a compra de novos dispositivos, deve-se assumir que todos os dispositivos que já possuem suporte a SDN, se existirem, serão migrados. Quando existe alguma regra associada a um enlace entre um dispositivo de rede e um dispositivo final, como um *switch* ligando computadores a rede, eles também devem ser obrigatoriamente migrados visto que regras SDN não são aplicáveis diretamente no dispositivo final. Também deve ser observado que caso exista algum dispositivo que tenha alguma funcionalidade associada a si, porém o mesmo não puder ser migrado, não será possível utilizar estas funcionalidades. Da mesma forma, se existir alguma funcionalidade associada a um enlace, e os dois dispositivos do enlace não puderem ser migrados, também não será possível utilizar estas funcionalidades. Para concluir esta subetapa, todos os dispositivos que obrigatoriamente devem ser migrados recebem o estado `migrar`, assim como os dispositivos que não serão migrados recebem o estado `não migrar`. Os demais continuam com estado indefinido. A Figura 11 demonstra o `RT1` e o `SW5` sendo

migrados, visto que existe uma funcionalidade em um de seus enlaces com outros dispositivos (Internet e computadores) que não receberão regras SDN.

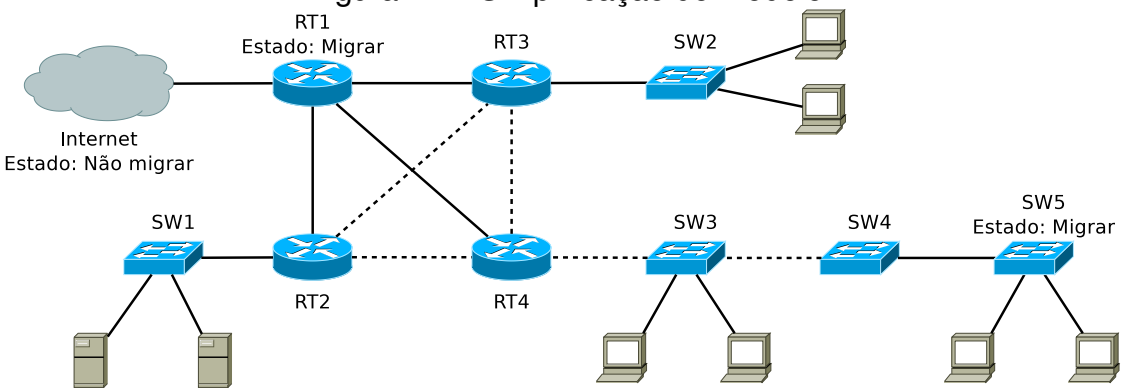
Figura 11 – Restrição de migração



Fonte: Elaborada pelo autor, 2021.

A quarta subetapa do processo a ser executada é a simplificação do modelo. O objetivo desta subetapa é simplificar o modelo reatribuindo as funcionalidades dos enlaces aos dispositivos que já possuem estado `migrar`. Para executar esta subetapa, todas as funcionalidades atribuídas a um enlace que possua um dispositivo com o estado `não migrar` devem ser reatribuídas ao outro dispositivo do mesmo enlace, que por sua vez receberá o estado `migrar`, caso ainda não possua. Todas as funcionalidades atribuídas a um enlace que possua dispositivos com o estado `migrar` devem ser reatribuídas a este dispositivo. Caso os dois dispositivos do enlace tenham este estado, as funcionalidades podem ser atribuídas em qualquer um dos dois dispositivos. A Figura 12 mostra a simplificação das funcionalidades dos enlaces do `RT1` e `SW5`. Nela se observa o retorno das linhas ligando esses dispositivos para sólidas, representando que essas funcionalidades já foram resolvidas.

Figura 12 – Simplificação do modelo



Fonte: Elaborada pelo autor, 2021.

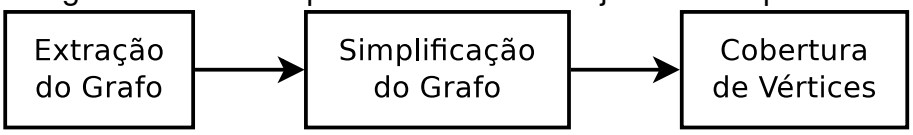
É possível executar essa etapa para qualquer rede, embora para algumas redes nem todas as etapas terão otimizações a serem feitas, por exemplo, dispositivos que serão obrigatoriamente migrados ou não. Entretanto, para qualquer rede de computadores essa etapa terá como resultado um modelo da rede com as funcionalidades

desejadas, sendo a única diferença o quanto da migração já estará resolvida (dispositivos que já receberam os estados `migrar` ou `não migrar`), deixando para a próxima etapa a resolução da parte ainda não resolvida da rede.

3.2.2 Busca do conjunto de dispositivos

O objetivo da etapa de busca do conjunto de dispositivos é otimizar o conjunto de dispositivos que precisam ser migrados para que as funcionalidades de rede possam ser implantadas e utilizadas por meio de SDN. Ela tem como resultado o conjunto de dispositivos que devem ser utilizados de fato na migração. Esta etapa é composta de três subetapas, conforme Figura 13. De forma resumida, a primeira subetapa é a extração do grafo. Nela é extraído um grafo a partir do modelo de rede gerado na etapa anterior. A segunda subetapa é a simplificação deste grafo. Nela é efetuado um pré-processamento para reduzir a carga a ser processada posteriormente. A terceira subetapa é a cobertura de vértices. Nela é resolvida a instância do problema MVC para as otimizações do tipo i e ii, ou do problema MWVC para as otimizações do tipo iii, referente ao grafo preprocessado, apontando assim os dispositivos que precisam ser migrados.

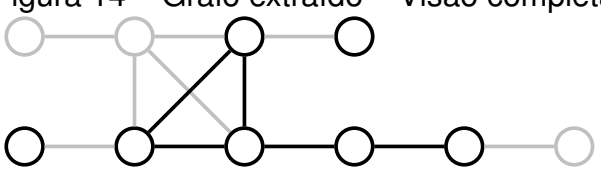
Figura 13 – Subetapas da busca do conjunto de dispositivos



Fonte: Elaborada pelo autor, 2021.

A primeira subetapa da busca do conjunto de dispositivos é a extração do grafo. O objetivo desta etapa é extrair um grafo do modelo resultante da etapa anterior. A criação do grafo se dá com a representação dos dispositivos como vértices, e os enlaces como arestas ligando esses vértices. Porém, o grafo gerado deve conter apenas os vértices que representem dispositivos que ainda não possuem estado de migração definido, e as arestas dos enlaces que ainda possuem alguma funcionalidade atribuída (se mantiveram em linhas pontilhadas), visto que é nesse ponto da rede que se deseja trabalhar. A Figura 14 mostra o grafo de toda a rede, enquanto a Figura 15 mostra apenas os vértices e as arestas conforme realizado nesta etapa.

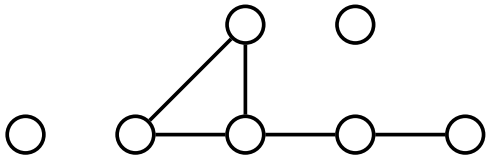
Figura 14 – Grafo extraído – Visão completa



Fonte: Elaborada pelo autor, 2021.

A segunda subetapa é a simplificação do grafo. O objetivo desta subetapa é reduzir o tamanho do grafo a fim de reduzir o tempo de seu processamento posterior.

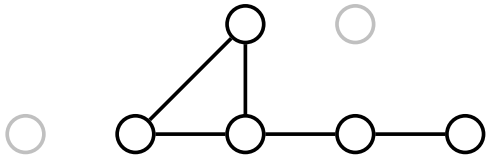
Figura 15 – Grafo extraído – Resultado efetivo



Fonte: Elaborada pelo autor, 2021.

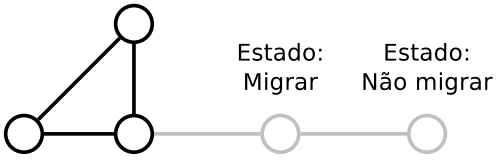
A primeira técnica que pode ser aplicada é atribuir o estado `não migrar` para os dispositivos relativos aos vértices que não possuem nenhuma aresta, e remover esses vértices do grafo, uma vez que esses dispositivos não tiveram nenhuma funcionalidade atribuída a si ou a um de seus enlaces. A Figura 16 mostra dois vértices que podem ser removidos utilizando esta técnica. Uma segunda técnica que pode ser aplicada é buscar por vértices que possuem uma única aresta. Os dispositivos referentes a esses vértices podem receber o estado `não migrar`, enquanto seu vértice adjacente recebe as funcionalidades daquela e de todas as suas demais arestas, e o estado `migrar`. Com isso, os dois vértices em questão, junto a suas arestas, podem ser removidos do grafo. Essa segunda técnica pode ser repetida até que não haja mais vértices com apenas uma única aresta. Entretanto, quando aplicada, pode evitar que possíveis alternativas de resposta (mesmo nível de otimização) sejam encontradas, mas não impede que a resposta ótima seja encontrada, como ocorre com algumas heurísticas. Porém, essa técnica não pode ser aplicada para a otimização do tipo iii, uma vez que ela não leva em conta os pesos dos vértices. A Figura 17 demonstra a aplicação desta técnica removendo dois vértices.

Figura 16 – Simplificação do grafo – Vértices sem arestas



Fonte: Elaborada pelo autor, 2021.

Figura 17 – Simplificação do grafo – Vértices com apenas uma aresta

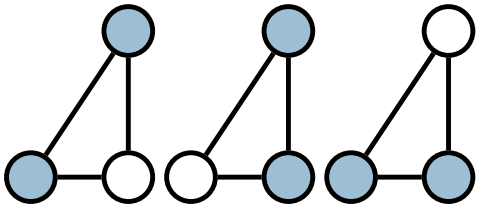


Fonte: Elaborada pelo autor, 2021.

A terceira subetapa é a cobertura de vértices. O objetivo desta etapa é encontrar um conjunto otimizado (menor quantidade possível para as otimizações dos tipos i e ii, e a menor soma de pesos para o tipo iii) de dispositivos que possam receber as funcionalidades que ainda estão atribuídas aos enlaces de rede, definindo assim o estado dos dispositivos que ainda estão indefinidos. Para isso, dado o grafo resultante

da subetapa anterior, bastaria encontrar um subconjunto do conjunto de vértices do grafo otimizado de tal forma que, pelo menos, um dos vértices de cada aresta esteja contido nesse subconjunto. Esse problema é conhecido como o problema da cobertura mínima de vértices, ou em inglês *Minimum Vertex Cover* (MVC) quando se deseja o menor conjunto possível, e peso mínimo da cobertura de vértices, ou em inglês *Minimum Weight Vertex Cover* (MWVC) quando se deseja a menor soma de pesos, sendo discutido diferentes formas de tratá-los no Capítulo 4. Sempre existirá uma resposta para esses problemas, e para um mesmo grafo podem existir diferentes subconjuntos com o mesmo nível de otimização. Quando existem diferentes respostas, cada uma é uma opção de migração que pode ser escolhida posteriormente. Um exemplo pode ser visto na Figura 18 que apresenta três opções distintas de resposta, onde os vértices presentes no conjunto de resposta do problema MVC estão em destaque com a cor azul. Encontrado o subconjunto que corresponde ao MVC ou MWVC, basta atribuir o estado `migrar` aos dispositivos referentes aos vértices presentes nesse subconjunto, enquanto aos demais dispositivos, que ainda não tem estado definido, atribui-se o estado `não migrar`. As funcionalidades que ainda estão associadas aos enlaces de rede podem ser reatribuídas a um dos dispositivos que serão migrados daquele enlace. Desta forma, encontra-se quais dispositivos precisam ser migrados, e em qual dispositivo cada funcionalidade será implementada pelo controlador SDN.

Figura 18 – Resultados possíveis do MVC



Fonte: Elaborada pelo autor, 2021.

Desta forma, é possível modelar a migração de qualquer rede de computadores com qualquer funcionalidade, desde que para isso não necessite migrar algum dispositivo que não possa ser migrado, como apresentado na subetapa restrição de migração da etapa de modelagem da rede e funcionalidades, onde o processo indicará a impossibilidade da migração. Para os demais casos, este processo encontrará um conjunto de dispositivos que permitirá a utilização das funcionalidades desejadas, variando apenas o quanto otimizada será a sua resposta.

4 RESOLUÇÃO DA COBERTURA DE VÉRTICES

Quando o processo de migração chega na subetapa cobertura de vértices, é necessário tratar os problemas MVC ou MWVC que são conhecidos como NP-difícil. Embora o ideal seja encontrar o conjunto com o menor número de vértices possível no caso do MVC, ou a menor soma de pesos possível no caso do MWVC, uma aproximação que inclua mais vértices que a resposta ótima, ou tenha uma soma de pesos maior, é uma resposta válida. Entretanto, tal resposta, no caso aplicado à implantação de funcionalidades em uma rede tradicional por meio de SDN, exigirá a migração de mais dispositivos de rede, ou com maior custo. Ainda, nesse caso, quando duas respostas com a mesma quantidade de vértices são encontradas, ou com a mesma soma de pesos, é possível optar livremente entre elas, pois ambas têm o mesmo nível de otimização e permitem a implantação das funcionalidades SDN requeridas.

Devido ao crescimento exponencial dos problemas MVC e MWVC em função do número de vértices do grafo que representa a parte não resolvida da migração (2^n), uma das formas de facilitar sua resolução é dividindo o problema, lidando com dois ou mais subproblemas com uma quantidade menor de vértices cada. Assim, embora a soma de vértices desses subproblemas se mantenha a mesma, a soma dos tamanhos dos subproblemas será menor que o tamanho do problema inicial ($2^a + 2^b < 2^{a+b}$, onde a e b são as quantidades de vértices de cada subproblema). Essa divisão pode ser feita quando o grafo a ser processado for desconexo, dividindo-o em grafos conexos menores, visto que esses pequenos grafos não interagem entre si, uma vez que não possuem arestas ligando-os.

Para identificar se existem regiões desconexas, possibilitando a divisão do grafo, o seguinte procedimento pode ser aplicado: Primeiro adiciona-se um vértice do grafo a um conjunto. Depois adiciona-se ao mesmo conjunto outro vértice que seja adjacente a algum vértice que foi adicionado nesse conjunto. Repete-se esse segundo passo até que todos os vértices adjacentes aos vértices do conjunto tenham sidos adicionados ao conjunto. Os vértices desse conjunto formam um grafo conexo, e devem ser resolvidos juntos. Caso existam vértices do grafo que não foram adicionados ao conjunto, cria-se outro conjunto e se repete o procedimento, até que todos os vértices do grafo tenham sido adicionados a um conjunto. Cada um desses conjuntos representam um problema menor, e podem ser resolvido separadamente de diferentes formas.

4.1 FORÇA BRUTA

Uma forma de se resolver o problema MVC ou MWVC é testando todas as possibilidades de resposta, também conhecido como força bruta. Para um grafo $G = (V, A)$, a força bruta consiste em testar todas as possibilidades para o subconjunto

$V' \subseteq V$ (possibilidades de resposta). Para cada vértice $v \in V$ individualmente existem duas possibilidades, estar presente ou não estar presente no subconjunto V'. Assim para um vértice existem duas possibilidades, e para dois vértices deve-se verificar a interação dessas possibilidades, ou seja, a possibilidade do segundo vértice estar presente com todas as possibilidades do primeiro vértice, e a possibilidade do segundo vértice não estar presente com todas as possibilidades do primeiro vértice, totalizando quatro verificações. Assim para n vértices deve-se testar duas vezes as possibilidades de $n-1$ vértices, uma vez incluindo o vértice n no subconjunto V' e outra vez não o incluindo. Desta forma, existem $2^{|V|}$ possibilidades de resposta a serem validadas para o subconjunto V' do grafo G.

Cada subconjunto V' pode ser representado por um valor numérico com uma quantidade de bits igual à quantidade de vértices do subconjunto, onde cada bit do número representa um vértice (v) do grafo G, sendo 1 quando $v \in V'$ e 0 quando $v \notin V'$. Desta forma, todas as possibilidades de V' podem ser representadas pela sequência numérica de 0 até $2^{|V|}-1$ (incluindo os extremos). Dada a sequência numérica, é necessário filtrar os números que representam coberturas de vértices válidas. Para tal, percorre-se todas as arestas do grafo para cada número da sequência, verificando se, pelo menos, um dos vértices da aresta possui bit 1 ($u \in V'$ ou $v \in V' \mid (u,v) \in A$), descartando os números (subconjuntos V') que não atendem essa propriedade. Esta etapa será chamada de `filter`, uma vez que filtra as possibilidades válidas. A partir das possibilidades válidas, para o MVC encontra-se aquela que possui o menor número de vértices ($|V'|$), ou seja, o valor numérico com a menor quantidade de bits 1; enquanto para o MWVC encontra-se o subconjunto com a menor soma dos pesos dos vértices presentes no subconjunto V'. Esta etapa será chamada de `reduce`, uma vez que analisa diferentes respostas válidas, resultando na solução do problema.

Nesse sentido, para calcular a resposta aos problemas MVC e MWVC por força bruta são apresentadas diversas implementações envolvendo desde *Single Thread*, que não apresenta paralelismo, até paralelismo envolvendo as opções em CPU, em GPU, bem como uma mescla das duas opções de paralelismo.

4.1.1 Single Thread

A abordagem *Single Thread* é a implementação do processo já descrito, executando mutuamente as etapas `filter` e `reduce`. Assim, ao identificar uma resposta válida para a cobertura de vértices, ela é comparada com a melhor resposta válida descoberta até então, e armazena-se apenas a melhor das duas respostas, voltando à etapa `filter` para validar a próxima resposta. Esta execução mútua das etapas é feita visando não ser necessário guardar todas as respostas válidas em memória, reduzindo seu consumo e evitando perdas de desempenho pelas operações de escrita e leitura em excesso na memória.

