Walmara de Paula Herman

Formulação Algébrica para a Modelagem de Algoritmos de Roteamento Multi-restritivo *Hop-by-hop*

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia.

São Paulo
2008

Walmara de Paula Herman

Formulação Algébrica para a Modelagem de Algoritmos de Roteamento Multi-restritivo *Hop-by-hop*

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia.

Área de concentração:
Sistemas Eletrônicos

Orientador:
Prof. Dr. José Roberto de Almeida Amazonas

São Paulo
2008

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com anuência de seu orientador.

São Paulo, 28 abril de 2008.

Assinatura do autor

Assinatura do orientador

Ficha Catalográfica

Herman, Walmara de Paula
Formulação Algébrica para a Modelagem de Algoritmos de Roteamento Multi-restritivo *Hop-by-hop*/ W.P. Herman. ed.rev. São Paulo, 2008. 190 p.

Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Telecomunicações e Controle.

1. Algoritmos. 2. Álgebra. 3. Teoria dos grafos. 4. Roteamento com múltiplas restrições. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle. II. t.

À minha família.

Agradecimentos

Agradeço a meu companheiro Carlos, que fomentou o início deste trabalho e cujo incentivo foi fundamental para sua conclusão.

Aos meus pais que nunca desacreditaram dessa empreitada e sempre me apoiaram nos momentos de desespero.

Ao Prof. Dr. Amazonas meu orientador e amigo cujo positivismo e confiança foram fundamentais para a conclusão deste trabalho e que por seu incentivo e apoio terá sempre a minha mais profunda admiração e carinho.

A todos meus companheiros de trabalho da Telefônica e de pesquisa do Laboratório de Comunicações e Sinais (LCS) da Escola Politécnica da USP que me incentivaram, ajudaram e sofreram comigo nesse período.

Resumo

Este trabalho apresenta uma nova estrutura matemática para a álgebra de caminhos, que permite analisar a convergência dos algoritmos de roteamento multi-restritivos *hop-by-hop* e, sob o ponto de vista da engenharia de tráfego e da Qualidade de Serviço (QoS) na arquitetura *Generalized Multiprotocol Label Switching* (GMPLS), garantir de maneira confiável a incorporação de novas métricas de roteamento aos algoritmos de roteamento baseados em múltiplas restrições.

Baseando-se nessa nova álgebra de caminhos, são analisadas as propriedades de monotonicidade, isotonicidade e liberdade, conhecidas por garantir a convergência dos algoritmos de roteamento e, ao contrário do indicado na literatura até o momento, verifica-se que a propriedade de monotonicidade não é condição necessária e nem suficiente para garantir a convergência dos algoritmos de roteamento multi-restritivos *hop-by-hop*. Sendo assim, este trabalho propõe uma nova propriedade, denominada coerência, para a garantia da convergência do roteamento *hop-by-hop* e um novo algoritmo de roteamento *hop-by-hop* com convergência garantida.

Para avaliar os resultados teóricos obtidos, são analisados dois estudos de casos de aplicação do roteamento multi-restritivos *hop-by-hop* com o uso de uma ferramenta de simulação desenvolvida em MATLAB e baseada no algoritmo Eliminação de *Loop* pelo Nó de Destino (ELND) também proposto.

Como resultado das simulações desses estudos de casos, verifica-se que as diferentes estratégias de otimização, necessárias às redes (GMPLS), impõem a necessidade de trabalhar com algoritmos de roteamento que permitam a definição de mais de duas métricas de roteamento com diferentes critérios de otimização para cada uma delas, comprovando, portanto, a necessidade do desenvolvimento e da continuação deste trabalho.

Abstract

This work presents a new mathematical structure for paths algebra that allows the convergence analysis of hop-by-hop multi-constrained routing algorithms and, under the traffic engineering and quality of service perspectives in the Generalized Multiprotocol Label Switching (GMPLS) architecture, trustily ensures the aggregation of new routing metrics in a constrained-based routing.

Based on this new paths algebra, we analyze the monotonicity, isotonicity and freeness properties, known as ensuring routing algorithms convergence, and despite of what has been indicated in the literature, we verified that the monotonicity property is not sufficient to ensure the hop-by-hop routing convergence. Therefore, this work proposes a new property, called coherence, as a necessary and sufficient condition to ensure it, as well as, a new multi-constrained hop-by-hop routing algorithm with ensured convergence.

In order to evaluate the theoretical results obtained, two study cases of the hop-by-hop multi-constrained routing applications are analyzed in the present thesis by using the *Eliminação de Loop pelo Nó de Destino* (ELND) simulation tool, developed in MATLAB and also presented as a product of this work.

As result of these study cases simulations, we verified that different optimization strategies, requested by the (GMPLS) networks, compel the use of routing algorithms that allow the specification of more than two routing metrics with different opti-mization criteria for each one of them, thus proving the necessity of this work and its continuation.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

1 Introdução **1**

2 Revisão Teórica **7**

2.1 Rede 8

2.1.1 Relação 9

2.1.1.1 Estruturas de Ordem 11

2.1.2 Dígrafos 14

2.1.2.1 Caminho 17

2.1.2.2 Árvore 20

2.2 Conceito de Algoritmo 22

2.2.1 Classificação dos Algoritmos 23

2.2.1.1 Classificação em Função da Implementação . . . 23

2.2.1.2 Classificação em Função da Metodologia 24

2.2.1.3 Classificação por Campo de Estudo 25

2.2.1.4 Classificação por Complexidade 25

2.2.2 Análise de Algoritmos 26

2.2.2.1 Pseudocódigos 26

2.2.2.2 Representações Assintóticas - Notação θ 30

2.2.2.3 Representações Assintóticas - Notação O 30

2.2.3 Algoritmos de Roteamento 31
2.2.3.1 Pesquisa Primeiro na Largura 33
2.2.3.2 Pesquisa Primeiro na Profundidade 35
2.2.3.3 Bellman-Ford 36
2.2.3.4 Dijkstra 38
2.2.4 Convergência 39
2.3 Álgebra 40
2.3.1 Álgebra de Caminhos 43
2.3.1.1 Proposta de Bernard Carré 44
2.3.1.2 Proposta de João Luís Sobrinho 49
2.3.2 Espaço Métrico 51
2.3.2.1 Proposta de Gouda e Schneider 51

3 Desenvolvimento do Trabalho 56
3.1 Álgebra de Caminhos Proposta 57
3.1.1 Métrica de Roteamento 58
3.1.2 Função de Combinação de Métricas 58
3.1.3 Sínteses Léxicas 59
3.1.4 Relação de Ordenação Léxica Multi-dimensional $\preceq_{ML}$ . . . 62
3.1.5 Resumo da Álgebra de Caminhos Proposta 64
3.1.6 Exemplos de Utilização 65
3.1.6.1 Exemplo Didático 66
3.1.6.2 Modelagem dos Algoritmos de Pesquisa Primeiro na Largura e Primeiro na Profundidade 67
3.1.6.3 Modelagem dos Algoritmos de Dijkstra e Bellman-Ford 68
3.2 Descrição das Propriedades Conhecidas 69
3.2.1 Monotonicidade e Monotonicidade Estrita 70
3.2.2 Isotonicidade e Isotonicidade Estrita 74

3.2.3 Liberdade 80

3.3 Análise de Convergência 84

3.3.1 Análise de Convergência na Obtenção de Todos os Caminhos Elegíveis 85

3.3.2 Análise de Convergência no Estabelecimento da Coerência 87

3.3.2.1 Análise de Convergência *versus* Monotonicidade . 88

3.3.2.2 Análise de Convergência *versus* Isotonicidade . . 90

3.3.2.3 Análise de Convergência *versus* Liberdade 95

3.4 Propriedade de Coerência 96

3.5 Algoritmo de Eliminação de *Loop* pelo Nó de Destino (ELND) . . 105

3.6 Exemplo Didático: Análise em Grafos com Ciclos Negativos . . . 115

4 Estudos de Casos 119

4.1 Rede SAm-1 119

4.2 Rede Internet2 131

5 Conclusão e Trabalhos Futuros 138

Referências 140

Anexo A – Código Fonte do Algoritmo ELND 147

A.1 Rotina Principal 147

A.2 Rotina Buscapath 151

A.3 Rotina Separapath 154

A.4 Rotina Calculapath 158

A.4.1 Rotina Sintesemin 160

A.4.2 Rotina Sintesemax 161

A.4.3 Rotina Sinteseadd 162

A.4.4 Rotina Sinteseprod 163

A.4.5 Rotina Lexmaior 164

A.4.5.1 Rotina Ordenachar 166
A.4.5.2 Rotina Geraindex 168
A.4.5.3 Rotina Ordenasize 168
A.4.5.4 Rotina Ajustindexsize 169
A.4.6 Rotina Lexmenor 170
A.5 Rotina Ordenapaths 172
A.5.1 Rotina Findpos 180
A.5.2 Rotina Ordenam 181
A.6 Rotina GeraAnexthop 182
A.7 Rotina GeraMnexthop 184
A.8 Rotina Calcdtree 184
A.9 Rotina Buscaloop 185

Lista de Figuras

2.1 Representação dos conceitos de vértice e aresta 8

2.2 Exemplo de dígrafo 16

2.3 Teorema da desigualdade triangular 18

3.1 Exemplo de utilização da álgebra de caminhos 66

3.2 Conceito de monotonicidade 70

3.3 Exemplo de isotonicidade 75

3.4 Exemplo de liberdade 80

3.5 Primeiro exemplo de rede para análise de convergência 85

3.6 Árvores geradoras enraizadas em (u), (v) e (s) 86

3.7 Segundo exemplo de rede para análise de convergência 87

3.8 Exemplo: região de convergência 89

3.9 Exemplo: região de monotonicidade 90

3.10 Exemplo: análise de convergência *versus* monotonicidade 91

3.11 Exemplo: regiões de isotonicidade 92

3.12 Exemplo: análise de convergência *versus* isotonicidade 94

3.13 Exemplo: análise de convergência *versus* liberdade 96

3.14 Análise da propriedade de coerência em uma rede genérica 97

3.15 Prova da garantia de convergência: primeira hipótese 101

3.16 Prova da garantia de convergência: segunda hipótese 102

3.17 Prova da garantia de convergência: terceira hipótese 103

3.18 Dígrafo da rede exemplo 115

4.1 Digrama esquemático da rede SAm-1 120

4.2 Dígrafo equivalente da rede SAm-1 122

4.3 Diagrama ilustrativo da rede Internet2 132

4.4 Dígrafo da abstração parcial da rede Internet2 132

Lista de Tabelas

2.1 Listas de adjacências 16

2.2 Exemplos de aplicação propostos por Carré 47

2.3 Exemplos de aplicação propostos por Sobrinho 50

2.4 Métricas propostas por Gouda e Schneider 54

2.5 Composição de métricas propostas por Gouda e Schneider 54

3.1 Sínteses Léxicas 62

3.2 Elemento neutro de $\vee$ 62

3.3 Representação vetorial da álgebra de caminhos proposta 66

3.4 Resultados das sínteses adotadas para as métricas-combinadas . . 67

3.5 Resultados das ordenações léxicas da métricas-combinadas 67

3.6 Resultado do exemplo de utilização da álgebra de caminhos 67

3.7 Relações de ordenação léxica entre os caminhos α e $(\alpha \circ l)$ para $b = b_\infty$ 72

3.8 Análise da monotonicidade $\times$ síntese $\times$ ordenação léxica 72

3.9 Relações de ordenação léxica entre os caminhos α e $(\alpha \circ l)$ para $b = \{b_l : l < z \text{ e } l \neq \infty\}$ 73

3.10 Análise da monotonicidade estrita$\times$ síntese $\times$ ordenação léxica . . 74

3.11 Análise da isotonicidade $\times$ síntese com $b = b_\infty$ 77

3.12 Análise da isotonicidade estrita $\times$ síntese 78

3.13 Relações de ordenação léxica entre os caminhos $\cup^n(\alpha \circ \beta)$ e $\cup^{n-1}(\alpha \circ \beta)$ 83

3.14 Análise da liberdade $\times$ síntese $\times$ ordenação léxica 84

3.15 Ordenação dos caminhos para os vértices em *loop* 116

Lista de Acrônimos

AHOP *All Hops Optimal Path*

ASON *Automatically Switched Optical Network*

BFS *Breadth First Search*

BGP *Border Gateway Protocol*

DFS *Depth First Search*

DiffServ *Differentiated Services*

DV *Distance Vector*

DWDM *Dense Wavelength Division Multiplexing*

ELND Eliminação de *Loop* pelo Nó de Destino

GMPLS *Generalized Multiprotocol Label Switching*

HDX *High Density Cross-conect*

IETF *Internet Engineering Task Force*

IGP *Interior Gateway Protocols*

IGRP *Internet Gateway Routing Protocol*

IntServ *Integrated Services*

IP *Internet Protocol*

IP-TV *Television over Internet Protocol*

IS-IS *Intermediate System to Intermediate System*

ITU-T *International Telecommunication Union - Telecommunication Standardization Sector*

LS *Link State*

MEF *Metro Ethernet Forum*

MPLS *Multiprotocol Label Switching*

NNI *Network-Network Interface*

NP *Non-deterministic Polynomial time*

NPC *Non-deterministic Polynomial time - Complete*

OIF *Optical Internetworking Forum*

OSPF *Open Shortest Path First*

P *Polynomial time*

QoS Qualidade de Serviço

RIP *Routing information Protocol*

RSP *Restricted Shortest Path*

SAm-1 *South America - 1*

SDH *Synchronous Digital Hierarchy*

STM-1 *Synchronous Transport Modules*

T-MPLS *Transport MPLS*

UNI *User-Network Interface*

1 Introdução

Desde o seu surgimento e da sua primeira utilização comercial, inúmeros aplicativos vêm sendo desenvolvidos no intuito de aproveitar a popularização e a penetração da Internet na vida cotidiana das pessoas.

A partir de uma rede originalmente desenvolvida para a transferência de arquivos entre um grupo controlado de usuários, em um meio não confiável, até a complexa rede como é conhecida hoje, com uma estimativa de 1 bilhão de "internautas" [1] e um incontável número de aplicações, que vão desde as mais simples, como transferência de arquivos, até as mais complexas como, *Television over Internet Protocol* (IP-TV), multi-conferência, tele-medicina e outras, muitas modificações foram propostas e adotadas, tanto em termos de segurança e confiabilidade, quanto em termos da Qualidade de Serviço (QoS) proporcionada.

Na busca de um eficiente mecanismo de medida e controle da QoS, três diferentes paradigmas vêm sendo trabalhados. O primeiro a citar é aquele que atua nas aplicações, adaptando-as às redes, com o objetivo de manter os parâmetros adotados de QoS, tais como atraso e a variação do atraso, que direta ou indiretamente afetam a percepção do usuário em relação à qualidade do serviço usado. Uma das técnicas utilizadas e mais comum é a implementação de *buffers* de acomodação, que permitem à aplicação mitigar, por exemplo, os efeitos da variação do atraso da rede [2, 3].

O segundo paradigma é aquele que atua diretamente sobre as redes, adaptando-as, por meio de diferentes mecanismos, aos diferentes parâmetros de QoS exigidos pelas aplicações. Nesse contexto, a QoS percebida pela aplicação é uma função dos caminhos da rede pelos quais a informação foi enviada ou, dito de uma outra forma, a QoS percebida pela aplicação depende da decisão feita na rede de por qual caminho a informação deve seguir. Entre esses mecanismos, podem ser citados os proporcionados pela arquitetura *Integrated Services* (IntServ) e pela arquitetura *Differentiated Services* (DiffServ), nas redes *Internet Protocol* (IP) [4, 5, 6].

No modelo IntServ, antes do envio da informação, são reservados na rede os recursos necessários para garantir a QoS solicitada, tais como largura de banda nos canais de transmissão e espaços nos *buffers* ou na filas dos roteadores. Nesse modelo são inclusos mecanismos que vão além dos mecanismos normalmente utilizados para proporcionar o envio da informação segundo a QoS original da Internet, o melhor esforço. Em outras palavras, nesse modelo são inclusos mecanismos de classificação do fluxo de informação, agendamento e controle de admissão na rede. Para o modelo IntServ, foram definidos dois tipos de serviços: o serviço garantido [7] e o serviço controlado [8]. O serviço garantido pode ser usado por aplicações que requerem um tempo limitado para o envio da informação, isto é, foi desenvolvido para proporcionar um limite no atraso com que a informação chega a seu destino, sem, no entanto, garantir um limite na variação desse atraso. O serviço controlado ou de carga controlada, por outro lado, foi desenvolvido para ser usado por aplicações adaptativas, que podem tolerar algum atraso, mas que são sensíveis às condições de sobrecarga de tráfego na rede [9]. A principal dificuldade desse modelo tem sido a sua implementação em escala ("escalabilidade"), especialmente em uma rede pública, em que podem ser transmitidos milhões de diferentes fluxos de informação simultaneamente [10].

Já, o modelo DiffServ busca tratar o problema da "escalabilidade" encontrado no modelo IntServ, criando um mecanismo para categorizar os fluxos de tráfego em comportamentos agregados, denominados classes de serviço, que serão tratados diferentemente pela rede, principalmente, em condições de escassez de recursos. Com esse modelo, dois novos tipos de serviços especiais surgiram: o serviço *premium* e o serviço assegurado. O serviço *premium* é prestado às aplicações que exigem baixo atraso e baixa variação do atraso, mesmo em caso de congestionamento na rede. O serviço assegurado, por outro lado, garante uma percentagem da banda ao usuário. O serviço assegurado normalmente é subdividido em três categorias ouro, prata e bronze, definidas em função da quantidade de banda assegurada.

O terceiro modelo a citar é, na verdade, uma composição dos dois anteriores, em que por meio de interfaces padronizadas entre a aplicação e a rede provedora, tanto uma quanto a outra, podem se adaptar na busca do atendimento à QoS solicitada. Nesse contexto, observam-se os esforços realizados pelo *Optical Internetworking Forum* (OIF) e o *Metro Ethernet Forum* (MEF) na definição de uma interface padronizada no limite de responsabilidade entre o provedor de serviços e o assinante, *User-Network Interface* (UNI) [11, 12].

Sob o ponto de vista das redes, o contínuo crescimento observado da de-

manda de tráfego devido não só à consolidação da Internet, mas também à disseminação do uso de aplicações multimídia como, por exemplo, jogos interativos, vídeo-conferências e tele-educação, tem motivado o desenvolvimento e a pesquisa de uma arquitetura de rede que, entre outras coisas, busca integrar transversalmente, do ponto de vista da configuração dos serviços e, conseqüentemente, da sua qualidade, as diversas camadas de rede que constituem as redes comerciais da atualidade. Nesse contexto, o conceito da engenharia de tráfego surge como uma importante ferramenta, utilizada para maximizar, de forma eficiente, o uso dos recursos da rede [13]. A engenharia de tráfego é, basicamente, o processo de acomodar o tráfego, que flui em uma rede, com o objetivo de evitar os congestionamentos causados pela utilização desigual dessa rede. Ela é direcionada à otimização de desempenho das redes operacionais e seu principal objetivo é, de forma eficiente e confiável, facilitar a operação da rede e, ao mesmo tempo, otimizar a utilização de seus recursos e do desempenho do tráfego transportado por ela [14].

Seguindo essa tendência, desenvolveu-se uma arquitetura de rede denominada *Multiprotocol Label Switching* (MPLS), cujo objetivo inicial foi ampliar a capacidade de encaminhamento das informações nas redes IP, mas que, em conjunto com o modelo DiffServ e a engenharia de tráfego, permitiu uma solução tecnológica elegante para o problema de acomodar nas redes IP o crescimento da demanda e das suas diferentes QoS.

Na arquitetura MPLS, dentro do contexto da engenharia de tráfego, aparece o conceito do roteamento baseado em restrições. O roteamento baseado em restrições se refere a uma classe de algoritmos de roteamento, que buscam caminhos em uma rede que satisfaçam uma lista de requisitos ou restrições e, por isso, eles também são conhecidos como algoritmos de roteamento mono/ multi-restritivos. Do ponto de vista da engenharia de tráfego, a definição das restrições pode ser imposta tanto pela rede, quanto por políticas administrativas, e, normalmente, incluem parâmetros como largura de banda, número de saltos e atraso, que não só afetam a otimização dos recursos da rede, mas também a qualidade dos serviços providos por ela. Por essa razão, o roteamento baseado em restrições também é freqüentemente chamado de roteamento com QoS [14].

Do ponto de vista da Internet, o roteamento mono/ multi-restritivo permite à arquitetura MPLS atender as necessidades de verificação de disponibilidade de recursos na rede, em um paradigma de roteamento que coexiste com o paradigma de roteamento *hop-by-hop*[1] atualmente provido pelos *Interior Gateway Protocols*

[1] A definição do paradigma de roteamento *hop-by-hop* é apresentada na Seção 2.2.3 desta

(IGP) existentes na Internet.

Um outro conceito importante e adotado na arquitetura MPLS é a separação da rede em dois planos de atuação: o plano de dados, para as funções de envio e repasse da informação, e o plano de controle para as funções de coordenação, roteamento e sinalização. Com a difusão da tecnologia MPLS e seguindo a filosofia de integração operativa entre as redes, surge pelo *Internet Engineering Task Force* (IETF) uma proposta, conhecida como *Generalized Multiprotocol Label Switching* (GMPLS), para a generalização/ extensão do plano de controle do MPLS, cujo objetivo é englobar o controle de outros tipos de redes, tais como, as redes de transporte *Synchronous Digital Hierarchy* (SDH) e *Dense Wavelength Division Multiplexing* (DWDM) [15]. Paralelamente a esse esforço, sob o ponto de vista das redes ópticas, em 2002, o *International Telecommunication Union - Telecommunication Standardization Sector* (ITU-T) definiu a arquitetura *Automatically Switched Optical Network* (ASON) [16] e, em 2006, sob o ponto de vista das redes de transporte por comutação de pacotes e orientada a conexão, definiu a arquitetura *Transport MPLS* (T-MPLS) [17, 18].

Com o desenvolvimento do GMPLS e a incorporação, no plano de controle, de novas tecnologias de rede, novos aspectos ou restrições, impostos pela peculiaridade de cada uma dessas novas tecnologias, passam a ter que ser incorporados ao roteamento mono/ multi-restritivo, motivando com isso o desenvolvimento deste trabalho.

Em geral, o problema da busca mono/ multi-restritiva de caminhos é um problema conhecido como intratável[2] [14] e, por isso, várias propostas de algoritmos vêm sendo feitas, entre os quais, podem ser citados: o algoritmo *Jaffe's approximation* [3], *Fallback* [19], TAMCRA [20], SAMCRA [21], os algoritmos que abordam o problema *Restricted Shortest Path* (RSP), ou seja, os algoritmos que buscam o caminho de menor custo entre aqueles que satisfazem uma única restrição como, por exemplo, o algoritmo *Constrained* Bellman-Ford, *ϵ-optimal Approximation, Backward-forward Heuristic*, *Lagrangian-based Linear Composition* [19] e *K-approximation Algorithm* [22].

No entanto, na prática, muitos dos algoritmos propostos foram desenvolvidos de maneira empírica e têm seu mérito restrito a um universo de validade muitas vezes definido pelo tamanho das redes ou pela topologia das mesmas, em que a

tese. Embora a tradução de *hop-by-hop* para o português seja salto-a-salto, em função do seu freqüente uso em inglês na literatura, optou-se por manter neste trabalho a denominação em inglês.

[2] A definição de problemas intratáveis é apresentada na Seção 2.2.1.4 desta tese.

portabilidade das soluções apresentadas para outros problemas similares não é intuitiva. Por exemplo, a utilização das heurísticas desenvolvidas originalmente para o problema da busca de caminhos mais curtos, tais como a de Dijkstra, pode não convergir quando utilizada na busca de caminhos com outros tipos de restrições, tais como a largura de banda disponível [23, 24, 25, 26, 27, 28].

Deste modo, analisando esse problema sob a perspectiva do desenvolvimento de protocolos de roteamento com múltiplas restrições, em que se busca analisar e conceber um algoritmo de roteamento com convergência garantida para diferentes tipos ou formas de combinações dos parâmetros de QoS, verifica-se a necessidade de abordar esse tema de maneira integrada e genérica, por meio de um arcabouço matemático único, que permita validar, de uma forma abrangente e independente da topologia da rede ou dos detalhes da implementação desses algoritmos, as soluções propostas e utilizadas até então, além de possibilitar o desenvolvimento de novos algoritmos de roteamento mono/ multi-restritivo com garantia de convergência [29].

Entre os arcabouços matemáticos disponíveis na literatura, este trabalho aborda o problema da garantia de convergência do roteamento mono/ multi-restritivo *hop-by-hop* por meio da Álgebra Universal [30, 31], buscando como objetivos:

- rever os modelos matemáticos relacionados e propostos até o momento, buscando e validando uma solução genérica e homogênea, capaz de integrá-los de uma maneira flexível o bastante para permitir sua utilização na validação ou no desenvolvimento de novos algoritmos de roteamento mono/ multi-restritivo;
- comparar sistematicamente diferentes heurísticas de roteamento mono/ multi-restritivo com relação à sua garantia de convergência;
- especificar e analisar as propriedades apresentadas na literatura como garantidoras da convergência dos algoritmos de roteamento mono/ multi-restritivo *hop-by-hop* e, em caso de inconsistência, definir novas propriedades para essa função; e
- apresentar, no contexto da arquitetura GMPLS, um exemplo de algoritmo de roteamento mono/ multi-restritivo *hop-by-hop*, com convergência garantida e orientado a QoS.

Assim, esta tese está organizada da seguinte forma:

- o Capítulo 1, contém uma introdução ao tema proposto para este trabalho, apresentando, sob o ponto de vista da convergência tecnológica das rede de telecomunicações e da evolução dos mecanismos de controle da QoS, os fatores que motivaram a análise de convergência dos algoritmos de roteamento multi-restritivo *hop-by-hop*, descrevendo também os objetivos que nortearam o desenvolvimento desta tese;

- o Capítulo 2, de uma forma orientada ao atendimento dos objetivos propostos, detalha os conceitos matemáticos apresentados pela Teoria das Estruturas, a partir do qual se definem os conceitos de rede, relação e dígrafo, bem como se integram os ferramentais matemáticos disponibilizados pela Álgebra Universal, focando-se na definição da Álgebra de Caminhos e resumindo as principais propostas disponíveis na literatura até o momento;

- em função das limitações encontradas, o Capítulo 3: propõe uma nova Álgebra de Caminhos, descrevendo, por meio dela, as propriedades conhecidas na literatura por garantir a convergência dos algoritmos de roteamento; discute e comenta a aplicabilidade dessas propriedades no contexto da análise de convergência dos algoritmos de roteamento multi-restritivo implementados no paradigma de roteamento *hop-by-hop*; e, em razão dos resultados obtidos, apresenta também uma nova propriedade, denominada coerência, como condição necessária e suficiente para a garantia da convergência do roteamento *hop-by-hop*; propõe para esse paradigma um algoritmo de roteamento com convergência garantida; e apresenta o algoritmo de simulação Eliminação de *Loop* pelo Nó de Destino (ELND) proposto e utilizado na análise dos resultados obtidos com imposição da propriedade de coerência nas redes com roteamento *hop-by-hop*;

- o Capítulo 4 propõe e analisa dois estudos de casos de aplicação da nova álgebra de caminhos proposta para a análise de engenharia de tráfego em redes GMPLS; e, finalizando,

- o Capítulo 5 apresenta as conclusões deste trabalho, descrevendo alguns tópicos de interesse para trabalhos futuros.

2 Revisão Teórica

A escola francesa de matemática denominada Bourbaki, que dedicou-se a reescrever uma boa parte da matemática com base no uso da lógica formal e em estruturas abstratas definidas por axiomas[1], desenvolveu um novo campo da matemática denominado Estrutura.

Informalmente, define-se uma estrutura como um ou mais conjuntos com vários objetos matemáticos associados, tais como subconjuntos, conjunto de subconjuntos, operações e relações, que satisfazem vários axiomas [33]. Entre as várias estruturas estudadas pelos matemáticos, pode-se citar: a estrutura algébrica, que é descrita pela Álgebra Universal; a estrutura topológica, que introduz o conceito de continuidade; a estrutura métrica, que introduz o conceito de distância; a estrutura de rede, que introduz os conceitos de relação, ordenação e dígrafo; e a estrutura de categoria, que formaliza a passagem de um objeto matemático de uma estrutura a outra [30, 31].

Assim, considerando-se que uma das possíveis formas de representar e analisar o comportamento de uma rede de telecomunicações é por meio de sua modelagem em dígrafos e dos teoremas derivados da Teoria dos Dígrafos, e que este trabalho, entre outras coisas, propõe uma nova estrutura algébrica para analisar a convergência dos algoritmos de roteamento pelo destino (*hop-by-hop*), este capítulo introduz o conceito de dígrafo a partir da definição de outros dois conceitos: rede e relação. Por meio de um objeto matemático denominado reticulado, que é definido tanto na estrutura de rede quanto na Álgebra Universal, introduz-se o conceito de estruturas algébricas e da álgebra de caminhos proposta por Bernard Carré [34] e João Luís Sobrinho [35, 24, 36]. Em função da utilização da mesma nomenclatura e com objetivo de evitar equívocos de interpretação, este capítulo apresenta também uma rápida descrição matemática do espaço métrico e da métrica, com suas diferentes interpretações, tanto em matemática como em

[1] "Um axioma é uma premissa considerada necessariamente evidente e verdadeira, fundamento de uma demonstração, porém ela mesma não é demonstrável, originada segundo a tradição racionalista, de princípios inatos da consciência ou, segundo os empiristas, de generalizações da observação empírica."[32]

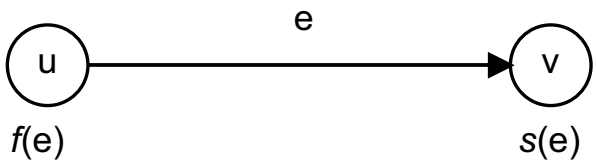


Figura 2.1: Representação dos conceitos de vértice e aresta

redes de telecomunicações, além de resumir a proposta de formalização da métrica em redes de telecomunicações feita por Mohamed G. Gouda e Marco Schneider em [23].

Por final, tendo em vista a análise do algoritmo ELND proposto neste trabalho, este capítulo também introduz os conceitos de algoritmo, descrevendo alguns algoritmos de roteamento mais comuns e as respectivas formas de análise utilizadas.

2.1 Rede

Do ponto de vista da matemática, uma rede é definida com base nas quatro primitivas e em dois axiomas descritos a seguir:

Primitiva 2.1. **V** é um conjunto de elementos denominados vértices,

Primitiva 2.2. **E** é um conjunto de elementos denominados arestas,

Primitiva 2.3. f é uma função cujo domínio é **E** e cuja imagem está contida em **V**,

Primitiva 2.4. s é uma função cujo domínio é **E** e cuja imagem está contida em **V**,

Axioma 2.1. **V** é um conjunto finito e não vazio

e

Axioma 2.2. **E** é um conjunto finito.

As primitivas 2.3 e 2.4 servem para identificar, respectivamente, o vértice de saída ou vértice sainte e o vértice de entrada ou vértice entrante de uma rede. Em outras palavras, sendo dado como exemplo um conjunto de vértices $V = \{(u), (v)\}$ e de arestas $E = \{e\}$, ilustrados na Figura 2.1, diz-se que:

Definição 2.1. Uma aresta (e) é sainte do vértice (u), se este corresponder à imagem da função $f(e)$: $(u) = f(e)$. O vértice (u) é denominado predecessor[2].

[2]Neste trabalho, a notação matemática de dois pontos deve ser lida como "tal que".

Definição 2.2. Uma aresta (e) é incidente no vértice ou entra no vértice (v), se este corresponder à imagem da função $s(e)$: $(v) = s(e)$. O vértice (v) é denominado sucessor.

Observe que uma aresta é definida como orientada, ou seja, a aresta e_1 com vértice sainte $(u) = f(e_1)$ e vértice entrante $(v) = s(e_1)$ é diferente da aresta e_2 com vértice sainte $(v) = f(e_2)$ e vértice entrante $(u) = s(e_2)$.

Definição 2.3. Duas arestas quaisquer e_1 e e_2 são paralelas se e somente se $f(e_1) = f(e_2)$ e $s(e_1) = s(e_2)$.

Um outro conceito relevante é o de laço[3].

Definição 2.4. Um laço é uma aresta cujos vértices entrante e sainte coincidem, ou seja, $f(e) = s(e)$.

2.1.1 Relação

Baseando-se no conceito de rede, define-se uma relação como uma rede que obedece o Axioma 2.3.

Axioma 2.3. Não existe arestas distintas paralelas.

Portanto, uma relação é uma rede, mas nem sempre uma rede é uma relação. Como, em uma relação, os vértices de uma aresta são sempre diferentes, este trabalho utiliza a notação de par ordenado $(f(e), s(e))$ para representar uma aresta.

Uma relação pode possuir algumas propriedades, listadas a seguir e que serão utilizadas neste trabalho.

Propriedade 2.1. Reflexividade: uma relação R é reflexiva se todos os vértices em R pertencem a um laço.

Propriedade 2.2. Irreflexividade: uma relação R é irreflexiva se nenhum vértice em R pertence a um laço.

Propriedade 2.3. Simetria: uma relação R é simétrica se, quando existir um par ordenado ou aresta (u, v) qualquer em R, então (v, u) também existirá.

[3] A definição de laço dada nesta seção pela Teoria dos Dígrafos é diferente da definição dada pela Álgebra Universal na Seção 2.3.

Propriedade 2.4. Assimetria: uma relação R é assimétrica se para quaisquer dois vértices (u), (v) em R, pelo menos um dos pares ordenados ou arestas (u, v) ou (v, u) não existe.

Propriedade 2.5. Transitividade: uma relação R é transitiva quando para quaisquer três vértices distintos (u), (v) e (z) em R, se os pares ordenados ou arestas (u, v) e (v, z) existirem em R, então sempre existirá um par ordenado ou aresta (u, z) em R.

Propriedade 2.6. Completude: uma relação R é completa se para cada par de distintos vértices (u) e (v) em R, existe ao menos um dos pares ordenados ou arestas (u, v) ou (v, u) em R.

Sejam dados como exemplos as relações $>$, $<$, $\leq$ e $\geq$ sobre o conjunto dos números inteiros. Analisando essas relações em função das Propriedades 2.1, 2.5, 2.3 e 2.6, tem-se:

- Reflexividade: as relações $>$ e $<$ são irreflexivas, pois um número inteiro a qualquer não é maior e nem menor que ele mesmo. Entretanto, as relações $\geq$ e $\leq$ são reflexivas.
- Transitividade: as relações $>$, $<$, $\leq$ e $\geq$ são transitivas, pois dados três números a, b e c, tal que:
  - $a > b$ e $b > c$, então $a > c$ é verdadeiro;
  - $a < b$ e $b < c$, então $a < c$ é verdadeiro;
  - $a \geq b$ e $b \geq c$, então $a \geq c$ é verdadeiro; e
  - $a \leq b$ e $b \leq c$, então $a \leq c$ é verdadeiro.
- Simetria: as relações $>$, $<$, $\leq$ e $\geq$ são assimétricas, pois se:
  - $a > b$ então $b \ngtr$ que a;
  - $a < b$ então $b \nless$ que a;
  - $a \leq b$ e $b \leq a$, então $b = a$; e
  - $a \geq b$ e $b \geq a$, então $b = a$.
- Completude: as relações $>$, $<$, $\leq$ e $\geq$ são completas, pois para dois números distintos quaisquer sempre um será $>$, $<$, $\leq$ ou $\geq$ que o outro.

Definição 2.5. Uma relação reflexiva, simétrica e transitiva é denominada relação de equivalência.

Se R é uma relação de equivalência em um conjunto A, então, para $a \in \mathbf{A}$, a classe de equivalência de $[a]$ é o conjunto de todos os elementos equivalentes a a e que pertençam ao conjunto A, $[a] = \{b \in A : a \ R \ b\}$. Por exemplo, dada a relação de equivalência $R = \{(a, b) : a, b \in \mathbf{N} \text{ e } a + b \text{ é um número par}\}$, observe que ela é: reflexiva pois $a + a$ é par; simétrica, pois dado que $a + b$ é par, então $b + a$ também é par; transitiva, pois dado que $a + b$ é par e $b + c$ é par, então $a + c$ também é par.

Um outro exemplo de relação de equivalência é a igualdade sobre os números inteiros. Neste trabalho, uma relação de equivalência será denotada por $\equiv$.

Uma relação é dita binária quando relaciona ou associa dois-a-dois os elementos de um conjunto ou de dois diferentes conjuntos. Por exemplo, dados dois vértices (v) e (u) no conjunto $\mathbf{V}$, a aresta (u, v) representa uma relação binária entre (v) e (u). Em particular, essa relação binária é denominada adjacência.

Um outro exemplo relevante de relação binária é a função.

Definição 2.6. Uma função f é uma relação binária R entre os elementos de dois conjuntos $\mathbf{X}$ e $\mathbf{Y}$, tal que cada elemento $x \in \mathbf{X}$ corresponde somente a um único $y \in \mathbf{Y}$, em que $y = f(x)$ se e somente se $x \ R \ y$.

2.1.1.1 Estruturas de Ordem

A Teoria das Estruturas de Ordem é um ramo da matemática que estuda os diferentes tipos de relações binárias que intuitivamente capturam a noção de ordenação [31].

Definição 2.7. Uma relação de ordenação ou simplesmente ordenação é uma relação binária assimétrica e transitiva.

Uma ordenação pode ser reflexiva ou irreflexiva se obedece ou não à propriedade reflexividade.

Neste trabalho, uma relação de ordenação será denotada por $\preceq$ ou $\succeq$. Portanto, dados dois elementos (v) e (u) pertencentes a $\mathbf{V}$, se $(v) \preceq (u)$, lê-se (v) é inferior, menos otimizado ou equivalente a (u), e se $(u) \succeq (v)$, lê-se (u) é superior, mais otimizado ou equivalente a (v), de maneira que em ambos os casos o significado é o mesmo, obedecendo o conceito da dualidade ou da oposição[4].

As representações $\prec$, denominada "estritamente inferior a", "estritamente

[4]Uma relação R^{op} é dual de R se R^{op} representar uma relação de ordem inversa de R, ou seja, se $(v) = (u)$ em R, então $(u) = (v)$ em R^{op}.

menos otimizado que", ou simplesmente "pior que", e $\succ$, denominada "estritamente superior a", "estritamente mais otimizado que", ou simplesmente "melhor que", são dadas, respectivamente, quando $(v) \preceq (u)$ mas $(v) \neq (u)$ e $(u) \succeq (v)$ mas $(u) \neq (v)$.

Analisando as propriedades de uma ordenação, tem-se que:

- Reflexividade: $(v) \preceq (v)$ para todos os $(v) \in \mathbf{A}$;
- Assimetria: se $(v) \preceq (u)$ e $(u) \preceq (v)$, então $(v) = (u)$; e
- Transitividade: se $(v) \preceq (u)$ e $(u) \preceq (z)$, então $(v) \preceq (z)$.

Dois elementos (v) e (u) pertencentes a $\mathbf{V}$ são ditos comparáveis se $(v) \preceq (u)$ ou $(u) \preceq (v)$; caso contrário, eles são ditos incomparáveis.

Quando uma ordenação obedece a propriedade de totalidade, ou seja, todos os elementos de um conjunto $\mathbf{V}$ tomados dois-a-dois sempre são comparáveis, a ordenação é dita total e o conjunto $\mathbf{V}$ é denominado totalmente ordenado ou cadeia; caso contrário, a ordenação é denominada ordenação parcial e o conjunto $\mathbf{V}$, parcialmente ordenado.

Propriedade 2.7. Totalidade: Uma relação R é total sobre um conjunto $\mathbf{V}$ se para qualquer par de elementos ou vértices $(v), (u) \in \mathbf{V}$ tem-se que: ou $(v) \preceq (u)$, ou $(u) \preceq (v)$.

Em uma relação de ordenação $\mathbf{V}$, um elemento ϕ é dito cota inferior se $\phi \preceq (v)$ para todo $(v) \in \mathbf{V}$. Observe que a relação de ordenação é assimétrica e, portanto, $\mathbf{V}$ deve ter somente um elemento cota inferior. Se $\phi \prec (v)$ para todo $(v) \in \mathbf{V}$, então ele é denominado elemento mínimo.

Da mesma forma, um elemento θ é dito cota superior se $\theta \succeq (v)$ para todo $(v) \in \mathbf{V}$. Obedecendo também a propriedade de assimetria, $\mathbf{V}$ deve ter somente um cota superior. Se $\theta \succ (v)$ para todo $(v) \in \mathbf{V}$, então ele é denominado elemento máximo.

Seja $\mathbf{U}$ um subconjunto qualquer de um conjunto parcialmente ordenado $\mathbf{V}$, denomina-se:

Definição 2.8. elemento majorante ou cota superior de $\mathbf{U}$, o elemento $M \in \mathbf{V}$: $u \preceq M, \forall\ u \in \mathbf{U}$; e

Definição 2.9. elemento minorante ou cota inferior de $\mathbf{U}$, o elemento $m \in \mathbf{V}$: $m \preceq u, \forall\ u \in \mathbf{U}$.

Definição 2.10. O elemento s é supremo de $\mathbf{U}$ se for menor do que todos os majorantes de $\mathbf{U}$.

Definição 2.11. O elemento i é ínfimo de $\mathbf{U}$ se for maior do que todos os minorantes de $\mathbf{U}$.

Propriedade 2.8. Monotonicidade, também conhecida como preservação da ordem, implica que dados dois conjuntos ordenados $\mathbf{V}$ e $\mathbf{U} : \mathbf{U} \subseteq \mathbf{V}$, existindo os elementos supremos e ínfimos dos conjuntos $\mathbf{U}$ e $\mathbf{V}$, então $s_{\mathbf{U}} \preceq s_{\mathbf{V}}$ e $i_{\mathbf{V}} \preceq i_{\mathbf{U}}$ [37].

Definição 2.12. Um reticulado[5] $\mathbf{L}$ é um conjunto parcialmente ordenado que, para quaisquer dois elementos u e v, há pelo menos: um supremo denominado união ou soma de u e v, representado por $u \vee v$, e um ínfimo denominado intersecção ou produto de u e v, representado por $u \wedge v$ [34, 38, 39].

Quando existir somente o supremo $u \vee v$, o conjunto $\mathbf{L}$ é denominado semirreticulado superior. Da mesma forma, quando existir somente o ínfimo $u \wedge v$, o conjunto $\mathbf{L}$ é denominado semirreticulado inferior.

Na Seção 2.3 desta tese, também será apresentada, sobre a perspectiva da Álgebra Universal, uma definição de reticulado equivalente à indicada anteriormente.

Dois outros conceitos muito utilizados e que servem de base para este trabalho são o conceito de ordenação de produto e o de ordenação lexicográfica ou léxica, também conhecida como ordenação do dicionário.

Definição 2.13. Uma ordenação de produto é um produto cartesiano ordenado, gerado de dois conjuntos ordenados quaisquer, em que para quaisquer pares de par ordenado (a_1, b_1) e (a_2, b_2) pertencentes ao produto cartesiano ordenado, $(a_1, b_1) \preceq (a_2, b_2)$ se e somente se $a_1 \preceq a_2$ e $b_1 \preceq b_2$.

Um produto cartesiano entre dois conjuntos $\mathbf{A}$ e $\mathbf{B}$, representado por $A \times B$, é o conjunto de todos os possíveis pares ordenados (a, b), cujo primeiro elemento a do par ordenado é membro do conjunto $\mathbf{A}$ e o segundo elemento b do par ordenado é membro do conjunto $\mathbf{B}$. Assim, um produto cartesiano $\mathbf{A} \times \mathbf{B}$ é dito ordenado se para quaisquer pares de par ordenado (a_1, b_1) e $(a_2, b_2) \in \mathbf{A} \times \mathbf{B}$, $(a_1, b_1) \preceq (a_2, b_2)$, em que $(a_1, b_1) \preceq (a_2, b_2)$ se e somente se $a_1 \preceq a_2$ e $b_1 \preceq b_2$.

Um produto cartesiano pode ser generalizado em um produto cartesiano sobre n conjuntos $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \ldots, \mathbf{A}_n$, denominado produto cartesiano n^{ário}, em que $\mathbf{A}_1 \times \mathbf{A}_2 \times \mathbf{A}_3 \times \cdots \times \mathbf{A}_n = \{(a_1, a_2, \ldots, a_n) : a_1 \in \mathbf{A}_1, a_2 \in \mathbf{A}_2, \ldots \text{ e } a_n \in \mathbf{A}_n\}$.

[5] Tradução da palavra em inglês *lattice*.

Os elementos de um produto cartesiano n^{ário} são denominados enuplas.

Definição 2.14. Uma enupla, também conhecida como n-tuplo ou n-upla, é uma seqüência ordenada e finita de n elementos resultante de um produto cartesiano n^{ário}.

Assim, um par ordenado também é chamado de 2-upla.

Definição 2.15. Uma ordenação lexicográfica ou léxica é um produto cartesiano ordenado, gerado de dois conjuntos ordenados quaisquer **A** e **B**, em que para quaisquer pares de par ordenado (a_1, b_1) e (a_2, b_2), pertencente ao produto cartesiano ordenado, $(a_1, b_1) \preceq (a_2, b_2)$ se $a_1 \prec a_2$, ou $a_1 = a_2$ e $b_1 \preceq b_2$.

Observe que um alfabeto pode ser interpretado como um conjunto ordenado de letras e as palavras como um subconjunto do produto cartesiano n^{ário} do alfabeto, de maneira que um dicionário, como um conjunto ordenado de palavras, pode ser interpretado como uma ordenação léxica, ou seja, dadas duas palavras quaisquer formadas, respectivamente, pela seqüência de letras "$a_1a_2 \ldots a_n$" e "$b_1b_2 \ldots b_n$", a palavra "$a_1a_2 \ldots a_n$" aparece no dicionário antes da palavra "$b_1b_2 \ldots b_n$" se e somente se $a_1 \prec b_1$, ou $a_1 = b_1$ e $a_2 \preceq b_2$, ou $a_2 = b_2$ e $a_3 \preceq b_3$ ... , ou $a_{n-1} = b_{n-1}$ e $a_n \preceq b_n$. Essa ordenação parte do princípio que todas as palavras têm o mesmo comprimento, uma vez que sempre se podem acrescentar caracteres brancos em uma palavra que originalmente tem menos caracteres que outra, e considerar que o caracter branco é um elemento mínimo do alfabeto.

2.1.2 Dígrafos

Baseando-se nos conceitos de rede e relação, define-se um dígrafo como sendo uma relação que obedece o Axioma 2.4.

Axioma 2.4. Não existem laços.

Em outras palavras, um dígrafo é uma relação irreflexiva. Observe que um dígrafo determina uma relação binária R_D sobre o conjunto de vértices **V** por meio da seguinte regra: $(v_i)\ R_D\ (v_j)$ se e somente se $((v_i), (v_j)) \in \mathbf{E}$ [34].

Quando um dígrafo obedece a propriedade de simetria ele é denominado grafo, ou seja, um grafo é um dígrafo que sempre apresenta o par de arestas (u, v) e (v, u) para qualquer par de vértices pertencentes a ele.

Definição 2.16. Um grafo é uma relação irreflexiva e simétrica.

