MINISTÉRIO DA EDUCAÇÃO

SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLOGIA

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS

CÂMPUS ANÁPOLIS

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

Stefany Fernandes

Avaliação de um algoritmo exato para um problema biobjetivo de roteamento de fluxos usando um emulador de redes

Anápolis - GO

2019

Stefany Fernandes

Avaliação de um algoritmo exato para um problema biobjetivo de roteamento de fluxos usando um emulador de redes

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Goiás como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

Instituto Federal de Educação, Ciência e Tecnologia de Goiás - Campus Anápolis

Orientador: Profa. Dra. Kátia Cilene Costa Fernandes

Anápolis - GO

2019

INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
SISTEMA INTEGRADO DE BIBLIOTECAS

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

[ ] Tese
[ ] Dissertação
[ ] Monografia – Especialização
[X] TCC - Graduação
[ ] Produto Técnico e Educacional - Tipo: _____
[ ] Artigo Científico
[ ] Capítulo de Livro
[ ] Livro
[ ] Trabalho Apresentado em Evento

Nome Completo do Autor: Stefany Fernandes
Matrícula: 20161060140256
Título do Trabalho: Avaliação de um algoritmo exato para um problema biobjetivo de roteamento de fluxos usando um emulador de redes.

Restrições de Acesso ao Documento

Documento confidencial: [X] Não [ ] Sim, justifique: _____

Informe a data que poderá ser disponibilizado no ReDi/IFG: 23/12/2019
O documento está sujeito a registro de patente? [ ] Sim [X] Não
O documento pode vir a ser publicado como livro? [ ] Sim [X] Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Anápolis, 03/01/2020
Local Data

Stefany Fernandes
Assinatura do Autor e/ou Detentor dos Direitos Autorais

FICHA CATALOGRÁFICA

F363a	FERNANDES, Stefany Avaliação de um algoritmo exato para um problema biobjetivo de roteamento de fluxos usando um emulador de redes. / Stefany Fernandes – – Anápolis: IFG, 2019. 40 p. : il. Orientadora: Prof. Dra. Kátia Cilene Costa Fernandes Trabalho de Conclusão do Curso de Ciência da Computação, Instituto Federal de Goiás, Campus Anápolis, 2019. 1. Emulador de redes. 2. Problema biobjetivo. 3. Mininet. 4.Computação. 5. Redes de computadores I. FERNANDES, Kátia Cilene Costa orien.. II. Título.
CDD 004.6	

Código 017.2019

Ficha catalográfica elaborada pelo Bibliotecário Matheus Rocha Piacenti CRB1/2992

Biblioteca Clarice Lispector, Campus Anápolis

Instituto Federal de Educação, Ciência e Tecnologia de Goiás

INSTITUTO FEDERAL
Goiás
Câmpus Anápolis

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
CÂMPUS ANÁPOLIS
COORDENAÇÃO DE CURSOS

ATA DE DEFESA PÚBLICA DO TRABALHO DE CONCLUSÃO DE CURSO II

Aos 17 dias do mês de dezembro de 2019, às 14:00 horas, em sessão pública na sala Multimeios 3 do Câmpus Anápolis / IFG, na presença da Banca Examinadora presidida pelo(a) Professor(a) Orientador(a) Kátia Cilene da Costa Fernandes
e composta pelos examinadores:
1. Hugo Vinicius Leão e Silva
2. Alexandre Bellezi José
o(a) Discente Stefany Fernandes apresentou o Trabalho de Conclusão de Curso intitulado:
"Avaliação de um algoritmo exato para um problema biobjetivo de roteamento de fluxos usando um emulador de redes e métricas de redes reais"
como requisito curricular indispensável para a integralização do Curso de Graduação de Bacharelado em Ciência da Computação.
Após reunião em sessão reservada, a Banca Examinadora deliberou e decidiu pela aprovação do referido trabalho, divulgando o resultado formalmente ao discente e demais presentes e eu, na qualidade de Presidente da Banca, lavrei a presente ata que será assinada por mim, pelos demais examinadores e pelo discente.

Presidente da Banca Examinadora

Examinador 01

Examinador 02

Discente

LISTA DE TABELAS

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

FTP	File Transfer Protocol
HTTP	HyperTextTransfer Protocol
OF	OpenFlow
SDN	Software-Defined Network
TCP/IP	Transmission Control Protocol/Internet Protocol suite

RESUMO

Um problema de programação inteira biobjetivo para roteamento de fluxos em rede consiste em uma função para minimizar o total de saltos de todos os fluxos e outra para minimizar o gargalo da rede. Esse problema pode ser aplicado, por exemplo, no roteamento de fluxos de dados em redes em malha sem fio ou redes de sensores. Existem, na literatura, trabalhos que propõem soluções exatas, porém avaliando seus algoritmos sob métricas tradicionais de grafos, como a cardinalidade do conjunto de soluções não dominadas, maior gargalo e ampliação do total de saltos. Também encontra-se trabalhos que realizam avaliações sob métricas de redes reais, porém utilizam, como solução, heurísticas que não garantem soluções exatas. Neste trabalho é investigado o uso de um algoritmo exato aplicado a cenários reais de redes. As avaliações do algoritmo exato são feitas através de um emulador de redes chamado Mininet, o qual permite capturar métricas clássicas como vazão e atraso. Além do roteamento, o Mininet permite imitar aplicações reais baseadas em protocolos de transporte da Internet (e.g., TCP ou UDP).

Palavras-chave: *Mininet*, *Emuladorderedes*, *ProblemasBiobjetivo*.

SUMÁRIO

INTRODUÇÃO

Um problema clássico de programação inteira biobjetivo (problemas com duas funções objetivos) para roteamento de fluxos em rede é minimizar o total de saltos de todos os fluxos em uma rede e, ainda, minimizar o gargalo presente nela.

Na literatura, vê-se vários trabalhos propondo algoritmos exatos para esse tipo de problema (PINTO; FERNANDES, 2016) e (PINTO et al., 2019), e podendo ser analisado várias métricas, por exemplo, a cardinalidade do conjunto de soluções eficientes, maior gargalo da rede e aumento do número de saltos em relação ao mínimo. É possível observar, ainda trabalhos que apresentam sob métricas de redes reais, algoritmos que utilizam heurísticas que garantem uma solução aproximada para o problema (MELLO, 2014).

Um modelo de programação inteira biobjetivo foi apresentado em (PINTO; FERNANDES, 2016), com aplicações em redes de computadores, juntamente com um método exato, baseado na técnica *e-constraint*. Esse trabalho gera um conjunto mínimo completo de soluções Pareto-ótimas (eficientes). Para avaliar o método, métricas como cardinalidade do conjunto mínimo completo de soluções não dominadas (X^*), o número de iterações e tempo de execução foram usadas. Já para a avaliação das soluções geradas pelo algoritmo, as métricas foram: o menor e o maior gargalo encontrado em X^* e a amplitude total dos comprimentos dos caminhos (medida da diferença entre as somas dos comprimentos de caminhos de todos os fluxos, do primeiro e o ultimo do conjunto de soluções Pareto-ótimas).

Este trabalho tem como objetivo avaliar cenários reais em redes de múltiplos saltos, a partir de um algoritmo exato para resolver problemas de roteamento de fluxos que minimiza, simultaneamente, o comprimento dos caminhos de todos os fluxos e o gargalo da rede, que, por sua vez, serão criados através de um emulador de redes (Mininet), que permite capturar métricas como vazão e atraso. Além do roteamento, a ferramenta permite representar aplicações reais baseadas em protocolos de Internet (como o TCP e UDP) (TEAM, 2018). Dessa forma, será possível avaliar as soluções geradas pelo algoritmo exato de uma maneira mais ampla. Portanto, nesse trabalho será usado o emulador Mininet (TEAM, 2018).

Os objetivos específicos desse trabalho são: estudar problemas biobjetivos em grafos, estudar e compreender o algoritmo exato que será utilizado para a solução de um problema biobjetivo de roteamento de fluxos em rede e estudar a ferramenta Mininet com intuito de identificar como as rotas geradas pelo algoritmo exato podem ser implantadas, e como as métricas de interesse podem ser coletadas.

Esse documento está organizado da seguinte forma: o Capítulo 1 aborda toda a

fundamentação teórica desse trabalho, bem como definições importantes sobre redes, protocolos, problemas de otimização, entre outros. O Capítulo 2 contém a metodologia usada nesse trabalho (o que foi feito e como foi feito). O Capítulo 3 apresenta o funcionamento da ferramenta Mininet. O Capítulo 4 aborda os resultados obtidos a partir de todo o desenvolvimento desse trabalho, e o Capítulo 5 aborda as considerações finais, bem como possíveis trabalhos futuros.

1 REFERENCIAL TEÓRICO

O desenvolvimento significativo dos equipamentos e das tecnologias de comunicação acarreta um cenário de intensos gargalos nas redes de computadores. A expressão “rede de computadores” diz respeito a um conjunto de dois ou mais computadores autônomos interconectados. Um computador está conectado a outro quando ambos podem trocar informações, sendo que a conexão estabelecida entre eles não precisa ser necessariamente física (por um fio de cobre ou fibra óptica), podendo se utilizar, por exemplo, conexões *wireless*, microondas, entre outras (TANENBAUM, 2011).

O aumento de dispositivos interconectados, bem como o fluxo de dados entre eles, gera o que se chama de gargalo. Define-se como gargalo da rede a(s) aresta(s) (ou enlace) que recebe a maior quantidade de fluxos (MELLO, 2014). Um exemplo simples de uma rede de computadores pode ser observado na Figura 1.

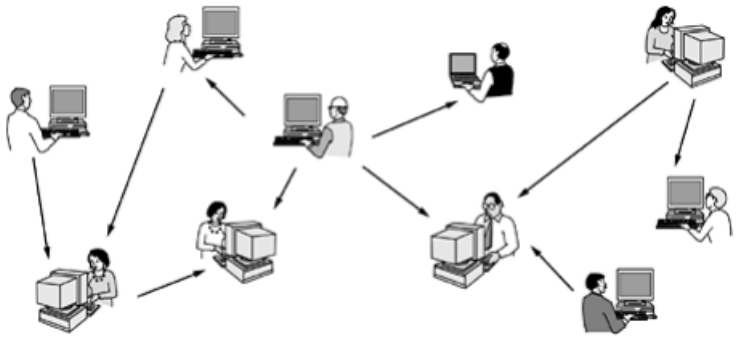

Figura 1 – Rede de computadores simples.

Fonte: Tanenbaum (2011).

