Fernando Ávila Fossi Silveira

Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego

Vitória-ES

2017

Fernando Ávila Fossi Silveira

Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego

Dissertação de mestrado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do título de Mestre em Informática. Área de concentração: Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Departamento de Informática

Programa de Pós-Graduação em Informática

Orientador: Rodolfo da Silva Villaça

Coorientador: Renato Elias Nunes de Moraes

Vitória-ES

2017

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)
Sandra Mara Borges Campos – CRB-6 ES-000593/O

Silveira, Fernando Ávila Fossi, 1989-
S587m Minimização do conjunto de observadores para geração de matrizes de tráfego / Fernando Ávila Fossi Silveira. – 2017.
66 f. : il.

Orientador: Rodolfo da Silva Villaça.
Coorientador: Renato Elias Nunes de Moraes.
Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Algoritmos. 2. Otimização combinatória. 3. Redes de computadores – Gerência. 4. Matrizes de trafego (Computação). 5. Algoritmos de processamento de cadeia de dados. I. Villaça, Rodolfo da Silva. II. Moraes, Renato Elias Nunes de. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego

Fernando Ávila Fossi Silveira

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 07 de agosto de 2017:

Prof. Dr Rodolfo da Silva Villaca
Orientador

Prof. Dr. Renato Elias Nunes de Moraes
Coorientador

Prof. Dr. Eduardo Zambon
Membro Interno

Prof. Dr. Fabio Luciano Verdi
Membro Externo

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 07 de agosto de 2017.

Resumo

A sociedade atual é dependente de redes de computadores, portanto gerenciar essas redes é tarefa de fundamental importância. Para que se possa gerenciar redes de computadores é necessário que se conheça como a informação trafega por elas. Uma das maneiras de se obter esse conhecimento é através da geração das matrizes de tráfego, que indicam o volume de informações trocadas entre cada par de dispositivos da rede. Para gerar essas matrizes é necessária a instalação de observadores afim de medir o tráfego nos enlaces, entretanto essa operação é muito custosa. Nos últimos anos, algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas permitiram o monitoramento do tráfego a um baixo custo computacional. A manipulação de estruturas de dados probabilísticas nos observadores pode ser feita de forma rápida e com pouco espaço de memória. Entretanto, mesmo com estruturas de dados probabilísticas, ainda é necessária a instalação de observadores em todos os dispositivos da rede monitorada para geração das matrizes de tráfego, o que incorre em altos custos de implantação. Nesse contexto, esta dissertação propõe e define o problema de Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego (MCO-MT) que consiste em minimizar a quantidade de observadores instalados na rede, ou seja, minimizar o tamanho do conjunto de observadores necessários para gerar matrizes de tráfego utilizando estruturas de dados probabilísticas. O problema é modelado e resolvido como um problema de Cobertura de Conjuntos. Além da definição do MCO-MT, este trabalho propõe: i) o desenvolvimento de uma ferramenta, denominada BitMatrix, para simulação e validação da geração de matrizes de tráfego utilizando algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas, ii) dois algoritmos para resolver o MCO-MT: uma heurística gulosa e uma heurística gulosa aleatória combinada com uma busca local para formar um algoritmo GRASP e iii) um algoritmo de processamento de cadeia de dados baseado em estrutura de dados probabilísticas para gerar matrizes de tráfego a partir de um conjunto reduzido de observadores. Extensos experimentos computacionais são apresentados para validar a eficácia das soluções propostas.

Palavras-chave: Matrizes de Tráfego. Algoritmos de Processamento de Cadeia de Dados. Otimização Combinatória

Abstract

Actual society rely on computer networks, so the management of these networks is an essential task. In order to manage a computer network it is necessary to know how information travels through it. A way to obtain this knowledge is by generating traffic matrices, which indicates the traffic volume exchanged between each pair of devices in the network. To generate these matrices is necessary to install observers in order to measure the traffic in the links, however this is a very costly operation. In recent years, data streaming algorithms based on probabilistic data structures have enabled traffic monitoring at a low computational cost. The manipulation of probabilistic data structures in observers can be done quickly and with little memory usage. However, even with probabilistic data structures, it is still necessary to install observers on all the devices to be monitored in the network in order to generate the traffic matrices, which leads to high implementation costs. In this context, this master's thesis proposes and defines the problem of Minimizing the Set of Observers for Generating Traffic Matrices (MCO-MT), which consists of minimizing the number of observers installed in the network, that is, minimizing the size of the set of observers required in generating traffic matrices using probabilistic data structures. The problem is modeled and solved as a Minimum Set Covering Problem. In addition to the definition of MCO-MT, this work proposes: i) the development of a tool, called BitMatrix, for simulating and validating the generation of traffic matrices using data streaming algorithms based on probabilistic data structures; ii) two algorithms to solve the MCO-MT: a greedy heuristic and a greedy randomized heuristic combined with a local search to create a Greedy Randominzed Adpatative Search Procedure (GRASP), and iii) an data streaming algorithm to generate traffic matrices from a reduced set of observers. Extensive computational experiments are presented to validate the effectiveness of the proposed solutions.

Keywords: Traffic Matrices. Data Streaming Algorithms. Combinatorial Optimization

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AS	Autonomous System
CAIDA	Center for Applied Internet Data Analysis
CDF	Cumulative Distribution Function
DoS	Denial of Services
GRASP	Greed Randomized Adaptative Search Procedure
IS-IS	Intermediate System to Intermediate System
LCR	Lista de Candidatos Restrita
MSCP	Minimum Set Cover Problem
PoP	Point of Presence
OSPF	Open Shortest-Path First
SNMP	Simple Network Management Protocol
VoIP	Voice over IP

Sumário

1 Introdução

O tráfego de dados na Internet vive em constante mudança devido ao grande volume de informações gerado e acessado diariamente. Tendências dessas mudanças podem ser citadas como, por exemplo, a popularização de serviços como *streaming* de vídeos, telefonia na internet (*Voice over IP* - VoIP) e jogos *online*. Tais conteúdos requerem alto índice de Qualidade de Serviço (*Quality of Service* - QoS), pois são intolerantes à perda de pacotes. Por exemplo, a perda excessiva de pacotes gera travamentos durante a execução de um vídeo. Além disso, o aumento da capacidade computacional dos dispositivos móveis e a crescente velocidade das conexões móveis torna evidente que a maior parte do tráfego no futuro será gerado por esses dispositivos que não param de se multiplicar (1).

Essas tendências fazem com que o entendimento da composição e do comportamento do tráfego na Internet seja de vital importância para a sua operação contínua, pois permite a execução de várias atividades de gerenciamento de redes como, planejamento e provisionamento de capacidade, engenharia de tráfego, diagnóstico de falhas, performance de aplicações, detecção de anomalias e apreçamento (2). O gerenciamento de redes permite aos seus administradores dimensionar a capacidade necessária para o tráfego corrente, realizar a manutenção contínua da rede e preparar sua infraestrutura para as novas tendências. Para realizar o gerenciamento de uma rede, os seus administradores precisam ter, entre outras coisas, o conhecimento do tráfego entre os pares ingresso e egresso, isto é, necessita-se da matriz de tráfego da rede (3). Uma matriz de tráfego é uma representação abstrata do volume de dados trafegados entre conjuntos de pares ingresso e egresso em uma rede, normalmente medidos em quantidade de *bytes* ou pacotes.

Matrizes de tráfego são muito importantes para os administradores de rede. De acordo com Tune e Roughan(4), se a matriz de tráfego é conhecida, então, munido com a topologia e as informações de roteamento, o administrador sabe exatamente o que está acontecendo com a rede, facilitando assim suas tarefas de tarefas de operação e manutenção. Por outro lado, se a matriz de tráfego não é conhecida, então o administrador está às cegas e falhas repentinas podem assolar a rede, reduzindo seu desempenho. Congestionamentos se tornam frequentes e mudanças repentinas no tráfego podem causar perdas de informações. Por exemplo, um aumento repentino no tráfego entre um par ingresso e egresso pode indicar uma anomalia no tráfego decorrente de uma ataque de negação de serviços (*Denial of Services* - DoS).

Apesar de sua importância, gerar matrizes de tráfego não é uma tarefa fácil. Os desafios estão associados à necessidade de se conhecer exatamente o volume de dados trafegados por cada dispositivo (por exemplo, *switches* ou roteadores) ou em cada *link*

na rede. A geração de matrizes de tráfego depende, geralmente, que os dispositivos de rede realizem uma coleta minuciosa e o armazenamento de grandes volumes de informações sobre os dados trafegados. Os dispositivos que realizam essas tarefas de coleta e armazenamento são chamados nesta dissertação de *observadores*. Um problema que surge com esta abordagem é que esses observadores possuem fortes restrições de capacidade de processamento e armazenamento. Além disso, a grande quantidade de informação coletada deve ser enviada, posteriormente, para um local centralizado para análise: a estação de monitoramento de rede. Dessa maneira, esta abordagem ainda gera uma quantidade significativa de tráfego que compartilha a mesma infra-estrutura de rede com serviços de usuário. Do ponto de vista desses serviços, esse tráfego é uma sobrecarga, uma vez que ele interfere na disponibilidade de largura de banda para os dados. Ou seja, a própria atividade de monitoramento interfere no tráfego de rede.

O custo associado à coleta minuciosa de informações na rede pode ser diminuído com a utilização de algoritmos de processamento de cadeia de dados baseadas em estruturas de dados probabilísticas (5, 6, 7, 8, 9, 10). Essas estruturas de dados permitem estimar estatísticas sobre diversas métricas e possuem um forte compromisso entre a precisão das medidas obtidas e a quantidade de recursos de memória necessária para o armazenamento dessas estruturas (10). Desta maneira, a ideia por trás destes algoritmos de processamento de cadeia de dados está em que os observadores instanciem uma dessas estruturas de dados para guardar informações sobre os dados trafegados na rede, o que reduz a complexidade de armazenamento nos observadores. Em posse dessas estruturas de dados com os dados do tráfego, a estação de monitoramento é capaz de estimar a matriz de tráfego da rede.

Apesar da redução do custo na geração de matrizes de tráfego alcançada com a utilização de algoritmos de processamento de cadeia de dados, para sua utilização é necessário que se instale muitos observadores na rede. Desta maneira, além da sobrecarga de tráfego já discutida, incorrem também custos fixos de implantação, tais como, custo de hardware/software e custo de manutenção. Esses custos ocorrem porque nem todos os observadores de rede já instalados estão aptos a armazenar e instanciar as estruturas de dados probabilísticas necessárias. Portanto, para a geração das matrizes de tráfego em uma rede real é importante que se minimize a quantidade de observadores instalados na rede.

Desta maneira, com o objetivo principal de diminuir o custo da geração de matrizes de tráfego, esta dissertação apresenta a proposta da combinação de abordagens que utilizam algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas para monitoramento de tráfego, com abordagens de redução da quantidade total de observadores implantados na rede através da identificação de locais estratégicos para sua instalação (11, 12, 13, 14, 15, 16).

Neste contexto, a contribuição principal desta dissertação é a definição do problema

de otimização combinatória denominado como o Problema de Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego (MCO-MT) que consiste em minimizar a quantidade de observadores instalados na rede, ou seja, minimizar o tamanho do conjunto de observadores necessários para gerar matrizes de tráfego utilizando estruturas de dados probabilísticas.

O MCO-MT pode ser modelado como um problema de Minimização de Cobertura de Conjuntos (*Minimum Set Cover Problem* - MSCP), que é um importante problema da classe NP-Completo (17). Para resolver esse problema esta dissertação apresenta dois algoritmos para revolver o MCO-MT: um algoritmo guloso e um algoritmo GRASP (*Greedy Randomized Adaptative Search Procedure*) (18, 19), denominados nesta dissertação, a partir dos sobrenomes dos autores que propuseram os algoritmos, de SMV (20) e GSMV (21), respectivamente.

Esta dissertação propõe ainda um novo algoritmo de processamento de cadeia de dados baseados em estruturas de dados probabilísticas para geração de matrizes de tráfego, denominado GeraMatriz e baseado no trabalho de Zhao et al.(22). Esse algoritmo é capaz de gerar matrizes de tráfego a partir de um conjunto reduzido de dispositivos observadores encontrados ao se resolver o MCO-MT, isto é, o algoritmo GeraMatriz é capaz de gerar matrizes de tráfego sem a necessidade de se instalar observadores em todos os dispositivos da rede. Para esse algoritmo, foi proposta a utilização de uma estrutura de dados *counter-matrix*. O *counter-matrix* é basicamente uma matriz de contadores instanciada em um observador afim de monitorar um caminho na rede. Cada elemento da *counter-matrix* indica o volume de dados entre um par ingresso e egresso no caminho monitorado. Em posse dessas estruturas de dados e dos caminhos utilizados pela rede, a estação de monitoramento é capaz de gerar as matrizes de tráfego. Os resultados apresentados nesta dissertação permitem afirmar que com essa estratégia é possível implantar observadores em apenas uma fração dos dispositivos da rede sem perda de precisão na matriz de tráfego gerada.

Para realizar os testes com as matrizes de tráfego geradas pelo algoritmo GeraMatriz, esta dissertação propõe a construção de uma ferramenta de simulação computacional, denominada BitMatrix, para geração da matriz de tráfego estimada de uma rede utilizando algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas. Mais especificamente, a BitMatrix implementa o *Bitmap Based Algorithm* (22) e o algoritmo GeraMatriz. A ferramenta permite comparar a matriz estimada com a matriz real para diversas redes e fluxos de dados. Esta comparação é importante para o entendimento e validação dos algoritmos de processamento de cadeia de dados.

Esta dissertação está organizada da seguinte maneira:

- O Capítulo 2 discute os trabalhos relacionados, analisando-os a fim de reforçar a justificativa e a relevância do trabalho que está sendo proposto, além de apresentar

características e métodos que serviram de fonte de inspiração para o presente trabalho.

- O Capítulo 3 apresenta o modelo de rede utilizado ao longo da dissertação e formaliza o problema da geração de matrizes de tráfego;
- O Capítulo 4 descreve o *Bitmap Based Algorithm* para geração de matrizes de tráfego estimadas. Em seguida é apresentada a ferramenta de simulação BitMatrix e ao fim do capítulo são apresentados os resultados e conclusões obtidos com a ferramenta.
- O Capítulo 5 formaliza o problema de Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego e apresenta o algoritmo GeraMatriz, para gerar matrizes de tráfego a partir de um conjunto reduzido de observadores. Na sequencia, a ferramenta BitMatrix é utilizada para gerar e comparar os resultados obtidos pelos algoritmos GeraMatriz e *Bitmap Based Algorithm*. As conclusões são apresentadas ao fim do capítulo.
- O Capítulo 6 enuncia o problema de Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego como um problema de Minimização de Cobertura de Conjuntos e apresenta os dois algoritmos propostos que o resolvem: um algoritmo guloso, denominado SMV, e a combinação de um algoritmo guloso aleatório com uma busca local para formar um algoritmo GRASP, denominado GSMV. O capítulo é finalizado com apresentação de extensos resultados computacionais comparando os resultados obtidos pelos algoritmos SMV e GSMV.
- O Capítulo 7 traz conclusões gerais e perspectivas de trabalhos futuros.

Destaca-se que todas as contribuições mencionadas nesta dissertação se encontram publicadas nos trabalhos dos autores: Silveira, Moraes e Villaça(20), Nogueira et al.(23) e Cardoso et al.(21).

2 Trabalhos Relacionados

A abordagem apresentada nesta dissertação relaciona o problema da geração de matrizes de tráfego (24, 22, 25, 26) com o problema de localização de observadores de tráfego na rede (11, 12, 13, 15, 14, 16).

Segundo Tune e Roughan(4) as estratégias para construir matrizes de tráfego podem ser classificadas em dois tipos: medições indiretas ou diretas. Conceitualmente, medições indiretas inferem a matriz de tráfego combinando os dados medidos através do *Simple Network Management Protocol* (SNMP) (27) com as informações topológicas e informações de roteamento da rede. Já as estratégias de medição direta não necessitam de modelos nem de dados externos, pois realizam a medição observando diretamente o tráfego em múltiplos pontos da rede. De modo geral, estratégias de medição diretas são mais precisas do que as estratégias indiretas. Técnicas baseadas na utilização de algoritmos de processamento de cadeia de dados são considerados técnicas de medição direta. A seguir algumas técnicas para se gerar matrizes de tráfego diretamente.

As informações coletadas pelas técnicas de medição direta são observadas através de *packet traces*. Um *packet trace* representa uma coleção de cabeçalhos de pacotes e *timestamps*. Coletar esses *traces* é uma tarefa bastante custosa por dois motivos: primeiro, necessita-se de dispositivos específicos para a coleta de dados e, segundo, a quantidade de informações geradas é muito grande, podendo sobrecarregar os dispositivos de monitoramento. As alternativas mais utilizadas para diminuir a quantidade de informações geradas é a de agregação dos pacotes em fluxos, onde os pacotes são agregados de acordo com uma chave, por exemplo, a 5-tupla formada pelos endereços de origem e destino, portas de origem e destino e o número do protocolo de comunicação (25, 4).

Técnicas de amostragem de pacotes podem ser utilizadas para reduzir ainda mais a quantidade de registros de fluxos coletados. Nelas, os pacotes de ingresso são amostrados baseados em uma regra predeterminada, como no NetFlow (28) e sFlow (29). Durante a amostragem é mais provável que se escolha um pacote de um fluxo maior, privilegiando assim esses fluxos e, consequentemente, degradando a qualidade dos resultados. Nos trabalhos dos autores Duffield, Lund e Thorup(30) e Duffield, Lund e Thorup(31) são propostos métodos mais eficazes de se amostrar fluxos com menor perda de qualidade nos resultados.

Outro problema da medição direta, além da grande quantidade de dados, é a múltipla contagem de fluxos. Nesse caso, um único fluxo pode ser amostrado mais de uma vez por vários elementos da rede aumentando o erro de amostragem. O trabalho de (32) utiliza-se a pseudo aleatoriedade das funções de *hashing* para rastrear os fluxos na rede e

assim resolver a múltipla contagem dos fluxos.

Independente da estratégia de amostragem adotada, deve-se lembrar de que a amostragem é, em sua essência, um processo com perdas. A perda de informações se traduz em erros ou ruídos nos dados. Para uma melhor utilização dos dados, a magnitude desses erros deve ser estimada. Conforme dito no Capítulo 1, essa dificuldade pode ser superada através do uso de algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas. Esses algoritmos vêm sendo estudados para serem utilizados em redes de computadores, por exemplo, com a criação de ferramentas para monitoramento de rede (10, 33) e a proposta de uma única estrutura de dados probabilística capaz de armazenar informações para várias métricas (34). Especificamente para geração de matrizes de tráfego, Zhao et al.(22) propôs o *Bitmap Based Algorithm* para geração de matrizes de tráfego baseado na utilização de estruturas de dados do tipo *bitmap*, onde a precisão da matriz gerada depende de dois parâmetros dos *bitmaps*: tamanho e limite de acessos.

Nesse contexto, esta dissertação contribui ao disponibilizar a implementação de uma ferramenta que permite simular a geração de matrizes de tráfego utilizando o algoritmo de processamento de cadeia de dados *Bitmap Based Algorithm*, permitindo o ajuste fino dos parâmetros tamanho e limite de acessos dos *bitmaps* utilizados pelo algoritmo. As matrizes de tráfego estimadas serão comparadas com matrizes reais, permitindo ao administrador da rede alcançar a maior precisão possível com o melhor valor de ajuste dos parâmetros.

Conforme dito no Capítulo 1, mesmo com a utilização de algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas, é importante que se minimize a quantidade de observadores instalados na rede. Identificar os locais estratégicos para os observadores de tráfego é um problema difícil já conhecido na literatura (11, 12, 13). Chaudet et al.(11) apresentam modelos de otimização combinatória e uma modelagem inteira para o problema. Em especial apresenta uma modelagem do problema como um *Minimum Set Cover* onde é considerado o conjunto do tráfego de dados e sua disposição pelos *links* da rede. Suh et al.(13) tratam o problema de posicionar os observadores e definir taxas de amostragem de pacotes a fim de maximizar a fração de fluxos a ser amostrado em função dos custos de operação e instalação. Cantieni et al.(12) também determinam um conjunto reduzido de observadores e suas taxas de amostragem a fim de obter uma medição de alta precisão com baixo consumo de recursos. Citando trabalhos um pouco mais recentes, Raza et al.(15) propõe fazer o roteamento da rede para utilizar observadores já instalados ao invés de escolher o posicionamento desses dispositivos, enquanto Huang et al.(14) propõe que ambos posicionamento de dispositivos e o roteamento da rede devem ser otimizados.

Neste contexto, esta dissertação propõe uma nova abordagem para geração de matrizes de tráfego, que consiste em utilizar algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas em um conjunto reduzido de observadores.

Desta forma, o problema a ser resolvido consiste em selecionar um número reduzido de nós observadores e suas posições (problema de seleção e posicionamento de observadores) e gerar a matriz de tráfego da rede usando os algoritmos de processamento cadeia de dados baseados em estruturas de dados probabilísticas instaladas nos observadores, sem que a precisão da matriz seja prejudicada (problema da estimativa de matriz de tráfego de uma rede).