Este trabalho utiliza a notação $G = (\mathbf{V}, \mathbf{E})$ e $D = (\mathbf{V}, \mathbf{E})$ para representar, respectivamente, um grafo e um dígrafo formados pelo conjunto $\mathbf{V}$ de vértices e $\mathbf{E}$ de arestas. Quando dois vértices pertencerem a uma mesma aresta e, ou seja, $(u) = f(e)$ e $(v) = s(e)$, então eles são denominados vértices adjacentes, pois uma aresta pode ser interpretada como uma relação binária denominada adjacência, conforme descrito na Seção 2.1.1.

Um dígrafo é dito orientado se o conjunto de arestas $\mathbf{E}$ for composto de pares ordenados, ou seja, as arestas são orientadas. Quando um dígrafo for formado por arestas não orientadas, ele é dito não orientado. Entretanto, todo dígrafo não orientado D' pode ser associado a um dígrafo orientado D, no qual cada aresta de D' corresponde biunivocamente a uma par de arestas de sentidos opostos em D [40].

Para um dígrafo não orientado, define-se o grau de um vértice como sendo o número de arestas que incide nele. Já em um dígrafo orientado, define-se o grau de saída de um vértice como sendo o número de arestas que saem dele e grau de entrada de um vértice como sendo o número de arestas que entram nele.

Quando um dígrafo obedece a propriedade de totalidade, então para um conjunto $\mathbf{V}$ de n elementos, o conjunto $\mathbf{E}$ apresenta para grafos orientados $2 \times C_{n,2}$ elementos e para grafos não orientados $C_{n,2}$ elementos, em que $C_{n,2}$ representa a combinação de n elementos tomados dois a dois.

O número de vértices em um dígrafo é denominado a ordem do dígrafo e a relação entre o número de arestas do dígrafo e o maior número de arestas possíveis ($2 \times C_{n,2} = n \times (n-1)$ para dígrafos orientados e $C_{n,2} = n \times (n-1)/2$ para dígrafos não orientados) é denominada densidade do dígrafo.

Muitas vezes é conveniente representar um dígrafo por meio de uma matriz ou um conjunto de listas, como será verificado ao longo deste trabalho.

Representando um dígrafo como uma matriz, denominada matriz de adjacências $\overline{Adj}$, as linhas da matriz representam os vértices saintes ou de origem, as colunas os vértices entrantes ou de chegada e cada elemento da matriz uma aresta do dígrafo. Dessa forma, a matriz de adjacência é quadrada de dimensão $n \times n$ ou ordem n. Observe o dígrafo exemplificado na Figura 2.2 e sua correspondente matriz de adjacências 2.1. Quando um dígrafo representa um grafo, a matriz de adjacências é simétrica com a diagonal principal nula.

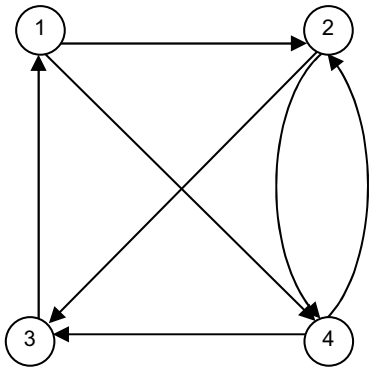


Figura 2.2: Exemplo de dígrafo

$$\overline{Adj} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.1}$$

Representando um dígrafo como um conjunto de listas, denominadas listas de adjacência, cada lista é, na verdade, uma lista de vértices formada por um vértice de referência e pelo conjunto dos vértices que com ele compartilham uma aresta. Observe a lista de adjacência dada na Tabela 2.1 e correspondente ao dígrafo da Figura 2.2.

Tabela 2.1: Listas de adjacências

Vértices Saintes		Vértices Entrantes	
Origem	Destino	Destino	Origem
1	2 4	1	3
2	3 4	2	1 4
3	1	3	2 4
4	2 3	4	1 2

Diz-se que um dígrafo é ponderado ou valorado quando existem uma ou mais funções, denominadas funções de peso, que relacionam as arestas do dígrafo a um ou mais conjuntos de números, denominados pesos. Por exemplo, quando uma rede óptica é modelada por um dígrafo, em que uma aresta representa os enlaces físicos dessa rede, a cada enlace pode-se atribuir uma ou mais funções de peso para relacioná-lo aos correspondentes atributos: distância física, atenuação acumulada, dispersão cromática total, etc.

Em um dígrafo ponderado, a matriz de adjacências $\overline{Adj}$, também pode ser representada como uma matriz de adjacências ponderada, tendo seus elementos $a_{ij} = w(v_i, v_j)$, em que $w(v_i, v_j)$ corresponde ao peso da aresta (v_i, v_j) e, caso a

aresta (v_i, v_j) não exista, $w(v_i, v_j) = \infty$.

2.1.2.1 Caminho

Um outro conceito definido na Teoria dos Dígrafos e muito utilizado neste trabalho é o conceito de caminho, que será introduzido a partir de um conceito mais genérico que é o de percurso, cadeia ou itinerário.

Definição 2.17. Um percurso é uma seqüência qualquer de arestas adjacentes que ligam dois vértices.

Esse conceito se aplica também em dígrafos orientados, bastando que se ignore o sentido da orientação das arestas. Um percurso é dito elementar se não passar duas vezes pelo mesmo vértice e dito simples se não passar duas vezes pela mesma aresta. O comprimento de um percurso é o número de arestas que o compõe.

Um percurso é abrangente ou gerador (*spanning*) em relação a um dos conjuntos **E** ou **V** de um dígrafo, quando utiliza todos os elementos desse conjunto ao menos uma vez.

Por outro lado, somente para dígrafos orientados define-se o conceito de caminho.

Definição 2.18. Um caminho é um percurso no qual todas as arestas possuem a mesma orientação.

Em outras palavras, em um dígrafo orientado, um caminho é um conjunto de sucessivos vértices adjacentes $v_1, v_2, \ldots, v_n$, em que o primeiro vértice (v_1), ou vértice de origem do caminho, corresponde ao vértice sainte da primeira aresta que compõe o caminho e o último vértice (v_n), ou vértice de término do caminho, corresponde ao vértice entrante da última aresta que compõe esse caminho.

Dessa forma, um caminho $p(v_1, v_n)$ pode ser representado pela seqüência de vértices $p(v_1, v_n) = v_1, v_2, v_3, \ldots, v_n$ ou pela seqüência correspondente de arestas $p(v_1, v_n) = e_1, e_2, e_3, \ldots, e_{n-1}$, em que $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3)$, ..., $e_{n-1} = (v_{n-1}, v_n)$. Para simplificar a notação e sempre que não houver dúvidas de sua interpretação, em alguns casos, este trabalho utilizará também a notação $p_{1,n}$ para representar um caminho que começa no vértice v_1 e termina no vértice v_n, ou seja, um caminho de comprimento $d(v_1, v_n) = n - 1$ ou também denominado caminho de ordem $n - 1$, em que a distância $d(v_1, v_n)$ é dada pelo número de arestas que compõem o caminho.

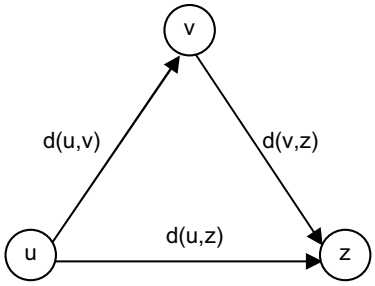


Figura 2.3: Teorema da desigualdade triangular

Um percurso ou um caminho pode ser classificado como sendo aberto ou fechado. Ele é dito fechado se a sua última aresta é incidente em seu vértice de origem, ou seja, se o seu primeiro e o seu último vértice forem adjacentes e a aresta que forma essa adjacência pertence ao caminho ou percurso. Caso contrário, ele é dito aberto.

Observe que sempre existirá um caminho de distância $d(u, v) = 0$, denominado trivial, que começa e termina no mesmo vértice, ou seja, composto somente por um único vértice (v), qualquer, pertencente a um dígrafo. Quando existir um caminho p a partir do vértice (u) até o vértice (v), com comprimento $d(u, v)$ qualquer, diz-se que o vértice (v) é atingível ou acessível a partir de (u) via p.

O teorema definido a seguir é conhecido como Teorema da Desigualdade Triangular e é utilizado como fundamento para o desenvolvimento de muitos algoritmos de busca de caminhos em dígrafos.

Teorema 2.1. Se em um dígrafo, o vértice (v) é acessível de (u) e o vértice (z) é acessível de (v), então $d(u, z) \leq d(u, v) + d(v, z)$.

***Demonstração*:** Observe a Figura 2.3. Pela hipótese do Teorema 2.1, existe um caminho do vértice (u) ao (v) e do vértice (v) ao (z). Percorrendo esses caminhos por sucessão, é possível sair de (u) e chegar a (z) em $d(u, v)+d(v, z)$ passos. Mas, uma vez que a distância $d(u, z)$ requer menos passos para de (u) chegar a (z), então $d(u, z) \leq d(u, v) + d(v, z)$, como se queria demonstrar. ■

Em um dígrafo ponderado, o conceito de distância está relacionado com os valores resultantes obtidos em um caminho pela aplicação de uma operação associativa sobre os valores das arestas que compõem esse caminho, ou seja, $d(p(v_1, v_n)) = \Omega_{i=1}^{n-1} w(v_i, v_{i+1})$, em que Ω representa uma operação associativa qualquer como soma, produto, maximização ou minimização e $w(v_i, v_{i+1})$ representa o peso da aresta (v_i, v_{i+1}).

Para qualquer caminho $p = v_1, v_2, v_3, \ldots, v_n$ não trivial, define-se um subca-

minho $p^{'}$ de p como qualquer caminho $p^{'} = v_i, \ldots, v_j$, tal que $1 \leq i \leq j \leq n$.

Por outro lado, dois caminhos $p(1, j) = v_1, v_2, v_3, \ldots, v_j$ e $p_2 = v_i, \ldots, v_n$ podem ser concatenados[6], produzindo o caminho $p(1, n) = p(1, j) \circ p(i, n)$, se e somente se o vértice de origem do segundo caminho (v_i) coincidir com o vértice de término do primeiro caminho (v_j), ou seja, $(v_i) = (v_j)$.

Na literatura, as definições de ciclo e circuito muitas vezes aparecem com denominação trocadas [40]. Este trabalho, utiliza as Definições 2.19 e 2.20.

Definição 2.19. Um ciclo é um caminho simples não trivial $p_c = v_1, v_2, v_3, \ldots, v_n$ e fechado, ou seja, $v_1 = v_n$.

Definição 2.20. Um circuito é um percurso simples não trivial e fechado.

Portanto, o conceito de ciclo aplica-se somente para dígrafos orientados. Um dígrafo sem ciclos é denominado acíclico.

Como visto anteriormente, em um dígrafo ponderado, a distância de um caminho $p(v_1, v_n)$ é dada por $d(p(v_1, v_n)) = \Omega_{i=1}^{n-1} w(v_i, v_{i+1})$, em que $w(v_i, v_{i+1})$ representa o peso da aresta (v_i, v_{i+1}) e Ω uma operação associativa qualquer. Utilizando-se esse conceito, define-se a distância de um ciclo $c = v_1, v_2, \ldots, v_{c-1}, v_c, v_1$ como $d(c) = \Omega_{i=1}^{c} w(v_i, v_{i+1})$.

Quando os pesos das arestas de um dígrafo podem assumir valores negativos, então pode ocorrer que $d(c) < 0$ e, nesse caso, o ciclo é denominado ciclo de distância negativa, ou simplesmente ciclo negativo. Caso ocorra que $d(c) > 0$, o ciclo é denominado ciclo de distância positiva ou simplesmente ciclo positivo.

Um dígrafo é dito conexo se todos os vértices estão ligados por algum percurso, ou seja, se todos os vértices do dígrafo são acessíveis a partir de todos os demais vértices; e dito desconexo, se houver pelo menos um vértice do dígrafo que não esteja ligado por nenhum percurso a nenhum outro vértice do dígrafo. Um dígrafo é dito fortemente conectado se para todos os vértices do dígrafo, tomado-se qualquer combinação de vértices dois-a-dois, o par de vértices é mutuamente acessível.

A conectividade de um dígrafo é o menor número de vértices cuja remoção desconecta o dígrafo ou o reduz a um único vértice. A conectividade ou mesmo uma estimativa da conectividade de um dígrafo não é um parâmetro fácil de se determinar [40]. Caso o leitor tenha interesse, em [40] estão descritas algumas aproximações.

[6]Este trabalho adota o símbolo $\circ$ para representar a operação de concatenação.

2.1.2.2 Árvore

Uma classe de dígrafos muito utilizada no estudo de roteamento de redes de telecomunicações é a árvore.

Definição 2.21. Uma árvore é um dígrafo conexo D, com ordem $n > 2$, sem ciclos e caracterizado pelas Propriedades 2.9, 2.10, 2.11, 2.12 e 2.13, definidas a seguir.

Propriedade 2.9. D é conexo e sem ciclos.

Propriedade 2.10. D é sem ciclos e tem $n - 1$ arestas.

Propriedade 2.11. D é sem ciclos e por adição de uma aresta se cria um ciclo e somente um.

Propriedade 2.12. D é conexo, mas deixa de sê-lo se uma aresta é suprimida

Propriedade 2.13. Todo par de vértices de D é unido por um e somente um percurso simples.

É interessante comentar que um dígrafo sem ciclos é na verdade uma relação irreflexiva e assimétrica.

Definição 2.22. Uma folha é qualquer vértice de grau 1 que pertence a uma árvore.

Quando o dígrafo desconexo é composto por árvores ele é denominado floresta.

Genericamente, a base $\mathbf{B}$ de um dígrafo $D(\mathbf{V}, \mathbf{E})$ é um subconjunto contido em $\mathbf{V}$, tal que dois vértices quaisquer de $\mathbf{B}$ não são ligados por nenhum caminho e todos vértices não pertencentes a $\mathbf{B}$ podem ser atingidos por um caminho partindo de $\mathbf{B}$. Quando a base de um dígrafo é um conjunto unitário, ela é denominada raiz.

Assim, uma arborescência ou uma árvore enraizada é uma árvore que possui uma raiz.

Uma árvore $D_\pi = (\mathbf{V}_\pi, \mathbf{E}_\pi)$ é dita predecessora de um vértice (v) se houver uma aresta e que conecte o dígrafo D_π ao vértice (v), de maneira que o dígrafo $D = (\{\mathbf{V}_\pi \cup v\}, \{\mathbf{E}_\pi \cup e\})$ seja uma árvore e o vértice (v) um vértice sucessor do vértice sainte da aresta e para qualquer caminho simples até (v).

Em função da orientação da aresta em um caminho, o vértice sainte de uma aresta é denominado antecessor do vértice entrante da mesma aresta. Da mesma

forma, o vértice entrante de uma aresta é denominado sucessor do vértice sainte da mesma aresta.

Partindo-se desse conceito, define-se uma árvore de saída ou sainte, como uma árvore em que as arestas são orientadas tendo a raiz como o vértice predecessor de qualquer árvore derivada dela, e uma árvore de chegada ou entrante, como árvore em que as arestas são orientadas tendo a raiz como o vértice sucessor de qualquer árvore derivada dela. Para simplificar a notação, esta tese utilizará somente a denominação árvore, quando se referir a uma árvore de saída, e árvore de chegada, caso contrário.

Em uma árvore, a maior distância entre o vértice raiz e suas folhas é denominada profundidade ou amplitude da árvore.

Com o advento do computador, as aplicações da Teoria dos Dígrafos sofreram um grande impulso, permitindo a resolução, por meio de algoritmos, de muitos problemas que envolvem dígrafos. A seguir, estão listados alguns problemas com aplicação em redes de telecomunicações.

- O problema de exploração de grafos com respeito à sua conectividade, abordado, entre outros, pelo algoritmo de pesquisa na profundidade descrito na Seção 2.2.3.2 desta tese [40].
- O problema da localização de um vértice pertencente à base de um dígrafo.
- O problema da definição do caminho mais curto entre dois vértices de um dígrafo, abordado, por exemplo, pelos algoritmos de Dijkstra e Bellman-Ford descritos nas Seções 2.2.3.4 e 2.2.3.3 desta tese.
- O problema da definição dos caminhos simples e elementares com restrições, tais como a exclusão ou inclusão ao caminho de arestas ou nós de um dígrafo, limites no valor da distância total do caminho, etc.
- O problema da determinação dos caminhos que unem todos os vértices.
- O problema da determinação, entre dois vértices quaisquer de um dígrafo, dos caminhos simples, elementares e mais otimizados, que obedecem a múltiplas restrições, abordado pelo algoritmo ELND proposto neste trabalho.

Na Seção 2.2 é introduzido o conceito de algoritmo, descrevendo algumas estratégias de análise e alguns exemplos aplicados a este trabalho.

2.2 Conceito de Algoritmo

Embora a palavra algoritmo, derive das palavras em latim *algorismus*, cujo significado é numeração decimal em arábicos, e *arithmós*, cujo significado é número [32], a forma mais próxima do conceito de algoritmo utilizado neste trabalho foi formalizada em 1936 por Alan Turing. Ao definir a máquina de Turing como um modelo abstrato de um computador, tomando-se como base apenas os aspectos lógicos de seu funcionamento (memória, estados e transições), Turing criou uma seqüência bem definida de operações que utilizava um conjunto de valores como entrada e produzia um conjunto de valores como saída [41]. Dessa maneira, um algoritmo pôde ser considerado como uma seqüência de operações que poderiam ser simuladas por uma máquina de Turing completa [42].

Entretanto, este trabalho e alguns autores utilizam um conceito mais moderno de algoritmo que vai além de restringir a sua definição a um conjunto de procedimentos, rotinas ou métodos bem definidos utilizados na resolução de um problema. Um algoritmo deve possuir cinco características importantes[7] [43].

- Finitude: um algoritmo precisa sempre terminar após um número finito de passos.
- Assertividade: cada passo de um algoritmo precisa ser precisamente definido, ou seja, as ações que serão executadas deverão ser especificadas rigorosamente e de forma não ambígua para cada caso e em todos os casos necessários.
- Entrada: um algoritmo tem zero ou mais entradas que são um conjunto de valores providos para ele inicialmente ou dinamicamente.
- Saída: um algoritmo tem uma ou mais saídas que são um conjunto de valores entregues por ele e que estão relacionados com o conjunto de valores utilizados como entrada.
- Efetividade: em geral, se espera que um algoritmo seja efetivo, ou seja, que todas suas operações sejam suficientemente básicas, a ponto de poderem ser executadas manualmente por alguém em um tempo finito.

[7] A definição de algoritmo indicada varia para programação funcional e programação lógica.

2.2.1 Classificação dos Algoritmos

Os algoritmos podem ser classificados de diversas maneiras, em função da implementação, da metodologia ou do paradigma de seu desenvolvimento, do campo de estudo ou aplicação, e da complexidade [42].

2.2.1.1 Classificação em Função da Implementação

Em função da maneira como um algoritmo é implementado, ele pode ser classificado nos cinco tipos, listados a seguir.

- Recursivo ou Iterativo: um algoritmo é recursivo quando ele possui a característica de invocar, repetidamente, a si mesmo até que uma certa condição de término seja satisfeita. Algoritmos iterativos utilizam, freqüentemente, estruturas de repetição como, por exemplo, laços[8] e estruturas de dados adicionais tais como pilhas.

- Lógico: um algoritmo pode ser visto como uma dedução lógica controlada. O componente lógico expressa os axiomas usados na computação e o componente de controle determina a maneira como a dedução é aplicada aos axiomas. Tal conceito é base para a programação lógica.

- Serial ou Paralelo: geralmente se assume que um algoritmo executa suas instruções individualmente, como uma lista de execução, o que constitui um algoritmo serial. Tal conceito é base para a programação imperativa. Por outro lado, existem algoritmos executados paralelamente que levam em conta arquiteturas de computadores com mais de um processador para executar mais de uma instrução ao mesmo tempo. Tais algoritmos dividem os problemas em sub-problemas e os delegam a quantos processadores estiverem disponíveis, agrupando no final o resultado dos sub-problemas em um resultado final para o algoritmo. Esse conceito é base para a programação paralela.

- Determinístico ou Não-determinístico: algoritmos determinísticos resolvem o problema com uma decisão exata a cada passo, enquanto algoritmos não-determinísticos resolvem o problema ao deduzir os melhores passos, por meio de estimativas, sob forma de heurísticas[9].

[8] Tradução do inglês *loop*. Neste contexto, essa definição se refere às estruturas do tipo *while* e *for*, comumente utilizadas em linguagem de programação.

[9] Heurística: "método de programação baseado na aproximação progressiva de um dado problema" [32].

- Exato ou Aproximado: algoritmos que encontram uma resposta exata da verdadeira solução são denominados algoritmos exatos. Por outro lado, alguns algoritmos, denominados aproximados, procuram uma resposta próxima da verdadeira solução, seja por meio de uma estratégia determinística, ou por meio de uma estratégia aleatória. Esses algoritmos possuem aplicações práticas, sobretudo na resolução de problemas muito complexos, para os quais uma resposta exata é inviável.

2.2.1.2 Classificação em Função da Metodologia

Os algoritmos também podem ser classificados em função da metodologia adotada para resolver um problema, conforme listado a seguir.

- Divisão e conquista: algoritmos de divisão e conquista desmembram repetidamente e recursivamente o problema original em vários sub-problemas menores, semelhantes ao original, que são resolvidos e cujas soluções são combinadas de forma a obter a solução do problema original.

- Programação dinâmica: aplica-se essa metodologia quando um problema é desmembrado em subproblemas que não são independentes, ou seja, compartilham sub-subproblemas. Nessa situação, a metodologia de divisão e conquista recalcula várias vezes o mesmo sub-subproblema, não sendo uma metodologia eficiente. A metodologia de programação dinâmica calcula uma só vez o sub-subproblema repetido e grava a sua resposta em uma tabela. Essa metodologia é geralmente aplicada em problemas de otimização [41].

- Guloso: um algoritmo guloso é similar ao de programação dinâmica, entretanto, um algoritmo guloso sempre faz a escolha que parece ser a melhor no momento de decisão. Em outras palavras, ele faz uma escolha ótima para as condições locais, na esperança de que essa escolha leve à solução ótima para a situação global. Como um exemplo de algoritmo guloso pode-se citar os algoritmos de Dijkstra e Bellman-Ford para a busca de caminhos mais curtos a partir de uma origem [41].

- Programação linear: algoritmos de programação linear permitem resolver problemas que podem ser formulados como a ação de maximizar ou minimizar um objetivo, sendo dados recursos limitados e restrições concorrentes. Nesse caso, o objetivo é expresso como uma função linear de certas variáveis e as restrições como desigualdades dessas mesmas variáveis.

- Redução ou "transformação e conquista": a redução resolve o problema ao transformá-lo em outro problema cuja solução já é conhecida.

- Busca e enumeração: vários problemas podem ser modelados como problemas de pesquisa em dígrafos. Pesquisar um dígrafo significa acompanhar sistematicamente suas arestas de modo a alcançar seus vértices ou nós[10][41].

- Paradigma heurístico e probabilístico: algoritmos probabilísticos realizam escolhas aleatoriamente. Algoritmos heurísticos, em geral, não têm como objetivo encontrar uma solução ótima, e sim uma solução aproximada, quando o tempo e os recursos são limitados.

2.2.1.3 Classificação por Campo de Estudo

Os algoritmos também podem ser classificados em função do seu campo de aplicação. Cada aplicação tem problemas específicos, com algoritmos adequados para resolvê-los. Exemplos clássicos são os algoritmos de busca, de ordenação, de análise numérica, de teoria dos grafos, de manipulação de cadeias de texto, de geometria computacional, de análise combinatória, de aprendizagem de máquina, de criptografia, de compressão de dados e de interpretação de texto.

2.2.1.4 Classificação por Complexidade

Uma outra forma de classificar os algoritmos é com relação a complexidade dos problemas que eles abordam. Alguns problemas podem ser resolvidos por algoritmos que são executados em tempo linear, ou seja, o seu tempo de execução é uma função polinomial das n variáveis de entrada. Outros algoritmos, no entanto, são executados em tempo exponencial ou, até mesmo, nunca chegam a uma resposta, não importando quanto tempo lhes seja dado. Em geral, os problemas que podem ser resolvidos por algoritmos de tempo polinomial são ditos tratáveis e o algoritmos que os resolvem são classificados como sendo do tipo *Polynomial time* (P), pois o tempo de execução do pior caso é $O(n^k)$ para alguma constante k. Já os problemas que exigem um tempo superpolinomial são denominados intratáveis ou difíceis e os algoritmos que os resolvem são classificados como sendo do tipo *Non-deterministic Polynomial time* (NP) [41].

Há, porém, uma classe de problemas dita *Non-deterministic Polynomial time - Complete* (NPC) ou NP-Completo, cuja complexidade ainda não pôde ser com-

[10]Na literatura, o termo nó é freqüentemente utilizado como sinônimo de vértice. Neste trabalho ele será utilizado quando um vértice representar um equipamento de uma rede de telecomunicações.

provada, ou seja, ainda não foi possível provar que não existe nenhum algoritmo de tempo polinomial que resolva esses problemas, mas tampouco se encontrou algum algoritmo de tempo polinomial que os resolva.

É importante comentar que, embora não sejam tratáveis, os problemas que pertencem à classe de problemas NP são verificáveis em um tempo polinomial, isto é, dada uma solução qualquer, é possível verificar em um tempo polinomial em função do tamanho da entrada, se ela resolve o problema NP e, portanto, os problemas NP são problemas de decisão em P. Note que os problemas P também são problemas de decisão em P e, portanto, representam um subconjunto dos problemas NP. Por outro lado, os problemas NPC representam um subconjunto de problemas de decisão em NP, ou seja, não se consegue conhecer uma solução significativamente mais rápida para resolver o problema, além de testar todos os subconjuntos das possíveis soluções do problema.

2.2.2 Análise de Algoritmos

A análise de algoritmos estuda as técnicas de projeto de algoritmos e os algoritmos de forma abstrata, sem estarem implementados por uma linguagem de programação específica. Ela se preocupa com os recursos que o algoritmo necessita para sua execução, tais como: memória, largura de banda de comunicação, *hardware* de um computador, permitindo, entre outras coisas, comparar a eficiência de diferentes algoritmos na resolução de um mesmo problema. Os parâmetros utilizados freqüentemente para comparação de eficiência entre algoritmos é o tempo de execução, obtido para as mesmas variáveis de entrada, ou seja, o número de operações primitivas ou etapas executadas de seu pseudocódigo, e o espaço de memória necessário para armazená-lo.

2.2.2.1 Pseudocódigos

Uma forma de expressar um algoritmo, a fim de analisá-lo, é por meio de sua implementação em pseudocódigos. Um pseudocódigo é uma forma genérica de escrever um algoritmo utilizando uma linguagem simples como, por exemplo, a linguagem nativa de quem escreve, sem a necessidade de conhecer a sintaxe de nenhuma linguagem de programação. Em um pseudocódigo emprega-se qualquer método expressivo para especificar, de forma mais clara e concisa, um dado algoritmo, não se preocupando com as questões de engenharia de *software* como, por exemplo, a abstração de dados, a modularidade, o tratamento de erros, etc [42].

Para os pseudocódigos utilizados neste trabalho são adotadas as convenções listadas a seguir.

1. Um recuo ou endentação indica o corpo de um bloco que pode ser ou não repetido em um *loop*[11].

2. Construções de *loop* como **while**, **for** e **repeat**, e as construções condicionais **if**, **then**, **else**, **case** e **return** têm interpretações semelhantes às apresentadas na linguagem Pascal e Matlab[12].

3. O símbolo "$\triangleright$" indica que o restante da linha é um comentário.

4. Uma atribuição é expressa da forma $i \leftarrow n$, em que a variável i recebe o valor n. Dessa maneira, uma atribuição múltipla é expressa da seguinte forma: $i \leftarrow j \leftarrow n$, em que a variável i recebe o valor da variável j, que, por sua vez, recebe o valor n, ou seja, $j \leftarrow n$ e $i \leftarrow j$.

5. As variáveis serão sempre locais, a menos que se indique o contrário.

6. Os elementos de um arranjo são acessados, indicando-se o nome do arranjo seguido dos índices entre colchetes. Exemplo: $P[i][j]$ corresponde ao j-ésimo elemento do i-ésimo elemento do arranjo $P[i]$. $P[i][1 \ldots j]$ corresponde aos elementos $P[1]$, $P[2]$, ..., $P[j]$ do arranjo $P[i]$.

7. Quando um índice não fizer referência a nenhum objeto, ele terá o valor especial NIL.

8. Os parâmetros são passados a um procedimento por meio de seus valores. Um procedimento chamado recebe sua própria cópia dos parâmetros do procedimento chamador e, caso ele atribua um novo valor a esse parâmetro, essa mudança não será vista pelo procedimento chamador se o procedimento chamado não retornar esse parâmetro.

Observe como exemplo, o pseudocódigo InserirOrdenar(A) descrito a seguir, em que é apresentado um algoritmo de ordenação por inserção, cuja variável de entrada é um arranjo $A[1 \ldots n]$, contendo uma seqüência de n números a serem ordenados.

[11]Embora alguns autores utilizem a palavra laço em seus trabalhos como a tradução em português da palavra *loop*, para evitar confusão de notação com o conceito de laço da Teoria dos Dígrafos e da Álgebra Universal, optou-se por utilizar neste trabalho a forma em inglês.

[12]Embora fosse possível utilizar as denominações *while*, *for*, *repeat*, *if*, *then*, *else*, *case* e *return* em português, foram mantidos os termos em inglês com o objetivo de facilitar a interpretação dos pseudocódigos, já que nas linguagens comerciais de programação esses comandos aparecem em inglês.

```
InserirOrdenar(A)
1   for cada (j) ∈ A
2       chave ← A[j]
        ▷ inserir A[j] na seqüencia ordenada A[1 ... j − 1]
3       i ← j − 1
4   While i > 0 e A[i] > chave
5           A[i + 1] ← A[i]
6           i ← i − 1
7   A[i + 1] ← chave
```

Buscando encontrar uma forma simples de contabilizar os recursos necessários para um algoritmo, sem entrar nos detalhes da tecnologia de implementação utilizada, mas ainda sim permitir a análise de suas principais características, considere que cada uma das instruções de um algoritmo, representado por um pseudocódigo, demora um tempo de execução fixo e diferente para cada tipo de operação. Em geral, verifica-se que o tempo de execução do algoritmo cresce com o tamanho da sua entrada [41].

O tamanho da entrada varia de um algoritmo para outro, em função do problema abordado. Em muitos casos, esse tamanho representa o número de elementos de uma mesma natureza como, por exemplo, o número de amostras de um sinal para o cálculo de sua série de Fourier. Em outros casos, como, por exemplo, para um algoritmo que trabalha com dígrafos, o tamanho da entrada pode ser descrito pelo número de vértices ou arestas existentes no dígrafo analisado.

No entanto, para uma entrada definida, o tempo de execução de um algoritmo é dado pelo número de etapas ou operações primitivas executadas por ele e expressa em seu pseudocódigo. Embora operações diferentes tenham tempos diferentes de execução, para uma mesma operação o tempo de execução é constante. Assim, dada uma entrada de tamanho n, qualquer, é possível contabilizar o tempo $T(n)$ de execução total de um algoritmo, em função do tamanho da entrada e das constantes de tempo de cada operação primitiva.

Observe o exemplo a seguir, em que é ilustrado o pseudocódigo InserirOrdenar(A) com os respectivos tempos/ custos de cada operação e o correspondente número de vezes que ela é executada.

InserirOrdenar(A)	tempo/custo	vezes
1 **for** cada $(j) \in A$	c_1	n
2 $chave \leftarrow A[j]$	c_2	$n-1$

3	$i \leftarrow j - 1$	c_3	$n - 1$
4	**While** $i > 0$ e $A[i] > chave$	c_4	$\sum_{j=2}^{n} b_j$
5	$A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(b_j - 1)$
6	$i \leftarrow i - 1$	c_6	$\sum_{j=2}^{n}(b_j - 1)$
7	$A[i+1] \leftarrow chave$	c_7	$n - 1$

O tempo de execução do algoritmo InserirOrdenar(A) é a soma dos tempos de execução de cada instrução, multiplicada pelo número de vezes que a instrução é executada, ou seja,

$$T(n) = c_1 \times n + c_2 \times (n-1) + c_3 \times (n-1) + c_4 \times \sum_{j=2}^{n} b_j + c_5 \times \sum_{j=2}^{n}(b_j - 1) + c_6 \times \sum_{j=2}^{n}(b_j - 1) + c_7 \times (n-1) \tag{2.2}$$

Considerando o pior caso, ou seja, quando o arranjo $A[1 \ldots n]$ está em ordem inversa ou ordem decrescente, em que deve-se comparar cada elemento $A[j]$ do arranjo com cada elemento do sub-arranjo ordenado $A[1 \ldots j-1]$ e, portanto $b_j = j$ para $2, 3, \ldots, n$. Como $\sum_{j=2}^{n} j = n \times (n-1)/2 - 1$ e $\sum_{j=2}^{n}(j-1) = n \times (n-1)/2$, verifica-se que $T(n)$ pode ser escrito na forma $a \times n^2 + b \times n + c$, ou seja, o tempo de execução total é uma função quadrática de n.

Na maioria do casos em que se analisa um algoritmo em comparação com outros, busca-se saber um valor aproximado do seu tempo de execução [43], seja pela comparação do pior caso, isto é, o tempo de execução mais longo para qualquer entrada de tamanho n, seja pela comparação do caso médio, isto é, o tempo de execução do caso mais esperado ou daquele que ocorre em média. Embora seja possível em muitos casos obter a função $T(n)$ de um algoritmo, a precisão muitas vezes não é necessária, pois, para entradas muito grandes, os termos de menor ordem podem ser desprezados. Uma forma simplificada de obter a aproximação do tempo de execução do pior caso ou do caso médio é calcular a taxa de crescimento ou a ordem de crescimento da função $T(n)$, ou seja, o grau do termo n de maior ordem [41]. Observe que no exemplo adotado a ordem de crescimento da função $T(n)$ é 2.

A análise da eficiência dos algoritmos com tamanhos de entradas grandes o suficiente para se desprezar os demais termos de menor ordem da função $T(n)$ é denominada análise assintótica. Várias notações são propostas para essa análise, a seguir serão descritas somente as convenções adotadas neste trabalho.

2.2.2.2 Representações Assintóticas - Notação θ

Embora, em alguns casos, se analise o tempo de execução de um algoritmo utilizando os casos esperados, isto é, os que ocorrem em média, este trabalho adota como referência o pior caso, já que esse é o limite superior do tempo de execução do algoritmo para qualquer entrada, ou seja, espera-se que o tempo de execução do algoritmo nunca será muito pior que esse valor.

Dessa forma, a descrição do tempo de execução assintótica de um algoritmo é definida em termos de funções, cujos domínios são o conjunto dos números naturais que em geral representam o tamanho das entradas do algoritmo. Assim, define-se a função $\theta(g(n))$ como:

Definição 2.23. $\theta(g(n))$ é um conjunto de funções, tal que $\theta(g(n)) = \{f(n)$, em que existem constantes positivas c_1, c_2 e n_0, tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0\}$.

Observe que, embora $\theta(g(n))$ seja um conjunto de funções $f(n)$ e, portanto, $f(n) \in \theta(g(n))$, adota-se a notação $f(n) = \theta(g(n))$ para expressar o mesmo significado de pertinência.

Portanto, uma função $f(n)$ pertence ao conjunto $\theta(g(n))$ ($f(n) = \theta(g(n))$ se existirem constantes positivas c_1 e c_2, tais que permitam à função $f(n)$ estar limitada entre as funções $c_1 g(n)$ e $c_2 g(n)$, para valores de n suficientemente grandes, ou seja, maiores que n_0. Nesse caso, define-se que $g(n)$ é um limite assintoticamente restrito para $f(n)$. Observe que, segundo essa definição, $f(n)$ deve ser uma função positiva para n suficientemente grande, ou seja, uma função assintoticamente positiva, em que os termos de mais baixa ordem, bem como o coeficiente do termo de mais alta ordem podem ser ignorados na determinação do limite assintoticamente restrito [41].

2.2.2.3 Representações Assintóticas - Notação O

A notação O limita assintoticamente por cima uma função, ou seja, utiliza-se a notação O para dar um limite superior sobre uma função.

Definição 2.24. $O(g(n)) = \{f(n)$, em que existem constantes positivas c e n_0, tais que $0 \leq f(n) \leq cg(n)$ para todo $n \geq n_0\}$.

Note que para indicar que a função $f(n)$ pertence ao conjunto de funções O(g(n)), escreve-se $f(n) = O(g(n))$. Observe também que a restrição imposta

pela notação $\theta(g(n))$ é mais forte que a imposta pela notação $O(g(n))$, pois $\theta(g(n)) \subseteq O(g(n))$.

Observação 2.1. Embora seja um abuso dizer que o tempo de execução do pior caso de um algoritmo é $O(g(n))$, quando utilizado neste trabalho, isso significa que existe uma função $f(n)$ que é $O(g(n))$, tal que para qualquer valor de n, não importando a entrada específica de tamanho n utilizada, o tempo de execução desse algoritmo tem um limitante superior determinado por $g(n)$.

2.2.3 Algoritmos de Roteamento

Dentre os diversos tipos de algoritmos classificados anteriormente, este trabalho aborda fundamentalmente os algoritmos de busca de caminhos elementares, simples e otimizados de única origem em grafos[13], genericamente denominados neste trabalho como algoritmos de roteamento, que vêm sendo amplamente estudados ao longo da história, em função da sua aplicabilidade em diferentes campos de pesquisa, [44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55].

Com objetivo didático, esta seção sub-classifica os algoritmos de roteamento em função da sua implementação: centralizado ou distribuído [56]; e em função de sua metodologia de seleção de rotas: pela origem (explícito) ou pelo destino (*hop-by-hop*) [57, 58, 59, 60].

A utilização dos algoritmos de roteamento centralizado implica que uma entidade central tenha o conhecimento de todo o dígrafo e seu respectivo estado e, com isso, compute e distribua para cada vértice do dígrafo os caminhos escolhidos. Esse tipo de algoritmo é usualmente implementado em ferramentas de apoio no planejamento de redes, que simulam e analisam o comportamento das redes em função de uma matriz de tráfego e de eventos de simulação de falhas [61, 62, 63], ou em sistemas de gerência de rede como apoio à ativação de serviços [64, 65, 56].

Por outro lado, nos algoritmos de roteamento distribuído, cada nó da rede conhece o dígrafo ou uma abstração do mesmo, computando e selecionando, de forma individual e independente, os caminhos na rede. Como exemplo clássico dessa abordagem podem ser citados os algoritmos de roteamento IP.

Nos algoritmos de roteamento pela origem, o nó de origem calcula e seleciona

[13]Quando se utiliza como parâmetros de otimização a distância entre dois nós, o tempo, o custo, a penalidade, perdas ou qualquer outra quantidade que se acumule linearmente ao longo do caminho e que alguém deseje minimizar, esse problema é conhecido como problema de caminhos mais curtos de única origem.

os caminhos para cada destino na rede. Esses caminhos são, então, explicitamente estabelecidos ou declarados com a informação a ser transmitida, de maneira que os demais nós da rede simplesmente encaminham a informação com base no caminho já pré-definido. Como descrito no Capítulo 1, essa é uma abordagem utilizada no modelo IntServ de implementação da qualidade de serviço.

Já nos algoritmos de roteamento pelo destino, também denominados roteamento *hop-by-hop*, cada nó da rede, como nos algoritmos de roteamento pela origem, calcula e seleciona os caminhos para cada destino. Entretanto, como a escolha do caminho para o qual a informação será enviada é feita, independentemente, nó-a-nó, à medida em que a informação trafega na rede, o caminho definido pelo nó de origem pode não corresponder ao caminho realizado de fato pela informação ao longo da rede.

Em função do tipo de informação manipulada pelos algoritmos de roteamento *hop-by-hop*, eles são sub-classificados como: algoritmos de roteamento global, quando o cálculo do caminho é baseado no conhecimento detalhado e completo de todos os enlaces da rede; e algoritmos de roteamento descentralizado, quando o cálculo do caminho é feito de maneira acumulativa e distribuída, baseado na abstração acumulada da rede proporcionada pelos nós adjacentes. Como exemplo clássico de algoritmos de roteamento global pode-se citar o algoritmo de Dijkstra, no qual se baseia os protocolos de roteamento *Link State* (LS): *Open Shortest Path First* (OSPF) e *Intermediate System to Intermediate System* (IS-IS). Como exemplo de algoritmos de roteamento descentralizado pode-se citar o algoritmo de Bellman-Ford, no qual estão baseados os protocolos de roteamento *Distance Vector* (DV): *Routing information Protocol* (RIP), *Internet Gateway Routing Protocol* (IGRP) [66, 67].

Cabe observar que alguns autores [68, 69] também classificam os algoritmos de roteamento como sendo estáticos ou dinâmicos, referindo-se, na verdade, ao conjunto dos processos de seleção de caminhos, atualização e distribuição da informação de topologia da rede. É importante notar que este trabalho faz a seguinte distinção entre algoritmos de roteamento e protocolos de roteamento. Protocolo de roteamento é o conjunto de normas e especificações técnicas implementadas em uma rede que permite a manutenção das informações necessárias para que se possa executar um algoritmo de roteamento. Algoritmos de roteamento são um conjunto de processos que, obedecendo as propriedades descritas no ítem 2.2, permitem a seleção de rotas ou caminhos em uma rede, em função de alguns parâmetros ou métricas.

Em função da quantidade de restrições imposta na busca do melhor caminho, um algoritmo de roteamento é também classificado como mono-restritivo, quando analisa somente uma restrição na busca do melhor caminho, ou multi-restritivo, quando analisa mais de uma restrição na busca do melhor caminho.

Devido à vasta utilização, de alguns desses algoritmos de roteamento, na solução de uma grande variedade de problemas e, em particular, problemas de roteamento em redes de telecomunicações, as próximas seções descrevem de forma resumida os algoritmos de roteamento de Bellman-Ford, Dijkstra, bem como os algoritmos de pesquisa primeiro na largura e primeiro na profundidade, devido ao seu caráter didático e à sua aplicação no desenvolvimento do algoritmo ELND proposto neste trabalho.

2.2.3.1 Pesquisa Primeiro na Largura

Dado um dígrafo $D = (\mathbf{V}, \mathbf{E})$, o algoritmo de pesquisa primeiro na largura tem como objetivo buscar para um determinado vértice (s) todos os vértices acessíveis por ele no dígrafo. Ele explora sistematicamente as arestas do dígrafo para montar uma árvore primeiro na largura com raiz em (s), considerando o menor número de arestas entre o vértice (s) e qualquer outro vértice (u) do dígrafo.

O algoritmo de pesquisa primeiro na largura, *Breadth First Search* (BFS), descrito abaixo, funciona para dígrafos orientados e não orientados e recebe esse nome porque amplia uniformemente a fronteira entre os vértices descobertos e os não descobertos, ou seja, ele descobre todos os vértices a k arestas a partir do vértice (s) antes de descobrir quaisquer vértices a $k+1$ arestas a partir do vértice (s) [41].

BFS$(D, (s))$

1 **for** cada vértice $(u) \in V[D] - (s)$ ▷ inicializa estado dos vértices
  ▷ BRANCO: vértices não descobertos
  ▷ CINZA: vértices descobertos com algum vértice adjacente não descoberto
  ▷ PRETO: vértices descobertos com nenhum vértice adjacente não descoberto

2 $cor[(u)] \leftarrow BRANCO$

3 $d[(u)] \leftarrow \infty$ ▷ inicializa variável que contém a distância entre o vértice de origem (s) e um vértice (u), ou seja, o número de arestas entre os vértices (s) e (u)

```
4   cor[(s)] ← CINZA
5   d[(s)] ← 0
6   Q ← ∅    ▷ inicializa a lista de "vértices descobertos com algum
                vértice adjacente não descoberto"
7   InsereNaFila (Q, (s))
8   while Q ≠ ∅    ▷ enquanto houver vértices CINZA
9         (u) ← RemoveDaFila (Q)
10        for cada (v) ← Adj[(u)]    ▷ para cada vértice (v) adjacente ao
                                       vértice (u) verifica se ele já foi descoberto
11            if cor[(v)] = BRANCO then
12               cor[(v)] ← CINZA
13               d[(v)] ← d[(u)] + 1
14               π[(v)] ← (u)    ▷ armazena o predecessor de (u). Se (u) = (s),
                                    então π[(v)] = NIL
15               InsereNaFila (Q, (v))    ▷ inserir vertice adjacente na lista
16        cor[(u)] ← PRETO
```

```
RemoveDaFila (Q)
1   (s) ← Q[inicio[Q]]
2   if inicio[Q] = comprimento[Q] then
3      inicio[Q] ← 1
4   else inicio[Q] ← inicio[Q] + 1
5   return (s)
```

```
InsereNaFila(Q, (s))
1   Q[fim[Q]] ← (s)
2   if fim[Q] = comprimento[Q] then
3      fim[Q] ← 1
4   else fim[Q] ← fim[Q] + 1
```

O algoritmo BFS$(D, (s))$, primeiramente, inicializa todos os vértices com a cor $BRANCO$, exceto o vértice de origem (s), que começa com a cor $CINZA$, e a distância $d[(.)]$ entre o vértice de origem e todos os demais vértices como sendo ∞, exceto a distância do vértice de origem com ele mesmo que é 0.

Em seguida, ele percorre todos os vértices adjacentes ao vértice de origem, armazenado-os com cor $CINZA$ e $d[(.)] = 1$. Depois, para cada vértice $CINZA$ de $d[(.)] = 1$, o algoritmo busca seus respectivos vértices adjacentes e, ao terminar, passa a cor do vértice analisado para $PRETO$. Se esses vértices adjacentes ainda

não foram descobertos, o algoritmo altera a cor desses vértices para $CINZA$ e $d[(.)] = 2$. O algoritmo segue com esse mesmo processo até que se tenha varrido todos os vértices do dígrafo.

Observe que o algoritmo BFS$(D, (s))$ constrói uma árvore primeiro na largura com raiz em (s) representada pela variável π em cada vértice, ou seja, para um dígrafo $D = (\mathbf{V}, \mathbf{E})$ com origem em (s), define-se o subdígrafo $D_\pi = (\mathbf{V}_\pi, \mathbf{E}_\pi)$ como predecessor de D, tal que $\mathbf{V}_\pi = \{(v) \in \mathbf{V} : \pi[(v)] \neq NIL\} \cup \{(s)\}$ e $\mathbf{E}_\pi = \{\{(\pi[(v)], (v)) : (v) \in \mathbf{V}_\pi\} - (s)\}$. Na linha 14, $\pi[(v)] \leftarrow (u)$ se e somente se a aresta $(u, v) \in \mathbf{E}$, (v) é acessível a partir de (s) e (u) ainda não pertence a $\pi[(v)]$. Assim, como $D_\pi = (\mathbf{V}_\pi, \mathbf{E}_\pi)$ contém um único caminho simples desde (s) a qualquer vértice (v), D_π é uma árvore primeiro na largura com raiz em (s).

A algoritmo BFS, com tempo de execução $O(V + E)$, em que V é o número de vértices e E o número de arestas do dígrafo $D = (\mathbf{V}, \mathbf{E})$, é um dos algoritmos mais simples de pesquisa em dígrafos e serve de modelo para algoritmos mais conhecidos como, por exemplo, o algoritmo de Dijkstra, descrito na Seção 2.2.3.4, uma vez que a árvore primeiro na largura é uma árvore de caminhos mais curtos [41].