O funcionamento das redes de computadores atuais depende do que se chama pilha de protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol suite*). Isso é caracterizado como pilha por possuir vários outros protocolos que trabalham em conjunto. Por exemplo, o HTTP (*HyperText Transfer Protocol*) e o IP (*Internet Protocol*). O primeiro é responsável por definir como os servidores farão a transferência de páginas (como páginas *web*) ou documentos para o cliente, enquanto o segundo oferece seviços para o protocolo HTTP. Em (COMER, 2016) a pilha de protocolos TCP/IP é definida tendo como característica a entrega, o ordenamento e a não duplicidade dos dados enviados de um nó para outro.

Nesse sentido, faz-se necessário um estudo de representação de rede e o desenvolvimento de técnicas de otimização para roteamento de fluxos a fim de reduzir o gargalo existente em uma determinada rede.

1.1 Grafos

Para representar geometricamente uma rede, na literatura, utilizam-se grafos.

Definição 1.1 *Um grafo pode ser descrito por $G(V, E)$, sendo V um conjunto arbitrário cujos elementos são chamados de vértices, e E, um subconjunto que define as arestas (ligação entre os vértices) desse grafo.*

Um vértice pode possuir uma aresta que tem início e fim em si. Essa característica é chamada de *laço* (ou *loop*) (Figura 2 (b), vértice c). São denominadas arestas múltiplas aquelas que possuem os mesmos vértices nos extremos (veja Figura 2 (b), arestas (a, d)). Os grafos que não possuem laços e nem arestas múltiplas são chamados de *grafos simples*.

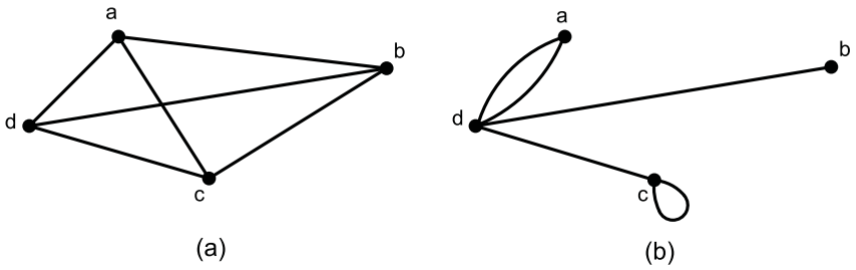

Figura 2 – Exemplos de grafos.

Fonte: Costa (2011).

Na Figura 2 (a), vê-se um grafo G. Nele é possível observar o conjunto de vértices $V(G) = \{a, b, c, d\}$ e o conjunto de arestas $E(G) = \{ab, ac, ad, bc, bd, cd\}$, assim a cardinalidade de $|V(G)| = 4$ e cardinalidade de $|E(G)| = 6$. Já na Figura 2 (b), tem-se um grafo que foi denotato por H, onde se observa um conjunto de vértices $V(H) = \{a, b, c, d\}$, e um conjunto de arestas $E(H) = \{ad, ad, db, dc, cc\}$, com as cardinalidades $|V(G)| = 4$ e $|E(H)| = 5$.

Definição 1.2 *Um grafo dirigido ou direcionado $G = (V, E)$ consiste em um conjunto não-vazio de vértices V e, ainda, um conjunto de arestas direcionadas E. Cada aresta $a \in E$ é especificada por um par ordenado de vértices $u, v \in V$.*

Na Figura 3, observa-se um grafo com $|V(G)| = 4$ e $E(G) = 6$, onde E é um conjunto de arestas direcionadas. Por exemplo a aresta que liga os vértices a e b pode ser descrita por $e = (a \to b)$ (LEHMAN et al., 2010).

Na literatura existem vários algoritmos para resolver problemas de programação biobjetivo (definidos na seção 1.2) em grafos, dirigidos ou não, que podem representar

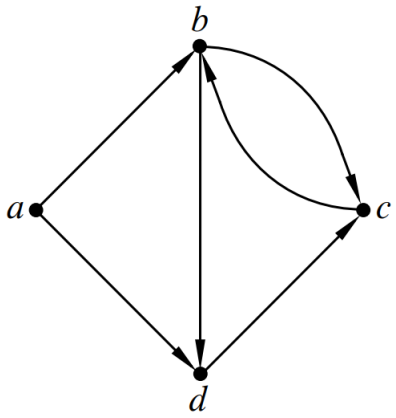

Figura 3 – Grafo dirigido com seis arestas.

Fonte: Lehman et al. (2010).

uma rede. Ainda, pode ocorrer o que se chama de *ciclo*. Um ciclo é um passeio de comprimento mínimo três (no mínimo de três vértices), em que o primeiro e o último vértices coincidem, mas nenhum outro vértice é repetido. Um *passeio* em um grafo G é uma lista de vértices, em que cada vértice é adjacente ao seguinte. Ou seja, um passeio $\{u = u_o, u_1, u_2, \ldots, u_k = v\}$ é um passeio em um grafo cujo primeiro vértice é u e o último vértice é v (SCHEINERMAN, 2011). É possível observar um exemplo de ciclo na Figura 2 (a), formado pelos vértices $\{a, b, c, d\}$.

Como os grafos são parte importante para a representação de redes na resolução de problemas envolvendo roteamento de fluxos em rede, a seguir são definidos os problemas multiobjetivo, bem como métodos para resolvê-los e tipos de soluções geradas por esses métodos.

1.2 Problemas multiobjetivo

Os problemas de otimização multiobjetivo visam a obtenção de um vetor (conjunto) de variáveis de decisão que satisfaça algumas restrições e otimize um vetor cujos elementos são diversas funções-objetivo. Dentre os problemas multiobjetivo, existem os problemas biobjetivo. Esses, por sua vez, buscam atender a dois objetivos.

Os problemas multiobjetivo (PMO) partem da constatação de que, quando existirem vários objetivos, existirão, também, soluções que satisfarão todos os objetivos de forma a melhorá-las. Em outras palavras, existirão soluções que, se comparadas a outras soluções, serão melhores considerando-se algum dos objetivos, e piores, considerando outros objetivos. Esse último conjunto de soluções é um dos principais objetivos da otimização multiobjetivo (TEIXEIRA, 2001).

Em (RAUPP, 2012), otimização multiobjetivo é definido como: dado um vetor de

variáveis de decisão com dimensão n, $x = \{x_1, \ldots, x_n\}$ no espaço de busca X, queremos encontrar um vetor $x^* \in X$ que minimiza simultaneamente as r funções-objetivo $f(x^*) = [f_1(x^*), \ldots, f_r(x^*)]$.

No cotidiano é comum encontrar problemas que consideram vários objetivos. Um exemplo é a compra de um computador: a solução ótima consiste no menor preço, tendo o melhor desempenho da máquina. Ambos os objetivos são conflitantes, uma vez que um alto desempenho implica um alto custo, e um baixo desempenho implica um baixo custo (RAUPP, 2012).

Nos PMO existem vários tipos de soluções, tais como: solução dominada, não-dominada e Pareto-ótima. Diz-se que uma solução A é dominada quando, a partir do método de busca utilizado, encontra-se outra solução B de forma que o valor para algum dos objetivos em B foi melhor que em A, mas sem prejudicar as outras restrições do problema, fazendo com que B domine A. Portanto B é uma solução não-dominada, quando comparada com A (HASHIMOTO, 2004).

Eficiência de Pareto ou Pareto-ótima é um conceito proposto por Vilfredo Pareto. Ele define um modo de alocação de recursos em que é impossível realocá-los sem que a otimização de um componente atrapalhe a de outro (PARETO, 2014).

A seguir apresenta-se as definições formais desses tipos de soluções dos problemas multiobjetivo.

Vamos adotar a notação para induzir ordenação sobre $\mathbb{R}^n$ como segue. Sejam $z^1 = (z_1^1, z_2^1, \ldots, z_n^1) \in \mathbb{R}^n$ e $z^2 = (z_1^2, z_2^2, \ldots, z_n^2) \in \mathbb{R}^n$, então (i) $z^1 \leq z^2 \iff z_k^1 \leq z_k^2, \forall k = 1, 2, \ldots, n$ e $z^1 \neq z^2$; (ii) $z^1 < z^2 \iff z_k^1 < z_k^2, \forall k = 1; 2, \ldots, n$.

Considere X o *conjunto de soluções viáveis* de um problema multiobjetivo na *minimiza*ção de $z(x)$, onde $z(x) = [z_1(x), z_2(x), \ldots, z_p(x)]$, o *vetor objetivo* associado a $x \in X$.

Definição 1.3 *Para $x, x' \in X$, dizemos que x domina x' quando $z(x) < z(x')$.*

Definição 1.4 *Uma solução viável $x' \in X$ é uma solução fracamente eficiente se não existe nenhum $x \in X$ tal que $z(x) < z(x')$.*

Definição 1.5 *Uma solução viável $x' \in X$ é uma solução Pareto-ótima (estritamente eficiente) se não existe nenhuma solução em X que domine x.*

Definição 1.6 *O conjunto $X^* \subseteq X$, formado por soluções Pareto-ótimas, é um conjunto mínimo completo quando:* 1) *para cada par de soluções $x^*, \bar{x} \in X^*$ temos $z(x^*) \neq z(\bar{x})$; e,* 2) *para qualquer solução Pareto-ótima $x \in X$ existe $x^* \in X^*$ tal que $z(x) = z(x^*)$.*

1.2.1 Técnicas para resolver problemas multiobjetivo

Na literatura existem várias técnicas para se resolver problemas com vários objetivos. Duas delas são o método da Soma Ponderada e o método ϵ-constraint (HASHIMOTO, 2004).

O método da soma ponderada consiste em somar todas as funções-objetivo, usando diferentes coeficientes (pesos) para cada uma delas. Isso significa que o problema de otimização multi-objetivo é transformado em um problema de otimização escalar, na forma:

$$minimizar \quad \sum_{i=1}^{k} w_i f_i(x)$$

$onde \mathrm{w}_i \geq 0$ são os coeficientes de peso, representando a importância relativa dos objetivos $f_i(x)$ (COELLO, 2000).

Outro método que se destaca é o método das restrições (ϵ-constraint). Suponha que o problema multiobjetivo tenha p funções objetivo:

$$\begin{aligned} &minimizar && f_1(x) \\ &minimizar && f_2(x) \\ & && . \\ & && . \\ & && . \\ &minimizar && f_p(x) \\ &sujeito\ a\text{:} && \\ & && x \in X \end{aligned}$$

O método das restrições (ϵ-constraint) consiste em criar um problema restrito (subproblema monobjetivo) onde a função objetivo é uma das p funções objetivo e as $(p-1)$ funções restante do problema multiobjetivo se transforma em restrições desse subproblema, resolvendo-o, repetidamente.

$$\begin{aligned} &minimizar && f_k(x) \\ &sujeito\ a\text{:} && x \in X \\ & && f_j(x) \leq \epsilon_j, \text{ para todo } j = 1, \ldots, p \text{ e } j \neq k \end{aligned}$$

Portanto, o método ϵ-constraint se baseia na minimização ou maximização de um dos objetivos, sendo que os demais são considerados como restrições. Dessa forma, um

problema de otimização mono-objetivo é resolvido para diferentes valores de ϵ. A variação desse parâmetro permite a geração das soluções Pareto-ótimas do problema. (TEIXEIRA, 2001)

Por exemplo, o método ϵ-constraint é utilizado em (GOVINDAN et al., 2016), como um método comparativo para os resultados. Nesse trabalho foi projetado, com programação matemática, um modelo multiobjetivo que se aproxima de uma rede de logística reversa sustentável. Esse modelo visa minimizar o custo do transporte, o impacto ambiental, bem como otimizar a responsabilidade social, tornando-se uma otimização com três funções-objetivo.