3 Conceitos e Definições

Este capítulo define formalmente os conceitos e definições necessários à construção dos algoritmos e estruturas usadas na sequência dos capítulos seguintes. Especificamente define o modelo das redes e das matrizes de tráfego utilizadas nos capítulos seguintes.

O tráfego de dados dentro de uma estrutura de rede de computadores é resumido a seguir. A unidade básica de informação trafegando na rede é chamada de pacote. Todo pacote se origina em um ponto de ingresso e é entregue a um ponto de egresso. Normalmente, esses pontos são identificados por dispositivos de rede como *switches* ou roteadores, porém, um ponto pode representar um aglomerado de dispositivos como em um Ponto de Presença (*Point of Presence* - PoP) ou Sistema Autônomo (*Autonomous System* - AS). O tráfego viaja por um conjunto de *links* estabelecido entre um conjunto de ingressos e egressos. Os *links* que conectam esses dispositivos definem a topologia da rede, sendo que os caminhos escolhidos pelo tráfego determinam o roteamento. Dada a ideia geral do tráfego de dados em uma rede, na sequencia serão apresentas as notações utilizadas ao longo desta dissertação.

Logicamente, uma topologia de rede pode ser definida como um grafo conexo não direcionado $G = (V,A)$ onde os pontos (dispositivos) são representados pelo conjunto de vértices V e numerados de 0 a $|V|-1$. O conjunto de arestas A contém pares não ordenados de vértices (u,v), com $u,v \in V$, representando o *link* existente entre u e v. Quando o *link* (u,v) está presente em A, significa que pacotes podem ser enviados diretamente de u para v, ou de v para u, através do *link* (u,v). No contexto desta dissertação, a cada aresta (u, v) é associado um custo unitário.

Os vértices podem ser divididos no conjunto de vértices ingressos $V^I \subseteq V$ e no conjunto de vértices egressos $V^E \subseteq V$. Um vértice pode pertencer aos dois conjuntos. Todo pacote, representado como $p_{[v_i,v_e]}$, está associado a um par de nós $[v_i, v_e]$ tal que $v_i \in V^I$ é o vértice de ingresso e $v_e \in V^E$ é o vértice de egresso do pacote. Um pacote então viajará pela rede partindo do seu vértice de ingresso v_i até o seu vértice de egresso v_e passando por uma sequencia de arestas dentro do conjunto A. Essa sequencia de arestas é chamada de caminho. Como uma aresta é representada pelo par de vértices (u,v) pode-se representar o caminho $C_{[v_i,v_e]}$ pela sequencia dos vértices que formam as arestas que formam o caminho. Uma característica importante é que um caminho não contém ciclos, isto é, um mesmo vértice v não aparece duas vezes em um caminho.

Como exemplo, a Figura 1 mostra uma topologia com os vértices $V = \{0,1,...,7\}$ conectados pelas arestas exibidas como linhas entre eles. Um pacote $p_{[0,7]}$ passa pela sequencia de arestas $\{(0,3),(3,4),(4,7)\}$, consequentemente, o caminho do pacote é dado

pela sequência de vértices $C_{[0,7]} = \{0, 3, 4, 7\}$. É importante ressaltar que podem existir vértices que não são ingresso nem egresso de nenhum pacote, mas que participam de seus caminhos, isto é, podem existir vértices $v \in V$ tal que $v \notin V^I \cup V^E$ e $v \in C_{[v_i,v_e]}$. Por exemplo, na Figura 1, se o conjunto de vértices de ingresso e egresso forem $V^I = \{0, 1, 2\}$ e $V^E = \{5, 6, 7\}$, respectivamente, os nós 3 e 4 podem não ser ingresso nem egresso de nenhum pacote, mas participam de todos os caminhos da rede.

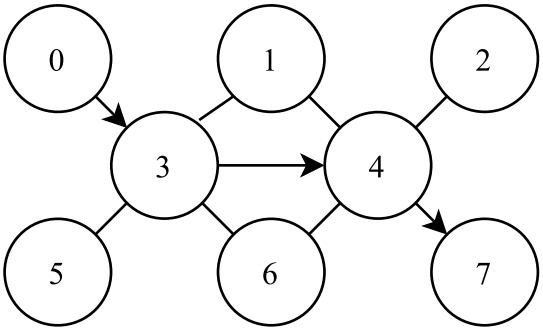


Figura 1 – Exemplo de um pacote trafegando em uma topologia de rede.

Entre um par de vértices ingresso-egresso $[v_i, v_e]$ podem existir vários caminhos. Geralmente, esses caminhos podem ser inferidos a partir da tabela de roteamento e dependem dos protocolos de roteamento utilizados pela rede, por exemplo o *Open Shortest-Path First* (OSPF) (35). Dessa maneira, nesta dissertação, o caminho $C_{[v_i,v_e]}$ é considerado como o único caminho entre o par $[v_i, v_e]$ e definido por algum algoritmo como, por exemplo, o algoritmo de Dijkstra(36). Desta maneira, o conjunto dos caminhos entre todos os pares ingresso-egresso é dado por $\mathcal{C} = \{C_1, C_2, \ldots, C_c\}$ tal que c é a quantidade de pares de nós ingresso-egresso na rede e $c = |V^I \times V^E| - |V^I \cap V^E|$.

Dados os modelos de topologia e tráfego em uma rede, várias matrizes de tráfego podem ser construídas em um tempo T de observação do tráfego de pacotes pela topologia da rede. De acordo com Tune e Roughan(4), uma matriz de tráfego pode ser representada como uma hiper matriz $M(\delta t)$, onde cada elemento $M_{v_i,v_e}(\delta t)$ representa o volume de dados entre o vértice de ingresso v_i e o vértice de egresso v_e no intervalo de tempo $\delta t \subset T$. Nesta dissertação, o volume de dados $M_{v_i,v_e}(\delta t)$ foi medido como a quantidade de pacotes trafegados entre v_i e v_e no intervalo de tempo $\delta t = [t, t + \Delta t) \subset T$, onde t é o momento inicial de medição de $M(\delta t)$ e $t + \Delta t$ o momento final de medição de $M(\delta t)$. Para simplificar, sempre que apenas a noção estrutural da matriz for necessária, a notação tempo será omitida.

A Figura 2 mostra a estrutura mais genérica de uma matriz de tráfego, quando todos os vértices são ingressos e egressos, isto é, $V^I = V^E = V$. As linhas representam os ingressos, com tamanho $|V^I|$, e as colunas representam os egressos, com tamanho $|V^E|$, sendo assim, o tamanho da matriz é dado por $|V^I \times V^E|$, no caso em que todos os vértices são ingresso e egresso o tamanho da matriz é dado por $|V \times V|$.

$$M = \begin{bmatrix} M_{0,0} & \dots & M_{0,v} & \dots & M_{0,|V|-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ M_{v,0} & \dots & M_{v,v} & \dots & M_{v,|V|-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ M_{|V|-1,0} & \dots & M_{|V-1|,v} & \dots & M_{|V|-1,|V|-1} \end{bmatrix}$$

Figura 2 – Estrutura geral de uma matriz de tráfego com a notação tempo omitida

A estrutura da matriz de tráfego utilizada nesta dissertação leva em conta as limitações do *Bitmap Based Algorithm* (22), que foi o algoritmo escolhido para validar a geração de matrizes de tráfego utilizando processamento de cadeia de dados baseados em estruturas de tráfego probabilísticas. As limitações acontecem quando existe pelo menos um vértice $v_c \in V^I \cap V^E$, isto é, quando $|V^I \cap V^E| > 0$. A primeira limitação é que o algoritmo não distingue a direção dos pacotes. No algoritmo os pacotes $p_{[v_i,v_c]}$ e $p_{[v_c,v_i]}$ são contabilizados no mesmo elemento da matriz, por exemplo, M_{v_i,v_c}. Além disso, quando um vértice v_c, tal que $v_c \neq v_i$ e $v_c \neq v_e$, participa do caminho $C_{[v_i,v_e]}$, além de contabilizar o tráfego $M_{[v_i,v_e]}$ o algoritmo contabiliza também os tráfegos $M_{[v_i,v_c]}$ e $M_{[v_c,v_e]}$. O *Bitmap Based Algorithm* é discutido posteriormente no Capítulo 4.

$$M = \begin{bmatrix} M_{0,0} & M_{0,1} & M_{0,2} & (M_{0,3}+1) & (M_{0,4}+1) & M_{0,5} & M_{0,6} & (M_{0,7}+1) \\ 0 & M_{1,1} & M_{1,2} & M_{1,3} & M_{1,4} & M_{1,5} & M_{1,6} & M_{1,7} \\ 0 & 0 & M_{2,2} & M_{2,3} & M_{2,4} & M_{2,5} & M_{2,6} & M_{2,7} \\ 0 & 0 & 0 & M_{3,3} & (M_{3,4}+1) & M_{3,5} & M_{3,6} & (M_{3,7}+1) \\ 0 & 0 & 0 & 0 & M_{4,4} & M_{4,5} & M_{4,6} & (M_{4,7}+1) \\ 0 & 0 & 0 & 0 & 0 & M_{5,5} & M_{5,6} & M_{5,7} \\ 0 & 0 & 0 & 0 & 0 & 0 & M_{6,6} & M_{6,7} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & M_{7,7} \end{bmatrix}$$

Figura 3 – Exemplo da estrutura de uma matriz de tráfego após a contagem de um pacote

Dessa maneira, para que seja possível comparar as matrizes de tráfego geradas, nesta dissertação a contagem da quantidade de pacotes em cada elemento da matriz de tráfego adota a seguinte estratégia: os elementos da matriz são contabilizados apenas nos elementos da matriz triangular superior da matriz de tráfego. A decisão pela matriz triangular superior é arbitraria. Cada pacote $p_{[v_i,v_e]}$ que trafega pelo caminho $C_{[v_i,v_e]}$ é contabilizado para cada par ingresso e egresso formado neste caminho. A formação dos pares é feita através da combinação simples dos vértices ingresso e egresso contidos no caminho. Como exemplo a Figura 3 mostra a alteração na matriz de tráfego de tamanho $|V \times V|$ da topologia da Figura 1 após a contagem do pacote $p_{[0,7]}$.

4 Geração de Matrizes de Tráfego utilizando *Bitmaps*

Conforme discutido no Capítulo 2, a diminuição do custo de construção de uma matriz de tráfego pode ser alcançada por meio da substituição da coleta minuciosa de informações pela utilização do *Bitmap Based Algorithm*, que é um algoritmo de processamento de cadeia de dados baseado na estrutura de dados probabilística do tipo *bitmap* (22). Um *bitmap* em si não é uma estrutura de dados probabilística. A probabilidade vem da distribuição da cadeia de dados de entrada (37, 38), neste caso, dos pacotes na rede.

No algoritmo, cada vez que um pacote passa por um vértice da rede, um valor inteiro é gerado a partir da aplicação de uma função *hash* simples em alguns campos específicos do cabeçalho do pacote. O *bitmap*, que é basicamente um vetor de *bits*, tem um *bit* marcado na posição indicada pelo valor inteiro resultado da função *hash*. Se todos os vértices da rede aplicam a mesma função *hash* no pacote, as mesmas posições serão marcadas nos *bitmaps* de cada vértice. A partir desses *bitmaps* marcados em cada vértice é possível estimar o volume de tráfego entre pares de vértices da rede. O algoritmo será melhor explicado na Seção 4.1.

A utilização do *Bitmap Based Algorithm* para estimação da matriz de tráfego reduz significativamente o custo da coleta de informações. Entretanto, essa mesma simplificação provoca perdas de precisão na matriz gerada. Imprecisões acontecem, por exemplo, quando ocorrem colisões de *hash* nas marcações dos *bitmaps*. Ajustes na precisão podem ser realizados através de ajustes dos seguintes parâmetros dos *bitmaps*:

- **Tamanho:** Afeta o intervalo em que a função *hash* pode realizar a marcação. Intuitivamente, a precisão da matriz de tráfego estimada deve aumentar proporcionalmente ao tamanho do *bitmap*. Do ponto de vista operacional, o tamanho deve ser pequeno para não ocupar muito espaço de memória volátil dos observadores;
- **Limite de acessos:** Afeta a quantidade de vezes que a marcação acontece em um mesmo *bitmap*. Intuitivamente, a precisão da matriz de tráfego estimada deve aumentar quanto menor for o limite de acessos. Do ponto de vista operacional, o limite de acessos deve ser alto para não ocupar muito espaço de armazenamento dos observadores.

Neste contexto, esta dissertação propõe uma ferramenta de simulação computacional, denominada BitMatrix, que permite:

- a geração da matriz de tráfego estimada de uma rede;
- a comparação da precisão das matrizes de tráfego estimadas com a matriz de tráfego real;
- o ajuste fino dos parâmetros de configuração dos *bitmaps* em busca da melhor precisão;
- a visualização dinâmica do tráfego de dados na rede;

A BitMatrix foi avaliada utilizando *traces* anonimizados de dados de tráfego real da Internet (39) e topologias reais de Sistemas Autônomos (ASes - *Autonomous Systems*) (40), ambos obtidos diretamente do CAIDA (*Center for Applied Internet Data Analysis*). Os resultados mostram a eficácia da ferramenta no auxílio ao ajuste fino dos parâmetros de configuração dos *bitmaps* em função da precisão obtida na estimação das matrizes de tráfego das redes simuladas.

O restante deste capítulo está organizado da seguinte forma:

- a Seção 4.1 descreve a técnica *Bitmap Based Algorithm* utilizada para estimação de matrizes de tráfego a partir dos *bitmaps*;
- a Seção 4.2 apresenta a ferramenta BitMatrix, seus módulos e funcionalidades.
- a Seção 4.3 descreve os testes realizados para a avaliação da ferramenta e apresenta os resultados obtidos;
- a Seção 4.4 fecha o capítulo com conclusões obtidas com a ferramenta e descreve como a ferramenta foi importante para a continuação desta dissertação.

4.1 *Bitmap Based Algorithm*

A arquitetura do sistema para aplicação do algoritmo *Bitmap Based Algorithm* é mostrada no exemplo da Figura 4. No algoritmo, o conjunto de vértices de ingresso e egresso fazem o papel de observadores, isto é, $V^O = V^I \cup V^E$. Na Figura 4 os conjuntos de ingresso e egresso são $V^I = \{I_1, I_2\}$ e $V^E = \{E_1, E_2\}$, respectivamente. Cada observador deve instanciar e gerenciar a estrutura de dados probabilística chamada de *bitmap*. Essa estrutura será melhor explicada na Seção 4.1.1. O algoritmo é dividido em dois módulos de processamento.

No primeiro módulo, denominado de *Online Streaming Module*, a contagem dos pacotes é feita por cada um dos vértices observadores através dos *bitmaps* em uso no observador. Um *bitmap* está em uso quando ele está instanciado na memória volátil de

um vértice observador. Quando o limite de acessos ao *bitmap* é alcançado, o *bitmap* em memória volátil é movido para a memória permanente do observador. O intervalo de tempo em que o *bitmap* ficou em memória é denominado *bitmap epoch* e é importante durante o segundo módulo do *Bitmap Based Algortihm*.

De maneira análoga ao intervalo de tempo de medição da matriz de tráfego, definido no Capítulo 3, o *bitmap epoch* pode ser definido como um intervalo de tempo $\delta e = [e, e+\Delta e]$, sendo o Δe o tamanho do *bitmap epoch*. O *bitmap epoch* varia de acordo com os parâmetros tamanho e limite de acessos do *bitmap* e com a velocidade com a qual chegam dados ao observador. Por causa disso, o *bitmap epoch* não é um parâmetro definido diretamente. Observe que, como o *bitmap epoch* depende da velocidade com que os dados chegam ao observador, os *epochs* entre os observadores não são alinhados, ou seja, os tamanhos dos *bitmap epoch* Δe não são os mesmos entre os observadores e, dessa maneira, a quantidade de *bitmaps* armazenados localmente varia entre os observadores.

Os *bitmaps* permanecem armazenados localmente nos observadores durante um período de tempo definido pelo administrador da rede. Esse período de tempo não interfere na precisão da medição, visto que, uma vez armazenado, o *bitmap* não sofre mais alterações. Em seguida, os *bitmaps* são enviados à Estação de Monitoramento sob demanda. O segundo módulo do algoritmo, denominado *Data Analysis Module* é executado pela Estação de Monitoramento, onda a matriz de tráfego é estimada à partir dos *bitmaps* e seus *epochs*. Na sequencia desta seção, os módulos *Online Streaming Module* e *Data Analysis Module* serão explicados em detalhes.

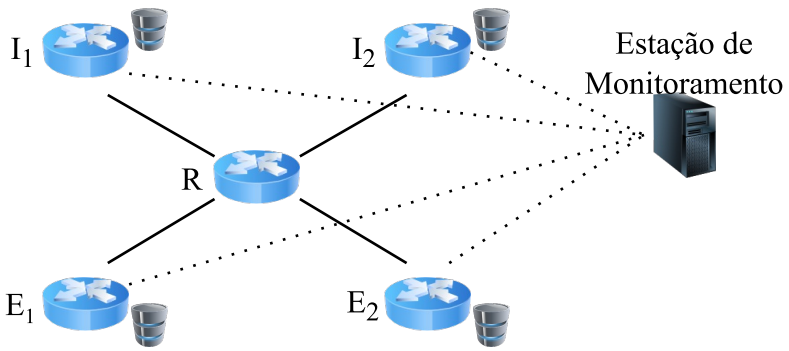


Figura 4 – Arquitetura do sistema na aplicação do algoritmo *Bimap Based Algorithm*

4.1.1 *Online Streaming Module*

Conforme dito na Seção 4.1, cada observador deve instanciar e gerenciar um *bitmap*. Um *bitmap* é basicamente um vetor de *bits*, isto é, cada posição do vetor pode assumir os valores 0 ou 1. O Algoritmo 1 mostra o procedimento de gerenciamento dos *bitmaps*

em cada observador. O algoritmo é executado no momento em que o observador entra em funcionamento e recebe como entrada os dois parâmetros de configuração do *bitmap*:

- *tamanhoBitmap*: indica quantos *bits* o *bitmap* possui, isto é, indica a quantidade máxima de posições no vetor de *bits*;
- *limiteAcessosBitmap*: indica o limite máximo de acessos que podem ser realizados no *bitmap*. É representada pela razão da quantidade atual de acessos em relação ao tamanho do *bitmap*.

Algoritmo 1: *Bitmap Based Algorithm - Online Streaming Module*

Entrada: *tamanhoBitmap* ← tamanho do *bitmap*
Entrada: *limiteAcessosBitmap* ← limiar de número de acessos ao *bitmap*

```
1  início
2      bmp ← NovoBitmap(tamanhoBitmap)
3      numeroAcessos ← 0
4      inicioEpoch ← DataHoraAtual()
5      enquanto observador estiver funcionando faça
6          pkt ← RecebePacote()
7          i ← Hash(φ(pkt))
8          bmp[i] ← 1
9          numeroAcessos ← numeroAcessos + 1
10         se numeroAcessos / tamanhoBitmap ≥ limiteAcessosBitmap então
11             Armazena(bmp, inicioEpoch,DataHoraAtual())
12             bmp ← NovoBitmap(tamanhoBitmap)
13             numeroAcessos ← 0
14             inicioEpoch ← DataHoraAtual()
15         fim se
16     fim enqto
17 fim
```

O Algoritimo 1 inicia na linha 2 onde a função `NovoBitmap` instancia um *bitmap* com vetor de bits de tamanho *tamanhoBitmap*, onde todas as posições são marcadas como 0, em seguida na linha 3 o contador de acessos ao *bitmap* é zerado e na linha 4 a data e hora atuais são salvos como o início do *bitmap epoch* na variável *inicioEpoch*.

O laço da linha 5 até a linha 16 controla os *bitmaps* e executa enquanto o observador estiver em funcionamento na rede. Na linha 6 a função `RecebePacote` espera até que um pacote seja recebido em uma das interfaces do observador. No momento em que um pacote é recebido, a função `RecebePacote` retorna o pacote para a variável *pkt*. Em posse do pacote, na linha 7 é aplicado a função `Hash` no cabeçalho do pacote, definido como a função $\phi(pkt)$, que seleciona os campos: endereços IP de origem e destino, portas de origem e destino, tipo do protocolo de transporte e o primeiro *byte* do campo de dados do pacote. Para a função de *hashing* Zhao et al.(22) falam que devem ser utilizadas funções de

rápida computação e que possam ser implementadas em *hardware* de modo que a operação dos dispositivos de rede não seja afetada. A função `Hash` é implementada utilizando a função MD5 (41) que é uma função conhecida e de rápida computação. O resultado da função `Hash` é um valor inteiro i no intervalo $[0; tamanhoBitmap)$ que é então utilizado na linha 8 para atribuir o valor 1 ao *bit* na posição i do *bitmap*, indicando que a passagem do pacote foi registrada na posição i. Em seguida na linha 9 o contador de acessos ao *bitmap* é incrementado em 1.