2.2.3.2 Pesquisa Primeiro na Profundidade

Dado um dígrafo $D = (\mathbf{V}, \mathbf{E})$, o algoritmo de pesquisa primeiro na profundidade busca as arestas a partir do vértice (v), mais recentemente descoberto e que ainda tem arestas a descobrir. Quando todas as arestas de (v) foram exploradas, o algoritmo recomeça a busca das arestas que deixam o vértice, a partir do qual (v) foi explorado. Se ainda assim não houver mais arestas, o algoritmo busca o próximo vértice não descoberto e reinicializa a busca pela profundidade a partir desse novo vértice. A idéia é procurar o mais profundamente no dígrafo até encontrar todos os vértices.

Observe que esse procedimento gera uma nova sub-árvore primeiro na profundidade sempre que a primeira busca não conseguir varrer todos os vértices do dígrafo. Se o dígrafo D é conexo, então sempre que um novo vértice (v) for descoberto, há ao menos uma aresta que o conecta às demais árvores primeiro na profundidade, já encontradas. A seguir, como referência, está descrito em pseudocódigo o algoritmo de pesquisa primeiro na profundidade, *Depth First Search* (DFS):

DFS(D)

1 **for** cada vértice $(u) \in V[D]$ ▷ inicializa estado dos vértices
  ▷ BRANCO: vértices não descobertos
  ▷ CINZA: vértices descobertos com algum vértice adjacente não descoberto
  ▷ PRETO: vértices descobertos com nenhum vértice adjacente não descoberto
2 $cor[(u)] \leftarrow BRANCO$
3 $\pi[(u)] \leftarrow NIL$ ▷ inicializa as listas de adjacência
4 **for** cada vértice $(u) \in V[D]$ ▷ visita todos os vértices do dígrafo
5 **if** $cor[(u)] = BRANCO$ **then** VISITA((u))

VISITA((u))
1 $cor[(u)] \leftarrow CINZA$ ▷ indica que o vértice (u) acaba de ser descoberto
2 **for** $(v) \in Adj[(u)]$ ▷ explora a aresta (u, v)
3 **if** $cor[(u)] = BRANCO$ **then**
4 $\pi[(v)] \leftarrow (u)$
5 VISITA((v))
6 $cor[(u)] \leftarrow PRETO$

O tempo de execução do algoritmo DFS é dado por $\Theta(V + E)$, em que V é o número de vértices e E é o número de arestas do dígrafo $D = (\mathbf{V}, \mathbf{E})$. Note que a inicialização (linhas 1 a 3) demora um tempo $\Theta(V)$. Durante o *loop* entre das linhas 4 a 5, o procedimento VISITA((u)) é executado $|Adj[v]|$ vezes, em que $\sum_{v \in \mathbf{V}} |Adj[v]| = \Theta(E)$, resultando em uma tempo total de execução $\Theta(E)$ entre das linhas 4 e 5 [41].

2.2.3.3 Bellman-Ford

O algoritmo de Bellman-Ford baseia-se nos procedimentos propostos por Bellman (1957) e Ford (1956) e fundamentados na metodologia de programação dinâmica. Ele resolve o problema de busca de caminhos mais curtos de única origem em um dígrafo orientado ponderado $D = (\mathbf{V}, \mathbf{E})$, com uma função peso, tal que $E \rightarrow \Re$[14] e, portanto, os pesos w das arestas podem ser negativos, desde que não haja nenhum ciclo de distância negativa a partir do vértice de origem (s), pois o problema de busca de caminhos mais curtos de única origem em dígrafos com ciclos de distância negativa é do tipo NPC [70].

[14]Denota-se $\Re$ como o conjunto dos números reais, $\Re^+$ como o conjunto dos número reais positivos e $\Re_0^+$ como o conjunto dos números reais não negativos, ou seja $\Re^+ \cup 0$.

Como uma variedade de outros algoritmos de busca de caminhos mais curtos de única origem, o algoritmo de Bellman-Ford baseia-se na propriedade de desigualdade triangular ou condição de relaxamento definida no Teorema 2.1 [41, 71].

O pseudocódigo BELLMAN-FORD$(D, w, (s))$, descrito a seguir, considera de início que o vértice de origem (s) tem distância zero e todos os demais vértices têm distância infinita. Então, para todas as arestas existentes no dígrafo, aplica-se a operação de relaxamento, diminuindo progressivamente a estimativa $d[(v)]$ do peso do caminho com origem no vértice (s) para cada vértice $(v) \in \mathbf{V}$, até alcançar o peso do caminho mais curto.

```
BELLMAN-FORD (D, w, (s))
1   for cada vértice (v) ∈ V[D]     ▷ Inicialização
2       d[(v)] ← ∞
3       π[(v)] ← NIL
4   d[(s)] ← 0
5   for i ← 1 até |V[D]| − 1     ▷ para os vértices existentes, menos a origem,
                                     relaxa as arestas entrantes
6       for cada aresta (u, v) ∈ E[D]     ▷ relaxa as arestas existentes
7           RELAXA((u), (v), w)
8   for cada aresta (u, v) ∈ E[D]     ▷ Verifica se há ciclo de distância negativa
9       if d[(v)] > d[(u)] + w(u, v) then return FALSO
10      else return VERDADEIRO
```

```
RELAXA((u), (v), w)
1   if d[(v)] > d[(v)] + w(u, v) then
2      d[(v)] ← d[(v)] + w(u, v)
3      π[(v)] ← (u)
```

Observe que, caso haja algum ciclo de distância negativa, o algoritmo de Bellman-Ford retorna indicando $FALSO$, isto é, indicando que a árvore de caminho mais curto encontrada é falsa, pois quando há algum ciclo de distância negativa, o pseudocódigo RELAXA$((u), (v), w)$ sempre retornará um valor menor do que o obtido em um momento anterior e, portanto, o algoritmo nunca converge para uma árvore de caminho mais curto com origem (s).

Sendo uma referência como o algoritmo de menor complexidade para resolver o problema de busca de caminhos mais curtos de única origem em um dígrafo

orientado ponderado com arestas de pesos negativos, é interessante observar que o tempo de execução no pior caso para o algoritmo de Bellman-Ford é $O(V \times E)$. Note que a inicialização (linhas 1 a 4) demora o tempo $\theta(V)$, cada aresta entrante a um vértice é relaxada uma vez (linhas 5 a 7) com tempo $O(V \times E)$ e leva-se o tempo de $\theta(E)$ para verificar que não existe ciclos de distância negativa. Portanto, no pior caso, o limitante superior do tempo de execução desse algoritmo é dado por $O(V \times E)$.

2.2.3.4 Dijkstra

Como o algoritmo de Bellman-Ford, o algoritmo de Dijkstra resolve o problema de busca de caminhos mais curtos de única origem em um dígrafo orientado e ponderado $D = (\mathbf{V}, \mathbf{E})$. Entretanto, diferentemente do algoritmo de Bellman-Ford, no algoritmo de Dijkstra a função peso $E \rightarrow \Re_0^+$ permite somente arestas de peso w não negativo.

O algoritmo de Dijkstra trabalha com uma metodologia gulosa, baseando-se na propriedade de uma subestrutura ótima, na qual um caminho mais curto entre dois vértices origem-destino é composto por sub-caminhos, que também são caminhos mais curtos entre o vértice de origem e qualquer outro vértice que compõe o caminho estendido.

O pseudocódigo DIJKSTRA(D, s, w), descrito a seguir, considera de início que o vértice de origem (s) tem distância zero e todos os demais vértices têm distância infinita. $\mathbf{S}$ contém o conjunto de vértices, cujos pesos dos respectivos caminhos mais curtos até a origem já são conhecidos, e o conjunto $\mathbf{Q} = \mathbf{V} - \mathbf{S}$. Então, repetidamente, o algoritmo seleciona do conjunto $\mathbf{S}$ a aresta sainte de menor peso e o respectivo vértice de chegada (u), que é adicionado ao conjunto S e cujas arestas saintes são relaxadas.

DIJKSTRA(D, s, w)
1 **for** cada vértice $(v) \in V[D]$ ▷ Inicialização
2 $\quad d[(v)] \leftarrow \infty$
3 $\quad \pi[(v)] \leftarrow NIL$
4 $d[(s)] \leftarrow 0$
5 $S \leftarrow \emptyset$
6 $Q \leftarrow V[D]$
7 **while** $Q \neq \emptyset$
8 $\quad (u) \leftarrow$ EXTRAI-MIN(Q) ▷ extrai o vértice cuja aresta entrante tem menor peso

9 $S \leftarrow S \cup u$

10 **for** cada vértice $(v) \in Adj[(u)]$

11 RELAXA$((u), (v), w)$

Observe que os vértices só são inseridos no conjunto **Q** na linha 6, o *loop* entre as linhas 7 e 11 repete V vezes, em que V é o número de vértices do dígrafo $D = (\mathbf{V}, \mathbf{E})$. Por outro lado, extrair do conjunto **Q** o vértice cuja aresta entrante tem menor peso, depende de uma seleção de vértices que corresponde a no mínimo $V + (V - 1) + (V - 2) + \cdots + 1$ interações. Assim, no total, o tempo de execução, no pior caso, para o algoritmo de Dijkstra é $O(V^2)$ [70].

2.2.4 Convergência

Em termos genéricos, convergência significa a tendência de uma abordagem a um ponto definido, seja este um valor no tempo, uma posição no espaço, ou um estado de equilíbrio [72, 32].

Em matemática, a convergência descreve um comportamento limitado, particularmente em uma seqüência ou série infinita, em que a convergência significa a aproximação ou a tendência a um valor limite.

Em algoritmos de roteamento, a convergência é o processo de concordância realizado por todos os roteadores[15] da rede na definição do melhor caminho. Em outras palavras, quando um evento de rede impede a continuidade de um caminho ou proporciona o surgimento de novos caminhos, os roteadores distribuem na rede mensagens de atualização, provocando o recálculo do melhor caminho por meio de um algoritmo de roteamento. Nesse caso, se o algoritmo de roteamento utilizado converge, ou seja, há coerência entre os roteadores na definição do melhor caminho entre dois ou mais elementos de rede, a informação direcionada por um roteador a um caminho não será enviada de volta a esse roteador por outro roteador da rede [67]. No entanto, quando não há convergência, a informação encaminhada por um roteador é enviada de volta a ele, por meio de outros roteadores que pertencem ao ciclo formado pelo caminho percorrido pela informação até retornar ao roteador "de origem" do ciclo.

Definição 2.25. O envio contínuo de uma informação dentro de um ciclo de roteadores é denominado *loop* de roteamento ou simplesmente *loop*[67].

[15]Este trabalho denomina roteador qualquer elemento em uma rede de telecomunicações com capacidade de comparar e definir os caminhos pelos quais a informação será enviada entre dois ou mais elementos de rede.

2.3 Álgebra

Segundo a Álgebra Universal, que é uma ramo da matemática que estuda as estruturas comuns em todos os ramos da Álgebra, uma álgebra ou uma estrutura algébrica consiste em um conjunto associado a uma ou mais operações, que satisfazem certos axiomas.

Algumas estruturas algébricas são definidas com mais de um conjunto como, por exemplo, um espaço vetorial, que tem dois conjuntos: o conjunto dos vetores e dos escalares. Dependendo das operações e axiomas obedecidos, as estruturas algébricas recebem diferentes denominações. Por exemplo:

- grupóide: uma estrutura $\langle \mathbf{S}, \oslash \rangle$, formada por um conjunto $\mathbf{S}$ com uma única operação binária $\oslash$[16];
- quase-grupo: um grupóide no qual a divisão é sempre possível;
- laço[17]: um quase-grupo com um elemento neutro;
- semigrupo: uma estrutura $\langle \mathbf{S}, \odot \rangle$, formada por um conjunto $\mathbf{S}$ sobre o qual está definida uma operação binária e associativa $\odot$;
- monóide: um semigrupo com um elemento neutro denominado identidade;
- grupo: um monóide, no qual cada elemento tem um inverso ou, o que é equivalente, um laço associativo;
- grupo abeliano: um grupo que obedece a comutatividade;
- anel: uma estrutura $\langle \mathbf{S}, +, \otimes \rangle$, formada por um conjunto $\mathbf{S}$ com uma operação de grupo abeliano, definida como adição $(+)$, e uma operação de semigrupo definida como a multiplicação $\otimes$ e que seja distributiva;
- corpo: um anel no qual os elementos não-zero formam um grupo abeliano sob a multiplicação;
- reticulado: um conjunto com duas operações comutativas, associativas e idempotentes, que satisfazem a lei de absorção, definida adiante na Propriedade 2.17; e

[16]Em matemática, uma operação é qualquer tipo de procedimento que é realizado sobre certa quantidade de elementos, e que obedece sempre a uma mesma lógica ou regra. Conforme o número de termos necessários em uma operação, essa pode ser classificada como operação unária, operação binária, operação ternária e assim por diante[73].

[17]A definição de laço dada nesta seção pela Álgebra Universal é diferente da definição dada pela Teoria dos Dígrafos na Seção 2.1.

- álgebra booleana: um reticulado limitado, distributivo e complementado.

A maioria dessas estruturas não tem aplicação direta neste trabalho. Assim, esta seção descreve alguns conceitos básicos para a introdução à Álgebra de Caminhos, a partir do conceito de reticulado, já definido, sob a perspectiva da Teoria da Ordenação, na Seção 2.1.1.1.

Definição 2.26. Um reticulado é uma estrutura $\langle \mathbf{L}, \vee, \wedge \rangle$, formada por um conjunto $\mathbf{L}$ não vazio equipado com duas operações binárias, união ou soma $\vee$ e intersecção ou produto $\wedge$, aplicáveis entre quaisquer elementos x, y e z de L, que satisfaçam as Propriedades 2.14, 2.15, 2.16 e 2.17 [39], definidas a seguir.

Propriedade 2.14. Comutatividade: $x \vee y = y \vee x$ e $x \wedge y = y \wedge x$.

Propriedade 2.15. Associatividade: $x \vee (y \vee z) = (x \vee y) \vee z$ e $x \wedge (y \wedge z) = (x \wedge y) \wedge z$.

Propriedade 2.16. Idempotência: $x \vee x = x$ e $x \wedge x = x$.

Propriedade 2.17. Absorção: $x = x \vee (x \wedge y)$ e $x = x \wedge (x \vee y)$.

A definição de reticulado dada na Seção 2.1.1.1 relaciona-se com essa quando $\mathbf{L}$ é um conjunto parcialmente ordenado, em que para qualquer par de elementos x e y, tem-se obedecida a expressão 2.3.

Teorema 2.2. A estrutura $\langle \mathbf{L}, \vee, \wedge \rangle$, formada por um conjunto $\mathbf{L}$ não vazio e equipado com duas operações binárias, $\vee$ e $\wedge$, aplicáveis entre quaisquer elementos $x, y, z \in L$, obedecendo as propriedades de comutatividade, associatividade, idempotência e absorção, é um conjunto parcialmente ordenado, que para quaisquer dois elementos x e y há pelo menos um elemento supremo $x \vee y$ e um elemento ínfimo $x \wedge y$, se

$$x \preceq y \Leftrightarrow x = x \wedge y. \tag{2.3}$$

***Demonstração*:** Para provar que um reticulado segundo a Álgebra Universal corresponde ao reticulado definido na Seção 2.1.1.1, parte-se da definição dada pela Álgebra Universal em que: $\mathbf{L}$ é um conjunto equipado com as operações binárias $\vee$ e $\wedge$, que obedece as propriedades de comutatividade, associatividade, idempotência e absorção.

Suponha que R seja uma relação binária qualquer entre os elementos do conjunto $\mathbf{L}$, tal que:

$$x \; R \; y \Leftrightarrow x = x \wedge y. \tag{2.4}$$

Por meio da propriedade de idempotência, tem-se que $x = x \wedge x$. Aplicando-se diretamente a suposição 2.4, tem-se que $x \; R \; x$, ou seja, a relação R é reflexiva.

Por outro lado, a suposição 2.4 pode ser reescrita como $y \; R \; x \Leftrightarrow y = y \wedge x$. Por meio da propriedade comutativa $y = y \wedge x \Leftrightarrow y = x \wedge y$. Portanto, tem-se que $x \; R \; y \Leftrightarrow x = x \wedge y$ e $y \; R \; x \Leftrightarrow y = x \wedge y$, ou seja, $x = y$ e, dessa forma, conclui-se que a relação R é assimétrica.

Além disso, se $x \; R \; y$ e $y \; R \; z$, pela suposição 2.4 tem-se, respectivamente, que $x = x \wedge y$ e $y = y \wedge z$. Substituindo-se $y = y \wedge z$ em $x = x \wedge y$, tem-se $x = x \wedge (y \wedge z)$ que, pela propriedade associativa, resulta em $x = x \wedge (y \wedge z) = (x \wedge y) \wedge z$. Entretanto, como $x = x \wedge y$, então $x = x \wedge (y \wedge z) = (x \wedge y) \wedge z = x \wedge z$, ou seja, $x = x \wedge z$ que, pela suposição 2.4, implica em $x \; R \; z$. Assim, $x \; R \; y$ e $y \; R \; z$ implicam em $x \; R \; z$ e, portanto, a relação R é transitiva.

Como R é uma relação binária reflexiva, assimétrica e transitiva, então R é uma relação de ordenação e a suposição 2.4 pode ser reescrita como 2.3.

Por meio da propriedade de absorção, tem-se que $x = x \wedge (x \vee y)$, $y = y \wedge (y \vee x)$ e, pela propriedade de comutatividade, $y = y \wedge (y \vee x) \Rightarrow y = y \wedge (x \vee y)$. Portanto, reescrevendo a suposição 2.3, tem-se:

$$x = x \wedge (x \vee y) \Rightarrow x \preceq x \vee y \text{ e}$$
$$y = y \wedge (x \vee y) \Rightarrow y \preceq x \vee y.$$

Assim, sendo $\mathbf{U}$ um subconjunto de $\mathbf{L}$, tal que $\mathbf{U} = \{x, y\}$, então $x \vee y$ é majorante.

Por outro lado, se $x \preceq m$ e $y \preceq m$, pela suposição 2.3 tem-se, respectivamente, que $x = x \wedge m$ e $y = y \wedge m$. Portanto, utilizando as propriedades comutativa e absorção, pode-se escrever:

$$x \vee m = (x \wedge m) \vee m \Rightarrow m \vee (m \wedge x) = m \text{ e}$$
$$y \vee m = (y \wedge m) \vee m \Rightarrow m \vee (m \wedge y) = m.$$

Por meio da propriedade de idempotência, tem-se que $m \vee m = m \Rightarrow (x \vee m) \vee (y \vee m) = m$. Utilizando-se as propriedades associativa e comutativa, a equação anterior pode ser reescrita como $(x \vee y) \vee (m \vee m) = m$ e, pela propriedade de idempotência, tem-se $(x \vee y) \vee m = m$.

Assim, reescrevendo $(x \vee y) \wedge m$ como $(x \vee y) \wedge ((x \vee y) \vee m)$, ou seja, $(x \vee y) \wedge m = (x \vee y) \wedge ((x \vee y) \vee m)$, que pela propriedade de absorção pode ser reduzida a

$(x \vee y) \wedge m = (x \vee y)$. Devido à suposição 2.3 obtém-se $(x \vee y) \preceq m$.

Portanto, sendo $\mathbf{U}$ um subconjunto de $\mathbf{L}$, tal que $\mathbf{U} = \{x, y\}$, em que para $\forall\ m \in \mathbf{L} : x \preceq m$ e $y \preceq m \Rightarrow x \vee y \preceq m$ e, portanto, $x \vee y$ é o supremo de L.

Diretamente pela propriedade de absorção tem-se: $(x \wedge y) = (x \wedge y) \wedge (((x \wedge y) \vee y))$ e $(x \wedge y) = (x \wedge y) \wedge (((x \wedge y) \vee x))$, que pela suposição 2.3 e novamente absorção obtêm-se:

$$(x \wedge y) \preceq (x \wedge y) \vee y \Rightarrow x \wedge y \preceq y \text{ e}$$
$$(x \wedge y) \preceq (x \wedge y) \vee x \Rightarrow x \wedge y \preceq x.$$

Assim, sendo $\mathbf{U}$ um subconjunto de $\mathbf{L}$, tal que $\mathbf{U} = \{x, y\}$, então $x \wedge y$ é minorante.

Por outro lado, se $u \preceq x$ e $u \preceq y$, pela suposição 2.3 tem-se, respectivamente, que $u = u \wedge x$ e $u = u \wedge y$. Portanto, utilizando a propriedade idempotência $u \wedge u = u$ e pela substituição das equações anteriores tem-se $(u \wedge x) \wedge (u \wedge y) = u$. Por meio das propriedades associativa e comutativa obtém-se $(x \wedge y) \wedge (u \wedge u) = u \Rightarrow (x \wedge y) \wedge u = u$.

Diretamente pela suposição 2.3 tem-se que $u = u \wedge (x \wedge y) \Rightarrow u \preceq (x \wedge y)$. Assim, sendo $\mathbf{U}$ um subconjunto de $\mathbf{L}$, tal que $\mathbf{U} = \{x, y\}$, em que para $\forall\ u \in \mathbf{L} : u \preceq x$ e $u \preceq y \Rightarrow u \preceq x \wedge y$ e, portanto, $x \wedge y$ é o ínfimo de L.

Portanto, como $\mathbf{L}$ é um conjunto parcialmente ordenado que para quaisquer dois elementos x e y há um supremo e um ínfimo, então como se queria demonstrar, $\mathbf{L}$ é um reticulado segundo a definição dada na Seção 2.1.1.1. ■

Se um reticulado L contém um elemento mínimo ϕ, então $\phi \vee x = x$ e $\phi \wedge x = \phi$ para $\forall\ x \in \mathbf{L}$, ou seja, ϕ é um elemento identidade ou neutro para $\vee$ e nulo ou absorvitivo para $\wedge$.

Similarmente, se L contém um elemento máximo θ, então $\theta \vee x = \theta$ e $\theta \wedge x = x$ para $\forall\ x \in \mathbf{L}$, ou seja, θ é um elemento nulo ou absorvitivo para $\vee$ e identidade ou neutro para $\wedge$.

2.3.1 Álgebra de Caminhos

Em 1979, com o objetivo de resolver e analisar alguns problemas que envolvem dígrafos e busca de caminhos em dígrafos, Carré, em [34], propôs uma estrutura matemática $\langle \mathbf{P}, \vee, \bullet \rangle$ denominada álgebra de caminhos. Baseando-se neste trabalho, aproximadamente 20 anos depois, Sobrinho, em [35], e Gouda

e Schneider, em [23], voltaram a visitar esse tema com o objetivo de formalizar matematicamente as métricas de roteamento e analisar as propriedades dos algoritmo de roteamento que garantem a sua convergência.

2.3.1.1 Proposta de Bernard Carré

Segundo [34], a álgebra de caminhos é definida como um conjunto $\mathbf{P}$, equipado com duas operações binárias $\vee$ e $\bullet$, denominadas, respectivamente, de união e produto, que obedecem as Propriedades 2.18, 2.19, 2.20 e 2.21, descritas a seguir.

Propriedade 2.18. A operação $\vee$ é idempotente, comutativa e associativa.

Propriedade 2.19. A operação $\bullet$ é associativa e distributiva em $\vee$:

$$x \bullet (y \vee z) = (x \bullet y) \vee (x \bullet z) \text{ para } \forall\ x, y, z \in \mathbf{P} \text{ e}$$
$$(y \vee z) \bullet x = (y \bullet x) \vee (z \bullet x) \text{ para } \forall\ x, y, z \in \mathbf{P}.$$

Propriedade 2.20. O conjunto $\mathbf{P}$ contém um elemento ϕ,genericamente denominado elemento zero, que é identidade ou neutro para $\vee$ e nulo ou absorvitivo para $\bullet$.

$$\phi \vee x = x \text{ para } \forall\ x \in \mathbf{P} \text{ e}$$
$$\phi \bullet x = \phi = x \bullet \phi \text{ para } \forall\ x \in \mathbf{P}.$$

Propriedade 2.21. O conjunto $\mathbf{P}$ contém um elemento e, genericamente denominado elemento unidade, que é identidade ou neutro para $\bullet$.

$$e \bullet x = x = x \bullet e \text{ para } \forall\ x \in \mathbf{P}.$$

Como a operação $\vee$ é idempotente, comutativa e associativa o conjunto $\mathbf{P}$ pode ser ordenado conforme a regra:

$$x \preceq y \Leftrightarrow x \vee y = y.$$

Dessa forma, pode-se concluir que:

$$\phi \preceq x \text{ para } \forall\ x \in \mathbf{P}.$$

Como a operação $\vee$ é idempotente, tem-se pela definição dessa propriedade que:

$$x \preceq x \vee y \text{ e } y \preceq x \vee y \text{ para } \forall\ x, y \in \mathbf{P}.$$

Note que $\vee$ e $\bullet$ são monotônicas para a ordenação $\preceq$, ou seja,

$$\text{se } x \preceq y \text{ então } x \vee z \preceq y \vee z \text{ para } \forall\ z \in \mathbf{P} \text{ e}$$
$$\text{se } x \preceq y \text{ então } x \bullet z \preceq y \bullet z \text{ para } \forall\ z \in \mathbf{P}.$$

Com essa álgebra pode-se modelar, entre outros, os seguintes problemas:

P1. Lista de todos os caminhos:

Sejam:

1. $\sum$ qualquer alfabeto;
2. $\sum^*$ o conjunto de todas as palavras em $\sum$; e
3. $\mathbf{P}(\sum^*)$ o conjunto de partes também denominado conjunto potência, ou seja, o conjunto de todos os subconjuntos de $\sum^*$, em que cada elemento de $\mathbf{P}(\sum^*)$, ou cada subconjunto de $\sum^*$ é denominado linguagem de $\sum$;

para duas linguagens $\mathbf{X}, \mathbf{Y} \in \mathbf{P}(\sum^*)$, definem-se as operações:

1. $\mathbf{X} \vee \mathbf{Y} = \mathbf{X} \cup \mathbf{Y}$ e
2. $\mathbf{X} \bullet \mathbf{Y} = \{\chi \circ \psi : \chi \in \mathbf{X} \text{ e } \psi \in \mathbf{Y}\}$, em que $\chi \circ \psi$ representa a concatenação da palavra χ e ψ. Exemplo: $\mathbf{X} = \{\lambda, a, ba\}$ e $\mathbf{Y} = \{aa, b\}$ então $\chi \circ \psi = \{aa, b, aaa, ab, baaa, bab\}$, em que:
   (a) $\mathbf{\Lambda} = \{\lambda\}$ em que λ é uma palavra vazia e representa o elemento unidade, e
   (b) $\emptyset = \{\}$ representa o elemento zero.

P2. Lista de todos os caminhos simples:

Sejam:

1. $\sum$ qualquer alfabeto;
2. w uma palavra simples em $\sum$, ou seja, nenhuma letra aparece mais de uma vez;
3. $\mathbf{S}$ o conjunto de todas as palavras simples em $\sum$; e
4. $\mathbf{P}(\mathbf{S})$ o conjunto potência de $\mathbf{S}$;

para duas linguagens $\mathbf{X}, \mathbf{Y} \in \mathbf{P}(\mathbf{S})$, definem-se as operações:

1. $\mathbf{X} \vee \mathbf{Y} = \mathbf{X} \cup \mathbf{Y}$ e
2. $\mathbf{X} \bullet \mathbf{Y} = \{\chi \circ \psi \in \mathbf{S} : \chi \in \mathbf{X} \text{ e } \psi \in \mathbf{Y}\}$, em que:
   (a) $\mathbf{\Lambda} = \{\lambda\}$ em que λ é uma palavra vazia e representa o elemento unidade, e
   (b) $\emptyset = \{\}$ representa o elemento zero.

P3. Lista de todos os caminhos elementares:

Sejam:

1. $\sum$ qualquer alfabeto;
2. w uma palavra simples em $\sum$;
3. $b(w)$ a abreviação da palavra w obtida pela remoção de pelo menos uma das letras, ou até todas as letras de w. Observe que qualquer palavra com pelo menos uma letra tem abreviação dada por λ;
4. Para qualquer linguagem $\mathbf{X}$ de $\sum$, define-se uma palavra $w \in \mathbf{X}$ como básica se $\mathbf{X}$ não contém nenhuma abreviação de w; e
5. Uma linguagem $\mathbf{B}$ é básica quando $\mathbf{B}$ é o conjunto de todas as palavras básicas de uma linguagem. Observe que as linguagens $\emptyset$ e $\mathbf{\Lambda}$ são básicas;

para duas linguagens $\mathbf{X}, \mathbf{Y} \in \mathbf{B}$, definem-se as operações:

1. $\mathbf{X} \vee \mathbf{Y} = b(\mathbf{X} \cup \mathbf{Y})$ e
2. $\mathbf{X} \bullet \mathbf{Y} = \{\chi \circ \psi : \chi \in \mathbf{X} \text{ e } \psi \in \mathbf{Y}\}$, em que:
   (a) $\mathbf{\Lambda} = \{\lambda\}$ em que λ é uma palavra vazia e representa o elemento unidade, e
   (b) $\emptyset = \{\}$ representa o elemento zero.

P4. Determinação do caminho mais curto:

Sejam:

1. $\mathbf{P}$ o conjunto de todos os pesos w das arestas $\mathbf{E}$ de um dígrafo ponderado $D = (\mathbf{V}, \mathbf{E})$, com função peso tal que $E \rightarrow \Re$; e
2. A distância entre dois vértices x e y quaisquer é dada por $d(p(x, y)) = \sum_{i=1}^{n-1} w(v_i, v_{i+1})$ em que $x = v_1$ e $y = v_n$;

para o conjunto $\mathbf{P}$, definem-se as operações:

1. $\mathbf{X} \vee \mathbf{Y} = min(d(p(x,y)))$ e
2. $\mathbf{X} \bullet \mathbf{Y} = x + y$, em que:
   (a) 0 representa o elemento unidade e
   (b) ∞ representa o elemento zero.

A Tabela 2.2 apresenta um resumo dos problemas modelados pela álgebra de caminhos proposta por Carré.

Tabela 2.2: Exemplos de aplicação propostos por Carré

Aplicação	$\mathbf{P}$	$\mathbf{X} \vee \mathbf{Y}$	$\mathbf{X} \bullet \mathbf{Y}$	ϕ	$\mathbf{e}$
P1. Lista de todos os caminhos:	$\mathbf{P}(\sum^*)$	$\mathbf{X} \cup \mathbf{Y}$	$\{\chi \circ \psi : \chi \in \mathbf{X} \text{ e } \psi \in \mathbf{Y}\}$	$\emptyset$	$\mathbf{\Lambda}$
P2. Lista de todos os caminhos simples:	$\mathbf{P}(\mathbf{S})$	$\mathbf{X} \cup \mathbf{Y}$	$\{\chi \circ \psi \in \mathbf{S} : \chi \in \mathbf{X} \text{ e } \psi \in \mathbf{Y}\}$	$\emptyset$	$\mathbf{\Lambda}$
P3. Lista de todos os caminhos elementares:	$\mathbf{B}$	$b(\mathbf{X} \cup \mathbf{Y})$	$\{\chi \circ \psi : \chi \in \mathbf{X} \text{ e } \psi \in \mathbf{Y}\}$	$\emptyset$	$\mathbf{\Lambda}$
P4. Determinação do caminho mais curto:	$\Re \cup \{\infty\}$	$min(d(p(x,y)))$	$x + y$	∞	0

Dada uma álgebra de caminhos $\langle \mathbf{P}, \vee, \bullet \rangle$ e $\mathbf{M_n}(\mathbf{P})$ o conjunto de todas as $n \times n$ matrizes cujas entradas a_{ij} pertençam a P, define-se duas operações binárias em $\mathbf{M_n}(\mathbf{P})$: $\vee$ e $\bullet$, tal que dadas duas matrizes quaisquer $\overline{X} = [x_{ij}]$ e $\overline{Y} = [y_{ij}]$ em $\mathbf{M_n}(\mathbf{P})$:

$$\overline{X} \vee \overline{Y} = [x_{ij} \vee y_{ij}] \text{ e}$$
$$\overline{X} \bullet \overline{Y} = [x_{ij} \bullet y_{ij}].$$

Por meio das Propriedades 2.18, 2.19, 2.20 e 2.21, verifica-se que a estrutura $\langle \mathbf{M_n}(\mathbf{P}), \vee, \bullet \rangle$ com as operações $\vee$ e $\bullet$ também corresponde a uma álgebra de caminhos.

Nessa álgebra de caminhos, define-se a matriz $\overline{\Phi}$, cujos elementos são todos ϕ, ou seja, são todos elementos zero em P, como o elemento zero em $\langle \mathbf{M_n}(\mathbf{P}), \vee, \bullet \rangle$, pois $\overline{\Phi} \vee \overline{X} = \overline{X}$ e $\overline{\Phi} \bullet \overline{X} = \overline{\Phi} = \overline{X} \bullet \overline{\Phi}$ para $\forall\ \overline{X} \in \mathbf{M_n}(\mathbf{P})$ e a matriz

$$\overline{E} = \begin{bmatrix} e & \phi & \phi & \dots & \phi \\ \phi & e & \phi & \dots & \phi \\ \phi & \phi & e & \dots & \phi \\ \dots & \dots & \dots & \dots & \dots \\ \phi & \phi & \phi & \dots & e \end{bmatrix}$$

como o elemento unidade em $\langle \mathbf{M_n}(\mathbf{P}), \vee, \bullet \rangle$, pois $\overline{E} \bullet \overline{X} = \overline{X} = \overline{X} \bullet \overline{E}$ para $\forall\ \overline{X} \in \mathbf{M_n}(\mathbf{P})$.

Como $\langle \mathbf{M_n}(\mathbf{P}), \vee, \bullet \rangle$ é uma álgebra de caminhos, então $\forall\ \overline{X} \in \mathbf{M_n}(\mathbf{P})$, por similaridade, obtém-se:

$$\overline{X} \preceq \overline{Y} \Leftrightarrow \overline{X} \vee \overline{Y} = \overline{Y},$$

em que $\overline{X} \preceq \overline{Y}$ se e somente se $x_{ij} \preceq y_{ij}$ para $\forall\ i, j$.

Assim, dada uma matriz $\overline{A}$ qualquer $\in\ \mathbf{M_n}(\mathbf{P})$, define-se as potências de $\overline{A}$ como:

$$\overline{A}^0 = E \qquad \text{e} \qquad \overline{A}^k = \overline{A}^{k-1} \bullet \overline{A}.$$

Para um dígrafo ponderado D, com matriz de adjacências $\overline{A}$ de ordem n dada na Seção 2.1.2 e um elemento qualquer representado por a_{im}, o elemento a_{ij}^k da matriz $\overline{A}^k$ é definido como:

$$a_{ij}^k = \bigvee_{m=1}^{n} (a_{im}^{k-1} \bullet a_{mj}) \qquad (k = 0, 1, 2, ...). \tag{2.5}$$

Definindo-se a distância de um caminho $p(v_1, v_n) = v_1, v_2, v_3, \ldots, v_n$, como $d(p(v_1, v_n)) = a(v_1, v_2) \bullet a(v_2, v_3) \bullet \cdots \bullet a(v_{i-1}, v_i) \bullet a(v_i, v_{i+1})$, então o elemento a_{ij}^k da matriz $\overline{A}^k$ pode ser reescrito como:

$$a_{ij}^k = \bigvee_{p \in S_{ij}^k} d(p) \qquad (k = 0, 1, 2, ...),$$

em que S_{ij}^k é o conjunto de todos os caminhos de ordem k do vértice v_i ao v_j de D.

Utilizando-se a álgebra de caminhos proposta em P4, ou seja, substituindo-se na equação 2.5 as operações $(a_{im}^{k-1} \bullet a_{mj})$ por $(a_{im}^{k-1} + a_{mj})$ e $(a_{im}^{k-2} + a_{mj}) \vee (a_{im}^{k-1} + a_{mj})$ por $min((a_{im}^{k-2} + a_{mj}), (a_{im}^{k-1} + a_{mj}))$, é fácil observar que o elemento a_{ij}^k corresponde à distância de um caminho mais curto de ordem k do vértice v_i ao v_j.

Esse resultado é utilizado em algoritmos de programação dinâmica para busca do caminho mais curto entre todos os pares com complexidade $\Theta(V^4)$ [41].

2.3.1.2 Proposta de João Luís Sobrinho

Em 2002, com o objetivo de analisar algoritmos de roteamento pelo destino que calculam a qualidade de serviço de um caminho na Internet, Sobrinho em [35] propôs uma nova estrutura matemática $\langle \mathbf{W}, \preceq, \mathbf{L}, \sum, \phi, \oplus, f \rangle$ para a álgebra de caminhos, que foi expandida em [24, 36] e reutilizada em [74].

Segundo essa estrutura, define-se:

- $\mathbf{W}$ como um conjunto de pesos totalmente ordenado pela ordenação $\preceq$,
- $\mathbf{L}$ como um conjunto de rótulos,
- $\sum$ com um conjunto de assinaturas, com uma assinatura especial ϕ,
- uma ordenação total reflexiva $\preceq$ em $\mathbf{W}$,
- uma operação $\oplus$ associativa em $\mathbf{L}$ e
- uma função f que mapeia as assinaturas em pesos, ou seja, $\sum \rightarrow \mathbf{W}$,

em que são obedecidas as seguintes propriedades:

- maximização: $\forall\ \alpha \in \{\sum - \{\phi\}\} \Rightarrow f(\alpha) \prec f(\phi)$, ou seja, o elemento ϕ é máximo em $\sum$;
- absorção: $\forall\ l \in \mathbf{L} \Rightarrow l \oplus \phi = \phi$;
- monotonicidade: $\forall\ l \in \mathbf{L}$ e $\forall\ \alpha \in \sum \Rightarrow f(\alpha) \preceq f(l \oplus \alpha)$;
- monotonicidade estrita: $\forall\ l \in L$ e $\forall\ \alpha \in \sum \Rightarrow f(\alpha) \prec f(l \oplus \alpha)$; e
- isotonicidade: $\forall\ l \in \mathbf{L}$ e $\forall\ \alpha, \beta \in \sum$ então $f(\alpha) \preceq f(\beta) \Rightarrow f(l \oplus \alpha) \preceq f(l \oplus \beta)$.

Assim, em uma rede de telecomunicações representada por um dígrafo ponderado D, em que cada aresta está associada a um rótulo $l \in \mathbf{L}$ e cada caminho a uma assinatura $s \in \sum$, a assinatura de um caminho trivial (u, u) é representada por $s((u, u))$ e a assinatura de um caminho não trivial $(u, v) \circ Q$ é representada por $s(uv \circ Q) = l(u, v) \oplus s(Q)$, tal que $(u, v) \circ Q$ representa o caminho resultante da concatenação da aresta (u, v) com o subcaminho Q.

Segundo essa álgebra de caminhos, o peso de um caminho p qualquer é dado por $f(s(p)) \in \mathbf{W}$, em que $\mathbf{W}$ é um conjunto totalmente ordenado pela ordenação

$\preceq$ e, portanto, devido à propriedade de monotonicidade tem-se $f(s(u,v)) \preceq f(s(uv \circ Q))$.

Nesse arcabouço matemático, define-se o caminho ótimo δ entre dois vértices distintos u e v, como o caminho com menor peso entre todos os demais caminhos possíveis entre u e v, ou seja, $f(s(\delta)) \preceq f(s(d))$ para $\forall$ d entre u e v $\in D$.

A Tabela 2.3 apresenta um resumo dos problemas modelados pela álgebra de caminhos proposta por Sobrinho.

Tabela 2.3: Exemplos de aplicação propostos por Sobrinho

Problema	**W**	$\oplus$	$\emptyset$	**Relação de Ordenação**$\alpha \preceq \beta$
Caminho Mais Curto	$\Re_0^+ \cup +\infty$	$+$	$+\infty$	$\alpha \geq \beta$
Caminho Mais Largo	$\Re_0^+ \cup +\infty$	min	0	$\alpha \leq \beta$
Caminho Mais Confiável	$[0,1]$	$\times$	1	$\alpha \leq \beta$
Caminho Mais Curto e Mais Largo	$\{(d,b) : d \in \Re_0^+, b \in \Re_0^+ \cup \{+\infty\}\} \cup \{\emptyset\}$	$(d+d', min(b,b'))$	$\emptyset$	$(d_\alpha > d_\beta$ ou $((d_\alpha = d_\beta)$ e $(b_\alpha \leq b_\beta))$

Segundo Sobrinho, em [36], um algoritmo de roteamento converge para qualquer rede sempre que a propriedade de monotonicidade é obedecida e converge para um caminho ótimo sempre que a propriedade de isotonicidade é obedecida. Em [24], Sobrinho indica que a propriedade de monotonicidade é uma condição necessária para os protocolos de roteamento DV convergirem em qualquer rede e que a propriedade de isotonicidade é uma condição necessária para eles convergirem para um caminho ótimo. A discussão desses resultados será feita na Seção 3.3 desta tese, visto que os algoritmos de roteamento utilizados nos protocolos de roteamento DV são do tipo *hop-by-hop*.

Buscando endereçar o problema da convergência dos algoritmos de roteamento, em [36], Sobrinho define o conceito de ciclos livres, traduzido do inglês *freeness*, como uma generalização do conceito de ciclos de distância positiva, em que um ciclo em uma rede é dito livre se ao menos um de seus nós encaminha a informação para um nó vizinho fora do ciclo, evitando-se com isso que a informação fique circulando em *loop* dentro do ciclo, ou seja, um ciclo $u_n, u_{n-1}, \ldots, u_1, u_0$, com $u_n = u_0$ é livre se para cada assinatura $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}, \alpha_n \in \sum$, existe um índice $i : 1 \leq i \leq n$ tal que $f(\alpha_i) \prec f(S(u_i, u_{i-1}, \alpha_i))$. Assim, segundo Sobrinho, um algoritmo de roteamento converge sempre que uma rede for composta somente por ciclos livres. A discussão desses resultados será feita na Seção 3.3 desta tese.

2.3.2 Espaço Métrico

Em matemática, um espaço métrico $\langle \mathbf{X}, d \rangle$ é uma estrutura matemática formada por um conjunto X equipado com uma métrica d, ou seja, uma função $d : \mathbf{X} \times \mathbf{X} \rightarrow \Re_0^+$ para quaisquer $x, y, z \in \mathbf{X}$, em que [75, 76]:

- $d(x, y)$ é um número real não negativo e finito;
- $d(x, y) = 0 \Leftrightarrow x = y$;
- $d(x, y) = d(y, x)$, ou seja, a propriedade de simetria é respeitada; e
- $d(x, z) \leq d(x, y) + d(y, z)$, ou seja, a propriedade da desigualdade triangular é respeitada.

Observe que um dígrafo D não orientado, portanto simétrico, pode ser convertido em um espaço métrico pela definição da função $d(x, y)$, que é a distância do caminho entre os vértices x e y.

2.3.2.1 Proposta de Gouda e Schneider

Em redes de telecomunicações, denomina-se métrica a característica, de um caminho ou de uma rota, utilizada por um algoritmo de roteamento para determinar se um caminho é melhor ou não que outro [67, 77]. As métricas mais comumente utilizadas são: número de saltos (*hops*), atraso, variação do atraso, disponibilidade, largura de banda de transmissão, volume de tráfico, tamanho máximo do pacote a ser transmitido, custo, etc [78].

Explorando esse conceito sob o ponto de vista de um espaço métrico, Gouda e Schneider, em [23], propõem uma definição formal para a métrica de roteamento com três exemplos: distância, fluxo e disponibilidade. Da mesma forma, buscando abordar o problema do roteamento com múltiplas restrições, os autores propõem duas composições binárias de métricas: aditiva e léxica.

Uma métrica é uma 5-upla $\langle \mathbf{M}, \mathbf{W}, met, mr, \prec \rangle$, em que:

1. $\mathbf{M}$ é um conjunto dos valores das métricas;
2. $\mathbf{W}$ é um conjunto dos pesos das arestas;
3. met é uma função $met : \mathbf{M} \times \mathbf{W} \rightarrow \mathbf{M}$;
4. mr é o valor máximo de uma métrica em M com respeito à ordenação $\prec$ e está associado à raiz de uma árvore de máxima métrica; e

5. $\prec$ é uma relação de ordenação "menor que", aplicada ao conjunto $\mathbf{M}$, que satisfaz as seguintes propriedades:

   (a) irreflexividade: $m \nprec m$;

   (b) transitividade: se $m \prec m^{'}$ e $m^{'} \prec m^{''}$, então $m \prec m^{''}$; e

   (c) assimetria[18]: se $m \prec m^{'}$ ou $m^{'} \prec m$, então $m = m^{'}$.

Para $\forall\ m \in \mathbf{M}$, em que $m \neq mr$, há uma seqüência de métricas $m_0, m_1, \ldots, m_k$ ($m_i \in \mathbf{M}$ e $i = 1, 2, \ldots, k$), correspondente à seqüência não vazia de arestas de pesos $w_0, w_1, \ldots, w_k$ ($w_i \in \mathbf{W}$ e $i = 1, 2, \ldots, k$), dada por:

$$
\begin{aligned}
m_0 &= mr \\
m_1 &= met(m_0, w_0) \\
&\vdots \\
m_k &= met(m_{k-1}, w_{k-1}) \\
m &= m_k
\end{aligned}
$$

Métrica de Distância $\langle \mathbf{M}, \mathbf{W}, +, mr, > \rangle$, em que:

1. $\mathbf{M}$ é um conjunto finito de inteiros não negativos, em que figuram os valores da métrica de distância;

2. $\mathbf{W}$ é um conjunto finito de inteiros não negativos composto pelos valores dos custos da rede, associados às arestas do dígrafo que a representa;

3. $+$ é a função que retorna a soma de dois inteiros não negativos;

4. $mr = 0$ é a menor distância na rede com relação a mr, ou seja, é a distância de mr para mr; e

5. $>$ é a relação "maior que" sobre inteiros não negativos.

Métrica de Fluxo $\langle \mathbf{M}, \mathbf{W}, min, mr, < \rangle$, em que:

1. $\mathbf{M}$ é um conjunto finito de inteiros não negativos, em que figuram os valores da métrica de fluxo;

2. $\mathbf{W}$ é um conjunto finito de inteiros não negativos composto pelos valores das capacidades da rede, associados às arestas do dígrafo que a representa;

[18] Os autores denominam essa propriedade de totalidade. Todavia utilizou-se a denominação correta para não confundir o leitor.

3. min é a função que retorna o mínimo entre dois inteiros não negativos;

4. mr é a capacidade máxima de um enlace/aresta em $\mathbf{W}$; e

5. $<$ é a relação "menor que" sobre inteiros não negativos.

Métrica de Disponibilidade$\langle \mathbf{M}, \mathbf{W}, \times, mr, < \rangle$, em que:

1. $\mathbf{M}$ é um conjunto de todos os números reais $p : 0 < p \leq 1$;

2. $\mathbf{W}$ é um conjunto de todos os números reais $p : 0 < p \leq 1$;

3. $\times$ é a função que retorna a multiplicação de dois números reais;

4. $mr = 1$ é a disponibilidade máxima de um enlace/aresta em $\mathbf{W}$; e

5. $<$ é a relação "menor que" sobre inteiros não negativos.