Outro trabalho que utiliza essa técnica pode ser encontrado em (SALVADOR, 2017), onde se buscava otimizar três funções objetivo: determinar a melhor configuração das cargas (sua organização), minimizar o custo e, ainda, a pegada de carbono (método que mede a emissão de gases estufa). Outros modelos de otimização foram usados, mas todos resultaram na mesma solução ótima, juntamente com o ϵ-constraint. Assim, é possível observar que esse método é utilizado nas diversas áreas de otimização, não somente na computação.

Todos esses conceitos dão base para o estudo dos problemas com mais de um objetivo em redes e, posteriormente, facilitará na compreensão do modelo biobjetivo de roteamento de fluxos em rede e do algoritmo exato para resolvê-lo, que é avaliado neste trabalho.

1.3 Uma investigação de problemas com mais de um objetivo em redes

Em (HANSEN, 1980) são apresentados vários problemas biobjetivo e algoritmos (polinomiais, pseudopolinomiais e polimoniais completos) que resolvem esses problemas de caminho bicritério. Para representar esses caminhos é utilizado grafo.

Dentre os modelos apresentados no artigo, existe um algoritmo que resolve o problema com uma função objetivo totalizadora (custo) e uma gargalo. As funções totalizadora e gargalo são funções que avaliam as soluções a partir da soma dos pesos dos elementos, e pelo pior peso de seus elementos, respectivamente.

Para o problema de caminho tri-critério, com duas funções gargalo e uma função totalizadora (custo). Em (PINTO et al., 2009) foi proposto o primeiro algoritmo exato, de complexidade polinomial, que obtinha um conjunto mínimo completo de soluções Pareto-ótimas. O algoritmo consiste em determinar caminhos mínimos em subgrafos obtidos de um conjunto restrito de arestas, atribuindo limites para a função gargalo. Um ano mais tarde, foi apresentado em (PINTO; PASCOAL, 2010), uma otimização para o trabalho anterior, com melhor desempenho computacional. Esse problema foi generalizado

em (BORNSTEIN et al., 2012). Nesse trabalho, o algoritmo proposto em (PINTO; PASCOAL, 2010) foi reotimizado, considerando, então, uma função totalizadora e n funções gargalo.

Vários outros algoritmos foram propostos para resolver esses tipos de problemas, mas eles o fazem considerando fixos os pesos dos elementos que compõem as soluções (por exemplo arestas de um grafo). Isso se deve ao fato de que eles trabalham com ordenação dos pesos associados às funções-objetivo de gargalo e, a cada iteração, definem e resolvem um problema mono-objetivo adicionando ou removendo elementos a partir de tal ordenação. Já em (PINTO et al., 2019; PINTO; FERNANDES, 2016) os valores relacionados à função gargalo não são fixos.

Em (GÁLVEZ; RUIZ, 2013) e (MELLO, 2014) observam-se propostas de heurísticas para resolver problemas de roteamento de fluxos em redes. Eles fazem o que se chama de *balanceamento de carga*, que consiste em rotear os fluxos de dados de forma a minimizar o gargalo na rede. Outro objetivo, além do balanceamento descrito, consiste em minimizar o número de saltos dentro dessa mesma rede, e, nesses casos, os enlaces passam a ter pesos dinâmicos (o que não seria resolvido pelos algoritmos anteriores).

Agora, será apresentado, com mais detalhes, o modelo biobjetivo de roteamento de fluxos em rede e o algoritmo exato que determina um conjunto mínimo completo de soluções Pareto-ótimas para esse problema, apresentado em (FERNANDES, 2019; PINTO et al., 2019).

Considere um conjunto de fluxos F, com $|F| = r$, a ser roteado em um grafo orientado $G = (V, E)$, onde V representa o conjunto de nós e E, o conjunto de arestas, com $|V| = n$ e $|E| = m$. Cada fluxo $f \in F$ deve ser roteado por um único caminho em G, indo de sua origem $s^f \in V$ e ao seu destino $d^f \in V$. Para cada aresta $(i, j) \in E$, indo de $i \in V$ para $j \in V$, e fluxo $f \in F$ definimos uma variável binária $x^f_{i,j}$, que irá indicar se o fluxo f vai passar (ou não) pela aresta (i, j). Considere, ainda, que $c^f_{i,j}$ seja o custo, não negativo, para o fluxo f utilizar a aresta (i, j).

Assim, o modelo de programação inteira biobjetivo, que é formulado para o problema de rotear os r fluxos, minimizando o gargalo da rede e o custo total, é apresentado a seguir:

$$(P) \quad \textit{minimizar} \left\{ \max_{(i,j) \in E} \left\{ \sum_{f \in F} x^f_{ij} \right\} \right\} \tag{1.1}$$

$$\textit{minimizar} \left\{ \sum_{f \in F} \sum_{(i,j) \in E} c^f_{ij}.x^f_{ij} \right\} \tag{1.2}$$

sujeito a:

$$x_{ij}^{f} \in \{0,1\}\,,\ \forall (i,j) \in E \text{ e } \forall f \in F \tag{1.4}$$

Enquanto a Equação (1.1) busca minimizar a carga da(s) aresta(s) mais congestionada(s), a Equação (1.2) busca minimizar o custo total do roteamento dos fluxos. As restrições apresentadas nas Equações (**??**) garantem que cada fluxo $f \in F$ saia da sua origem, s^f, e chegue ao seu destino, d^f, passando por um único caminho.

Observe que as funções objetivo são conflitantes, pois minimizar a carga da(s) aresta(s) mais congestionada(s) deve implicar custos maiores (caminhos mais longos) para roteamento de fluxos. Da mesma forma, minimizar o custo total de todos os fluxos deve levar a um maior valor para a função gargalo. O algoritmo para resolver esse problema foi baseado no método das restrições ($\epsilon - constraint$) com o intuito de acharmos um conjunto mínimo completo de soluções Pareto-ótimas para o problema (P). Em cada iteração, um subproblema mono-onjetivo é definido e resolvido, o qual é um problema de programação inteira $0-1$ que denominamos por (P_ϵ).

A função objetivo de (P_ϵ) é dada por z_2, isto é, a função totalizadora (1.2) do problema biobjetivo (P). As restrições de (P_ϵ) são (**??**) e (1.4) de (P), juntamente com uma restrição sobre o gargalo, a qual é definida através do parâmetro inteiro e positivo que denotamos por ϵ. Esse parâmetro é usado para controlar os valores da função gargalo e para garantir a otimalidade do algoritmo. Assim, obteve o seguinte subproblema mono-objetivo PLI:

$$\begin{aligned}
(P_\epsilon)\ \ & minimizar \left\{ \sum_{f \in F} \sum_{(i,j) \in E} c_{ij}^{f}.x_{ij}^{f} \right\} \\
& sujeito\ a: \\
& \sum_{(i,j) \in E} x_{ij}^{f} - \sum_{(j,i) \in E} x_{ji}^{f} = \begin{cases} 1, & \text{se } i = s^f \\ 0, & \forall i \in V - \left\{d^f, s^f\right\} \\ -1, & \text{se } i = d^f \end{cases} \qquad \forall f \in F
\end{aligned} \tag{1.5}$$

$$\begin{aligned}
\sum_{f \in F} x_{ij}^{f} \ \leq\ \epsilon,\ \forall (i,j)\ \in\ E \\
x_{ij}^{f}\ \in\ 0,1,\ \forall (i,j)\ \in\ E\ \text{ e }\ \forall f \in F
\end{aligned} \tag{1.6}$$

Inicialmente faz-se $\epsilon = r$ (quantidade de fluxos a serem roteados) e, após a obtenção da solução ótima $\bar{x}$ para (P_ϵ), fazemos $\epsilon = z_1(\bar{x}) - 1$. Dessa forma, o gargalo da nova solução, $\bar{x}$, obtida na iteração seguinte, será menor do que o da anterior, renomeada para $\hat{x}$. Então, caso o valor de z_2 seja o mesmo para essas duas soluções, temos que a solução anterior é

dominada pela nova solução, dai a solução anterior é deletada. Assim, segue o algoritmo até uma iteração em que (P_ϵ seja inviável. O algoritmo completo é apresentado a seguir.

Algoritmo 1

1. $X^* \leftarrow \emptyset$; $\epsilon \leftarrow r$.
2. ***Enquanto*** (P_ϵ) *não for inviável* ***faça***
3. $\quad \bar{x} \leftarrow$ *solução ótima de* (P_ϵ)
4. $\quad X^* \leftarrow X^* \cup \{\bar{x}\}$
5. $\quad \epsilon \leftarrow z_1(\bar{x}) - 1$
6. $\quad$ ***Se*** $|X^*| > 1$ ***e*** $z_2(\bar{x}) = z_2(\hat{x})$ ***então***
7. $\quad\quad X^* \leftarrow X^* - \{\hat{x}\}$
8. $\quad \hat{x} \leftarrow \bar{x}$

O decremento de exatamente uma unidade no valor de ϵ, de uma iteração para a próxima, é necessário para garantir a obtenção de um conjunto mínimo completo de soluções Pareto-ótimas. Como os valores de z_1 são inteiros, a atualização de ϵ garante que qualquer solução, com z_1 menor do que o da última solução obtida, será viável para o subproblema atual. Porém, embora ϵ sofra esse pequeno decréscimo, a solução obtida pode ter um valor para z_1 menor do que esse limite.

Como o intuito do trabalho é a avaliação desse algoritmo exato. Para tanto, será utilizado um emulador de redes, de forma que seja possível testar o algoritmo em uma rede real. Por esse motivo é apresentado a seguir uma seção que define e compara o simulador com o emulador.

1.4 Simuladores e Emuladores

Existem várias formas de testar a corretude e avaliar o desempenho de um protocolo de rede. Uma forma de fazer isso é a partir da simulação. Na simulação, as operações de dispositivos reais e suas interações são modelados e executados em programas de software. Uma das maiores vantagens da simulação é que tem um baixo custo computacional. Além disso, ela é flexível, controlável, escalável, duplicável e é acessível a vários usuários. No entanto, se a modelagem de dispositivos reais não for precisa, os resultados da simulação podem ser diferentes dos resultados de experimentos reais (WANG et al., 2013).