Após a marcação é realizado o teste de substituição do *bitmap* na linha 10. Se a razão do $numeroAcessos$ em relação ao $tamanhoBitmap$ atingiu o limiar $limiteAcessosBitmap$, o *bitmap* será armazenado permanentemente em disco na linha 11. A função `Armazena` salva o *bitmap* em disco juntamente com seu *bitmap epoch* que é dado pelas data e hora do início e fim do *epoch*, dados pela variável $inicioEpoch$ pela data e hora em que o *bitmap* foi movido para o disco, respectivamente. Os *bitmaps* permanecem em disco até que sejam solicitados pela estação estação de monitoramento. Conforme dito anteriormente neste na Seção 4.1 este tempo deve ser configurado pelo administrador da rede e deve ser grande o suficiente para que não sobrecarregue os dispositivos da rede.

Após mover o *bitmap* para o disco, na linha 12, a função `NovoBitmap` é chamada para instanciar em memória um novo *bitmap* com tamanho $tamanhoBitmap$ e todas as posições marcadas como 0. Na sequencia o contador de acessos e o início do *epoch* são reinicializados nas linhas 13 e 14, respectivamente.

A solução proposta pelo *Bitmap Based Algorithm* considera que o Algoritmo 1 é executado em todos os observadores da rede que se deseja monitorar. Considera também que os parâmetros de ajuste ($tamanhoBitmap$ e $limiarAcessoBitmap$), e as funções de extração de cabeçalho (ϕ) e *hashing* (`Hash`) são os mesmos em todos os observadores.

4.1.2 *Data Analysis Module*

A ideia geral do *Data Analysis Module* é a seguinte: quando for necessário estimar o volume de tráfego $M_{v_i,v_e}(\delta t)$ entre dois vértices observadores v_i e v_e em um intervalo de tempo δt qualquer, a Estação de Monitoramento recupera o conjunto de *bitmaps* marcados durante o intervalo de tempo δt. Em seguida, os dois conjuntos de *bitmaps* são comparados. Quanto mais *bits* em comum eles tiverem nas mesmas posições, mais pacotes em comum passaram por esses elementos. Se a interseção de *bits* em uma mesma posição entre dois conjuntos de *bitmaps* for pequena, significa que poucos pacotes em comum trafegaram por esses elementos.

O Algoritmo 2 mostra o procedimento executado na Estação de Monitoramento para estimativa de tráfego entre dois observadores. O algoritmo recebe como parâmetro os conjuntos de *bitmaps*, B^{v_i} e B^{v_e} de dois observadores v_i e v_e na rede e o intervalo de

medição δt para o qual se deseja estimar a matriz de tráfego. O retorno do algoritmo é a quantidade de pacotes estimada $M_{v_i,v_e}(\delta t) \in \mathbb{R}$ entre os observadores v_i e v_e no intervalo de medição δt.

Algoritmo 2: *Bitmap Based Algorithm - Data Analysis Module*

Entrada: Conjunto de *bitmaps* B^{v_i} e B^{v_e} dos observadores v_i e v_e
Entrada: Intervalo de tempo de medição δt
Saída: Tráfego estimado $M_{v_i,v_e}(\delta t)$ entre os vértices observadores v_i e v_e no intervalo de medição δt

```
1  início
2      [I^{v_i}, I^{v_e}] ← SelecionaNoIntervalo(B^{v_i},B^{v_e},δt)
3      M_{v_i,v_e}(δt) ← 0
4      para cada bmp^i ∈ I^{v_i} faça
5          para cada bmp^e ∈ I^{v_e} faça
6              se Sobrepoe(bmp^i, bmp^e) então
7                  D_{bmp^i} ← Estima(bmp^i)
8                  D_{bmp^e} ← Estima(bmp^e)
9                  D_{bmp^{i∪e}} ← Estima(bmp^{i∪e})
10                 D_{bmp^{i∩e}} ← D_{bmp^i} + D_{bmp^e} − D_{bmp^{i∪e}}
11                 M_{v_i,v_e}(δt) ← M_{v_i,v_e}(δt) + D_{bmp^{i∩e}}
12             fim se
13         fim para cada
14     fim para cada
15     retorna M_{v_i,v_e}(δt)
16 fim
```

Na linha 2 a função `SelecionaNoIntervalo` seleciona os subconjuntos $I^{v_i} \subseteq B^{v_i}$ e $I^{v_e} \subseteq B^{v_e}$ de *bitmaps* que estão no intervalo de medição δt. Para que um *bitmap* se encontre dentro do intervalo de medição δt, basta que seu *bitmap epoch* se encontre dentro do intervalo de medição δt. Em seguida, na linha 3 a estimativa da quantidade de pacotes $M_{v_i,v_e}(\delta t)$ é inicializada como zero.

O laço da linha 4 até a linha 14 executa para cada *bitmap* bmp^i do observador v_i dentro do intervalo de medição δt, isto é para cada $bmp^i \in I^{v_i}$. O laço da linha 5 até a linha 13 executa para cada *bitmap* bmp^e do observador v_e dentro do intervalo de medição δt, isto é para cada $bmp^e \in I^{v_e}$. Dessa maneira, os laços executam $|I^{v_i}| \times |I^{v_e}|$ vezes, porém, como dito na Seção 4.1, não é possível saber a priori quantos *bitmaps* estão dentro do intervalo de medição, pois dependem do tráfego que passou pelos observadores.

Na linha 6, a função `Sobrepoe` verifica se os *bitmaps* bmp^i e bmp^e se sobrepõe, isto é, se os *epochs* dos *bitmaps* se interceptam. Caso os *bitmaps* se sobreponham, significa que os *bitmaps* foram marcados durante um mesmo *epoch*, sendo assim, se um pacote $p_{[v_i,v_e]}$ trafegou de v_i para v_e neste *epoch*, a mesma posição nos *bitmaps* bmp^i e bmp^e estarão marcadas indicando a passagem do pacote $p_{[v_i,v_e]}$. Dessa maneira, da linha 7 até a linha 10 é feita a estimativa de quantos pacotes trafegarem entre os *bitmaps*. A estimativa de quantos pacotes passaram por um *bitmap* é feita através da função `Estima` que, dado um

bitmap, calcula a estimativa da quantidade de pacotes marcados D_{bmp} de acordo com a seguinte equação (22):

$$D_{bmp} = b \ln \frac{b}{z} \tag{4.1}$$

onde b é o tamanho do *bitmap* e z é a quantidade de elementos com valor 0 no *bitmap*.

Para estimar a quantidade de pacotes que trafegaram entre os *bitmaps* bmp^i para bmp^e é preciso estimar a quantidade de pacotes da interseção dos *bitmaps* $D_{bmp^{i \cap e}}$, que é dada pela seguinte equação:

$$D_{bmp^{i \cap e}} = D_{bmp^i} + D_{bmp^e} - D_{bmp^{i \cup e}} \tag{4.2}$$

onde $D_{bmp^{i \cup e}}$ é a estimativa da quantidade de pacotes da união dos *bitmaps*. A união dos *bitmaps* é calculada como um *ou bit* a *bit* entre os *bitmaps* bmp^i e bmp^e.

Na linha 11, a quantidade de pacotes estimada $M_{v_i,v_e}(\delta t)$ é atualizado com a estimativa da quantidade de pacotes trafegados entre os *bitmaps* bmp^i e bmp^e, durante o *epoch* em que eles foram marcados.

4.2 Projeto e Implementação da Ferramenta BitMatrix

A BitMatrix é uma ferramenta de simulação de tráfego focada na geração das matrizes de tráfego utilizando a técnica *Bitmap Based Algorithm* descrita na Seção 4.1. Conforme mostrado na Figura 5, a ferramenta é dividida em 3 módulos principais: `Entrada`, `Processamento` e `Saída`. A seguir serão descritos cada um dos módulos, suas funcionalidades e o relacionamento entre eles.

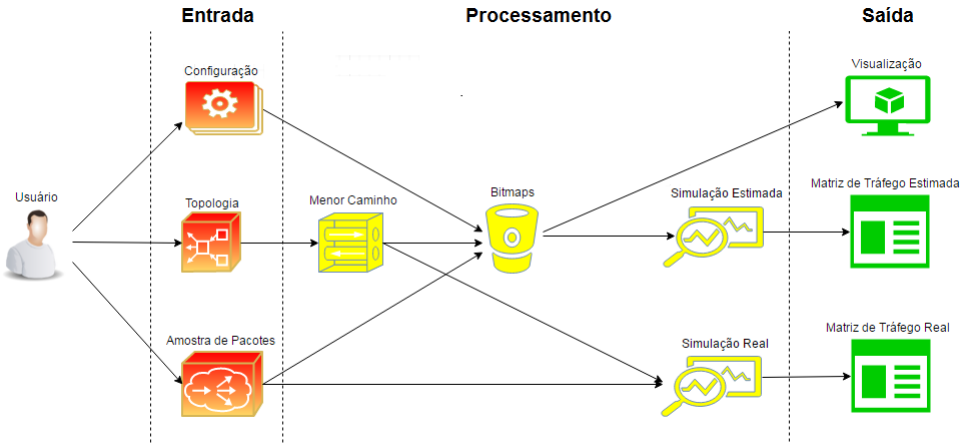


Figura 5 – Módulos da Ferramenta BitMatrix

4.2.1 Módulo de Entrada

O `Módulo de Entrada` permite ao usuário definir o cenário da simulação a ser executada. Três blocos de entrada estão disponíveis:

- `Configuração`: permite ao usuário definir os parâmetros desejados para os *bitmaps*, ou seja, o tamanho e o limite do número de acessos. Neste módulo define-se também o intervalo de tempo de medição δt para o qual se deseja gerar a matriz.

- `Topologia`: faz a leitura dos arquivos com as topologias das redes simuladas e as carrega em memória. Conforme as definições do Capítulo 3, cada topologia é representada como um grafo $G = (V,A)$ onde o conjunto de vértices V representa os dispositivos da rede e o conjunto de arestas A representam os *links* entre os dispositivos.

- `Amostra de Pacotes`: faz a leitura dos arquivos de *traces* com dados de tráfego (pacotes) num formato .csv. Cada registro de *trace* contém: *timestamp*, endereço IP de origem, endereço IP de destino, porta de origem, porta de destino, tipo de protocolo e o primeiro *byte* de dados do pacote.

4.2.2 Módulo de Processamento

O `Módulo de Processamento` executa a simulação definida pelas entradas e definições feitas no módulo anterior. No decorrer da simulação é feita a marcação dos *bitmaps*, a geração da matriz de tráfego estimada e a geração da matriz de tráfego real. Em seguida, as duas matrizes, a estimada e a real, podem ser comparadas e a escolha dos parâmetros de simulação pode ser avaliada.

A simulação inicia-se pelo bloco `Menor Caminho` conforme mostrado na Figura 5. Nesse bloco é construído o conjunto de caminhos mínimos entre cada par de vértices da rede definida em `Topologias`. Os caminhos mínimos são descobertos aplicando-se o algoritmo de *Shortest Path* criado por Dijkstra(36).

O segundo passo na execução deste bloco consiste em, dado o conjunto de caminhos mínimos (`Menor Caminho`), os dados de tráfego anonimizados (`Amostras de Pacotes`) e as configurações dos *bitmaps*, simular o tráfego na rede, ou seja o fluxo de pacotes que atravessa os vértices e as arestas do grafo. Para tanto, associa-se, de modo determinístico, blocos de endereços IP de origem e destino a pares Ingresso-Egresso na rede simulada. A simulação do tráfego é feita de maneira que, cada vez que um pacote passa por um dispositivo, marca-se uma posição no *bitmap* deste dispositivo, conforme o Algoritmo 1 apresentado na Seção 4.1.1.

O bloco de `Simulação Real` gera uma matriz de tráfego sem perdas de precisão, obtida através da contagem direta do tráfego à partir dos *traces*. Para cada registro de *trace*, recupera-se o par de Ingresso-Egresso, deterministicamente a partir dos endereços de IP de origem e IP de destino, e o caminho entre o par Ingresso-Egresso. Então a contagem do pacote entre os pares de ingresso e egresso contidos no caminho encontrado é realizado

conforme mostrado no Capítulo 3. Essa matriz de tráfego real serve como base na avaliação da precisão das matrizes estimadas.

O bloco de `Simulação Estimada` estima uma matriz de tráfego utilizando o Algoritmo 2 descrito na Seção 4.1.2. Esse bloco é executado após a criação dos *bitmaps* em cada dispositivo na rede.

4.2.3 Módulo de Saída

A Figura 5 mostra também o `Módulo de Saída` responsável por apresentar as informações geradas no `Módulo de Processamento` possibilitando a análise dos parâmetros de configuração.

No bloco `Visualização`, é possível apresentar o caminho de cada um dos pacotes trafegados pela rede. A BitMatrix usa o Graphviz como ferramenta de visualização. Como exemplo, na Figura 6, os arcos azuis mostram o caminho que um pacote específico percorre na rede definida pelo usuário. Neste exemplo, o vértice 1 atua como Ingresso e o vértice 9 representa o vértice de Egresso do pacote na rede.

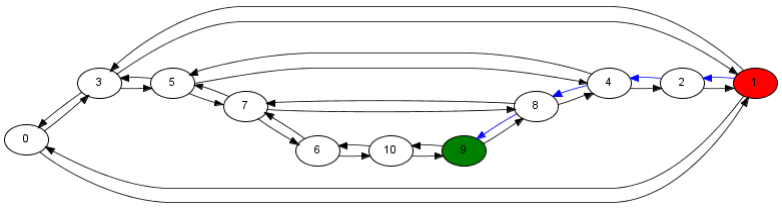


Figura 6 – Exemplo de visualização de um pacote trafegando na rede

No bloco `Matriz de Tráfego Real`, a matriz de tráfego real construída ao final da execução do bloco `Simulação Real` é armazenada em um arquivo `.csv`. No bloco `Matriz de Tráfego Estimada` a matriz de tráfego estimada gerada ao final da execução do bloco `Simulação Estimada` também é salva em arquivo `.csv`.

A partir dessas matrizes, o administrador da rede, responsável pela configuração dos *bitmaps*, pode fazer uma avaliação das diferentes configurações. Para realizar a comparação entre as matrizes de tráfego basta calcular a diferença percentual entre a quantidade de pacotes da matriz real e da matriz simulada. Caso não esteja satisfeito com a precisão obtida nas matrizes de tráfego estimadas, ele mesmo pode reconfigurar os *bitmaps* e executar o BitMatrix novamente. Ou seja, a ferramenta BitMatrix permite que seja feito o ajuste fino dos parâmetros de configuração dos *bitmaps* até que eles atendam à precisão desejada para a rede simulada.

4.3 Avaliações e Resultados

A BitMatrix foi implementada na linguagem de programação Java, devido ao domínio dos autores com essa linguagem. O código, bem como todos os resultados obtidos com a ferramenta se encontram disponíveis publicamente no endereço: <https://github.com/nerds-ufes/bitmatrix>.

Na avaliação da ferramenta foi utilizada uma amostra de pacotes com aproximadamente 100.000 registros de *traces* de tráfego real (39). Os dados do primeiro *byte* do campo de dados do pacote foram gerados aleatoriamente, pois os *traces* utilizados são anonimizados e, portanto, não possuem tal informação. Para apresentação, a topologia utilizada foi a Abilene (40), com quantidade de vértices $|V| = 11$. O tamanho do intervalo de medição Δt foi de aproximadamente 2 minutos.

Nessas condições o tráfego foi simulado na rede com diversas configurações dos parâmetros. Os tamanhos dos *bitmaps* variam entre 64 e 65536 *bits*, o que abrange o maior tráfego entre os pares na rede. Os limites de número de acessos aos *bitmaps* variam em 30% e 50% do tamanho total dos *bitmaps*. Conforme dito no início deste Capítulo, intuitivamente, tamanhos maiores com menor limite de acessos devem apresentar menos colisões de *hashing* e, portanto, melhores resultados.

De acordo com (22), a estratégia de medição utilizando estruturas de dados probabilísticas do tipo *bitmap* é mais adequada à detecção de grandes tráfegos entre pares de dispositivos de rede. Como a ferramenta BitMatrix faz uso dessa estratégia de estimação da matriz de tráfego, a visualização e demonstração dos resultados serão apresentados em função dos dez maiores tráfegos entre vértices de Ingresso-Egresso, medidos em quantidade de pacotes trocados entre os pares. Esses dez maiores tráfegos são chamados nesta dissertação de Top 10. Nos resultados quanto menor a distância da medição entre a análise real e a análise estimada melhor será o ajuste entre tamanho e limite de acessos aos *bitmaps*.

Dentre os testes realizados, foram selecionados 8 tipos de configurações para serem exibidas e seus desempenhos podem ser avaliados pelas Tabelas 1, 2 e 3. As Tabelas 1 e 2 mostram a comparação da quantidade de pacotes medida em cada um dos Top 10 pares de vértices. Na Tabela 1 são exibidos os valores absolutos obtidos pela matriz real e os valores absolutos das matrizes estimadas obtidos por cada configuração dos *bitmaps*. Na Tabela 2 são exibidos os valores absolutos obtidos pela matriz real e o as diferenças percentuais das matrizes estimadas em relação à matriz real obtidos por cada configuração dos *bitmaps*. Em ambas as tabelas, os valores destacados em negrito exibem as configurações que mais se aproximaram do valor obtido pela matriz real para cada um dos Top 10 pares de vértices.

Observando as Tabelas 1 e 2, é possível observar que, ao contrário do que suposto inicialmente, aumentar o tamanho do *bitmap* não necessariamente é a melhor solução, dado que os resultados das matrizes estimadas mais precisos se encontram nas configurações

Tabela 1 – Quantidade de pacotes amostrados pela matriz real e pelas matrizes estimadas em cada configuração dos *bitmaps*

	Quantidade de Pacotes Amostrados								
Top 10	Real	64 bits		512 bits		16384 bits		65536 bits	
		30%	50%	30%	50%	30%	50%	30%	50%
#1	43328	**41977**	41960	41509	41790	41209	41318	40471	40046
#2	34267	34628	**34275**	33895	34051	33874	33950	33115	32720
#3	28134	28352	27924	27809	**28126**	27636	27874	27093	26792
#4	27079	27202	27118	26989	**27051**	26556	26490	26214	25727
#5	22558	22680	22744	**22597**	22246	22342	22241	21477	20585
#6	22193	22449	22477	**22173**	21488	21880	21641	21365	20413
#7	21311	21558	21476	21706	21129	**21400**	21055	20058	19072
#8	19971	**19951**	19629	19891	19556	19586	19343	19199	18508
#9	19466	19306	19236	**19538**	19538	19157	18961	18709	18261
#10	19402	**19235**	18963	19221	18988	19036	18766	18659	18244

Tabela 2 – Distância percentual da quantidade de pacotes amostrados pelas matrizes estimadas, em cada configuração de *bitmaps*, em relação à matriz real

	Distância Percentual								
Top 10	Real	64 bits		512 bits		16.384 bits		65.536 bits	
		30%	50%	30%	50%	30%	50%	30%	50%
#1	43328	**-3.12**	-3.16	-4.20	-3.55	-4.89	-4.64	-6.59	-7.57
#2	34267	1.05	**0.02**	-1.09	-0.63	-1.15	-0.93	-3.36	-4.51
#3	28134	0.77	-0.75	-1.16	**-0.03**	-1.77	-0.92	-3.70	-4.77
#4	27079	0.45	0.14	-0.33	**-0.10**	-1.93	-2.18	-3.19	-4.99
#5	22558	0.54	0.82	**0.17**	-1.38	-0.96	-1.41	-4.79	-8.75
#6	22193	1.15	1.28	**-0.09**	-3.18	-1.41	-2.49	-3.73	-8.02
#7	21311	1.16	0.77	1.85	-0.85	**0.42**	-1.20	-5.88	-10.51
#8	19971	**-0.10**	-1.71	-0.40	-2.08	-1.93	-3.14	-3.87	-7.33
#9	19466	-0.82	-1.18	**0.37**	**0.37**	-1.59	-2.59	-3.89	-6.19
#10	19402	**-0.86**	-2.26	-0.93	-2.13	-1.89	-3.28	-3.83	-5.97

com menores tamanho de *bitmaps*. De fato, percebe-se que as matrizes de tráfego obtidas com tamanho de *bitmaps* de 65536 *bits* possuem as piores precisões. Algumas possíveis explicações para esses resultados: i) o conjunto de pacotes utilizados existirem muitas retransmissões, o que leva a acontecer muitas colisões de *hashing*, pois um mesmo pacote sempre vai marcar as mesmas posições ou ii) a função de *hashing* pode não estar fazendo uma boa distribuição e causando muitas colisões mesmo para pacotes diferentes.

Ainda nas Tabelas 1 e 2, percebe-se que para tamanhos de *bitmaps* pequenos (64 e 512 *bits*), a configuração de limites de acesso aos *bitmaps* fizeram pouca diferença, visto que a precisão dos resultados varia entre as configurações de limites de acesso 30% e 50%. Já no caso dos *bitmaps* de tamanhos maiores (16384 e 65536 *bits*), o limite de acessos começa a fazer diferença e configurações com menor limite de acessos (30%) tem resultados melhores que resultados com maior limite de acessos (50%). Uma explicação possível é que, muitos *bitmaps*, com diferentes *bitmaps epochs*, são armazenados em configurações de menor tamanho, então é possível que a combinação desses *bitmaps* para gerar as matrizes de tráfego tenham mascarado o efeito do parâmetro do limite de acessos.