É importante observar uma possibilidade de otimização na etapa `filter`. Ao se verificar que o primeiro dos dois vértices de uma aresta está no subconjunto de resposta (V'), não existe a necessidade de verificar a presença do segundo vértice, podendo continuar com a verificação da aresta seguinte. De forma semelhante, ao verificar que os dois vértices de alguma aresta não estão presentes no subconjunto ($u \notin V'$ e $v \notin V'$), é possível deduzir que a resposta não é válida, não havendo a necessidade de verificar as demais arestas, podendo continuar com a verificação da próxima resposta (subconjunto V').

Como resposta válida inicial, o algoritmo pode utilizar o próprio conjunto V, que embora seja a pior resposta, é uma resposta válida. Na etapa `reduce`, qualquer outra resposta será melhor que ela, substituindo-a como melhor resposta até então. Dessa forma, o conjunto V funciona como um elemento neutro para o `reduce`, não interferindo na resposta do algoritmo.

O algoritmo 1 exemplifica o processo descrito.

Algoritmo 1: Resolução do MVC por força bruta - *Single Thread*

```
1  Função forcaBruta(G)
       Entrada: Grafo G = (V, A)
       Saída: conjunto de vértices
2      melhorResposta ← G.V
3      para cada resposta ∈ respostasPossiveis faça
4          valido ← Verdadeiro
5          para cada (u, v) ∈ G.A faça
6              se não (u ∈ resposta ou v ∈ resposta) então
7                  valido ← Falso
8                  interrompe
9              fim
10         fim
11         se valido então
12             se |resposta| < |melhorResposta| então
13                 melhorResposta ← resposta
14             fim
15         fim
16     fim
17     retorna melhorResposta
18 fim
```

A abordagem *Single Thread* é a mais simples e não apresenta paralelismo, e justamente por não apresentar paralelismo será utilizada como base de comparação, apontando se houve ganho com a aplicação do paralelismo nas demais abordagens.

4.1.2 Paralelismo em CPU

Aproveitando os vários núcleos existentes nos processadores atuais, é possível utilizá-los no cálculo da resposta aos problemas MVC e MWVC. E como esses núcleos podem executar tarefas independentes de forma paralela, espera-se dividir o tempo de processamento pela quantidade de núcleos utilizados.

Para permitir a utilização de diversos núcleos na busca pelo conjunto mínimo de vértices, é necessário adaptar o processo já mencionado. A abordagem escolhida divide os vários subconjuntos V' a serem testados entre os núcleos, onde cada núcleo executa o processo de forma independente, encontrando a melhor resposta para sua parte do problema (ótimo local). Assim, ao final do processo, é verificado qual das respostas locais é a melhor (ótimo global). Desta forma, as etapas deste processo são: `filter`, executado em paralelo pelos diferentes núcleos; `reduce` locais, também executado em paralelo junto com o `filter`, onde cada núcleo encontra o seu ótimo local; e o `reduce` global, executado por um único núcleo (região crítica) comparando e agregando apenas os ótimos locais.

O algoritmo 2 exemplifica o processo descrito.

Esse paralelismo é possível uma vez que a verificação de se um subconjunto V' é uma cobertura de vértices válida é totalmente independente dos demais subconjuntos V'. É importante notar que a operação `reduce` possui as propriedades de comutatividade e associatividade, o que permite a sua execução sem necessidade de manter a ordem dos subconjuntos envolvidos. Outra característica relevante é que o subconjunto adotado como melhor resposta inicial ($V' = V$) se comporta como um elemento neutro, possibilitando que ele seja utilizado diversas vezes no processo, uma vez como melhor local para cada núcleo, e uma vez como melhor resposta global, sem alterar o resultado final.

Uma vez que as etapas `filter` e `reduce` locais, executadas em cada núcleo de CPU, dependem dos conjuntos de potenciais respostas a serem verificadas, existe a questão de como dividir os subconjuntos V' entre os núcleos. Existem duas formas ditas principais de fazer a divisão: a estática, onde é definido, de forma determinística, quais subconjuntos serão processados por quais núcleos; e a dinâmica, onde conforme os núcleos estejam disponíveis, recebem subconjuntos para serem processados. O detalhamento de cada opção destas formas são apresentados em profundidade junto aos experimentos realizados na subseção 4.1.5.

4.1.3 Paralelismo em GPU

Também é possível utilizar os vários núcleos da GPU para calcular a resposta do MVC ou MWVC, onde cada núcleo pode validar uma resposta diferente simultaneamente, uma vez que todos executarão as mesmas instruções. Contudo, a implementação em GPU, pode ter o seu ganho de desempenho prejudicado, devido as

Algoritmo 2: Resolução do MVC por força bruta com paralelismo

```
1  Função forcaBrutaParalela(G)
       Entrada: Grafo G = (V, A)
       Saída: conjunto de vértices
2      melhorGlobal ← G.V
3      paralelo
4          melhorLocal ← G.V
5          para cada em paralelo resposta ∈ respostasPossiveis faça
6              valido ← Verdadeiro
7              para cada (u, v) ∈ G.A faça
8                  se não (u ∈ resposta ou v ∈ resposta) então
9                      valido ← Falso
10                     interrompe
11                 fim
12             fim
13             se valido então
14                 se |resposta| < |melhorLocal| então
15                     melhorLocal ← resposta
16                 fim
17             fim
18         fim
19         região crítica
20             se |melhorLocal| < |melhorGlobal| então
21                 melhorGlobal ← melhorLocal
22             fim
23         fim
24     fim
25     retorna melhorGlobal
26 fim
```

otimizações que quebram o fluxo de execução entre os núcleos, quando uma resposta é identificada como inválida, por exemplo. Essa questão deve ser verificada e levada em consideração na adoção da abordagem de paralelismo ideal para a resolução desses problemas.

Assim, o processo utilizado nesta abordagem é semelhante ao utilizado no paralelismo em CPU, porém com as etapas `filter` e `reduce` locais sendo executadas na GPU. Ao final, a etapa `reduce` global é executada em um único núcleo de CPU, uma vez que a velocidade de processamento da CPU é maior que o da GPU.

Um detalhe a ser observado é que não é possível alocar todos os conjuntos de possíveis respostas (V') na memória da GPU para problemas grandes. Se cada possível resposta for representada em um inteiro de 8 bytes, para um problema com 30 vértices seriam necessários 8 GB ($2^{30} \times 8$) de memória da GPU, e para cada vértice a mais, a necessidade de memória dobraria. Desta forma, é necessário agrupar alguns conjuntos, de forma que cada núcleo de GPU calcule ótimos locais, permitindo assim

que as respostas possam ser alocadas na memória.

Para a abordagem em GPU será tratada unicamente a distribuição dos subconjuntos V' de forma estática, onde a atribuição dos subconjuntos para cada núcleo é feita de forma predeterminada.

4.1.4 Paralelismo híbrido

Com a utilização de vários núcleos de GPU, quanto mais ótimos locais a serem processados, melhor será a granularidade para o paralelismo. Porém, haverá também mais ótimos locais a serem processados sem paralelismo no `reduce` global. Visando paralelizar também o `reduce` global, é possível fazer um `reduce` parcial, onde diferentes núcleos de CPU podem comparar alguns ótimos locais, reduzindo a quantidade de ótimos locais a serem processados em um único núcleo (região crítica). Assim, essa abordagem utiliza duas técnicas de paralelismo diferentes, paralelismo em GPU e em CPU (solução híbrida).

4.1.5 Ajustes de parâmetros na força bruta

Esta seção abordará os experimentos realizados para verificar os parâmetros mais eficientes para a resolução dos problemas MVC e MWVC com força bruta aplicando paralelismo. Para os experimentos foi utilizado um computador com 2 processadores Intel Xeon E5-2640 v2, frequência 2 GHz, totalizando 16 núcleos (32 *threads*), com uma placa de vídeo NVIDIA Tesla K20m (2496 CUDA *cores*), utilizando o sistema operacional Debian GNU/Linux 10 (Buster). O software desenvolvido para a resolução dos problemas MVC e MWVC foi escrito em C e compilado com o GCC 8.3.0, utilizando a biblioteca OpenMP para paralelismo em CPU, e OpenCL para paralelismo em GPU (driver OpenCL 1.2 CUDA versão 440.33.01). A métrica de comparação utilizada foi o tempo de execução da função de busca da resposta. Os dados obtidos, juntamente com as análises estatísticas, podem ser verificados no Apêndice A.

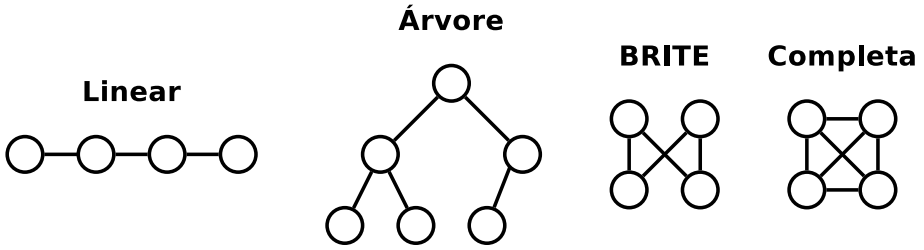
4.1.5.1 Topologias

Uma rede de computadores, com uma quantidade determinada de dispositivos, pode apresentar diferentes disposições de conexão entre seus dispositivos, tanto nos dispositivos que estão conectados, quanto na quantidade de interligações. Ao converter a rede para um grafo, isso é refletido na disposição e quantidade de arestas. Uma vez que o algoritmo de força bruta percorre as arestas do grafo, é razoável pensar que a disposição e quantidade de arestas pode influenciar no desempenho da função de resolução.

Visando cobrir diversas topologias, foi escolhido realizar testes com a menor e maior densidade de arestas possíveis para cada quantidade de vértices testados, de forma que essas densidades representem o melhor e pior casos em relação ao tempo

de resposta. Para a menor densidade de arestas foram escolhidas duas topologias: a primeira representando redes em cascateamento, que recebeu o nome de topologia linear; a segunda representando redes em estrela estendida, que recebeu o nome de topologia árvore. Redes em estrela são um caso particular da estrela estendida, e podem ser facilmente resolvidas com um conjunto contendo apenas o vértice central, por isso não foram abordadas diretamente. Para a maior densidade de arestas, a única topologia possível é um grafo completo, que representa uma rede onde todos os dispositivos possuem um enlace para cada outro dispositivo da rede (malha completa), que recebeu o nome de topologia completa. Para estudar uma densidade intermediária, também foi utilizado o *software* BRITE (MEDINA et al., 2001), que gera topologias representativas de redes de computadores e possui densidade entre o mínimo e o máximo. Um exemplo das topologias empregadas nos experimentos pode ser visto na Figura 19, enquanto todas as topologias geradas pelo BRITE (junto como todos os parâmetros utilizados) podem ser verificadas no Apêndice A.

Figura 19 – Topologias



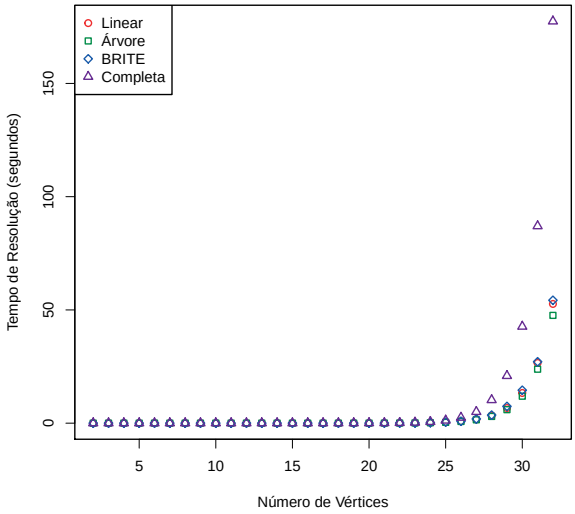
Fonte: Elaborada pelo autor, 2021.

Usando a implementação *Single Thread* do MVC, por ser a mais simples, e usada como base de referência, a resolução foi testada 30 vezes para cada topologia e cada tamanho de 2 até 32 vértices. A Figura 20 mostra o comparativo das médias do tempo de execução entre as topologias conforme a variação do tamanho do grafo, podendo-se perceber uma maior diferença nas médias conforme o aumento no tamanho do grafo. Comparando apenas os grafos com 26 vértices, o *boxplot* na Figura 21 permite observar diferenças nos tempos de resolução de todas as topologias, que são estatisticamente significativas ao se aplicar o teste de Tukey com um nível de confiança de 95%. A topologia mais demorada para se resolver foi a completa, seguida da BRITE, linear e árvore. Desta forma, foi verificado que o tempo de execução sofre influência da densidade de arestas do grafo, e espera-se que qualquer topologia tenha um tempo de execução menor ou igual ao tempo de execução da topologia completa (pior caso).

4.1.5.2 Single Thread

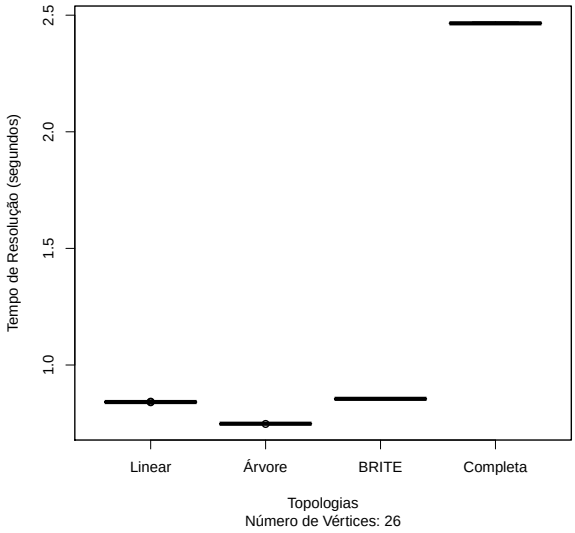
A implementação *Single Thread* não possui nenhum parâmetro a ser ajustado, porém ela será utilizada como base de comparação para as demais implementações, verificando se houve ganho com paralelismo, e de quanto foi.

Figura 20 – Tempo médio de resolução das topologias



Fonte: Elaborada pelo autor, 2021.

Figura 21 – Tempo de resolução das topologias com 26 vértices

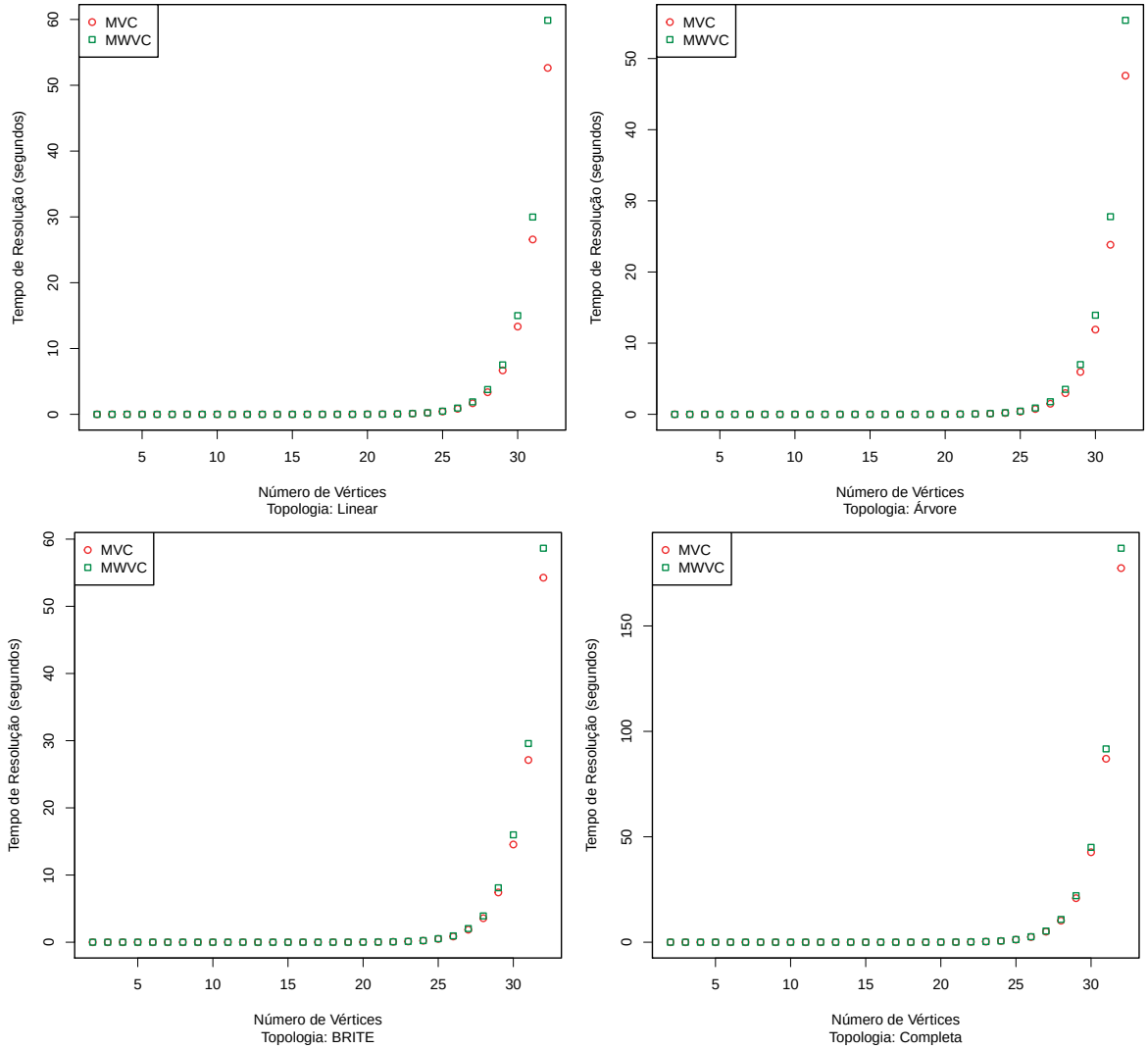


Fonte: Elaborada pelo autor, 2021.