Para superar esses problemas com a modelagem do software de simulação, existe outra aproximação: a emulação. A emulação se difere da simulação no sentido de que a primeira

deve ser como um experimento real, e deve ser executado também em tempo real. No caso da simulação, o tempo de execução pode ser maior ou menor, dependendo da implementação. Além disso, na emulação, os dispositivos reais que executam sistemas operacionais vão interagir com dispositivos simulados. Já na simulação, normalmente não existem aplicações ou sistemas operacionais reais envolvidos.

Assim, a principal diferença entre *simulação* e *emulação* é a forma que o software é representado por eles. As características diferentes dessas técnicas fazem cada uma delas ser indicada para determinadas aplicações. Por exemplo, a simulação exige menos recursos de hardware, e pode ser usada no início do desenvolvimento de um projeto para avaliar conceitos e estratégias a serem usadas no mesmo. A simulação depende dos modelos de software e hardware para avaliação. Já a emulação se utiliza do software real implantado em um modelo de infraestrutura e hardware (MCGREGOR, 2002).

As técnicas de emulação e simulação são aproximações de algum sistema real. Portanto, existirá uma grande diferença de desempenho entre o que é emulado/simulado e do modelo real. Essas diferenças geram o que se chama de *credibility gap*, que traduzindo quer dizer "lacuna de credibilidade". Modelos de emulação tentam minimizar essa lacuna, de forma a trazer o sistema emulado para mais perto do sistema real. Isso é feito trazendo uma parte do sistema real para a emulação (CALHEIROS et al., 2013). Um exemplo de simulação são os simuladores de direção, utilizados em centros de formação de condutores.

De acordo com (MATTOS, 2008b), na seção "7. Cinco Perguntas" de um website construído a partir da pesquisa "Virtualização: VMWare e Xen", do mesmo autor (MATTOS, 2008a), um emulador implementa todas as instruções realizadas por uma determinada máquina (real) em uma camada de *software*, que executa sobre um *hardware*. Esse *hardware* pode ser completamente diferente do que está sendo emulado. No caso do Mininet, os *hardwares* a serem emulados são os *switches*, controladores, *hosts*, entre outros. Ou seja, um emulador simula uma máquina diferente do computador sobre o qual o mesmo opera. Isso é feito através de *software*, e de forma que todas as instruções do *hardware* que é emulado sejam traduzidas para instruções do sistema em que o *software* será executado.

Neste trabalho é utilizado o emulador denominado *Mininet*. A seguir apresenta-se a importância dessa ferramenta bem como suas vantagens e desvantagens de uso.

1.4.1 O Mininet

O Mininet cria uma rede virtual realística, sendo executada em um kernel Linux, *switch* e códigos da aplicação em uma máquina (*virtual machine* ou em um computador) (TEAM, 2018). Um *switch* pode ser compreendido como um equipamento para extensão física dos pontos de rede, ou seja, todos os dispositivos que se conectam em uma rede.

Esse emulador cria uma rede definida por software (SDN - *Software-Defined Network*). O princípio de uma SDN é a separação do plano de controle (sistema operacional da rede) do plano de dados (plano de encaminhamento) (FONTES; ROTHENBERG, 2019). Não existem muitos hardwares físicos que implementam características de SDNs, e os que existem são caros (OLIVEIRA et al., 2014), e por isso essa ferramenta foi utilizada. O conceito de SDN é ideal para experimentos com o OpenFlow que, por sua vez, fornece um protocolo aberto que permite programar o que se chama de tabela de fluxos (do inglês *flow-table*, que é responsável pelo plano de encaminhamento dos pacotes). Além disso, utilizou-se o RYU, um *Framework* que seria responsável por permitir a programação de um controlador e a criação das regras de roteamento (COMMUNITY, 2017). O principal intuito de utilizar esse *framework* é a criação de regras de acordo com as saídas obtidas a partir do algoritmo avaliado.

Portanto, um administrador de rede pode particionar o tráfego de pacotes em produção (*production*) e pesquisa (*research*). Os pesquisadores (*researchers*) controlam seus próprios fluxos, escolhendo para onde seus pacotes vão seguir (rotas) e o processamento que esses mesmos pacotes receberão. Assim, eles podem testar modelos de segurança, esquemas de endereçamento e, ainda, testar novos protocolos de roteamento. Algumas empresas desenvolvedoras de *switches* e roteadores começaram a implementar as tabelas de fluxos (do protocolo OpenFlow), no hardware de seus dispositivos (MCKEOWN et al., 2008).

Criar uma rede física para testar o algoritmo seria inviável, uma vez que o teste precisa ser feito com *switches* e *hosts* que implementassem características de SDNs. Portanto, foi necessário emular uma rede de forma que o algoritmo pudesse ser avaliado sob duas métricas: vazão e tempo de entrega dos pacotes.

Para atender essa necessidade, realizou-se uma pesquisa sobre os emuladores/simuladores que estavam sendo utilizados na literatura. Existem muitos pesquisadores avaliando SDNs com o Mininet, um deles é (OLIVEIRA et al., 2014), que avalia a criação e prototipação de SDNs na ferramenta.

Oliveira et al. (2014) apresentam, também, as dificuldades de se testar novos recursos de SDNs em controladores, computadores e até mesmo no protocolo *OpenFlow*. Observa-se, ainda, que em alguns casos específicos, utilizar a Internet pode não ser uma boa ideia. Um desses casos, por exemplo, pode ser a necessidade simular redes com grandes números de *hosts*, controladores, entre outros. Isso se deve ao fato de que a Internet pode apresentar configurações inadequadas e pode causar problemas indesejados.

Para resolver os problemas citados anteriormente, Oliveira et al. (2014) apontam uma solução: criar protótipos e simular os mesmos em um ambiente virtual. Uma das ferramentas que possibilita isso é o Mininet.

O autor descreve essa ferramenta como um “sistema que permite prototipagem rápida de

grandes redes de computadores em um simples computador". O trabalho mostra, ainda, algumas características do Mininet, tais como: flexibilidade, aplicabilidade, interatividade, escalabilidade, aproximação da realidade, entre outros.

Para ele, o software Mininet traz uma contribuição única para o avanço nos estudos de SDNs. Estudantes, pesquisadores, administradores de rede, entre outros, podem usar a ferramenta para prototipar rapidamente uma ideia de SDN.

A ferramenta está sendo usada por mais de 100 pesquisadores, em mais de 18 instituições (não só de ensino), incluindo Princeton, Berkeley, Purdue, ICSI, UMass, University of Alabama Huntsville, NEC, NASA, Deutsche Telekon Labs, Stanford e em companhias de *startups*. Além de internacionalmente usada, um total de sete universidades utilizam essa ferramenta no Brasil. Isso é um bom indicador das vantagens do Mininet (OLIVEIRA et al., 2014).

Em (KETI; ASKAR, 2015), a ferramenta é testada nos ambientes Windows e Linux, entre outros. Assim foi possível observar as vantagens e desvantagens dessa ferramenta. Além de pesquisar a preferência de alguns autores pela ferramenta sobre as outras disponíveis, verificou-se que o Mininet é uma ferramenta gratuita.

Assim, neste trabalho optou-se pelo Mininet, devido às pesquisas realizadas, e a partir da constatação de que essa ferramenta está sendo utilizada por vários pesquisadores da área, além disso, considerou-se as vantagens demonstradas numericamente em (OLIVEIRA et al., 2014).

Por exemplo, nesse artigo, apresentam resultados do teste de escalabilidade como pode ser observado no quadro da Figura 4.

A topologia usada nessa análise é do tipo *Tree* (árvore), definida na coluna *Topology*. Essa estrutura pode ser melhor compreendida em (XIE; WANG, 2014). Além do tipo de rede, as métricas usadas são o número total de nós (*Node numbers*), que é a soma de *Host numbers* (número de *hosts*) e o número de *switches* (*Switch numbers*). Além disso, é avaliado o tempo entre o início e o fim da execução (*start/stop time (in seconds)*). Por último, tem-se o uso de memória para a criação da rede (*Memory usage (in MB)*).

Topology	Node numbers	Host numbers	Switch numbers	start/stop time (in seconds)	Memory usage (in MB)
Tree	3	2	1	0,190	441
Tree	7	4	3	0,525	442
Tree	15	8	7	1,175	445
Tree	31	16	15	2,795	452
Tree	63	32	31	7,088	466
Tree	127	64	63	20,210	449
Tree	255	128	127	64,818	572
Tree	511	256	255	242,842	739

Figura 4 – Teste de escalabilidade.

Fonte: Oliveira et al. (2014).

Na Figura 4 nota-se que para a criação de redes pequenas o tempo também é pequeno. Aumentando-se o número de nós, o tempo também aumenta. Por exemplo, para criar uma rede com 255 nós, o emulador levou aproximadamente 64 segundos. Já para uma rede de 511 nós, o tempo aumentou substancialmente para cerca de 242 segundos.

Verifica-se, ainda, que até o valor de 127 nós, os tempos foram relativamente próximos. Depois disso, com o aumento dos dispositivos virtuais, o uso de memória começou a crescer.

Um fator de influência, mas não tão importante, foi o suporte que a ferramenta oferece à linguagem de programação *Python*. Isso acelerou um pouco o desenvolvimento, pois a linguagem já é previamente conhecida.

2 METODOLOGIA

Neste capítulo, é explicitado sobre a metodologia utilizada para este trabalho. Primeiramente, apresenta-se natureza da pesquisa. A seguir, é descrito o contexto da pesquisa. E, por último, elucida-se os instrumentos de avaliação e os procedimentos de análise dos dados.

2.1 Abordagem

Para atingir os objetivos deste trabalho foram usadas duas abordagens: uma abordagem teórica e uma prática. Na primeira, foi necessária uma ampla investigação sobre problemas multiobjetivo e problemas biobjetivo, estudando técnicas para resolvê-los. Além disso, foi necessário um estudo sobre grafos e como eles podem ser utilizados na representação gráfica de uma rede. Depois, partiu-se para um estudo aprofundado sobre o modelo e o algoritmo apresentado em Fernandes (2019), com intuito de avaliar o algoritmo exato apresentado nesse trabalho. Por último, estudou-se a ferramenta Mininet, para fins de avaliação desse algoritmo.

A segunda abordagem, a parte prática, toda a pesquisa realizada foi utilizada para implementar as saídas do algoritmo na ferramenta Mininet. Quando implementado, foi possível capturar métricas como tempo de entrega de pacotes e o número de pacotes enviados.

2.2 Procedimentos

Para o desenvolvimento deste trabalho, focou-se na parte dos problemas biobjetivo. Esse aprofundamento foi necessário porque o algoritmo avaliado nesse trabalho trata de uma algoritmo exato para resolver um problema biobjetivo de roteamento de fluxos em redes.