Os resultados obtidos nas Tabelas 1 e 2 mostram a BitMatrix consegue gerar matrizes de trafego utilizando o *Bitmap Based Algorithm*, porém para entender melhor a influência dos parâmetros verificados nesse capítulo, deve-se levar em consideração outras variáveis como: as características do tráfego utilizado e a combinação dos vários *bitmaps* gerados pelo algoritmo.

A Tabela 3, mostra quais são os Top 10 pares de vértices obtidos. São exibidos os Top 10 pares encontrados pela matriz real e os Top 10 pares encontrados nas matrizes estimadas por cada configuração dos *bitmaps*. Os pares destacados em negrito mostram os pares encontrados pelas matrizes estimadas que não condizem com os pares obtidos pela matriz real.

Tabela 3 – Top-10 pares de vértices amostrados pela matriz real e pelas matrizes estimadas em cada configuração dos *bitmaps*

	Pares de Vértices								
Top 10	Real	64 bits		512 bits		16.384 bits		65.536 bits	
		30%	50%	30%	50%	30%	50%	30%	50%
#1	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)	(5,9)
#2	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)	(9,10)
#3	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)	(5,10)
#4	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)	(3,5)
#5	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	(4,6)	**(2,4)**
#6	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	**(4,6)**
#7	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)	(6,8)
#8	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	**(2,5)**
#9	(2,5)	(2,5)	(2,5)	**(3,9)**	**(3,9)**	**(3,9)**	**(3,9)**	**(3,9)**	**(3,9)**
#10	(3,9)	(3,9)	(3,9)	**(2,5)**	**(2,5)**	**(2,5)**	**(2,5)**	**(2,5)**	**(2,3)**

Na Tabela 3 é possível notar que o conjunto de Top-10 pares foi mantido em todas as configurações. Comparando os pares de vértices destacados em negrito, onde os pares não foram os mesmos que a matriz real, com a quantidade de pacotes medidas pela matriz real na Tabela 2, percebe-se que os pares fora de posição são aqueles com uma quantidade de pacotes bem próxima. Por exemplo, o par #8 na matriz real terminou na posição #10 na configuração com tamanho 65536 *bits* e limite de acesso de 50%. A quantidade de pacotes medidas para esses pares, #8 e #10, foram bem próximos na matriz real, sendo de 19971 e 19402 pacotes, respectivamente, sendo então a diferença entre as quantidades medidas de 569 pacotes (aproximadamente 3% de diferença entre as medições). Portanto, dependendo da aplicação, essa troca entre posições pode ser considerada aceitável. Dessa maneira, a Tabela 3 mostra a viabilidade de se utilizar o *Bitmap Based Algorithm* na geração de matrizes de tráfego.

4.4 Conclusões

Este capítulo apresentou uma ferramenta que implementa a estratégia de construção de matrizes de tráfego através de medição direta usando o *Bitmap Based Algorithm*. A

BitMatrix possibilita a experimentação dos diferentes parâmetros de configuração dos *bitmaps* que afetam a precisão das matrizes de tráfego estimadas. Essa precisão pode ser avaliada a partir da comparação com as matrizes de tráfego reais, obtidas através da simples contagem de pacotes.

Concluiu-se também que algumas melhorias podem ser implementadas na BitMatrix afim de se tentar verificar uma tendência na alteração dos parâmetros dos *bitmaps*, melhorando a capacidade de ajuste da ferramenta. Exemplos de de melhoria podem ser: i) o levantamento de distribuição do tráfego simulado, afim de se conseguir caracterizar o tráfego tornando possível entender melhor o comportamento do algoritmo para determinados tipos de tráfego e ii) implementação de diferentes tipos de funções de *hashing* afim de se verificar a influência dessas funções nos resultados. Mesmo sem essas melhorias, o objetivo deste capítulo foi alcançando, pois a BitMatrix mostrou que é possível gerar matrizes de tráfego utilizando um algoritmo de processamento de cadeia de dados baseado em estrutura de dados probabilísticas, mostrando uma boa indicação dos maiores tráfegos na rede.

No Capítulo 5 a seguir, será adicionado um módulo à ferramenta BitMatrix para gerar matrizes de tráfego utilizando o algoritmo GeraMatriz, proposto no próprio Capítulo 5. Dessa maneira, as matrizes geradas pelo *Bitmap Based Algorithm* serão utilizadas para comparação com as matrizes obtidas pelo algoritmo GeraMatriz proposto no Capítulo 5.

5 Conjunto Reduzido de Observadores para Geração de Matrizes de Tráfego

No Capítulo 4 foi mostrado que é possível gerar matrizes de tráfego utilizando algoritmos para processamento de cadeia de dados baseados em estruturas de dados probabilísticas do tipo *bitmap* instaladas nos observadores. Porém, conforme já discutido nos Capítulos 1 e 2, apesar de, em geral, esses algoritmos poderem ser implementados com baixa complexidade de armazenamento e processamento, sua utilização depende da instalação de muitos observadores na rede, o que incorre em:

- **Sobrecarga de Tráfego:** as informações coletadas pelos observadores, em algum momento, devem ser enviadas para a estação de monitoramento de rede;
- **Custos de Implantação:** custos de *hardware*, *software* e de manutenção, pois nem todos os observadores de rede já instalados estão aptos a instanciar e gerenciar as estruturas de dados probabilísticas necessárias.

Esta abordagem gera uma quantidade significativa de tráfego que compartilha a mesma infra-estrutura de rede com serviços de usuário. Do ponto de vista desses serviços, esse tráfego é uma sobrecarga, uma vez que ele interfere na disponibilidade de largura de banda para os dados. Ou seja, a própria atividade de monitoramento interfere no tráfego de rede. Portanto, para a geração das matrizes de tráfego em uma rede real é importante que se minimize a quantidade de observadores instalados na rede.

Com o objetivo de minimizar a quantidade de observadores, essa dissertação apresenta a proposta a seguir. Conforme dito nos capítulos anteriores, para a construção de uma Matriz de Tráfego é necessário monitorar o tráfego da rede, que consiste em contar quantos pacotes trafegaram entre todos os pares de vértices ingresso-egresso $[v_i, v_e]$. Dessa maneira, observa-se que quando um pacote $p_{[v_i,v_e]}$ viaja de v_i para v_e, ele passa por todos os vértices no caminho $C_{[v_i,v_e]}$. Portanto, considerando a restrição definida no Capítulo 3 onde foi adotado que o caminho $C_{[v_i,v_e]}$ entre os vértices v_i e v_e é único, para contar quantos pacotes trafegaram entre um determinado par de vértices ingresso-egresso $[v_i, v_e]$, pode-se instalar um único observador em um vértice $v_o \in C_{[v_i,v_e]}$. Para monitorar toda a rede, deve-se instalar observadores em um conjunto de vértices observadores $V^O \subseteq V$, tal que o conjunto V^O cubra todo o conjunto de caminhos $\mathcal{C}$, definido no Capítulo 3 como o conjunto de todos os caminhos entre os pares de ingresso-egresso $C_{[v_i,v_e]}$ da rede. Como exemplo, a topologia exibida no Capítulo 3 é mostrada novamente na Figura 7. Nessa topologia, se o conjunto de vértices de ingresso for $V^I = \{0,1,2\}$ e o conjunto de vértices

de egressos for $V^E = \{5,6,7\}$, o conjunto reduzido, que nesse caso também é mínimo, de observadores é $V^O = \{3, 4\}$, pois estes nós cobrem todos os caminhos presentes entre os pares de ingresso-egresso na rede, logo todos os pacotes passam por eles.

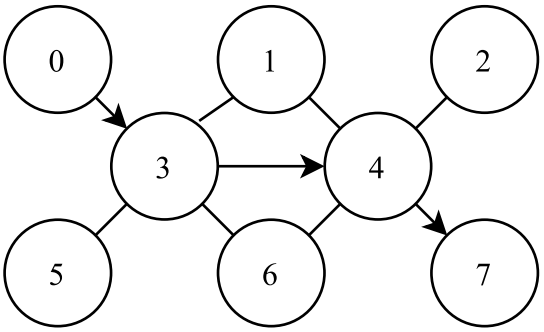


Figura 7 – Exemplo de um pacote trafegando em um caminho da rede.

Dessa maneira, esta dissertação define o problema de Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego (MCO-MT) que, conforme já discutido nos capítulos anteriores, consiste em minimizar a quantidade de vértices observadores instalados na rede, ou seja, minimizar o tamanho do conjunto de vértices observadores necessários para gerar matrizes de tráfego utilizando estruturas de dados probabilísticas. Formalmente, utilizando a notação adotada no Capítulo 3, o MCO-MT consiste em: dados i) uma topologia de rede definida como um grafo $G = (V,A)$; ii) o conjunto de vértices de ingresso $V^I \subseteq V$; iii) e o conjunto de vértices de egresso $V^E \subseteq V$, encontrar o conjunto de caminhos $\mathcal{C}$ que conecta todos os pares de vértices ingresso-egresso e um conjunto de vértices observadores V^{O^*} capaz de monitorar todos os caminhos encontrados tal que o tamanho do conjunto $|V^{O^*}|$ seja minimizado. O conjunto de vértices observadores V^{O^*} é um conjunto reduzido de observadores em relação ao conjunto completo de observadores que utiliza todos os vértices ingresso e egresso da rede.

Neste contexto, este capítulo apresentada o algoritmo GeraMatriz que é um algoritmo de processamento de cadeia de dados baseado em estruturas de dados probabilísticas capaz de gerar matrizes de tráfego a partir de um conjunto reduzido de observadores. Para isso, o algoritmo utiliza a estrutura de dados denominada de *counter-matrix*, também proposto neste capítulo. O algoritmo foi baseado no trabalho de Zhao et al.(22). O *counter-matrix* é basicamente uma matriz de contadores construída em um observador afim de monitorar um caminho na rede, onde cada elemento indica o volume de dados, medido em quantidade de pacotes, entre um par ingresso e egresso no caminho monitorado. Em posse dessas informações dos *counter-matrix* e dos caminhos utilizados pela rede, a estação de monitoramento centralizada é capaz de gerar as matrizes de tráfego. Os resultados apresentados neste capítulo permitem afirmar que, com essa estratégia, implantando apenas um conjunto reduzido de observadores é possível gerar matrizes de tráfego sem perdas de precisão.

O restante deste capítulo está organizado na seguinte forma:

- a Seção 5.1 apresenta o algoritmo GeraMatriz para geração de matrizes de tráfego;
- a Seção 5.2 apresenta os resultados obtidos pelo algoritmo GeraMatriz;
- a Seção 5.3 conclui o capítulo.

5.1 GeraMatriz

A arquitetura do sistema para aplicação do algoritmo GeraMatriz é mostrada na Figura 8. O conjunto de dispositivos de ingresso e egresso são indicados nas figuras como $V^I = \{I_1, I_2, I_3\}$ e $V^E = \{E_1, E_2, E_3\}$, respectivamente. O algoritmo é dividido em três módulos.

No primeiro módulo, denominado *MCO-MT Solver Module*, em posse da topologia da rede e dos conjuntos de nós ingresso e egresso, a Estação de Monitoramento define o conjunto de caminhos e um conjunto reduzido de observadores capaz de monitorar esses caminhos. Na Figura 8, o conjunto reduzido, e nesse caso também mínimo, de observadores é representado pelo conjunto $V^{O^*} = \{R_1, R_2\}$, visto que todos os caminhos entre os conjuntos de ingressos e egressos passam pelos dispositivos R_1 ou R_2. Observe que, diferente do *Bitmap Based Algorithm* apresentado na Seção 4.1, o algoritmo GeraMatriz pode utilizar dispositivos que não estejam nos conjuntos de ingressos ou egressos. Os outros dois módulos são baseados nos dois módulos do *Bitmap Based Algorithm* e serão denominados igualmente no algoritmo GereMatriz.

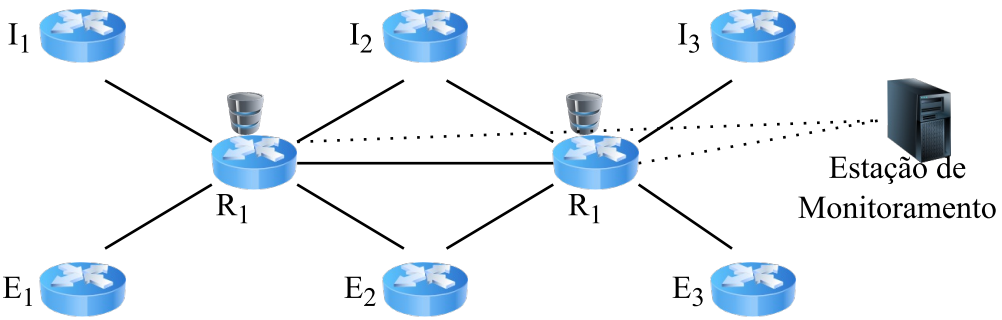


Figura 8 – Arquitetura do sistema na aplicação do algoritmo GeraMatriz de geração de matrizes com alocação reduzida de observadores.

No módulo *Online Streaming Module*, cada vértice observador $v_o \in V^{O^*}$ selecionado no primeiro módulo realiza a contagem utilizando o *counter-matrix* CM atualmente em uso, que está instanciado em memória volátil do vértice observador v_o. Após um intervalo de tempo, os *counter-matrix* em uso são movidos para a memória permanente e armazenados localmente em cada observador. O intervalo de tempo em que um *counter-matrix* permanece

em memória volátil é denominado *counter-matrix epoch*. Conforme notação adotada no Capítulo 3, o *counter-matrix epoch* é definido como $\delta e = [e, e + \Delta e)$, onde e é o início da marcação do *counter-matrix* e $e + \Delta e$ o fim da marcação. Diferentemente do *bitmap epoch* utilizado no *Bitmap Based Algorithm*, o tamanho do *counter-matrix epoch* Δe é um parâmetro, ou seja, os tamanhos dos *counter-matrix epoch* Δe devem ser configurados nos observadores.

Os *counter-matrix* permanecem armazenados localmente nos observadores durante um período de tempo definido pelo administrador da rede. Esse período de tempo não interfere na precisão da medição. Em seguida, os *counter-matrix* são enviados à Estação de Monitoramento sob demanda. A Estação de Monitoramento então executa o *Data Analysis Module* para gerar a matriz de tráfego a partir das informações dos caminhos da rede e dos *counter-matrix*.

Na sequencia desta seção, os módulos *Online Streaming Module* e *Data Analysis Module* serão explicados em detalhes. Algoritmos para resolver o problema MCO-MT no módulo *MCO-MT Solver Module* serão apresentados no Capítulo 6.

5.1.1 *Online Streaming Module*

Cada observador deve instanciar e gerenciar um *counter-matrix*, que é basicamente uma matriz de contadores CM, onde cada posição $CM_{[v_i,v_e]}$ irá armazenar a quantidade de pacotes trafegados no caminho $C_{[v_i,v_e]}$. O Algoritmo 3 mostra o procedimento de gerenciamento dos *counter-matrix* em cada observador. O algoritmo é executado no momento em que o observador entra em funcionamento e recebe como entrada: o tamanho do conjunto de ingresso $|V^I|$, o tamanho do conjunto de egresso $|V^E|$ e o tamanho do *counter-matrix epoch* Δe.

Para se definir o tamanho do *counter-matrix epoch* Δe, deve-se levar em consideração o tipo de dado utilizado pela *counter-matrix*, isto é, até quanto um elemento $CM_{[v_i,v_e]}$ consegue contar. Por exemplo, se a *counter-matrix* for do tipo `short`, o intervalo de pacotes que o elemento $CM_{[v_i,v_e]}$ consegue contar é $[0, 65536)$. Dessa maneira, deve-se escolher um tamanho de *counter-matrix epoch* Δe que permita contar a quantidade de pacotes entre os pares ingresso-egresso com maior tráfego no caminho monitorado pelo observador. Além disso, será mostrado na Seção 5.1.2 que para estimar o tráfego entre um par Ingresso-Egresso, o algoritmo GeraMatriz precisa de todos os *counter-matrix* ao mesmo tempo. Dessa maneira, o intervalo dos *counter-matrix epoch* Δe devem ser alinhados entre os observadores e com o tempo de medição δt da matriz de tráfego (ver Capítulo 3).

O Algoritmo 3 inicia na linha 2 onde a função `NovoCounterMatrix` instancia em memória o *counter-matrix* em uso com $|V^I|$ linhas e $|V^E|$ colunas, onde todas as posições são inicializadas como 0. Observe que, conforme definição no Capítulo 3, o tamanho máximo

Algoritmo 3: *GeraMatriz - Online Streaming Module*

Entrada: Tamanho dos conjunto de ingresso e egresso $|V^I|$ e $|V^E|$
Entrada: Tamanho do *counter-matrix epoch* Δe

```
1  início
2      CM ← NovoCounterMatrix(|V^I|, |V^E|)
3      inicioEpoch ← DataHoraAtual()
4      enquanto observador estiver funcionando faça
5          pkt ← RecebePacote()
6          [v_i, v_e] ← RecuperaInterfaces(pkt)
7          CM_[v_i,v_e] ← CM_[v_i,v_e] + 1
8          se (DataHoraAtual()−inicioEpoch) ≥ Δe então
9              Armazena(CM,inicioEpoch,DataHoraAtual())
10             CM ← NovoCounterMatrix(|V^I|, |V^E|)
11             inicioEpoch ← DataHoraAtual()
12         fim se
13     fim enqto
14 fim
```

da matriz pode ser de até $|V| \times |V|$, visto que, no pior caso, todos os nós participam como ingresso e egresso. Do ponto de vista prático, o tamanho da matriz pode ser um problema, pois a quantidade de memória necessária para instanciar o *counter-matrix* cresce conforme a matriz aumenta e, conforme já dito nesta dissertação, os dispositivos na rede possuem fortes restrições quanto à quantidade de memória e de armazenamento.

Seguindo com o Algoritmo 3, a data e hora atuais são salvas como o início do *counter-array epoch* na variável *inicioEpoch* na linha 3. O laço da linha 4 até a linha 13 controla os *counter-matrix* e executa enquanto o observador estiver em funcionamento na rede. Na linha 5 a função `RecebePacote` espera até que um pacote seja recebido em uma das interfaces do observador. No momento em que um pacote é recebido, a função `RecebePacote` retorna o pacote para a variável *pkt*. Em posse do pacote, na linha 6, a função `RecuperaInterfaces` recupera, a partir do cabeçalho do pacote, os vértices de ingresso e egresso nas variáveis v_i e v_e que são utilizados na linha 7 para incrementar em 1 o valor do elemento $CM_{[v_i,v_e]}$, registrando a passagem de mais um pacote do ingresso v_i para o egresso v_e.

Observe que, do ponto de vista prático, a utilização da função `RecuperaInterfaces` imprime uma forte restrição ao tipo de redes onde o algoritmo GeraMatriz pode ser aplicado, visto que encontrar os vértice de ingresso e egresso de um pacote a partir do seu cabeçalho não é uma tarefa fácil. Essa dificuldade acontece porque nos cabeçalhos IP de um pacote é possível identificar seus IP de origem e IP de destino, mas não existe nenhuma informação a respeito dos vértices por onde o pacote entra e sai na rede. Um exemplo onde é possível implementar a função `RecuperaInterfaces` é uma rede onde o conjunto de IPs que entram e saem de cada vértice sejam bem definidos e conhecidos. Dessa maneira é possível construir uma tabela que faça a tradução de IP de origem e

destino em vértice de ingresso e egresso.

Após o registro do pacote, a linha 8 realiza o teste de substituição do *counter-array*. Se o intervalo atual do *counter-array epoch*, dado pela data e hora atuais subtraído do *inicioEpoch*, for maior que o intervalo máximo Δe, o *counter-array* será movido da memória para o disco na linha 9. A função `Armazena` grava em disco o *counter-matrix* CM juntamente com seu *counter-matrix epoch* δe. O *counter-matrix epoch* δe é calculado da seguinte maneira: $\delta e = [inicioEpoch, \texttt{DataHoraAtuais})$. Em um vértice observador $v_o \in V^{O^*}$ podem existir um conjunto de *counter-matrix* armazenados $\mathcal{CM}_{v_o}$. Um *counter-matrix* CM que foi armazenado com o *counter-matrix epoch* δe em um vértice observador v_o é definido como $CM_{v_o}^{\delta e} \in \mathcal{CM}_{v_o}$.

Na linha 10, a função `NovoCounterMatrix` é chamada para instanciar em memória uma nova *counter-matrix* com tamanho $|V^I| \times |V^E|$ e, em seguida, na linha 11 o início do novo *counter-matrix epoch* é armazenado na variável *inicioEpoch*.

5.1.2 *Data Analysis Module*

Quando for necessário estimar a matriz de tráfego da rede $M(\delta t)$ em um intervalo de medição δt qualquer, a Estação de Monitoramento recupera o conjunto $\mathcal{CM}^{\delta t}$ de todos os *counter-matrix* $CM_{v_o}^{\delta e} \in \mathcal{CM}^{\delta t}$ que estão armazenados localmente em cada vértice observador $v_o \in V^{O^*}$ e cujos *counter matrix epoch* δe estejam dentro do intervalo de medição δt.