Para medir os tempos de resolução dos problemas MVC e MWVC, as funções foram executadas 30 vezes para cada tamanho de grafo de 2 até 32, onde os pesos dos vértices foram definidos como 1, permitindo uma comparação direta entre o MVC e o MWVC. A Figura 22 mostra a média dos tempos para todas as topologias. O comportamento do tempo de execução dos dois problemas são bastante semelhantes, mesmo entre as diferentes topologias, tendo uma curva semelhante, embora cada uma na sua escala, conforme a densidade e disposição das arestas de cada topologia, com um tempo superior para o problema MWVC, o que pode ser explicado pela necessidade de ler da memória e somar os pesos dos vértices, quando comparado ao simples incremento na contagem de vértices do problema MVC. Essa diferença nos tempos de execução do MVC e do MWVC não é estatisticamente significativa quando aplicado o teste ANOVA com nível de confiança de 95%, porém essa diferença se torna

estatisticamente significativa quando analisa-se apenas um tipo de topologia.

Figura 22 – Tempo de resolução do *Single Thread*



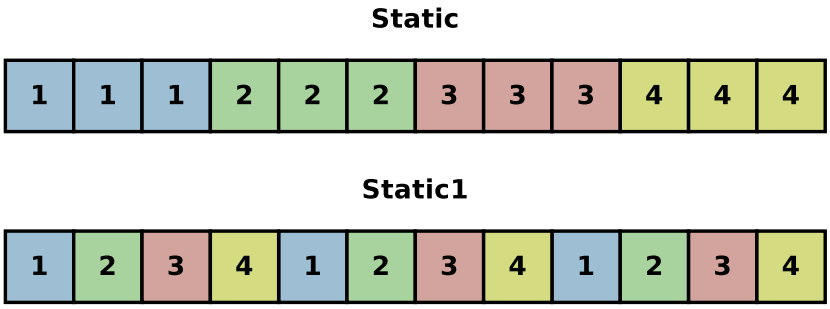
Fonte: Elaborada pelo autor, 2021.

4.1.5.3 Escalonadores CPU

Implementando o paralelismo em CPU, o OpenMP oferece diferentes formas de escalonamento, ou seja, diferentes formas de dividir os dados que serão processados entre os núcleos da CPU. O escalonador `static` divide os dados de forma determinística, atribuindo de forma prévia o que cada núcleo de CPU deverá processar. Esta divisão ocorre segundo o parâmetro `chunk size` que pode ser definido, informando a quantidade de dados sequenciais atribuídos a cada núcleo. O escalonador divide os dados em *chunks* de tamanho igual ao parâmetro `chunk size`, atribuindo-os sequencialmente, um *chunk* para cada núcleo, e ao chegar ao último núcleo, retoma a atribuição desde o primeiro. Caso o `chunk size` não seja definido, a divisão ocorrerá de forma que cada núcleo receba um único *chunk*, contendo dados contínuos, e que todos os *chunks* tenham tamanhos iguais, ou próximos, se a divisão não for exata.

Um exemplo pode ser visto na Figura 23, onde cada quadrado representa um dado a ser processado, e sua cor e número o núcleo de CPU que irá processá-lo. Nela, o escalonador `Static` representa o cenário onde o `chunk size` não foi definido, e o `Static1` onde o `chunk size` foi definido como 1.

Figura 23 – Escalonador estático



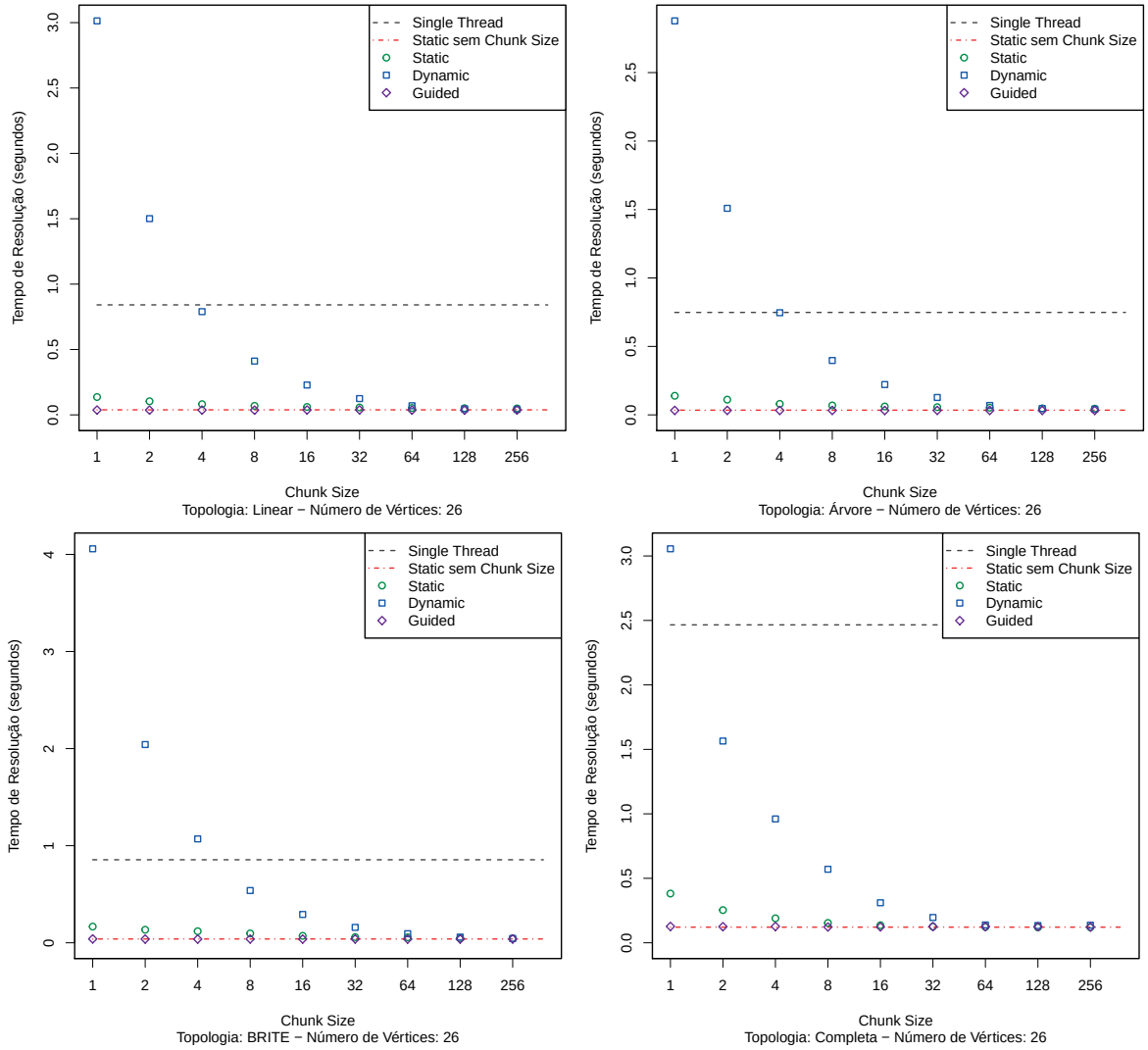
Fonte: Elaborada pelo autor, 2021.

O OpenMP também oferece o escalonador `dynamic`, que divide da mesma forma os dados em *chunks*, porém atribui apenas um *chunk* para cada núcleo. Estes núcleos, ao finalizar o processamento do *chunk* recebem o próximo até que todos os *chunks* tenham sidos processados. Esta estratégia visa não deixar os núcleos ociosos, uma vez que no `static` se um núcleo terminar seu processamento antes dos demais, não é feito um novo balanceamento dos *chunks*. Porém o `dynamic` introduz um *overhead* devido ao gerenciamento dos *chunks*. Visando reduzir esse *overhead*, o escalonador `guided` funciona da mesma forma que o `dynamic`, porém atribuindo *chunks* maiores no começo e reduzindo o tamanho até no final chegar no tamanho definido pelo `chunk size`. Desta forma, o *overhead* é reduzido no início do processamento, uma vez que será necessário menos comunicação entre os núcleos, mas balanceando mais adequadamente a carga ao final, visando não deixar os núcleos ociosos.

Para avaliar o comportamento dos escalonadores em função dos diferentes `chunk size` na paralelização em CPU, foram selecionados os tamanhos 1, 2, 4, 8, 16, 32, 64, 128, 256 e 512 para o `chunk size`. Utilizando um grafo completo com 26 vértices, cada combinação de escalonador com o `chunk size` foi executada 30 vezes na implementação paralelizada do MVC. A média dos tempos de execução para cada escalonador, conforme o `chunk size` podem ser vistos na Figura 24.

O que se pode observar inicialmente na Figura 24 é que o escalonador `dynamic` com `chunk size` pequenos teve um tempo de execução maior que a versão sem paralelismo, indicando uma perda de desempenho nesses cenários, e que uma divisão mais granular do processamento a ser feito acarretaria em *overheads* maiores. Portanto, para o escalonador `dynamic` é necessário utilizar `chunk size` maiores para se ter ganho com o paralelismo. Isso não é observado ao utilizar-se o escalonador `guided`, mesmo porque a maior parte de sua execução ocorre com tamanhos maiores de `chunk size`. O `static` também melhora seu desempenho quando utiliza-se `chunk size` maiores,

Figura 24 – Tempo de resolução dos escalonadores do OpenMP



Fonte: Elaborada pelo autor, 2021.

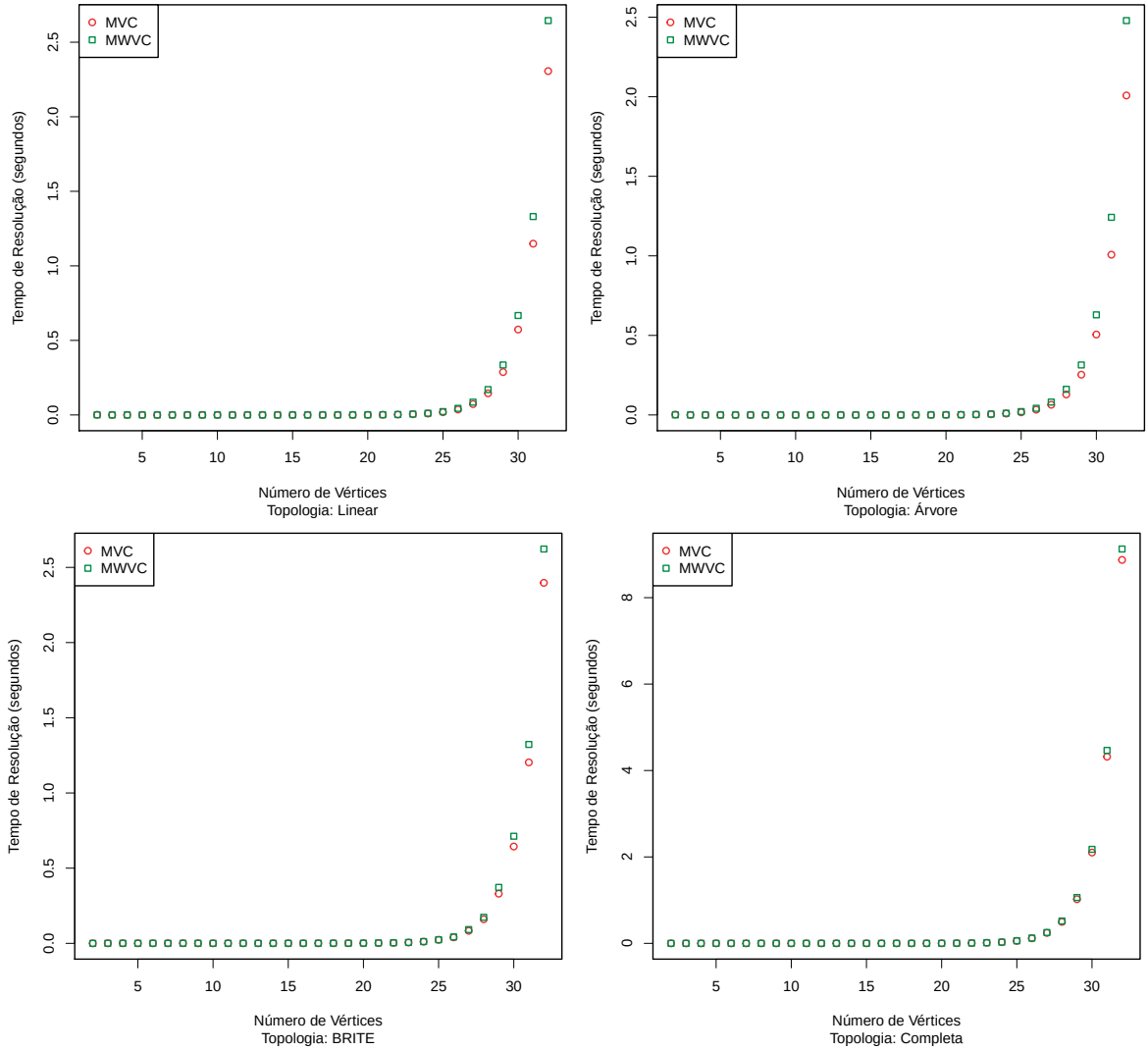
até que todos os dados caibam em um *chunk* para cada núcleo, que é o mesmo comportamento de quando o `chunk size` não é definido. Embora a única diferença estatisticamente significativa apontada pelo teste de Tukey com 95% de confiança foi apena do `dynamic` para os demais.

Porém considerando as menores médias e praticidade, o melhor escalonador determinístico para a solução paralelizada do MVC é o escalonador `static` sem definir o `chunk size`; e o melhor escalonador que trabalha ativamente fazendo o balanceamento dos *chunks* é o `guided` com `chunk size` 1, de forma que esses parâmetros também não precisam ser calculados conforme o tamanho do grafo, podendo ser utilizados para qualquer grafo. Quando o grafo apresenta algum desbalanceamento em seu processamento, o `guided` é a melhor opção por conseguir lidar melhor com o balanceamento de carga. Porém como isso não foi observado nos experimentos, será utilizado o escalonador `static` dado sua menor média.

Para o MWVC, os parâmetros são os mesmos, e ele apresenta comportamento

semelhante ao MVC, com um tempo de execução levemente superior. Portanto, em termos de escalonador e `chunk size`, deve-se utilizar os mesmos valores que o MVC. A Figura 25 mostra uma comparação do tempo de resolução dos dois problemas utilizando os melhores parâmetros na implementação em OpenMP.

Figura 25 – Tempo de resolução do OpenMP



Fonte: Elaborada pelo autor, 2021.

4.1.5.4 Escalonadores GPU

Para avaliar o paralelismo em GPU na solução do MVC e do MWVC, foi implementado o processo descrito na subseção 4.1.3 com a biblioteca OpenCL. Nesta biblioteca não existem tantas opções de escalonadores como no OpenMP, porém é possível implementar a distribuição sequencial ou alternada, replicando o comportamento demostrado na Figura 23. Devido ao limite de memória, é necessário trabalhar com os parâmetros de tamanhos de vetores utilizados pelo OpenCL para que ele divida os dados a serem processados entre os núcleos, de forma semelhante ao `chunk size`,

assim como definir que memória deve ser utilizada para cada dado para se obter o melhor desempenho da GPU.

No OpenCL, a tarefa a ser processada deve ser dividida em um vetor, que além de seu tamanho (tamanho global), deve possuir um tamanho local. Esses dois tamanhos são utilizados para distribuir a tarefa entre os núcleos da GPU, onde o vetor a ser processado é dividido em *chunks* de tamanho igual ao tamanho local, e cada *chunks* é atribuido a um grupo de núcleos da GPU.

A GPU possui sua própria memória, que o OpenCL divide em: `global`, que é a memória que pode ser lida ou escrita pela CPU, assim como por qualquer núcleo da GPU; `constant`, que é uma região da memória `global` específica para dados constantes; `local`, que é compartilhado entre um grupo de núcleos da GPU; `private`, que é específica de cada núcleo. É importante observar que para que os dados sejam processados, é necessário que sejam copiados para a memória da GPU.

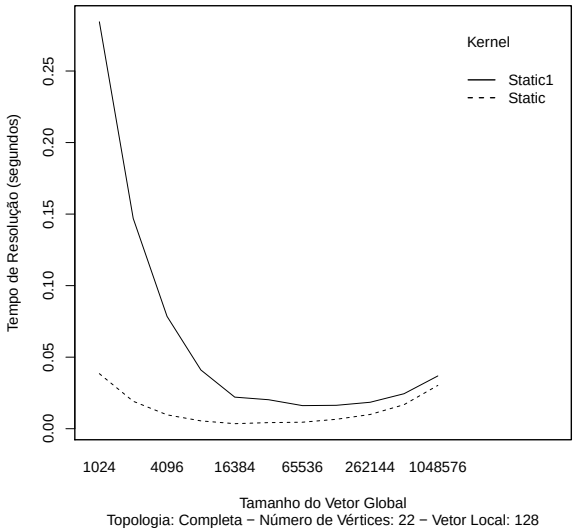
Em relação aos tipos de memória disponibilizados pelo OpenCL, a função utilizada para processar a resposta (chamada de `kernel` no OpenCL) recebe as seguintes informações: número de vértices, número de arestas, lista com as arestas e o endereço de memória onde deixar os ótimos locais calculados. Destes, a lista de arestas é passada para a memória `constant`, uma vez que ela pode ser acessada por todos os núcleos, e permite que seja feito *cache*, uma vez que seu valor não recebe alterações. A resposta dos ótimos locais é guardada na memória `global` que é acessada apenas ao final do processo e permite ser acessada pelo processador para continuar a resolução do problema. As demais informações são valores pequenos, que podem ser facilmente copiados, e por isso é utilizada a memória `private` que permite um melhor desempenho.