Posteriormente, após a compreensão do que são problemas biobjetivo, iniciou-se o estudo do modelo proposto em Fernandes (2019). O modelo busca satisfazer, simultaneamente, duas funções objetivo: minimizar o número total de saltos dentro de uma rede e minimizar o gargalo nos enlaces dessa mesma rede. Após a criação desse modelo, eles criaram um algoritmo baseado na técnica ϵ-constraint (TEIXEIRA, 2001). Os resultados gerados (soluções Pareto-ótimas) pelo algoritmo serviram de base para a implementação da rede que seria emulada pela ferramenta Mininet.

Depois de todo o processo citado anteriormente, iniciou-se a fase de estudo da ferramenta Mininet que foi utilizada para realizar a avaliação do algoritmo. Primeiramente foi

necessário estudar a documentação desse emulador e compreender suas ferramentas.

Assim, neste trabalho apresenta um capítulo do funcionamento dessa ferramenta, juntamente com a descrição de como ela foi utilizada no decorrer desse trabalho, assim vai facilitar a compreensão do mesmo por parte do leitor.

Por último, foi implementado o algoritmo proposto em (FERNANDES, 2019), sendo executado em C++, juntamente com o *solver* CPLEX, tendo como saída um conjunto contendo rotas eficientes para o roteamento dos fluxos.

Esse conjunto de soluções pode retornar várias soluções eficientes. No entanto, para questões de análise, foram utilizadas a primeira e a última solução Pareto-ótima gerada.

Por último, foi feito as avaliações do algoritmo exato através do emulador de redes Mininet, o qual permitiu capturar métricas como tempo mínimo e máximo para entrega dos pacotes, usando as rotas eficientes geradas pelo algoritmo exato.

3 FUNCIONAMENTO DA FERRAMENTA MININET

O Mininet suporta pesquisa, desenvolvimento, aprendizado, prototipação, testes, ou qualquer outra tarefa que se beneficie disso. Ele fornece uma experiência de rede completa em qualquer máquina (computador, notebook, máquinas virtuais, entre outros), e isso foi explorado na análise do modelo avaliado.

A partir disso, a ferramenta foi instalada em ambiente *Linux*, na distribuição *Mint* versão 14, em um dispositivo com espaço de 8GB de memória RAM DDR3, um processador Intel Core i7 6500u. Para instalar o ambiente, foi necessário clonar um repositório do *GitHub*. Isso é feito executando, em um terminal, o seguinte comando:

```
$ git clone git://github.com/mininet/mininet
```

Dessa forma, todos os arquivos do Mininet foram clonados do repositório. Agora, basta navegar, a partir do terminal, até a pasta onde se encontra o arquivo de instalação. Para tanto, os seguintes comandos podem ser executados:

```
$ mininet/util/install.sh -a
```

onde "-a" é um parametro que instala todos os recursos da Máquina Virtual do Mininet.

A ferramenta provê, ainda, comandos que criam redes simples. Por exemplo, o comando abaixo gera uma topologia mínima

```
$ sudo mn
```

Essa topologia é uma rede virtual com um switch e dois hosts ligados a ele. Para testar se a rede está funcionando, basta testar a conexão enter os hosts (se os dois conseguem se comunicar). Isso é feito com o comando:

```
mininet> h1 ping h2
```

Quando a rede é criada, o CLI (*command-line interface*), interface de linha de comando, do mininet é aberto. Com isso, é possível executar alguns comandos básicos, tais como:

```
mininet> help
mininet> nodes
mininet> net
mininet> dump
```

Onde *help*, *nodes* e *net* exibem, respectivamente, os comandos do CLI, os nós, os enlaces. O comando *dump* "esvazia" todas as informações sobre os nós. Se a primeira palavra

digitada no comando for um *host*, *switch* ou controlador, o mesmo será executado no nó em questão. Por exemplo, no comando:

```
mininet> h1 ping h2
```

citado anteriormente, o *host* h_1 enviará pacotes para o *host* h_2.

No entanto, uma rede simples como essa não atenderia às necessidades desse trabalho, então foi necessário implementar uma rede maior (para isso, utilizou-se o Grade). Essa, por sua vez, foi implementada em Python (versão 2.8), juntamente com o Mininet e o *Framework* Ryu, que suporta o protocolo OpenFlow.

3.1 Criação da rede e manipulação das regras de roteamento

A primeira tentativa de executar uma rede maior não obteve sucesso, já que não existiam regras de roteamento (para qual *switch* o pacote recebido de um *host* seria encaminhado) definidas nos *switches* e no controlador. Daí a necessidade de se utilizar o *framework* Ryu: ele possibilita a programação de um controlador (responsável por conectar os dispositivos de uma rede) e das regras de roteamento dos *switches* OpenFlow.

Inicialmente, a criação das regras foi feita de forma manual para maior entendimento do funcionamento das mesmas. Foi necessário alterar o arquivo do controlador, que estava escrito em Python. Uma estrutura de dados, comumente conhecida como "dicionário", nessa mesma linguagem, foi criada. Nela são definidos o *host* que está mandando os pacotes, o *switch* responsável por encaminhar esses pacotes, e as portas em que os hosts e os outros switches se comunicam com o mesmo, formando assim o que se chama de "tabela de fluxos". Um exemplo dessa estrutura de dados pode ser observado na Figura 5.

```
1  TOPOLOGY = {
2      "00:00:00:00:00:01": {
3          "00:00:00:00:00:02": {
4              1: {'in_port': 1, 'out_port': 2},
5              2: {'in_port': 1, 'out_port': 2},
6          }
7      },
8      "00:00:00:00:00:02": {
9          "00:00:00:00:00:01": {
10             1: {'in_port': 2, 'out_port': 1},
11             2: {'in_port': 2, 'out_port': 1},
12         },
13     }
14 }
```

Figura 5 – Estrutura de dados Dicionário.

Fonte: Autoria própria.

A estrutura funciona da seguinte forma: as duas primeiras linhas da regra definem a origem e o destino de um pacote. Na linha 2, tem-se o endereço MAC do *host* que envia o

pacote. No caso do exemplo, quem está fazendo isso é o $host_1$. Na linha 3, tem-se o endereço MAC do *host* que receberá o pacote, que nesse caso, é o $host_2$.

As linhas 4 e 5 são referentes aos *switches* em que esses *hosts* estão conectados. O primeiro número em ambas as linhas indica que aquilo é relativo ao *switch* que o pacote precisará passar. A linha 4 refere-se ao $switch_1$. Então, se um pacote está sendo enviado do $host_1$ para o $host_2$, quando ele chegar ao $switch_1$, pela porta de entrada (*in_port*) 1, deverá ser encaminhado para a porta de saída 2 (*out_port*), que é relativa ao $switch_2$. Depois disso, quando chegar ao $switch_2$ pela porta 1 (onde o $switch_1$ está ligado), será encaminhado para a porta 2, que é a porta que o $host_2$ está conectado.

A segunda parte, das linhas 8 até 13, é relativa ao caminho de volta do pacote. Agora a rota é do $host_2$ para o $host_1$. A linha 11 indica que, quando o pacote chegar ao $switch_2$ pela porta 2 (onde o $host_2$ está conectado), ele deverá ser encaminhado para a porta 1, que é onde o $switch_1$ está conectado. Assim que o pacote chega ao $switch_1$ (verifique a linha 10), pela porta de entrada (*in_port)* 2, que é onde o $switch_2$ está conectado, ele deve ser encaminhado para a porta 1, e isso fará com que o pacote finalmente chegue ao $host_1$.

Depois disso, inicia-se um processo de "automatizar" as regras, pois seria inviável escrevê-las manualmente, já que qualquer mudança no número de *switches* acarretaria na necessidade de reescrevê-las. As regras são criadas a partir da execução de algum algoritmo (inicialmente aplicou-se o algoritmo Dijkstra, para encontrar os caminhos mais curtos dentro da rede). A criação passou a ser feita de forma iterativa, de forma que cada regra é adicionada à tabela de fluxos a partir de um método pertencente ao *framework* Ryu.

As regras para essa rede são criadas a partir da estrutura definida na Figura 5. As implementações utilizadas para criar topologia e o controlador podem ser encontradas em um repositório do *GitHub* (verifique o repositório clicando aqui, ou acessando "https://github.com/stefany-fernandes/mininet-controlador-topologia").

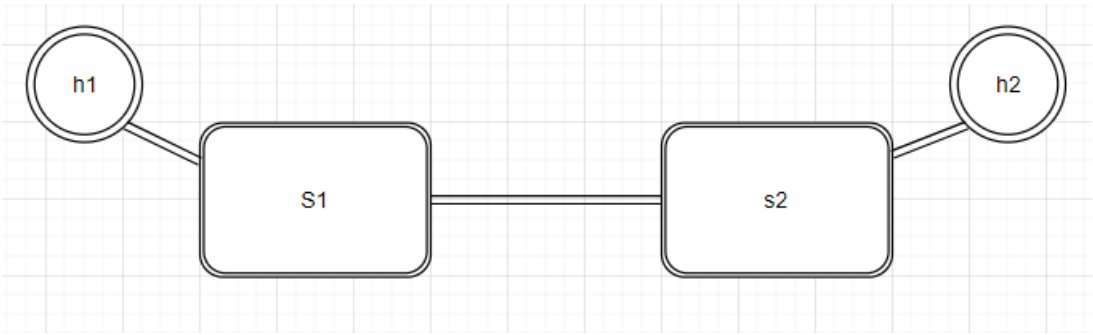

Figura 6 – Rede simples de dois *switches* ($s1$ e $s2$) e dois *hosts* ($h1$ e $h2$).

Fonte: Autoria própria.

O arquivo "controlador.py" é, como o próprio nome diz, o controlador referente à rede. Já

o arquivo "rede.py" é referente à criação da rede. A criação das regras é feita como explicitado na Figura 5, assim sendo, é possível fazer as alterações de acordo com as necessidades. Basta alterar a rede em seu respectivo arquivo e alterar as regras no arquivo do controlador, na variável TOPOLOGY. Além desses dois arquivos, existe um arquivo que é a figura apresentada nesse trabalho (Figura 6).

3.2 Implementação do algoritmo Dijkstra

Para fins de testes no controlador, decidiu-se implementar o algoritmo Dijkstra. O funcionamento do algoritmo, retirado de (JASIKA et al., 2012), pode ser visto no Algoritmo 2.