Sobre essa estrutura matemática, são definidas três propriedades, listadas a seguir.

- **Isotonicidade**[19]: $m \prec m' \Rightarrow met(m, w) \prec met(m', w)$ ou $(met(m, w) = met(m', w)$.
- **Isotonicidade Estrita**[20]: $m \prec m' \Rightarrow met(m, w) \prec met(m', w)$.
- **Liberdade**[21]: $met(m, w) \prec m$ ou $met(m, w) = m$.

Por outro lado, sendo dadas duas métricas quaisquer $\langle \mathbf{M}_1, \mathbf{W}_1, met_1, mr_1, \prec_1 \rangle$ e $\langle \mathbf{M}_2, \mathbf{W}_2, met_2, mr_2, \prec_2 \rangle$, elas podem ser combinadas de duas maneiras:

Composição léxica: em que a relação $\prec$ sobre o conjunto $\mathbf{M}_1 \times \mathbf{M}_2$ é uma relação léxica entre os conjuntos $\mathbf{M}_1 \times \mathbf{M}_2$, ou seja, $(m_1, m_2) \prec (m_1', m_2') \Leftrightarrow m_1 \prec_1 m_1'$ ou $(m_1 =_1 m_1'$ e $m_2 \prec_2 m_2')$.

Composição aditiva: em que a relação $\prec$ sobre o conjunto $\mathbf{M}_1 \times \mathbf{M}_2$ é a relação "menor que" $<$ sobre os números inteiros e a função utilizada é a soma, ou seja, $(m_1, m_2) \prec (m_1', m_2') \Leftrightarrow (m_1 + m_2) < (m_1' + m_2')$.

[19] Os autores denominam essa propriedade de monotonicidade. Todavia utilizou-se a denominação correta para não confundir o leitor.

[20] Os autores denominam essa propriedade de monotonicidade estrita. Todavia utilizou-se a denominação correta para não confundir o leitor.

[21] Os autores denominam essa propriedade de limite (tradução do inglês *boundedness*). Todavia utilizou-se a denominação dada por Sobrinho e utilizada neste trabalho para facilitar a compreensão do leitor da aplicação dessa propriedade.

Observe que segundo essa composição, para os todos os possíveis caminhos entre dois vértices, as duas métricas diferentes são somadas no final de cada caminho antes de serem ordenadas, ou em outras palavras, antes de todos os possíveis caminho serem ordenados.

As Tabelas 2.4 e 2.5 apresentam, respectivamente, o resumo das métricas e das combinações de métricas propostas por Gouda e Schneider.

Tabela 2.4: Métricas propostas por Gouda e Schneider

Tipos de Métrica	**M**	**W**	**met**	**mr**	**Relação de Ordenação** $\prec$
Fluxo	$m \in \Re_0^+$	$\mathbf{W} \subseteq \mathbf{M}$	min	$\lceil \mathbf{W} \rceil$	$m \prec m' \Rightarrow m < m'$
Distância	$m \in \Re_0^+$	$\mathbf{W} \subseteq \mathbf{M}$	$+$	$\lfloor \mathbf{W} \rfloor = 0$	$m \prec m' \Rightarrow m > m'$
Disponibilidade	$m \in \Re / 0 \leq m \leq 1$	$\mathbf{W} \subseteq \mathbf{M}$	$\times$	$\lceil \mathbf{W} \rceil = 1$	$m \prec m' \Rightarrow m < m'$

Tabela 2.5: Composição de métricas propostas por Gouda e Schneider

Composição de Métricas	**M**	**W**	**met**	**mr**	**Relação de Ordenação** $\prec$
Aditiva	$\mathbf{M}_1 \times \mathbf{M}_2$	$\mathbf{W}_1 \times \mathbf{W}_2$	$(met_1 + met_2)$	$(mr_1 + mr_2)$	$(m_1, m_2) \prec (m_1', m_2') \Leftrightarrow$ $(m_1 + m_2) \prec (m_1' + m_2')$ ou $(m_1, m_2) \equiv (m_1', m_2') \Leftrightarrow$ $(m_1 + m_2) \equiv (m_1' + m_2')$
Léxica	$\mathbf{M}_1 \times \mathbf{M}_2$	$\mathbf{W}_1 \times \mathbf{W}_2$	(met_1, met_2)	(mr_1, mr_2)	$(m_1, m_2) \prec (m_1', m_2') \Leftrightarrow$ $(m_1 \prec_1 m_1') \vee$ $\big((m_1 = m_1') \wedge (m_2 \prec_2 m_2')\big)$

Em [23], os autores também propõem uma 6-upla $\langle \mathbf{M}, \mathbf{W}, met, mr, \prec, wf \rangle$ aplicada a uma rede N, em que $\langle \mathbf{M}, \mathbf{W}, met, mr, \prec \rangle$ é a métrica de roteamento definida por eles e wf é uma função que associa cada aresta $\{u, v\}$ em N a um peso $wf(\{u, v\})$ em W. Com essa álgebra de caminhos, eles analisam a propriedade de isotonicidade, para os três tipos propostos de métricas, bem como a preservação ou não dessas propriedades com o uso da combinação aditiva ou léxica.

Segundo indicado em [23], a propriedade de isotonicidade é condição necessária e suficiente para que seja possível construir uma árvore geradora para qualquer nó da rede, em que cada caminho é um caminho de máxima métrica em N, ou seja, cada caminho é um caminho cujo valor resultante da métrica associada a ele é maior do que os valores resultantes obtidos para os demais caminhos encontrados para o mesmo par de vértices origem e destino em N. Os autores também indicam que a propriedade de liberdade é necessária e suficiente para

garantir a construção de uma árvore geradora em N, quando N é um dígrafo com ciclos.

3 Desenvolvimento do Trabalho

Buscando abordar o problema da análise de convergência dos algoritmos de roteamento *hop-by-hop* mono/ multi-restritivo e analisando as propostas das álgebras de caminhos apresentadas na literatura e resumidamente descritas no Capítulo 2, verificou-se a necessidade de propor uma nova álgebra de caminhos capaz de:

- integrar, em uma única estrutura matemática, os diferentes conceitos apresentados;
- flexibilizar a incorporação de novas métricas, novas formas de computar os valores resultantes dessas métricas em um caminho e novas formas de comparar esses caminhos mantendo-se os mesmos critérios de análise; e
- estender a capacidade de modelagem de problemas de roteamento mono/ multi-restritivo para a incorporação da análise do problema da busca de caminhos, que utiliza, entre outros parâmetros, características ou restrições que se compensam, enlace-a-enlace, ao longo de um caminho. Por exemplo, a modelagem do problema da busca do melhor caminho em relação a variação do atraso, que pode ser medida em função da variação dinâmica da carga na rede ou em função de um atraso referencial de uma aplicação, quando é abordado o problema da restauração[1]; a modelagem do problema da busca do melhor caminho em função da relação atenuação/ ganho em uma rede DWDM com configuração automática de lambdas e de ajuste de potência, em que o ajuste das potências é limitado pelo intervalo de operação dos transmissores/ receptores da rede[79, 68, 80]; e outros problemas similares. Observe que, nesses casos, os parâmetros de roteamento, definidos na literatura como métricas de roteamento e neste trabalho, para simplificar a nomenclatura, muitas vezes indicados simplesmente

[1]Este cenário é melhor detalhado na Seção 4.2.

como métricas, difere do conceito matemático dado na definição do espaço métrico, podendo assumir tanto valores positivos como valores negativos.

Sendo assim, ao analisar a convergência dos algoritmos de roteamento *hop-by-hop* mono/ multi-restritivo, verificou-se também a necessidade de rever as conclusões encontradas na literatura para a garantia de convergência dos algoritmos de roteamento *hop-by-hop* mono/ multi-restritivo.

Portanto, este capítulo: descreve a nova álgebra de caminhos proposta; reapresenta as propriedades conhecidas de monotonicidade, monotonicidade estrita, isotonicidade, isotonicidade estrita e liberdade; discute os resultados obtidos na análise da convergência de algoritmos de roteamento *hop-by-hop* em comparação com os resultados apresentados na literatura para a garantia de convergência; apresenta uma nova propriedade denominada coerência, demonstrando sua condição de necessidade e suficiência na garantia da convergência de algoritmos de roteamento *hop-by-hop*; propõe uma nova definição para o melhor caminho em uma rede, que implementa um algoritmo de roteamento *hop-by-hop*, de maneira a garantir a observância da propriedade de coerência; e propõe e descreve o algoritmo de simulação ELND, utilizado para analisar os resultados que se obtêm em uma rede que adota um algoritmo de roteamento mono/ multi-restritivo *hop-by-hop* e que, segundo a definição da álgebra de caminhos proposta, pela imposição da propriedade de coerência tem garantida a sua convergência.

3.1 Álgebra de Caminhos Proposta

Modelando-se uma rede de telecomunicações como um dígrafo orientado e ponderado $N = (\mathbf{V}, \mathbf{E})$, composto por um conjunto $\mathbf{V}$ de vértices (v_i), que representam os nós ou equipamentos da rede a ser analisada, e um conjunto $\mathbf{E}$ de arestas (v_i, v_j), que representam as conexões física ou lógica entre esses nós e possuem atributos ou características que as qualificam, a partir das idéias descritas em [34, 23, 35, 24, 36], este trabalho propõe uma nova álgebra de caminhos definida pela seguinte estrutura matemática $\langle \mathbf{M}, \mathbf{F}, \overline{\mathbf{S}}, \preceq_{ML} \rangle$, em que:

- $\mathbf{M}$ é um conjunto de métricas de roteamento;
- $\mathbf{F}$ é um conjunto de funções de combinação de métricas;
- $\overline{\mathbf{S}}$ é um conjunto de operações binárias denominadas sínteses; e
- $\preceq_{ML}$ é uma relação de ordenação léxica multi-dimensional.

3.1.1 Métrica de Roteamento

Adota-se como métrica de roteamento qualquer atributo ou característica que, segundo uma estratégia de otimização, se analisa na escolha do melhor caminho em uma rede. Assim, este trabalho define uma métrica de roteamento como a 2-upla $\langle \mathbf{Q_j}, \mathbf{w}^{\mathbf{Q_j}} \rangle$, em que:

- $\mathbf{Q_j}$ é o conjunto de valores que cada métrica pode assumir, sem restrição quanto ao seu domínio; e
- $\mathbf{w}^{\mathbf{Q_j}}$ é o conjunto dos valores da métrica $\mathbf{Q_j}$ associados às arestas do dígrafo que representa a rede, ou seja, o conjunto dos pesos das arestas do dígrafo associados à métrica $\mathbf{Q_j}$.

3.1.2 Função de Combinação de Métricas

Em função da estratégia de otimização e da natureza das métricas adotadas, um algoritmo de roteamento pode combinar diferentes métricas de roteamento entre si e mapear para cada enlace ou aresta do dígrafo $N = (\mathbf{V}, \mathbf{E})$ os novos valores obtidos dessa combinação. A partir de então, esses novos valores são utilizados pelo algoritmo de roteamento como novos pesos para computar, no final de um caminho, o valor resultante ou sintetizado que o qualificará com o objetivo de eleger, entre os vários caminhos existentes entre dois nós da rede, o melhor de todos.

A função de combinação é comumente aplicada quando se utiliza algoritmos de roteamento consagrados, como o algoritmo de Dijkstra, com métricas que pela sua natureza original não poderiam ser utilizadas, como por exemplo, a variação em tempo real do atraso. Por poder assumir tanto valores positivos quanto negativos, a variação em tempo real do atraso não poderia ser utilizada pelo algoritmo de Dijkstra na busca do caminho com menor variação de atraso. Entretanto, algumas aplicações utilizam Dijkstra com a variância do atraso, ou seja, aplica-se a função de combinação "variância" à métrica "variação do atraso" e, sobre o valor resultante, aplica-se Dijkstra [81, 82]. O mesmo pode ocorrer com a métrica de disponibilidade, uma vez que a busca do caminho de maior disponibilidade pode ser feita pela busca do caminho de menor indisponibilidade, ou seja, aplica-se a função de combinação dada por $(1 - \textit{disponibilidade})$ para obter os novos valores de pesos associados a cada enlace e que serão computados ao longo de um caminho, por meio da soma das indisponibilidades[2]. A seleção

[2]Neste exemplo, considera-se como indisponibilidade a probabilidade de falha de um enlace.

do melhor caminho passa a ser feita, então, buscando o caminho que apresenta menor indisponibilidade.

Uma outra aplicação observada da função de combinação de métrica é no uso de algoritmos consagrados de roteamento mono-restritivo para a resolução do problema de roteamento multi-restritivo, em que um só parâmetro de otimização, dado pela função de combinação de métricas, é adotado para a busca do melhor caminho. Como exemplo, pode-se citar o algoritmo de roteamento utilizado pelo IGRP, que combina em um só valor as métricas: largura de banda, carga, atraso e disponibilidade, segundo a função $(K_1 \times b + (K_2 \times b)/(256 - l) + K_3 \times d) \times (K_5/(r + K_4))$, em que: b é o inverso da largura de banda do enlace; l é a carga que varia de 0 a 255, em que 255 equivale saturação; d é o atraso; r é a disponibilidade que varia de 0 a 255, em que 255 representa 100% de disponibilidade; e K_1, K_2, K_3, K_4 e K_5 são constantes [23].

Observe que algoritmos que não utilizam a função de mapeamento, também podem ser analisados por esta álgebra de caminhos, bastando para isto considerar a função de mapeamento como uma função identidade.

Buscando abordar o problema do roteamento com múltiplas-restrições, define-se, genericamente, um conjunto $\mathbf{F} = \{f_1, f_2, \ldots, f_k\}$, composto por k funções de combinações de métricas, em que k representa o número de restrições impostas. Assim, a aplicação das funções de combinação de métricas sobre os valores das métricas ou os pesos de uma aresta i qualquer define um vetor de métricas-combinadas $\overline{C}_i = [f_1(w_i^{Q_1}, w_i^{Q_2}, \ldots, w_i^{Q_m}); \ldots ; f_j(w_i^{Q_1}, w_i^{Q_2}, \ldots, w_i^{Q_m}); \ldots ; f_k(w_i^{Q_1}, w_i^{Q_2}, \ldots, w_i^{Q_m})]$, ou simplesmente, $\overline{C}_i = [c_{i,1}; c_{i,2}; \ldots, c_{i,j}; \ldots, c_{i,k}]$.

3.1.3 Sínteses Léxicas

A síntese $S_j[.]$ é um conjunto de operações binárias aplicadas sobre os valores das métricas-combinadas dos enlaces ao longo de um caminho de maneira a obter um valor resultante que caracterizará esse caminho em relação à restrição imposta para a métrica-combinada.

Se o algoritmo de roteamento analisado é mono-restritivo, para cada caminho obtém-se somente um valor resultante da síntese, denominado palavra-peso. No entanto, se o algoritmo de roteamento é multi-restritivo, com k restrições impostas pelos novos parâmetros obtidos pelas k funções de combinação de métricas utilizadas, em que $k \in N : k > 1$, então, para cada caminho obtêm-se k valores

resultantes, um para cada uma das k sínteses aplicadas ao longo desse caminho. O conjunto dos valores resultantes obtidos das k sínteses é denominado palavra-peso multi-dimensional.

Assim, uma síntese $S_j[.]$ qualquer computa ou sintetiza ao longo de um caminho a métrica-combinada $c_j = f_j(w^{Q_1}, w^{Q_2}, ..., w^{Q_m})$, que será analisada segundo uma estratégia de otimização imposta pela restrição j.

As sínteses propostas são denominadas léxicas, pois as operações binárias efetuadas buscam a obtenção de uma palavra-peso multi-dimensional $\overline{S}[\overline{C}(p_{1,z})] = b_{l_1}(S_1[c_1(p_{1,z})]); b_{l_2}(S_2[c_2(p_{1,z})]); \ldots; b_{l_j}(S_j[c_j(p_{1,z})]); \ldots; b_{l_k}(S_k[c_k(p_{1,z})])$ resultante em um caminho $p_{1,z}$. Neste contexto, uma letra $b_{l_j}(S_j[c_j(p_{1,z})])$ corresponde a uma palavra-peso $b_{l_j}(S_j[c_j(p_{1,z})]) = b_{l_j}(W_z; W_{z-1}; \ldots; W_{z-l_j+1})$, em que $b_{l_j}(S_j[c_j(p_{1,z})])$ representa a abreviação com l_j letras da palavra-peso $S_j[c_j(p_{1,z})]$, conforme o conceito de abreviação descrito na Seção 2.3.1.1, e cada letra W_i corresponde ao peso resultante de um sub-caminho $p_{1,i}$, obtido segundo a métrica-combinada c_j.

Note que a primeira letra W_z da palavra-peso $S_j[c_j(p_{1,z})]$, corresponde ao valor resultante da métrica-combinada c_j ao longo do caminho completo $p_{1,z}$; a segunda letra W_{z-1} corresponde ao valor resultante da métrica-combinada c_j ao longo do sub-caminho $p_{1,z-1}$, formado pela extração da última aresta do caminho completo; a terceira letra W_{z-2} corresponde ao valor resultante da métrica-combinada c_j ao longo do sub-caminho $p_{1,z-2}$ formado pela extração da última e da penúltima aresta do caminho completo; e, sucessivamente, até que a última letra W_1 da palavra-peso corresponde ao valor da métrica-combinada c_j, associado à primeira aresta do caminho, ou seja, $p_{1,2}$.

Portanto, definem-se:

- $\mathbf{C_j}$ como um conjunto composto por todos os valores que a métrica-combinada c_j pode assumir;
- $\mathbf{X_j}$ como um alfabeto, isto é, como um conjunto totalmente ordenado, segundo a relação de ordenação $\preceq_L$, de todas as palavras-pesos obtidas pela aplicação da síntese $S_j[.]$ sobre o conjunto $\mathbf{C_j}$;
- $\overline{S}_j[\overline{C}(p)]$, como uma palavra-peso multi-dimensional, ou seja, uma seqüência ou uma concatenação limitada de palavras-peso;
- $\mathbf{\Gamma_j}$ como um alfabeto de palavras-peso, isto é, como o conjunto totalmente ordenado, segundo a relação de ordenação $\preceq_{ML}$ de todas as palavras-peso

multi-dimensionais obtidas pela aplicação das sínteses $\overline{S}[.]$;

- $\wedge$ e $\vee$ como duas operações binárias, em que $\wedge$ é a operação binária de concatenação e $\vee$ é uma operação binária de adição, multiplicação, maximização ou minimização;

- $b_{l_j}(S_j[c_j(p)])$ como a abreviação de l_j letras da palavra-peso $S_j[c_j(p)]$, obtida pela remoção de uma ou mais letras da palavra-peso completa, e $b_{l_q}(\overline{S}[\overline{C}(p)])$ como a abreviação de l_q palavras-peso da palavra-peso multi-dimensional $\overline{S}[\overline{C}(p)]$, obtida pela remoção de uma ou mais palavras-peso da palavra-peso multi-dimensional, ou seja, a remoção de uma ou mais restrição de roteamento; e

- $b_\infty(S_j[c_j(p)])$ representa a palavra-peso $S_j[c_j(p)]$ não abreviada, ou seja, o número de letras de $b_\infty(S_j[c_j(p)])$ é igual à ordem do caminho sintetizado. Em outras palavras, $b_\infty(S_j[c_j(p)])$ tem tantas letras, quantas arestas tem o caminho que ela caracteriza.

Portanto, para um caminho $p_{1,z}$ com k restrições, uma palavra-peso multi-dimensional $\overline{S}[\overline{C}(p_{1,z})]$, em que $S_j[c_j(p_{1,z})]$ representa uma de suas palavras-peso e W_i uma letra dessa palavra, têm-se:

$$\overline{S}[\overline{C}(p_{1,z})] = \bigwedge_{j=1}^{k} S_j[c_j(p_{1,z})], \tag{3.1}$$

$$b_{l_q}(\overline{S}[\overline{C}(p_{1,z})]) = \bigwedge_{j=1}^{l_q} S_j[c_j(p_{1,z})] \qquad \text{com } l < k, \tag{3.2}$$

$$S_j[c_j(p_{1,z})] = \bigwedge_{i=z}^{1} W_i = W_z \wedge S_j[c_j(p_{1,z-1})], \tag{3.3}$$

$$b_{l_j}(S_j[c_j(p_{1,z})]) = \bigwedge_{i=z}^{z-l_j+1} W_i, \text{ em que} \tag{3.4}$$

$$W_i = \bigvee_{n=1}^{i} c_{n,j}. \tag{3.5}$$

Este trabalho propõe, mas não restringe, quatro maneiras diferentes de computar os valores resultantes das métricas-combinadas em um sub-caminho: minimizativa, maximizativa, aditiva ou multiplicativa. As sínteses léxicas utilizadas são, portanto, denominadas a partir das operações que realizam. A Tabela 3.1 ilustra a formação de cada letra e de cada palavra-peso das sínteses léxicas propostas neste trabalho.

Tabela 3.1: Sínteses Léxicas

Tipo de Sínteses Léxicas	**Formação da Palavra-Peso** $\mathbf{S[c(p_{1,z})] = W_z; W_{z-1}; \ldots ; W_1}$	**Formação da Letra**
Minimizativa	$S[c(p_{1,z})] = min(c_1, \ldots, c_z); min(c_1, \ldots, c_{z-1}); \ldots ; c_1$	$W_i = min(c_1, \ldots, c_i)$
Maximizativa	$S[c(p_{1,z})] = max(c_1, \ldots, c_z); max(c_1, \ldots, c_{z-1}); \ldots ; c_1$	$W_i = max(c_1, \ldots, c_i)$
Aditiva	$S[c(p_{1,z})] = (c_1 + \cdots + c_z); (c_1 + \cdots + c_{z-1}); \ldots ; c_1$	$W_i = \sum_{j=1}^{i} c_j$
Multiplicativa	$S[c(p_{1,z})] = (c_1 \times \cdots \times c_z); (c_1 \times \cdots . \times c_{z-1}); \ldots ; c_1$	$W_i = \prod_{j=1}^{i} c_j$

As operações binárias $\vee$ e $\wedge$ devem obedecer às Propriedades 3.1 e 3.2, descritas a seguir.

Propriedade 3.1. A operação $\vee$ é idempotente, comutativa e associativa. Cujos elementos neutro ou identidade ϕ estão listados na Tabela 3.2.

Tabela 3.2:
Elemento
neutro de $\vee$

Operação $\vee$	ϕ
Minimizativa	$+\infty$
Maximizativa	$-\infty$
Aditiva	0
Multiplicativa	1

Propriedade 3.2. A operação $\wedge$ é associativa e possui um elemento identidade ou neutro $e = \{\}$.

3.1.4 Relação de Ordenação Léxica Multi-dimensional $\preceq_{ML}$

Com o objetivo de ordenar os diferentes caminhos que possam existir entre dois nós de uma rede, este trabalho propõe uma relação de ordenação léxica multi-dimensional $\preceq_{ML}$ que se baseia na estrutura de ordem ou ordenação léxica apresentada na Seção 2.1.1.1, em que são obedecidas as propriedades de reflexividade e totalidade.

Assim, dados dois caminhos $\alpha_{1,z}$ e $\beta_{1,z}$ que conectam o mesmo par de nós origem-destino da rede, em que as palavras-peso multi-dimensionais resultantes são dadas, respectivamente, por $\overline{S}[\overline{C}(\alpha_{1,z})] =$

$b_{l_1}(S_1[c_1(\alpha_{1,z})]); b_{l_2}(S_2[c_2(\alpha_{1,z})]); \ldots; b_{l_k}(S_k[c_k(\alpha_{1,z})])$ e $\overline{S}[\overline{C}(\beta_{1,z}) = b_{l_1}(S_1[c_1(\beta_{1,z})]); b_{l_2}(S_2[c_2(\beta_{1,z})]); \ldots; b_{l_k}(S_k[c_k(\beta_{1,z})])$, diz-se que o caminho $\beta_{1,z}$ é mais otimizado ou equivalente ao caminho $\alpha_{1,z}$, ou seja, $\alpha_{1,z} \preceq \beta_{1,z}$ se e somente se $\overline{S}[\overline{C}(\alpha_{1,z})] \preceq_{ML} \overline{S}[\overline{C}(\beta_{1,z})]$.

Por outro lado, define-se que $\overline{S}[\overline{C}(\alpha_{1,z})] \preceq_{ML} \overline{S}[\overline{C}\beta_{1,z})]$ se e somente se $\overline{S}[\overline{C}(\alpha_{1,z})] \prec_{ML} \overline{S}[\overline{C}(\beta_{1,z})]$ ou $S[\overline{C}(\alpha_{1,z})] \equiv_{ML} \overline{S}[\overline{C}(\beta_{1,z})]$, em que:

- $\overline{S}[\overline{C}(\alpha_{1,z})] \equiv_{ML} \overline{S}[\overline{C}(\beta_{1,z})]$, se e somente se:

  $b_{l_1}(S_1[c_1(\alpha_{1,z})]) \equiv_L b_{l_1}(S_1[c_1(\beta_{1,z})]), b_{l_2}(S_2[c_2(\alpha_{1,z})]) \equiv_L b_{l_2}(S_2[c_2(\beta_{1,z})]), \ldots,$
  $b_{l_k}(S_k[c_k(\alpha_{1,z})]) \equiv_L b_{l_k}(S_k[c_k(\beta_{1,z})])$; e

- $\overline{S}[\overline{C}(\alpha_{1,z})] \prec_{ML} \overline{S}[\overline{C}(\beta_{1,z})]$, se e somente se:

  $b_{l_1}(S_1[c_1(\alpha_{1,z})]) \prec_L b_{l_1}(S_1[c_1(\beta_{1,z})])$, ou
  $b_{l_1}(S_1[c_1(\alpha_{1,z})]) \equiv_L b_{l_1}(S_1[c_1(\beta_{1,z})])$ e $b_{l_2}(S_2[c_2(\alpha_{1,z})]) \prec_L b_{l_2}(S_2[c_2(\beta_{1,z})])$, ou
  $\ldots b_{l_{k-1}}(S_{k-1}[c_{k-1}(\alpha_{1,z})]) \equiv_L b_{l_{k-1}}(S_{k-1}[c_{k-1}(\beta_{1,z})])$ e
  $b_{l_k}(S_k[c_k(\alpha_{1,z})]) \prec_L b_{l_k}(S_k[c_k(\beta_{1,z})])$.

Portanto, a ordenação léxica multi-dimensional $\preceq_{ML}$ estabelece uma relação de ordenação ou priorização entre as k restrições impostas na escolha do melhor caminho.

Como se pode impor uma restrição diferente na escolha do melhor caminho a cada métrica-combinada, cada restrição pode exigir diferentes critérios de ordenação que são representados pela relação de ordenação léxica $\preceq_L$ na definição da ordenação léxica multi-dimensional dada.

Sem perda de generalidade, adotando-se $b_{l_j} = b_\infty$ e dadas duas palavras-peso quaisquer $S_j[c_j(\alpha)]$ e $S_j[c_j(\beta)]$, com respectivas z e z' letras, define-se que:

- $S_j[c_j(\alpha)] \equiv_L S_j[c_j(\beta)]$ se e somente se $z = z'$ e:

$$W_1^{c_j(\alpha)} = W_1^{c_j(\beta)}, W_2^{c_j(\alpha)} = W_2^{c_j(\beta)}, \ldots, W_z^{c_j(\alpha)} = W_{z'}^{c_j(\beta)}, \text{ e}$$

- $S_j[c_j(\alpha)] \prec_L S_j[c_j(\beta)]$, se e somente se:

  $W_i^{c_j(\alpha)} = W_i^{c_j(\beta)}$ para $\forall\ i : 1 \leq i \leq z'$ e $z' \leq z$, ou
  $W_i^{c_j(\alpha)} = W_i^{c_j(\beta)}$ para $\forall\ i : \{z, z'\} \leq i < k$ e $W_k^{c_j(\alpha)} \prec W_k^{c_j(\beta)}$ para $1 \leq k \leq min(z, z')$,

  em que $\prec$ pode representar uma ordenação $<$ ou $>$ dependendo da estratégia de otimização adotada.

Com o objetivo de simplificar a notação, este trabalho denota $S_j[c_j(\alpha)] \prec_L S_j[c_j(\beta)]$ por $S_j[c_j(\alpha)] <_L S_j[c_j(\beta)]$, quando a relação $\prec$ adotada for $<$, e $S_j[c_j(\alpha)] \prec_L S_j[c_j(\beta)]$ por $S_j[c_j(\alpha)] >_L S_j[c_j(\beta)]$, quando a relação $\prec$ adotada for $>$. Genericamente, denota-se $S_j[c_j(\alpha)] \preceq_L S_j[c_j(\beta)]$ por $S_j[c_j(\alpha)] \leq_L S_j[c_j(\beta)]$, quando a relação $\prec$ adotada for $<$ ou $S_j[c_j(\alpha)] \equiv_L S_j[c_j(\beta)]$, e $S_j[c_j(\alpha)] \preceq_L S_j[c_j(\beta)]$ por $S_j[c_j(\alpha)] \geq_L S_j[c_j(\beta)]$, quando a relação $\prec$ adotada for $>$ ou $S_j[c_j(\alpha)] \equiv_L S_j[c_j(\beta)]$.

A existência de mais de uma letra nas palavras-peso resultantes dos caminhos permite um novo critério de desempate para os caminhos cujas primeiras letras são idênticas, ou seja, para caminhos equivalentes segundo proposto por Gouda e Schneider [23].

Já a existência de mais de uma palavra-peso na palavra-peso multi-dimensional permite um critério de desempate para os caminhos cujas primeiras palavras-peso são idênticas, ou seja, para caminhos equivalentes segundo proposto por Sobrinho em [24, 36], tornando independente a busca do melhor caminho segundo diferentes estratégias de otimização.

Assim, dado o conjunto de valores $\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{s,d}})]$, composto pelas palavras-peso multi-dimensionais correspondentes a cada elemento do conjunto $\mathbf{P_{s,d}}$, composto por todos os caminhos elegíveis que conectam o mesmo par de nós origem (s) e destino (d), define-se $\Theta(\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{s,d}})])$ como um elemento cota superior do conjunto $\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{s,d}})]$, segundo o critério da ordenação léxica multi-dimensional $\preceq_{ML}$ adotada, e $\lceil(\mathbf{P_{s,d}})$ como o caminho correspondente à palavra-peso multi-dimensional $\Theta(\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{s,d}})])$.

Cabe comentar que o conjunto dos caminhos elegíveis é um subconjunto ordenado de caminhos que está contido no conjunto de todos os caminhos possíveis que conectam o mesmo par de nós origem (s) e destino (d). Na prática, para os algoritmos de roteamento *hop-by-hop*, os caminhos elegíveis são caminhos simples e elementares.

3.1.5 Resumo da Álgebra de Caminhos Proposta

Com o objetivo de facilitar a leitura dos próximos capítulos desta tese e a consulta às definições da álgebra de caminhos proposta, esta seção apresenta um resumo dos principais conceitos da estrutura matemática $\langle \mathbf{M}, \mathbf{F}, \overline{\mathbf{S}}, \preceq_{ML} \rangle$.

- $\mathbf{M} = M_1, M_2, \ldots, M_m$ é o conjunto das m métricas de roteamento utilizadas.

- $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2, ..., \mathbf{Q}_m$ é o conjunto dos conjuntos de valores que cada métrica pode assumir.

- $\mathbf{w} = \mathbf{w^{Q_1}}, \mathbf{w^{Q_2}}, \ldots, \mathbf{w^{Q_m}}$ é o conjunto dos conjuntos dos valores das métricas associados às arestas do dígrafo, em que $w_i^{Q_q}$ representa o valor Q_q da métrica q associado à aresta i, ou seja, $w_i^{Q_q}$ é o peso da aresta i referente à métrica q.

- $\mathbf{F}$ é um conjunto de k funções de combinação de métricas f_j, com $j : 1 \leq j \leq k$, que mapeiam os pesos $w_i^{Q_1}, w_i^{Q_2}, \ldots, w_i^{Q_m}$ de uma aresta i em uma correspondente métrica-combinada $c_{i,j} = f_j(w_i^{Q_1}, w_i^{Q_2}, ..., w_i^{Q_m})$, em que k é o número de restrições impostas à busca do melhor caminho.

- $\overline{C}$ uma matriz de métricas-combinadas, em que o vetor: $\overline{C}_i = [f_1(w_i^{Q_1}, \ldots, w_i^{Q_m}); \ldots; f_j(w_i^{Q_1}, \ldots, w_i^{Q_m}); \ldots; f_k(w_i^{Q_1}, \ldots, w_i^{Q_m})]$ é o vetor de métricas-combinadas correspondente à aresta i.

- $\overline{\mathbf{S}}$ é o conjunto sínteses, em que cada síntese $S_j[.]$, por sua vez, é uma 3-upla $\langle \vee, \wedge, b \rangle$ de operações binárias aplicadas sobre os valores resultantes da métrica-combinada c_j ao longo de um caminho $p_{1,z}$, em que:

  - $S_j[c_j(p_{1,z})]$ é a palavra-peso resultante da síntese $S_j[.]$;
  - $b_{l_j}(S_j[c_j(p_{1,z})]) = W_z; W_{z-1}; \ldots; W_{z-l_j+1}$ é a abreviação de l_j letras da palavra-peso resultante; e
  - $\overline{S}[\overline{C}(p)] = b_{l_1}(S_1[c_1(p_{1,z})]); b_{l_2}(S_2[c_2(p_{1,z})]); \ldots; b_{l_j}(S_j[c_j(p_{1,z})]); \ldots;$ $b_{l_k}(S_k[c_k(p_{1,z})])$ é a palavra-peso multi-dimensional resultante de $\overline{\mathbf{S}}$.

- $\preceq_{ML}$ é uma relação de ordenação denominada ordenação léxica multi-dimensional, aplicada sobre as palavras-peso multi-dimensionais obtidas para todos os caminhos comparáveis em uma rede.

A Tabela 3.3 esboça de forma vetorial a aplicação das funções de combinação de métricas e sínteses ao longo de um caminho $p_{1,z} = l_1, l_2, ..., l_z$.

3.1.6 Exemplos de Utilização

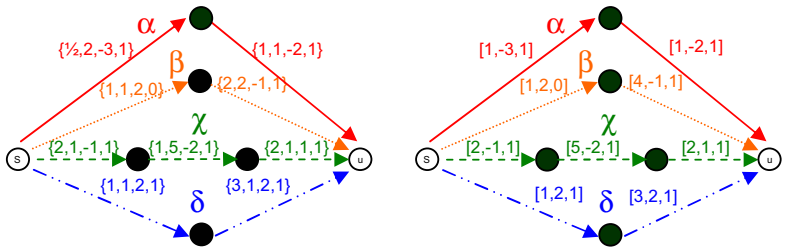
Com o objetivo de exemplificar a manipulação da álgebra de caminhos proposta, esta seção apresenta alguns exemplos da sua utilização.

Tabela 3.3: Representação vetorial da álgebra de caminhos proposta

Arestas	Pesos das Métricas $\mathbf{Q_1}$	$\mathbf{Q_2}$	$\dots$	$\mathbf{Q_m}$		$\downarrow \mathbf{j}$					
l_1	$w_1^{Q_1}$	$w_1^{Q_2}$	$\dots$	$w_1^{Q_m}$		$c_{1,1}$	$\dots$	$\mathbf{c_{1,j}}$	$\dots$	$c_{1,k}$	$\overline{C}_1$
l_2	$w_2^{Q_1}$	$w_2^{Q_2}$	$\dots$	$w_2^{Q_m}$		$c_{2,1}$	$\dots$	$\mathbf{c_{2,j}}$	$\dots$	$c_{2,k}$	$\overline{C}_2$
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
$\mathbf{l_i}$	$\mathbf{w_i^{Q_1}}$	$\mathbf{w_i^{Q_2}}$	$\dots$	$\mathbf{w_i^{Q_m}}$	$\xrightarrow{(\mathbf{f_1},\dots,\mathbf{f_j},\dots,\mathbf{f_k})}$	$\mathbf{c_{i,1}}$	$\dots$	$\mathbf{c_{i,j}}$	$\dots$	$\mathbf{c_{i,k}}$	$\overline{\mathbf{C}}_\mathbf{i}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
l_z	$w_z^{Q_1}$	$w_z^{Q_2}$	$\dots$	$w_z^{Q_m}$		$c_{z,1}$	$\dots$	$\mathbf{c_{z,j}}$	$\dots$	$c_{z,k}$	$\overline{C}_z$
						$S_1[c_1(p_{1,z})]$	$\dots$	$\mathbf{S_j[c_j(p_{1,z})]}$	$\dots$	$S_k[c_k(p_{1,z})]$	
						$b_{l_1}(S_1[c_1(p_{1,z})])\dots$		$\mathbf{b_{l_j}(S_j[c_j(p_{1,z})])}$	$\dots$	$b_{l_k}(S_k[c_k(p_{1,z})])$	$\overline{S}[\overline{C}(p_{1,z})]$

3.1.6.1 Exemplo Didático

Considere como exemplo a busca do melhor caminho entre dois vértices (s) e (u) indicados no dígrafo da Figura 3.1(a). Definindo-se $\mathbf{M} = \{M_1, M_2, M_3, M_4\}$ como o conjunto de métricas de roteamento adotadas e $\mathbf{F} = \{f(w_i^{Q_1}, w_i^{Q_2}) = w_i^{Q_1} \times w_i^{Q_2}, w_i^{Q_3}, w_i^{Q_4}\}$ como o conjunto das funções de combinação de métricas utilizadas, os resultados obtidos dos vetores de métricas-combinadas estão ilustrados na Figura 3.1(b).



(a) Dígrafo das métricas adotadas (b) Dígrafo das combinações das métricas

Figura 3.1: Exemplo de utilização da álgebra de caminhos

Utilizando-se, respectivamente, as sínteses léxicas minimizativa, maximizativa e multiplicativa para as componentes do vetor de métricas-combinadas (c_1, c_2, c_3) ao longo dos caminhos encontrados, obtêm-se os resultados das sínteses indicados na Tabela 3.4.

Adotando-se para as sínteses utilizadas, $\{S_1[.], S_2[.], S_3[.]\}$, a abreviação b_∞, ou seja, nenhuma abreviação, e as ordenações léxicas $\preceq_L$, respectivamente, como $\leq_L$, $\geq_L$ e $\leq_L$, obtêm-se os resultados indicados na Tabela 3.5.

A partir dessa classificação inicial e adotando-se a relação de ordenação léxica multi-dimensional $\preceq_{ML}$ dada por $(S_1[c_1(p)], S_2[c_2(p)], S_3[c_3(p)])$ $\prec_{ML}$

Tabela 3.4: Resultados das sínteses adotadas para as métricas-combinadas

$\mathbf{S_1[c_1]}$ **Léxica Minimizativa**	$\mathbf{S_2[c_2]}$ **Léxica Maximizativa**	$\mathbf{S_3[c_3]}$ **Léxica Multiplicativa**
$S_1[c_1(\alpha)] = 1;1$	$S_2[c_2(\alpha)] = -2;-3$	$S_3[c_3(\alpha)] = 1;1$
$S_1[c_1(\beta)] = 1;1$	$S_2[c_2(\beta)] = 2;2$	$S_3[c_3(\beta)] = 0;0$
$S_1[c_1(\chi)] = 2;2;2$	$S_2[c_2(\chi)] = 1;-1;-1$	$S_3[c_3(\chi)] = 1;1;1$
$S_1[c_1(\delta)] = 1;1$	$S_2[c_2(\delta)] = 2;2$	$S_3[c_3(\delta)] = 1;1$

Tabela 3.5: Resultados das ordenações léxicas da métricas-combinadas

Ordenação	$\mathbf{S_1[c_1]}$ **Léxica Minimizativa** $\leq_L$	$\mathbf{S_2[c_2]}$ **Léxica Maximizativa** $\geq_L$	$\mathbf{S_3[c_3]}$ **Léxica Multiplicativa** $\leq_L$
Melhor	$S_1[c_1(\chi)] = 2;2;2$	$S_2[c_2(\alpha)] = -2;-3$	$S_3[c_3(\alpha)] = S_3[c_3(\delta] = 1;1$
	$S_1[c_1(\alpha)] = S_1[c_1(\beta)] = S_1[c_1(\delta)] = 1;1$	$S_2[c_2(\chi)] = 1;-1;-1$	$S_3[c_3(\chi)] = 1;1;1$
Pior		$S_2[c_2(\beta)] = S_2[c_2(\delta)] = 2;2$	$S_3[c_3(\beta)] = 0;0$

$(S_1[c_1(p^{'})], S_2[c_2(p^{'})], S_3[c_3(p^{'})])$, obtém-se como melhor caminho o caminho χ, conforme indicado na Tabela 3.6.

Tabela 3.6: Resultado do exemplo de utilização da álgebra de caminhos

Ordenação	**Relação de Ordenação** $\preceq_{\mathbf{ML}}$	**Critério de Decisão**
Melhor	χ	$(\{\alpha, \beta, \delta\} \prec \chi)_{c_1}$
	α	$(\{\delta, \beta\} \prec \alpha)_{c_2}$
	δ	$(\beta \equiv \delta)_{c_2}$ mas $(\beta \prec \delta)_{c_3}$
Pior	β	$(\beta \equiv \delta)_{c_2}$ mas $(\beta \prec \delta)_{c_3}$

3.1.6.2 Modelagem dos Algoritmos de Pesquisa Primeiro na Largura e Primeiro na Profundidade

Conforme descrito nas Seções 2.2.3.2 e 2.2.3.1 desta tese, os algoritmos de pesquisa primeiro na largura e primeiro na profundidade têm como objetivo obter, em um dígrafo $D = (\mathbf{V}, \mathbf{E})$, todos os vértices acessíveis por qualquer vértice $(s) \in \mathbf{V}$, produzindo assim uma árvore enraizada em (s).

Utilizando a álgebra de caminhos proposta, esses algoritmos podem ser modelados por meio da estrutura matemática $\langle \mathbf{M}, \mathbf{F}, \overline{\mathbf{S}}, \preceq_{ML} \rangle$, em que:

- $\mathbf{M}$ é o conjunto da métrica de roteamento utilizada, ou seja, $M = \{\textit{número de saltos}\}$. Note que nesse caso, o algoritmo é um algoritmo de roteamento mono-restritivo;

- $\mathbf{Q}$ é o conjunto dos valores que a métrica "número de saltos" pode assumir, ou seja, $\mathbf{Q} = \Re_0^+$ e
- $\mathbf{w}$ é o conjunto dos valores da métrica associados às arestas do dígrafo, ou seja,

$$w_i = \begin{cases} 1 & \text{se } (i-1,i) \in \mathbf{E} \\ 0 & \text{se } (i-1,i) \notin \mathbf{E} \end{cases};$$

- $\mathbf{F} = \{f(w)\}$, em que $f(w) = w_i$ e, portanto, $c_i = w_i$;
- $\overline{C}$ é um vetor de métrica ao longo de um caminho, em que cada elemento $\overline{C}_i = w_i$ corresponde à aresta i;
- $\overline{\mathbf{S}} = \{S[.]\}$, em que a síntese $S[.]$ é dada pela 3-upla $\langle \vee, \wedge, b \rangle$ de operações binárias, definidas como:
  - $\vee$ é a operação de adição, ou seja, a síntese léxica é aditiva;
  - $\wedge$ é a operação de concatenação;
  - $b = b_1$ é a abreviação da primeira letra da palavra-peso obtida pela aplicação da síntese léxica aditiva ao longo de um caminho $p_{1,z}$ e,

  portanto, $\overline{S}[\overline{C}(p_{1,z})] = b_1(S[c(p_{1,z})]) = W_z = \sum_{i=1}^{z} w_i$ é a palavra-peso multi-dimensional correspondente ao caminho $p_{1,z}$; e
- $\preceq_{ML}$ é uma relação de ordenação léxica em que $\preceq_L$ é dada por $\geq$.

3.1.6.3 Modelagem dos Algoritmos de Dijkstra e Bellman-Ford

Tanto o algoritmo de Dijkstra, quanto o de Bellman-Ford, são algoritmos de roteamento mono-restritivo que têm como objetivo resolver o problema de busca de caminhos mais curtos de única origem em um dígrafo orientado ponderado $D = (\mathbf{V}, \mathbf{E})$. Entretanto, para Bellman-Ford impõe-se que o valor da métrica adotada esteja no conjunto $\Re$ e para Dijkstra, que o valor da métrica adotada esteja no conjunto $\Re_{\mathbf{0}}^+$.

Utilizando a álgebra de caminhos proposta, esses algoritmos são modelados de maneira similar, com exceção do conjunto dos valores que a métrica adotada pode assumir. Assim, definem-se:

- $\mathbf{M}$ como o conjunto da métrica de roteamento utilizada, que é a distância;
  - $\mathbf{Q}$ como o conjunto dos valores que a métrica "distância" pode assumir (para Bellman-Ford $\mathbf{Q} = \Re$ e para Dijkstra $\mathbf{Q} = \Re_0^+$) e

– $\mathbf{w}$ como o conjunto dos valores da métrica associados às arestas do dígrafo, ou seja,

$$w_i = \begin{cases} d(v-1,v) & \text{se } (v-1,i) \in \mathbf{E} \\ \infty & \text{se } (v-1,i) \notin \mathbf{E} \end{cases},$$

em que $d(v-1,v)$ é o valor da distância entre os vértices $(v-1)$ e (v) do dígrafo, ou seja, a função peso definida nas Seções 2.2.3.3 e 2.2.3.4;

- $\mathbf{F} = \{f(w)\}$, em que $f(w) = w_i$ e, portanto, $c_i = d_i$;
- $\overline{C}$ é um vetor de métrica ao longo de um caminho, em que cada elemento $\overline{C}_i = w_i$ corresponde à aresta i;
- $\overline{\mathbf{S}} = \{S[.]\}$, em que a síntese $S[.]$ é dada pela 3-upla $\langle \vee, \wedge, b \rangle$ de operações binárias definidas como:
  - $\vee$ é a operação de adição, ou seja, a síntese léxica é aditiva;
  - $\wedge$ é a operação de concatenação;
  - $b = b_1$ é a abreviação da primeira letra da palavra-peso obtida pela aplicação da síntese léxica aditiva ao longo de um caminho $p_{1,z}$, e

  portanto, $\overline{S}[\overline{C}(p_{1,z})] = b_1(S[c(p_{1,z})]) = W_z = \sum_{i=1}^{z} w_i$ é a palavra-peso multi-dimensional correspondente ao caminho $p_{1,z}$; e
- $\preceq_{ML}$ é uma relação de ordenação léxica em que $\preceq_L$ e dada por $\geq$.

3.2 Descrição das Propriedades Conhecidas

Conforme pôde ser observado na Seção 3.1, em função do problema abordado, os algoritmos de roteamento são modelados pela álgebra de caminhos de maneira muito semelhante, o que permite vincular algumas de suas macro-características à observância de algumas propriedades da álgebra.

Em [23] e [35], os autores definem cinco propriedades fundamentais, analisadas pela álgebra de caminhos que, se atendidas pelo algoritmo de roteamento, garantem a convergência e/ou a convergência ao caminho ótimo. São elas: monotonicidade, monotonicidade estrita, isotonicidade, isotonicidade estrita e liberdade.

Em função da relevância dessas propriedades para este trabalho, esta seção reapresenta essas propriedades sob a ótica da álgebra de caminhos proposta.