É possível simular o comportamento do escalonador `static` com `chunk size` 1 (nomeado como `Static1`) e não definido (nomeado como `Static`) no OpenCL. Para tanto, foram implementados funções diferentes (`kernels`), representando diferentes formas de se percorrer os dados. Porém a divisão dos dados em *chunks* ocorre através dos tamanhos definidos para os vetores global e local, e sua atribuição aos núcleos de GPU varia conforme a implementação do OpenCL e *hardware* utilizados. Para os testes foram selecionados os seguintes tamanhos de vetor global: 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288 e 1048576, e os seguintes tamanhos para o vetor local: 1, 2, 4, 8, 16, 32, 64, 128, 256 e 512.

Desta forma, foram executadas 30 vezes cada combinação de `kernel`, tamanho de vetor global e local. Assim foi observado que todos os tempos de execução do `kernel Static` foram inferiores quando comparados aos mesmos parâmetros com o `kernel Static1` (Figura 26). Isso implica nesta ser a melhor opção para a implementação para o *hardware* utilizado nestes testes. Considerando apenas o `kernel Static`, a Figura 27 demostra a interação entre os diferentes tamanhos de vetor global e local, onde o melhor desempenho foi observado no tamanho 16384 para o

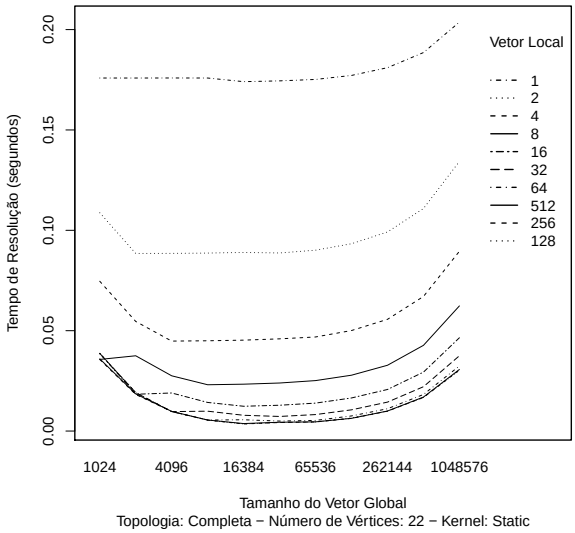
vetor global, embora não exista uma diferença estatisticamente significativa dele para as opções 4096 e maiores; e 128 para o vetor local, embora não exista uma diferença estatisticamente significativa dele para as opções 16 e maiores. Desta forma, por apresentar as melhores médias, foi escolhido o `kernel Static` com tamanho de vetor global 16384 e vetor local 128 como a melhor opção para o *hardware* testado.

Figura 26 – Tempo de resolução conforme o *kernel* do OpenCL



Fonte: Elaborada pelo autor, 2021.

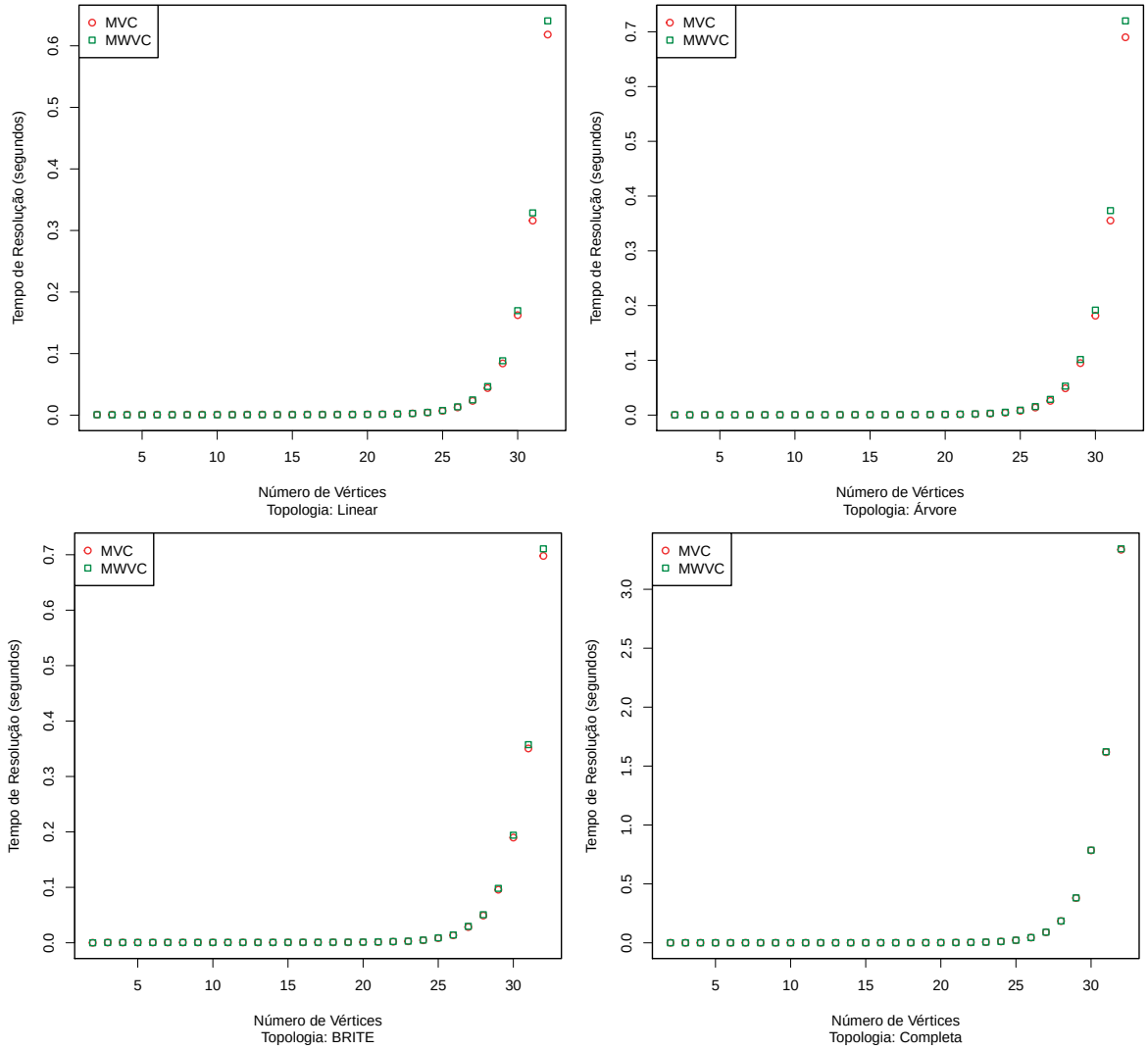
Figura 27 – Tempo de resolução conforme o tamanho dos vetores do OpenCL



Fonte: Elaborada pelo autor, 2021.

Para o MWVC, os parâmetros são os mesmos, portanto deve-se utilizar os mesmos parâmetros do MVC devido à similaridade de comportamento dos dois problemas. A Figura 28 mostra uma comparação do tempo de resolução dos dois problemas utilizando os melhores parâmetros na implementação em OpenCL.

Figura 28 – Tempo de resolução do OpenCL



Fonte: Elaborada pelo autor, 2021.

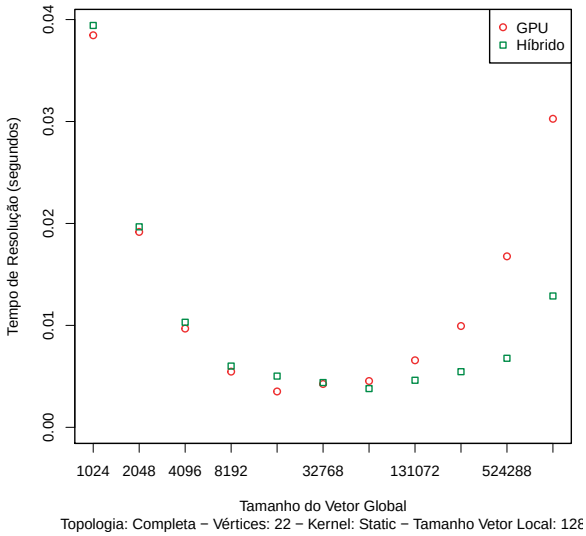
4.1.5.5 Paralelismo híbrido

A solução de paralelismo híbrido entre GPU e CPU visa melhorar o desempenho da parte executada em CPU para permitir uma divisão maior da tarefa a ser processada em GPU. Para isso são utilizados os melhores parâmetros já observados nos casos de paralelismo em CPU e GPU isoladamente, variando apenas o tamanho do vetor global, o que permitiria extrair o melhor dessa implementação de paralelismo.

Esta implementação foi executada 30 vezes para a topologia completa com 22 vértices para cada tamanho de vetor global testado (1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288 e 1048576). A Figura 29 mostra uma comparação das médias desta implementação para a implementação com paralelismo puramente em GPU, onde é observado um *overhead* no paralelismo híbrido nos tamanhos até 16384, ponto ao qual o tempo de execução da implementação em GPU começa a crescer, enquanto o paralelismo híbrido continua diminuindo seu tempo até o tamanho 65536, onde também começa a crescer. A partir do ponto em que

os tempos de execução das duas implementações crescem há uma inversão, e a implementação híbrida apresenta melhores tempos, com uma taxa de crescimento do tempo de execução abaixo da taxa da implementação com paralelismo puramente em GPU. Porém a implementação híbrida não conseguiu apresentar alguma configuração de parâmetros que resulte em tempos melhores do que a implementação que utiliza paralelismo puramente em GPU, além de possuir um código mais complexo, resultando em nenhum ganho adicional ao se utilizar essa implementação.

Figura 29 – Comparação dos tempos do paralelismo em GPU e híbrido



Fonte: Elaborada pelo autor, 2021.

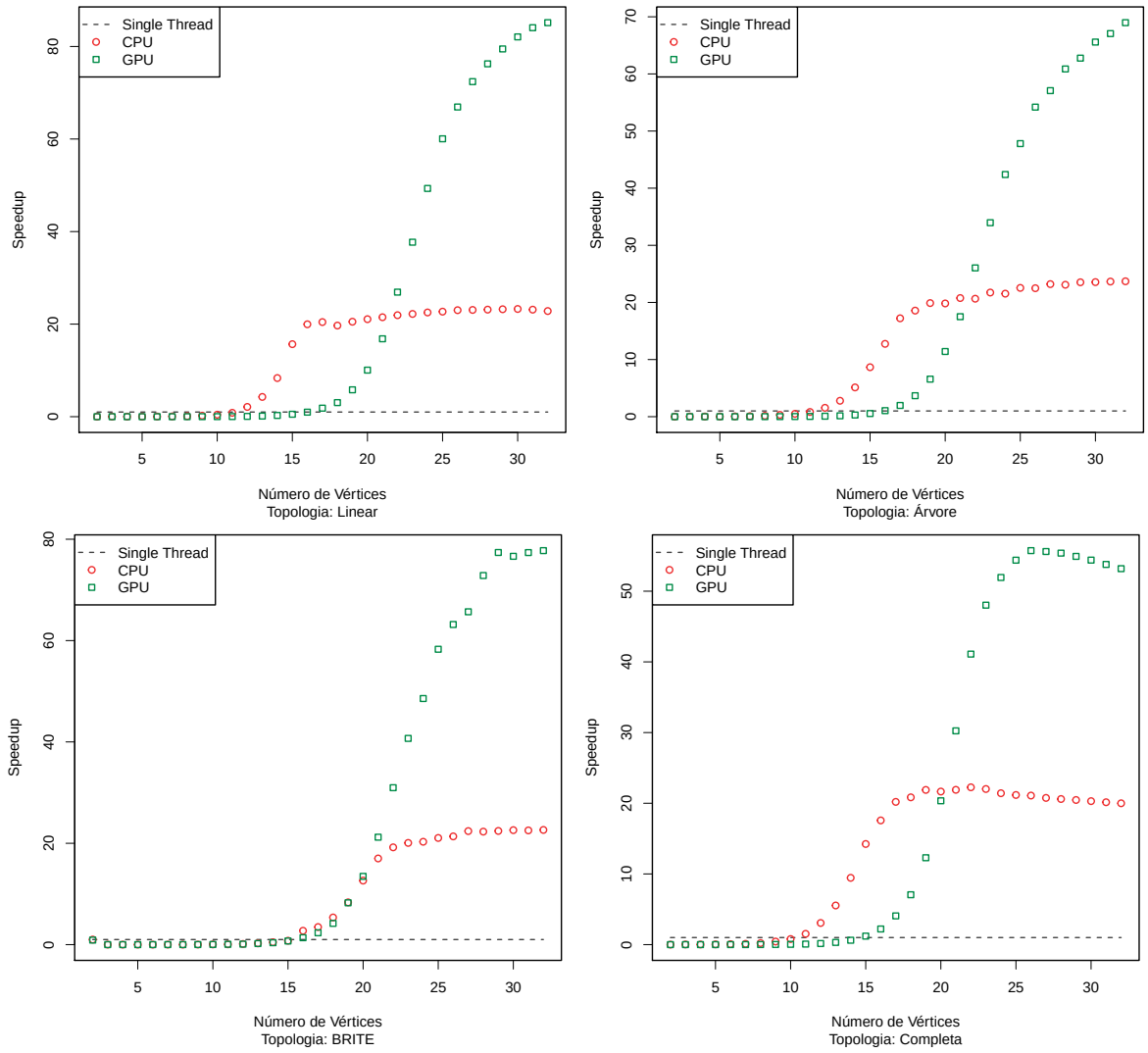
4.1.5.6 Speedup

O *speedup* é uma métrica utilizada para comparar o tempo de execução de diferentes soluções, apontando quantas vezes mais rápido (ou mais lento quando menor que um) uma determinada solução executou em relação a sua base de comparação. Seu cálculo se dá dividindo o tempo de execução da base de comparação (*baseline*) pelo tempo de execução da solução que está sendo comparada.

A Figura 30 mostra a comparação dos tempos das melhores opções de paralelismo em CPU e GPU com o tempo da implementação *Single Thread* (*baseline*) para o problema do MVC. Nele observa-se que a implementação *Single Thread* é a mais rápida para problemas pequenos, onde o *overhead* de controle do processamento introduzido pelo paralelismo torna as versões paralelas mais demoradas. Porém, ao aumentar o tamanho do problema, as opções com paralelismo começam a apresentar melhores tempos. O *speedup* do paralelismo em CPU ficou um pouco maior que o número de núcleos do processador, indicando um bom aproveitamento do *hyperthread* disponível no *hardware*. O paralelismo em GPU foi a opção que apresentou os melhores tempos de execução para problemas maiores, alcançando entre as topologias testadas uma média de *speedup* superior a 64, o que daria para resolver um grafo com

6 vértices a mais do que a versão *Single Thread* no mesmo período de tempo, visto o crescimento exponencial do tamanho do problema.

Figura 30 – *Speedup* das técnicas de paralelismo no MVC



Fonte: Elaborada pelo autor, 2021.

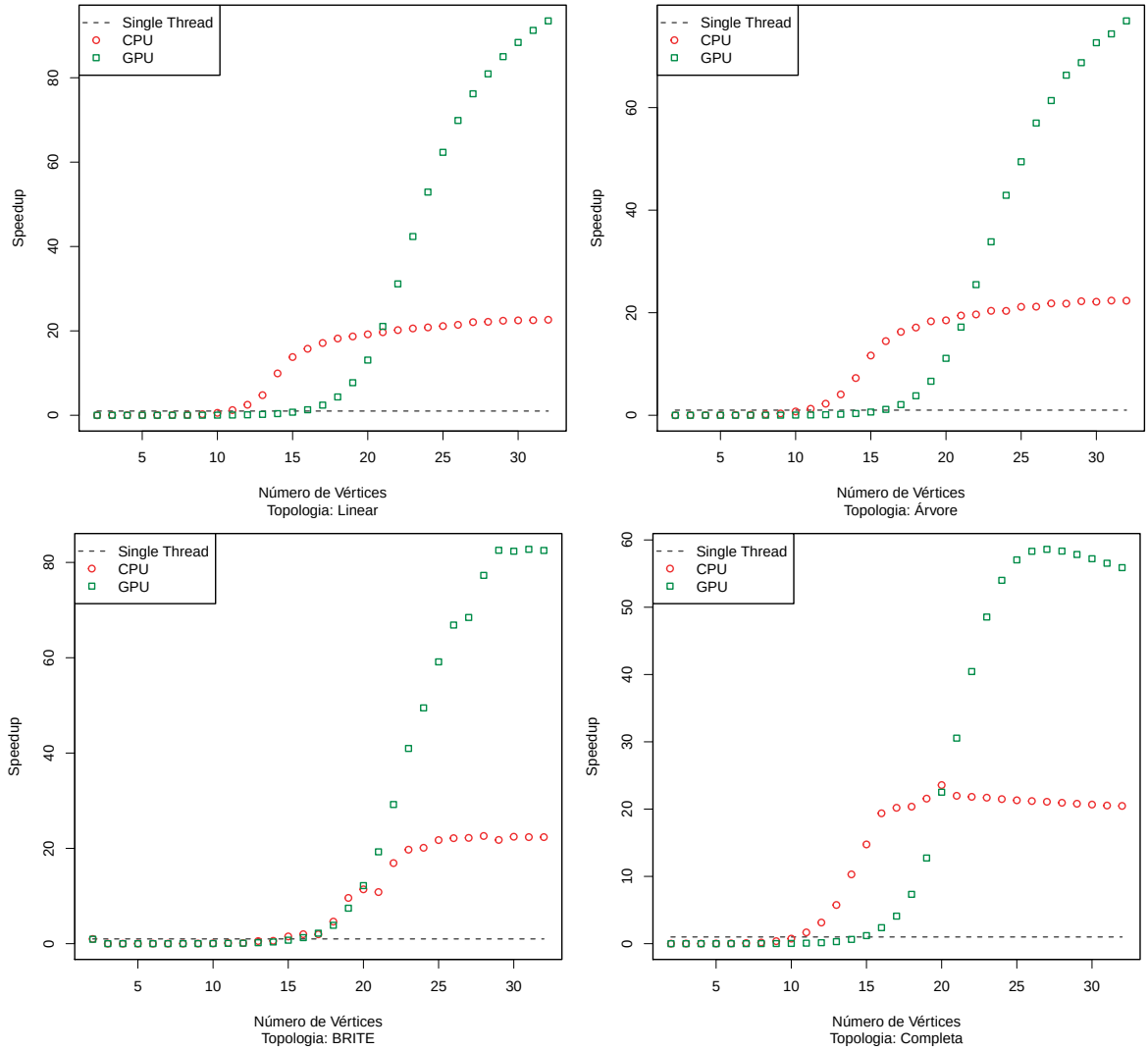
A Figura 31 mostra a mesma comparação, porém para o problema MWVC, que apresenta resultados semelhantes aos do problema MVC.