Algoritmo 2

1. *Escolha o vértice de partida;*
2. *Defina um conjunto S e incialize-o como conjunto vazio. Enquanto o algoritmo progride, o conjunto S vai guardar os vértices em que os caminhos mais curtos já foram encontrados;*
3. *Rotule o vértice de partida como 0, e insira-o em S;*
4. *Considere cada vértice que não esteja em S conectado por uma aresta para o vértice recém-inserido nesse mesmo conjunto. Rotule o vértice que não está em S com o mesmo rótulo do vértice que foi inserido + o peso da aresta. No entanto, se o vértice que não está em S já tiver sido rotulado, seu novo rótulo será mínimo(rótulo do vértice inserido + peso da aresta, rótulo antigo);*
5. *Escolha um vértice que não esteja em S com o menor rótulo e insira-o em S;*
6. *Repita do passo* 4 *em diante, até que o vértice de destino esteja no conjunto S ou não existam mais vértices (que não estejam em S) que possam ser rotulados.*

   Se o vértice de destino está rotulado com algum valor, esse valor será a distância entre o vértice inicial e o vértice de destino. Se não estiver, então não existe caminho do vértice inicial para o destino.

O Algoritmo 2 foi implementado em Python, juntamente com o controlador. O primeiro passo foi a criação da rede que o Mininet fornecia. Foi criada uma rede com cinco *switches*, cada um contendo um host ligado em si. Todos os switches dessa mesma rede estavam ligados entre si, formando um grafo completo. Um exemplo disso pode ser visto na Figura 7, onde h_1, h_2, h_3, h_4 e h_5 representam os cinco *hosts*, e s_1, s_2, s_3, s_4 e s_5 representam os cinco *switches*.

Assim, sempre que a topologia de rede fosse criada, com os *switches* (vértices) e os pesos dos enlaces (arestas ou *links*), o algoritmo seria executado. Dessa forma, as rotas com os

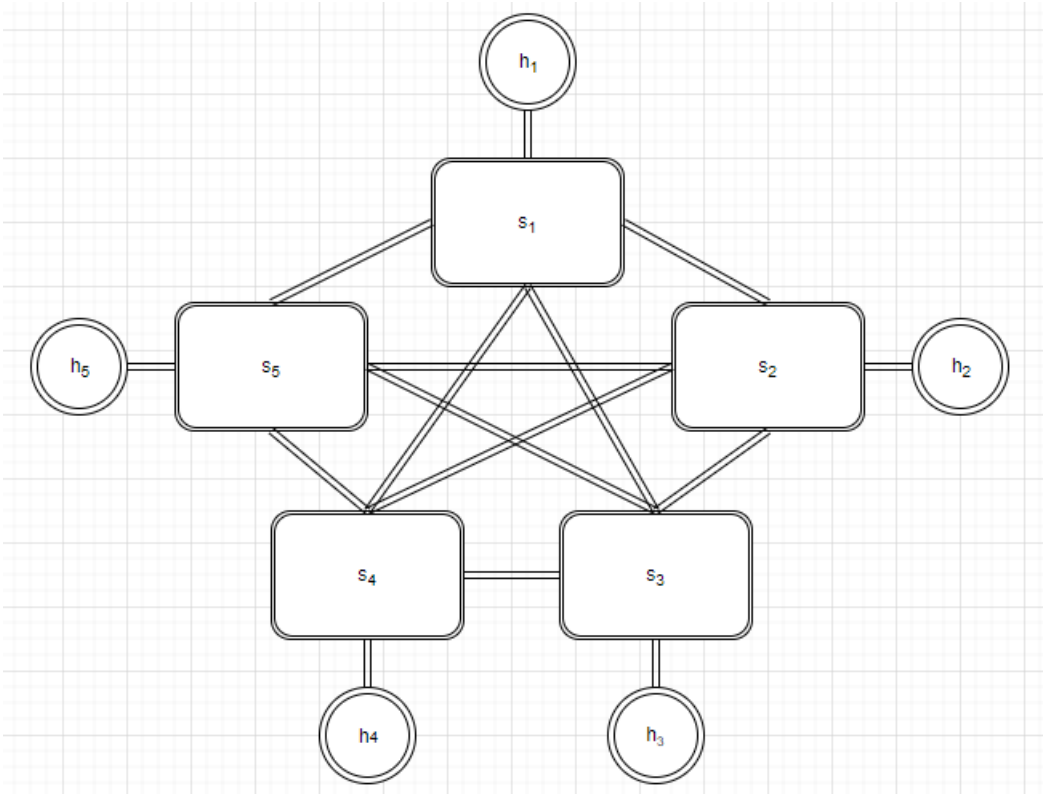

Figura 7 – Rede de cinco *switches* (s_i) e cinco *hosts* (h_i).

Fonte: Autoria própria.

melhores caminhos seriam gerados, possibilitando a construção da tabela de roteamento. Mais detalhes sobre a implementação do algoritmo na ferramenta serão dados a seguir.

3.3 Emular o Algoritmo 1

Por fim, após um estudo aprofundado sobre o Mininet (TEAM, 2018) e suas ferramentas, iniciou-se o processo de análise do Algoritmo 1.

Esse algoritmo consiste em encontrar um conjunto mínimo completo de soluções Pareto-ótimas que satisfaça, simultaneamente, os objetivos de minimizar o gargalo sobre os enlaces (arestas) da rede, enquanto minimiza, também, o número de saltos de todos os fluxos ao fazer o roteamento dos fluxos na rede.

Em um primeiro momento, o algoritmo proposto em (FERNANDES, 2019) foi executado em C++, juntamente com o *solver* CPLEX para implementar o modelo mono-objetivo, tendo como saída um conjunto mínimo completo de soluções Pareto-ótimas (X^*), ou seja, um conjunto contendo rotas eficientes para o roteamento dos fluxos.

O conjunto de soluções Pareto-ótimas pode retornar várias soluções eficientes. No entanto, para questões de análise, foram utilizadas a primeira e a última solução Pareto-ótima. Isso foi feito para que fosse possível analisar se o tempo teria um aumento significativo, já que

última solução Pareto-ótima gerada o gargalo diminui em relação a primeira solução Pareto-ótima, porém o número total de saltos aumenta.

Depois disso, o arquivo do controlador é responsável por fazer o devido encaminhamento dos pacotes nessas rotas. O controlador é necessário para o caso de não existir regras de roteamento nos *switches*. Se o *switch* não tem uma tabela de fluxos (ou *flow table*) definida, então o pacote é encaminhado para o controlador e, esse, por sua vez, faz o devido encaminhado para o *switch* de destino. Depois disso, a regra é adicionada à tabela de fluxos. Caso exista uma regra (criada a partir da rota percorrida por um fluxo), o pacote é encaminhado sem a necessidade do controlador, passando diretamente de um *switch* para outro.

O primeiro passo foi a alteração no controlador fornecido pelo Mininet. Como já dito, esse controlador não é capaz de atender as necessidades desse trabalho. Isso se deve ao fato de que cada fluxo (ou pacote) deve ser roteado por um único caminho gerado pelo Algoritmo 1. Em outras palavras, o algoritmo gera caminhos para o roteamento dos fluxos considerados, se preocupando em minimizar o número de saltos de todos os fluxos e minimizar o gargalo da rede, ou seja, minimizar a quantidade máxima de fluxos que atravessa os enlaces(arestas) da rede. Então, o caminho entre dois nós da rede, A e B, por exemplo, é definido de acordo com as saídas do algoritmo em questão.

Para testar se as regras criadas a partir das saídas do algoritmo estavam corretas, foi necessário checar se dois nós de uma rota eram capazes de se comunicar. Em todos os casos testados eles se comunicavam, isso deve ao fato de a topologia da rede que foi considerada é o grafo Grade.

Outro teste necessário foi verificar se outra topologia para rede ocorresse de dois nós que não estavam conectados por nenhuma rota conseguiam se comunicar. O certo é que a comunicação não ocorresse, uma vez que, como não existia rota definida para aquela comunicação, não deveria existir, também, uma regra que conectasse ambos. Para isso, foi necessário utilizar o comando *iperf*.

O comando *iperf* funciona da seguinte forma: um nó (de origem ou destino, a ordem não importa) é definido como *server* (servidor), e o outro é definido como *client* (cliente). Se existir uma regra de roteamento entre esses dois nós, eles começarão a trocar dados no momento em que se conectarem. No entanto, caso não exista uma regra, não haverá nenhuma troca de informações entre eles. Dessa forma foi possível checar a "corretude" das regras criadas a partir dos resultados do algoritmo.

4 RESULTADOS

Na análise, considerou-se o grafo grade, a-por-b, definido da seguinte forma: seja V o produto cartesiano $\{1, 2, \ldots, a\} \times \{1, 2, \ldots, b\}$, ou seja, o conjunto de todos os pares ordenados (u, v) com $u \in \{1, 2, \ldots, a\}$ e $v \in \{1, 2 \ldots, b\}$. Diga-se que dois elementos (u, v) e (u', v') de V são adjacentes se $u = u'$ e $|v - v'| = 1$ ou se $v = v'$ e $|u - u'| = 1$ ou se $u' = u + 1$ e $v' = v + 1$ ou se $u' = u + 1$ e $v' = v - 1$, onde $|u - u'|$ representa o módulo da diferença entre os dois números inteiros u e u'. Essa relação de adjacência define um grafo sobre o conjunto V de vértices, como mostra a Figura 1.

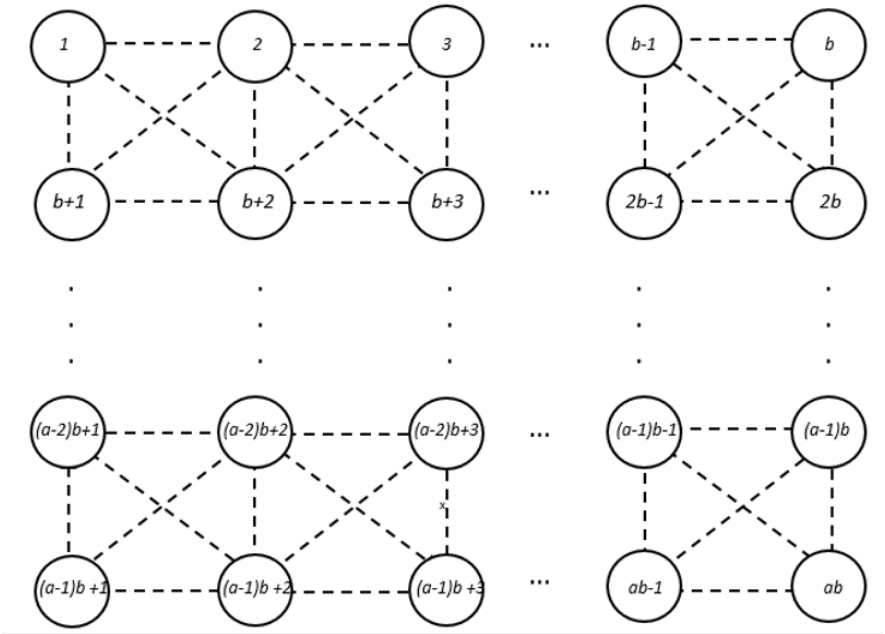

Figura 8 – Grafo do tipo *Grade*.