Conforme discutido na Seção 5.1.1, cada vértice observador $v_o \in V^{O^*}$ possui um conjunto $\mathcal{CM}_{v_o}$ de *counter-matrix* armazenados localmente. Cada *couter-matrix* $CM_{v_o}^{\delta e} \in \mathcal{CM}_{v_o}$ está associado a um único *counter-matrix epoch* δe. Para selecionar o conjunto $\mathcal{CM}_{v_o}^{\delta t} \subseteq \mathcal{CM}_{v_o}$ de *counter-matrix* que estejam dentro do intervalo de medição δt no conjunto de *counter-matrix* $\mathcal{CM}_{v_o}$ armazenados em um vértice observador v_o, basta buscar os *counter-matrix* $CM_{v_o}^{\delta e}$ cujos *counter-matrix epoch* δe estejam dentro do intervalo de medição δt. Como os *counter-matrix epoch* δe são alinhados com o intervalo de medição δt e alinhados dentro do conjunto de vértices observadores V^{O^*}, a quantidade $|\mathcal{CM}_{v_o}^{\delta t}|$ de *counter-matrix* dentro de intervalo de medição δt selecionados em cada vértice observador v_o é a mesma. Dessa maneira, o tamanho do conjunto $\mathcal{CM}^{\delta t}$ é dado por $|\mathcal{CM}^{\delta t}| = |\mathcal{CM}_{v_o}^{\delta e}| \times |V^{O^*}|$.

Conforme apresentado no início deste capítulo, o conjunto reduzido de observadores V^{O^*} é construído de maneira que exista pelo menos um vértice observador $v_o \in V^{O^*}$ em cada caminho $C_{[v_i,v_e]} \in \mathcal{C}$ entre os pares de vértices de ingresso e egresso, isto é, $|C_{[v_i,v_e]} \cap V^{O^*}| \geq 1$. Como cada vértice observador v_o possui um conjunto $\mathcal{CM}_{v_o}^{\delta t}$ de *counter-matrix* que estão dentro do intervalo de medição δt, pode-se dizer que um caminho $C_{[v_i,v_e]}$ possui um conjunto $\mathcal{CM}_{C_{[v_i,v_e]}}^{\delta t} \subset \mathcal{CM}^{\delta t}$ de *counter-matrix* dentro de intervalo de medição δt. Considerando que os *counter-matrix epoch* δe dos *counter-matrix* $CM_{v_o}^{\delta e} \in \mathcal{CM}_{C_{[v_i,v_e]}}^{\delta t}$

são alinhados entre si, quando um pacote $p_{[v_i,v_e]}$ trafega pelo caminho $C_{[v_i,v_e]}$ durante o *counter-matrix epoch* δe, esse pacote é contabilizado em todos os *counter-matrix* $CM_{v_o}^{\delta e}$ de todos vértices observadores v_o no caminho $C_{[v_i,v_e]}$. Logo, para contabilizar quantos pacotes passaram pelo caminho $C_{[v_i,v_e]}$, são necessários apenas os *counter-matrix* $CM_{v_o}^{\delta e}$ de um vértice observador qualquer $v_o \in C_{[v_i,v_e]} \cap V^{O^*}$. Desta maneira, deste ponto em diante o conjunto $\mathcal{CM}_{C_{[v_i,v_e]}}^{\delta t} \subset \mathcal{CM}^{\delta t}$ representa o conjunto de *counter-matrix* $CM_{v_o}^{\delta e} \in \mathcal{CM}_{C_{[v_i,v_e]}}^{\delta t}$ de um vértice observador qualquer $v_o \in C_{[v_i,v_e]} \cap V^{O^*}$.

Para calcular a matriz de tráfego $M(\delta t)$, conforme estratégia adotada no Capítulo 3, cada pacote $p_{[v_i,v_e]}$ que trafega pelo caminho $C_{[v_i,v_e]}$ deve ser contabilizado no elemento $M_{[v_j,v_k]}(\delta t)$ da matriz de tráfego para cada par de vértice de ingresso $v_j \in V^I \cap C_{[v_i,v_e]}$ e vértice egresso $v_k \in V^E \cap C_{[v_i,v_e]}$ formado neste caminho. A formação dos pares ingresso e egresso $[v_j, v_k]$ é realizada através da combinação simples dos vértices de ingresso e egresso ao longo do caminho (para exemplo, ver Capítulo 3).

Levando em consideração todos os pontos citados até aqui, o cálculo da matriz $M(\delta t)$ é mostrado no Algoritmo 4. O algoritmo recebe como parâmetros o conjunto $\mathcal{CM}^{\delta t}$ de todos os *counter-matrix* $CM_{v_o}^{\delta e} \in \mathcal{CM}^{\delta t}$ que estão armazenados localmente em cada vértice observador $v_o \in V^{O^*}$ e cujos *counter matrix epoch* δe estejam dentro do intervalo de medição δt, o conjunto $\mathcal{C}$ de caminhos entres os pares de ingresso e egresso da rede, o conjunto V^I de vértices de ingresso e o conjunto V^E de vértices de egresso. O retorno do algoritmo é a matriz de tráfego $M(\delta t)$ calculada no intervalo de tempo δt.

Algoritmo 4: *GeraMatriz - Data Analysis Module*

Entrada: Conjunto $\mathcal{CM}^{\delta t}$ de todos os *counter-matrix* $CM_{v_o}^{\delta e} \in \mathcal{CM}^{\delta t}$ que estão armazenados localmente em cada vértice observador $v_o \in V^{O^*}$ e cujos *counter matrix epoch* δe estejam dentro do intervalo de medição δt
Entrada: Conjunto $\mathcal{C}$ de caminhos entres os pares de ingresso e egresso da rede
Entrada: Conjunto V^I de vértices de ingresso
Entrada: Conjunto V^E de vértices de egresso
Saída: Matriz de tráfego $M(\delta t)$ calculada no intervalo de tempo δt

```
1  início
2      M(δt) ← 0
3      para cada C_[v_i,v_e] ∈ C faça
4          para cada CM_{v_o}^{δe} ∈ CM_{C_[v_i,v_e]}^{δt} faça
5              para cada v_j ∈ V^I ∩ C_[v_i,v_e] faça
6                  para cada v_k ∈ V^E ∩ C_[v_j,v_e] faça
7                      M_[v_j,v_k](δt) = M_[v_j,v_k](δt) + CM_{v_o}^{δe}[v_i, v_e]
8                  fim para cada
9              fim para cada
10         fim para cada
11     fim para cada
12     retorna M(δt)
13 fim
```

O Algoritmo 4 inicia na linha 2 onde a matriz de tráfego $M(\delta t)$ calculada no intervalo de tempo δt é inicializada com todas as suas posições zeradas. O laço da linha 3 até a linha 11 executa para cada caminho entre os pares de ingresso e egresso da rede $C_{[v_i,v_e]} \in \mathcal{C}$. O laço da linha 4 até a linha 10 executa para cada *counter-matrix* $CM^{\delta e}_{v_o} \in \mathcal{CM}^{\delta t}_{C_{[v_i,v_e]}}$ de um vértice observador qualquer no caminho $v_o \in C_{[v_i,v_e]} \cap V^{O^*}$. Os laços das linhas seguintes realizam a combinação dos pares de vértices de ingresso e egresso existentes no caminho $C_{[v_i,v_e]}$. O laço da linha 5 até a linha 9 executa para cada vértice de ingresso $v_j \in V^I$ existente no caminho $C_{[v_i,v_e]}$, isto é, $v_j \in V^I \cap C_{[v_i,v_e]}$. O laço da linha 6 até a linha 8 executa para cada vértice de egresso $v_k \in V^E$ existente no caminho $C_{[v_j,v_k]}$, isto é, $v_k \in V^E \cap C_{[v_j,v_e]}$. Na linha 7, o elemento da matriz de tráfego $M_{[v_j,v_k]}$ é atualizado com a quantidade de pacotes que trafegaram no caminho $C_{[v_i,v_e]}$ que foram contabilizadas pela posição *counter-matrix* $CM^{\delta e}_{v_o}[v_i, v_e]$ do *counter-array* $CM^{\delta e}_{v_o}$ durante seu *counter-matrix epoch* δe. Finalmente, na linha 12, a matriz de tráfego $M(\delta t)$ é retornada.

5.2 Avaliações e Resultados

Para avaliar o algoritmo de geração de matrizes de tráfego GeraMatriz, foi feita uma adaptação na ferramenta BitMatrix, incluindo na ferramenta a capacidade de instanciar os *counter-matrix* e gerar a matriz de tráfego conforme o algoritmo GeraMatriz apresentado. Os dados utilizados para avaliação foram *traces* anonimizados de dados de tráfego real (39) e topologias reais de Sistemas Autônomos (ASes) da Internet (40), os mesmos utilizados para testar a ferramenta BitMatrix na Seção 4.3.

Foram utilizadas 8 diferentes redes, representadas como grafos $G = (V,A)$, nos testes computacionais. Cada grafo representa a topologia de rede de um sistema autônomo (AS) diferente com quantidade de vértices $|V|$ variando de 11 a 115. Nos testes realizados, todos os nós participam como ingresso e egresso, configurando o pior caso onde tem-se $|V|(|V|-1)$ pares ingresso-egresso. Os *counter-matrix* dos nós observadores e a matriz de tráfego $M(\delta t)$ gerada têm dimensões $|V| \times |V|$. Para cada grafo, foram colocados em circulação um total de 100.000 pacotes retirados dos *traces*. Os mesmos 100.000 pacotes foram utilizados em todos os testes de tal forma a caracterizar o mesmo tráfego em todos os ASes.

O estudo da geração do conjunto reduzido de observadores encontrados pelo módulo *MCO-MT Solver Module* será feita no Capítulo 6 já que, para avaliar a qualidade da matriz gerada com o algoritmo GeraMatriz, não é necessário conhecimento sobre o conjunto de observadores. Assim, após a seleção dos observadores, procedeu-se à simulação do tráfego de 100.000 pacotes. Em cada nó observador executou-se o Algoritmo 3 para marcação dos *counter-matrix* do módulo *Online Streaming Module* do algoritmo GeraMatriz. A matriz de tráfego foi gerada aplicando o Algoritmo 4 a todos os pares ingresso-egresso no módulo

Data Analysis Module.

A Figura 9 mostra a comparação, em números de pacotes, entre a matriz de tráfego gerada pelo algoritmo GeraMatriz e a matriz real gerada pela contagem dos pacotes. Para apresentação da comparação foi escolhida a topologia do AS-3257 com $|V| = 41$. O comportamento verificado nessa topologia representa bem o padrão em todos as topologias testadas e seu tamanho é mais adequado a uma melhor visualização. O tamanho do conjunto reduzido de vértices observadores utilizando nessa rede foi de $|V^{O^*}| = 16$, alcançando uma redução de aproximadamente 61% em relação à quantidade total de vértices na rede. Detalhes de como esse conjunto foi gerado serão discutidos no Capítulo 6.

Na Figura 9, os pontos representam a quantidade de pacotes entre cada par ingresso e egresso estimados pelo algoritmo GeraMatriz. A reta Referência indica o valor obtido pela matriz real. Portanto, quanto mais próximo um ponto está da reta Referência, mais precisa foi a estimativa em relação à quantidade de pacotes na matriz de tráfego real. As linhas tracejadas representam um erro de $\pm 20\%$ no quantitativo de pacotes da matriz gerada, e foram plotados para servirem de referência do nível de erro obtido com o algoritmo GeraMatriz. Dessa maneira, como os pontos estão exatamente sobre a linha de Referência, pode-se concluir que a técnica GeraMatriz proposta neste artigo utilizando estruturas do tipo *counter-matrix* e um número reduzido de nós observadores conseguiu reproduzir uma matriz de tráfego sem erros. O motivo da matriz de tráfego estimada ser gerada sem erros é porque no *counter-matrix* proposto não existe colisões de *hash* como acontece com os *bitmaps*. A falta de colisões acontece pois, como dito na Seção 5.1, o algoritmo GeraMatriz só pode ser aplicado em redes onde é possível identificar os vértices de ingresso e egresso a partir do cabeçalho do pacote, logo a contagem de pacotes por par ingresso e egresso é feita sem erros.

Em seguida é apresentado o procedimento para a comparação entre as matrizes de tráfego geradas pelo algoritmo GeraMatriz e as matrizes de tráfego estimadas pelo *Bitmap Based Algorithm* proposto por Zhao et al.(22) e apresentado no Capítulo 4. A principal diferença entre os algoritmos GeraMatriz e *Bitmap Based Algorithm* está na forma como o tamanho da memória de trabalho é gerenciada. No algoritmo GeraMatriz o tamanho do *counter-matrix* é dependente de $|V|$, ou seja, em redes com grande quantidade de nós a memória dos roteadores se torna um gargalo. No *Bitmap Based Algorithm* o tamanho do *bitmap* é um parâmetro definido pelo programador, ou seja, que pode ser ajustado de acordo com a memória disponível. Em compensação, o tamanho do *bitmap* está diretamente relacionado com a precisão da estimativa da matriz de tráfego.

Como os algoritmos se diferenciam na forma como a memória de trabalho é definida, para uma avaliação justa, certificou-se de que o mesmo espaço de memória foi utilizado por ambos os algoritmos. Essa condição é importante pois, conforme já discutido, o tamanho da memória alocado para os observadores é um ponto crítico dos roteadores. Para apresentação

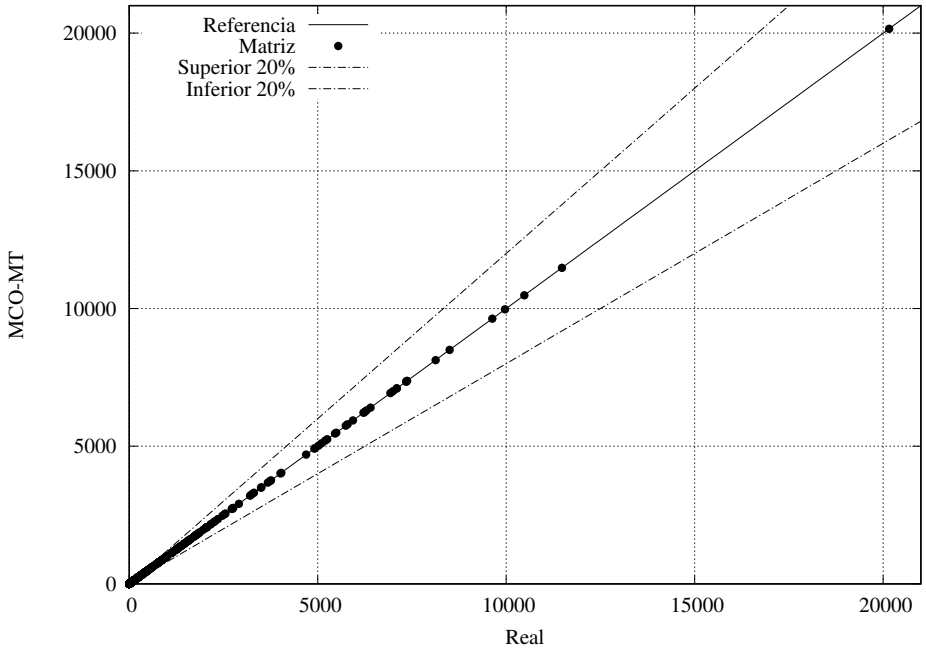


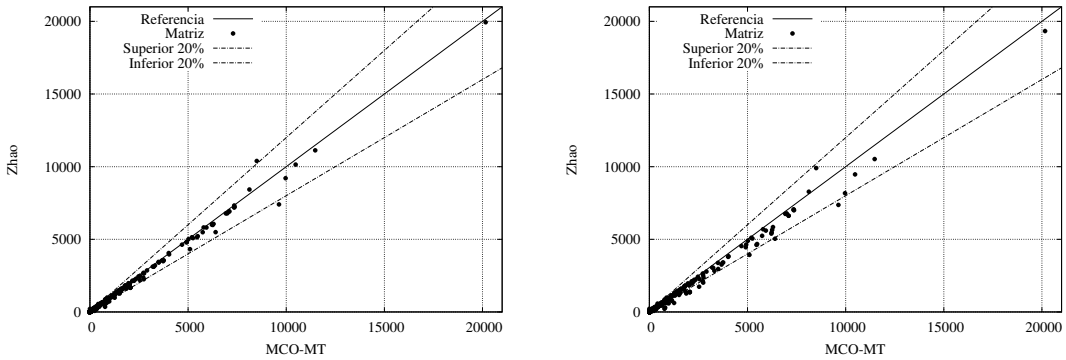
Figura 9 – Comparação entre a matriz de tráfego gerada pelo algoritmo GeraMatriz em relação à matriz de tráfego real para o AS-3257.

da comparação entre os algoritmos foi mantida a topologia do AS-3257 com $|V| = 41$. Dessa maneira, um *counter-matrix* do tipo `int` consome $41 \times 41 \times 4$ *bytes* (6,57 kB), da mesma forma, o tamanho do *bitmap* utilizado foi de 6,57 kB. Os limites de acessos aos *bitmaps* foram iguais a 10%, 30% e 50% do tamanho total dos *bitmaps*, dado que menores limites de acesso levam a melhores resultados conforme mostrado no Capítulo 4.

Nas Figuras 10(a), 10(b) e 10(c), os pontos representam a quantidade de pacotes entre cada par ingresso e egresso estimados pelo *Bitmap Based Algorithm*. Quanto mais próximo da reta Referencia, mais precisa foi a estimativa em relação à quantidade de pacotes na matriz de tráfego do algoritmo GeraMatriz. As linhas tracejadas representam um erro de $\pm 20\%$ no quantitativo de pacotes da matriz gerada, e foram plotados para servirem de referência do nível de erro obtido com o *Bitmap Based Algorithm*. As Figuras 10(a), 10(b) e 10(c) apresentam os resultados obtidos para 3 (três) diferentes configurações de *bitmaps*, respectivamente, com limites de acessos iguais a 10%, 30% e 50% do tamanho total dos *bitmaps*, fixado em 6,57 kB neste AS.

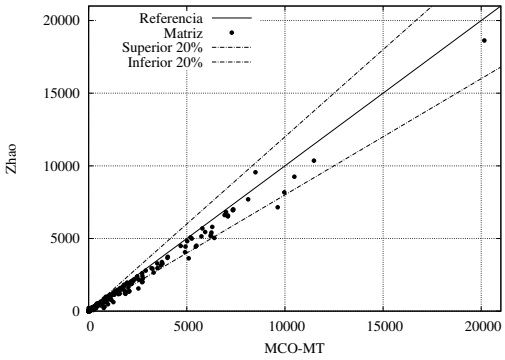
A partir dos resultados apresentados nas Figuras 10(a), 10(b) e 10(c) é possível verificar que o *Bitmap Based Algorithm* estima matrizes de tráfego próximas da matrizes geradas pelo algoritmo GeraMatriz, porém com observadores em todos os nós. Conforme verificado no Capítulo 4, o *Bitmap Based Algorithm* é mais preciso quanto menor for o limite de acessos. Todas essas conclusões a respeito do AS-3257 se repetiram nas outras 7 topologias de rede simuladas.

A partir dos resultados apresentados neste capítulo, pode-se confirmar que o



(a) *Bitmaps* - Limite 10%.

(b) *Bitmaps* - Limite 30%.



(c) *Bitmaps* - Limite 50%.

Figura 10 – Comparação entre as matrizes estimadas e a matriz real para o AS-3257.

algoritmo GeraMatriz reproduz a matriz de tráfego real sem erros utilizando-se de um conjunto reduzido de observadores e, em cada observador, as estruturas de dados do tipo *counter-matrix* ocupam exatamente o mesmo espaço em memória nos roteadores que os *bitmaps* propostos no *Bitmap Based Algorithm* de (22).

5.3 Conclusões

A principal contribuição desse capítulo foi mostrar que é possível gerar matrizes de tráfego usando estruturas de dados do tipo *counter-matrix* com um conjunto reduzido de observadores. Nesse contexto, foi apresentado o problema de Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego (MCO-MT) e o algoritmo GeraMatriz para a geração da matriz de tráfego a partir dos dados coletados por um conjunto reduzido de nós observadores por meio dos *counter-matrix*.

A técnica foi avaliada usando *traces* reais de pacotes e topologias de sistemas autônomos da Internet obtidos no CAIDA. Os resultados mostraram que a abordagem apresentada consegue gerar matrizes de tráfego sem nenhuma perda de precisão, desde que sejam obedecidos as seguintes restrições impostas pelo modelo adotado: i) o tamanho da memória de trabalho disponível deve ser grande o suficiente para armazenar um *counter-*

matrix, que cresce em função do número de vértices da rede e que ii) os nós observadores conheçam os dispositivos de ingresso e egresso na rede à partir do cabeçalho do pacote.

No Capítulo 6 a seguir, serão discutidos algoritmos que podem ser implementados no módulo *CMO-MT Solver Module* para encontrar um conjunto reduzido de observadores.