De forma geral, para o *hardware* testado, qualquer problema com até 15 vértices pode ser resolvido utilizando força bruta em *Single Thread*, enquanto problemas com mais vértices podem ser resolvidos com o paralelismo em GPU, utilizando o `kernel Static` com tamanho de vetor global 16384 e tamanho de vetor local 128.

4.1.6 Regressão do tempo de execução

A força bruta sempre consegue encontrar a resposta ótima para os problemas MVC e MWVC. Porém, conforme o tamanho do problema aumenta, aumenta-se também o tempo necessário para encontrar essa resposta, que como já visto, ocorre de forma exponencial. Isso pode tornar essa técnica inviável para problemas grandes,

Figura 31 – *Speedup* das técnicas de paralelismo no MWVC



Fonte: Elaborada pelo autor, 2021.

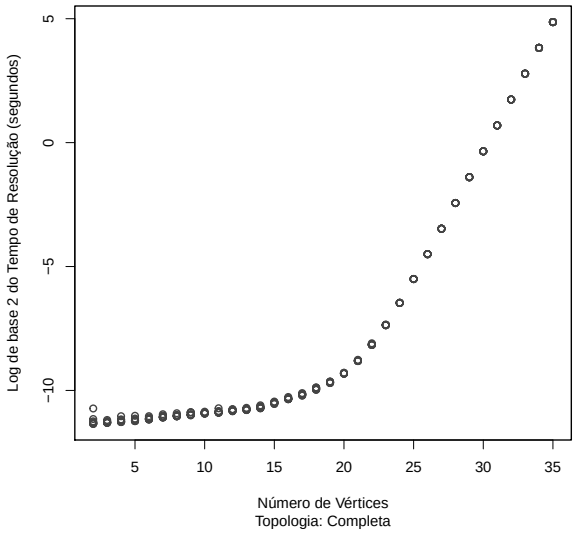
necessitando de muito tempo para ser calculado. Entretanto ela funciona muito bem para problemas cujo tempo de resolução esteja dentro do que for considerado aceitável.

Uma forma de verificar a viabilidade da utilização da força bruta é estimando o tempo de execução, verificando se o mesmo está dentro do que se pode esperar para se obter a resposta. Uma forma de fazer essa estimativa, é ajustando uma regressão com o tempo de execução de problemas menores, e extrapolá-la para problemas maiores. Assim, obtém-se uma estimativa do tempo que seria necessário para calcular a resposta do problema desejado em um determinado *hardware*.

Para modelar uma regressão foi escolhida a topologia completa (Figura 19), visto que ela é a mais demorada para se calcular a resposta, funcionando como um limite, indicando em até quanto tempo se espera calcular a resposta. Devido ao comportamento exponencial do tempo de execução, é possível fazer uma transformação logarítmica do mesmo, visando aproximar a curva do tempo de execução a um polinômio que pode ser obtido por meio de uma regressão. Para fazer essa regressão são

necessários os dados de tempo de execução. Para a geração de tais dados foi escolhida a solução de força bruta com paralelismo em GPU, uma vez que ela apresenta o melhor desempenho para problemas maiores nos experimentos da subseção 4.1.5.6. Assim, no mesmo *hardware* do experimento anterior, foram executadas, dez repetições do MVC de uma topologia completa com dois vértices, repetindo as execuções com topologias com um vértice a mais que a anterior, até que se ultrapasse o tempo de um minuto para calcular a resposta, ponto onde o teste foi finalizado. Esta quantidade de repetições e limite de tempo visa não exigir muito tempo na obtenção dos dados. Os resultados obtidos, com a transformação logarítmica de base 2 já aplicada, pode ser visto na Figura 32.

Figura 32 – Transformação logaritmica do tempo de execução do MVC



Fonte: Elaborada pelo autor, 2021.

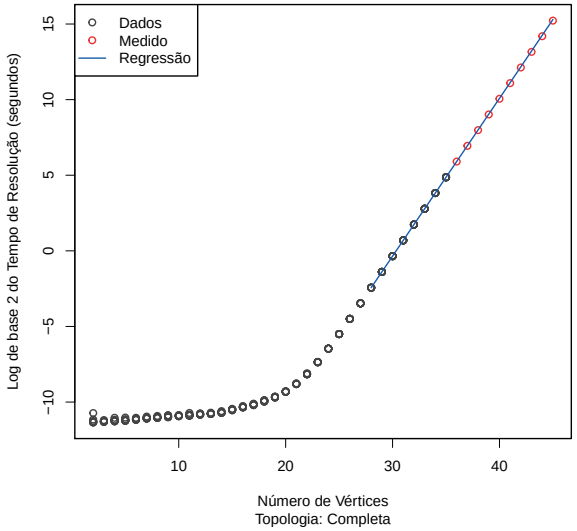
No gráfico é possível observar alguma variação para tamanhos pequenos, porém uma tendência de se aproximar a uma reta conforme o tamanho da topologia aumenta. Como o objetivo da regressão é extrapolar para topologias maiores, foram descartados os tempos das topologias que executaram em menos de 0,1 segundos, focando a regressão na tendência que se observa nos tamanhos maiores, mas mantendo um volume de dados significativo para estabelecer uma regressão.

Uma questão importante é o grau do polinômio a ser ajustado na regressão. Embora graus maiores possam se ajustar melhor às variações dos dados obtidos, suas curvas tendem a se afastar mais da tendência de reta observada do que graus menores quando se extrapola o tamanho da topologia. Desta forma, optou-se pera regressão linear. Assim, para o *hardware* testado, o tempo de execução t pode ser previsto com o modelo $t = 2^{1{,}043n - 31.651}$ (considerando a aplicação da transformação logarítmica), onde n é o número de vértices do grafo. Os dois coeficientes calculados e o modelo apresentam significância estatística.

Com a regressão modelada, foram medidos os tempos de execuções com topologias maiores para validar se as previsões utilizando a regressão podem ser

consideradas. A Figura 33 mostra a regressão linear ajustada anteriormente, junto aos tempos de execução das topologias maiores, onde é possível observar uma boa aproximação, com diferença de até 5,5% entre o valor calculado e o medido. A regressão também obteve um valor superior ao medido, e como se espera que para outra topologia com a mesma quantidade de vértices seu tempo de execução seja menor, visto que a topologia completa é o tipo de topologia que necessitou de maior tempo de execução quando comparada as outras, essa regressão funciona muito bem como um limite superior do tempo de execução.

Figura 33 – Resultado da regressão do MVC



Fonte: Elaborada pelo autor, 2021.

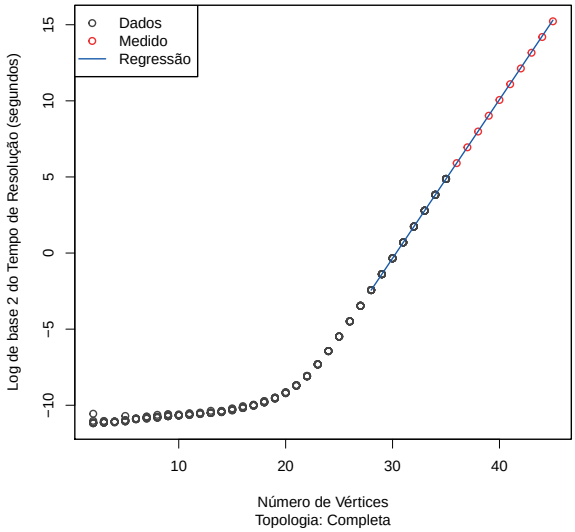
Em relação ao MWVC, foi repetido todo o procedimento aplicado ao MVC, resultando no modelo $t = 2^{1,043n-31,637}$. A Figura 34 mostra que o comportamento dos dois problemas é similar, inclusive com a maior diferença entre os valores previstos e medidos sendo de 5,3%. Isso indica que a regressão modelada para o MWVC também funciona como um limite superior do tempo de execução.

4.2 ALGORITMO GULOSO

Outra opção para resolver os problemas MVC e MWVC é a utilização de um algoritmo guloso, que consiste em tomar as melhores escolhas locais. Escolhas locais reduzem o espaço de busca do algoritmo, reduzindo também seu tempo de execução. Porém, para esses problemas, ela também pode tirar a resposta ótima do espaço de busca, e caso isso ocorra, levar a respostas não otimizadas, embora essas respostas continuem sendo coberturas de vértices válidas.

Uma forma de implementar o algoritmo guloso para o MVC é incluindo no conjunto de vértices da resposta o vértice que possui a maior quantidade de arestas incidindo sobre ele, visando cobrir o maior número de arestas. Uma vez que um vértice tenha sido incluído no conjunto, deve-se remover suas arestas do grafo, repetindo o

Figura 34 – Resultado da regressão MWVC



Fonte: Elaborada pelo autor, 2021.

processo até que todas as arestas tenham sido removidas (cobertas por algum vértice). O algoritmo 3 modela esse processo.

Algoritmo 3: Algoritmo guloso para encontrar uma cobertura de vértices

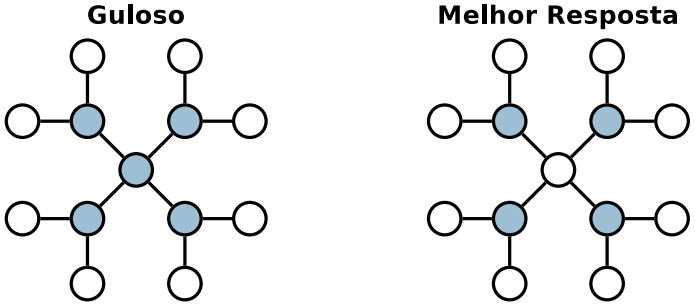
```
1  Função algoritmoGuloso(G)
       Entrada: Grafo G = (V, A)
       Saída: conjunto de vértices
2      arestas ← G.A
3      resposta ← ∅
4      enquanto |arestas| > 0 faça
5          graus ← {v ← 0 | v ∈ G.V}
6          para cada (u, v) ∈ arestas faça
7              graus[u] ← graus[u] + 1
8              graus[v] ← graus[v] + 1
9          fim
10         vertice ← indiceDeMaiorValor(graus)
11         arestas ← {(u, v) | (u, v) ∈ arestas e u ≠ vertice e v ≠ vertice}
12         resposta ← resposta ∪ {vertice}
13     fim
14     retorna resposta
15 fim
```

Para o MWVC, como visa-se otimizar a soma dos pesos em vez da quantidade de vértices, é necessário adotar outra métrica na escolha do vértice a ser adicionado ao conjunto. Visando cobrir a maior quantidade de arestas por peso adicionado ao conjunto, pode-se optar pelo vértice que tiver o menor valor na relação do seu peso pela quantidade de arestas incidindo sobre ele. Enquanto os demais passos do algoritmo podem continuar os mesmos.

Entretanto o espaço de busca estritamente local pode induzir a respostas não

otimizadas, como já mencionado. A Figura 35 demonstra um caso onde o algoritmo guloso não consegue encontrar a melhor resposta para o problema do MVC, encontrando uma resposta com um vértice a mais que a resposta otimizada. O mesmo se repete com o MWVC ao assumir o mesmo peso para todos os vértices desse mesmo grafo.

Figura 35 – Exemplo de resposta não ótima do algoritmo guloso



Fonte: Elaborada pelo autor, 2021.

Desta forma, o algoritmo guloso pode ser interessante para se ter um resultado rápido para a migração. Ele pode ser executado antes da força bruta, para se ter uma ideia de resultado, enquanto a força bruta é executada. Entretanto, uma migração feita a partir de seus resultados pode necessitar migrar mais dispositivos ou envolver uma soma maior de pesos quando comparado com a resposta ótima, podendo refletir em maior custo ou trabalho para a migração.

4.3 3-OPT

O 3-opt é uma heurística da classe n-opt para problemas que podem ser modelados em um grafo. O n-opt, tendo como base uma resposta inicial, trabalha com n elementos, neste caso vértices do grafo, tentando encontrar uma otimização a cada execução (BLAZINSKAS; MISEVICIUS, 2011). O 3-opt é a versão que trabalha com três elementos dessa classe. Desta forma, é possível utilizar o 3-opt visando melhorar a resposta encontrada pelo algoritmo guloso, embora também não se tenha garantias da resposta ótima.

A escolha do 3-opt ocorre por ele ser uma heurística simples, e não precisar de ajuste em seus parâmetros de execução. Algumas meta-heurísticas, como o algoritmo genético, possuem mecanismos para testar combinações de respostas que não levam em conta as características do MVC, ou de um problema modelado em um grafo, não sendo tão eficientes quanto uma heurística específica, que leva em consideração as características do problema (CAI; SU; CHEN, 2010). Outras heurísticas específicas para o MVC necessitam de algum parâmetro para sua execução, como o COVER que busca uma solução com k vértices, onde o k deve ser informado previamente (RICHTER; HELMERT; GRETTON, 2007), e o EWLS que possui um parâmetro $delta$, o qual atua no tamanho do espaço de busca, e seu valor ideal varia conforme o grafo (CAI; SU; CHEN, 2010). Quando comparado ao 2-opt, o 3-opt permite uma análise

ampla da interação entre os vértices adjacentes, porém sem a necessidade de optar por uma disposição específica desses vértices, como um vértice central com adjacentes ou um caminho como seria necessário para o 4-opt ou maiores. Desta forma, o 3-opt pode ser executado para qualquer grafo sem uma análise prévia, ou que seja necessário algum conhecimento prévio para extrair o melhor da heurística.

A resolução do problema MVC utilizando o 3-opt pode ser dividida em alguns passos. O primeiro passo é resolver o problema utilizando o algoritmo guloso, o que permite iniciar o processo já com algum nível de otimização, principalmente quando comparado a iniciá-lo com a pior resposta possível (conjunto com todos os vértices do grafo). O segundo passo é selecionar três vértices a serem analisados. Eles podem ser escolhidos aleatoriamente, porém devem ser adjacentes, o que permite analisar a interação que existe entre eles ao remover ou adicioná-los no conjunto de resposta. Isso é necessário, uma vez que ao remover o vértice que cobria uma aresta, seu adjacente deve estar no conjunto para ser uma resposta válida. O terceiro passo é analisar as partes de um conjunto com esses três vértices, que nada mais é do que todos os subconjuntos que podem ser formados com base nesses três vértices, podendo incluir os três vértices, dois vértices, apenas um deles, ou nenhum dos vértices. Cada uma dessas partes deve ser analisada se ela representa uma resposta válida, porém como esses vértices envolvem uma região específica do grafo, e já se possui anteriormente uma resposta válida, é possível verificar apenas essa região e imediações para saber se a parte quando unida à resposta válida anterior continua sendo válida. Entre as partes que forem válidas, deve-se optar por aquela que quando unida a resposta anterior, contiver a menor quantidade de vértices. Para a próxima execução do 3-opt, retorna-se ao segundo passo, com a melhor resposta substituindo a resposta do algoritmo guloso nesse processo. O algoritmo 4 modela o processo descrito.

Uma otimização possível para esse algoritmo é alterar a forma como o grafo está representado. Enquanto o algoritmo da força bruta percorre todas as arestas do grafo, o 3-opt percorre as adjacências dos vértices. Desta forma, a representação do grafo que antes envolvia uma lista das arestas (par ordenado com os vértices da aresta), pode ser trocada por uma lista de adjacências de cada vértice, que embora represente a aresta duas vezes (uma em cada lista de adjacência de seus vértices), já possui pré calculado as adjacências de cada vértice, agilizando as pesquisas efetuadas pelo algoritmo do 3-opt.