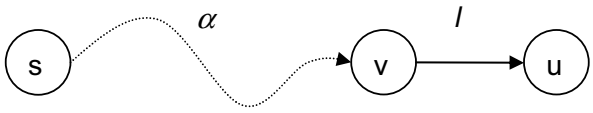


Figura 3.2: Conceito de monotonicidade

3.2.1 Monotonicidade e Monotonicidade Estrita

Conforme definida na Seção 2.1.1.1, a propriedade de monotonicidade implica que, dados dois conjuntos ordenados $\mathbf{V}$ e $\mathbf{U}$: $\mathbf{U} \subseteq \mathbf{V}$, existindo os elementos supremos e ínfimos dos conjuntos $\mathbf{U}$ e $\mathbf{V}$, então $s_{\mathbf{U}} \preceq s_{\mathbf{V}}$ e $i_{\mathbf{V}} \preceq i_{\mathbf{U}}$. Na Seção 2.3.1.2, a propriedade de monotonicidade é re-apresentada segundo a álgebra de Sobrinho como, dado $\forall\ l \in \mathbf{L}$ e $\forall\ \alpha \in \sum \Rightarrow f(\alpha) \preceq f(l \oplus \alpha)$.

Colocando em palavras, a propriedade de monotonicidade traduz para a álgebra de caminhos a característica de um caminho não se tornar mais otimizado à medida que ele se estende, ou seja, a monotonicidade é a propriedade que garante que, para todos os possíveis caminhos em uma rede, que unem um nó (s) de origem a um nó (d) de destino, a partir do nó de origem e à medida que os caminhos vão se estendendo até o destino, os sub-caminhos estendidos são menos otimizados ou iguais ao sub-caminho não-estendido.

Observe o exemplo da Figura 3.2. Se houver monotonicidade, então o caminho α é mais otimizado ou igual ao caminho $(\alpha \circ l)$, ou seja, $\alpha \circ l \preceq \alpha$.

Assim, segundo a álgebra de caminhos proposta, define-se a propriedade de monotonicidade como:

$$\overline{S}[\overline{C}(\alpha \circ l)] \preceq_{ML} \overline{S}[\overline{C}(\alpha)]. \tag{3.6}$$

Como a propriedade de monotonicidade está vinculada aos valores que as métricas podem assumir, às funções de combinação de métricas e à relação de ordenação léxica multi-dimensional adotadas, sua análise será exemplificada a seguir em um problema de roteamento mono-restritivo e, em seqüência, será estendida para um caso genérico de roteamento multi-restritivo.

Considere um exemplo em que a métrica adotada é a largura de banda disponível em cada enlace da rede e que a estratégia de otimização do caminho é obter, como caminho mais otimizado, aquele que possui maior largura de banda. Portanto, definem-se:

- **M** como o conjunto da métrica de roteamento utilizada que é a largura de banda;

- $\mathbf{Q}$ é o conjunto dos valores que a largura de banda pode assumir, ou seja, $\mathbf{Q} = \Re_0^+$;
- $\mathbf{w}$ é o conjunto dos valores da largura de banda associados às arestas do dígrafo, ou seja,

$$w_i = \begin{cases} banda(v-1,v) & \text{se } (v-1,v) \in \mathbf{E} \\ 0 & \text{se } (v-1,v) \notin \mathbf{E} \end{cases},$$

em que $banda(v-1,v)$ é o valor da largura de banda entre os nós $(v-1)$ e (v) da rede;

- $\mathbf{F} = \{f(w)\}$, em que $f(w) = w_i$ e, portanto, $c_i = w_i$;
- $\overline{C}$ é um vetor de métrica ao longo de um caminho, em que cada elemento $\overline{C}_i = w_i$ corresponde à aresta i;
- $\overline{\mathbf{S}} = \{S[.]\}$, em que a síntese $S[.]$ é dada pela 3-upla $\langle \vee, \wedge, b \rangle$ de operações binárias definidas como:
  - $\vee$ é a operação de minimização, ou seja, a síntese léxica é minimizativa;
  - $\wedge$ é a operação de concatenação;
  - $b = b_\infty$ implica a não abreviação da palavra-peso obtida pela aplicação da síntese léxica minimizativa ao longo de um caminho $p_{1,z}$ e,

  portanto, $\overline{S}[\overline{C}(p_{1,z})] = b_\infty(S[c(p_{1,z})]) = W_z; W_{z-1}; ...; W_2; W_1$ é a palavra-peso multi-dimensional correspondente ao caminho $p_{1,z}$, em que $W_i = min(w_1, \ldots, w_i)$; e
- $\preceq_{ML}$ é uma relação de ordenação léxica, em que $\preceq_L$ é dada por $\leq_L$.

Assim, dado um caminho α qualquer, a palavra-peso multi-dimensional é dada por $\overline{S}[\overline{C}(\alpha)] = min(w_1, ..., w_z); min(w_1, ..., w_{z-1}); ...; w_1$, cujo comprimento é z. Estendendo-se esse caminho pelo enlace l, a palavra-peso multi-dimensional resultante é dada por $\overline{S}[\overline{C}(\alpha \circ l)] = min(w_1, ..., w_z, w_l); min(w_1, ..., w_z); ...; w_1$, cujo comprimento é $z+1$.

Por construção $W_{z+1} \leq W_z$, pois $min(w_1, ..., w_z, w_l) \leq min(w_1, ..., w_z)$. Portanto, se $W_{z+1} < W_z$, tem-se que $S[C(\alpha \circ l)] \prec_L S[C(\alpha)]$. Por outro lado, se $W_{z+1} = W_z$, por construção, para $\forall\ z-1 \leq i \leq 1 \Rightarrow W_{i+1} \leq W_i$. No caso em que $W_{i+1} < W_i$ para algum $i \in [1, z-1]$, então $S[C(\alpha \circ l)] \leq_L S[C(\alpha)]$ e, no caso em que $W_{i+1} = W_i$ para todos os $i \in [1, z-1]$, então $S[C(\alpha \circ l)] <_L S[C(\alpha)]$ pois a palavra-peso $S[C(\alpha)]$ tem uma letra a menos que a palavra-peso $S[C(\alpha \circ l)]$.

Extrapolando esse raciocínio para as quatro sínteses léxicas propostas e mantendo a abreviação $b = b_\infty$, as relações de ordenação léxica $\preceq_L$ obtidas entre os caminhos α e $(\alpha \circ l)$, em função do critério de otimização adotado, estão listadas na Tabela 3.7.

Tabela 3.7: Relações de ordenação léxica entre os caminhos α e $(\alpha \circ l)$ para $b = b_\infty$

Sínteses Léxicas	Relação entre $\mathbf{W_{i+1}}$ e $\mathbf{W_i}$ para $\forall\ \mathbf{i} \in [\mathbf{1}, \mathbf{z}]$	Relação entre $\mathbf{S_j}[\mathbf{C_j}(\alpha \circ \mathbf{l})]$ e $\mathbf{S_j}[\mathbf{C_j}(\alpha)]$ para $\forall\ \mathbf{j} \in [\mathbf{1}, \mathbf{k}]$
Minimizativa	$\mathbf{W_{i+1}} \leq \mathbf{W_i}$	$S_j[C_j(\alpha \circ l)] <_L S_j[C_j(\alpha)]$
Maximizativa	$\mathbf{W_{i+1}} \geq \mathbf{W_i}$	$S_j[C_j(\alpha \circ l)] >_L S_j[C_j(\alpha)]$
Aditiva	$\mathbf{W_{i+1}} \geq \mathbf{W_i}$ para $c_i \leq 0$ ou $c_i \geq 0$	$S_j[C_j(\alpha \circ l)] >_L S_j[C_j(\alpha)]$
	Não estabelecida para $-\infty < c_i < \infty$	Não estabelecida
Multiplicativa	$\mathbf{W_{i+1}} \leq \mathbf{W_i}$ para $0 \leq c_i \leq 1$	$S_j[C_j(\alpha \circ l)] <_L S_j[C_j(\alpha)]$
	$\mathbf{W_{i+1}} \geq \mathbf{W_i}$ para $1 \leq c_i < \infty$	$S_j[C_j(\alpha \circ l)] >_L S_j[C_j(\alpha)]$
	Não estabelecida para $-\infty < c_i < \infty$	Não estabelecida

Assim, dependendo da síntese e da correspondente relação de ordenação adotada, conforme indicado na Tabela 3.8, é possível verificar se a propriedade de monotonicidade é ou não obedecida.

Tabela 3.8: Análise da monotonicidade $\times$ síntese $\times$ ordenação léxica

Sínteses Léxicas	$\preceq_\mathbf{L} \Rightarrow \geq_\mathbf{L}$	$\preceq_\mathbf{L} \Rightarrow \leq_\mathbf{L}$
Minimizativa	monotônica	não-monotônica
Maximizativa	não-monotônica	monotônica
Aditiva	monotônica para $c_i \leq 0$ ou $c_i \geq 0$	não-monotônica
	não-monotônica para $-\infty < c_i < \infty$	
Multiplicativa	monotônica para:	monotônica para:
	$1 \leq c_i < \infty$ ou $-\infty < c_i \leq -1$	$0 \leq c_i \leq 1$ ou $-1 \leq c_i \leq 0$
	não-monotônica para:	não-monotônica para:
	$-\infty < c_i < \infty$	$-\infty < c_i < \infty$

Observe que para os problemas de roteamento mono-restritivo, a Tabela 3.8 permite verificar diretamente se o algoritmo de roteamento é monotônico ou não.

Definindo-se a propriedade de monotonicidade estrita como uma restrição da propriedade de monotonicidade, em que:

$$\overline{S}[\overline{C}(\alpha \circ l)] \prec_{ML} \overline{S}[\overline{C}(\alpha)], \tag{3.7}$$

nas Tabelas 3.7 e 3.8 verifica-se que para $b = b_\infty$, sempre que a propriedade de monotonicidade é obedecida, então a propriedade de monotonicidade estrita também é.

No entanto, aplicando-se uma abreviação $b = \{b_l : l < z \text{ e } l \neq \infty\}$ para as quatro sínteses léxicas propostas, obtêm-se as relações de ordenação léxica $\preceq_L$ entre os caminhos α e $(\alpha \circ l)$, listadas na Tabela 3.9, em função do critério de otimização adotado.

Tabela 3.9: Relações de ordenação léxica entre os caminhos α e $(\alpha \circ l)$ para $b = \{b_l : l < z \text{ e } l \neq \infty\}$

Sínteses Léxicas	Relação entre $\mathbf{W_{i+1}}$ e $\mathbf{W_i}$ para $\forall\ \mathbf{i} \in [\mathbf{1}, \mathbf{z}]$	Relação entre $\mathbf{b_l}(\mathbf{S_j}[\mathbf{C_j}(\alpha \circ \mathbf{l})])$ e $\mathbf{b_l}(\mathbf{S_j}[\mathbf{C_j}(\alpha)])$ para $\forall\ \mathbf{j} \in [\mathbf{1}, \mathbf{k}]$
Minimizativa	$\mathbf{W_{i+1}} \leq \mathbf{W_i}$	$b_l(S_j[C_j(\alpha \circ l)]) \leq_L b_l(S_j[C_j(\alpha)])$
Maximizativa	$\mathbf{W_{i+1}} \geq \mathbf{W_i}$	$b_l(S_j[C_j(\alpha \circ l)]) \geq_L b_l(S_j[C_j(\alpha)])$
Aditiva	$\mathbf{W_{i+1}} \geq \mathbf{W_i}$ para $c_i \leq 0$ ou $c_i \geq 0$	$b_l(S_j[C_j(\alpha \circ l)]) \geq_L b_l(S_j[C_j(\alpha)])$
	$\mathbf{W_{i+1}} > \mathbf{W_i}$ para $c_i < 0$ ou $c_i > 0$	$b_l(S_j[C_j(\alpha \circ l)]) >_L b_l(S_j[C_j(\alpha)])$
	Não estabelecida para $-\infty < c_i < \infty$	Não estabelecida
Multiplicativa	$\mathbf{W_{i+1}} \leq \mathbf{W_i}$ para $0 \leq c_i \leq 1$	$b_l(S_j[C_j(\alpha \circ l)]) \leq_L b_l(S_j[C_j(\alpha)])$
	$\mathbf{W_{i+1}} < \mathbf{W_i}$ para $0 < c_i < 1$	$b_l(S_j[C_j(\alpha \circ l)]) <_L b_l(S_j[C_j(\alpha)])$
	$\mathbf{W_{i+1}} \geq \mathbf{W_i}$ para $1 \leq c_i < \infty$	$b_l(S_j[C_j(\alpha \circ l)]) \geq_L b_l(S_j[C_j(\alpha)])$
	$\mathbf{W_{i+1}} > \mathbf{W_i}$ para $1 < c_i < \infty$	$b_l(S_j[C_j(\alpha \circ l)]) >_L b_l(S_j[C_j(\alpha)])$
	Não estabelecida para $-\infty < c_i < \infty$	Não estabelecida

Por fim, dependendo da síntese e da correspondente relação de ordenação adotada, a Tabela 3.10 indica se a propriedade de monotonicidade estrita é ou não obedecida.

As Tabelas 3.9 e 3.10 permitem verificar se um algoritmo de roteamento mono-restritivo é monotônico estrito ou não para $b = b_l : l < z$ e $l \neq \infty$.

Ampliando essa análise para um problema de roteamento multi-restritivo com k restrições, se o algoritmo de roteamento é monotônico, então a equação $\overline{S}[\overline{C}(\alpha \circ l)] \preceq_{ML} \overline{S}[\overline{C}(\alpha)]$ é reescrita como:

- $\overline{S}[\overline{C}(\alpha \circ l)] \equiv_{ML} \overline{S}[\overline{C}(\alpha)]$, se e somente se:

$$b_{l_1}(S_1[c_1(\alpha \circ l)]) \equiv_L b_{l_1}(S_1[c_1(\alpha)]), b_{l_2}(S_2[c_2(\alpha \circ l)]) \equiv_L b_{l_2}(S_2[c_2(\alpha)]), \ldots,$$
$$b_{l_k}(S_k[c_k(\alpha \circ l)]) \equiv_L b_{l_k}(S_k[c_k(\alpha)]) \text{ e} \tag{3.8}$$

Tabela 3.10: Análise da monotonicidade estrita× síntese × ordenação léxica

Sínteses Léxicas	$\prec_{\mathbf{L}} \Rightarrow >_{\mathbf{L}}$	$\prec_{\mathbf{L}} \Rightarrow <_{\mathbf{L}}$
Minimizativa	não-monotônica estrita	não-monotônica
Maximizativa	não-monotônica	não-monotônica estrita
Aditiva	monotônica estrita para $c_i < 0$ ou $c_i > 0$	não-monotônica
	não monotônica estrita para $c_i \leq 0$ ou $c_i \geq 0$	
	não-monotônica para $-\infty < c_i < \infty$	
Multiplicativa	monotônica estrita para:	monotônica estrita para:
	$1 < c_i < \infty$ ou $-\infty < c_i < -1$	$0 < c_i < 1$ ou $-1 < c_i < 0$
	não monotônica estrita para:	não monotônica estrita para:
	$1 \leq c_i < \infty$ ou $-\infty < c_i \leq -1$	$0 \leq c_i \leq 1$ ou $-1 \leq c_i \leq 0$
	não-monotônica para:	não-monotônica para:
	$-\infty < c_i < \infty$	$-\infty < c_i < \infty$

- $\overline{S}[\overline{C}(\alpha \circ l)] \prec_{ML} \overline{S}[\overline{C}(\alpha)]$, se e somente se:

$$\begin{aligned}
&b_{l_1}(S_1[c_1(\alpha \circ l)]) \prec_L b_{l_1}(S_1[c_1(\alpha)]), \text{ ou} \\
&b_{l_1}(S_1[c_1(\alpha \circ l)]) \equiv_L b_{l_1}(S_1[c_1(\alpha)]) \text{ e } b_{l_2}(S_2[c_2(\alpha \circ l)]) \prec_L b_{l_2}(S_2[c_2(\alpha)]), \text{ ou} \\
&\ldots b_{l_{k-1}}(S_{k-1}[c_{k-1}(\alpha \circ l)]) \equiv_L b_{l_{k-1}}(S_{k-1}[c_{k-1}(\alpha)]) \text{ e} \\
&b_{l_k}(S_k[c_k(\alpha \circ l)]) \prec_L b_{l_k}(S_k[c_k(\alpha)]).
\end{aligned} \tag{3.9}$$

Analisando as equações 3.8 e 3.9, observa-se que para a propriedade de monotonicidade ser obedecida é necessário que cada restrição imposta seja monotônica. Por outro lado, da equação 3.9, observa-se que para a propriedade de monotonicidade estrita ser obedecida é necessário que as primeiras $k-1$ restrições impostas sejam monotônicas e a última restrição k seja monotônica estrita.

3.2.2 Isotonicidade e Isotonicidade Estrita

Na Seção 2.3.1.2, a propriedade de isotonicidade é apresentada, segundo Sobrinho, como $\forall\ l \in \mathbf{L}$ e $\forall\ \alpha, \beta \in \sum$, então $f(\alpha) \preceq f(\beta) \Rightarrow f(l \oplus \alpha) \preceq f(l \oplus \beta)$ e, na Seção 2.3.2.1, essa mesma propriedade é reapresentada, segundo Gouda e Schneider, como $m \prec m' \Rightarrow (met(m,w) \prec met(m',w)$ ou $met(m,w) = met(m',w))$.

Colocando em palavras, a propriedade de isotonicidade traduz para álgebra de caminhos a característica da manutenção da relação de ordenação entre dois caminhos distintos, entre o mesmo par de nós origem-destino, quando eles se estendem por um mesmo enlace l. Observe o exemplo da Figura 3.3. Se houver

isotonicidade, então dados dois caminhos α e β, que conectam o nós (s) e (v), em que $\alpha \preceq \beta$, então ao se estender esses caminhos pelo enlace l para conectar o nó (u), os caminhos estendidos $(\alpha \circ l)$ e $(\beta \circ l)$ mantêm a mesma relação de ordenação dada pelos caminhos não estendidos, ou seja, $(\alpha \circ l) \preceq (\beta \circ l)$.

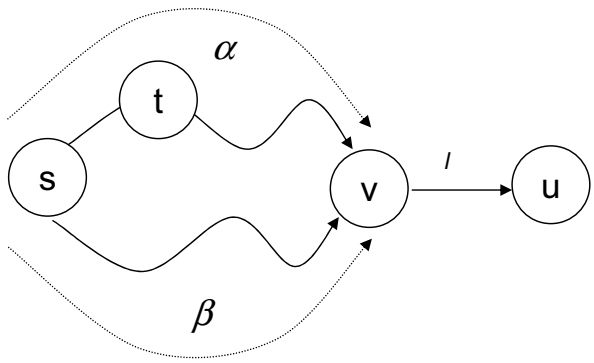


Figura 3.3: Exemplo de isotonicidade

Portanto, segundo a álgebra de caminhos proposta, define-se a propriedade de isotonicidade como:

$$\overline{S}[\overline{C}(\alpha)] \preceq_{ML} \overline{S}[\overline{C}(\beta)] \Rightarrow \overline{S}[\overline{C}(\alpha \circ l)] \preceq_{ML} \overline{S}[\overline{C}(\beta \circ l)]. \tag{3.10}$$

Como a propriedade de isotonicidade está vinculada aos valores que as métricas podem assumir, às funções de combinação de métricas e à relação de ordenação léxica multi-dimensional adotadas, sua análise será exemplificada primeiramente em um problema de roteamento mono-restritivo.

Considere o mesmo exemplo de problema mono-restritivo dado na Seção 3.2.1, em que a métrica adotada é a largura de banda disponível em cada enlace e a estratégia de otimização do caminho é obter, como melhor caminho, o caminho que possui maior largura de banda, utilizando as palavras-peso resultantes não abreviadas.

Dados dois caminhos comparáveis α e β entre um par de nós origem-destino, as respectivas palavras-peso multi-dimensionais são dadas por $\overline{S}[\overline{C}(\alpha)] = min(w_1, \ldots, w_z); min(w_1, \ldots, w_{z-1}); \ldots; w_1 = W_z^\alpha; W_{z-1}^\alpha; \ldots; = W_1^\alpha$ e $\overline{S}[\overline{C}(\beta)] = min(w_1', \ldots, w_y'); min(w_1', \ldots, w_{y-1}'); \ldots; w_1' = W_y^\beta; W_{y-1}^\beta; \ldots; = W_1^\beta$, em que z e y são seus respectivos números de letras.

Ao estender esses dois caminhos por um mesmo enlace l, cujo peso é w_l, as palavras-peso multi-dimensionais resultantes são dadas, respectivamente, por $\overline{S}[\overline{C}(\alpha \circ l)] = W_{z+1}^\alpha; \overline{S}[\overline{C}(\alpha)]$ de comprimento $z+1$ e $\overline{S}[\overline{C}(\beta \circ l)] = W_{y+1}^\beta; \overline{S}[\overline{C}(\beta)]$ de comprimento $y+1$, em que $W_{z+1}^\alpha = min(w_1, \ldots, w_z, w_l)$ e

$W^{\beta}_{y+1} = min(w'_1, \ldots, w'_y, w_l)$. Então, se:

- $\overline{S}[\overline{C}(\alpha)] \equiv_L \overline{S}[\overline{C}(\beta)]$, ou seja, $z = y$ e $W^{\alpha}_i = W^{\beta}_i$ para todo $i \in [1, z = y]$, então, por construção, $\overline{S}[\overline{C}(\alpha \circ l)] \equiv_L \overline{S}[\overline{C}(\beta \circ l)]$;
- $\overline{S}[\overline{C}(\alpha)] \prec_L \overline{S}[\overline{C}(\beta)]$, em que:
  - $W^{\alpha}_i = W^{\beta}_i$ para todo $i \in [1, y]$ e $z > y$, ou seja, $min(w_1, \ldots, w_i) = min(w'_1, \ldots, w'_i)$ para todo $i \in [1, y]$ e $z > y$, tem-se que $W^{\alpha}_z = W^{\beta}_y$. Como $min(w_1, ..., w_z, w_l) = min(w'_1, \ldots, w'_y, w_l)$, ou seja, $W^{\alpha}_{z+1} = W^{\beta}_{y+1}$, em que $z + 1 > y + 1$, portanto, $\overline{S}[\overline{C}(\alpha)] \prec_L \overline{S}[\overline{C}(\beta)] \Rightarrow \overline{S}[\overline{C}(\alpha \circ l)] \prec_L \overline{S}[\overline{C}(\beta \circ l)]$; e
  - $W^{\alpha}_i = W^{\beta}_i$ para $\forall\ i : \{z, y\} \leq i < k$ e $W^{\alpha}_k \prec W^{\beta}_k$ para $k : 1 \leq k \leq \{z, y\}$. Então, se para os caminhos α e β existe um $k : 1 \leq k \leq \{z, y\}$, em que $W^{\alpha}_k \prec W^{\beta}_k$ e $W^{\alpha}_i = W^{\beta}_i$ para $\forall\ i : \{z, y\} \leq i < k$ e, por construção, $W^{\alpha}_{z+1} = W^{\beta}_{y+1}$, pois $min(w_1, \ldots, w_z, w_l) = min(w'_1, \ldots, w'_y, w_l)$, para os caminhos $(\alpha \circ l)$ e $(\beta \circ l)$, também existirá um $k : 1 \leq k \leq \{z, y\}$, tal que $W^{(\alpha \circ l)}_k \prec W^{(\beta \circ l)}_k$ e $W^{(\alpha \circ l)}_i = W^{(\beta \circ l)}_i$ para $\forall\ i : \{z, y\} \leq i < k$ e,

  portanto, a propriedade de isotonicidade é obedecida.

Observe que, nesse exemplo, segundo a álgebra de caminhos proposta, para verificar se a propriedade de isotonicidade é obedecida no problema de roteamento mono-restritivo, basta analisar se as novas letras, geradas pela aplicação da síntese léxica adotada nos caminhos estendidos, preservam a relação de ordenação léxica definida para os caminhos não estendidos, ou seja, se $S[c(\alpha)] \preceq S[c(\beta)] \Rightarrow W^{\alpha}_z \vee c_l \preceq W^{\beta}_y \vee c_l$. Aplicando, portanto, esse raciocínio para as demais sínteses propostas e mantendo a não abreviação das palavras-peso resultantes, observam-se os resultados indicados na Tabela 3.11.

Note que para os problemas de roteamento mono-restritivo, a Tabela 3.11 permite verificar diretamente se o algoritmo de roteamento é isotônico ou não.

Analisando a aplicação de uma abreviação $b = \{b_l : l \leq min(z, y)$ e $l \neq \infty\}$ no exemplo anterior, ou seja, $\overline{S}[\overline{C}(\alpha)] = b_l(min(w_1, \ldots, w_z); min(w_1, \ldots, w_{z-1}); \ldots; w_1) = W^{\alpha}_z; W^{\alpha}_{z-1}; \ldots; W^{\alpha}_{z-l+1}$ e $\overline{S}[\overline{C}(\beta)] = b_l(min(w'_1, \ldots, w'_y); min(w'_1, \ldots, w'_{y-1}); \ldots; w'_1) = W^{\beta}_y; W^{\beta}_{y-1}; \ldots; W^{\beta}_{z-l+1}$, com respectivas l letras, e $\overline{S}[\overline{C}(\alpha \circ l)] = b_l(W^{\alpha}_{z+1}; \overline{S}[\overline{C}(\alpha)])$ e $\overline{S}[\overline{C}(\beta \circ l)] = b_l(W^{\beta}_{y+1}; \overline{S}[\overline{C}(\beta)])$ com comprimentos l, em que $W^{\alpha}_{z+1} = min(w_1, \ldots, w_z, w_l)$ e $W^{\beta}_{y+1} = min(w'_1, \ldots, w'_y, w_l)$. Então, se:

Tabela 3.11: Análise da isotonicidade × síntese com $b = b_\infty$

Sínteses Léxicas	Obediência à Propriedade
Minimizativa	isotônica
Maximizativa	isotônica
Aditiva	isotônica para $0 \leq c_i$ ou $0 \geq c_i$
	não-isotônica para $-\infty < c_i < \infty$
Multiplicativa	isotônica para $0 \leq c_i$
	não-isotônica caso contrário

- $\overline{S}[\overline{C}(\alpha)] \equiv_L \overline{S}[\overline{C}(\beta)]$, como $W_i^\alpha = W_i^\beta$ para todo $i \in [1, l]$, por construção, $\overline{S}[\overline{C}(\alpha \circ l)] \equiv_L \overline{S}[\overline{C}(\beta \circ l)]$;

- $\overline{S}[\overline{C}(\alpha)] \prec_L \overline{S}[\overline{C}(\beta)]$, em que:

  – $W_i^\alpha = W_i^\beta$ para todo $i \in [1, y]$ e $z > y$. Como $\overline{S}[\overline{C}(\alpha)]$ e $\overline{S}[\overline{C}(\beta)]$ têm l letras, então $W_i^\alpha = W_i^\beta$ para $\forall\ i : \{z, y\} \leq i < l$, ou seja, $min(w_1, \ldots, w_i) = min(w_1^{'}, \ldots, w_i^{'})$ para todo $\forall\ i : \{z, y\} \leq i < l$, tem-se que $W_z^\alpha = W_y^\beta$. Como $min(w_1, ..., w_z, w_l) = min(w_1^{'}, \ldots, w_y^{'}, w_l)$, ou seja, $W_{z+1}^\alpha = W_{y+1}^\beta$, em que $z + 1 > y + 1$, portanto, $\overline{S}[\overline{C}(\alpha)] \prec_L \overline{S}[\overline{C}(\beta)] \Rightarrow \overline{S}[\overline{C}(\alpha \circ l)] \prec_L \overline{S}[\overline{C}(\beta \circ l)]$; e

  – $W_i^\alpha = W_i^\beta$ para $\forall\ i : \{z, y\} \leq i < k$ e $W_k^\alpha \prec W_k^\beta$ para $k : 1 \leq k \leq \{z, y\}$, como $\overline{S}[\overline{C}(\alpha)]$ e $\overline{S}[\overline{C}(\beta)]$ tem l letras, para que $\overline{S}[\overline{C}(\alpha)] \prec_L \overline{S}[\overline{C}(\beta)]$, tem-se que $W_i^\alpha = W_i^\beta$ para $\forall\ i : \{z, y\} \leq i < k$ e $W_k^\alpha \prec W_k^\beta$ para $k : \{z, y\} \leq k \leq l$.

  Então, se para os caminhos α e β existe um $k : \{z, y\} \leq k \leq l$, em que $W_k^\alpha \prec W_k^\beta$ e $W_i^\alpha = W_i^\beta$ para $\forall\ i : \{z, y\} \leq i < k$ e, por construção, $W_{z+1}^\alpha = W_{y+1}^\beta$, pois $min(w_1, \ldots, w_z, w_l) = min(w_1^{'}, \ldots, w_y^{'}, w_l)$, para os caminhos $(\alpha \circ l)$ e $(\beta \circ l)$ também existirá um $k : 1 \leq k \leq \{z, y\}$, tal que $W_k^{(\alpha \circ l)} \prec W_k^{(\beta \circ l)}$ e $W_i^{(\alpha \circ l)} = W_i^{(\beta \circ l)}$ para $\forall\ i : \{z, y\} \leq i < k$.

Estendendo essa mesma análise para as demais sínteses propostas e aplicando uma abreviação $b = \{b_l : l \leq min(z, y) \text{ e } l \neq \infty\}$ às palavras-peso resultantes, obtêm-se os mesmos resultados indicados na Tabela 3.11.

Definindo-se a propriedade de isotonicidade estrita como uma restrição da propriedade de isotonicidade, em que:

$$\overline{S}[\overline{C}(\alpha)] \prec_{ML} \overline{S}[\overline{C}(\beta)] \Rightarrow \overline{S}[\overline{C}(\alpha \circ l)] \prec_{ML} \overline{S}[\overline{C}(\beta \circ l)], \tag{3.11}$$

Tabela 3.12: Análise da isotonicidade estrita × síntese

Sínteses Léxicas	**Abreviação *b***	**Obediência à Propriedade**
Minimizativa	$b = b_\infty$ ou	isotônica estrita
	$b = \{b_l : l \leq min(z, y) \text{ e } l \neq \infty\}$	não-isotônica estrita
Maximizativa	$b = b_\infty$ ou	isotônica estrita
	$b = \{b_l : l \leq min(z, y) \text{ e } l \neq \infty\}$	não-isotônica estrita
Aditiva	$b = b_\infty$	isotônica estrita para
		$c_i \leq 0$ ou $c_i \geq 0$
		não-isotônica para
		$-\infty < c_i < \infty$
	$b = \{b_l : l \leq min(z, y) \text{ e } l \neq \infty\}$	isotônica estrita para
		$c_{i,j} < 0$ ou $c_i > 0$
		não-isotônica para
		$-\infty < c_i < \infty$
Multiplicativa	$b = b_\infty$	isotônica estrita para
		$0 \leq c_i$
		não-isotônica, caso contrário
	$b = \{b_l : l \leq min(z, y) \text{ e } l \neq \infty\}$	isotônica estrita para
		$0 < c_i$
		não-isotônica, caso contrário

observa-se, a partir dos exemplos anteriores, que para verificar se a propriedade de isotonicidade estrita é obedecida no problema de roteamento mono-restritivo, basta analisar se as novas letras, geradas pela aplicação da síntese léxica adotada nos caminhos estendidos, preservam a relação de ordenação léxica definida para os caminhos não estendidos, ou seja, se $S[c(\alpha)] \prec S[c(\beta)] \Rightarrow W_z^\alpha \vee c_l \prec W_y^\beta \vee c_l$. Aplicando esse raciocínio para as demais sínteses propostas com ou sem abreviação das palavras-peso resultantes, obtêm-se os resultados indicados na Tabela 3.12.

Ampliando essa análise para um problema do roteamento multi-restritivo com k restrições, se o algoritmo de roteamento é isotônico, então $\overline{S}[\overline{C}(\alpha)] \preceq_{ML} \overline{S}[\overline{C}(\beta)] \Rightarrow \overline{S}[\overline{C}(\alpha \circ l)] \preceq_{ML} \overline{S}[\overline{C}(\beta \circ l)]$ é reescrita como:

$\overline{S}[\overline{C}(\alpha)] \preceq_{ML} \overline{S}[\overline{C}(\beta)]$:

$$\begin{aligned}
&b_{l_1}(S_1[c_1(\alpha)]) \prec_L b_{l_1}(S_1[c_1(\beta)]), \text{ ou}\\
&b_{l_1}(S_1[c_1(\alpha)]) \equiv_L b_{l_1}(S_1[c_1(\beta)]) \text{ e } b_{l_2}(S_2[c_2(\alpha)]) \prec_L b_{l_2}(S_2[c_2(\beta)]), \text{ ou}\\
&\ldots b_{l_{k-1}}(S_{k-1}[c_{k-1}(\alpha)]) \equiv_L b_{l_{k-1}}(S_{k-1}[c_{k-1}(\beta)]) \text{ e}\\
&b_{l_k}(S_k[c_k(\alpha)]) \prec_L b_{l_k}(S_k[c_k(\alpha)]), \text{ ou}\\
&b_{l_1}(S_1[c_1(\alpha)]) \equiv_L b_{l_1}(S_1[c_1(\beta)]), b_{l_2}(S_2[c_2(\alpha)]) \equiv_L b_{l_2}(S_2[c_2(\beta)]), \ldots,\\
&b_{l_k}(S_k[c_k(\alpha)]) \equiv_L b_{l_k}(S_k[c_k(\beta)]) \Rightarrow
\end{aligned} \tag{3.12}$$

$\overline{S}[\overline{C}(\alpha \circ l)] \preceq_{ML} \overline{S}[\overline{C}(\beta \circ l)]$:

$$\begin{aligned}
&b_{l_1}(S_1[c_1(\alpha \circ l)]) \prec_L b_{l_1}(S_1[c_1(\beta \circ l)]), \text{ ou}\\
&b_{l_1}(S_1[c_1(\alpha \circ l)]) \equiv_L b_{l_1}(S_1[c_1(\beta \circ l)]) \text{ e}\\
&b_{l_2}(S_2[c_2(\alpha \circ l)]) \prec_L b_{l_2}(S_2[c_2(\beta \circ l)]), \text{ou} \ldots\\
&b_{l_{k-1}}(S_{k-1}[c_{k-1}(\alpha \circ l)]) \equiv_L b_{l_{k-1}}(S_{k-1}[c_{k-1}(\beta \circ l)]) \text{ e}\\
&b_{l_k}(S_k[c_k(\alpha \circ l)]) \prec_L b_{l_k}(S_k[c_k(\alpha \circ l)]), \text{ ou}\\
&b_{l_1}(S_1[c_1(\alpha \circ l)]) \equiv_L b_{l_1}(S_1[c_1(\beta \circ l)]), b_{l_2}(S_2[c_2(\alpha \circ l)]) \equiv_L b_{l_2}(S_2[c_2(\beta \circ l)]),\\
&\ldots, b_{l_k}(S_k[c_k(\alpha \circ l)]) \equiv_L b_{l_k}(S_k[c_k(\beta \circ l)]).
\end{aligned} \tag{3.13}$$

Analisando as equações 3.12 e 3.13, observa-se que para a propriedade de isotonicidade ser obedecida, é necessário que cada restrição imposta seja isotônica.

Utilizando-se esse mesmo raciocínio para a propriedade de isotonicidade estrita, se o algoritmo de roteamento é isotônico estrito, então a $\overline{S}[\overline{C}(\alpha)] \prec_{ML} \overline{S}[\overline{C}(\beta)] \Rightarrow \overline{S}[\overline{C}(\alpha \circ l)] \prec_{ML} \overline{S}[\overline{C}(\beta \circ l)]$ é reescrita como:

$\overline{S}[\overline{C}(\alpha)] \prec_{ML} \overline{S}[\overline{C}(\beta)]$:

$$\begin{aligned}
&b_{l_1}(S_1[c_1(\alpha)]) \prec_L b_{l_1}(S_1[c_1(\beta)]), \text{ ou}\\
&b_{l_1}(S_1[c_1(\alpha)]) \equiv_L b_{l_1}(S_1[c_1(\beta)]) \text{ e}\\
&b_{l_2}(S_2[c_2(\alpha)]) \prec_L b_{l_2}(S_2[c_2(\beta)]), \text{ ou}\\
&\ldots b_{l_{k-1}}(S_{k-1}[c_{k-1}(\alpha)]) \equiv_L b_{l_{k-1}}(S_{k-1}[c_{k-1}(\beta)]) \text{ e}\\
&b_{l_k}(S_k[c_k(\alpha)]) \prec_L b_{l_k}(S_k[c_k(\alpha)]) \Rightarrow
\end{aligned} \tag{3.14}$$

$\overline{S}[\overline{C}(\alpha \circ l)] \prec_{ML} \overline{S}[\overline{C}(\beta \circ l)]$:

$$\begin{aligned}
&b_{l_1}(S_1[c_1(\alpha \circ l)]) \prec_L b_{l_1}(S_1[c_1(\beta \circ l)]), \text{ ou} \\
&b_{l_1}(S_1[c_1(\alpha \circ l)]) \equiv_L b_{l_1}(S_1[c_1(\beta \circ l)]) \text{ e} \\
&b_{l_2}(S_2[c_2(\alpha \circ l)]) \prec_L b_{l_2}(S_2[c_2(\beta \circ l)]), \text{ ou} \\
&\ldots b_{l_{k-1}}(S_{k-1}[c_{k-1}(\alpha \circ l)]) \equiv_L b_{l_{k-1}}(S_{k-1}[c_{k-1}(\beta \circ l)]) \text{ e} \\
&b_{l_k}(S_k[c_k(\alpha \circ l)]) \prec_L b_{l_k}(S_k[c_k(\alpha \circ l)]).
\end{aligned} \tag{3.15}$$

Assim, analisando as equações 3.14 e 3.15, observa-se que se qualquer restrição imposta $j : 1 \leq j \leq k-1$ for isotônica, ou seja, se $b_{l_j}(S_j[c_j(\alpha)]) \preceq_L b_{l_j}(S_j[c_j(\beta)]) \Rightarrow b_{l_j}(S_j[c_j(\alpha \circ l)]) \preceq_L b_{l_j}(S_j[c_j(\beta \circ l)])$ e a restrição imposta k for isotônica estrita, ou seja, se $b_{l_k}(S_k[c_k(\alpha)]) \prec_L b_{l_k}(S_k[c_k(\beta)]) \Rightarrow b_{l_k}(S_k[c_k(\alpha \circ l)]) \prec_L b_{l_k}(S_k[c_k(\beta \circ l)])$, então propriedade de isotonicidade estrita é obedecida.

3.2.3 Liberdade

Em [36], Sobrinho define o conceito de ciclos livres como uma generalização do conceito de ciclos de distância positiva.

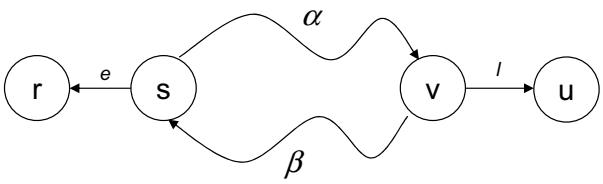


Figura 3.4: Exemplo de liberdade

Observe o exemplo indicado na Figura 3.4. Se o ciclo $p_c = \alpha \circ \beta$ é livre, então $\overset{n}{\cup}(\alpha \circ \beta) \prec \overset{n-1}{\cup}(\alpha \circ \beta)$, em que $\overset{n}{\cup}(\alpha \circ \beta)$ denota n concatenações do caminho $(\alpha \circ \beta)$, e $\overset{n}{\cup}(\beta \circ \alpha) \prec \overset{n-1}{\cup}(\beta \circ \alpha)$, em que $\overset{n}{\cup}(\beta \circ \alpha)$ denota n concatenações do caminho $(\beta \circ \alpha)$.

Assim, segundo a álgebra de caminhos proposta, define-se a propriedade de liberdade como:

$$\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta))] \prec_{ML} \overline{S}[\overline{C}(\overset{n-1}{\cup}(\alpha \circ \beta))] \tag{3.16}$$

ou

$$\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha))] \prec_{ML} \overline{S}[\overline{C}(\overset{n-1}{\cup}(\beta \circ \alpha))]. \tag{3.17}$$

Uma vez que a propriedade de liberdade está vinculada aos valores que a as métricas podem assumir, às funções de combinação de métricas e à relação

de ordenação léxica multi-dimensional adotadas, sua análise será exemplificada a seguir em um problema de roteamento mono-restritivo e em seguida estendida para um caso genérico de roteamento multi-restritivo.

Considere como exemplo, o problema mono-restritivo em que a métrica adotada é a distância de cada enlace da rede e que a estratégia de otimização do caminho é obter, como caminho mais otimizado, aquele que possui menor distância, ou seja, a relação de ordenação léxica $\preceq_L$ é dada por $\geq_L$, e as palavras-peso resultantes não são abreviadas, $b = b_\infty$.

Sem perda de generalidade, dados os dois caminhos α e β compostos somente por uma aresta e com palavras-peso multi-dimensionais dadas, respectivamente, por $\overline{S}[\overline{C}(\alpha)] = w_\alpha = W^\alpha$ e $\overline{S}[\overline{C}(\beta)] = w_\beta = W^\beta$, ao concatená-los formando o ciclo $p_c = \alpha \circ \beta$ e $p_{c'} = \beta \circ \alpha$, as palavras-peso multi-dimensionais resultantes são dadas, respectivamente, por:

$$\overline{S}[\overline{C}(\alpha \circ \beta)] = (w_\alpha + w_\beta); w_\alpha = W_2^{(\alpha\circ\beta)}; W_1^\alpha \tag{3.18}$$

e

$$\overline{S}[\overline{C}(\beta \circ \alpha)] = (w_\beta + w_\alpha); w_\beta = W_2^{(\beta\circ\alpha)}; W_1^\beta \tag{3.19}$$

Concatenando os caminhos $(\alpha \circ \beta)$ e $(\beta \circ \alpha)$:

- n vezes, ou seja, $\overset{n}{\cup}(\alpha \circ \beta)$ e $\overset{n}{\cup}(\beta \circ \alpha)$, tem-se que:

$$\begin{aligned}\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta))] =& n \times w_\alpha + n \times w_\beta; n \times w_\alpha + (n-1) \times w_\beta; \\ & (n-1) \times w_\alpha + (n-1) \times w_\beta; \\ & (n-1) \times w_\alpha + (n-2) \times w_\beta; \ldots; w_\alpha + w_\beta; w_\alpha \Rightarrow \end{aligned} \tag{3.20}$$

$$\begin{aligned}\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta))] =& W_{2.n}^{(n.\alpha \circ n.\beta)}; W_{2.n-1}^{(n.\alpha \circ (n-1).\beta)}; W_{2.n-2}^{((n-1).\alpha \circ (n-1).\beta)}; \ldots; \\ & W_2^{(\alpha\circ\beta)}; W_1^\alpha, \end{aligned} \tag{3.21}$$

$$\begin{aligned}\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha))] =& n \times w_\beta + n \times w_\alpha; n \times w_\beta + (n-1) \times w_\alpha; \\ & (n-1) \times w_\beta + (n-1) \times w_\alpha; \\ & (n-1) \times w_\beta + (n-2) \times w_\alpha; \ldots; w_\beta + w_\alpha; w_\beta \Rightarrow \end{aligned}$$

$$\begin{aligned}\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha))] =& W_{2.n}^{(n.\beta \circ n.\alpha)}; W_{2.n-1}^{(n.\beta \circ (n-1).\alpha)}; W_{2.n-2}^{((2.n-1).\beta \circ (2.n-1).\alpha)}; \ldots; \\ & W_2^{(\beta\circ\alpha)}; W_1^\beta, \end{aligned} \tag{3.22}$$

- $(n-1)$ vezes, ou seja, $\overset{n-1}{\cup}(\alpha \circ \beta)$ e $\overset{n-1}{\cup}(\beta \circ \alpha)$, tem-se que:

$$\begin{aligned}
\overline{S}[\overline{C}(\overset{n-1}{\cup}(\alpha \circ \beta))] =& (n-1) \times w_\alpha + (n-1) \times w_\beta; \\
& (n-1) \times w_\alpha + (n-2) \times w_\beta; \\
& (n-2) \times w_\alpha + (n-2) \times w_\beta; \ldots; w_\alpha + w_\beta; w_\alpha \Rightarrow \\
\overline{S}[\overline{C}(\overset{n-1}{\cup}(\alpha \circ \beta))] =& W_{2.n-2}^{((n-1).\alpha \ \circ \ (n-1).\beta)}; \ldots; W_2^{(\alpha \circ \beta)}; W_1^{\alpha} \text{ e} \qquad (3.23)
\end{aligned}$$

$$\begin{aligned}
\overline{S}[\overline{C}(\overset{n-1}{\cup}(\beta \circ \alpha))] =& (n-1) \times w_\beta + (n-1) \times w_\alpha; \\
& (n-1) \times w_\beta + (n-2) \times w_\alpha; \\
& (n-2) \times w_\beta + (n-2) \times w_\alpha; \ldots; w_\beta + w_\alpha; w_\beta \Rightarrow \\
\overline{S}[\overline{C}(\overset{n-1}{\cup}(\beta \circ \alpha))] =& W_{n-2}^{((n-1).\beta \ \circ \ (n-1).\alpha)}; \ldots; W_2^{(\beta \circ \alpha)}; W_1^{\beta} \qquad (3.24)
\end{aligned}$$

Por construção, tem-se que:

- $W_{2.n-2}^{((n-1).\alpha \ \circ \ (n-1).\beta)} < W_{2.n}^{(n.\alpha \ \circ \ n.\beta)}$, ou seja, $(n-1)\times(w_\alpha+w_\beta) < n\times(w_\alpha+w_\beta)$, se $(w_\alpha + w_\beta) > 0$ e, portanto, $\overset{n-1}{\cup}(\alpha \circ \beta) <_L \overset{n}{\cup}(\alpha \circ \beta)$;
- $W_{2.n-2}^{((n-1).\alpha \ \circ \ (n-1).\beta)} > W_{2.n}^{(n.\alpha \ \circ \ n.\beta)}$, ou seja, $(n-1)\times(w_\alpha+w_\beta) > n\times(w_\alpha+w_\beta)$, se $(w_\alpha + w_\beta) < 0$ e, portanto, $\overset{n-1}{\cup}(\alpha \circ \beta) >_L \overset{n}{\cup}(\alpha \circ \beta)$;
- se $(w_\alpha + w_\beta) = 0$, tem-se que $W_{2.n-2}^{((n-1).\alpha \ \circ \ (n-1).\beta)} = W_{2.n}^{(n.\alpha \ \circ \ n.\beta)} = 0$. Entretanto, $\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta))] \prec_L \overline{S}[\overline{C}(\overset{n-1}{\cup}(\alpha \circ \beta))]$, pois $\overline{S}[\overline{C}(\overset{n-1}{\cup}(\alpha \circ \beta))]$ tem menos letras que $\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta))]$;
- $W_{2.n-2}^{((n-1).\beta \ \circ \ (n-1).\alpha)} < W_{2.n}^{(n.\beta \ \circ \ n.\alpha)}$, ou seja, $(n-1)\times(w_\beta+w_\alpha) < n\times(w_\beta+w_\alpha)$, se $(w_\beta + w_\alpha) > 0$ e, portanto, $\overset{n-1}{\cup}(\beta \circ \alpha) <_L \overset{n}{\cup}(\beta \circ \alpha)$;
- $W_{2.n-2}^{((n-1).\beta \ \circ \ (n-1).\alpha)} > W_{2.n}^{(n.\beta \ \circ \ n.\alpha)}$, ou seja, $(n-1)\times(w_\beta+w_\alpha) < n\times(w_\beta+w_\alpha)$, se $(w_\beta + w_\alpha) < 0$ e, portanto, $\overset{n-1}{\cup}(\beta \circ \alpha) >_L \overset{n}{\cup}(\beta \circ \alpha)$; e
- se $(w_\beta + w_\alpha) = 0$, tem-se que $W_{2.n-2}^{((n-1).\beta \ \circ \ (n-1).\alpha)} = W_{2.n}^{(n.\beta \ \circ \ n.\alpha)} = 0$. Entretanto $\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha))] \prec_L \overline{S}[\overline{C}(\overset{n-1}{\cup}(\beta \circ \alpha))]$, pois $\overline{S}[\overline{C}(\overset{n-1}{\cup}(\beta \circ \alpha))]$ tem menos letras que $\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha))]$.