6 Algoritmos para o MCO-MT

Conforme discutido no Capítulo 5 o problema de Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego (MCO-MT) é resolvido no *MCO-MT Solver Module* do algoritmo GeraMatriz. O MCO-MT consiste em minimizar a quantidade de observadores instalados na rede, ou seja, minimizar o tamanho do conjunto de observadores necessários para gerar matrizes de tráfego utilizando estruturas de dados probabilísticas. Foi mostrado no Capítulo 5 que com um conjunto reduzido de vértices observadores e o conjunto de caminhos entre os pares de ingressos e egressos, o algoritmo GeraMatriz é capaz de monitorar e gerar a matriz de tráfego de uma rede de computadores.

Conforme Capítulo 5, o MCO-MT pode ser definido formalmente como: dados uma topologia de rede definida como um grafo $G = (V,A)$, o conjunto de vértices de ingresso $V^I \subseteq V$ e o conjunto de vértices de egresso $V^E \subseteq V$, encontrar o conjunto de caminhos $\mathcal{C}$ que conecta todos os pares de vértices ingresso-egresso e um conjunto de vértices observadores V^{O^*} capaz de monitorar todos os caminhos encontrados tal que o tamanho do conjunto $|V^{O^*}|$ seja minimizado. O conjunto de vértices observadores V^{O^*} é um conjunto reduzido de observadores em relação ao conjunto completo de observadores que utiliza todos os vértices ingresso e egresso da rede. A estratégia adotada para resolver o MCO-MT consiste em dividir o problema em duas partes: primeiramente é gerado o conjunto de caminhos entre os vértices ingresso-egresso $\mathcal{C}$ e em seguida é encontrado o conjunto de observadores V^{O^*} que cobrem o conjunto de caminhos $\mathcal{C}$.

Para gerar o conjunto de caminhos $\mathcal{C}$ foram utilizados algoritmos baseados no algoritmo de Dijkstra(36). Como podem existir diferentes caminhos entre um par de vértices $[u, v]$ em um grafo, a escolha do conjunto de observadores V^{O^*} depende dos caminhos escolhidos. Por exemplo, a Figura 11 mostra um grafo com os vértices $V = \{0,1,...,4\}$ conectados pelas arestas exibidas como linhas entre eles. Se for utilizado um conjunto de caminhos contendo os menores caminhos entre os pares de vértices (menor quantidade de saltos, no caso de arestas com peso unitário), o conjunto mínimo de observadores será $V^{O^*} = \{0, 2, 4\}$ (ou $V^{O^*} = \{1, 2, 3\}$), com cardinalidade $|V^{O^*}| = 3$. Por outro lado, se o conjunto de caminhos for gerado partindo-se de uma árvore com raiz no vértice 2, o conjunto mínimo de observadores passa a ser $V^{O^*} = \{2\}$ com cardinalidade $|V^{O^*}| = 1$.

O problema de encontrar o conjunto reduzido de observadores V^{O^*} dado o conjunto de caminhos $\mathcal{C}$ foi modelado e resolvido como um problema de Minimização de Cobertura de Conjuntos (*Minimum Set Cover Problem* - MSCP), que é um importante problema da classe NP-Completo (17, 42). A definição do *MSCP* é a seguinte: dados um conjunto S de m elementos e uma coleção $\mathcal{K}$ de n conjuntos destes elementos, encontrar o subconjunto

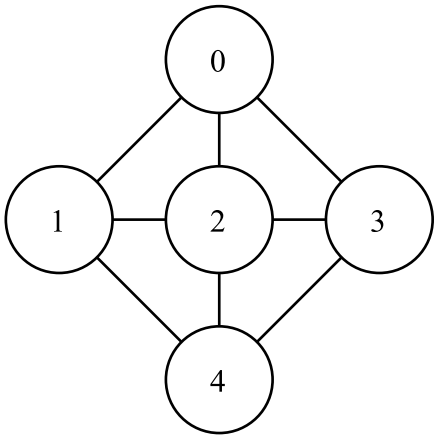


Figura 11 – Exemplo da escolha do conjunto de caminhos

$S^* \subseteq S$, de cardinalidade $|S^*|$ mínima, tal que $S^* \cap K \neq \emptyset$, para todo $K \in \mathcal{K}$. Isto é, cada subconjunto $K \in \mathcal{K}$ deve possuir, pelo menos, um elemento $s \in S^*$. Dessa maneira, encontrar o conjunto reduzido de observadores V^{O^*} para monitorar o conjunto de caminhos $\mathcal{C}$ é definido como um *MSCP* da seguinte maneira: dados o conjunto V de $|V|$ vértices e o conjunto de caminhos $\mathcal{C}$ de $|\mathcal{C}| \leq |V|(|V|-1)$ sequencias destes vértices, encontrar o subconjunto $V^{O^*} \subseteq V$, de cardinalidade $|V^{O^*}|$ mínima, tal que $V^{O^*} \cap C \neq \emptyset$, para todo $C \in \mathcal{C}$. Isto é, cada caminho $C \in \mathcal{C}$ deve possuir, pelo menos, um vértice observador $v \in V^{O^*}$.

Neste capítulo serão apresentados dois algoritmos para resolver o MCO-MT: um algoritmo guloso e um algoritmo GRASP (*Greedy Randomized Adaptative Search Procedure*) (18, 19), denominados nessa dissertação de SMV e GSMV, respectivamente. Os nomes dos algoritmos foram dados em referência aos nomes dos autores nos artigos onde os algoritmos foram inicialmente apresentados (20, 21). O restante do capítulo está organizado da seguinte forma:

- a Seção 6.1 propõe um algoritmo guloso para resolver o MCO-MT;
- a Seção 6.2 propõe um algoritmo GRASP para resolver o MCO-MT;
- a Seção 6.3 apresenta extensos resultados computacionais;
- a Seção 6.4 conclui o capítulo.

6.1 Algoritmo Guloso para o MCO-MT

Nesta seção será apresentada a estratégia adotada pelo algoritmo guloso. Nesta dissertação, o algoritmo é denominado de SMV em referência aos autores do artigo onde o algoritmo foi publicado (20).

A geração do conjunto de caminhos $\mathcal{C}$ é realizada pela aplicação do algoritmo de Dijkstra(36) para cada vértice do conjunto V^I. Dessa maneira, o algoritmo guloso considera o menor caminho entre dois vértices, o que é coerente com protocolos de roteamento bastante conhecidos, tais como OSPF (*Open Shortest-Path First*) (35) e IS-IS (*Intermediate System to Intermediate System*) (43), conforme já discutido no Capítulo 3.

Conforme já discutido, minimizar o conjunto de observadores que monitoram o conjunto de caminhos $\mathcal{C}$ é um problema NP-Difícil. Dessa maneira, o algoritmo guloso proposto utiliza o algoritmo de Chvatal(44), que é uma heurística gulosa que consegue bons resultados para o MSCP(*Minimum Set Cover Problem*).

O algoritmo guloso SMV pode ser visto no Algoritmo 5. Ele recebe como entrada uma topologia de rede definida como um grafo $G = (V,A)$, o conjunto de vértices de ingresso V^I e o conjunto de vértices de egresso V^E e tem como saída o conjunto de caminhos $\mathcal{C}$ e o conjunto reduzido de vértices observadores V^{O^*}.

Algoritmo 5: Algoritmo SMV

Entrada: Grafo $G(V,A)$
Entrada: Conjunto de vértices de ingresso V^I
Entrada: Conjunto de vértices de egresso V^E
Saída: Conjunto de caminhos $\mathcal{C}$
Saída: Conjunto de vértices observadores V^O

```
1 início
2 |  C ← Dijkstra(G, V^I, V^E)
3 |  V^O ← Chvatal(C, V)
4 |  retorna C, V^O
5 fim
```

Na linha 2, o conjunto de caminhos $\mathcal{C}$ é gerado pela função `Dijkstra` que executa o algoritmo de Dijkstra para encontrar os caminhos entre os pares de ingresso e egresso. Na linha 3, o conjunto de vértices observadores V^{O^*} é calculado pela função `Chvatal` que executa o guloso baseado no algoritmo de Chvatal, mostrado no Algoritmo 6. Por fim, o algoritmo retorna $\mathcal{C}$ e V^{O^*} na linha 4.

O algoritmo de Chvatal, mostrado no Algoritmo 6 recebe como entrada um conjunto de caminhos $\mathcal{C}$, neste caso calculado pela função `Dijkstra` do Algoritmo 5, e o conjunto de vértices V da topologia de rede $G(V,A)$.

O Algoritmo 6 inicia na linha 2 onde o conjunto de vértices observadores V^{O^*} é inicializado como um conjunto vazio. O laço da linha 3 até a linha 7 executa enquanto o conjunto de caminhos não é vazio. Na linha 4, a função `EscolheObservador` recebe o conjunto $V \setminus V^O$ de vértices que ainda não são observadores e o conjunto $\mathcal{C}$ de caminhos ainda não cobertos por pelo menos um vértice observador. A função `EscolheObservador` seleciona do conjunto $V \setminus V^O$ o vértice contido no maior número de caminhos de $\mathcal{C}$, isto é, seleciona o vértice que cobre a maior quantidade de caminhos ainda não cobertos.

Algoritmo 6: Algoritmo Chvatal

Entrada: Conjunto de caminhos entre os pares de ingresso-egresso $\mathcal{C}$
Entrada: Conjunto de vértices V da topologia de rede $G(V,A)$
Saída: Conjunto de vértices observadores V^O que cobrem os caminhos $\mathcal{C}$

```
1 início
2     V^O ← ∅
3     enquanto C ≠ ∅ faça
4         v_o ← EscolheObservador(V \ V^O, C)
5         C ← RemoveCaminhos(C, v_o)
6         V^O ← V^O ∪ {v_o}
7     fim enqto
8     retorna O
9 fim
```

O vértice selecionado pela função `EscolheObservador` é retornado como o novo vértice observador v_o. Em seguida, o conjunto $\mathcal{C}$ de caminhos ainda não cobertos é atualizado pela remoção dos caminhos que contém o novo observador v_o na linha 5 e o conjunto de vértices observadores V^O é atualizado com a inclusão do novo vértice observador v_o na linha 6. O conjunto final de vértices observadores é retornado na linha 8.

Os resultados obtidos com o algoritmo SMV são mostrados na Seção 6.3 e mostram que o algoritmo consegue reduzir a quantidade de observadores mantendo os menores caminhos na rede. Conforme discutido no início deste capítulo, a escolha dos caminhos interfere na escolha dos observadores. Como os caminhos utilizados pelo SMV já são os menores caminhos, na Seção 6.2 a seguir será adotada uma estratégia para variar o conjunto de caminhos escolhidos, afim de se diminuir ainda mais o conjunto de observadores na rede.

6.2 GRASP para o CMO-MT

Nesta seção será apresentada a estratégia adotada pelo algoritmo GRASP. Nesta dissertação, o algoritmo é denominado de GSMV em referência aos autores do artigo onde o algoritmo foi publicado (21).

Um GRASP (*Greedy Randomized Adaptive Search Procedure*) (18), procedimento de busca gulosa, aleatória e adaptativa, pode ser visto como uma metaheurística que captura boas qualidades de um algoritmo guloso e de procedimentos de construção aleatórios (45).

O GRASP é um método iterativo de múltiplas partidas no qual cada iteração consiste de dois passos: um passo de partida e um passo de melhoramento. No passo de partida, utiliza-se um algoritmo construtivo aleatório para se obter uma solução viável, não necessariamente a melhor. A cada iteração do GRASP, o algoritmo construtivo aleatório é capaz de obter uma nova solução diferente da anterior. No passo de melhoramento,

utiliza-se uma estratégia de busca local para melhorar a qualidade da solução obtida no passo de partida. A melhor solução encontrada em todas as iterações do GRASP é retornada como resultado.

Dessa maneira, a estratégia geral adotada pelo GSMV é a de gerar diferentes conjuntos de caminhos na fase de construção aleatória. Para cada conjunto é aplicado o algoritmo de Chvatal(44) para encontrar o conjunto de observadores da rede. Por fim é aplicada uma busca local afim de se reduzir o conjunto de observadores encontrados.

O Algoritmo 7 mostra o pseudocódigo do GSMV para solucionar o problema MCO-MT. O algoritmo recebe como entrada uma topologia de rede definida como um grafo $G = (V,A)$, o conjunto de vértices de ingresso V^I e o conjunto de vértices de egresso V^E. O algoritmo retorna o menor conjunto reduzido de vértices observadores encontrado V^{O^*} e o conjunto de caminhos $\mathcal{C}^*$ cobertos pelo conjunto reduzido de observadores V^{O^*}.

Algoritmo 7: Algoritmo GSMV

Entrada: Topologia da rede definida como um grafo $G = (V,A)$
Entrada: Conjunto de vértices de ingresso V^I
Entrada: Conjunto de vértices de egresso V^E
Saída: Menor conjunto reduzido de vértices observadores encontrado V^{O^*}
Saída: Conjunto de caminhos $\mathcal{C}^*$ cobertos pelo menor conjunto reduzido de vértices observadores encontrado

```
1  início
2      V^O* ← V
3      C* ←Dijkstra(G, V^I, V^E)
4      enquanto crit_parada faça
5          V^O, C ← SMV-A(G, V^I, V^E)
6          V^O ← BuscaLocal(V^O, C)
7          se |V^O| < |V^O*| então
8              V^O*, C* ← V^O, C
9          fim se
10     fim enqto
11     retorna V^O*, C*
12 fim
```

O Algoritmo 7 se inicia na linha 2, onde o menor conjunto de vértices observadores encontrado V^{O^*} é inicializado como o conjunto de vértices do grafo e na linha 3, o conjunto de caminhos cobertos pelo menor conjunto de vértices observadores $\mathcal{C}^*$ é inicializado como um conjunto de menores caminhos entre os pares de ingressos e egressos, dada pela aplicação do algoritmo de Dijkstra(36).

O laço da linha 4 até a linha 10 implementa o procedimento iterativo de busca aleatorizada. O laço é executado enquanto o critério de parada *crit_parada* não é alcançado. Nesta dissertação o *crit_parada* é definido como a quantidade de iterações em que a melhor solução não foi atualizada. Na linha 5, a estratégia de construção aleatória computa o conjunto de caminhos $\mathcal{C}$. A função `SMV-A()`, apresentada adiante no Algoritmo 8 da

Seção 6.2.1, garante que existe um único caminho $C_{[v_i,v_e]}$ para cada par de vértices ingresso-egresso v_i,$v_e \in V$. Diferente do algoritmo SMV apresentado na Seção 6.1, que utiliza os menores caminhos definidos pelo algoritmo de Dijkstra entre os vértices, os caminhos da estratégia de construção aleatória não são necessariamente os menores, podendo ser quaisquer caminhos que liguem os pares de vértices. Além disso, a estratégia aleatória computa também o conjunto de observadores V^O que cubram todos os caminhos de $\mathcal{C}$. Uma discussão detalhada de como a estratégia aleatória funciona ocorrerá na Seção 6.2.1.

Na linha 6 do Algoritmo 7, é executada a estratégia de busca local, apresentada adiante no Algoritmo 10 da Seção 6.2.2, com o objetivo de reduzir o conjunto vértices observadores V^O encontrado na estratégia construtiva. Na linha 7, é verificado se o conjunto de vértices observadores encontrados V^O na iteração atual é menor do que o menor conjunto de vértices observadores V^{O^*} encontrado até o momento. Caso o conjunto de vértices observadores V^O seja menor, o menor conjunto reduzido de observadores V^{O^*} e o conjunto de caminhos que ele cobre $\mathcal{C}^*$ são atualizados na linha 8. Por fim, o GSMV retorna o menor conjunto reduzido de vértices observadores V^{O^*} encontrado e o respectivo conjunto de caminhos $\mathcal{C}^*$ na linha 11.

A seguir, é feita a proposta de uma estratégia aleatória para solucionar o MCO-MT é feita na Subseção 6.2.1. Na Subseção 6.2.2 é discutida uma proposta de busca local para o problema.

6.2.1 Algoritmo Construtivo Aleatório

Nesta dissertação, o algoritmo construtivo aleatório implementado é uma aleatorização do algoritmo SMV apresentado na Seção 6.1. Dessa maneira, a estratégia adotada é a mesma do algoritmo SMV: primeiro gerar um conjunto de caminhos e em seguida calcular o conjunto de vértices observadores que cobrem os caminhos encontrados no primeiro passo. Conforme já discutido na Seção 6.2, o algoritmo construtivo aleatório não utiliza os menores caminhos entre os pares de vértices ingresso-egresso. Ao invés disso, o algoritmo construtivo aleatório baseia-se na geração de caminhos considerando diferentes árvores contidas em um grafo. Nesta dissertação, a construção aleatória é nomeada como SMV-A e pode ser vista no Algoritmo 8. Ele recebe como entrada uma topologia de rede definida como um grafo $G = (V,A)$, o conjunto de vértices de ingresso V^I e o conjunto de vértices de egresso V^E. O Algoritmo 8 tem como saída um conjunto reduzido de observadores V^O e o conjunto de caminhos $\mathcal{C}$.

Na linha 2, o SMV-A constrói o conjunto de caminhos $\mathcal{C}$ a partir de G. Na linha 3, o SMV-A obtém o conjunto de observadores V^O utilizando o conjunto de caminhos $\mathcal{C}$ obtidos no passo anterior. Por fim, o algoritmo retorna $\mathcal{C}$ e V^O na linha 4.

O primeiro passo do SMV-A, visto na linha 2 do Algoritmo 8, é o Algoritmo 9

Algoritmo 8: Algoritmo SMV-A

Entrada: Topologia de rede definida como um grafo $G = (V,A)$
Entrada: Conjunto de vértices de ingresso V^I
Entrada: Conjunto de vértices de egresso V^E
Saída: Conjunto de caminhos $\mathcal{C}$
Saída: Conjunto de vértices observadores V^O

```
1 início
2 |  C ← EscolheCaminhos(G, V^I, V^E)
3 |  V^O ← Chvatal(C, V)
4 |  retorna V^O, C
5 fim
```

nomeado como `EscolheCaminhos`. Sua entrada é um grafo $G = (V,A)$, o conjunto de vértices de ingresso V^I e o conjunto de vértices de egresso V^E. Sua saída é um conjunto de caminhos entre os pares de ingressos e egressos $\mathcal{C}$. A ideia geral desse algoritmo é de utilizar o conjunto de caminhos de uma árvore gerada aleatoriamente. Os motivos para isso são: i) em uma árvore só existe um caminho entre um par de nós e ii) os vértices não folha da árvore participam de muitos caminhos na rede, o que os torna bons candidatos a vértices observadores tendendo a diminuir o conjunto de vértices observadores escolhidos.

Na linha 2 do Algoritmo 9 é computada uma árvore aleatória $T = (V_t,A_t)$, onde $V_t = V$ e $A_t \subseteq A$. Para isso é utilizada uma variação aleatória do algoritmo de Dijkstra(36) para calcular T. O algoritmo de Dijkstra original escolhe, dentre os vértices candidatos, o vértice com a menor distância até a raiz da árvore. Um vértice é considerado candidato se ele ainda não pertencer a algum caminho partindo da raiz. A distância de um vértice é denominada como $d(v)$, que representa a quantidade de saltos da raiz até o vértice v.

No Dijkstra aleatório, acrescenta-se uma Lista de Candidatos Restrita (LCR). A LCR é formada por todos os vértices candidatos tais que

$$d(v) \leq d(v)_{min} + \alpha(d(v)_{max} - d(v)_{min}) \tag{6.1}$$

onde $\alpha \in [0,1]$, $d(v)_{min} = \min\{d(v) : v \in V$ e v é um vértice candidato$\}$ e $d(v)_{max} = \max\{d(v) : v \in V$ e v é um vértice candidato$\}$. Além da LCR, o vértice raiz também é escolhido aleatoriamente.

Após calcular uma árvore, o algoritmo `EscolheCaminhos` constrói um conjunto de caminhos $\mathcal{C}$ para todo par de vértices (v_i,v_e) tal que $v_i \in V^I$ e $v_e \in V^E$. Isso ocorre no Algoritmo 9 da linha 3 até a linha 7. A função `MontaCaminho` funciona da seguinte maneira: constrói-se um caminho $C_{[v_i,r]}$ partindo do vértice v_i até raiz r de T. Depois, a estratégia monta um segundo caminho $C_{[v_e,v]}$ partindo de v_e até um vértice v qualquer pertence ao caminho $C_{[v_i,r]}$. Finalmente a estratégia une os dois caminhos gerados $C_{[v_i,v]}$ e $C_{[v,v_e]}$, formando o caminho $C_{[v_i,v_e]}$. Por fim, o algoritmo `EscolheCaminhos` retorna o conjunto de caminhos $\mathcal{C}$ na linha 8.

Algoritmo 9: Algoritmo EscolheCaminhos

Entrada: Grafo $G = (V,A)$
Entrada: Conjunto de vértices de ingresso V^I
Entrada: Conjunto de vértices de egresso V^E
Saída: Conjunto de caminhos $\mathcal{C}$

```
1 início
2     T ← DijkstraAleatorio(G)
3     para todo v_i ∈ V^I faça
4         para todo v_e ∈ V^E faça
5             C ← MontaCaminho(T,v_i,v_e)
6         fim para todo
7     fim para todo
8     retorna C
9 fim
```

O segundo passo da construção aleatória, visto na linha 3 do Algoritmo 8, é encontrado um cojunto de observadores para o conjunto de caminhos $\mathcal{C}$ através da aplicação do algoritmo de Chvatal, que já foi apresentado no algoritmo 5 da Seção 6.1.