Em relação a paralelização deste algoritmo, cada execução do 3-opt precisa verificar oito possibilidades de resposta (2^3), e considerando apenas uma parte reduzida do grafo (busca local), o que torna essas respostas mais rápidas de serem verificadas que a mesma quantidade de respostas na força bruta (busca global). Desta forma, paralelizar essas verificações causa um *overhead* de controle, semelhante ao visto nas implementações com força bruta, onde o escalonador *dynamic* precisou de uma carga

Algoritmo 4: Resolução do MVC pelo 3-opt

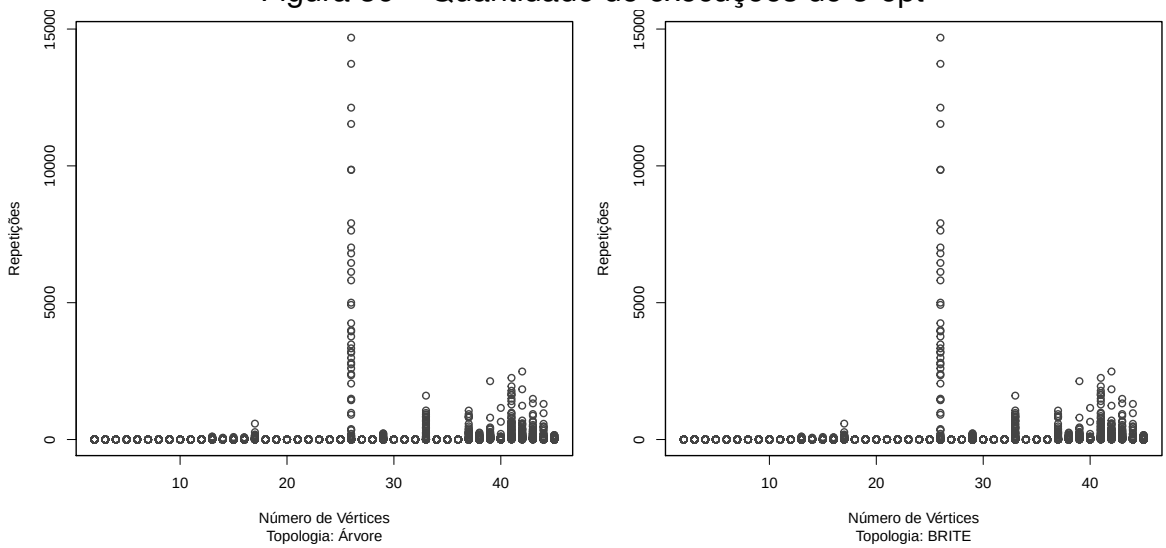
```
1  Função 3-opt(G, repeticoes)
       Entrada: Grafo G = (V, A)
       Entrada: repeticoes ∈ ℕ
       Saída: conjunto de vértices
2      resposta ← algoritmoGuloso(G)
3      para cada repeticao ∈ {1, 2, ···, repeticoes} faça
4          a ← sortear(G.V)
5          b ← sortearVerticeAdjacente({a}, G)
6          c ← sortearVerticeAdjacente({a, b}, G)
7          resposta ← resposta \ {a, b, c}
8          melhor ← {a, b, c}
9          para cada parte ∈ partes({a, b, c}) faça
10             cobertura ← True
11             para cada u ∈ {a, b, c} faça
12                 se u ∈ parte então
13                     continua
14                 fim
15                 para cada v ∈ verticesAdjacentes(u) faça
16                     se v ∈ (resposta ∪ parte) então
17                         continua
18                     fim
19                     cobertura ← False
20                     interrompe 2
21                 fim
22             fim
23             se cobertura e |parte| < |melhor| então
24                 melhor ← parte
25             fim
26         fim
27         resposta ← resposta ∪ melhor
28     fim
29     retorna resposta
30 fim
```

de processamento para verificar aproximadamente quatro respostas para compensar esse *overhead*, limitando bastante o ganho que se poderia alcançar com o paralelismo nesse algoritmo.

A quantidade de vezes que o 3-opt será executado é um parâmetro passado para o algoritmo. Quanto mais execuções, maior a possibilidade do 3-opt encontrar respostas mais otimizadas, porém levará mais tempo para executar. Também é possível definir por quanto tempo o 3-opt tentará otimizar a resposta, parando ao atingir o limite desejado. Considerando as topologias em que o algoritmo guloso não conseguiu encontrar a resposta ótima para algum caso, a Figura 36 mostra a quantidade de execuções necessárias para o 3-opt atingir a resposta ótima encontrada pela força

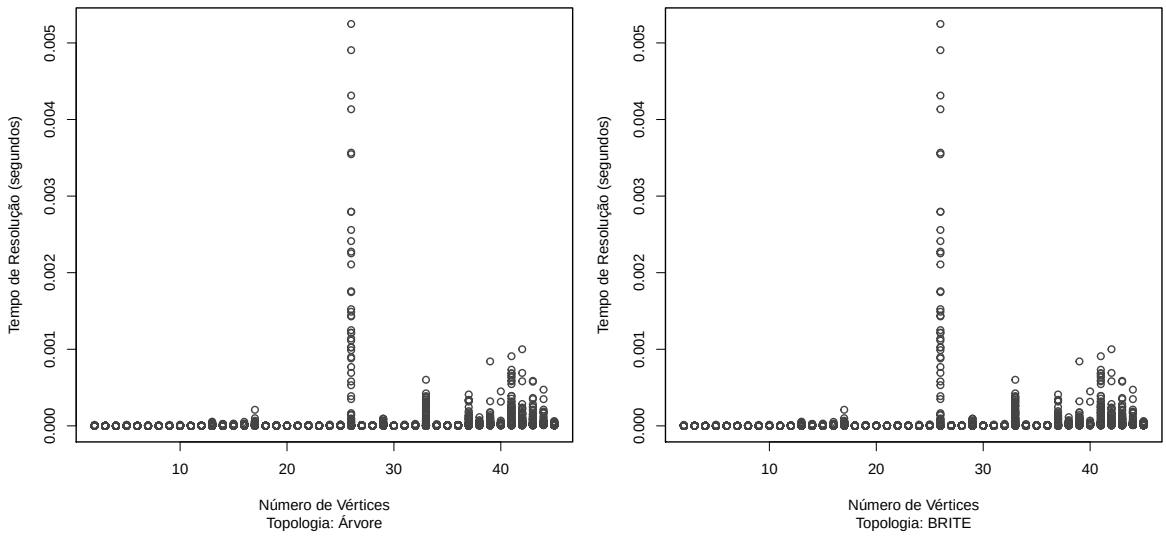
bruta, enquanto a Figura 37 mostra o tempo de execução necessário para esses mesmos casos. O algoritmo 3-opt foi implementado em Rust (compilado pelo rustc 1.48.0) e executado no mesmo servidor descrito na subseção 4.1.5. Nesses testes é possível observar que alguns grafos precisaram de mais execuções que outros para encontrar a resposta ótima, mesmo possuindo menos vértices. Isso é explicado por alguns grafos terem otimizações mais difíceis de serem encontradas do que outros, por exemplo, precisando que uma resposta específica seja encontrada, e que no ciclo seguinte determinados vértices sejam selecionados para que uma resposta mais otimizada seja encontrada, o que ocorre aleatoriamente.

Figura 36 – Quantidade de execuções do 3-opt



Fonte: Elaborada pelo autor, 2021.

Figura 37 – Tempo de execução do 3-opt



Fonte: Elaborada pelo autor, 2021.

Em relação ao MWVC, a única alteração necessária é na comparação entre as opções válidas, considerando como a melhor opção aquela que tiver a menor soma de

pesos, em vez da quantidade de vértices como no MVC. Os demais passos continuam iguais aos descritos no algoritmo 4.

Desta forma, o 3-opt pode ser utilizado para melhorar a resposta obtida do algoritmo guloso, tanto para o MVC, quanto para o MWVC. Ele consegue otimizar a resposta do algoritmo guloso em um tempo muito menor do que a força bruta leva para executar, principalmente para grafos com muitos vértices, embora não dê a certeza da resposta ótima. Assim o 3-opt é uma boa opção para os grafos que levariam muito tempo para se encontrar a resposta por força bruta.

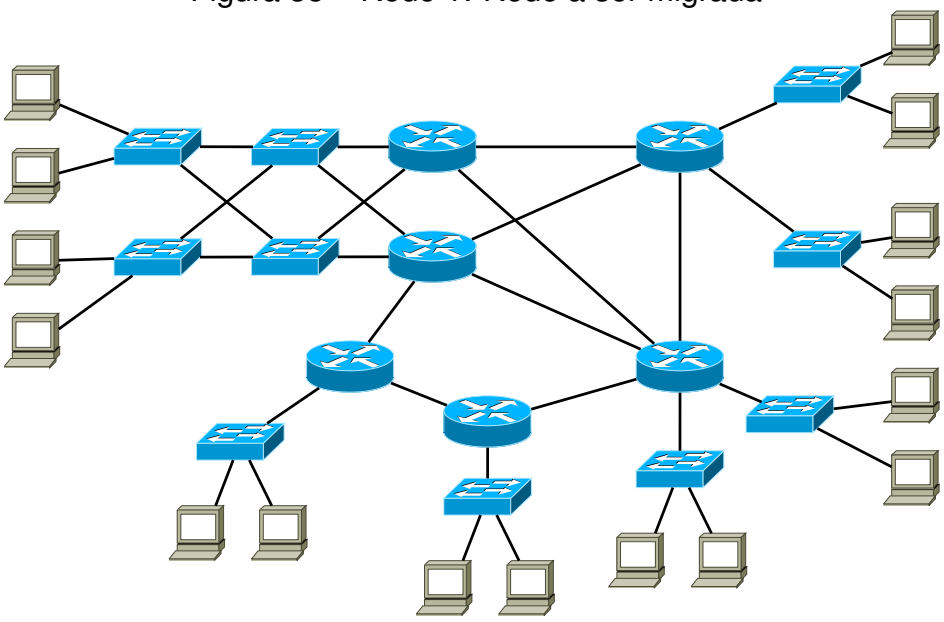
5 EXEMPLOS DE MIGRAÇÃO

Com o processo de migração modelado, e discutido como executar cada uma de suas etapas, este capítulo apresenta a execução do processo descrito na seção 3.2 em algumas redes, de forma a exemplificar sua aplicação. Isto ocorre de forma a verificar os resultados do processo, analisando sua utilidade ao adicionar funcionalidades em uma rede de computadores por meio de SDN.

5.1 REDE 1

Para a primeira execução do processo, é proposta a migração da rede presente na Figura 38. Nesse caso, a motivação para a migração é o monitoramento do estado e uso dos enlaces da rede através de uma solução SDN. Considerando que os dispositivos de encaminhamento presentes atualmente na rede não possuem suporte a SDN, será necessária a substituição dos mesmos para a implementação dessa funcionalidade. Assim, o tipo de otimização desejada na migração é o tipo i, onde se reduz o número de dispositivos a serem migrados, implicando num custo menor na aquisição de novos dispositivos.

Figura 38 – Rede 1: Rede a ser migrada

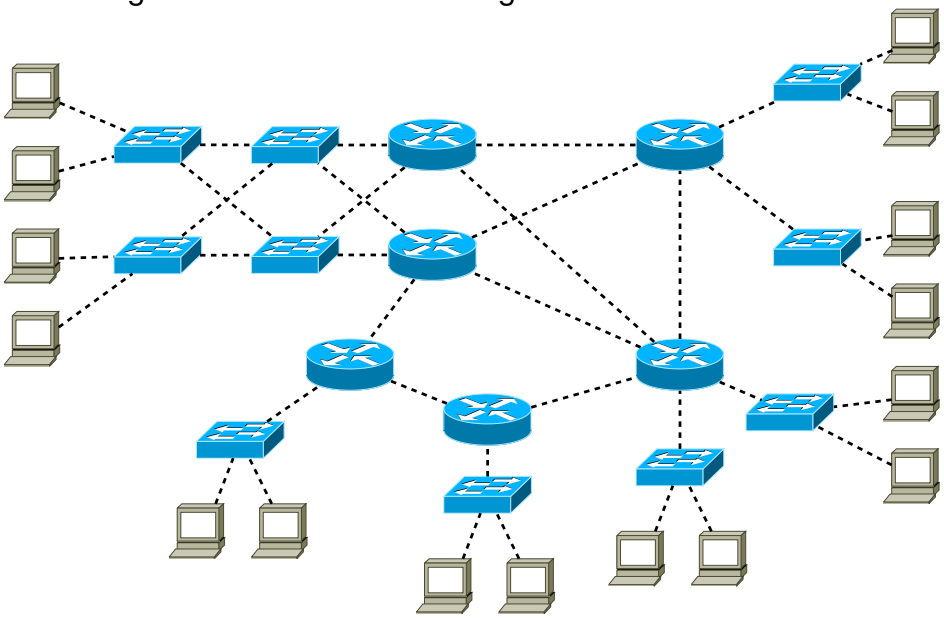


Fonte: Elaborada pelo autor, 2021.

Iniciando o processo pela etapa de modelagem da rede e funcionalidades (descrita na subseção 3.2.1), são executadas todas as suas subetapas. Assim, a primeira subetapa é a construção do modelo de rede, que envolve a descrição dos dispositivos presentes na rede, assim como de seus enlaces, conforme pode ser verificado na Figura 38.

A segunda subetapa é a modelagem das funcionalidades, associando as funcionalidades ao modelo da rede. Seguindo a recomendação de, sempre que possível, associar as funcionalidades aos enlaces ao invés de aos dispositivos, associa-se o monitoramento em cada enlace da rede, visto o objetivo de monitorar toda a rede. Assim, todos os enlaces recebem regras e devem ser analisados, conforme pode ser verificado na Figura 39.

Figura 39 – Rede 1: Modelagem das funcionalidades



Fonte: Elaborada pelo autor, 2021.

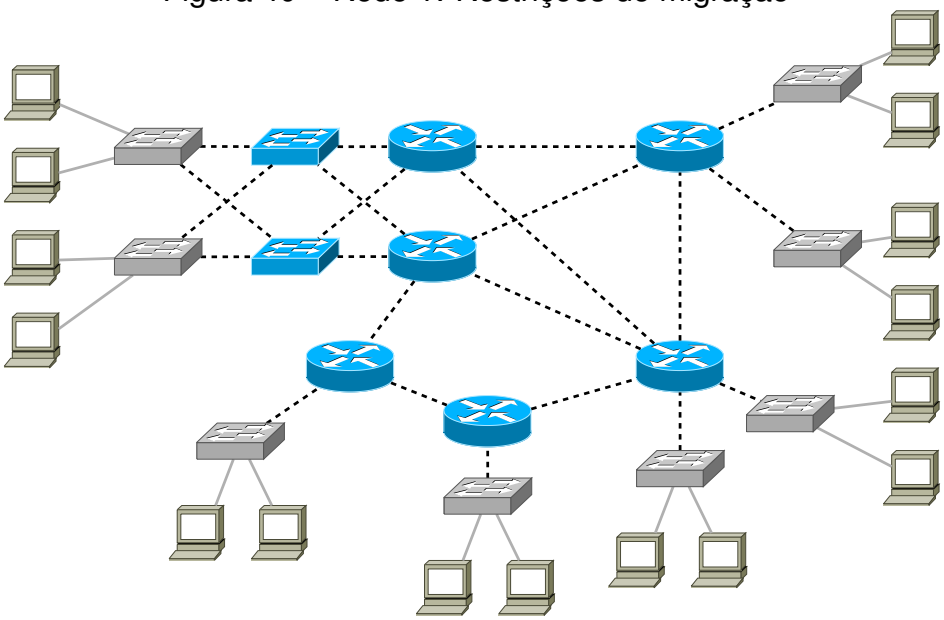
A terceira subetapa é a restrições de migração, onde os dispositivos finais do modelo (computadores) recebem o estado de que não serão migrados. Assim as regras associadas aos enlaces com esses computadores são associadas aos *switches* onde esses dispositivos estão conectados, que por sua vez, recebem o estado `migrar`, visto que eles serão os responsáveis por monitorar os enlaces com os dispositivos finais. A Figura 40 demostra esse processo, tirando o destaque (colocando em cinza) dos enlaces e dispositivos que já foram resolvidos.

A quarta subetapa é a simplificação do modelo, onde é verificado que esses *switches* que serão migrados também podem monitorar seus enlaces com os outros *switches* ou roteadores, e a funcionalidade desses enlaces são reatribuídas ao *switches* que serão migrados. Uma visão geral do modelo, após a conclusão das subetapas da modelagem da rede e funcionalidades, pode ser visualizado na Figura 41, que mantém o destaque (cor azul e linhas pontilhadas) apenas da parte da rede que ainda precisa ser resolvida.

Até então a primeira etapa buscou modelar e simplificar a rede no que concerne as trocas de equipamentos obrigatórias para a aplicação da funcionalidade desejada, o que ocorreu particularmente nas bordas da rede.

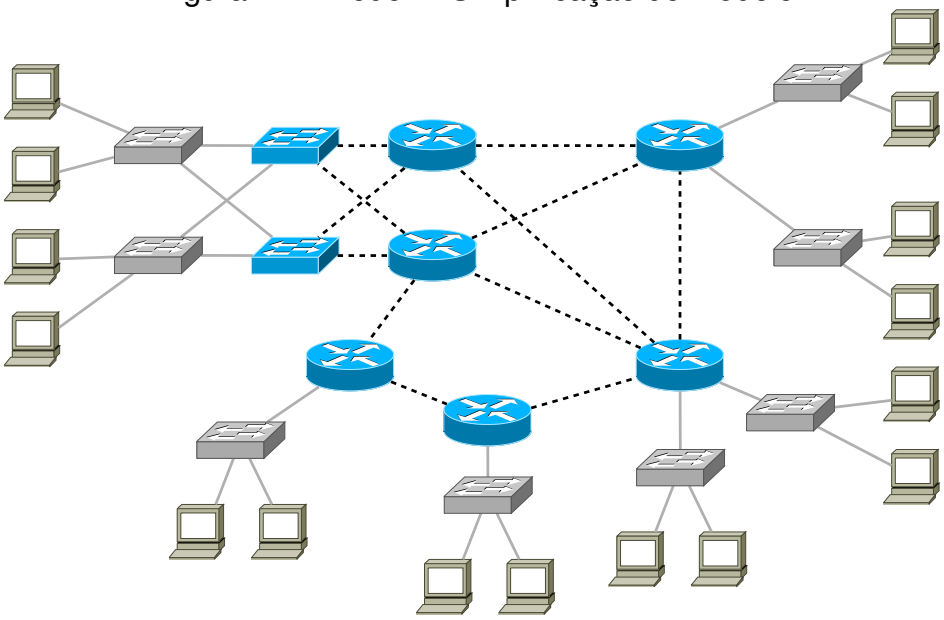
A etapa de busca do conjunto de dispositivos (descrita na subseção 3.2.2) visa

Figura 40 – Rede 1: Restrições de migração



Fonte: Elaborada pelo autor, 2021.

Figura 41 – Rede 1: Simplificação do modelo

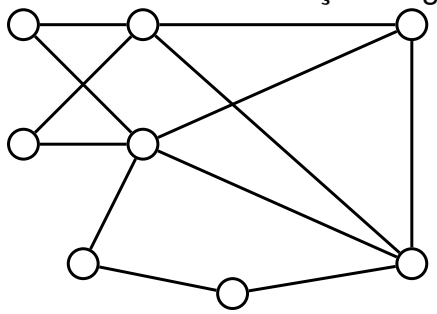


Fonte: Elaborada pelo autor, 2021.

tratar a parte do modelo (rede) que ainda precisa ser resolvido. A primeira de suas subetapas é a extração do grafo, que transforma a parte do modelo ainda não resolvido em um grafo. Essa transformação é direta, ou seja, cada dispositivo da rede que ainda precisa ser resolvido é mapeado como um vértice, enquanto os enlaces são mapeados em arestas. O resultado dessa transformação pode ser verificado na Figura 42.