Como, no exemplo, a relação de ordem léxica $\preceq_L$ adotada é $\geq_L$, tem-se que o ciclo somente será livre, ou seja, $\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta))] \prec_{ML} \overline{S}[\overline{C}(\overset{n-1}{\cup}(\alpha \circ \beta))]$ ou $\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha))] \prec_{ML} \overline{S}[\overline{C}(\overset{n-1}{\cup}(\beta \circ \alpha))]$, se $(w_\beta + w_\alpha) > 0$.

Extrapolando esse raciocínio para as quatro sínteses léxicas propostas, $b = b_\infty$ e $b = \{b_l : l < n-3\}$, as relações de ordenação léxica $\preceq_L$ obtidas entre os caminhos

$\overset{n}{\cup}(\alpha \circ \beta)$ e $\overset{n-1}{\cup}(\alpha \circ \beta)$, em função do critério de otimização adotado, estão listadas na Tabela 3.13.

Assim, dependendo da síntese e da correspondente relação de ordenação adotadas, conforme indicado na Tabela 3.14, é possível verificar se o ciclo é livre ou não.

Tabela 3.13: Relações de ordenação léxica entre os caminhos $\cup^n(\alpha \circ \beta)$ e $\cup^{n-1}(\alpha \circ \beta)$

		Relação entre	
Sínteses	**Relação entre**	$\cup^{\mathbf{n}}(\alpha \circ \beta)$ e $\cup^{\mathbf{n-1}}(\alpha \circ \beta)$	
Léxicas	$\mathbf{w}^\alpha$ e $\mathbf{w}^\beta$	$\mathbf{b} = \mathbf{b}_\infty$	$\mathbf{b} = \mathbf{b_l} : \mathbf{l} < \mathbf{n} - \mathbf{3}$
Aditiva	$w^\alpha + w^\beta = 0$	$\cup^n(\alpha \circ \beta) \prec_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha + w^\beta > 0$	$\cup^n(\alpha \circ \beta) >_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) >_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha + w^\beta < 0$	$\cup^n(\alpha \circ \beta) <_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) <_L \cup^{n-1}(\alpha \circ \beta)$
Multiplicativa	$w^\alpha \times w^\beta = 1$	$\cup^n(\alpha \circ \beta) \prec_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha \times w^\beta > 1$	$\cup^n(\alpha \circ \beta) >_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) >_L \cup^{n-1}(\alpha \circ \beta)$
	$0 < w^\alpha \times w^\beta < 1$	$\cup^n(\alpha \circ \beta) <_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) <_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha \times w^\beta \leq 0$	não definida	não definida
Minimizativa	$w^\alpha = w^\beta$	$\cup^n(\alpha \circ \beta) \prec_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha < w^\beta$	$\cup^n(\alpha \circ \beta) \prec_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha > w^\beta$	$\cup^n(\alpha \circ \beta) <_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$
Maximizativa	$w^\alpha = w^\beta$	$\cup^n(\alpha \circ \beta) \prec_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha < w^\beta$	$\cup^n(\alpha \circ \beta) >_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$
	$w^\alpha > w^\beta$	$\cup^n(\alpha \circ \beta) \prec_L \cup^{n-1}(\alpha \circ \beta)$	$\cup^n(\alpha \circ \beta) \equiv_L \cup^{n-1}(\alpha \circ \beta)$

Ampliando essa análise para um problema do roteamento multi-restritivo com

k restrições, quando o ciclo não é livre por equivalência da primeira restrição na relação léxica entre $\overset{n}{\cup}(\alpha \circ \beta)$ e $\overset{n-1}{\cup}(\alpha \circ \beta)$, então a verificação da liberdade do ciclo deverá ser feita para a segunda restrição. Se constatado que o ciclo não é livre por equivalência da segunda restrição na relação léxica entre $\overset{n}{\cup}(\alpha \circ \beta)$ e $\overset{n-1}{\cup}(\alpha \circ \beta)$, o processo se repete até a última restrição imposta, seguindo o critério de ordenação léxica multi-dimensional adotado.

Tabela 3.14: Análise da liberdade × síntese × ordenação léxica

Sínteses	**Relação entre**	$\preceq_{\mathbf{L}} \Rightarrow \geq_{\mathbf{L}}$		$\preceq_{\mathbf{L}} \Rightarrow \leq_{\mathbf{L}}$	
Léxicas	$\mathbf{w}^{\alpha}$ e $\mathbf{w}^{\beta}$	$\mathbf{b} = \mathbf{b}_{\infty}$	$\mathbf{b} = \mathbf{b}_{\mathbf{l}} : \mathbf{l} < \mathbf{n} - \mathbf{3}$	$\mathbf{b} = \mathbf{b}_{\infty}$	$\mathbf{b} = \mathbf{b}_{\mathbf{l}} : \mathbf{l} < \mathbf{n} - \mathbf{3}$
Aditiva	$w^{\alpha} + w^{\beta} = 0$	livre	não-livre	livre	não livre
	$w^{\alpha} + w^{\beta} > 0$	livre	livre	não livre	não livre
	$w^{\alpha} + w^{\beta} < 0$	não livre	não livre	livre	livre
Multiplicativa	$w^{\alpha} \times w^{\beta} = 1$	livre	não livre	livre	não livre
	$w^{\alpha} \times w^{\beta} > 1$	livre	livre	não livre	não livre
	$0 < w^{\alpha} \times w^{\beta} < 1$	não livre	não livre	livre	livre
	$w^{\alpha} \times w^{\beta} \leq 0$	não definida	não definida	não definida	não definida
Minimizativa	$w^{\alpha} = w^{\beta}$	livre	não livre	livre	não livre
	$w^{\alpha} < w^{\beta}$	livre	não livre	livre	não livre
	$w^{\alpha} > w^{\beta}$	não livre	não livre	livre	não livre
Maximizativa	$w^{\alpha} = w^{\beta}$	livre	não livre	livre	não livre
	$w^{\alpha} < w^{\beta}$	livre	não livre	não livre	não livre
	$w^{\alpha} > w^{\beta}$	livre	não livre	livre	não livre

3.3 Análise de Convergência

O problema da análise de convergência dos algoritmos de roteamento *hop-by-hop* mono/ multi-restritivo pode ser dividido, genericamente, em três sub-problemas: a obtenção de todos os caminhos elegíveis que unem qualquer par de nós origem-destino da rede, a ordenação desses caminhos com base em um critério de otimização e o estabelecimento da coerência desses caminhos entre os nós da rede.

Cabe comentar também que, para os algoritmos de roteamento pela origem, o problema da análise da convergência pode ser dividido, genericamente, em dois sub-problemas: a obtenção de todos os caminhos elegíveis que unem qualquer par de nós origem-destino e a ordenação desses caminhos com base em um critério de otimização.

O problema da ordenação de todos os caminhos elegíveis que unem qualquer par de nós origem-destino em uma rede de telecomunicações, modelada por um dígrafo orientado, ponderado e conectado $N = (\mathbf{V}, \mathbf{E})$, foi abordado, na definição da álgebra de caminhos proposta, pelos critérios de ordenação léxica e ordenação léxica multi-dimensional estabelecidos com base em uma estratégia de otimização, que pode ser espelhada nas definições das sínteses léxicas e das funções de combinação de métricas adotadas.

Esta seção, aborda o problema da obtenção de todos os caminhos elegíveis que unem qualquer par de nós origem-destino e o problema do estabelecimento da coerência desses caminhos entre os nós da rede. Para isso, as propriedades descritas na Seção 3.2 são analisadas em busca de uma garantia de convergência para os algoritmos de roteamento *hop-by-hop*. Os resultados obtidos são comparados com os apresentados por Sobrinho em [35] e Gouda e Scheneider em [23], descritos, respectivamente, nas Seções 2.3.1.2 e 2.3.2.1.

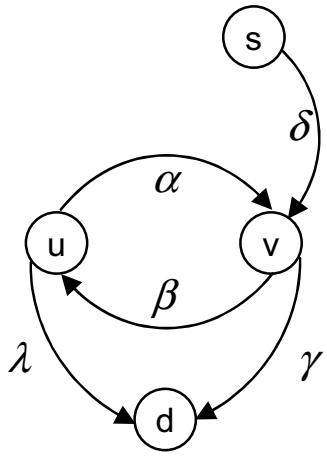


Figura 3.5: Primeiro exemplo de rede para análise de convergência

3.3.1 Análise de Convergência na Obtenção de Todos os Caminhos Elegíveis

Para a obtenção de todos os caminhos elegíveis que unem qualquer par de nós origem-destino, cada nó da rede, como um nó de origem e raiz, tem que ser capaz de construir uma ou mais árvores geradoras, ou seja, árvores compostas por todos os caminhos elegíveis que conectam todos os outros nós alcançáveis da rede, representada por um dígrafo orientado, ponderado e conexo $N = (\mathbf{V}, \mathbf{E})$.

Observe o exemplo de rede mostrado na Figura 3.5. Para a escolha dos melhores caminhos dos nós de origem (u) e (v) para o nó de destino (d), obtêm-se as árvores geradoras ilustradas na Figura 3.6.

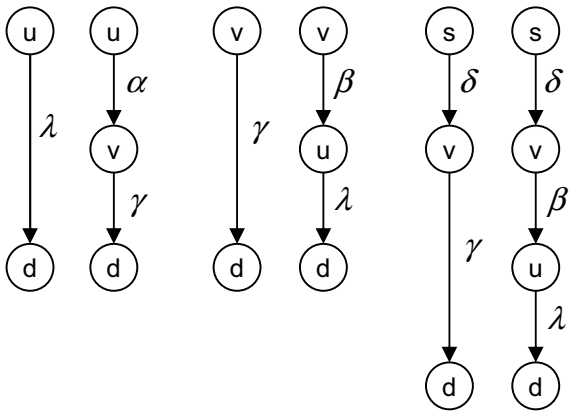


Figura 3.6: Árvores geradoras enraizadas em (u), (v) e (s)

Assim, mesmo havendo um ciclo qualquer no dígrafo, os nós desse ciclo são capazes de obter uma ou mais árvores geradoras, sem a necessidade da imposição de nenhuma propriedade aos caminhos indicados.

Por outro lado, analisando os caminhos do nó de origem (s) para o nó de destino (d), observa-se que eles podem ser escritos como:

$$(\delta \circ \gamma), (\delta \circ \beta \circ \alpha \circ \gamma), (\delta \circ \beta \circ \alpha \circ \beta \circ \alpha \circ \gamma), \ldots, (\delta \circ \overset{n}{\cup}(\beta \circ \alpha) \circ \gamma)$$

ou

$$(\delta \circ \beta \circ \lambda), (\delta \circ \beta \circ \alpha \circ \beta \circ \lambda), (\delta \circ \beta \circ \alpha \circ \beta \circ \alpha \circ \beta \circ \lambda), \ldots, (\delta \circ \beta \circ \overset{n}{\cup}(\alpha \circ \beta) \circ \lambda),$$

em que $\overset{n}{\cup}(p)$ denota n $(n = 1, 2, 3 \ldots)$ concatenações do caminho p.

Assim, para que seja possível obter as árvores geradoras ilustradas na Figura 3.6, é necessário que o ciclo $p_c = \alpha \circ \beta$ seja livre e, portanto:

$$\delta \circ \overset{n}{\cup}(\beta \circ \alpha) \circ \gamma \prec \delta \circ \overset{n-1}{\cup}(\beta \circ \alpha) \circ \gamma \tag{3.25}$$

e

$$\delta \circ \beta \circ \overset{n}{\cup}(\alpha \circ \beta) \circ \lambda \prec \delta \circ \beta \circ \overset{n-1}{\cup}(\alpha \circ \beta) \circ \lambda. \tag{3.26}$$

Substituindo os subcaminhos $p = \delta \circ \overset{n-1}{\cup}(\beta \circ \alpha)$ na equação 3.25 e $g = \delta \circ \beta \circ \overset{n-1}{\cup}(\alpha \circ \beta)$ na equação 3.26, obtêm-se:

$$p \circ (\beta \circ \alpha) \circ \gamma \prec p \circ \gamma \tag{3.27}$$

e

$$g \circ (\alpha \circ \beta) \circ \lambda \prec g \circ \lambda. \tag{3.28}$$

Se a propriedade de monotonicidade estrita é obedecida, então:

$$p \circ (\beta \circ \alpha) \prec p \tag{3.29}$$

e

$$g \circ (\alpha \circ \beta) \prec g. \tag{3.30}$$

Por outro lado, se a propriedade de isotonicidade estrita é obedecida, então, dadas as equações 3.29 e 3.30, têm-se que as relações das equações 3.27 e 3.28 são obedecidas.

Portanto, para que cada nó da rede possa construir uma ou mais árvores geradoras, é necessário que as propriedades de liberdade, monotonicidade e isotonicidade estritas sejam obedecidas.

3.3.2 Análise de Convergência no Estabelecimento da Coerência

Buscando o estabelecimento da coerência entre os nós da rede na escolha dos melhores caminhos para um determinado nó de destino, esta seção verifica a necessidade e a suficiência das propriedades descritas na Seção 3.2 com o objetivo de comparar os resultados obtidos com os disponíveis na literatura.

Assim, sem perda de generalidade e por simplicidade, será adotado como exemplo de comparação o dígrafo ilustrado na Figura 3.7 e um caso de roteamento mono-restritivo, em que a função de combinação de métricas $f(w_i) = c_i = w_i$, a síntese léxica é aditiva com abreviação de uma letra, ou seja, $b_{l=1}(S[c(p_{1,z})]) = w_1 + w_2 + \cdots + w_z$ e a relação de ordenação léxica multi-dimensional $\preceq_{ML}$ é $\geq_{ML}$.

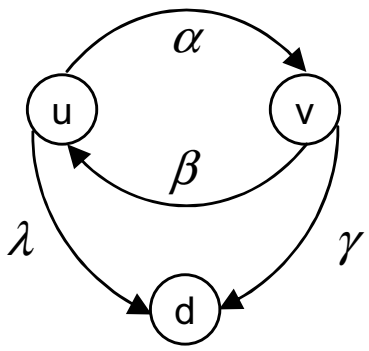


Figura 3.7: Segundo exemplo de rede para análise de convergência

3.3.2.1 Análise de Convergência *versus* Monotonicidade

No exemplo simples indicado na Figura 3.7, na escolha dos melhores caminhos dos nós de origem (u) e (v) para o nó de destino (d), podem ocorrer duas situações: $\big((\lambda \preceq \alpha \circ \gamma)$ e $(\gamma \preceq \beta \circ \lambda)\big)$ ou $\big((\alpha \circ \gamma \preceq \lambda)$ ou $(\beta \circ \lambda \preceq \gamma)\big)$. Essas situações podem ser reescritas como:

$$\overline{S}[\overline{C}(\lambda)] \preceq_{ML} \overline{S}[\overline{C}(\alpha \circ \gamma)] \text{ e } \overline{S}[\overline{C}(\gamma)] \preceq_{ML} \overline{S}[\overline{C}(\beta \circ \lambda)] \tag{3.31}$$

ou

$$\overline{S}[\overline{C}(\alpha \circ \gamma)] \preceq_{ML} \overline{S}[\overline{C}(\lambda)] \text{ ou } \overline{S}[\overline{C}(\beta \circ \lambda)] \preceq_{ML} \overline{S}[\overline{C}(\gamma)]. \tag{3.32}$$

Quando as situações das equações 3.31 ocorrem, a informação enviada com destino ao nó (d) pode permanecer em *loop* no ciclo $p_c = u, v, u$ e, portanto, nesse exemplo, o algoritmo de roteamento *hop-by-hop* não converge.

Por outro lado, quando as situações das equações 3.32 ocorrem, a informação enviada com destino ao nó (d) não é mantida no ciclo $p_c = u, v, u$, pois ao menos um dos nós (u) ou (v), enviará a informação diretamente ao nó d. Portanto, nesse exemplo, para que o algoritmo de roteamento *hop-by-hop* convirja, é necessário que a equação 3.32 seja obedecida.

Além disso, se a propriedade de monotonicidade é obedecida, então $(\alpha \circ \gamma) \preceq \alpha$ e $(\beta \circ \lambda) \preceq \beta$, que podem ser reescritas como:

$$\overline{S}[\overline{C}(\alpha \circ \gamma)] \preceq_{ML} \overline{S}[\overline{C}(\alpha)] \text{ e } \overline{S}[\overline{C}(\beta \circ \lambda)] \preceq_{ML} \overline{S}[\overline{C}(\beta)]. \tag{3.33}$$

Substituindo a função de combinação de métricas, a síntese léxica e a relação de ordenação léxica multi-dimensional $\preceq_{ML}$ adotadas, para que haja:

1. Convergência, tem-se:

$$\sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma) > \sum_{1}^{z'''} c(\lambda) \text{ ou } \sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda) > \sum_{1}^{z''} c(\gamma); \tag{3.34}$$

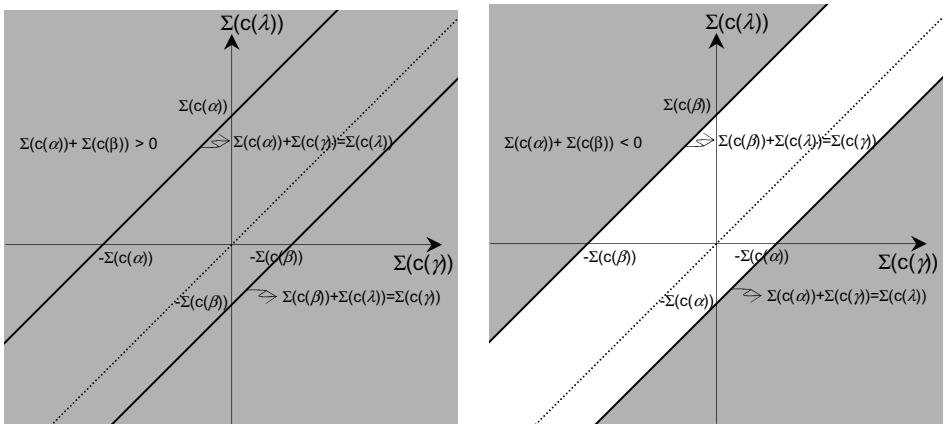
2. Monotonicidade, tem-se:

$$\sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma) > \sum_{1}^{z} c(\alpha) \text{ ou } \sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda) > \sum_{1}^{z'} c(\beta), \tag{3.35}$$

em que:

$$\begin{cases} c(\alpha) = [w_1^\alpha; w_2^\alpha; \cdots; w_z^\alpha] \\ c(\beta) = [w_1^\beta; w_2^\beta; \cdots; w_{z'}^\beta] \\ c(\gamma) = [w_1^\gamma; w_2^\gamma; \cdots; w_{z''}^\gamma] \\ c(\lambda) = [w_1^\lambda; w_2^\lambda; \cdots; w_{z'''}^\lambda] \end{cases} \quad \text{e} \quad \begin{cases} \sum_1^z c(\alpha) = w_1^\alpha + w_2^\alpha + \cdots + w_z^\alpha \\ \sum_1^{z'} c(\beta) = w_1^\beta + w_2^\beta + \cdots + w_{z'}^\beta \\ \sum_1^{z''} c(\gamma) = w_1^\gamma + w_2^\gamma + \cdots + w_{z''}^\gamma \\ \sum_1^{z'''} c(\lambda) = w_1^\lambda + w_2^\lambda + \cdots + w_{z'''}^\lambda \end{cases}.$$

Esboçando as inequações 3.34 em um gráfico, para os valores de $b_1(S[c(\lambda)])$, $b_1(S[c(\gamma)])$, $b_1(S[c(\alpha)])$ e $b_1(S[c(\beta)])$, obtêm-se as regiões de convergência indicadas em cinza nas Figuras 3.8(a) e 3.8(b).



(a) Região de convergência em que: $\sum_1^z c(\alpha) + \sum_1^{z'} c(\beta) > 0$

(b) Região de convergência em que: $\sum_1^z c(\alpha) + \sum_1^{z'} c(\beta) < 0$

Figura 3.8: Exemplo: região de convergência

Fazendo um gráfico a partir das inequações 3.35 para os valores de $b_1(S[c(\lambda)])$, $b_1(S[c(\gamma)])$, $b_1(S[c(\alpha)])$ e $b_1(S[c(\beta)])$, obtém-se a região de monotonicidade indicada em cinza na Figura 3.9.

Nas Figuras 3.10(a) e 3.10(b), é possível observar as regiões em que, embora haja monotonicidade não há convergência e, nas Figuras 3.10(c) e 3.10(d), as regiões em que, embora não haja monotonicidade há convergência.

Cabe comentar também que, para os casos multi-restritivos, a região de convergência é analisada independentemente para cada uma das palavras-peso multi-dimensionais, segundo a relação léxica adotada, e depois é composta conforme a relação de ordenação léxica multi-dimensional.

Portanto, diferentemente do que indicado em [24, 36], verifica-se que na convergência do estabelecimento da coerência na escolha dos melhores caminhos entre

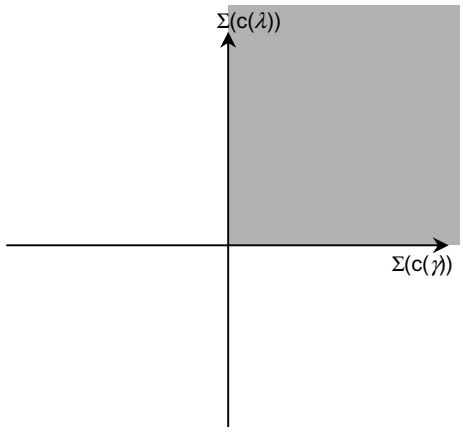


Figura 3.9: Exemplo: região de monotonicidade

os nós da rede no algoritmo de roteamento *hop-by-hop*, a propriedade de monotonicidade não é necessária e nem suficiente.

3.3.2.2 Análise de Convergência *versus* Isotonicidade

No exemplo simples indicado na Figura 3.7, se a propriedade de isotonicidade é obedecida, então $\overset{n}{\cup}(\alpha \circ \beta \circ \alpha) \preceq \alpha \Rightarrow \overset{n}{\cup}(\alpha \circ \beta \circ \alpha) \circ \gamma \preceq \alpha \circ \gamma$ e $\overset{n}{\cup}(\beta \circ \alpha \circ \beta) \preceq \beta \Rightarrow \overset{n}{\cup}(\beta \circ \alpha \circ \beta) \circ \lambda \preceq \beta \circ \lambda$ ou, de outra forma,

$$\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta \circ \alpha))] \preceq_{ML} \overline{S}[\overline{C}(\alpha)] \Rightarrow \overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta \circ \alpha) \circ \gamma)] \preceq_{ML} \overline{S}[\overline{C}(\alpha \circ \gamma)] \tag{3.36}$$

e

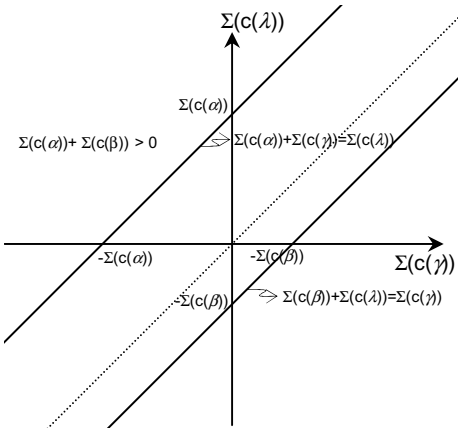
$$\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha \circ \beta))] \preceq_{ML} \overline{S}[\overline{C}(\beta)] \Rightarrow \overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha \circ \beta) \circ \lambda)] \preceq_{ML} \overline{S}[\overline{C}(\beta \circ \lambda)]. \tag{3.37}$$

Substituindo a função de combinação de métricas, a síntese léxica e a relação de ordenação léxica multi-dimensional $\preceq_{ML}$ adotadas, para que haja convergência a equação 3.34 deve ser obedecida e para que haja isotonicidade, tem-se:

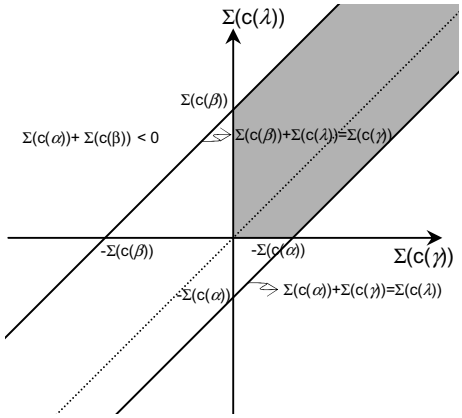
$$\begin{cases} n \times \left(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha)\right) \geq \sum_{1}^{z} c(\alpha) \Rightarrow \\ n \times \left(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha)\right) + \sum_{1}^{z''} c(\gamma) \geq \sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma) \end{cases} \tag{3.38}$$

e

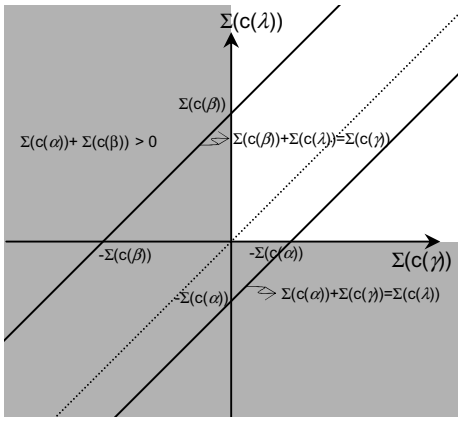
$$\begin{cases} n \times \left(\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)\right) \geq \sum_{1}^{z'} c(\beta) \Rightarrow \\ n \times \left(\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)\right) + \sum_{1}^{z'''} c(\lambda) \geq \sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda) \end{cases} . \tag{3.39}$$



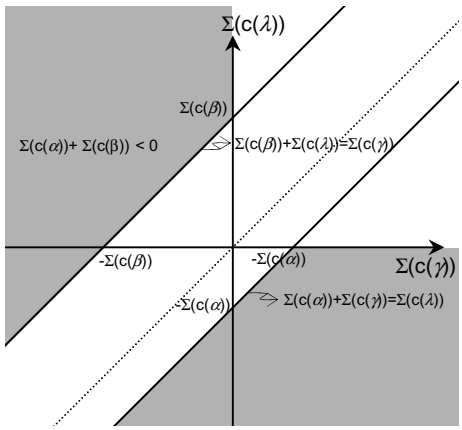
(a) Região de monotonicidade e sem convergência: $\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) > 0$



(b) Região de monotonicidade e sem convergência: $\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) < 0$



(c) Região de convergência e sem monotonicidade: $\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) > 0$



(d) Região de convergência e sem monotonicidade: $\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) < 0$

Figura 3.10: Exemplo: análise de convergência *versus* monotonicidade

Analisando o caso em que $n = 1$, pelas primeiras hipóteses de isotonicidade, têm-se que:

$$\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) \geq \sum_{1}^{z} c(\alpha) \Rightarrow \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) \geq 0 \tag{3.40}$$

e

$$\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) \geq \sum_{1}^{z'} c(\beta) \Rightarrow \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) \geq 0. \tag{3.41}$$

Reescrevendo as segundas hipóteses de isotonicidade,

$$\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma) \geq \sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma) \tag{3.42}$$

e

$$\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda) \geq \sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda), \tag{3.43}$$

como

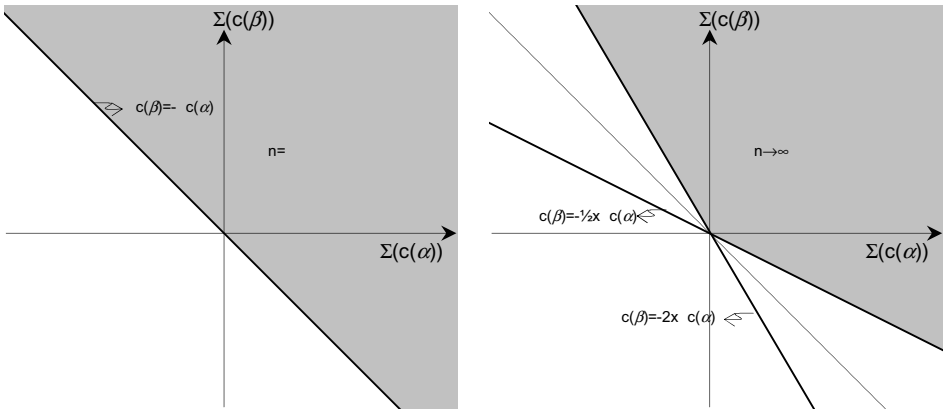
$$\Big(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)\Big) + \Big(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma)\Big) \geq \sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma) \tag{3.44}$$

e

$$\Big(\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha)\Big) + \Big(\sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda)\Big) \geq \sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda), \tag{3.45}$$

uma vez que pelas primeiras hipóteses de isotonicidade tem-se $\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) \geq 0$, então as segundas hipóteses são verdadeiras e, portanto, a região definida pelas primeiras hipóteses garante a isotonicidade, ou seja, é a região na qual a propriedade de isotonicidade é obedecida.

A região de isotonicidade está ilustrada na Figura 3.11(a).



(a) Exemplo: região de isotonicidade para n=1

(b) Exemplo: região de isotonicidade para $n \to \infty$

Figura 3.11: Exemplo: regiões de isotonicidade

Analisando o caso em que $n \to \infty$, pelas primeiras hipóteses de isotonicidade, ou seja, $n \times \big(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha)\big) \geq \sum_{1}^{z} c(\alpha)$ e $n \times \big(\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)\big) \geq$

$\sum_{1}^{z'} c(\beta)$, têm-se que:

$$2 \times n \times \sum_{1}^{z} c(\alpha) + n \times \sum_{1}^{z'} c(\beta) \geq \sum_{1}^{z} c(\alpha) \Rightarrow (2 \times n - 1) \sum_{1}^{z} c(\alpha) + n \times \sum_{1}^{z'} c(\beta) \geq 0 \tag{3.46}$$

e

$$2 \times n \times \sum_{1}^{z'} c(\beta) + n \times \sum_{1}^{z} c(\alpha) \geq \sum_{1}^{z'} c(\beta) \Rightarrow (2 \times n - 1) \sum_{1}^{z'} c(\beta) + n \times \sum_{1}^{z} c(\alpha) \geq 0. \tag{3.47}$$

Reescrevendo as segundas hipóteses de isotonicidade, ou seja,

$$n \times \big(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha)\big) + \sum_{1}^{z''} c(\gamma) \geq \sum_{1}^{z} c(\alpha) + \sum_{1}^{z''} c(\gamma) \tag{3.48}$$

e

$$n \times \big(\sum_{1}^{z'} c(\beta) + \sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)\big) + \sum_{1}^{z'''} c(\lambda) \geq \sum_{1}^{z'} c(\beta) + \sum_{1}^{z'''} c(\lambda), \tag{3.49}$$

como

$$(2 \times n - 1) \sum_{1}^{z} c(\alpha) + n \times \sum_{1}^{z'} c(\beta) + \sum_{1}^{z''} c(\gamma) \geq \sum_{1}^{z''} c(\gamma) \tag{3.50}$$

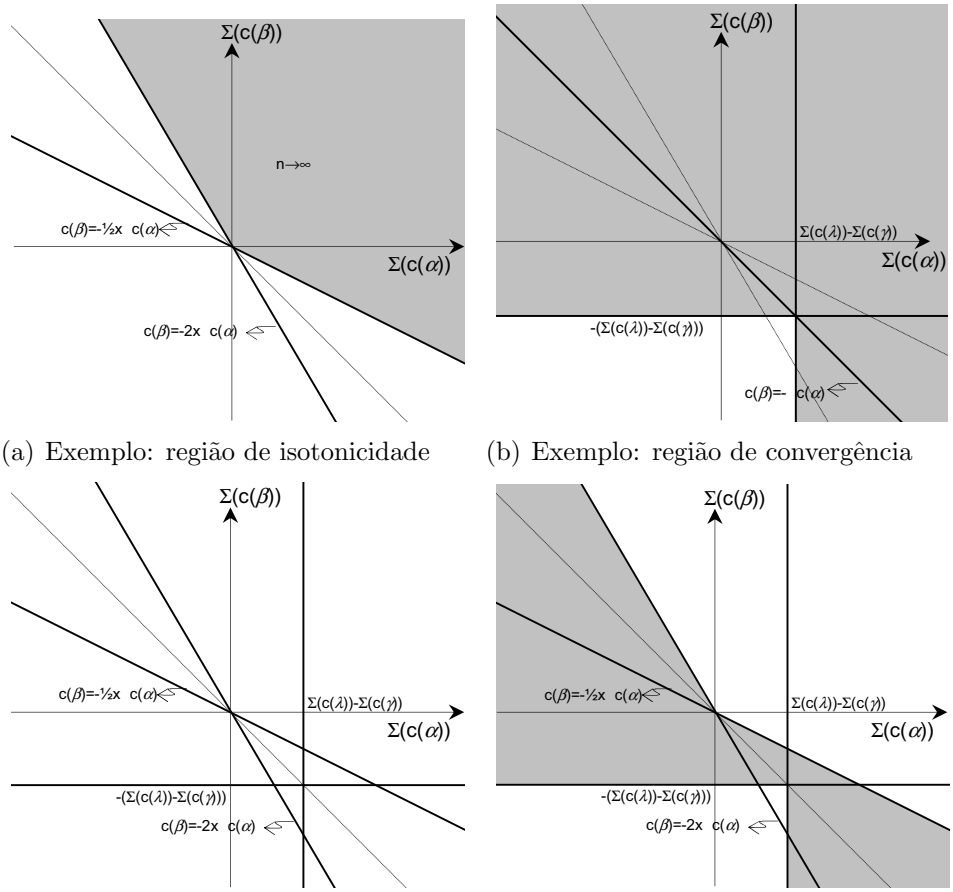
e

$$(2 \times n - 1) \sum_{1}^{z'} c(\beta) + n \times \sum_{1}^{z} c(\alpha) + \sum_{1}^{z'''} c(\lambda) \geq \sum_{1}^{z'''} c(\lambda), \tag{3.51}$$

uma vez que pelas primeiras hipóteses têm-se, respectivamente, que $(2 \times n - 1) \times \sum_{1}^{z} c(\alpha) + n \times \sum_{1}^{z'} c(\beta) \geq 0$ e $(2 \times n - 1) \times \sum_{1}^{z'} c(\beta) + n \times \sum_{1}^{z} c(\alpha) \geq 0$, então as segundas hipóteses são verdadeiras e, portanto, a região definida pelas primeiras hipóteses garante a isotonicidade, ou seja, é a região na qual a propriedade de isotonicidade é obedecida. Portanto, para $n \to \infty$, a região de isotonicidade está ilustrada na Figura 3.11(b).

Comparando os gráficos das Figuras 3.11(a) e 3.11(b), verifica-se que a região de isotonicidade para $n \to \infty$ está contida na região de isotonicidade para $n = 1$. Assim, para a comparação entre as regiões de isotonicidade e de convergência dada pela equação 3.34, em função de $\sum_{1}^{z} c(\alpha)$ e $\sum_{1}^{z'} c(\beta)$, ilustrada na Figura 3.12(b), será utilizada a região de isotonicidade dada por $n \to \infty$.

Na Figura 3.12(c) é possível observar que não existe uma região com isotonicidade e sem convergência. Por outro lado, na Figura 3.12(d), é possível observar as regiões em que não há isotonicidade, mas há convergência.



(a) Exemplo: região de isotonicidade

(b) Exemplo: região de convergência

(c) Região de isotonicidade e sem convergência

(d) Região de convergência e sem isotonicidade

Figura 3.12: Exemplo: análise de convergência *versus* isotonicidade

Observe que nesse exemplo, a região em que os valores da métrica-combinada são sempre positivos está sempre contida na região de convergência e na região de isotonicidade. Entretanto, a região em que o ciclo $p_c = \alpha \circ \beta$ é de distância positiva, ou seja, $\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) > 0$ só está contido na região de isotonicidade, quando $n = 1$.

Portanto, verifica-se, nesse exemplo, que para ocorrer convergência no estabelecimento da coerência dos melhores caminhos entre os nós da rede no algoritmo de roteamento *hop-by-hop*, a propriedade de isotonicidade é suficiente, mas não necessária.

3.3.2.3 Análise de Convergência *versus* Liberdade

No exemplo indicado na Figura 3.7, se a propriedade de liberdade é obedecida pelo ciclo, $\overset{n}{\cup}(\alpha \circ \beta) \prec \overset{n-1}{\cup}(\alpha \circ \beta)$ ou $\overset{n}{\cup}(\beta \circ \alpha) \prec \overset{n-1}{\cup}(\beta \circ \alpha)$, ou seja,

$$\overline{S}[\overline{C}(\overset{n}{\cup}(\alpha \circ \beta))] \prec_{ML} \overline{S}[\overline{C}(\overset{n-1}{\cup}(\alpha \circ \beta))] \tag{3.52}$$

ou

$$\overline{S}[\overline{C}(\overset{n}{\cup}(\beta \circ \alpha))] \prec_{ML} \overline{S}[\overline{C}(\overset{n-1}{\cup}(\beta \circ \alpha))]. \tag{3.53}$$

Substituindo a função de combinação de métricas, a síntese léxica e a relação de ordenação léxica multi-dimensional $\preceq_{ML}$ adotadas, para que haja convergência a equação 3.34 deve ser obedecida e para que haja liberdade, tem-se:

$$n \times \big(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)\big) > (n-1) \times \big(\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)\big). \tag{3.54}$$

A equação 3.54 pode ser reduzida à equação 3.55, já comentada na Seção 3.3.2.2.

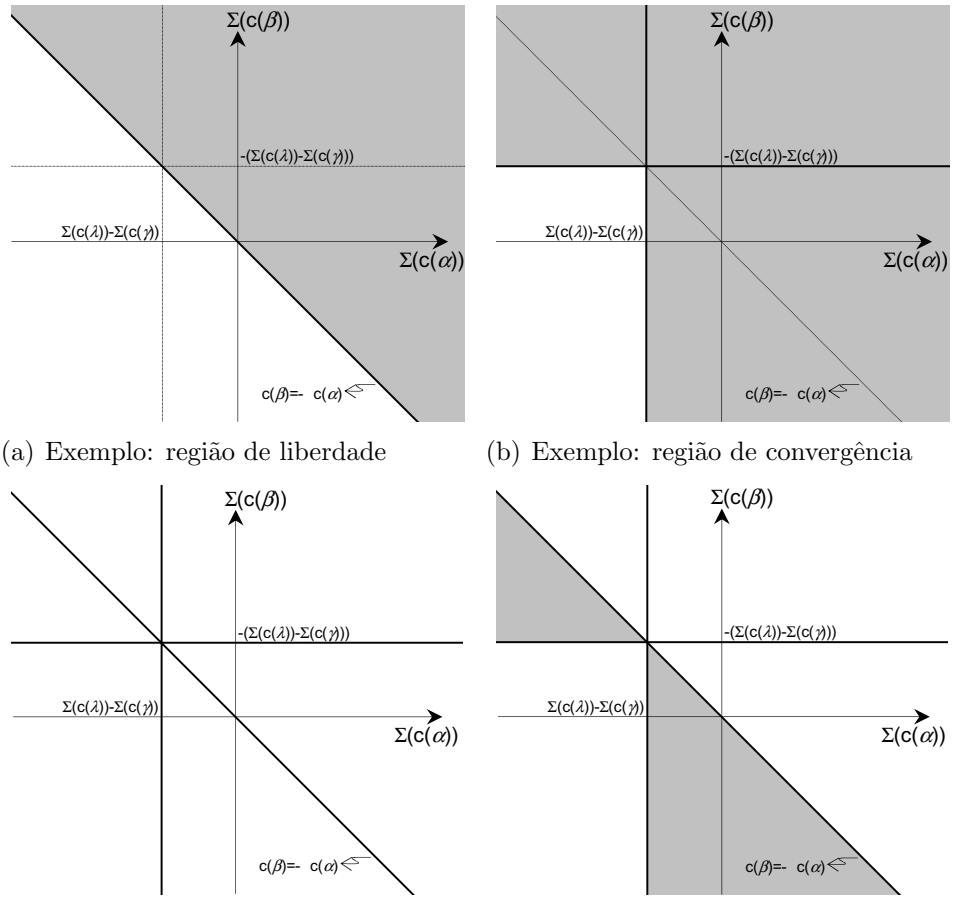
$$\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta) > 0 \tag{3.55}$$

Na Figura 3.13(c) é possível observar que não existe uma região com liberdade e sem convergência. Por outro lado, na Figura 3.13(d), é possível observar as regiões em que não há liberdade, mas há convergência.

Note que quando os valores da métrica-combinada são sempre positivos, cai-se em um caso conhecido, abordado pelo protocolo de roteamento LS, que aplica o algoritmo de Dijkstra na busca de caminhos mais curtos e, portanto, converge sempre que a métrica de roteamento adotada assume somente valores positivos.

Por outro lado, quando a métrica-combinada pode assumir valores tanto positivos como negativos, cai-se em um caso conhecido, abordado pelo protocolo de roteamento DV, que aplica o algoritmo de Bellman Ford na busca de caminhos mais curtos e, portanto, converge sempre que os ciclos existentes sejam de distâncias positivas. Note que $\sum_{1}^{z} c(\alpha) + \sum_{1}^{z'} c(\beta)$ representa a distância do ciclo $p_c = \alpha \circ \beta$.

Portanto, verifica-se nesse exemplo que para ocorrer convergência no estabelecimento da coerência dos melhores caminhos entre os nós da rede no algoritmo de roteamento *hop-by-hop*, a propriedade de liberdade é suficiente, mas não necessária.



(a) Exemplo: região de liberdade

(b) Exemplo: região de convergência

(c) Região de liberdade e sem convergência

(d) Região de convergência e sem liberdade

Figura 3.13: Exemplo: análise de convergência *versus* liberdade

3.4 Propriedade de Coerência

Com o objetivo de traduzir para a álgebra de caminhos uma propriedade que modele a convergência de algoritmos de roteamento *hop-by-hop*, em outras palavras, que seja necessária e suficiente para garantir a coerência entre os nós da rede na definição do melhor caminho para um nó de destino, conforme descrito na Seção 2.2.4, este trabalho define a propriedade coerência, introduzida a seguir.

Considere como exemplo a rede ilustrada na Figura 3.14, em que existe um ciclo dado por $p_c = u_c, v_c, z_c, y_c, \ldots, x_c, u_c$.

Define-se como condição de *loop* se os nós do ciclo p_c, como nós de origem, escolhem, como melhor caminho para o nó de destino (d), os caminhos nos quais a definição do "próximo nó" para todos os nós do ciclo é um nó que também

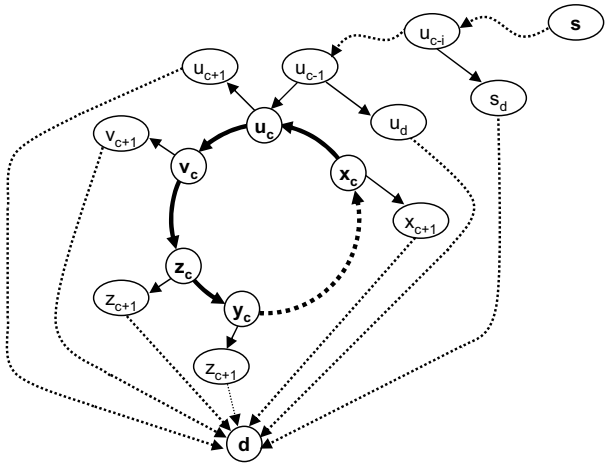


Figura 3.14: Análise da propriedade de coerência em uma rede genérica

pertence ao ciclo, conforme indicado a seguir:

$$\begin{cases} p_{\mathbf{u_c},d} = \mathbf{u_c}, \mathbf{v_c}, v_{c+1}, \ldots, d \\ p_{\mathbf{v_c},d} = \mathbf{v_c}, \mathbf{z_c}, z_{c+1}, \ldots, d \\ p_{\mathbf{z_c},d} = \mathbf{z_c}, \mathbf{y_c}, y_{c+1}, \ldots, d \\ \vdots \\ p_{\mathbf{x_c},d} = \mathbf{x_c}, \mathbf{u_c}, u_{c+1}, \ldots, d \end{cases} . \tag{3.56}$$

Note que, embora os nós do ciclo tenham encontrado, respectivamente, um caminho para o nó de destino (d) e, portanto, este seja um ciclo livre, conforme indicado em [36], como no protocolo de roteamento *hop-by-hop* a decisão do "próximo nó" para onde a informação deve ser encaminhada é tomada localmente, ou seja, nó-a-nó, nesse exemplo, quando a informação é encaminhada por qualquer nó do ciclo ao nó (d), ela permanece em *loop* no ciclo, sem jamais conseguir atingir seu nó de destino.

Observe agora um nó qualquer (s) fora do ciclo, em que os caminhos escolhidos pelos nós intermediários até o nó de destino (d) são dados por:

$$\begin{cases} p_{s,d} = s, \ldots, \mathbf{u_{c-i}}, s_d, \ldots, d \\ \vdots \\ p_{\mathbf{u_{c-i}},d} = \mathbf{u_{c-i}}, \ldots, \mathbf{u_{c-1}}, u_d, \ldots, d \\ \vdots \\ p_{\mathbf{u_{c-1}},d} = \mathbf{u_{c-1}}, \mathbf{u_c}, u_{c+1}, \ldots, d \end{cases} . \tag{3.57}$$

Embora o nó (s) tenha escolhido um melhor caminho para o nó (d) que não

passa por nenhum nó da condição de *loop*, devido à não coerência na escolha do melhor caminho feita pelos nós intermediários $(u_{c-i}), \ldots, (u_{c-1})$, que não pertencem ao ciclo, a informação, enviada pelo nó de origem (s) ao destino (d), atinge o nó do ciclo (u_c) em condição de *loop* e, portanto, também jamais sairá do ciclo.

Vale enfatizar que para um determinado nó de destino, a escolha do "próximo nó" feita por um nó de origem é baseada na sua definição do melhor caminho para o nó de destino (d) e, portanto, no critério de otimização adotado pelo algoritmo de roteamento multi-restritivo *hop-by-hop*.