Por fim, na linha 4 do Algoritmo 8 é retornado um conjunto de caminhos e o conjunto de observadores encontrados. A seguir, na Subseção 6.2.2 é apresentada a estratégia de busca local para tentar reduzir o conjunto de observadores encontrado pelo algoritmo SMV-A.

6.2.2 Busca Local

A busca local tem por objetivo melhorar uma solução. Para o problema proposto, a busca local tenta reduzir o número de vértices observadores, isto é, encontrar um novo conjunto de vértices observadores com tamanho menor do que a solução obtida pelo passo de partida.

O algoritmo de busca local utiliza a vizinhança de *1-flip* como descrita no trabalho de Bilal, Galinier e Guibault(46), onde, em cada movimento, um único vértice observador pode ser inserido ou removido de acordo com a avaliação da função objetivo. A estratégia da busca local implementada é a melhor aprimorante, ou seja, avalia-se todas as soluções na vizinhança da solução corrente e retorna a melhor encontrada.

Neste trabalho, a busca local é nomeada como BL e pode ser vista no Algoritmo 10. Ele recebe como entrada o conjunto de vértices V, o conjunto de observadores V^O e o conjunto de caminhos $\mathcal{C}$ e, caso encontre, retorna como saída um novo conjunto V^O com tamanho menor que o conjunto inicial. Caso contrário, o algoritmo retorna o conjunto inicial.

O algoritmo BL funciona como segue. O laço da linha 2 até a linha 7 é executado até que o algoritmo encontre um ótimo local. Na linha 3, a função `MelhorCandidato` busca

Algoritmo 10: Algoritmo BL

Entrada: Conjunto de vértices V
Entrada: Conjunto de vértices observadores V^O
Entrada: Conjunto de caminhos $\mathcal{C}$
Saída: Novo conjunto de vértices observadores V^O

```
1 início
2     repita
3         v_c ← MelhorCandidato(V,V^O,C)
4         se v_c ≠ null então
5             V^O ← FlipCandidato(V,V^O, v_c)
6         fim se
7     até v_c = null;
8     retorna V^O
9 fim
```

qual melhor vértice a ser feito um *flip* (melhor solução na vizinhança da solução corrente V^O), isto é, qual vértice que, ao ser inserido ou removido do conjunto V^O, apresenta melhor ganho da função objetivo. Para maiores informações sobre a função `MelhorCandidato` procurar no trabalho de Bilal, Galinier e Guibault(46). A função `MelhorCanditado` retorna um valor diferente de *null* como vértice candidato $v_c \in V$ quando uma solução melhor é encontrada na vizinhança. O condicional da linha 4 verifica o vértice candidato v_c. Se foi encontrada uma solução vizinha aprimorante, v_c é diferente de *null* e é executado a linha 5, onde é feito o *flip* do vértice candidato v_c e o novo conjunto de observadores é armazenado em V^O. Caso contrário, isto é, quando v_c for igual a *null*, significa que foi encontrado um ótimo local e o laço da linha 2 até a linha 7 finaliza. Finalmente, a melhor solução obtida é retornada na linha 8.

6.2.3 Ajustes dos parâmetros do GSMV

O parâmetro α é utilizado pelo GSMV para controlar a LCR dentro da construção aleatória (ver Seção 6.2.1). Ele pode assumir valores no intervalo [0.0,1.0]. Se $\alpha = 0.0$, então a escolha dos elementos da LCR se torna puramente gulosa. Se $\alpha = 1.0$ então a escolha dos elementos da LCR se torna puramente aleatória.

A utilização de um único valor fixo para α dificulta encontrar soluções de boa qualidade que podem ser encontradas quando se utiliza outros valores para α. Isto é mostrado em Prais e Ribeiro(47) e os autores propuseram uma estratégia reativa para o ajuste do α automaticamente.

Na estratégia proposta, o parâmetro α é selecionado dentro de um conjunto discreto de valores $X = \{\alpha_1,\alpha_2,\ldots,\alpha_n\}$. Probabilidades P_i são associadas a cada α_i com valores iniciais iguais a $P_i = 1/n$, $i = 1,\ldots,n$. Além disso, a distribuição de probabilidade é atualizada regularmente a cada bloco de repetição fixado em B iterações seguindo as

seguintes equações:

$$P_i = q_i / \sum_{j=1}^{n} q_j \tag{6.2}$$

onde

$$q_i = (1/Y_i)^{\delta} \tag{6.3}$$

O valor Y_i é a média das soluções encontradas utilizando o parâmetro α com o valor α_i. O expoente $\delta > 0$ é utilizado para atenuar a atualização dos valores das probabilidades P_i. O valor encontrado para este parâmetro na literatura é $\delta = 8$ (47).

6.3 Avaliações e Resultados

Os experimentos computacionais realizados para os algoritmos apresentados neste capítulo são apresentados nesta seção. Especificamente, foram avaliados os algoritmos SMV (Algoritmo 5), SMV-A (Algoritmo 8), BL (Algoritmo 10) e GSMV (Algoritmo 7). Foram realizados dois experimentos, caracterizados pelos conjuntos de entrada utilizados em cada experimento: o primeiro utilizando um conjunto de redes artificiais e o segundo um conjunto de redes reais. Todos os resultados foram gerados em um sistema com 8GB de memória RAM, processador Intel® Core™ i5-2400 de 3GHz e SO Ubuntu 16.04.

Os algoritmos foram implementados na linguagem de programação C++, devido a performance e ao domínio dos autores com essa linguagem. O código, bem como todos os resultados obtidos com a ferramenta se encontram disponíveis publicamente no endereço: <https://github.com/nerds-ufes/setcover-grasp>.

Nos testes realizados, todos os nós participam como ingresso e egresso, configurando o pior caso onde tem-se $|V|(|V|-1)$ pares ingresso-egresso. Os algoritmos foram configurados e executados de maneira independente. O SMV é equivalente ao algoritmo SMV-A quando executado com parâmetro α igual a zero, isto é, foram geradas árvores utilizando o algoritmo de Dijkstra(36) original. A busca local BL foi executada em cima dos resultados obtidos pelo SMV-A. Para o GSMV o critério de parada *crit_parada* foi definido empiricamente como 10000 iterações sem melhorias. A atualização da distribuição de probabilidades do parâmetro α foi atualizada a cada $B = 100$ iterações.

Para avaliar a qualidade dos caminhos utilizados pelos algoritmos SMV-A e GSMV, foi utilizada a métrica *Path Stretch*. Essa métrica calcula a razão entre os caminhos obtidos pelos algoritmos em relação aos menores caminhos do tráfego (48). Com essa métrica é

Tabela 4 – Comparação dos resultados obtidos pelos algoritmos, SMV, SMV-A, BL e GSMV para as redes artificiais.

	SMV		SMV-A		BL		GSMV	
$\|V\|$	Obser.	Dist.(%)	Obser.	Dist(%)	Obser.	Dist(%)	Obser.	Dist(%)
100	58,90	-41,10	31,70	-68,30	31,30	-68,70	**29,60**	**-70,40**
200	118,50	-40,75	68,50	-65,75	68,40	-65,80	**64,80**	**-67,60**
300	175,30	-41,57	103,70	-65,43	102,80	-65,73	**99,90**	**-66,70**
400	235,80	-41,05	139,90	-65,03	138,30	-65,43	**135,00**	**-66,25**
500	295,00	-41,00	174,40	-65,12	173,20	-65,36	**169,70**	**-66,06**

possível medir a quantidade de *links* adicionais necessários para fazer com que o tráfego passe pelo conjunto reduzido de vértices observadores. O algoritmo SMV não foi avaliado por essa métrica pois utiliza os menores caminhos definidos pelo algoritmo de Dijkstra. Os resultados do *Path Stretch* não foram exibidos para o algoritmo BL, pois os caminhos utilizados são os mesmos que os utilizados pelo algoritmo SMV-A.

O primeiro experimento foi realizado com as redes artificiais. Foi criado um conjunto de redes aleatórias com vértices V, $|V| \in \{100, 200, 300, 400, 500\}$. Para cada tamanho de V, foram geradas dez topologias diferentes, numeradas de 0 a 9, totalizando uma quantidade de cinquenta topologias. Os resultados apresentados mostram a média, para cada tamanho de V, das dez execuções independentes de cada algoritmo avaliado.

A Tabela 4 compara os resultados obtidos pelos algoritmos SMV, SMV-A, BL e GSMV para as redes artificiais. A primeira coluna mostra a quantidade de vértices das redes, que é a quantidade de vértices observadores necessários sem a resolução do MCO-MT. As colunas seguintes apresentam os resultados obtidos pelos algoritmos SMV, SMV-A, BL e GSMV, respectivamente. Para cada algoritmo, as informações são divididas em duas subcolunas, sendo a primeira subcoluna a quantidade média de observadores calculada pelos algoritmos e a segunda subcoluna a distância percentual média desses algoritmos em relação à quantidade de vértices na rede.

Analisando a Tabela 4, percebe-se que todos os algoritmos propostos neste trabalho (SMV, SMV-A, BL e GSMV) conseguiram reduzir a quantidade de observadores necessários para monitorar e gerar a Matriz de Tráfego em todas as redes artificiais, sendo que o GSMV conseguiu os melhores resultados. Esses resultados são exibidos graficamente na Figura 12. Nesta figura, o eixo horizontal representa o tamanho da rede em quantidade de vértices e o eixo vertical representa a quantidade média de observadores calculados pelos algoritmos.

A Figura 13 mostra o *Path Stretch* utilizando os algoritmos SMV-A e GSMV para a rede artificial com $|V| = 100$ de número 0. O comportamento verificado nesta rede representa bem o padrão da maioria das topologias artificiais testadas e o seu *Path Stretch* foi o mais adequado à visualização. Na Figura 13 o eixo horizontal representa o *Path Stretch* e o eixo vertical representa a *Cumulative Distribution Function*(CDF) (49) de

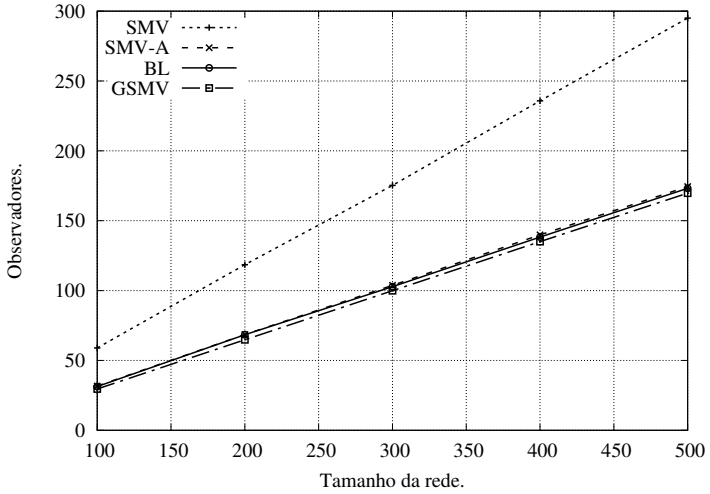


Figura 12 – Gráfico dos resultados obtidos pelos algoritmos, SMV, SMV-A, BL e GSMV para as redes artificiais.

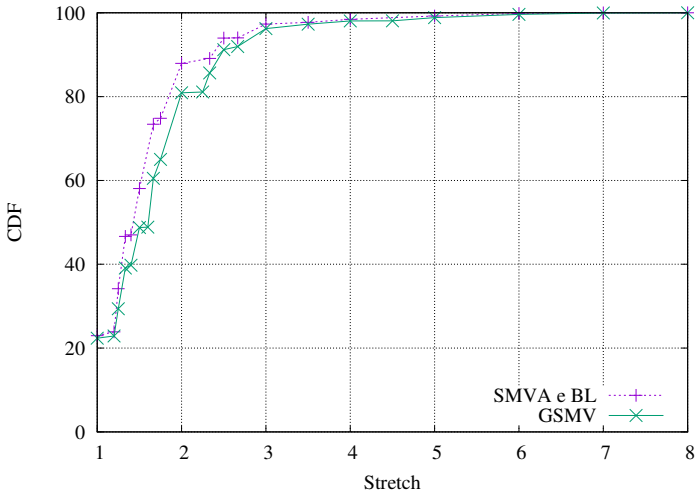


Figura 13 – Gráfico com o *PathStretch* da instância artificial com $|V| = 100$ de número 0

caminhos com o respectivo *Path Stretch*.

Analisando a Figura 13, percebe-se que ambos os algoritmos, SMV-A e GSMV, possuem um comportamento bem semelhante. Ambos possuem cerca de 20% dos caminhos utilizados com o mesmo tamanho dos menores caminhos da rede. O SMV-A é ligeiramente melhor, pois possui um CDF maior, até um *Path Stretch* de 3, quando ambos algoritmos, SMV-A e GSMV, se igualam e atingem um CDF em torno de 98%. Isto significa dizer que 98% dos caminhos utilizados pelos algoritmos, SMV-A e GSMV, possuem tamanhos até 3 vezes maiores que os menores caminhos da rede.

Com os resultados obtidos nesse primeiro experimento é possível afirmar que o fator determinante para a melhora dos resultados foi a utilização dos caminhos baseados em árvore ao invés dos menores caminhos calculados por Dijkstra, visto que o algoritmo guloso

SMV-A apresentou resultados melhores que o SMV em todas as baterias de testes. Foi mostrado também que o custo dessa melhora é uma piora dos caminhos gerados, indicando que existe um compromisso entre a redução do conjunto de observadores e a qualidade dos caminhos utilizados, isto é, para diminuir o tamanho do conjunto de observadores é necessário abrir mão dos menores caminhos na rede. Além disso pode-se afirmar que a estratégia de busca local funciona, mas não é efetiva, visto que os resultados obtidos pelos algoritmos BL e GSMV conseguem resultados muito próximos dos resultados obtidos pelo algoritmo SMV-A.

O segundo experimento foi realizado para redes reais. Foram utilizadas as mesmas topologias utilizadas nas avaliações realizadas nos Capítulos 4 e 5, obtidas diretamente do CAIDA (40). As topologias possuem tamanho que variam no intervalo [11,115] e seus respectivos nomes são: 1) Abilene; 2) AS-1221; 3) AS-1239; 4) AS-2914; 5) AS-3257; 6) AS-3356; 7) AS-7018 e 8) Geant.

A Tabela 5 compara os resultados obtidos pelos algoritmos SMV, SMV-A, BL e GSMV para redes reais. A primeira coluna mostra o nome e a quantidade de vértices de cada uma das redes, que é a quantidade de vértices observadores necessários sem a resolução do MCO-MT. As colunas seguintes apresentam os resultados obtidos pelos algoritmos SMV, SMV-A, BL e GSMV, respectivamente. Para cada algoritmo, as informações são divididas em duas subcolunas, sendo a primeira subcoluna a quantidade de observadores calculada pelos algoritmos e a segunda subcoluna a distância percentual dos algoritmos em relação à quantidade total de vértices na rede.

Tabela 5 – Comparação dos resultados obtidos pelos algoritmos, SMV, SMV-A, BL e GSMV para as redes reais.

ASes		SMV		SMV-A		BL		GSMV	
Nome	$\|V\|$	Obser.	Dist(%)	Obser.	Dist(%)	Obser.	Dist(%)	Obser.	Dist(%)
Abilene	11	7	-36,36	**4**	**-63,64**	**4**	**-63,64**	**4**	**-63,64**
Geant	22	12	-45,45	7	-68,18	7	-68,18	**6**	**-72,73**
AS-3257	41	16	-60,98	**10**	**-75,61**	**10**	**-75,61**	**10**	**-75,61**
AS-1221	44	**6**	**-86,36**	**6**	**-86,36**	**6**	**-86,36**	**6**	**-86,36**
AS-1239	52	22	-57,69	**15**	**-71,15**	**15**	**-71,15**	**15**	**-71,15**
AS-3356	63	20	-68,25	**13**	**-79,37**	**13**	**-79,37**	**13**	**-79,37**
AS-2914	70	29	-58,57	23	-67,14	23	-67,14	**21**	**-70,00**
AS-7018	115	21	-81,74	**15**	**-86,96**	**15**	**-86,96**	**15**	**-86,96**

Analisando a Tabela 5, percebe-se novamente que todos os algoritmos propostos neste trabalho (SMV, SMV-A, BL e GSMV) conseguiram reduzir a quantidade de observadores necessários para monitorar e gerar a Matriz de Tráfego em todas as redes reais avaliadas. Além disso, o algoritmo GSMV conseguiu alcançar todos os melhores resultados.

Para exemplificar a escolha dos conjuntos reduzidos de observadores, as Figuras 14, 15 e 16 apresentam a topologia da rede Abilene ($|V| = 11$), destacando o conjunto reduzido de vértices observadores selecionados pelos algoritmos SMV, SMV-A, BL e GSMV em vermelho, respectivamente. O comportamento verificado nesta rede representa bem o padrão

da maioria das topologias testadas e o sua topologia foi a mais adequada à visualização. Vale a pena ressaltar que, apesar do tamanho do conjunto reduzido de observadores ser igual entre os algoritmos SMV-A e GSMV, o conjunto reduzido de observadores é diferente, evidenciando a evidencia a natureza aleatória na geração de caminhos do algoritmo GSMV.

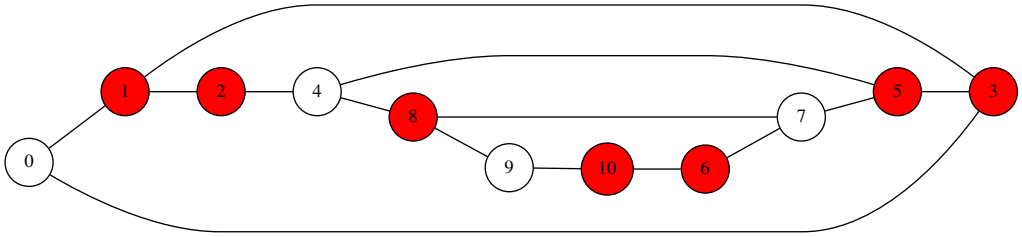


Figura 14 – Topologia da rede Abilene, com destaque em vermelho para o posicionamento dos 7 nós observadores selecionados pelo algoritmo SMV.

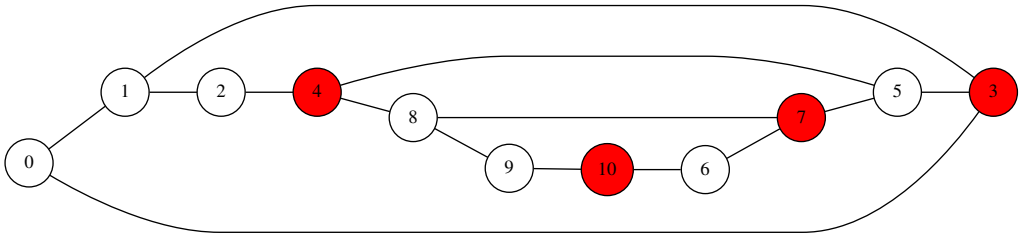


Figura 15 – Topologia da rede Abilene, com destaque em vermelho para o posicionamento dos 4 nós observadores selecionados pelo algoritmo SMV-A e BL.

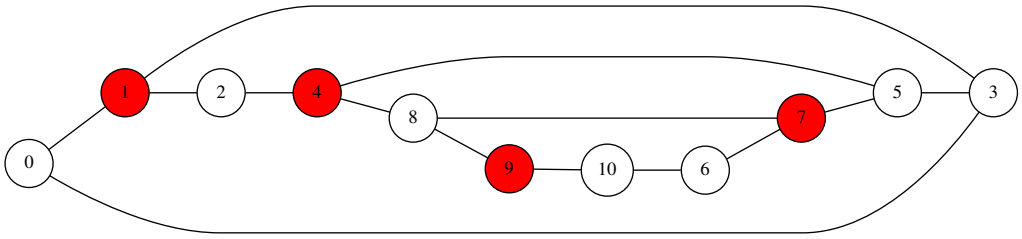


Figura 16 – Topologia da rede Abilene, com destaque em vermelho para o posicionamento dos 4 nós observadores selecionados pelo algoritmo GSMV.

Continuando a análise para a rede Abiliene, a Figura 17 mostra o *Path Stretch* utilizando os algoritmos SMV-A e GSMV para a rede real Abilene. Na Figura 17 o eixo horizontal representa o *Path Stretch* e o eixo vertical representa a CDF de caminhos com o respectivo *Path Stretch*.