Fonte: Autoria própria.

A justificativa dessa escolha da topologia para rede é que a mesma é uma das topologias de rede consideradas em (PINTO et al., 2019) e (FERNANDES, 2019).

Cada nó do grafo consiste em um nó na rede. Os nós representam os *switches*, e cada *switch* tem um *host* conectado a si.

Quanto aos nós de origem e de destino de cada fluxo $f \in F$, são gerados aleatoriamente, considerando a configuração de distribuição dos fluxos: origens quaisquer (múltiplas origens) e destinos quaisquer (múltiplos destinos). Esse cenário foi avaliado com 80 fluxos, como adotado em (PINTO et al., 2019), empregados em 10 instâncias.

Primeiramente, considerou-se $a = b = 3$, para uma análise inicial da eficiência do algoritmo. Em um segundo momento, a rede utilizada foi do tipo $a = b = 7$, sendo, então,

uma rede 7×7, totalizando 49 nós.

A partir do modelo de rede definido anteriormente, foi possível criar regras, definir rotas e capturar algumas métricas utilizando a ferramenta (Mininet). A regra de uma configuração para a solução eficiente é definida a partir do nó de origem, os nós intermediários (caminho), e o nó de destino. Em um exemplo numérico, suponha que o fluxo tem a seguinte configuração de rota eficiente:

origem: $host_6$, destino: $host_{33}$, caminho: $6 \rightarrow 22 \rightarrow 17 \rightarrow 33$

Antes de chegar ao seu destino, o pacote deve sair do $host_6$ para o *switch* em que o mesmo está conectado, para que ele faça o devido encaminhamento. Para facilitar a implementação, um *host* teria a mesma numeração que seu respectivo *switch*, ou seja, o $host_n$ está conectado ao $switch_n$. Portanto, o pacote sairia do $host_6$ para o $switch_6$, seguiria para o $switch_{22}$, depois para o $switch_{17}$, para o $switch_{33}$ e esse, por sua vez, encaminharia o pacote para o host conectado a ele ($host_{33}$).

Assim sendo, todos os fluxos de 10 instâncias diferentes foram avaliados. Cada um deles se tornaria uma regra no controlador. A cada instância, que possui um total de oitenta fluxos, a distribuição dos mesmos pode ser diferente. Ou seja, a distribuição dos fluxos da primeira instância é diferente da distribuição da segunda, apesar de utilizarem a mesma rede. A configuração de fluxos avaliada foi a de origens e destinos diferentes, como comentado anteriormente. Esse algoritmo avalia, ainda, qualquer configuração de fluxos, por exemplo, configuração para o mesmo destino, porém não foi implementada neste trabalho.

Inicialmente, uma das métricas a serem capturadas a partir da implementação seria a perda de pacotes. No entanto, o protocolo TCP/IP garante a entrega, ordenamento e a não-duplicidade dos dados. As regras de transporte foram criadas em cima desse protocolo, portanto não houve perda de pacotes (COMER, 2016).

A Tabela 4 apresenta a quantidade de soluções Pareto-ótimas ($|X^*|$), geradas pelo Algoritmo 1 e após emular as soluções (primeira e última Pareto-ótima) no Mininet, foram avaliados os tempos mínimo (t_{min}), máximo (t_{max}) para a entrega dos pacotes de um nó origem para o nó destino no roteamento dos 80 fluxos,além disso, a média dos tempos para cada fluxo a ser roteado (t_{med}) para cada uma dessas soluções. Os tempos são representados em milissegundos para dez instâncias diferentes do conjunto de dados, numa rede grade 3×3.

Para a configuração de distribuição de fluxos considerada (origens e destinos quaisquer) observa que, apesar da última solução Pareto-ótima ter mais saltos do que a primeira, os tempos médios não têm uma amplitude dos tempos significativa, quando comparamos essas duas soluções em cada instância. Isso ocorre pois o número de saltos da primeira solução para a última é uma quantidade pequena (pois a rede é pequena).

Tabela 1 – Tempo do roteamento para configuração de distribuição de fluxos origens quaisquer e destinos quaisquer numa rede grade 3×3.

		Primeira solução Pareto-ótima			Última solução Pareto-ótima		
Instância	$\|X^*\|$	t_{min}	t_{med}	t_{max}	t_{min}	t_{mex}	t_{max}
1	3	58,72	76,49	128,15	51,611	70,341	86,860
2	2	73,40	67,27	87,03	49,41	82,66	124,4
3	1	89,04	87,14	116,65	-	-	-
4	1	61,364	123,76	162,33	-	-	-
5	1	48,76	70,20	111,593	-	-	-
6	2	59,77	69,17	105,56	47,81	66,44	106,55
7	2	57,97	86,56	130,79	64,58	79,76	118,69
8	1	45,74	68,37	112,69	-	-	-
9	2	63,77	79,496	122,69	62,94	78,35	114,79
10	2	52,51	79,022	126,56	53,96	85,88	130,96

Verifica-se, também, que tanto na primeira solução como na segunda, para alguns casos os tempos foram maiores do que outros. Isso se deve ao fato de que os pacotes precisaram passar por uma quantidade maior de nós da rede do que em outras instâncias, para chegar ao seu destino. Apesar de o tempo ser maior, essa são rotas Pareto-ótimas geradas pelo Algoritmo 1.

Portanto, para a rede 3×3 (9 nós), esses resultados não foram conclusivos. Uma possível explicação para isso é que, por ser uma rede relativamente pequena, o número de saltos também o era, e isso fez com que não houvesse um padrão nos tempos retornados.

A seguir, é apresentado os resultados referentes ao roteamento dos fluxos sobre a rede Grade 7×7.

O conjunto mínimo completo de soluções Pareto-ótimas foi gerado a partir da execução do Algoritmo 1. Assim que essas soluções foram recebidas pelo controlador, iniciou-se a fase de construção das regras de roteamento, que são definidas a partir do caminho que um fluxo deve percorrer.

A Tabela 2 apresenta algumas métricas retornadas a partir da execução do Algoritmo 1, onde $|X^*|$ é a cardinalidade do conjunto mínimo completo das soluções Pareto-ótimas, $NumDel$ é o número de soluções deletadas (soluções dominadas), $Sol.Geradas$ é o número total de soluções encontradas pelo algoritmo, It é o número de iterações, $t(s)$ é o tempo de execução (em segundos), Gargalo_0 é o gargalo da primeira solução Pareto-ótima, Salto_0 é o número total de saltos de todos os fluxos da primeira solução Pareto-ótima, Gargalo_f é o gargalo da última solução Pareto-ótima, Salto_f é o número total de saltos de todos os fluxos da última solução Pareto-ótima.

Pode-se observar na Tabela 2 que o número de soluções geradas é superior a três. No entanto, o conjunto de soluções Pareto-ótimas é inferior ou igual a 3. A explicação para isso é que o algoritmo gera algumas soluções dominadas, como pode-se observar na coluna

Tabela 2 – Métricas das soluções geradas pelo Algoritmo 1, numa rede grade 7×7.

Instância	$\|X^*\|$	NumDel	Sol. Geradas	It	t(s)	Gargalo_0	Salto_0	Gargalo_f	Salto_f
1	2	3	5	6	40,462	3	271	2	272
2	2	1	3	4	25,349	3	261	2	263
3	1	3	4	5	30,831	2	256	2	256
4	1	5	6	7	36,373	2	278	2	278
5	1	4	5	6	31,229	2	262	2	262
6	2	2	4	5	26,241	3	272	2	273
7	1	3	4	5	31,389	2	232	2	232
8	3	1	4	5	27,094	2	249	2	285
9	2	2	4	5	26,108	2	255	2	275
10	2	2	4	5	26,484	2	259	2	279

$NumDel$ dessa tabela. Ainda, nota-se que o número de iterações é relativamente baixo, comparado com o número de fluxos (80 fluxos). A ocorrência disso se dá pelo fato de que, devido a distribuição dos fluxos (destinos e origens quaisquer), o roteamento fica espalhado pela rede, o que implica em um gargalo inicialmente bem menor do que 80.

As métricas de vazão e tempo de entrega foram retiradas da execução da troca de pacotes entre os nós. Cada instância retornou uma média de tempo que os pacotes levaram para chegar ao seu destino. Como já dito, foram utilizados 80 fluxos para cada instância. Assim, seria inviável representar os 80 valores para as 10 instâncias. Por esse motivo fez-se a média de tempo de todos os fluxos. Além disso, apresentam-se o tempo mínimo (t_{min}), tempo máximo (t_{max}) e tempo médio (t_{med}), representados em milissegundos (ms), bem como a média de pacotes enviados ($media_p$) (número de pacotes). Todas essas métricas representados na Tabela 3. Os campos da coluna "Última solução Pareto-ótima" que estão em branco são relativos ao fato de que a cardinalidade do conjunto de soluções Pareto-ótimas é $|X^*| = 1$, isso significa que existe apenas uma solução Pareto-ótima para aquela instância.

Tabela 3 – Valores médios dos tempos de entrega dos pacotes e número de pacotes numa rede 7×7

		Primeira solução Pareto-ótima				Última solução Pareto-ótima			
Instância	$\|X^*\|$	t_{max}	t_{min}	t_{med}	$media_p$	$t_{máx}$	t_{min}	t_{med}	$media_p$
1	2	76	25	29	29	89	32	41	29
2	2	76	24	32	27	80	34	38	27
3	1	81	25	28	28	-	-	-	-
4	1	76	24	33	28	-	-	-	-
5	1	63	22	32	27	-	-	-	-
6	2	58	26	33	28	102	32	35	29
7	1	76	26	29	28	-	-	-	-
8	3	43	26	28	26	89	27	52	29
9	2	79	25	34	35	85	43	42	33
10	2	78	20	58	35	80	43	34	35

Uma observação interessante é que, apesar de não muito distantes, os tempos médios da última solução Pareto-ótimas são maiores. Isso se deve ao fato de que nas última solução Pareto-ótima gerada, o comprimento do caminho a ser percorrido por um pacote aumenta (ver Tabela 2), já que o gargalo tem que diminuir. Ou seja, o pacote nessa última solução precisa passar por mais nós para chegar ao seu destino (mais saltos).

No Mininet, a distribuição de pacotes não se dá de forma completamente uniforme. Por vezes, pode acontecer de, na execução de uma instância, haver diferença entre os números de pacotes enviados da primeira para a última solução Pareto-ótima. Isso não é um parâmetro definido, mas sim retornado pela função que envia pacotes de um nó para outro. Isso não significa que os pacotes foram perdidos, mas sim que o número enviado foi diferente para as duas soluções.