Assim, dado qualquer ciclo $p_c = u_c, v_c, z_c, y_c, \ldots, x_c, u_c$ em um dígrafo $N(\mathbf{V}, \mathbf{E})$, traduz-se a condição de *loop* para a álgebra de caminhos proposta, quando ocorre:

$$\begin{cases} \overline{S}[\overline{C}(\lceil(\mathbf{P}_{(\mathbf{u_c,d})})] = \Theta(\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P}_{(\mathbf{u_c,d})})]) \\ \overline{S}[\overline{C}(\lceil(\mathbf{P}_{(\mathbf{v_c,d})}))] = \Theta(\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P}_{(\mathbf{v_c,d})})]) \\ \overline{S}[\overline{C}(\lceil(\mathbf{P}_{(\mathbf{z_c,d})})] = \Theta(\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P}_{(\mathbf{z_c,d})})]) \\ \vdots \\ \overline{S}[\overline{C}(\lceil(\mathbf{P}_{(\mathbf{x_c,d})}))] = \Theta(\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P}_{(\mathbf{x_c,d})})]) \end{cases} , \tag{3.58}$$

em que:

- $\mathbf{P_{u_c,d}}$, $\mathbf{P_{v_c,d}}$, $\mathbf{P_{z_c,d}}$, $\ldots$, $\mathbf{P_{x_c,d}}$: são os conjuntos de todos os caminhos elegíveis que conectam os respectivos nós do ciclo ao nó de destino (d);
- $\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{u_c,d}})]$, $\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{v_c,d}})]$, $\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{z_c,d}})]$, $\ldots$, $\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{x_c,d}})]$: são os conjuntos dos respectivos valores das palavras-peso multi-dimensionais desses caminhos; e
- os elementos genéricos $\lceil(\mathbf{P_{s,d}})$ e $\Theta(\overline{\mathbf{S}}[\overline{\mathbf{C}}(\mathbf{P_{s,d}})])$: são definidos conforme indicado na Seção 3.1.4.

Em geral, todos os caminhos que conectam quaisquer pares de nós origem (s) e destino (d), ou seja, todos os elementos do conjunto $\mathbf{P_{s,d}}$, podem ser classificados como:

- **caminho direto** $p(s,d)_{dir}$, isto é, qualquer caminho cujo:
  - nó de origem não pertença à condição de *loop*,
  - nó de origem não escolhe como "próximo nó", um nó que pertença à condição de *loop*, e

  - nó de origem não escolhe como "próximo nó", um nó que via escolha recursiva dos "próximos nós", pelos "próximos nós", não se conecte a um nó que pertença à condição de *loop*;

- **caminho indireto** $p(s,d)_{ind}$, isto é, qualquer caminho que não seja direto, ou seja, qualquer caminho cujo:
  - nó de origem pertença à condição de *loop*,
  - nó de origem escolhe como "próximo nó", um nó que pertença à condição de *loop*, ou
  - nó de origem escolhe como "próximo nó', um nó que via escolha recursiva dos "próximos nós", pelos "próximos nós", se conecte a um nó que pertença à condição de *loop*.

Dessa forma, este trabalho define uma nova propriedade denominada coerência como condição necessária e suficiente para garantir a convergência de um algoritmo de roteamento *hop-by-hop*.

Propriedade 3.3. Coerência é a existência de pelo menos um caminho direto entre quaisquer pares de nós origem-destino em uma rede $N = (\mathbf{V}, \mathbf{E})$, ou seja, $\mathbf{P}_{\mathbf{(s,d)_{dir}}} \neq \emptyset$ para $\forall\ (s,d)\ \in \mathbf{V}$.

Teorema 3.1. Para um algoritmo de roteamento *hop-by-hop* convergir, é necessário e suficiente que a propriedade de coerência seja obedecida.

***Demonstração da condição de necessidade*:** Por contradição, suponha que não haja nenhum caminho direto entre um par de nós origem-destino qualquer (s) e (d) em $\mathbf{V}$. Nesse caso, pode ocorrer duas situações: o nó de destino (d) não é acessível por (s), ou o nó de destino (d) é acessível por (s) por meio de um caminho indireto.

Se o nó de destino (d) não é acessível por (s), ou seja, não há nenhum caminho entre (s) e (d), então o algoritmo de roteamento *hop-by-hop* não se aplica e, portanto, não se aplica a necessidade da propriedade de coerência.

Por outro lado, se o nó de destino (d) é acessível por (s) por meio de um caminho indireto, então, pela própria definição do caminho indireto, a rede apresenta algum ciclo em condição de *loop* e, portanto, o algoritmo de roteamento *hop-by-hop* não converge.

Assim, como se queria demonstrar, para que o algoritmo de roteamento *hop-by-hop* convirja é necessário que a propriedade de coerência seja obedecida. ■

***Demonstração da condição de suficiência*:** Se a propriedade de coerência é obedecida e, portanto, em uma rede há pelo menos um caminho direto entre quaisquer pares de nós origem-destino, pela própria definição do caminho direto, então não pode haver nenhum nó da rede que seja um nó que pertença a algum ciclo em condição de *loop*. Sendo assim, a rede não pode ter nenhum ciclo em condição de *loop* e, conseqüentemente, o algoritmo de roteamento *hop-by-hop* converge.

Por outro lado, se a rede não apresenta nenhum ciclo em condição de *loop*, então qualquer nó (d) da rede, como nó de destino, pode construir uma árvore geradora de chegada enraizada nele a partir da informação do "próximo nó" escolhido por todos os outros nós da rede diferente de (d). Observe a Figura 3.14. Quando há condição de *loop*, o dígrafo obtido pelo nó de destino (d), em função das informações do "próximo nó" escolhido por todos os outros nós da rede, apresenta um ciclo que representa a condição de *loop*.

Assim, para completar a prova da condição de suficiência da propriedade de coerência na garantia da convergência dos algoritmos de roteamento *hop-by-hop*, basta provar que para qualquer nó da rede existe uma árvore geradora de chegada enraizada nele e gerada a partir das informações do "próximo nó" escolhido por todos os outros nós da rede diferente de (d), ou seja, que para qualquer nó (s_{ci}) pertencente a um ciclo, o caminho $\lceil(p_{(s_{ci},d)})$ não é um elemento único em $\mathbf{P_{s_{ci},d}}$ e, portanto, se (d) é acessível a partir de qualquer outro nó de origem da rede, sempre existe um caminho direto no conjunto de caminhos elegíveis entre esse nó de origem e o nó de destino (d).

Considere um pior caso, em que os $n-1$ nós da rede, representada por um dígrafo $N(\mathbf{V},\mathbf{E})$ de ordem n, pertence a um ciclo em condição de *loop* e, portanto, o nó de destino (d) não pertence ao ciclo, conforme representado na Figura 3.15.

Assim, se para qualquer $\mathbf{P_{s_{ci},d}}$ o caminho $\lceil(p_{(s_{ci},d)})$ é único, então o vértice (d) não é acessível por nenhum outro nó do dígrafo e, portanto, o dígrafo $N(\mathbf{V},\mathbf{E})$ não é um dígrafo fortemente conectado.

Entretanto, é razoável admitir que em uma rede de telecomunicação, em que o algoritmo de roteamento *hop-by-hop* é aplicável, o dígrafo que a modela é um dígrafo fortemente conectado, ou seja, para todos os nós da rede, tomando-se qualquer combinação dois-a-dois, o par de nós é mutuamente acessível. Portanto, deverá haver um nó (s_{cd}) qualquer cujo conjunto $\mathbf{P_{s_{cd},d}}$ é composto por pelo menos dois elementos $\{\lceil(p_{(s_{cd},d)}),(s_{cd},d))\}$, em que (s_{cd},d) é uma aresta que sai do nó (s_{cd}) e chega em (d).

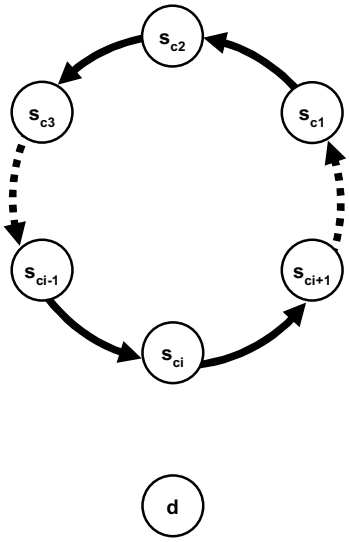


Figura 3.15: Prova da garantia de convergência: primeira hipótese

Assim, se o nó (s_{cd}) passar a escolher como "próximo nó" o nó (d), tornando o conjunto dos caminhos elegíveis entre (s_{cd}) e (d) dado por $\mathbf{P_{s_{cd},d}} = \{(s_{cd}, d))\}$, em que $\lceil(p_{(s_{cd},d)}) = s_{cd}, d$ é o novo valor escolhido, haverá também um caminho direto entre todos os nós (s_{ci}) ao destino (d). Isto porque, qualquer outro nó do ciclo, pela condição de *loop* dada pela escolha inicial dos "próximos nós", ao enviar uma informação ao nó de destino (d), essa informação necessariamente chega ao nó (s_{cd}) e, via (s_{cd}), a (d). Portanto, existe uma árvore geradora de chegada enraizada em (d).

Considere um caso um pouco mais genérico, em que os $n-1$ nós da rede, representada por um dígrafo $N(\mathbf{V}, \mathbf{E})$ de ordem n, pertençam a l ciclos em condição de *loop*, conforme representado na Figura 3.16.

Assim, se para qualquer $\mathbf{P_{s_{i,ci},d}}$ o caminho $\lceil(p_{(s_{i,ci},d)})$ é único, então o vértice (d) não é acessível por nenhum outro nó do dígrafo e, portanto, o dígrafo $N(\mathbf{V}, \mathbf{E})$ não é um dígrafo fortemente conectado.

Entretanto, como já comentado anteriormente, é razoável admitir que o dígrafo em questão é fortemente conectado e, portanto, deverá haver pelo menos um nó $(s_{ld,cd})$ em pelo menos um ciclo ld qualquer, cujo conjunto $\mathbf{P_{s_{ld,cd},d}}$ é composto por pelo menos dois elementos $\{\lceil(p_{(s_{ld,cd},d)}), (s_{ld,cd}, d))\}$, em que $(s_{ld,cd}, d)$ é uma aresta que sai do nó $(s_{ld,cd})$ e chega em (d).

Assim, se o nó $(s_{ld,cd})$ passar a escolher como "próximo nó" o nó (d), tornando o conjunto dos caminhos elegíveis entre $(s_{ld,cd})$ e (d) dado por $\mathbf{P_{s_{ld,cd},d}} = \{(s_{ld,cd}, d))\}$, em que $\lceil(p_{(s_{ld,cd}}, d)) = s_{ld,cd}, d$ é o novo valor escolhido, haverá também um caminho direto entre todos os nós $(s_{ld,ci})$ do ciclo ld ao destino (d),

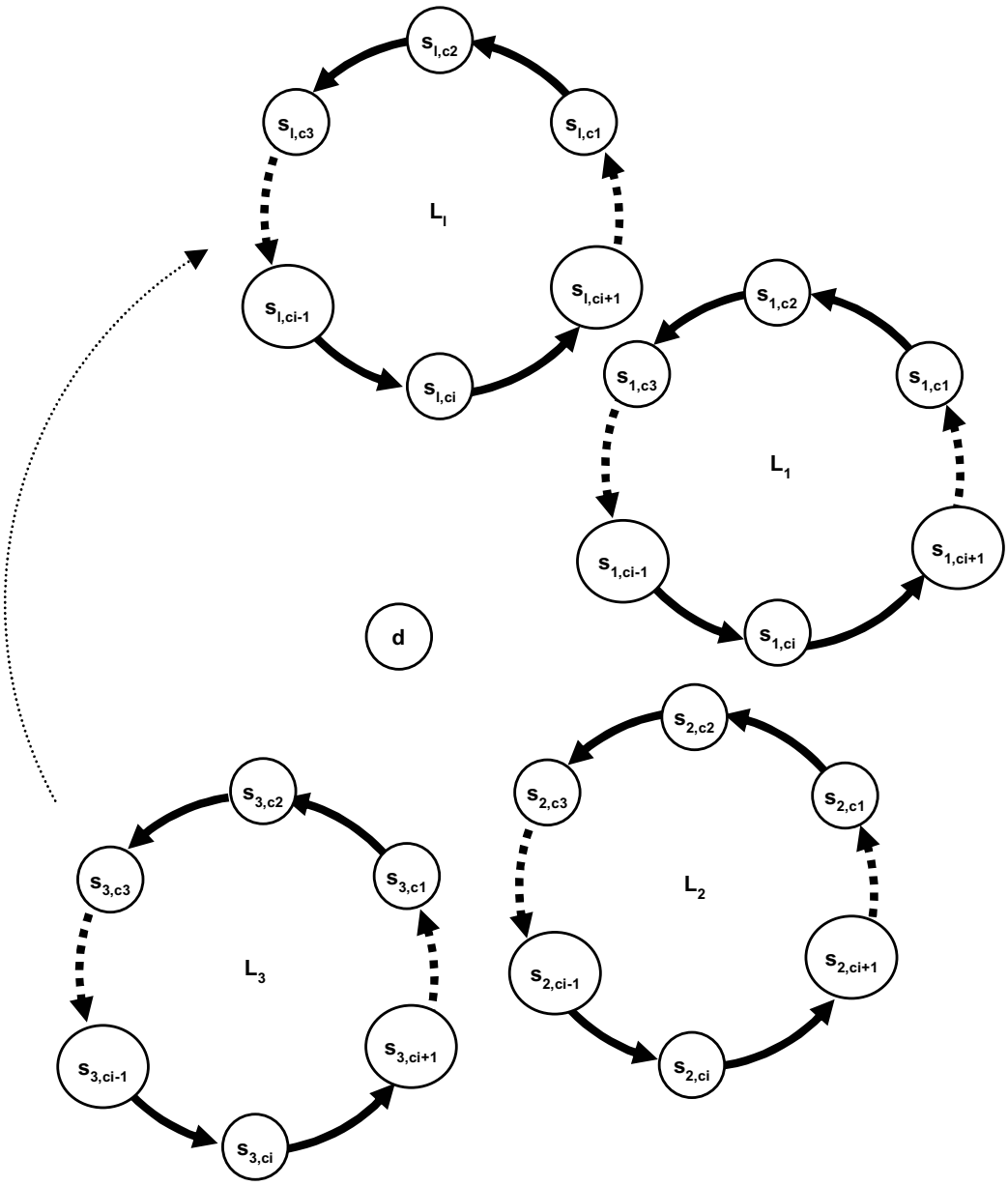


Figura 3.16: Prova da garantia de convergência: segunda hipótese

pois para o ciclo ld a condição inicial de *loop* é quebrada.

Por outro lado, como o dígrafo $N(\mathbf{V}, \mathbf{E})$ é fortemente conectado, para cada um dos demais $l-1$ ciclos da rede diferentes de ld, deverá haver um nó $(s_{l,cl})$ com uma aresta que sai do nó $(s_{l,cl})$ e, ou chega no nó (d), ou que chega em um nó $(s_{l',cl'})$ que pertence a um outro ciclo qualquer.

Para os ciclos em que a aresta existente é do tipo $(s_{l,cl}, d)$, se o nó $(s_{l,cl})$ passar a escolher como "próximo nó" o nó (d), a condição inicial de *loop* do ciclo l é quebrada.

Para os ciclos em que a aresta existente é do tipo $(s_{l,cl}, s_{l',cl'})$, como o dígrafo $N(\mathbf{V}, \mathbf{E})$ é fortemente conectado, então deverá haver, no conjunto de todas as

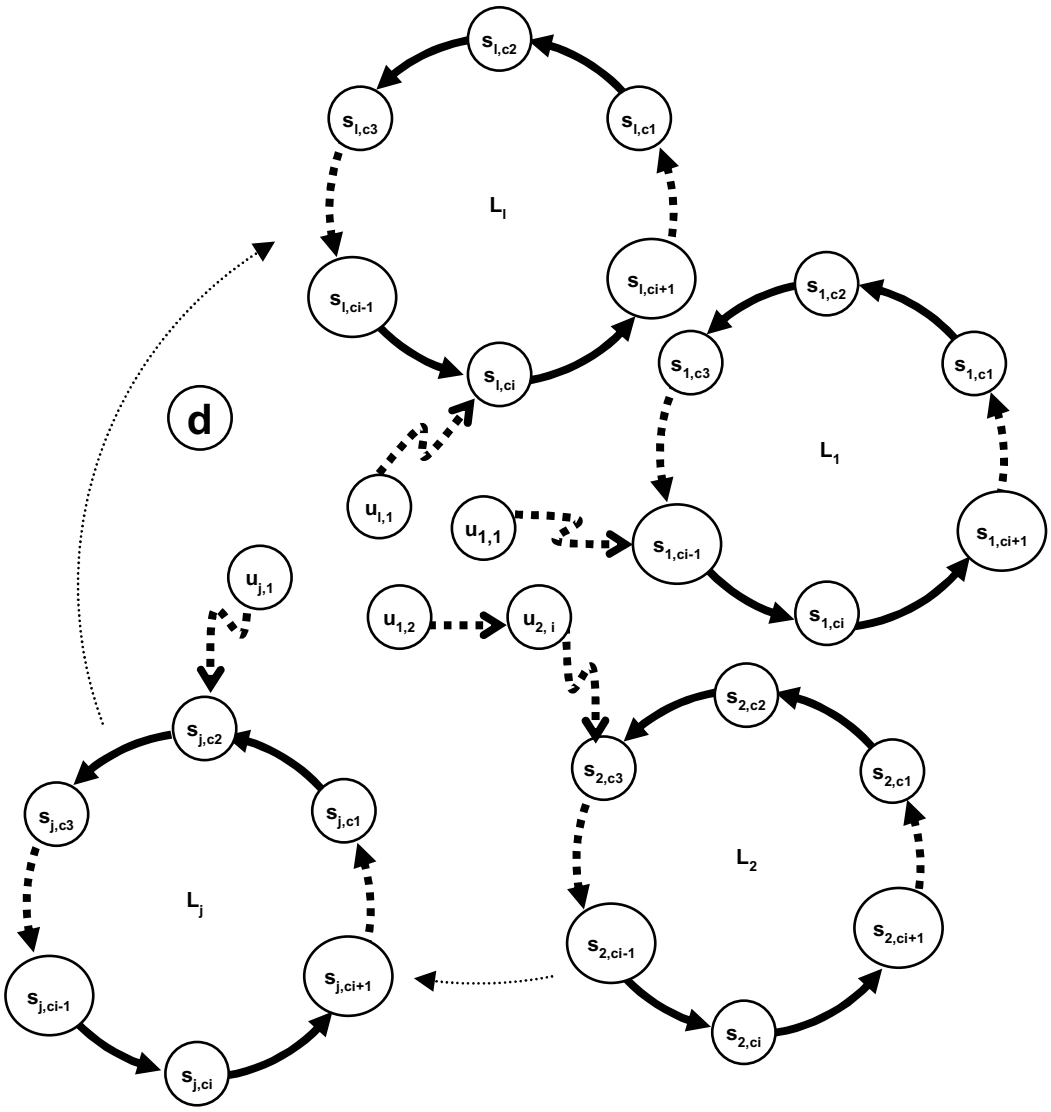


Figura 3.17: Prova da garantia de convergência: terceira hipótese

arestas $(s_{l,cl}, s_{l',cl'})$ existentes, pelo menos uma que se conecte a um ciclo ld qualquer, que tenha um nó $(s_{ld,cl})$ com uma aresta $(s_{ld,cl}, d)$. Assim, se os nós dos ciclos que se conectam aos outros ciclos, passarem a escolher como 'próximo nó" o nó do outro ciclo, como um desses ciclos teve sua condição inicial de *loop* quebrada, então para qualquer ciclo l, a condição inicial de *loop* é quebrada e, portanto, existe uma árvore geradora de chegada enraizada em (d).

Considere por fim, um caso ainda mais genérico, em que os $n-1$ nós da rede, representada por um dígrafo $N(\mathbf{V}, \mathbf{E})$ de ordem n, ou pertençam a um dos l ciclos em condição de *loop*, ou , via escolha recursiva dos "próximos nós", pelos "próximos nós", se conectam a um nó que pertença a um dos l ciclos em condição de *loop*, conforme representado na Figura 3.17.

Como o dígrafo $N(\mathbf{V}, \mathbf{E})$ é fortemente conectado, há pelo menos um conjunto de arestas que, quando escolhido pelos respectivos nós de saída, permite que todos

os nós que pertençam aos l ciclos em condição inicial de *loop* e todos os nós que se conectam a um nó que pertença a um dos l ciclos em condição inicial de *loop* estejam diretamente conectados a (d).

Como se pode encaixar, em um desses três casos genéricos, qualquer dígrafo em condição inicial de *loop*, obtida pelo nó de destino a partir da informação de "próximo nó" de todos os demais nós da rede, então, como se queria demonstrar, a propriedade de coerência é suficiente para garantir a convergência de um algoritmo de roteamento *hop-by-hop*. ■

Observe que, no exemplo utilizado na Seção 3.3.2 e indicado na Figura 3.7, as regiões de coerência são dadas diretamente pela definição das regiões de convergência, pois $\big((\lambda \preceq \alpha \circ \gamma)$ e $(\gamma \preceq \beta \circ \lambda)\big)$ ou $\big((\alpha \circ \gamma \preceq \lambda)$ ou $(\beta \circ \lambda \preceq \gamma)\big)$ definem as regiões em que existe pelo menos um caminho direto entre qualquer nó da rede ao nó de destino (d).

Este trabalho define que um algoritmo de roteamento *hop-by-hop* converge de maneira ótima, se a árvore geradora de chegada de qualquer nó (d) da rede é construída somente por arestas (s_i, s_j), que pertençam ao caminho $p_{s_i,d} = s_i, s_j, \ldots, d = \lceil(\mathbf{P_{s_i,d}})$, escolhido inicialmente por $\forall\ (s_i) \neq (d)$, ou em outras palavras, se os caminhos que constituem a árvore geradora de chegada de qualquer nó (d) da rede, coincidem com os caminhos obtidos pelas árvores geradoras de saída de qualquer nó (s) da rede.

Por outro lado, indo além da definição apresentada na literatura [74, 34, 23, 19, 80, 83, 84, 85] do melhor caminho para todos e quaisquer pares de nós origem-destino de uma rede que implementa um algoritmo de roteamento mono/ multi-restritivo *hop-by-hop*, e expandindo um pouco mais a discussão da definição do melhor caminho, nota-se que este pode ser definido tanto pelo nó de origem quanto pelo nó de destino.

Se o melhor caminho é definido pelo nó de origem, como se verifica nos algoritmos de roteamento mono/ multi-restritivo *hop-by-hop* conhecidos, então, em uma rede inicialmente com ciclos em condição de *loop*, através da imposição da propriedade de coerência verifica-se a eliminação dos *loops*. Note que, na imposição da propriedade de coerência, o processo de escolha do conjunto de nós, cujas arestas de saída permitem que todos os nós da rede escolham um caminho direto a qualquer nó de destino (d), não faz parte do escopo da álgebra de caminhos proposta. Entretanto, este trabalho propõe uma heurística de escolha baseada na identificação, pelos nós de destino (d), de todos os nós que escolheram um caminho indireto e na seleção, pelos nós de destino, de qual desses nós

deve alterar sua escolha de "próximo nó" para que se imponha a propriedade de coerência, indicando inclusive, com base nas opções de "próximo nó" fornecida pelo próprio nó escolhido, qual deveria ser sua opção de "próximo nó". Essa heurística é implementada e detalhada no algoritmo ELND, descrito na Seção 3.5.

Por outro lado, se o melhor caminho é definido pelo nó de destino, em uma estratégia em que o nó de destino informa ao nó de origem qual deve ser sua opção de "próximo nó", então a propriedade de coerência no algoritmo de roteamento *hop-by-hop* é implementada automaticamente.

Cabe ressaltar que essa conclusão é nova e a sua implementação garante a convergência de algoritmos de roteamento *hop-by-hop* mono/ multi-restritivo com qualquer estratégia de otimização. Por exemplo, em um algoritmo de roteamento *hop-by-hop* multi-restritivo, em que as métricas adotadas representam parâmetros de QoS. Para garantir a implementação da estratégia de QoS definida para a rede, basta que qualquer nó da rede, como nó de destino (d), construa, a partir da sua definição de melhor caminho, uma árvore geradora de chegada e notifique aos demais nós da rede, como nós de origem, a informação de "próximo nó", que eles devem adotar, para chegar até o nó de destino (d). Em outras palavras, a informação de "próximo nó", para qualquer nó da rede a um nó de destino, é imposta pelo nó de destino. Assim, somente neste caso, pois a propriedade de coerência já está garantida, para um algoritmo de roteamento mono/ multi-restritivo *hop-by-hop* convergir, basta que as propriedades de monotonicidade estrita, isotonicidade estrita e liberdade sejam obedecidas.

A implementação de uma ferramenta de simulação de algoritmos de roteamento multi-restritivo *hop-by-hop* com estratégia de otimização definida pelo nó de destino foi deixada para trabalhos futuros.

3.5 Algoritmo de Eliminação de *Loop* pelo Nó de Destino (ELND)

O algoritmo ELND foi desenvolvido com o objetivo de simular os resultados que se obteriam em uma rede que implementasse um algoritmo de roteamento multi-restritivo *hop-by-hop* com uma estratégia de otimização definida pelo usuário e adotada pelo nó de origem, sendo imposta, quando necessário, a propriedade de coerência.

Como variáveis de entrada, o algoritmo ELND utiliza: o dígrafo que modela

a rede a ser analisada, representado por uma matriz de adjacências; as várias métricas de roteamento adotadas e seus correspondentes valores associados a cada enlace da rede, em que os valores de cada métrica são representados como matrizes de adjacência ponderadas; as funções de combinação de métricas e as correspondentes sínteses léxicas e abreviações utilizadas; e o critério de ordenação léxica multi-dimensional, com os correspondentes critérios de ordenação léxica adotados.

Como variáveis de saída, o algoritmo ELND apresenta: uma matriz que contém todos os "próximos nós" escolhidos por cada nó da rede, para cada destino, ou seja, o resultado para o qual o algoritmo de roteamento multi-restritivo *hop-by-hop* convergiu segundo a estratégia de otimização adotada pelo nó de origem; uma lista ordenada, para cada par de nós origem-destino, de todos os caminhos obtidos; e as correspondentes palavras-peso resultantes de cada um desses caminhos.

O algoritmo ELND pode ser dividido em quatro grandes blocos listados abaixo e descritos na seqüência.

- Bloco 1: busca de todos os caminhos simples e elementares entre todos os pares de nós origem-destino da rede.
- Bloco 2: cálculo das palavras-peso multi-dimensionais correspondentes a todos os caminhos encontrados.
- Bloco 3: ordenação dos caminhos encontrados entre cada par de nós origem-destino e conseqüente ordenação das opções de "próximo nó" de cada nó de origem para cada nó de destino na rede.
- Bloco 4: busca, identificação e quebra de *loops*.

Note que inicialmente todos os caminhos simples e elementares encontrados são elegíveis e ordenados segundo a estratégia de otimização adotada. Assim, se os melhores caminhos eleitos pelos nós da rede não levarem a uma condição de *loop*, ou se a rede não possui ciclos, o algoritmo ELND converge de maneira ótima uma vez que, segundo definido, o algoritmo de roteamento multi-restritivo *hop-by-hop* convergiu de maneira ótima.

Portanto, se cada uma das métricas adotadas representar um parâmetro de QoS a ser otimizado na rede, pode-se dizer que, se o algoritmo ELND convergiu de maneira ótima, então a estratégia de otimização da QoS adotada é completamente

implementada, ou seja, a escolha do melhor caminho feita por todos os nós de origem para qualquer nó da rede é implementada.

Por outro lado, se o algoritmo ELND convergiu de maneira não ótima, ou seja, se um ou mais nós da rede não utilizarão sua escolha original do melhor caminho para um ou mais nós da rede, o usuário poderá reanalisar: as métricas, as funções de combinação de métricas, as correspondentes sínteses léxicas e abreviações utilizadas, e a ordenação léxica multi-dimensional adotada; buscando atender as propriedades de coerência, liberdade, monotonicidade e isotonicidade estritas e, conseqüentemente, a implementação completa da estratégia de otimização escolhida. De qualquer forma, vale ressaltar que pela heurística de seleção dos nós que alterarão suas escolhas de "próximo nó" e pela heurística da definição desses "próximos nós", o algoritmo ELND busca minimizar o número de nós que deverão alterar suas escolhas e, portanto, de certa forma implementar ao máximo a estratégia de otimização definida pelo usuário.

Com o objetivo de facilitar a compreensão e posterior análise de complexidade do algoritmo de simulação ELND, esta seção faz uma descrição detalhada em pseudocódigos do primeiro bloco funcional e em seguida descreve os demais blocos funcionais, discutindo seus respectivos tempos de execução. No Anexo A desta tese é apresentado o código fonte completo da implementação do algoritmo ELND em MATLAB.

Tendo em vista a compreensão do leitor, os pseudocódigos, descritos nesta seção, seguem a nomenclatura adotada nesta tese. Entretanto, cabe comentar que, como o algoritmo de simulação ELND foi implementado antes da redação desta tese, o leitor pode encontrar algumas incompatibilidades entre a nomenclatura desta seção e a contida no Anexo A.

A implementação da busca de todos os caminhos simples e elementares entre todos os pares de nós origem-destino, definida como Bloco 1, adota uma metodologia de busca e enumeração similar à utilizada pelo algoritmo de pesquisa primeiro na largura, descrito na Seção 2.2.3.1. Sua implementação é feita pelo seqüenciamento de dois algoritmos: BUSCAPATH(V, E) e SEPARAPATH(A, M).

O pseudocódigo BUSCAPATH(V, E), descrito a seguir, retorna, para todos os vértices do dígrafo $N = (\mathbf{V}, \mathbf{E})$, todos os caminhos encontrados para cada vértice, como vértice de origem, a partir da exploração sistemática das arestas que se conectam a h saltos do vértice de origem (u).

BUSCAPATH(V, E)

```
1  for cada vértice (u) ∈ V    ▷ busca todos os caminhos para todos os nós
2      h ← 1    ▷ inicializa profundidade dos caminhos
3      pn ← ∅    ▷ inicializa lista de "próximos nós"
4      n ← 0    ▷ inicializa índice de "próximos nós"
5      A[u, h, n] ← ∅    ▷ inicializa lista de enalces a partir da origem (u)
6      for cada (s) ∈ V    ▷ busca os "próximos nós"
7          if (s) : (u, s) ∈ E then    ▷ se há "próximo nó"
8              pn ← (s)    ▷ salva "próximo nó"
9              n ← n + 1    ▷ atualiza índice de "próximos nós"
10             A[(u), h, n] ← (s)    ▷ salva enlace
11     h ← h + 1    ▷ incrementa a profundidade dos caminhos
12     while pn ≠ ∅    ▷ enquanto houver novos "próximos nós" busca os
                           correspondentes "próximos nós"
13             npn ← ∅ inicializa lista de novos "próximos nós"
14             n ← 0    ▷ inicializa índice de "próximos nós"
15             for cada pn    ▷ para cada "próximo nó"
16                 for cada (s) ∈ V    ▷ busca novos "próximos nós"
17                     if (s) : (pn, s) ∈ E e (s) ∉ ∀ A[(u), h, pn] then
                       ▷ se há novo "próximo nó"
18                         n ← n + 1    ▷ atualiza índice de "próximos nós"
19                         A[(u), h, n] ← (s)    ▷ salva enlace
20                         npn ← (s)    ▷ salva novo "próximo nó"
21             if npn ≠ ∅ then    ▷ se há novos "próximos nós"
22                 pn ← npn    ▷ salva nova lista de "próximos nós"
23                 h ← h + 1    ▷ incrementa a profundidade do caminho
24             else pn ← ∅    ▷ limpa a lista de "próximos nós"
25 return (A)
```

Fazendo uma análise da complexidade desse algoritmo, observa-se que seu tempo de execução no pior caso é $O(n^3)$, em que n é a ordem do dígrafo N. Note que para cada um dos n vértices do dígrafo, as linhas 2 a 11 demoram o tempo $\theta(n)$, a busca dos "próximos nós" pelos correspondentes "próximos nós", as linhas 16 a 20 levam um tempo $\theta(n)$. No pior caso, em que se encontra um "próximo nó" a cada vez, as linhas 12 a 24 demoram um tempo $O(n^2)$ e, portanto, incluindo o laço da linha 1, o tempo de execução total no pior caso é $O(n^3)$.

A partir da lista dos caminhos encontrados para cada nó da rede, o pseudocódigo SEPARAPATH($A[(s), h, i], M[m]$), descrito a seguir, lista para todos os pares de nós origem-destino da rede, todos os correspondentes caminhos simples e

elementares existentes, com os correspondentes valores de métricas adotados para cada aresta. Observe que $M[m]$ corresponde à matriz ponderada de adjacências para a métrica m.

SEPARAPATH($A[(s), h, i], M[m]$)

1 **for** cada vértice $(s) \in V$ ▷ busca todos os nós de origem
2 **for** cada vértice $(d) \in V$ ▷ busca todos os nós de destino
3 **if** $(s) = (d)$ **then** ▷ inicializa caminho vazio com peso vazio
4 $(P[(s), (d)], w[m, (s), (d)]) \leftarrow$ ZERACAMINHO $(P[(s), (d)], w[m, (s), (d)])$
5 **else** ▷ inicializa lista de caminhos $p_{s,d}$
6 **if** $A[(s), h, i] \neq \emptyset \ \forall \ h, i$ **then** ▷ verifica se existe algum caminho com origem em (s)
7 $P' \leftarrow (s)$ ▷ inicializa caminhos com origem (s)
8 **for** cada métrica m ▷ inicializa pesos dos caminhos com origem (s)
9 $l \leftarrow NIL$
10 $w'[m, l] \leftarrow \emptyset$
11 $i \leftarrow j \leftarrow 0$ ▷ inicializa contadores de caminhos
12 **for** cada caminho $j \in A[(s), h, i] \ \forall \ h, i$
13 $j \leftarrow j + 1$
14 $P'[j] \leftarrow A[(s)]$ ▷ salva o caminho
15 **if** $P'[j] \equiv P'[j, (s) \ldots (d)]$ **then** ▷ verifica se é um caminho com destino (d)
16 $i \leftarrow i + 1$
17 $P[(s), (d), i] \leftarrow P'[j, (s) \ldots (d)]$
18 **for** cada $l \in P'[j]$ ▷ salva peso w da métrica m correspondente de todos os enlaces l do caminho
19 $w'[m, l] \leftarrow M[m]$
20 $w[m, (s), (d), i] \leftarrow w'[m, l] \ \forall \ l \in P'[j]$
21 **if** $i = 0$ **then** ▷ se não há nenhum caminho com destino (d)
22 $(P[(s), (d)], w[m, (s), (d)]) \leftarrow$ ZERACAMINHO $(P[(s), (d)], w[m, (s), (d)])$
23 **else** ▷ se não existe nenhum caminho
$(P[(s), (d)], w[m, (s), (d)]) \leftarrow$ ZERACAMINHO $(P[(s), (d)], w[m, (s), (d)])$

24 **return** $(P[(s),(d)], w[m,(s),(d)])$

ZERACAMINHO$(P[(s),(d)], w[m,(s),(d)])$
1 $P[(s),(d)] \leftarrow \emptyset$
2 **for** cada métrica m
3 $\quad w[m,(s),(d)] \leftarrow \emptyset$
4 **return** $(P[(s),(d)], w[m,(s),(d)])$

Fazendo uma análise da complexidade desse algoritmo, observa-se que seu tempo de execução no pior caso é $O(n^2 \times Np \times m)$, em que n é o número de nós da rede, m é o número de métricas adotadas e Np é o número máximo de caminhos simples e elementares existentes entre um nó de origem e qualquer nó de destino da rede. Note que para cada par de nós $((s),(d))$, em um total de $n \times (n-1)$ pares, o sistema busca no total de caminhos encontrados com origem em (s), quais caminhos têm o vértice (d) de destino como o último vértice do caminho e, para esses caminhos, o sistema salva, para cada métrica, os correspondentes pesos das arestas de cada caminho. Observe também que, para cada um dos n vértices de origem, as linhas 12 a 20 buscam, em um máximo de Np caminhos, todos os caminhos possíveis a partir de $A[(s)]$ para as m métricas adotadas. Portanto, no pior caso, o tempo de execução entra as linhas 12 e 20 é $O(Np \times m)$ e o tempo de execução total é $O(n^2 \times Np \times m)$.

No entanto, no pior cenário, quando o dígrafo obedece a propriedade de totalidade, possuindo $n \times (n-1)$ arestas, o número máximo de caminhos simples e elementares existentes, obtido por indução, é $\sum_{i=0}^{n-2-1} C_{n-2,i} \times (n-2-i)!$. Substituindo a combinação de $(n-2)$ tomados i-a-i por $(n-2)!/[i! \times (n-2-i)!]$, tem-se $Np = (n-2)! \sum_{i=0}^{n-3} 1/i!$. Portanto, nesse cenário, o tempo de execução desse algoritmo é $O(n^n \times m)$, uma vez que $\sum_{i=0}^{n-3} 1/i! < \sum_{i=0}^{\infty} 1/i! = e$, para $n \rightarrow \infty$ $e \times (n-2)! < e \times n^{n-2} \Rightarrow (n-2)! < n^{n-2}$, em que, segundo a definição da notação $O(g(n))$ dada na Seção 2.2.2.3, para $n \geq n_0 = 2$: $0 \leq f(n) = (n-2)! \leq cg(n) = n^{n-2}$.

Entretanto, nos casos práticos, como será visto no Capítulo 4, verifica-se que esta obtenção de complexidade NP não inviabiliza a utilização do algoritmo ELND.

A implementação do Bloco 2, ou seja, do cálculo das palavras-peso multidimensionais correspondentes a todos os caminhos encontrados no Bloco 1, está descrito na rotina CALCULAPATH listada no Anexo A.

Como variáveis de entrada, o procedimento CALCULAPATH utiliza: o con-

junto de caminhos **P**, obtidos para cada par de nós origem-destino; o conjunto **w** dos pesos das arestas para cada métrica adotada e correspondente a cada caminho em **P**; o conjunto ordenado das sínteses léxicas **SL**, correspondente a cada métrica utilizada; o conjunto ordenado das relações de ordenação léxica **OL**, correspondente às sínteses utilizadas; e o conjunto ordenado **b** das correspondentes abreviações adotadas para cada síntese léxica. Note que nessa implementação, com o objetivo de não restringir os possíveis cenários a serem analisados, as funções de combinação de métricas adotadas são sempre identidades. Caso o usuário queira utilizar outras funções, basta processar os valores correspondentes das métricas combinadas e utilizar o resultado como um conjunto **w** de variáveis de entrada.

Como variáveis de saída, o procedimento CALCULAPATH retorna: o conjunto **Po** de todos os caminhos, para cada par de nós origem-destino, ordenados lexicamente para cada uma das sínteses léxicas adotadas, ou seja, para $\forall\ (s),(v) \in \mathbf{V}$, tem-se um conjunto $\mathbf{Po}[(\mathbf{s}),(\mathbf{v}),\mathbf{m}^{'}]$ lexicamente ordenado, segundo a relação de ordenação léxica adotada para a métrica $m^{'}$, de todos os caminhos encontrados entre (s) e (v) e, portanto, $\mathbf{Po}[(\mathbf{s}),(\mathbf{v}),\mathbf{m}^{'}] \subset \mathbf{Po}$; e o conjunto **So** dos respectivos valores das palavras-peso dos caminhos ordenados, isto é, $\forall\ \mathbf{Po}[(\mathbf{s}),(\mathbf{v}),\mathbf{m}^{'},\mathbf{p}^{'}] \Rightarrow \mathbf{So}[(\mathbf{s}),(\mathbf{v}),\mathbf{m}^{'},\mathbf{S}]$, em que S é a palavra-peso resultante da aplicação da síntese $SL[m^{'}]$, sobre os pesos das arestas do caminho $p^{'}(s,v)$, correspondente à métrica $m^{'}$.

Fazendo uma análise da complexidade desse algoritmo, verifica-se que, no pior caso, o seu tempo de execução é $O(n^2 \times Np \times m)$. O algoritmo CALCULAPATH leva $\Theta(n^2 \times m)$ para listar para todas as métricas, todos os pares de nós origem-destino da rede e, para cada métrica e par de nós origem-destino, o algoritmo calcula, com um tempo de execução $O(n^1)$, o valor da palavra-peso correspondente em função da respectiva abreviação adotada.

A implementação do Bloco 3, isto é, da ordenação dos caminhos encontrados entre cada par de nós origem-destino, cujos valores das palavras-peso correspondentes foram calculados pelo procedimento CALCULAPATH, está descrito na rotina ORDENAPATHS listada no Anexo A.

Como variáveis de entrada, o procedimento ORDENAPATHS utiliza: o conjunto ordenado das sínteses léxicas **SL** correspondente a cada métrica utilizada, pois a ordem em que são listadas as sínteses é a ordem de precedência adotada para a relação de ordenação léxica multi-dimensional; o conjunto ordenado **Po** de todos os caminhos, para cada par de nós origem-destino, em que para $\forall\ (s),(v) \in$

V tem-se um conjunto $\mathbf{Po}[(\mathbf{s}),(\mathbf{v}),\mathbf{m}']$, lexicamente ordenado, segundo a relação de ordenação léxica adotada para a métrica m', de todos os caminhos encontrados entre (s) e (v); e o conjunto **So** dos respectivos valores das palavras-peso dos caminhos ordenados, isto é, $\forall\ \mathbf{Po}[(\mathbf{s}),(\mathbf{v}),\mathbf{m}',\mathbf{p}'] \Rightarrow \mathbf{So}[(\mathbf{s}),(\mathbf{v}),\mathbf{m}',\mathbf{S}]$.

Como variáveis de saída, o procedimento ORDENAPATHS retorna: o conjunto **Pfo** de todos os caminhos, para cada par de nós origem-destino, ordenados segundo a ordenação léxica multi-dimensional adotada e o conjunto **Sfo** das palavras-peso correspondentes a cada caminho **Pfo**, ou seja, para $\forall\ p_{(s),(v)} \in \mathbf{Pfo}[(\mathbf{s}),(\mathbf{v})] \Rightarrow Sfo[(s),(v),m,S] \in \mathbf{Sfo}$.

Fazendo uma análise da complexidade desse algoritmo, verifica-se que o seu tempo de execução no pior caso é $O(n^2 \times Np \times m)$, pois o algoritmo leva $\Theta(n^2)$ para listar para todos os pares de nós origem-destino da rede e, para cada 3-upla (s,d,m), o algoritmo busca para todos os caminhos com mesma palavra-peso obtida pela métrica m, a ordenação dada pela palavra-peso da métrica $m+1$. Para esses caminhos, se a palavra-peso da métrica $m+1$ é igual, o algoritmo analisa a ordenação dada pela palavra-peso da métrica $m+2$ e segue, sucessivamente, até a última métrica adotada. Essa busca tem, como pior caso, a condição em que todos os caminhos têm o mesmo valor de palavra-peso para cada métrica adotada, isto é, todos os caminhos encontrados são equivalentes. Nesse caso, o tempo de execução é dado por $O(Np \times m)$.

Baseando-se nos caminhos ordenados do conjunto **Pfo** o algoritmo gera, para cada par de nós origem-destino da rede, uma lista $\mathbf{NHl}[(\mathbf{s}),(\mathbf{v})]$ ordenada de "próximos nós", ou seja, tendo sido obtidos todos os caminhos elegíveis entre um par de nós origem-destino, em razão da ordenação léxica multi-dimensional adotada, define-se a ordenação dos "próximos nós" contida no conjunto ordenado $\mathbf{NHl}[(\mathbf{s})(\mathbf{d})]$. Com base nessa lista, o algoritmo ELND constrói uma matriz de "próximos nós", denominada $\overline{NH}$, em que as linhas da matriz representam o nós de origem, as colunas da matriz representam os nós de destino e o elemento da matriz indica o "próximo nó" escolhido pelo nó de origem para enviar uma informação ao correspondente nó de destino.

A implementação da busca, identificação e quebra de *loops*, pelos nós de destino para os caminhos eleitos pelos nós de origem (Bloco 4), está descrito na rotina MAIN listada no Anexo A, em que, para cada destino e por meio da rotina BUSCALOOP, o algoritmo verifica se existe algum *loop*. Se não houver nenhum *loop*, então o algoritmo ELND retorna a matriz $\overline{NH}$ originalmente obtida pelos nós de origem.

Entretanto, se para qualquer nó de destino ($d^{'}$) é encontrado algum *loop*, com base na correspondente lista de "próximos nós" $\mathbf{NHl}[(\mathbf{s})(\mathbf{d}^{'})]$, dos nós de origem que escolheram caminhos indiretos para o nó ($d^{'}$), o algoritmo, no papel do nó de destino ($d^{'}$), seleciona qual desses nós de origem deve alterar sua escolha de "próximo nó" para o destino ($d^{'}$), bem como, elege qual deverá ser o novo "próximo nó" para esse nó de origem. Note que para isso, pressupõe-se que os nós de destinos da rede, por meio de algum protocolo de roteamento, receberam de cada nó de origem as respectivas listas ordenadas de "próximos nós".

No algoritmo ELND, a alteração na escolha do "próximo nó" pelo nó de origem eleito é feita pela alteração na matriz $\overline{NH}$ do correspondente elemento e pela eliminação na correspondente lista $\mathbf{NHl}[(\mathbf{s})(\mathbf{d})]$ do "próximo nó" anteriormente escolhido.

Cada vez que a matriz $\overline{NH}$ é alterada, no término da análise de todos os nós de destino da rede, o algoritmo ELND retorna à busca de *loops* para cada nó de destino da rede.

Observe que, como a rede é representada por um dígrafo fortemente conectado, para todos os *loops* detectados por qualquer nó de destino ($d^{'}$), sempre haverá um nó ($s_{d'}$), no conjunto $\mathbf{s_{ind}}[\mathbf{d}^{'}]$ dos nós que escolheram caminhos indiretos para o nó de destino ($d^{'}$), que ao alterar sua escolha de "próximo nó" tornará, para um subconjunto de nós $\mathbf{s}^{'}[\mathbf{d}^{'}] \subset \mathbf{s_{ind}}[\mathbf{d}^{'}]$ a escolha dos caminhos indiretos em caminhos diretos. Como o algoritmo ELND elimina da lista elegível de "próximo nó", $\mathbf{NHl}[(\mathbf{s_{d'}})(\mathbf{d}^{'})]$, o "próximo nó" que levou o subconjunto $\mathbf{s}^{'}[\mathbf{d}^{'}]$ à condição de escolha de caminhos indiretos, para cada novo conjunto $\mathbf{s_{ind}}[\mathbf{d}^{'}]$ ele elege somente um nó ($s_{d'}$) para alterar a sua escolha, e como o dígrafo é fortemente conectado, então o algoritmo ELND sempre encontrará uma matriz $\overline{NH}$, em que a propriedade de coerência é obedecida.

A busca de *loops* por um nó de destino é feita pela rotina BUSCALOOP, listada no Anexo A. Como variáveis de entrada, o procedimento BUSCALOOP utiliza a matriz $\overline{NH}$ de "próximo nó" e o conjunto ordenado $\mathbf{NHl}[(\mathbf{s})(\mathbf{d})]$ dos 'próximos nós" eleitos pelos respectivos nós de origem para o destino (d).