Analisando a Figura 17, percebe-se que ambos os algoritmos, SMV-A e GSMV, possuem um comportamento bem semelhante. Ambos possuem cerca de 60% dos caminhos utilizados com o mesmo tamanho dos menores caminhos da rede. O GSMV tem uma melhor performance, pois os caminhos escolhidos pelo GSMV possuem tamanho até 4 vezes

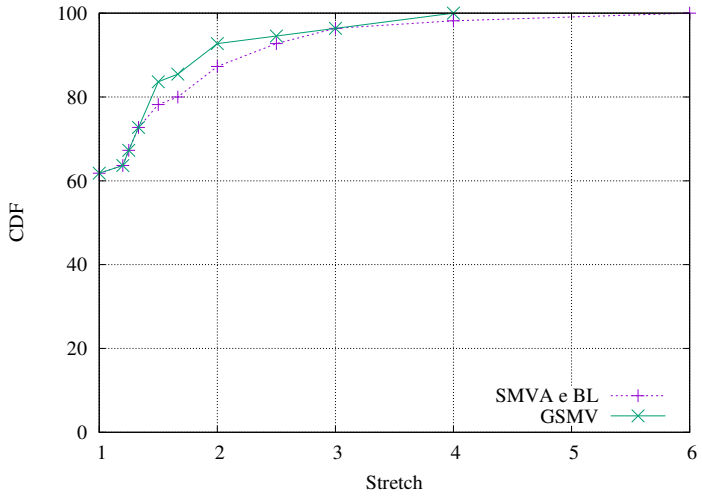


Figura 17 – Gráfico com o *PathStretch* da rede real Abilene

maior que os menores caminhos, enquanto os caminhos escolhidos pelo SMV-A possuem tamanho até 6 vezes maior que os menores caminhos.

Com os resultados obtidos no segundo experimento pode-se afirmar que o GSMV proposto funciona também para as redes reais. Além disso, os resultados reforçam a ideia de que o fator determinante para a melhora dos resultados foi a utilização dos caminhos baseados em árvore, principalmente pelo resultado obtido com a rede AS-1221.

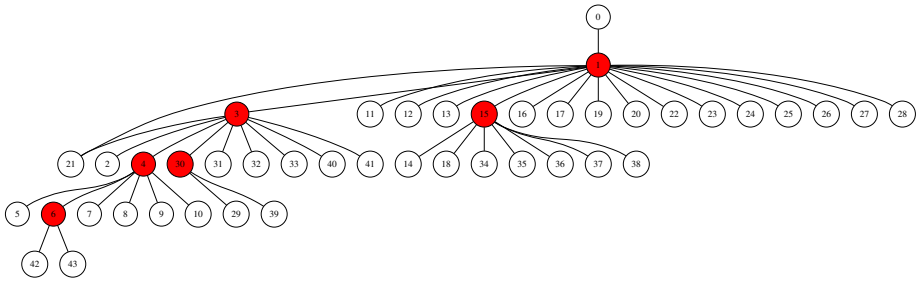


Figura 18 – Topologia do AS-1221, com destaque em vermelho para o posicionamento dos 6 vértices observadores selecionados

Para mostrar a característica da rede AS-1221 a Figura 18 mostra a topologia do AS-1221 que é praticamente uma árvore, apenas os *links* no vértice 21 violam as características de uma árvore. Os vértices observadores selecionados estão destacados em vermelho e os mesmos foram selecionados em todos os algoritmos (SMV, SMV-A, BL e GSMV). O fato da topologia ser uma árvore faz com que os caminhos gerados por todos os algoritmos são os mesmos, fazendo com que o conjunto de vértices observadores sejam os mesmos.

A escolha dos mesmos caminhos é mostrada na Figura 19. A Figura 19 mostra o *Path Stretch* utilizando os algoritmos SMV-A e GSMV para a rede real AS-1221. Na Figura 19 o eixo horizontal representa o *Path Stretch* e o eixo vertical representa a CDF

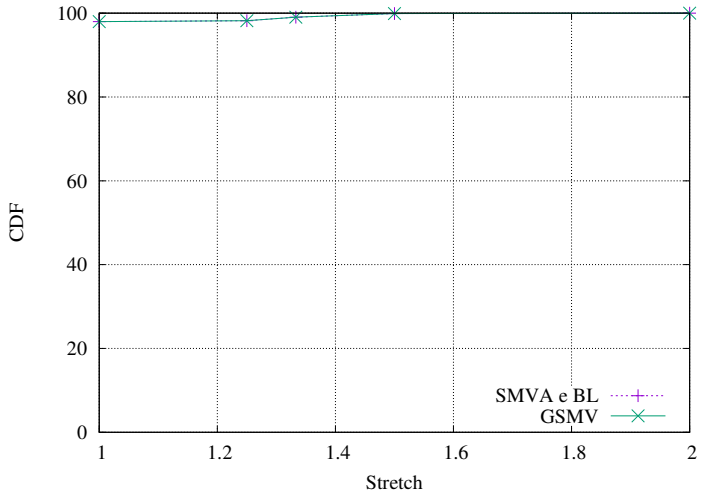


Figura 19 – Gráfico com o *PathStretch* da rede real AS-1221

de caminhos com o respectivo *Path Stretch*. Analisando esta figura, é possível perceber que os caminhos utilizados pelos algoritmos SMV-A e GSMV possuem o mesmo *Path Stretch*. Além disso cerca de 98% dos caminhos selecionados possuem tamanho igual aos menores caminhos da rede. Esses caminhos estão todos associados ao vértice 21, cujos *links* fogem ao padrão de uma árvore.

6.4 Conclusões

Neste capítulo foi estudado o problema do Conjunto Mínimo de Observadores para geração de Matrizes de Tráfego (CMO-MT). O problema foi modelado e resolvido como um problema NP-completo de Minimum Set Covering Problem (MSCP).

Para resolver esse problema, foram propostos dois algoritmos: um algoritmo guloso, denominado SMV, e a combinação do algoritmo guloso aleatorizado, denominado SMV-A, com uma busca local, denominada BL, para formar uma meta-heurística GRASP, denominado GSMV. Todos os algoritmos citados foram implementados, testados e comparados computacionalmente utilizando um conjunto de redes artificias e um conjunto de redes reais de sistemas autônomos da Internet, obtidos no CAIDA.

Com os experimentos realizados, mostrou-se que o GSMV conseguiu obter os melhores resultados (isto é, menores conjuntos de observadores) para as redes artificiais e reais. Além disso, os resultados obtidos permitem concluir que o fator determinante para a melhoria dos resultados foi a utilização de caminhos baseados em árvores, sendo que com a aleatorização destes caminhos o GSMV é capaz de conseguir bons resultados. Entretanto, a redução do conjunto de observadores utilizando caminhos baseados em árvores afeta a qualidade dos caminhos utilizados na rede. Isto indica que existe um compromisso a ser assumido entre a redução do conjunto de observadores e a qualidade dos caminhos na rede.

7 Conclusão e Trabalhos Futuros

A principal contribuição desta dissertação foi mostrar que é possível gerar matrizes de tráfego utilizando algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas utilizando apenas um conjunto reduzido de observadores em comparação ao conjunto completo de todos os ingressos e egressos utilizados como observadores.

Neste contexto, esta dissertação apresentou o algoritmo GeraMatriz que é um algoritmo de processamento de cadeia de dados baseados na estrutura de dados *counter-matrix*. Os resultados obtidos com o algoritmo GeraMatriz mostraram que o algoritmo consegue gerar matrizes de tráfego sem nenhuma perda de precisão, desde que as redes onde o algoritmo for aplicado respeitem as restrições impostas: i) tamanho da memória de trabalho cresce em função do número de dispositivos de ingresso e egresso e ii) os observadores da rede devem conseguir inferir os dispositivos de ingresso e egresso à partir dos cabeçalhos dos pacotes que trafegam na rede.

Para encontrar um conjunto reduzido de observadores na rede a ser utilizado pelo algoritmo GeraMatriz, esta dissertação apresentou o problema da Minimização do Conjunto de Observadores para Geração de Matrizes de Tráfego (MCO-MT). O problema foi modelado e resolvido como um problema NP-completo de Minimum Set Covering Problem (MSCP). Foram propostos dois algoritmos para resolver o problema: um algoritmo guloso, denominado SMV e um algoritmo GRASP, denominado GSMV. Com os resultados obtidos mostrou-se que ambos os algoritmos conseguiram reduzir o conjunto de observadores necessários para gerar matrizes de tráfego. Além disso, os resultados obtidos permitem concluir que o fator determinante para a melhoria dos resultados foi a utilização de caminhos baseados em árvores. Entretanto, a redução do conjunto de observadores utilizando caminhos baseados em árvores afeta a qualidade dos caminhos utilizados na rede. Isto indica que existe um compromisso a ser assumido entre a redução do conjunto de observadores e a qualidade dos caminhos na rede.

Para verificar os resultados dos algoritmos de processamento de cadeia de dados, esta dissertação apresentou a ferramenta de simulação BitMatrix que implementa dois algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticos: o algoritmo *Bitmap Based Algorithm*, proposto por Zhao et al.(22), e o algoritmo GeraMatriz. A ferramenta se provou de importância ao validar os algoritmos utilizando topologias e dados de tráfego reais retirados diretamente do CAIDA. Dessa maneira, a ferramenta validou que é possível utilizar algoritmos de processamento de cadeia de dados para geração de matrizes de tráfego. Além disso foi mostrado que com algumas melhorias, a BitMatrix

pode vir a se tornar uma importante ferramenta de simulação e ajustes de parâmetros dos algoritmos de processamento de cadeia de dados baseados em estruturas de dados probabilísticas.

Várias propostas de trabalhos futuros podem ser feitas a partir desta de dissertação. Com relação à ferramenta BitMatrix pretende-se, implementar novas funcionalidades para tornar a ferramenta mais robusta. Por exemplo a inclusão de um módulo de caracterização do conjunto de *traces* utilizados afim de ajudar a entender a influência dos parâmetros de configuração dos algoritmos.

Como trabalhos futuros em relação ao algoritmo GeraMatriz, pretende-se remover as restrições impostas aos tipos de rede onde o GeraMatriz pode ser aplicado e avaliar os impactos causados na geração das matrizes de tráfego. Outra proposta é a de se tentar adatar o GeraMatriz para ser utilizado com outras estruturas de dados probabilísticas, por exemplo, os *bitmaps* propostos por Zhao et al.(22).

Em relação aos algoritmos para resolver o problema MCO-MT, pretende-se avaliar os resultados do GMSV com a aleatorização do algoritmo de Chvatal(44) e a implementação de outras buscas locais com diferentes vizinhanças. Um outro trabalha interessante é transformar o MCO-MT em um problema multi-objetivo visando minimizar o conjunto de observadores e o *Path Stretch* dos caminhos escolhidos.

Referências

1 CISCO SYSTEMS INC. *Tráfego global de dados móveis crescerá quase 10 vezes entre 2014 e 2019*. [S.l.]. Disponível em: <http://www.cisco.com/c/pt_pt/about/press/news-archive-2015/20150203.html>. Acesso em: 26-11-2015. Citado na página 10.

2 CALLADO, A. et al. A survey on internet traffic identification. *IEEE communications surveys & tutorials*, IEEE, v. 11, n. 3, 2009. Citado na página 10.

3 CAO, J.; CHEN, A.; BU, T. A Quasi-Likelihood Approach for Accurate Traffic Matrix Estimation in a High Speed Network. In: *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. [S.l.: s.n.], 2008. Citado na página 10.

4 TUNE, P.; ROUGHAN, M. Internet Traffic Matrices. In: HADDADI, H.; BONAVENTURE, O. (Ed.). *Recent Advances in Networking*. [S.l.]: ACM SIGCOMM eBook, 2013. cap. 3. Citado 3 vezes nas páginas 10, 14 e 18.

5 KRISHNAMURTHY, B. et al. Sketch-based change detection: methods, evaluation, and applications. In: ACM. *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. [S.l.], 2003. p. 234–247. Citado na página 11.

6 KUMAR, A. et al. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In: ACM. *ACM SIGMETRICS Performance Evaluation Review*. [S.l.], 2004. v. 32, n. 1, p. 177–188. Citado na página 11.

7 ZHAO, Q.; KUMAR, A.; XU, J. Joint Data Streaming and Sampling Techniques for Detection of Super Sources and Destinations. In: *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*. [S.l.: s.n.], 2005. p. 7–7. Citado na página 11.

8 CORMODE, G.; MUTHUKRISHNAN, S. Space efficient mining of multigraph streams. In: ACM. *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. [S.l.], 2005. p. 271–282. Citado na página 11.

9 LI, P.; ZHANG, C.-H. A New Algorithm for Compressed Counting with Applications in Shannon Entropy Estimation in Dynamic Data. In: *COLT*. [S.l.: s.n.], 2011. p. 477–496. Citado na página 11.

10 YU, M.; JOSE, L.; MIAO, R. Software Defined Traffic Measurement with OpenSketch. In: *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013. p. 29–42. Citado 2 vezes nas páginas 11 e 15.

11 CHAUDET, C. et al. Optimal positioning of active and passive monitoring devices. In: ACM. *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*. [S.l.], 2005. p. 71–82. Citado 3 vezes nas páginas 11, 14 e 15.

12 CANTIENI, G. R. et al. Reformulating the Monitor Placement Problem: Optimal Network-wide Sampling. In: *Proceedings of the 2006 ACM CoNEXT Conference*. [S.l.: s.n.], 2006. p. 5:1–5:12. Citado 3 vezes nas páginas 11, 14 e 15.

13 SUH, K. et al. Locating network monitors: complexity, heuristics, and coverage. *Computer Communications*, Elsevier, v. 29, p. 1564–1577, 2006. Citado 3 vezes nas páginas 11, 14 e 15.

14 HUANG, G. et al. Measurement-aware monitor placement and routing: a joint optimization approach for network-wide measurements. *IEEE Transactions on Network and Service Management*, IEEE, v. 9, n. 1, p. 48–59, 2012. Citado 3 vezes nas páginas 11, 14 e 15.

15 RAZA, S. et al. Measurouting: a framework for routing assisted traffic monitoring. *IEEE/ACM Transactions on Networking (TON)*, IEEE Press, v. 20, n. 1, p. 45–56, 2012. Citado 3 vezes nas páginas 11, 14 e 15.

16 CLEGG, R. G. et al. On the selection of management/monitoring nodes in highly dynamic networks. *IEEE Transactions on Computers*, IEEE, v. 62, n. 6, p. 1207–1220, 2013. Citado 2 vezes nas páginas 11 e 14.

17 KARP, R. M. Reducibility among combinatorial problems. In: *Complexity of computer computations*. [S.l.]: Springer, 1972. p. 85–103. Citado 2 vezes nas páginas 12 e 45.

18 FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, Elsevier, v. 8, n. 2, p. 67–71, 1989. Citado 3 vezes nas páginas 12, 46 e 48.

19 FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995. Citado 2 vezes nas páginas 12 e 46.

20 SILVEIRA, F. A. F.; MORAES, R. E. N.; VILLAÇA, R. S. Conjunto Mínimo de Observadores para Geração de Matrizes de Tráfego. In: *Anais do XVLVIII Simpósio Brasileiro de Pesquisa Operacional*. Vitória/ES, Brasil: SOBRAPO, 2016. (SBPO '48), p. 3208–3219s. Citado 3 vezes nas páginas 12, 13 e 46.

21 CARDOSO, D. G. et al. GRASP para Conjunto Mínimo de Observadores para Geração de Matrizes de Tráfego. In: *Anais do XVLIX Simpósio Brasileiro de Pesquisa Operacional*. Blumenau/SC, Brasil: SOBRAPO, 2017. (SBPO '49). Citado 4 vezes nas páginas 12, 13, 46 e 48.

22 ZHAO, Q. G. et al. Data Streaming Algorithms for Accurate and Efficient Measurement of Traffic and Flow Matrices. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 33, n. 1, p. 350–361, jun. 2005. Citado 13 vezes nas páginas 12, 14, 15, 19, 20, 23, 26, 29, 34, 41, 43, 61 e 62.

23 NOGUEIRA, B. K. et al. BitMatrix: Ferramenta para Geração de Matrizes de Tráfego Ingresso-Egresso usando *Bitmaps*. In: *Anais do VIII Computer on the Beach*. Florianópolis/SC, Brasil: Universidade do Vale do Itajaí - UNIVALE, 2017. Citado na página 13.

24 MEDINA, A. et al. Traffic Matrix Estimation: Existing Techniques and New Directions. *SIGCOMM Comput. Commun. Rev.*, ACM, v. 32, p. 161–174, 2002. Citado na página 14.

25 ZHANG, Y. et al. Spatio-temporal compressive sensing and internet traffic matrices. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2009. v. 39, n. 4, p. 267–278. Citado na página 14.

26 WU, Q. et al. Efficient traffic flow measurement for ISP networks. In: IEEE. *Local Computer Networks (LCN), 2012 IEEE 37th Conference on*. [S.l.], 2012. p. 348–351. Citado na página 14.

27 CASE, J. D. et al. *Simple network management protocol (SNMP)*. [S.l.], 1990. Disponível em: <http://www.rfc-editor.org/rfc/rfc1157.txt>. Citado na página 14.

28 CISCO SYSTEMS INC. *Introduction to Cisco IOS NetFlow - A Technical Overview*. [S.l.]. Disponível em: <http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html>. Acesso em: 26-11-2015. Citado na página 14.

29 SFLOW. *Traffic Monitoring using sFlow*. 2003. Disponível em: <http://www.sflow.org/sFlowOverview.pdf>. Acesso em: 26-11-2015. Citado na página 14.

30 DUFFIELD, N.; LUND, C.; THORUP, M. Estimating Flow Distributions from Sampled Flow Statistics. In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2003. (SIGCOMM '03), p. 325–336. ISBN 1-58113-735-4. Citado na página 14.

31 DUFFIELD, N.; LUND, C.; THORUP, M. Learn more, sample less: control of volume and variance in network measurement. *IEEE Transactions on Information Theory*, v. 51, n. 5, p. 1756–1775, May 2005. ISSN 0018-9448. Citado na página 14.

32 DUFFIELD, N. G.; GROSSGLAUSER, M. Trajectory sampling with unreliable reporting. *IEEE/ACM Trans. Netw.*, v. 16, n. 1, p. 37–50, 2008. Citado na página 14.

33 MOSHREF, M. et al. Scream: Sketch resource allocation for software-defined measurement. In: ACM. *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. [S.l.], 2015. p. 14. Citado na página 15.

34 LIU, Z. et al. One sketch to rule them all: Rethinking network flow monitoring with univmon. In: ACM. *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. [S.l.], 2016. p. 101–114. Citado na página 15.

35 MOY, J. *OSPF Version 2*. [S.l.], 1998. Disponível em: <https://www.rfc-editor.org/rfc/rfc2328.txt>. Citado 2 vezes nas páginas 18 e 47.

36 DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische mathematik*, Springer, v. 1, n. 1, p. 269–271, 1959. Citado 7 vezes nas páginas 18, 27, 45, 47, 49, 51 e 54.

37 FLAJOLET, P.; MARTIN, G. N. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.*, Academic Press, Inc., Orlando, FL, USA, v. 31, n. 2, p. 182–209, set. 1985. ISSN 0022-0000. Citado na página 20.

38 WHANG, K.-Y.; VANDER-ZANDEN, B. T.; TAYLOR, H. M. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems (TODS)*, ACM, v. 15, n. 2, p. 208–229, 1990. Citado na página 20.

39 CAIDA. *The CAIDA UCSD Anonymized Internet Traces 2012 - 20/09/2012*. 2012. Disponível em: <http://www.caida.org/data/passive/passive_2012_dataset.xml>. Citado 3 vezes nas páginas 21, 29 e 40.

40 CAIDA. *CAIDA Topology Research - Datasets*. 2008. Disponível em: <https://www.caida.org/research/topology/#Datasets>. Citado 4 vezes nas páginas 21, 29, 40 e 57.

41 RIVEST, R. The MD5 message-digest algorithm. 1992. Disponível em: <https://tools.ietf.org/html/rfc1321>. Citado na página 24.

42 GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. San Francisco: W. H. Freeman and Company, 1979. Citado na página 45.

43 CALLON, R. W. *Use of OSI IS-IS for routing in TCP/IP and dual environments*. [S.l.], 1990. Disponível em: <https://tools.ietf.org/html/rfc1195>. Citado na página 47.

44 CHVATAL, V. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, INFORMS, v. 4, p. 233–235, 1979. Citado 3 vezes nas páginas 47, 49 e 62.

45 MARTINS, S. et al. Greedy randomized adaptive search procedures for the Steiner problem in graphs. In: AMERICAN MATHEMATICAL SOC. *Proc. of the Randomization Methods in Algorithm Design: DIMACS Workshop*. [S.l.], 1999. p. 133–145. Citado na página 48.

46 BILAL, N.; GALINIER, P.; GUIBAULT, F. A New Formulation of the Set Covering Problem for Metaheuristic Approaches. *ISRN Operations Research*, Hindawi Publishing Corporation, v. 2013, p. 1–10, 2013. ISSN 2314-6397. Disponível em: <http://www.hindawi.com/isrn/or/2013/203032/>. Citado 2 vezes nas páginas 52 e 53.

47 PRAIS, M.; RIBEIRO, C. C. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, INFORMS, v. 12, p. 164–176, 2000. Citado 2 vezes nas páginas 53 e 54.

48 BARI, M. F. et al. On orchestrating virtual network functions. In: IEEE. *Network and Service Management (CNSM), 2015 11th International Conference on*. [S.l.], 2015. p. 50–56. Citado na página 54.

49 NIST/SEMATECH. *e-Handbook of Statistical Methods*. 2013. Disponível em: <http://www.itl.nist.gov/div898/handbook/eda/section3/eda362.htm>. Citado na página 55.