A segunda subetapa é a simplificação do grafo, onde é feita uma análise para se reduzir o tamanho do grafo. Porém para essa migração específica, não foi possível remover nenhum vértice do grafo atual, devido a redundância presente na rede, uma vez que o mesmo não possui nenhum vértice sem ou com apenas uma aresta. Embora

Figura 42 – Rede 1: Extração do grafo

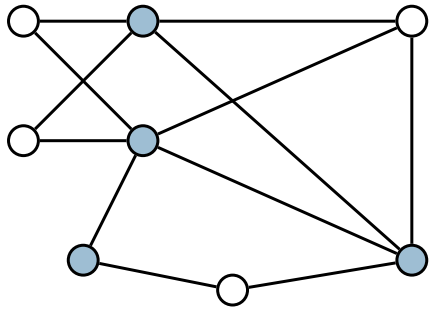


Fonte: Elaborada pelo autor, 2021.

essa subetapa não tenha conseguido remover nenhum vértice, do modelo produzido na subetapa de construção do modelo de rede, já foram removidos 24 vértices referentes aos dispositivos finais e *switches* onde esses dispositivos estão conectados. Considerando apenas dispositivos de rede (*switches* e roteadores), a remoção desses *switches* representou uma redução de 50% no número de vértices do modelo da rede.

Na subetapa cobertura de vértices, devido à quantidade de vértices (oito), optou-se por buscar a solução utilizando força bruta sem paralelismo, sendo que dessa forma ter-se-á a certeza da resposta ótima, e em função do estudo de regressão linear desenvolvido, espera-se uma execução rápida. Para essa quantidade de vértices, espera-se um *overhead* que aumente o tempo de execução quando utilizada alguma técnica de paralelismo, conforme observado nos experimentos executados na subseção 4.1.5.6. Nesse caso, a solução do MVC levou menos de 1 segundo para ser processada em *Single Thread*, e pode ser observada na Figura 43, que apresenta uma resposta com quatro vértices (círculos em azul), que representam os dispositivos que também recebem o estado `migrar`.

Figura 43 – Rede 1: Cobertura de vértices

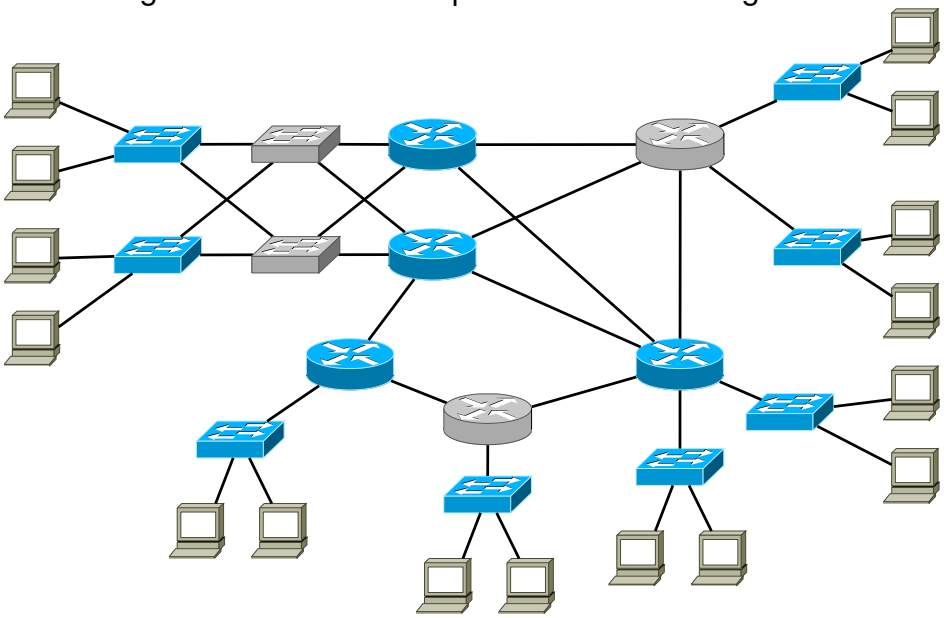


Fonte: Elaborada pelo autor, 2021.

A Figura 44 destaca (cor azul) todos os dispositivos que precisam ser migrados na rede original, para que seja possível monitorá-la por meio de SDN. Assim, nesse exemplo, cuja funcionalidade monitoramento completo é desejada, observa-se a necessidade de migração de 12 dos 16 dispositivos de infraestrutura de rede. Isso representa uma necessidade de migrar 75% dos dispositivos (*switches* e roteadores). Portanto, considerando um caso onde se faria a substituição de todos os equipamentos, ao invés

da migração híbrida, isso representa uma redução de 25% no número de dispositivos a serem substituídos, refletindo também na redução do custo da migração.

Figura 44 – Rede 1: Dispositivos a serem migrados



Fonte: Elaborada pelo autor, 2021.

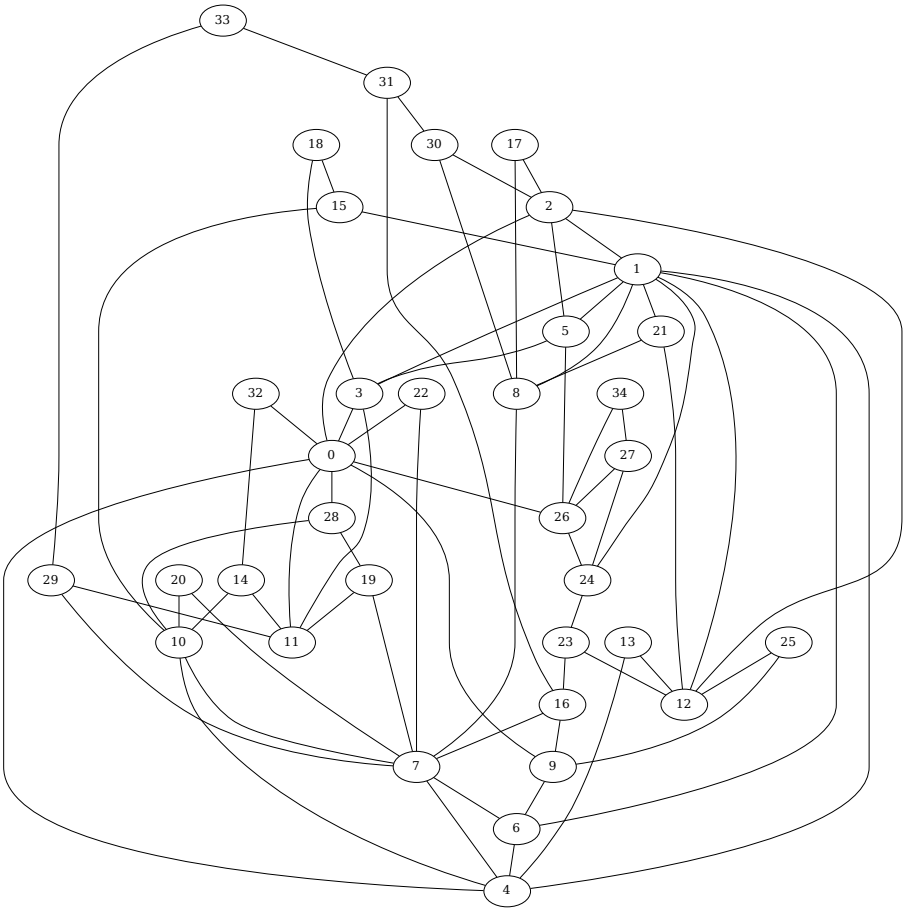
5.2 REDE 2

Para testar a migração de uma rede com mais dispositivos, foi utilizado o *software* BRITE (MEDINA et al., 2001) para gerar uma topologia representativa contendo 35 roteadores. A topologia gerada pode ser conferida na Figura 45. Nesse caso, a funcionalidade desejada será o monitoramento dos enlaces de rede entre esses roteadores através de SDN, de forma semelhante a efetuada no exemplo anterior.

Começando pela etapa modelagem da rede e funcionalidades, a construção do modelo de rede foi efetuado pelo próprio BRITE, que ao gerar a topologia, exporta-a para um arquivo. Na subetapa da modelagem das funcionalidades, todos os enlaces são incluídos, visto que todos eles serão monitorados. Porém, as subetapas restrições de migração e simplificação do modelo não são aplicadas a nenhum dispositivo, visto que não existe nenhum dispositivo que precisa receber especial atenção.

Para a etapa busca do conjunto de dispositivos, na subetapa extração do grafo, como não foi necessário tratar nenhum dispositivo ou enlace de forma especial, o grafo é gerado a partir de toda a topologia. E na subetapa simplificação do grafo, não foi observado nenhum vértice com apenas uma aresta. Desta forma, não foi possível reduzir o grafo a ser processado. Na subetapa cobertura de vértices, devido à quantidade de vértices (35), optou-se pelo método de força bruta paralelizada em GPU, uma vez que haverá ganho com o paralelismo, e como verificado por meio da generalização para o *hardware* onde o problema será computado, disponibilizada pela

Figura 45 – Rede 2: Rede a ser migrada



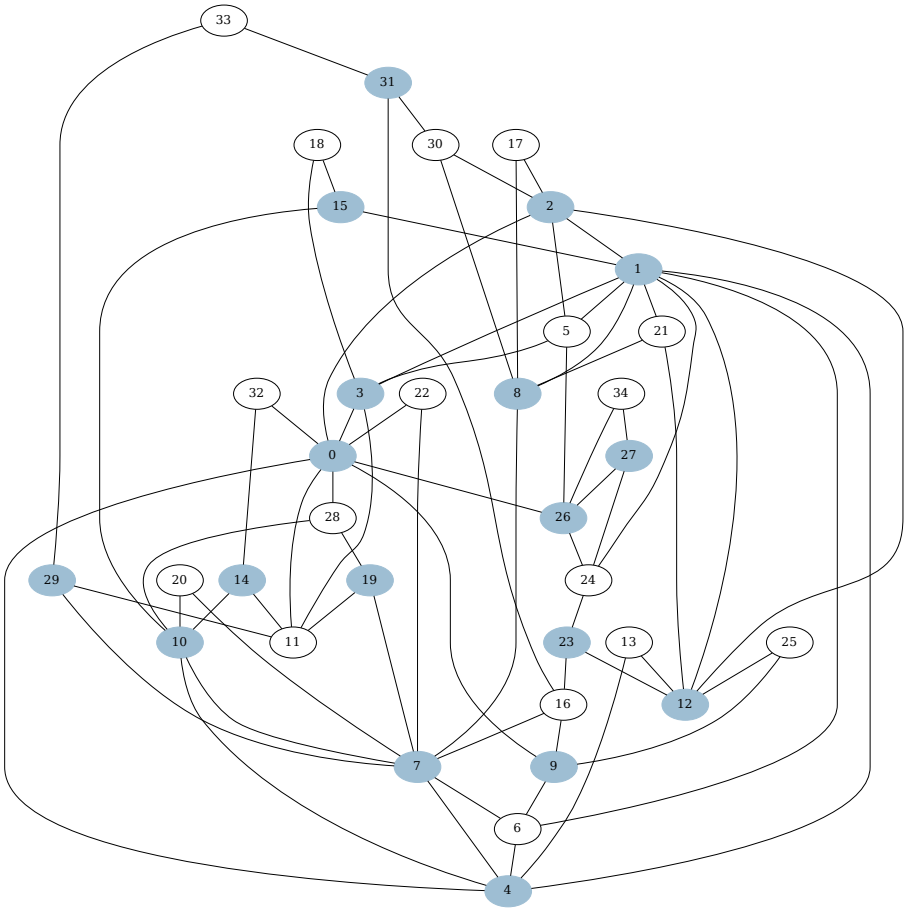
Fonte: Elaborada pelo autor, 2021.

análise de regressão executada na subseção 4.1.6, o tempo de execução do algoritmo é aceitável (até 30 segundos). Sua execução levou 5,59 segundos, apresentando uma solução com 18 vértices, conforme pode ser observado na Figura 46. Desta forma, para essa migração, é necessário trocar 18 dos 35 dispositivos, o que representa aproximadamente metade dos dispositivos, o que também é refletido no custo da migração.

5.3 REDE 3

Considerando um cenário onde existem diferenças entre os dispositivos de rede, como custo de aquisição de um novo dispositivo para substituisão do atual, ou diferentes esforços para migrá-los, uma otimização do tipo iii é a adequada. Para testar esse tipo de migração, foi utilizado o mesmo cenário do exemplo anterior (topologia e funcionalidade). Para modelar as restrições ou dificuldades específicas em relação aos equipamentos, foram associados pesos de valores randômicos entre 1 e 10 para cada dispositivo, seguindo as demais etapas, porém substituindo o problema MVC pelo MWVC.

Figura 46 – Rede 2: MVC



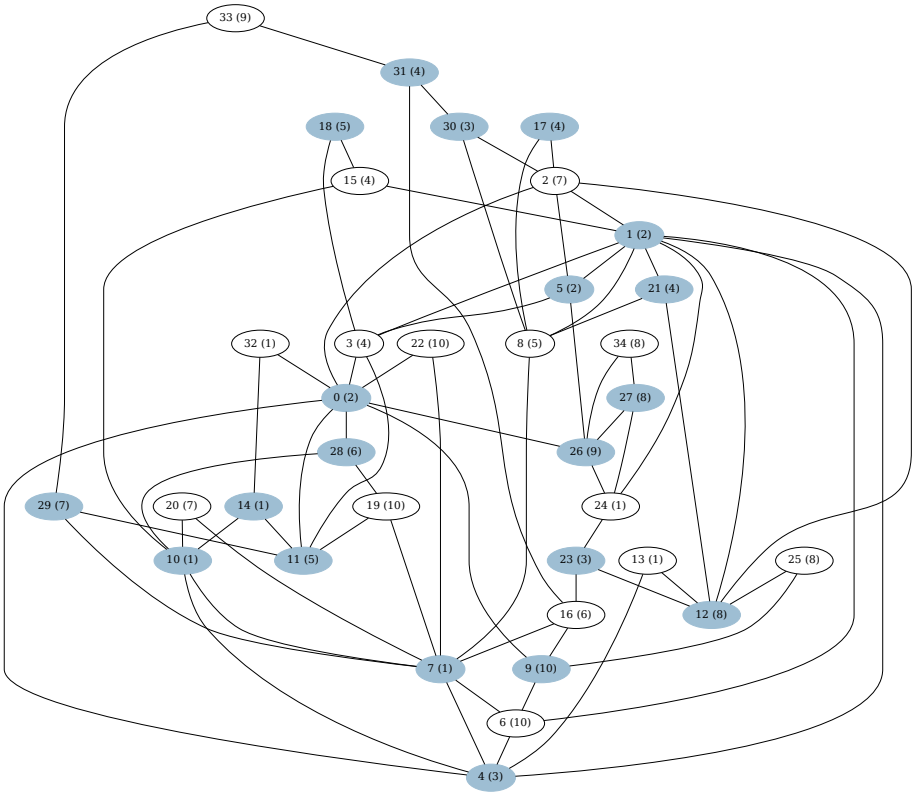
Fonte: Elaborada pelo autor, 2021.

A função de força bruta com paralelismo em GPU do MWVC executou em 5,66 segundos, um pouco mais demorada que a versão MVC. O resultado foi um conjunto com 20 vértices (57,14% do total de vértices), e soma de pesos 88 (49,16% do total dos pesos). Comparando com o resultado do MVC, esse resultado teve 2 vértices a mais, porém com soma de 1 peso a menos (sendo que a soma dos pesos do resultado do MVC é de 89). O resultado dessa migração pode ser visto na Figura 47, onde além do número do vértice, também é exibido entre parênteses o peso que lhe foi atribuído.

5.4 REDE 4

Considerando redes maiores, foi gerada outra topologia com o *software* BRITE, desta vez contendo 250 roteadores. Utilizando o modelo de regressão para força bruta, o tempo necessário para executar a solução com paralelismo em GPU seria de aproximadamente $2^{229,1886}$ segundos, em torno de $3,12 \cdot 10^{61}$ anos, inviabilizando a utilização de força bruta, e portanto, necessitando a aplicação de alguma heurística ou meta-heurística. Dessa forma, foi identificada uma resposta com 135 vértices pelo algoritmo guloso (54% do total de vértices). Tal resposta, aplicando o 3-opt,

Figura 47 – Rede 3: MWVC



Fonte: Elaborada pelo autor, 2021.

foi otimizada em quatro vértices, totalizando 131 em 0,02 segundos (52,4% do total de vértices), e não conseguiu encontrar mais nenhuma otimização nos 30 minutos de execução seguintes. De qualquer forma, embora nenhuma outra otimização foi encontrada, não se tem certeza de essa ser a resposta ótima. Os dados referentes aos vértices, disposição das arestas e resposta encontrada podem ser conferidos no Apêndice A.

5.5 CONSIDERAÇÕES

Foi possível migrar todas as topologias testadas sem a necessidade de migrar todos os dispositivos. Portanto, consegue-se assim reduzir o custo ou esforço da migração, e utilizar as funcionalidades planejadas com a migração por meio de SDN. Dessa forma, considerando-se que originalmente seriam migrados todos os dispositivos, no primeiro caso houve uma redução de 25% no número de dispositivos a serem migrados, enquanto nos demais casos foi possível fazer a migração com uma redução de aproximadamente metade dos dispositivos, mesmo envolvendo todos os dispositivos e enlaces de rede com redundância.

6 CONCLUSÃO

Este trabalho propôs um modelo de processo que aponta os dispositivos que devem ser migrados a fim de permitir a utilização de funcionalidades por meio de SDN em uma rede tradicional de computadores. Essa migração leva em conta características da rede a ser migrada, como também das funcionalidades desejadas, de forma a disponibilizar um conjunto otimizado de dispositivos de encaminhamento que necessitam ser substituídos por outros com suporte a SDN, ou serem reconfigurados caso já possuam o suporte.