O maior tempo médio (t_{med}), observado na Tabela 3, foi o da Instância 8, na última solução Pareto-ótima, sendo o mesmo de 52 milissegundos. A ocorrência disso é justificada pela quantidade de soluções Pareto-ótimas que foram geradas nessa instância ($|X^*| = 3$), ou seja, o comprimento do caminho é maior do que as demais soluções das outras instâncias que obtiveram cardinalidade menores ($1 \leq |X^*| < 3$), pois quanto menor o gargalo de cada solução Pareto-ótima gerada, maior será a quantidade de saltos.

Em relação à primeira solução Pareto-ótima, o tempo gasto para o roteamento dos fluxos (entrega dos pacotes) ficou entre 22 milissegundos e 81 milissegundos. Porém, observando-se a coluna de tempo médio (t_{med}), da Tabela 3, vê-se que o valor máximo é 58 milissegundos. Isso mostra que a maioria dos pacotes foi entregue com o tempo bem menor que 81 milissegundos.

Já em relação à ultima solução Pareto-ótima, observa-se que o tempo gasto para o roteamento dos fluxos ficou entre 27 e 102 milissegundos. É possível notar, que a maior média é 52 milissegundos. Como comentado anteriormente, isso mostra que a maioria dos pacotes foi entregue com um tempo bem inferior a 102 milissegundos (ver Tabela 3.

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Após a emulação das soluções geradas pelo Algoritmo 1 no Mininet, percebeu-se que as amplitudes de tempos da primeira para a última solução Pareto-ótima não foram muito grandes. Mas ainda assim houve uma pequena diferença entre elas, sendo a última um pouco maior. Isso pode ser explicado a partir do número de saltos. Depois da primeira solução, as outras são executadas para um menor gargalo e isso implica um número maior de saltos.

Quanto à perda de pacotes, nenhum teste e configuração de rede passou por esse problema, visto que o protocolo de transporte utilizado foi o TCP/IP. Para um trabalho futuro, pode-se utilizar o protocolo UDP e, assim, essa métrica poderá ser analisada com mais precisão.

Ainda, é possível resolver o Algoritmo 1 numa implementação de uma rede que não seja do tipo *Grade*, mas que seja gerada aleatoriamente. Uma configuração de rede que simula uma rede real pode ser, por exemplo, a rede Barabási-Albert. Uma descrição para esse tipo de rede pode ser encontrada em (BARABÁSI; ALBERT, 1999).

Além de outras configurações de rede, pode-se, ainda, utilizar-se diferentes configurações de distribuição de fluxos. Outra configuração possível, por exemplo, é a de que os fluxos saem de origens quaisquer mas são todos encaminhados para um mesmo nó (mesmo destino).

Neste trabalho a rede analisada não possuía pesos nas arestas. No entanto, o trabalho proposto em (FERNANDES, 2019) considera, também, qualidades e pesos nos enlaces. Isso também poderia ser analisado em um ambiente de emulação com o Mininet.

Portanto, o Mininet mostrou que, para esses testes, o Algoritmo 1 é viável para resolver problemas de roteamento de fluxos em redes, que minimizam o número total de saltos de todos os fluxos e o gargalo da rede. No entanto, como já dito, isso foi concluído a partir dos testes realizados para a configuração de rede do tipo Grade, tendo um total de 49 nós, considerando uma distribuição de fluxos com origens e destinos quaisquer. Para mais afirmações sobre outros cenários, é necessário um conjunto de testes maior onde será considerado outras topologias de rede, bem como diferentes distribuições de fluxos.

REFERÊNCIAS

BARABÁSI, A.-L.; ALBERT, R. Emergence of Scaling in Random Networks. *Science*, v. 286, n. 5439, p. 509–512, 1999. Citado na página 38.

BORNSTEIN, C.; MACULAN, N.; PASCOAL, M.; PINTO, L. L. Multiobjective combinatorial optimization problems with a cost and several bottleneck objective functions: An algorithm with reoptimization. Computers and Operations Research, 2012. Citado na página 18.

CALHEIROS, R. N.; NETTO, M. A.; ROSE, C. A. D.; BUYYA, R. Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience*, Wiley Online Library, v. 43, n. 5, p. 595–612, 2013. Citado na página 21.

COELLO, C. A. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys (CSUR)*, ACM, v. 32, n. 2, p. 109–143, 2000. Citado na página 16.

COMER, D. *Redes de Computadores e Internet - 6.ed.* Bookman Editora, 2016. ISBN 9788582603734. Disponível em: <https://books.google.com.br/books?id=1nwdDAAAQBAJ>. Citado 2 vezes nas páginas 12 e 34.

COMMUNITY, R. S. F. *Ryu SDN Framework*. 2017. Acessado em 17/04/2019. Disponível em: <https://osrg.github.io/ryu/>. Citado na página 22.

COSTA, P. P. d. Teoria dos grafos e suas aplicações. Universidade Estadual Paulista (UNESP), 2011. Citado na página 13.

FERNANDES, K. C. C. Técnicas de otimização multiobjetivo e otimização estocástica para o roteamento de fluxos em redes. Universidade Federal de Goiás, 2019. Citado 6 vezes nas páginas 18, 25, 26, 31, 33 e 38.

FONTES, R.; ROTHENBERG, C. *Emulando Redes sem Fio com Mininet-WiFi*. 1. ed. [S.l.: s.n.], 2019. Citado na página 22.

GÁLVEZ, J. J.; RUIZ, P. M. Efficient rate allocation, routing and channel assignment in wireless mesh networks supporting dynamic traffic flows. *Ad Hoc Networks*, Elsevier, v. 11, n. 6, p. 1765–1781, 2013. Citado na página 18.

GOVINDAN, K.; PAAM, P.; ABTAHI, A.-R. A fuzzy multi-objective optimization model for sustainable reverse logistics network design. *Ecological indicators*, Elsevier, v. 67, p. 753–768, 2016. Citado na página 17.

HANSEN, P. Bicriterion path problems. 1980. Citado na página 17.

HASHIMOTO, K. Técnicas de otimização combinatória multiobjetivo aplicadas na estimação do desempenho elétrico de redes de distribuição. 2004. Citado 2 vezes nas páginas 15 e 16.

JASIKA, N.; ALISPAHIC, N.; ELMA, A.; ILVANA, K.; ELMA, L.; NOSOVIC, N. Dijkstra's shortest path algorithm serial and parallel execution performance analysis. In: IEEE. *2012 proceedings of the 35th international convention MIPRO*. [S.l.], 2012. p. 1811–1815. Citado na página 30.

KETI, F.; ASKAR, S. Emulation of software defined networks using mininet in different simulation environments. In: IEEE. *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*. [S.l.], 2015. p. 205–210. Citado na página 23.

LEHMAN, E.; LEIGHTON, F. T.; MEYER, A. R. Mathematics for computer science. 2010. Citado 2 vezes nas páginas 13 e 14.

MATTOS, D. M. F. Virtualização: Vmware e xen. *Grupo de Teleinformática e Automação da UFRJ*, p. 13, 2008. Citado na página 21.

MATTOS, D. M. F. *Virtualização: VMWare e Xen*. 2008. Acessado em 22/04/2019. Disponível em: <https://www.gta.ufrj.br/grad/09_1/versao-final/virtualizacao/index.html>. Citado na página 21.

MCGREGOR, I. Equipment interface: the relationship between simulation and emulation. In: WINTER SIMULATION CONFERENCE. *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*. [S.l.], 2002. p. 1683–1688. Citado na página 21.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 2, p. 69–74, 2008. Citado na página 22.

MELLO, M. O. M. C. Roteamento em redes em malha sem fio com balanceamento de carga e caminhos mais curtos. Universidade Federal de Goiás, 2014. Citado 3 vezes nas páginas 10, 12 e 18.

OLIVEIRA, R. L. S. D.; SCHWEITZER, C. M.; SHINODA, A. A.; PRETE, L. R. Using mininet for emulation and prototyping software-defined networks. p. 1–6, 2014. Citado 3 vezes nas páginas 22, 23 e 24.

PARETO, V. *Manual of political economy: a critical and variorum edition*. [S.l.]: OUP Oxford, 2014. Citado na página 15.

PINTO, L. de L.; BORNSTEIN, C.; MACULAN, N. The tricriterion shortest path problem with at least two bottleneck objective functions. European Journal of Operational Research, 2009. Citado na página 17.

PINTO, L. de L.; FERNANDES, K. C. C. Um algoritmo exato para um problema de programação inteira biobjetivo. 2016. Citado 2 vezes nas páginas 10 e 18.

PINTO, L. L.; FERNANDES, K. C.; CARDOSO, K. V.; MACULAN, N. An exact and polynomial approach for a bi-objective integer programming problem regarding network flow routing. *Computers & Operations Research*, Elsevier, v. 106, p. 28–35, 2019. Citado 3 vezes nas páginas 10, 18 e 33.

PINTO, L. L.; PASCOAL, M. M. On algorithms for the tricriteria shortest path problem with two bottleneck objective functions. *Computers & Operations Research*, Elsevier, v. 37, n. 10, p. 1774–1779, 2010. Citado 2 vezes nas páginas 17 e 18.

RAUPP, F. M. P. *Um método heurístico para o problema de escalonamento multiobjetivo em vários ambientes de máquinas*. Tese (Doutorado) — PUC-Rio, 2012. Citado 2 vezes nas páginas 14 e 15.

SALVADOR, R. *Determinação da melhor configuração de ocupação de cargas para um sistema de transporte de cargas de modo a minimizar o custo e a pegada de carbono*. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2017. Citado na página 17.

SCHEINERMAN, E. R. *Matemática Discreta uma Introdução*. [S.l.]: Cengage Learning Edições Ltda, 2011. Citado na página 14.

TANENBAUM, A. S. *Redes de Computadores*. 5. ed. [S.l.: s.n.], 2011. Citado na página 12.

TEAM, M. *Mininet*. 2018. Accessado em: 24/03/2019. Disponível em: <http://mininet.org>. Citado 3 vezes nas páginas 10, 21 e 31.

TEIXEIRA, R. de A. Treinamento de redes neurais artificiais através de otimização multi-objetivo: Uma nova abordagem para o equilíbrio entre a polarização e a variância. UFMG, 2001. Citado 3 vezes nas páginas 14, 17 e 25.

WANG, S.-Y.; CHOU, C.-L.; YANG, C.-M. et al. Estinet openflow network simulator and emulator. *IEEE Communications Magazine*, v. 51, n. 9, p. 110–117, 2013. Citado na página 20.

XIE, S.; WANG, Y. Construction of tree network with limited delivery latency in homogeneous wireless sensor networks. *Wireless personal communications*, Springer, v. 78, n. 1, p. 231–246, 2014. Citado na página 23.