Esse modelo de processo trabalha tentando reduzir o tamanho do problema a ser tratado pela cobertura mínima de vértices (MVC), ou do peso mínimo da cobertura de vértices (MWVC), necessários para a otimização da migração. Essa redução no tamanho dos problemas permite tratá-los com mais facilidade, e com o paralelismo, possibilita a utilização de força bruta para resolver a migração para uma boa quantidade de casos, garantindo-se a menor quantidade possível de equipamentos para migração, ou tendo o menor custo com a compra de novos equipamentos.

Para se resolver o problema MVC e MWVC, que são classificados como NP-difícil, foram discutidas as limitações da solução baseada em força bruta, demonstrando como identificar até que tamanho do problema ela pode oferecer uma resposta em um período de tempo estipulado. Também foram aplicadas técnicas de paralelismo, discutindo como encontrar os melhores parâmetros, demonstrando em quais cenários existe ganho com paralelismo (permite tratar problemas maiores no mesmo período de tempo), e onde o paralelismo se torna uma solução inadequada, levando mais tempo que a opção não paralelizada.

Para lidar com as migrações que não forem tratáveis com força bruta, foi apresentada a heurística 3-opt. Em experimentos realizados, essa heurística conseguiu encontrar a resposta ótima de migrações, alcançadas anteriormente com a força bruta, em menor tempo. Porém, dado seu funcionamento através de buscas locais, na generalidade dos casos não é possível ter certeza se a resposta encontrada por essa heurística é a resposta ótima ou não. Entretanto ela conseguiu boas otimizações em todos os casos em que foi testada, podendo ser utilizada para tratar problemas maiores sem um estudo prévio de parâmetros como seria necessário em outras heurísticas.

Ao aplicar o processo descrito no trabalho em redes diversas, foram observadas reduções no tamanho do problema quando essas redes envolviam dispositivos com características específicas, como roteadores de borda e outros dispositivos que não poderiam ser migrados para SDN, e uma grande redução na quantidade de dispositivos a serem migrados, conforme cada caso. A redução no tamanho do problema se refletiu em permitir tratar o restante da migração com uma solução baseada em força bruta, o que confere a certeza de otimização da migração. Como consequência, a solução

do MVC ou MWVC com tamanho reduzido traduz-se em redução na quantidade de dispositivos a serem migrados. Sendo assim, isso facilita posteriormente o processo físico de migração, refletindo na necessidade de compra de menos equipamentos, ou seja, na diminuição de custos.

Embora não seja possível generalizar o resultado dos exemplos de migração testados, uma vez que a necessidade de migração é dependente da topologia da rede e da funcionalidade desejada, pode-se dizer que desde que bem planejada a funcionalidade necessária, o processo de migração proposto otimiza o custo final da migração. Com o processo sendo válido para qualquer rede, é possível utilizar as funcionalidades desejadas após a migração que tem por resultado uma rede híbrida, a não ser em casos específicos, onde o processo identifica que alguma funcionalidade não pode ser atribuída a algum dispositivo.

6.1 OPORTUNIDADES DE MELHORIAS

Com a execução deste trabalho, abrem-se oportunidades de pesquisa para outros trabalhos. Como por exemplo, a criação de uma ferramenta ou serviço para o usuário final, permitindo a utilização do conhecimento presente neste trabalho de forma facilitada. Isto envolveria a discussão sobre uma interface para usuário informar a rede, ou obtê-la de forma automatizada, informar e modelar as funcionalidades desejadas, e como receberá o retorno do processamento. Também é possível utilizar automatização para criar a infraestrutura necessária para processar a resposta em algum provedor de nuvem, conforme a necessidade da rede, ou a pedido do usuário.

Além disso, estudos envolvendo outras topologias e casos de migração reais, enriqueceriam a validação do processo. Ainda, o processo em si pode ser aditivado de outras heurísticas e meta-heurísticas para auxiliar na otimização da solução para redes maiores e complexas. Adicionalmente, pode-se inserir no processo outras restrições à migração, e formas de modelar as funcionalidades por meio de outros problemas conhecidos, permitindo modelar outros tipos de funcionalidade de rede na migração.

REFERÊNCIAS

ANDRIOLI, Leandro; RIGHI, Rodrigo Rosa da; AUBIN, Mateus Rauback. Analisando métodos e oportunidades em redes definidas por software (SDN) para otimizações de tráfego de dados. **Revista Brasileira de Computação Aplicada**, v. 9, n. 4, p. 2–14, dez 2017. Disponível em: ⟨http://seer.upf.br/index.php/rbca/article/view/6948⟩.

BARBETTA, Pedro Alberto; REIS, Marcelo Menezes; BORNIA, Antonio Cezar. **Estatística: Para Cursos de Engenharia e Informática**. 3. ed. São Paulo, SP, Brasil: Atlas, 2010. 410 p. ISBN 978-85-224-5994-0.

BARDALAI, Priyanka; MEDHI, Nabajyoti; CHAKRABORTY, Swarnendu Kumar. DoubleTrApp: A Weak Vertex Cover based DDoS Detection and Mitigation scheme using SDN approach. In: **2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)**. IEEE, 2019. p. 1–6. ISBN 978-1-7281-3715-5. ISSN 2153-1684. Disponível em: ⟨https://ieeexplore.ieee.org/document/9118136⟩.

BLAZINSKAS, Andrius; MISEVICIUS, Alfonsas. Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. In: **Proceedings of the 17th International Conference on Information and Software Technologies**. [s.n.], 2011. Disponível em: ⟨https://www.semanticscholar.org/paper/Combining-2-opt-%2C-3-opt-and-4-opt-with-K-swap-kick-Blazinskas/ab7cc83bb513a91b06f6c8bc3b9da7f60cbbaee5⟩.

BONFOH, El-Fadel; MEDJIAH, Samir; CHASSOT, Christophe. A Parsimonious Monitoring Approach for Link Bandwidth Estimation within SDN-based Networks. In: **2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)**. IEEE, 2018. p. 512–516. ISBN 978-1-5386-4633-5. Disponível em: ⟨https://ieeexplore.ieee.org/document/8459972⟩.

CAI, Shaowei; SU, Kaile; CHEN, Qingliang. EWLS: A New Local Search for Minimum Vertex Cover. In: **Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence**. Palo Alto, CA, USA: AAAI Press, 2010. p. 45–50. ISBN 978-1-57735-463-5. Disponível em: ⟨https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1679⟩.

CERVO, Amado Luiz; BERVIAN, Pedro Alcino; SILVA, Roberto da. **Metodologia científica**. 6. ed. São Paulo, SP, Brasil: Prentice Hall, 2007. 159 p. ISBN 8576050471.

CHANG, Michael Alan et al. Supercharge Me: Boost Router Convergence with SDN. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 45, n. 4, p. 341–342, ago. 2015. ISSN 0146-4833. Disponível em: ⟨https://dl.acm.org/doi/abs/10.1145/2829988.2790007⟩.

CSIKOR, Levente et al. Transition to SDN is HARMLESS: Hybrid Architecture for Migrating Legacy Ethernet Switches to SDN. **IEEE/ACM Trans. Netw.**, IEEE, v. 28, n. 1, p. 275–288, fev. 2020. ISSN 1063-6692. Disponível em: ⟨https://ieeexplore.ieee.org/document/8950290⟩.

EDEN, Amnon H. Three Paradigms of Computer Science. **Minds and Machines**, Springer, Berlin, Germany, v. 17, n. 2, p. 135–167, jul. 2007. ISSN 1572-8641. Disponível em: ⟨https://link.springer.com/article/10.1007/s11023-007-9060-8⟩.

FLYNN, Michael J. Some Computer Organizations and Their Effectiveness. **IEEE Transactions on Computers**, IEEE, Berlin, Germany, C-21, n. 9, p. 948–960, set. 1972. ISSN 1557-9956. Disponível em: ⟨https://ieeexplore.ieee.org/document/5009071⟩.

GAREY, Michael R.; JOHNSON, David S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman and Company, 1979. ISBN 0-7167-1044-7.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo, SP, Brasil: Atlas, 2002. 175 p. ISBN 85-224-3169-8.

GILESH, M. P.; KUMAR, S. D. Madhu; JACOB, Lillykutty. HyViDE: A Framework for Virtual Data Center Network Embedding. In: **Proceedings of the Symposium on Applied Computing**. New York, NY, USA: ACM, 2017. (SAC '17), p. 378–383. ISBN 9781450344869. Disponível em: ⟨https://dl.acm.org/doi/10.1145/3019612.3019629⟩.

GOLDBARG, Marco; GOLDBARG, Elizabeth. **Grafos: conceitos, algoritmos e aplicações**. Rio de Janeiro, RJ, Brasil: Elsevier, 2012. 622 p. ISBN 978-85-352-5716-8.

KLOSOWSKI, Eduardo Augusto; FIORESE, Adriano. Cobertura mínima de vértices na migração para SDN. In: **Anais da XIX Escola Regional de Alto Desempenho da Região Sul**. Porto Alegre, RS, Brasil: SBC, 2019. p. 139–142. ISSN 2595-4164. Disponível em: ⟨https://sol.sbc.org.br/index.php/eradrs/article/view/7073⟩.

____. FAEController: A Relative Efficiency Evaluation Framework of SDN Controllers. In: **Proceedings of the XV Brazilian Symposium on Information Systems**. ACM, 2019. p. 75:1–75:8. ISBN 978-1-4503-7237-4. Disponível em: ⟨http://doi.acm.org/10.1145/3330204.3330285⟩.

____. FREEController: A Framework for Relative Efficiency Evaluation of Software-Defined Networking Controllers. In: **Proceedings of the 21st International Conference on Enterprise Information Systems**. SciTePress, 2019. v. 1, p. 349–360. ISBN 978-989-758-372-8. Disponível em: ⟨http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0007761503490360⟩.

KREUTZ, Diego et al. Software-Defined Networking: A Comprehensive Survey. **Proceedings of the IEEE**, IEEE, v. 103, n. 1, p. 14–76, jan 2015. ISSN 1558-2256. Disponível em: ⟨https://ieeexplore.ieee.org/document/6994333⟩.

MARCONI, Marina Andrade de; LAKATOS, Eva Maria. **Fundamentos de Metodologia Científica**. 5. ed. São Paulo, SP, Brasil: Atlas, 2003. 311 p. ISBN 85-224-3397-6.

MCKEOWN, Nick et al. OpenFlow: Enabling Innovation in Campus Networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, abr. 2008. ISSN 0146-4833. Disponível em: ⟨https://dl.acm.org/doi/10.1145/1355734.1355746⟩.

MEDINA, Alberto et al. BRITE: an approach to universal topology generation. In: **MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems**. IEEE, 2001. p. 346–353. ISBN 0-7695-1315-8. ISSN 1526-7639. Disponível em: ⟨https://ieeexplore.ieee.org/document/948886⟩.

RICHTER, Silvia; HELMERT, Malte; GRETTON, Charles. A Stochastic Local Search Approach to Vertex Cover. In: HERTZBERG, Joachim; BEETZ, Michael; ENGLERT, Roman (Ed.). **Advances in Artificial Intelligence**. Berlin, Germany: Springer, 2007. (KI 2007), p. 412–426. ISBN 978-3-540-74565-5. Disponível em: ⟨http://hdl.handle.net/10072/19030⟩.

SELVARAJ, P.; NAGARAJAN, V. Migration from Conventional Networking to Software Defined Networking. In: **2017 International Conference on IoT and Application (ICIOT)**. IEEE, 2017. (ICIOT 2017), p. 1–7. ISBN 978-1-5386-1698-7. Disponível em: ⟨https://ieeexplore.ieee.org/document/8073632⟩.

SINGH, Arjun et al. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. **Commun. ACM**, ACM, New York, NY, USA, v. 59, n. 9, p. 88–97, ago. 2016. ISSN 0001-0782. Disponível em: ⟨https://dl.acm.org/doi/abs/10.1145/2975159⟩.

SU, Zhiyang et al. FlowCover: Low-cost flow monitoring scheme in software defined networks. In: **2014 IEEE Global Communications Conference**. IEEE, 2014. p. 1956–1961. ISBN 978-1-4799-3512-3. Disponível em: ⟨https://ieeexplore.ieee.org/document/7037094⟩.

SZWARCFITER, Jayme Luiz. **Teoria Computacional de Grafos: Os Algoritmos**. 1. ed. Rio de Janeiro, RJ, Brasil: Elsevier, 2018. 352 p. ISBN 978-85-352-8884-1.

TANENBAUM, Andrew S. **Redes de computadores**. Rio de Janeiro, RJ, Brasil: Elsevier, 2003. 945 p. ISBN 85-352-1185-3.

WAZLAWICK, Raul Sidnei. **Metodologia da Pesquisa para Ciência da Computação**. Rio de Janeiro, RJ, Brasil: Elsevier, 2009. 159 p. ISBN 978-85-352-3522-7.

YAN, Jiaqi; LIU, Xin; JIN, Dong. Simulation of a Software-Defined Network as One Big Switch. In: **Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation**. New York, NY, USA: ACM, 2017. (SIGSIM-PADS '17), p. 149–159. ISBN 9781450344890. Disponível em: ⟨https://dl.acm.org/doi/10.1145/3064911.3064918⟩.

Apêndice A – DADOS UTILIZADOS

Os dados utilizados nesse trabalho estão no arquivo dissertacao-dados.tar.gz anexo a esse arquivo `pdf`. O formato desse arquivo é um `tar` compactado com gzip, com a seguinte estrutura de arquivos:

- `topologias-brite/`n`.brite`
  Topologias com n vértices geradas pelo *software* BRITE, utilizadas na comparação com as demais topologias.
- `topologias-brite/exemplo-rede2.brite`
  Topologia gerada pelo *software* BRITE utilizada na seção 5.2.
- `topologias-brite/exemplo-rede4.brite`
  Topologia gerada pelo *software* BRITE utilizada na seção 5.4.
- `topologias-brite/exemplo-rede4.mvc`
  Resposta ao MVC da migração proposta na seção 5.4.
- `dados/bf-st-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta em *Single Thread* para o MVC.
- `dados/bf-st-mwvc.csv.gz`
  Resultados das execuções da implementação de força bruta em *Single Thread* para o MWVC.
- `dados/bf-omp-escalonadores-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta com OpenMP para o MVC variando os parâmetros de escalonador.
- `dados/bf-omp-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta com OpenMP para o MVC.
- `dados/bf-omp-mwvc.csv.gz`
  Resultados das execuções da implementação de força bruta com OpenMP para o MWVC.
- `dados/bf-ocl-escalonadores-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta com OpenCL para o MVC variando os parâmetros de `kernel` e tamanho do vetor.

- `dados/bf-ocl-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta com OpenCL para o MVC.

- `dados/bf-ocl-mwvc.csv.gz`
  Resultados das execuções da implementação de força bruta com OpenCL para o MWVC.

- `dados/bf-hibrido-escalonadores-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta com paralelismo híbrido viariando os tamanhos do vetor.

- `dados/regressao-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta utilizados na regressão do MVC.

- `dados/regressao-ext-mvc.csv.gz`
  Resultados das execuções da implementação de força bruta para validar a regressão do MVC.

- `dados/regressao-mwvc.csv.gz`
  Resultados das execuções da implementação de força bruta utilizados na regressão do MWVC.

- `dados/regressao-ext-mwvc.csv.gz`
  Resultados das execuções da implementação de força bruta para validar a regressão do MWVC.

- `dados/heuristica-otimo-mvc.csv.gz`
  Resultados das execuções da implementação em força bruta para analisar os resultados das heurísticas.

- `dados/heuristica-guloso-mvc.csv.gz`
  Resultados das execuções do algoritmo guloso para o MVC.

- `dados/heuristica-3opt-mvc.csv.gz`
  Resultados das execuções do 3-opt para o MVC.

- `analises/bf-topologias.ipynb`
  Análises efetuadas na subseção 4.1.5.1.

- `analises/bf-st.ipynb`
  Análises efetuadas na subseção 4.1.5.2.

- `analises/bf-omp.ipynb`
  Análises efetuadas na subseção 4.1.5.3.

- `analises/bf-ocl.ipynb`
  Análises efetuadas na subseção 4.1.5.4.

- `analises/bf-hibrido.ipynb`
  Análises efetuadas na subseção 4.1.5.5.

- `analises/bf-speedup.ipynb`
  Análises efetuadas na subseção 4.1.5.6.

- `analises/regressao.ipynb`
  Análises efetuadas na subseção 4.1.6.

- `analises/heuristica.ipynb`
  Análises efetuadas na seção 4.2 e seção 4.3.

Os arquivos `*.brite` são formatados conforme descrito por Medina et al. (2001), contendo todos os parâmetros utilizados na geração das topologias. Os arquivos `*.mvc` contem os vértices encontrados na resolução do problema MVC. Os arquivos `*.csv.gz` são `csv` compactados em gzip. Os arquivos `*.ipynb` são arquivos do Jupyter Notebook com o código na linguagem R, contendo os resultados das análises estatísticas.

Os dados presentes nas colunas dos arquivos `csv` são:

- `topologia`
  O tipo de topologia do grafo.

- `num_vertices`
  A quantidade de vértices do grafo.

- `tempo`
  Tempo de execução da função.

- `resposta`
  Vértices encontrados na resposta.

- `tamanho_resposta`
  Quantidade de vértices presentes na resposta.

- `peso_resposta`
  Soma do peso dos vértices presentes na resposta.

- `escalonador`
  Escalonador do OpenMP utilizado.

- `chunk_size`
  Tamanho de `chunk size` utilizado pelo escalonador.

- `kernel`
  Função do OpenCL (`kernel`) utilizada.

- `tamanho_vetor`
  Tamanho do vetor utilizado no OpenCL.

- `tamanho_vetor_local`
  Tamanho de vetor local utilizado no OpenCL.

- `repeticoes`
  Quantidade de execuções necessárias para se atingir a resposta ótima.