Como variáveis de saída, o procedimento BUSCALOOP retorna o nó que deve alterar a sua escolha de "próximo nó", o novo "próximo nó" e uma lista dos *loops* encontrados, como os respectivos nós que os compõem.

A partir da matriz $\overline{NH}$ de "próximo nó", o algoritmo BUSCALOOP monta em um tempo $O(n^2)$ uma matriz de "adjacência pelo destino" $\overline{C}$, ou seja, uma

matriz em que cada linha i corresponde a um nó de origem e cada coluna j a um "próximo nó", isto é, cada elemento $c_{i,j} \in \overline{C}$ é dado por:

$$c_{i,j} = \begin{cases} 1 & \text{se } j = \overline{NH}(i,d) \\ 0 & \text{caso contrário} \end{cases}.$$

Por inspeção da matriz $\overline{C}$, o algoritmo BUSCALOOP encontra todos os nós conectados diretamente ao nó de destino ($d^{'}$) analisado, isto é, todas as linhas da matriz $\overline{C} : c_{i,j} \neq 0$ e $j = d^{'}$ e que, portanto, não pertencem a nenhum *loop*. Pelo mesmo processo, o algoritmo BUSCALOOP obtém todos os nós que escolheram caminhos diretos pela escolha recursiva do "próximo nó", pelo próximo nó para o nó de destino ($d^{'}$) e, portanto, também obedecem a propriedade de coerência. No pior caso, em que todos os nós se conectam por caminhos diretos ao nó de destino ($d^{'}$) pela escolha recursiva do "próximo nó", pelo "próximo nó", esse procedimento de inspeção leva um tempo $O(n^1)$.

Ao final dessas inspeções, a matriz $\overline{C}$ é reduzida a uma matriz que contém somente nós que pertencem a caminhos indiretos para o nó de destino, isto é, pertencem ao conjunto $\mathbf{s_{ind}}[\mathbf{d}^{'}]$ e, portanto, são nós que devem ser evitados na escolha do novo "próximo nó".

Com base no conjunto $\mathbf{s_{ind}}[\mathbf{d}^{'}]$, dado pela matriz reduzida de $\overline{C}$, o algoritmo BUSCALOOP identifica pelas correspondentes listas $\mathbf{NHl}[(\mathbf{s_{ind}})(\mathbf{d}^{'})]$ dos "próximos nós" escolhidos pelos nós remanescentes na matriz reduzida de $\overline{C}$, quais nós têm como opção de "próximos nós", nós que não precisam ser evitados e, entre esses nós, o nó remanescente, eleito para alterar a sua escolha do "próximo nó", é aquele que tem maior número de conexões "entrantes", ou seja, aquele que tem o maior número de nós adjacentes que optaram por ele como "próximo nó".

Assim, com base na heurística de imposição da propriedade de coerência apresentada e fazendo uma agregação da complexidade de cada procedimento descrito anteriormente, o algoritmo ELND, com um tempo de execução $O(n^2 \times Np \times m)$, busca minimizar o número de nós que deverão alterar suas escolhas, buscando implementar ao máximo a estratégia de otimização definida pelo usuário.

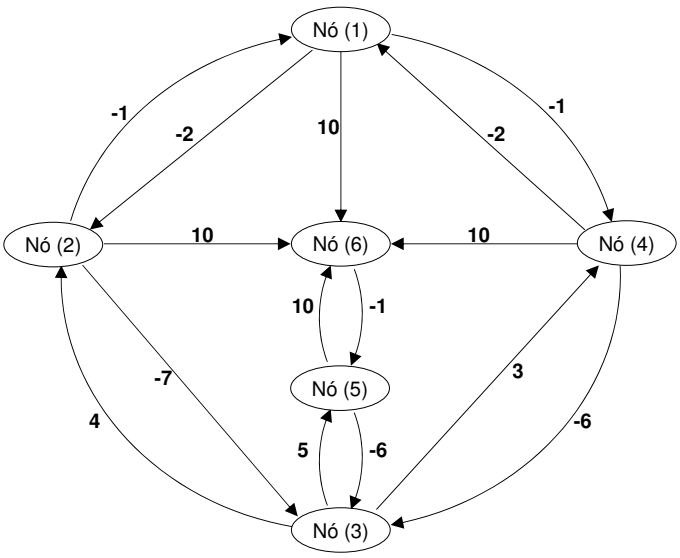


Figura 3.18: Dígrafo da rede exemplo

3.6 Exemplo Didático: Análise em Grafos com Ciclos Negativos

Visando ilustrar a imposição da propriedade de coerência pelo algoritmo ELND, considere o exemplo didático, mono-restritivo, cujo dígrafo e os respectivos pesos de cada aresta estão mostrados na Figura 3.18. A matriz $\overline{Adj}$ de adjacência e a matriz $\overline{M}$ dos correspondentes valores da métrica são listadas a seguir.

$$\overline{Adj} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\overline{M} = \begin{bmatrix} 0 & -2 & 0 & -1 & 0 & 10 \\ -1 & 0 & -7 & 0 & 0 & 10 \\ 0 & 4 & 0 & 3 & 5 & 0 \\ -2 & 0 & -6 & 0 & 0 & 10 \\ 0 & 0 & -6 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Definindo-se uma estratégia de otimização, em que o melhor caminho é aquele que apresenta o menor valor da palavra-peso resultante, dada pela síntese léxica

aditiva com abreviação $b_l = b_\infty$, obtém-se uma primeira matriz de "próximos nós" dada pela expressão 3.59, em que a Tabela 3.15 apresenta a ordenação, do melhor para o pior, de todos os caminhos elegíveis para todos os pares de nós origem-destino do dígrafo.

Tabela 3.15: Ordenação dos caminhos para os vértices em *loop*

Vértices de origem	**Vértices de destino**					
	1	**2**	**3**	**4**	**5**	**6**
1	-	1,2 1,4,3,2 1,4,6,5,3,2 1,6,5,3,2	1,2,6,5,3 1,4,6,5,3 1,6,5,3 1,4,3 1,2,3	1,4 1,2,3,4 1,2,6,5,3,4 1,2,6,5,3,4 1,6,5,3,4	1,4,3,5 1,2,3,5 1,2,3,4,6,5 1,4,3,2,6,5 1,2,6,5 1,4,6,5 1,6,5	1,2,3,4,6 1,2,3,5,6 1,4,3,2,6 1,2,6 1,4,3,5,6 1,4,6 1,6
2	2,1 2,3,4,1 2,6,5,3,4,1	-	2,1,4,6,5,3 2,1,6,5,3 2,6,5,3 2,3 2,1,4,3	2,1,4 2,3,4 2,1,6,5,3,4 2,6,5,3,4	2,3,5 2,1,4,3,5 2,3,4,1,6,5 2,3,4,6,5 2,1,4,6,5 2,1,6,5 2,6,5	2,3,4,1,6 2,3,4,6 2,1,4,3,5,6 2,1,4,6 2,3,5,6 2,1,6 2,6
3	3,4,1 3,2,1	3,4,1,2 3,2	-	3,2,1,4 3,4	3,5 3,4,1,2,6,5 3,4,1,6,5 3,2,1,4,6,5 3,2,1,6,5 3,4,6,5 3,2,6,5	3,4,1,2,6 3,4,1,6 3,2,1,4,6 3,2,1,6 3,4,6 3,2,6 3,5,6
4	4,1 4,3,2,1 4,6,5,3,2,1	4,3,2 4,1,2 4,1,6,5,3,2 4,6,5,3,2	4,1,2,6,5,3 4,1,6,5,3 4,6,5,3 4,3 4,1,2,3	-	4,3,5 4,1,2,3,5 4,1,2,6,5 4,3,2,1,6,5 4,1,6,5 4,3,2,6,5 4,6,5	4,1,2,3,5,6 4,1,2,6 4,3,2,1,6 4,1,6 4,3,2,6 4,3,5,6 4,6
5	5,3,2,1 5,3,4,1	5,3,2 5,3,4,1,2	5,3	5,3,4 5,3,2,1,4	-	5,3,4,1,2,6 5,3,4,1,6 5,3,2,1,4,6 5,3,2,1,6 5,3,4,6 5,3,2,6 5,6
6	6,5,3,2,1 6,5,3,4,1	6,5,3,2 6,5,3,4,1,2	6,5,3	6,5,3,4 6,5,3,2,1,4	6,5	-

$$\overline{NH_1} = \begin{bmatrix} 0 & 2 & 2 & 4 & 4 & 2 \\ 1 & 0 & 1 & 1 & 3 & 3 \\ 4 & 4 & 0 & 2 & 5 & 4 \\ 1 & 3 & 1 & 0 & 3 & 1 \\ 3 & 3 & 3 & 3 & 0 & 3 \\ 5 & 5 & 5 & 5 & 5 & 0 \end{bmatrix} \tag{3.59}$$

Nesse estado, a rede apresenta para os nós de destinos (2), (3) e (6) os seguintes *loops*:

$$\begin{cases} p_{\mathbf{3},2} = 3, \mathbf{4}, 1, 2 \\ p_{\mathbf{4},2} = 4, \mathbf{3}, 2 \end{cases}, \begin{cases} p_{\mathbf{1},3} = 1, \mathbf{2}, 6, 5, 3 \\ p_{\mathbf{2},3} = 2, \mathbf{1}, 4, 6, 5, 3 \end{cases} \text{ e } \begin{cases} p_{\mathbf{1},6} = 1, \mathbf{2}, 3, 4, 6 \\ p_{\mathbf{2},6} = 2, \mathbf{3}, 4, 1, 6 \\ p_{\mathbf{3},6} = 3, \mathbf{4}, 1, 2, 6 \\ p_{\mathbf{4},6} = 4, \mathbf{1}, 2, 3, 5, 6 \end{cases}.$$

Para cada um desses *loops*, o sistema selecionou os nós (1) e (3) para alterarem a opção inicial do "próximo nó", buscando impor a propriedade de coerência. A seguir estão listadas as correspondentes opções de novos "próximo nó", dos nós (1) e (3).

- nó de destino (2) nó de origem (3) → "próximo nó" (2).
- nó de destino (3) nó de origem (1) → "próximo nó" (6).
- nó de destino (6) nó de origem (1) → "próximo nó" (6).

Com base nessas novas opções, o sistema constrói a próxima matriz de "próximos nós" dada pela expressão 3.60, verificando que não há mais *loops* e, portanto, a propriedade de coerência foi imposta.

$$\overline{NH_2} = \begin{bmatrix} 0 & 2 & 6 & 4 & 4 & 6 \\ 1 & 0 & 1 & 1 & 3 & 3 \\ 4 & 2 & 0 & 2 & 5 & 4 \\ 1 & 3 & 1 & 0 & 3 & 1 \\ 3 & 3 & 3 & 3 & 0 & 3 \\ 5 & 5 & 5 & 5 & 5 & 0 \end{bmatrix} \tag{3.60}$$

Assim, pode-se observar que, mesmo quando a propriedade de monotonicidade e isotonicidade não são obedecidas, sendo obedecida a propriedade de

coerência, o algoritmo de roteamento *hop-by-hop* converge.

4 Estudos de Casos

Utilizando o algoritmo ELND, esta Seção discute e analisa os resultados obtidos, em dois estudos de casos, com a aplicação da álgebra de caminhos proposta e do algoritmo de roteamento multi-restritivo *hop-by-hop*, em que a estratégia de otimização definida pelo usuário é adotada pelo nó de origem e a propriedade de coerência, quando necessário, é imposta.

Cabe comentar que a escolha desses dois casos se baseou na facilidade da exploração, possibilitada pelas redes analisadas, de diferentes métricas, sínteses e aplicações das propostas apresentadas no Capítulo 3.

4.1 Rede SAm-1

A rede *South America - 1* (SAm-1) consiste em aproximadamente 23.000 km de cabo óptico submarino de 4 pares de fibras ópticas e 2.000 km de cabo óptico terrestre formando um anel que conecta as principais cidades de América Latina aos Estados Unidos, conforme ilustrado no diagrama da Figura 4.1. Utilizando tecnologia DWDM, que permite acomodar até 48 canais de 10 Gbps, a rede SAm-1 comporta uma capacidade (vazão) total de 1,92 Tbps [86].

Do ponto de vista de equipamentos de transmissão, com o objetivo de atender às constantes demandas de melhoria de desempenho e aumento de capacidade, a rede SAm-1 utiliza uma nova geração de equipamentos SDH, do fabricante Nortel, denominados HDX, que implementam as arquiteturas ASON e GMPLS [87].

Assim, como um equipamento ASON/GMPLS, o equipamento HDX permite criar um caminho entre dois nós da rede, de duas maneiras: explícita e implícita. Na maneira explícita, como qualquer equipamento SDH, o operador da rede define todas as conexões necessárias para construir um caminho entre dois nós da rede. Na forma implícita, no entanto, por meio de um plano de controle distribuído, o caminho entre dois nós, informados pelo operador, é definido e construído automaticamente, segundo alguns critérios de roteamento pré-definidos pelo fabricante

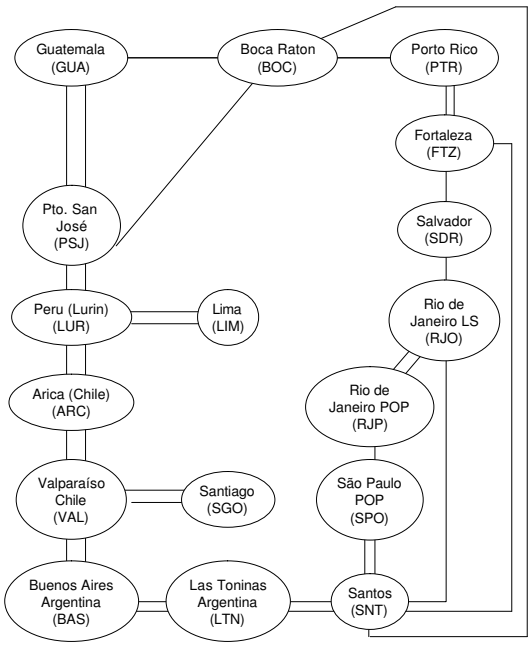


Figura 4.1: Digrama esquemático da rede SAm-1

de equipamentos [88].

Considere para este estudo de caso, que o equipamento utilizado adota duas métricas de roteamento: uma métrica qualquer M_1, cujos valores, inseridos enlace-a-enlace pelo operador, são acumulados pelo sistema ao longo de um caminho, e uma métrica $M_2 = \textit{número de saltos}$. Considere também, que o melhor caminho, eleito pela rede, é aquele que tem o menor valor acumulado para a métrica M_1 e, caso haja mais de um caminho nessa condição, o critério de desempate adotado é o de menor valor acumulado para a métrica M_2, isto é, o caminho com menor número de saltos. Cabe comentar que essa estratégia de otimização é conhecida na literatura com o nome de problema *All Hops Optimal Path* (AHOP) [89].

A implementação desses critérios de otimização pela álgebra de caminhos proposta pode ser feita de duas maneiras:

1. adotam-se as métricas M_1 e M_2, conforme descritas anteriormente; duas funções de combinação de métricas $f(w_1)$ e $f(w_2)$, tais que $f(w_1) = c_1 = w_1$ e $f(w_2) = c_2 = w_2$, em que w_1 e w_2 correspondem, respectivamente, aos pesos das métricas M_1 e M_2 em cada enlace da rede; as sínteses léxicas aditivas $S_1[.]$ e $S_2[.]$ e as abreviações $b = b_1$ para ambas as sínteses adotadas; a ordenação léxica $\preceq_L$ dada por $\geq$, para ambas as sínteses adotadas; e a ordenação léxica multi-dimensional $\preceq_{ML}$, tal que $\overline{S}[\overline{C}(\alpha)] \prec_{ML} \overline{S}[\overline{C}(\beta)]$ se $b_{l_1}(S_1[w_1(\alpha)]) >_L b_{l_1}(S_1[w_1(\beta)])$ ou $b_{l_1}(S_1[w_1(\alpha)]) =_L b_{l_1}(S_1[w_1(\beta)])$ e $b_{l_2}(S_2[w_2(\alpha)]) >_L b_{l_2}(S_2[w_2(\beta)])$; ou

2. utilizando somente a métrica M_1, conforme descrita anteriormente, uma função de combinação de métricas $f(w_1) = c_1 = w_1$, em que w_1 corresponde, respectivamente, aos pesos da métrica M_1 em cada enlace da rede; uma síntese léxica aditiva $S_1[.]$ e correspondente abreviação $b = b_\infty$; uma ordenação léxica $\preceq_L$ dada por $\geq_L$; e uma ordenação léxica multi-dimensional $\preceq_{ML}$, tal que $\overline{S}[\overline{C}(\alpha)] \prec_{ML} \overline{S}[\overline{C}(\beta)]$ se $b_{l_1}(S_1[w_1(\alpha)]) >_L b_{l_1}(S_1[w_1(\beta)])$.

   Observe que a palavra-peso multi-dimensional nesse caso é a abreviação de uma "letra", que corresponde a primeira palavra-peso obtida da palavra-peso multi-dimensional dada pela maneira anterior. Note ainda que, ao utilizar a abreviação $b = b_\infty$, pela própria definição da síntese léxica e da ordenação léxica, se incorpora o número de saltos na definição do melhor caminho para cada métrica analisada.

Como, pela definição apresentada na Seção 2.1.2, um dígrafo não pode ter laços e arestas distintas paralelas, o dígrafo equivalente ao diagrama de rede ilustrado na Figura 4.1 está apresentado na Figura 4.2. Note que os enlaces paralelos da rede foram substituídos por um único enlace equivalente e, para facilitar a análise, mas sem perda de generalidade, os nós de Arica e Porto San José foram removidos.

O dígrafo ilustrado na Figura 4.2 está ponderado com valores de disponibilidade, vazão e distância estimados. O valor do atraso foi obtido teoricamente em função da distância e da velocidade da luz na fibra[1], desprezando-se os atrasos gerados internamente pelos equipamentos da rede.

Assim, dado o conjunto de métricas $\mathbf{M} = \{M_1, M_2\}$, a matriz de adjacências $\overline{Adj}$; as matrizes $\overline{M}[1]$ e $\overline{M}[2]$ para os respectivos valores das métricas M_1 e M_2 em cada enlace, isto é, os pesos correspondentes w_1 e w_2 de cada enlace; o conjunto de sínteses léxicas $\mathbf{SL} = \{\text{aditiva, aditiva}\}$; o conjunto de correspondentes abreviações $\mathbf{b} = \{b_1, b_1\}$; e o conjunto de relações de ordenações léxicas $\preceq_L$ dado

[1] Adotou-se o valor de 1,45 para o índice de refração da fibra [90] e 299.792,458 km/s para a velocidade da luz no vácuo [91].

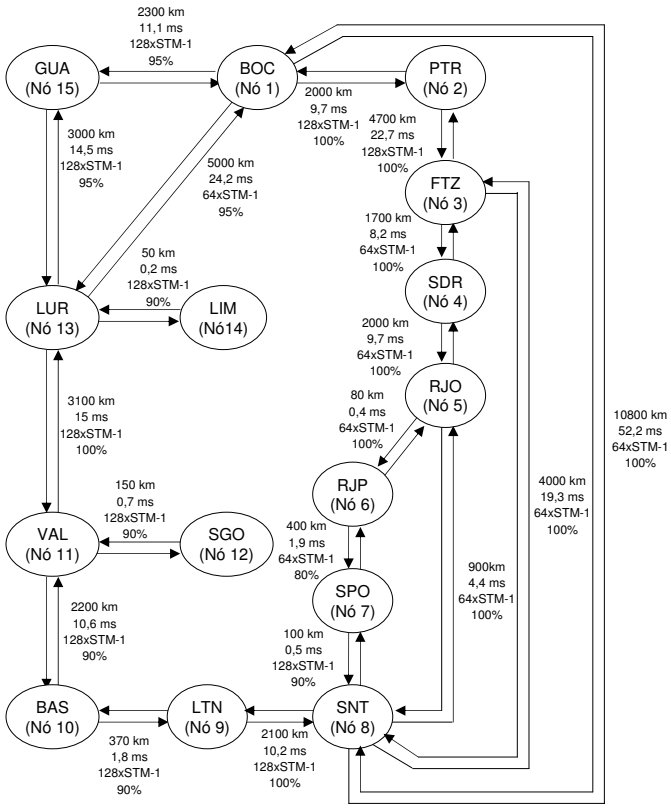


Figura 4.2: Dígrafo equivalente da rede SAm-1

por $\mathbf{OL} = \{\geq_L, \geq_L\}$, em que:

$$\overline{Adj} = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix},$$

$\overline{M}[2] =$ *número de saltos*, isto é,

$$\overline{M}[2] = \begin{bmatrix}
\infty & 1 & \infty & \infty & \infty & \infty & \infty & 1 & \infty & \infty & \infty & \infty & 1 & \infty & 1 \\
1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & 1 & \infty & 1 & \infty & \infty & \infty & 1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & 1 & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
1 & \infty & 1 & \infty & 1 & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty & 1 & 1 & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty & \infty & \infty & \infty \\
1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty & \infty & 1 & 1 \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty & \infty \\
1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty & \infty
\end{bmatrix} \text{ e}$$

$\overline{M}[1] =$ *distância* em 10^3 km, ou seja,

$$\overline{M}[1] = \begin{bmatrix}
\infty & 2 & \infty & \infty & \infty & \infty & \infty & 10,8 & \infty & \infty & \infty & \infty & 5 & \infty & 2,3 \\
2 & \infty & 4,7 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & 4,7 & \infty & 1,7 & \infty & \infty & \infty & 4 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & 1,7 & \infty & 2 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & 2 & \infty & 0,08 & \infty & 0,9 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & 0,08 & \infty & 0,4 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & 0,4 & \infty & 0,1 & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
10,8 & \infty & 4 & \infty & 0,9 & \infty & 0,1 & \infty & 2,1 & \infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & 2,1 & \infty & 0,37 & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0,37 & \infty & 2,2 & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 2,2 & \infty & 0,15 & 3,1 & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0,15 & \infty & \infty & \infty & \infty \\
5 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 3,1 & \infty & \infty & 0,05 & 3 \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0,05 & \infty & \infty \\
2,3 & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 3 & \infty & \infty
\end{bmatrix} \text{ ou}$$

$\overline{M}[1] = 1 - disponibilidade$, isto é,

$$\overline{M}[1] = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0,05 & 1 & 0,05 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0,1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0,1 & 1 & 0,2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0,2 & 1 & 0,1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0,1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0,1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0,1 & 1 & 0,1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0,1 & 1 & 0,1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0,1 & 1 & 1 & 1 & 1 \\ 0,05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0,1 & 0,05 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0,1 & 1 & 1 \\ 0,05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0,05 & 1 & 1 \end{bmatrix},$$

em que é utilizada a função de combinação de métrica $f(w_1) = 1 - disponibilidade$, utilizando, num primeiro caso, a distância como métrica M_1 e o número de saltos como métrica M_2, observa-se como resultado a matriz $\overline{NH}$ de "próximos nós" indicada na expressão 4.1.

Com o objetivo de fazer uma comparação de resultados, é tomado como exemplo os caminhos escolhido pelos nós 7 (SPO) e 10 (BAS) com destino ao nó 1 (BOC). Assim, segundo o nó 10 (BAS), a ordenação dos caminhos elegíveis, do melhor para o pior, é a seguinte:

$$p_{10,1} = \begin{cases} 10,11,13,1 & \text{(melhor)} \\ 10,11,13,15,1 & \\ 10,9,8,3,2,1 & \\ 10,9,8,1 & \\ 10,9,8,7,6,5,4,3,2,1 & \\ 10,9,8,5,4,3,2,1 & \text{(pior)} \end{cases} \Rightarrow \overline{S}[\overline{C}(p_{10,1})]) = \begin{cases} 10,30;3 \\ 10,60;4 \\ 13,17;5 \\ 13,27;3 \\ 13,45;9 \\ 13,77;7 \end{cases}.$$

$$\overline{NH} = \begin{bmatrix} 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 13 & 13 & 13 & 13 & 13 & 13 & 15 \\ 1 & 0 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 0 & 4 & 4 & 4 & 8 & 8 & 8 & 8 & 8 & 8 & 2 & 2 & 2 \\ 3 & 3 & 3 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 \\ 4 & 4 & 4 & 4 & 0 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 5 & 5 & 5 & 5 & 5 & 0 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 6 & 6 & 6 & 0 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 3 & 3 & 3 & 7 & 7 & 7 & 7 & 0 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 10 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 0 & 10 & 10 & 10 & 10 & 10 & 10 \\ 11 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 0 & 11 & 11 & 11 & 11 & 11 \\ 13 & 13 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 0 & 12 & 13 & 13 & 13 \\ 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 0 & 11 & 11 & 11 \\ 1 & 1 & 1 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 0 & 14 & 15 \\ 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 0 & 13 \\ 1 & 1 & 1 & 1 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 0 \end{bmatrix} \tag{4.1}$$

Da mesma forma, segundo o nó 7 (SPO), a ordenação dos caminhos elegíveis, do melhor para o pior, é:

$$p_{7,1} = \begin{cases} 7,8,3,2,1 & \text{(melhor)} \\ 7,6,5,4,3,2,1 \\ 7,8,1 \\ 7,8,5,4,3,2,1 \\ 7,6,5,8,3,2,1 \\ 7,6,5,8,1 \\ 7,8,9,10,11,13,1 \\ 7,8,9,10,11,13,15,1 \\ 7,6,5,8,9,10,11,13,1 \\ 7,6,5,8,9,10,11,13,15,1 \\ 7,6,5,4,3,8,1 \\ 7,6,5,4,3,8,9,10,11,13,1 \\ 7,6,5,4,3,8,9,10,11,13,15,1 & \text{(pior)} \end{cases} \Rightarrow \overline{S}[\overline{C}(p_{7,1})]) = \begin{cases} 10,80;4 \\ 10,88;6 \\ 10,90;2 \\ 11,40;6 \\ 12,08;6 \\ 12,18;4 \\ 12,87;6 \\ 13,17;7 \\ 14,15;8 \\ 14,45;9 \\ 18,98;6 \\ 20,95;10 \\ 21,25;11 \end{cases} .$$

Note que, embora nesse exemplo não haja caminhos com distâncias iguais, vale comentar que, entre os resultados ilustrados, nem sempre a distância mais curta é aquela com menor número de saltos.

Utilizando-se somente a métrica M_1 como distância e a abreviação $b = b_\infty$, como era esperado, obtém-se como resultado a matriz $\overline{NH}$ de "próximos nós", indicada na expressão 4.2, igual a matriz $\overline{NH}$ indicada na expressão 4.1.

$$\overline{NH} = \begin{bmatrix} 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 13 & 13 & 13 & 13 & 13 & 13 & 15 \\ 1 & 0 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 0 & 4 & 4 & 4 & 8 & 8 & 8 & 8 & 8 & 8 & 2 & 2 & 2 \\ 3 & 3 & 3 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 \\ 4 & 4 & 4 & 4 & 0 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 5 & 5 & 5 & 5 & 5 & 0 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 6 & 6 & 6 & 0 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 3 & 3 & 3 & 7 & 7 & 7 & 7 & 0 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 10 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 0 & 10 & 10 & 10 & 10 & 10 & 10 \\ 11 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 0 & 11 & 11 & 11 & 11 & 11 \\ 13 & 13 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 0 & 12 & 13 & 13 & 13 \\ 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 0 & 11 & 11 & 11 \\ 1 & 1 & 1 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 0 & 14 & 15 \\ 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 0 & 13 \\ 1 & 1 & 1 & 1 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 0 \end{bmatrix} \tag{4.2}$$

Considere agora a disponibilidade como métrica M_1 e o número de saltos como métrica M_2, observa-se como resultado a matriz $\overline{NH}$ de "próximos nós" indicada na expressão 4.3.

Utilizando como exemplo os caminhos escolhidos pelos mesmos nós 7 (SPO) e 10 (BAS) com destino ao nó 1 (BOC), segundo o nó 10 (BAS), a nova ordenação dos caminhos elegíveis, do melhor para o pior, é:

$$p_{10,1} = \begin{cases} 10,9,8,1 & \text{(melhor)} \\ 10,9,8,3,2,1 & \\ 10,9,8,5,4,3,2,1 & \\ 10,11,13,1 & \\ 10,11,13,15,1 & \\ 10,9,8,7,6,5,4,3,2,1 & \text{(pior)} \end{cases} \Rightarrow \overline{S}[\overline{C}(p_{10,1})]) = \begin{cases} 0,90;3 \\ 0,90;5 \\ 0,90;7 \\ 0,85;3 \\ 0,80;4 \\ 0,50;9 \end{cases}.$$

$$\overline{NH} = \begin{bmatrix} 0 & 2 & 2 & 2 & 8 & 8 & 8 & 8 & 8 & 8 & 13 & 13 & 13 & 13 & 15 \\ 1 & 0 & 3 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 0 & 4 & 4 & 4 & 8 & 8 & 8 & 8 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 0 & 5 & 5 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 8 & 4 & 4 & 4 & 0 & 6 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 5 & 5 & 5 & 5 & 5 & 0 & 7 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 8 & 8 & 8 & 8 & 8 & 6 & 0 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 1 & 1 & 3 & 3 & 5 & 5 & 7 & 0 & 9 & 9 & 1 & 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 0 & 10 & 8 & 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 0 & 11 & 11 & 11 & 11 & 11 \\ 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 10 & 0 & 12 & 13 & 13 & 13 \\ 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 0 & 11 & 11 & 11 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 11 & 11 & 11 & 0 & 14 & 15 \\ 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 0 & 13 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 13 & 13 & & 13 & 13 & 0 \end{bmatrix} \tag{4.3}$$

Da mesma forma, segundo o nó 7 (SPO), a ordenação dos caminhos elegíveis, do melhor para o pior, é:

$$p_{7,1} = \begin{cases} 7,8,1 & \text{(melhor)} \\ 7,8,3,2,1 \\ 7,8,5,4,3,2,1 \\ 7,6,5,8,1 \\ 7,6,5,4,3,2,1 \\ 7,6,5,4,3,8,1 \\ 7,6,5,8,3,2,1 \\ 7,8,9,10,11,13,1 \\ 7,8,9,10,11,13,15,1 \\ 7,6,5,8,9,10,11,13,1 \\ 7,6,5,4,3,8,9,10,11,13,1 \\ 7,6,5,8,9,10,11,13,15,1 \\ 7,6,5,4,3,8,9,10,11,13,15 & \text{(pior)} \end{cases} \Rightarrow \overline{S}[\overline{C}(p_{7,1})]) = \begin{cases} 0,90;2 \\ 0,90;4 \\ 0,90;6 \\ 0,70;4 \\ 0,70;6 \\ 0,70;6 \\ 0,70;6 \\ 0,75;6 \\ 0,60;7 \\ 0,45;8 \\ 0,45;10 \\ 0,40;9 \\ 0,40;11 \end{cases} .$$

Como se podia esperar, a ordenação obtida dos caminhos pela métrica de disponibilidade é diferente da ordenação obtida pela métrica de distância.

Note que para a métrica de disponibilidade, há caminhos com mesma disponibilidade, em que a métrica número de saltos é utilizada como critério de desempate. No entanto, para alguns caminhos encontrados para o nó 7 (SPO), somente a métrica número de saltos não é suficiente para distinguir os caminhos, por exemplo, para os caminhos $p'_{7,1} = 7,6,5,4,3,2,1$, $p''_{7,1} = 7,6,5,4,3,8,1$ e $p'''_{7,1} = 7,6,5,8,3,2,1$ que, embora não sejam os melhores caminhos entre SNT e BOC, são opções de caminhos em caso de uma falha na rede nos enlaces SPO - SNT.

No entanto, vale ressaltar que, conforme indicado no caso anterior para a métrica de distância, esses mesmos caminhos são ordenados, do melhor para o pior, como

$$p_{7,1} = \begin{cases} 7,6,5,4,3,2,1 & \text{(melhor)} \\ 7,6,5,8,3,2,1 \\ 7,6,5,4,3,8,1 & \text{(pior)} \end{cases}$$

e, portanto, sob este ponto de vista, eles não são iguais, justificando a necessidade da implementação de um outro critério de otimização ou da incorporação de mais

métricas de roteamento a essa estratégia de otimização.

Comparando os resultados das matrizes $\overline{NH}$ de "próximos nós", indicadas nas expressões 4.1 e 4.3, com os respectivos melhores caminhos, obtidos pelos nós 10 (SPO) e 7 (BAS) para o nó 1 (BOC), o caminho escolhido pelos nós de origem é de fato implementado, uma vez que não houve a necessidade da imposição da propriedade de coerência, isto é, o algoritmo ELND convergiu de maneira ótima.

No entanto, do ponto de vista da otimização de recursos e da QoS prestada por uma rede de transmissão internacional, espera-se que o melhor caminho seja aquele que apresente: maior disponibilidade, menor atraso, maior vazão e menor distância. Assim, implementando essa estratégia de otimização, define-se o conjunto de métricas $\mathbf{M} = \{M_1, M_2, M_3, M_4\}$, em que $M_1 = 1 - disponibilidade$, $M_2 = atraso$, $M_3 = vazão$ e $M_4 = distância$; o conjunto de funções de combinação de métricas $\mathbf{F} = \{f(c_1), f(c_2), f(c_3), f(c_4)\}$, em que $f(w_i) = c_i = w_i$ para $i = \{1, 2, 3, 4\}$; o conjunto ordenado das sínteses léxicas $\mathbf{SL} = \{aditiva, aditiva, minimizativa, aditiva\}$ com respectivas abreviações b_1; o conjunto ordenado das relações de ordenação léxica ($\preceq_L$) $\mathbf{OL} = \{\geq_L, \geq_L, \leq_L, \geq_L\}$; e uma ordenação léxica multi-dimensional $\preceq_{ML}$, tal que $\overline{S}[\overline{C}(\alpha)] \prec_{ML} \overline{S}[\overline{C}(\beta)]$ se:

$$
\begin{aligned}
& b_{l_1}(S_1[w_1(\alpha)]) >_L b_{l_1}(S_1[w_1(\beta)]) \text{ ou} \\
& \big(b_{l_1}(S_1[w_1(\alpha)]) =_L b_{l_1}(S_1[w_1(\beta)]) \text{ e} \\
& b_{l_2}(S_2[w_2(\alpha)]) >_L b_{l_2}(S_2[w_2(\beta)])\big); \\
\text{ou} \\
& \big(b_{l_1}(S_1[w_1(\alpha)]) =_L b_{l_1}(S_1[w_1(\beta)]), \\
& b_{l_2}(S_2[w_2(\alpha)]) =_L b_{l_2}(S_2[w_2(\beta)]) \text{ e} \\
& b_{l_3}(S_3[w_3(\alpha)]) <_L b_{l_3}(S_3[w_3(\beta)])\big); \\
\text{ou} \\
& \big(b_{l_1}(S_1[w_1(\alpha)]) =_L b_{l_1}(S_1[w_1(\beta)]), \\
& b_{l_2}(S_2[w_2(\alpha)]) =_L b_{l_2}(S_2[w_2(\beta)]), \\
& b_{l_3}(S_3[w_3(\alpha)]) =_L b_{l_3}(S_3[w_3(\beta)]) \text{ e} \\
& b_{l_4}(S_4[w_4(\alpha)]) >_L b_{l_4}(S_4[w_4(\beta)])\big).
\end{aligned}
$$

A matriz $\overline{NH}$ de "próximos nós", obtida nessas condições, está indicada na expressão 4.4.

$$\overline{NH} = \begin{bmatrix} 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 13 & 13 & 13 & 13 & 15 \\ 1 & 0 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 0 & 4 & 4 & 4 & 8 & 8 & 8 & 8 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 0 & 6 & 8 & 8 & 8 & 8 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 0 & 7 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 8 & 8 & 8 & 8 & 8 & 6 & 0 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 3 & 3 & 3 & 5 & 5 & 5 & 7 & 0 & 9 & 9 & 3 & 3 & 3 & 3 & 3 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 0 & 10 & 8 & 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 0 & 11 & 11 & 11 & 11 & 11 \\ 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 10 & 0 & 12 & 13 & 13 & 13 \\ 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 0 & 11 & 11 & 11 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 11 & 11 & 11 & 0 & 14 & 15 \\ 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 & 0 & 13 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 13 & 13 & 13 & 13 & 13 & 0 \end{bmatrix} \tag{4.4}$$

Segundo esse novo critério, estão listados a seguir, do melhor para o pior, os caminhos elegíveis escolhidos pelos nós 10 (SPO) e 7 (BAS) com destino ao nó 1 (BOC):

$$p_{10,1} = \begin{cases} 10,9,8,3,2,1 \\ 10,9,8,1 \\ 10,9,8,5,4,3,2,1 \\ 10,11,13,1 \\ 10,11,13,15,1 \\ 10,9,8,7,6,5,4,3,2,1 \end{cases} \Rightarrow \overline{S}[\overline{C}(p_{10,1})]) = \begin{cases} 0,90;63,70;64;13,17 \\ 0,90;64,20;64;13,27 \\ 0,90;66,70;64;13,77 \\ 0,85;49,80;128;10,30 \\ 0,80;51,20;64;10,60 \\ 0,50;65,10;64;13,45 \end{cases} \quad \text{e}$$

$$p_{7,1} = \begin{cases} 7,8,3,2,1 \\ 7,8,1 \\ 7,8,5,4,3,2,1 \\ 7,6,5,4,3,2,1 \\ 7,6,5,8,3,2,1 \\ 7,6,5,8,1 \\ 7,6,5,4,3,8,1 \\ 7,8,9,10,11,13,1 \\ 7,8,9,10,11,13,15,1 \\ 7,6,5,8,9,10,11,13,1 \\ 7,6,5,4,3,8,9,10,11,13,1 \\ 7,6,5,8,9,10,11,13,15,1 \\ 7,6,5,4,3,8,9,10,11,13,15,1 \end{cases} \Rightarrow \overline{S}[\overline{C}(p_{7,1})]) = \begin{cases} 0,90;52,20;64;10,8 \\ 0,90;52,70;64;10,9 \\ 0,90;55,20;64;11,4 \\ 0,70;55,60;64;10,88 \\ 0,70;52,40;64;12,08 \\ 0,70;58,90;64;12,18 \\ 0,70;91,70;64;18,98 \\ 0,65;62,30;64;12,87 \\ 0,60;63,70;128;13,17 \\ 0,45;68,50;64;14,15 \\ 0,45;101,30;64;20,95 \\ 0,40;69,90;64;14,45 \\ 0,40;102,70;64;21,25 \end{cases}$$

Note que, nesse caso, os melhores caminhos escolhidos pelos nós 10 (SPO) e 7 (BAS) com destino ao nó 1 (BOC) são diferentes dos obtidos nos casos anteriores. Cabe comentar que, comparando os resultados das matrizes $\overline{NH}$ de "próximos nós" com o melhor caminho obtido pelos nós 10 (SPO) e 7 (BAS) para o nó 1 (BOC), o caminho escolhido pelos nós de origem é de fato implementado, uma vez que não houve a necessidade da imposição da propriedade de coerência, ou seja, o algoritmo ELND convergiu de maneira ótima.

Portanto, os critérios impostos por uma rede de transmissão de longa distância ao roteamento multi-restritivo são completamente atendidos pela álgebra de caminhos proposta e implementada pelo algoritmo ELND, com resultados melhores do que os obtidos pela estratégia de otimização adotada pelos equipamentos GMPLS. Vale ressaltar que, entre os equipamentos de transmissão GMPLS atualmente disponíveis no mercado, a suposição feita para a estratégia de otimização multi-restritiva deste estudo de caso é bastante realista.

4.2 Rede Internet2

A rede Internet2, conforme ilustrada na Figura 4.3, é uma rede criada, para a comunidade de pesquisa e educação, com objetivo de permitir o desenvolvimento

de novas tecnologias de redes e protocolos dentro de uma infra-estrutura controlada. Sua implementação pode ser dividida em duas camadas: uma camada de rede IP, composta por equipamentos Juniper, e uma camada comutação dinâmica de circuitos ópticos, composta por equipamentos Ciena e Infinera [92].

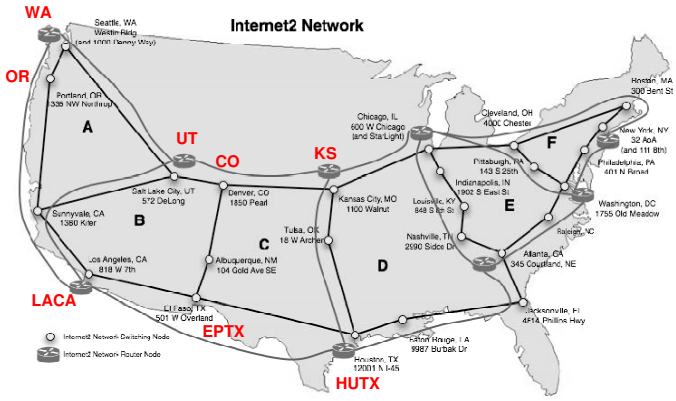


Figura 4.3: Diagrama ilustrativo da rede Internet2

Para esse estudo de caso, considere: uma abstração da rede Internet2, conforme ilustrado na Figura 4.4; um cenário em que as buscas de caminhos são realizadas separadamente pela camada óptica e pela camada de rede; e uma aplicação que usa a camada de rede e é sensível à variação do atraso, medida com relação a um atraso referencial, ajustado para o caminho escolhido pela camada de rede com base em uma estratégia de otimização que busca, como melhor caminho, o caminho que apresenta a menor ocupação, o menor atraso e o menor número de saltos. Portanto, do ponto de vista da camada de rede, as métricas de roteamento adotadas são: a ocupação dos enlaces, o atraso e o número de saltos.

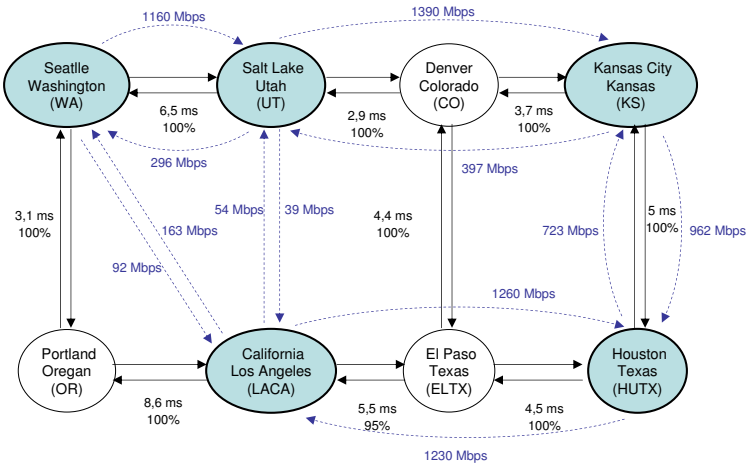


Figura 4.4: Dígrafo da abstração parcial da rede Internet2

Por outro lado, considere uma rede óptica com capacidade de autoconfiguração, em que a estratégia de otimização adota inicialmente como métricas

a disponibilidade e o atraso da rede, sendo definido que o melhor caminho é aquele que apresenta, na seqüência, a menor indisponibilidade[2] e o menor atraso.

Dessa forma, do ponto de vista da rede IP, os valores das métricas adotadas para cada enlace vão depender dos caminhos escolhidos pela camada óptica para conectar os roteadores adjacentes.

Adotando-se para a camada óptica a matriz de adjacências $\overline{Adj_o}$; as matrizes $\overline{M_o}[1]$ e $\overline{M_o}[2]$ para os respectivos valores estimados das métricas de indisponibilidade e atraso (em milisegundos) em cada enlace; o conjunto de sínteses léxicas $\mathbf{SL} = \{\text{aditiva, aditiva}\}$; o conjunto de correspondentes abreviações $\mathbf{b} = \{b_1, b_1\}$; e o conjunto de relações de ordenações léxicas $\preceq_L$ dado por $\mathbf{OL} = \{\geq_L, \geq_L\}$, tal que:

$$\overline{Adj_o} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\overline{M_o}[1] = \begin{bmatrix} 1 & 0,0 & 1 & 1 & 1 & 1 & 1 & 0,0 \\ 0,0 & 1 & 0,0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0,0 & 1 & 0,0 & 1 & 0,0 & 1 & 1 \\ 1 & 1 & 0,0 & 1 & 0,0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0,0 & 1 & 0,0 & 1 & 1 \\ 1 & 1 & 0,0 & 1 & 0,0 & 1 & 0,05 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0,05 & 1 & 0,0 \\ 0,0 & 1 & 1 & 1 & 1 & 1 & 0,0 & 1 \end{bmatrix} \quad \text{e}$$

[2]Nos estudos de caso apresentados neste capítulo, considera-se como indisponibilidade a probabilidade de falha de um enlace.

$$\overline{M_o}[2] = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty & \infty & \infty & 3,1 \\ 6,5 & \infty & 2,9 & \infty & \infty & \infty & \infty & \infty \\ \infty & 2,9 & \infty & 3,7 & \infty & 4,4 & \infty & \infty \\ \infty & \infty & 3,7 & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 5 & \infty & 4,5 & \infty & \infty \\ \infty & \infty & 4,4 & \infty & 4,5 & \infty & 5,5 & \infty \\ \infty & \infty & \infty & \infty & \infty & 5,5 & \infty & 8,6 \\ 3,1 & \infty & \infty & \infty & \infty & \infty & 8,6 & \infty \end{bmatrix}.$$

A matriz de "próximos nós" obtida, $\overline{NH_o}$, está indicada na expressão 4.5, a seguir:

$$\overline{NH_o} = \begin{bmatrix} 0 & 2 & 2 & 2 & 2 & 2 & 8 & 8 \\ 1 & 0 & 3 & 3 & 3 & 3 & 1 & 1 \\ 2 & 2 & 0 & 4 & 4 & 6 & 2 & 2 \\ 3 & 3 & 3 & 0 & 5 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 0 & 6 & 4 & 4 \\ 3 & 3 & 3 & 3 & 5 & 0 & 3 & 3 \\ 8 & 8 & 8 & 8 & 8 & 8 & 0 & 8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 7 & 0 \end{bmatrix}, \tag{4.5}$$

em que os nós aparecem enumerados conforme segue:

- Nó 1: WA (Seatlle - Washington),
- Nó 2: UT (Salt Lake - Utah),
- Nó 3: CO (Denver - Colorado),
- Nó 4: KS (Kansas City - Kansas),
- Nó 5: HUTX (Houston - Texas),
- Nó 6: ELTX (El Paso - Texas),
- Nó 7: LACA (California - Los Angeles) e
- Nó 8: OR (Portland- Oregon).

Utilizando, como exemplo, os caminhos escolhidos pelo nó 2 (UT) com destino ao nó 7 (LACA), cuja ordenação dos caminhos elegíveis, do melhor para o pior,

é:

$$p_{2,7} = \begin{cases} 2,1,8,7 & \text{(melhor)} \\ 2,3,6,7 & \\ 2,3,4,5,6,7 & \text{(pior)} \end{cases} \Rightarrow \overline{S}[\overline{C}(p_{2,7})]) = \begin{cases} 0;18,2 \\ 0,05;12,8 \\ 0,05;21,6 \end{cases},$$

quando o melhor caminho entre os nós 2 (UT) e 7 (LACA) é interrompido, a próxima melhor opção de indisponibilidade é dada pelos caminhos: $p_{7,1} = 2,3,6,7$ e $p_{7,1} = 2,3,4,5,6,7$ com igual valor de 0,05% de indisponibillidade. No entanto, pela métrica de atraso, tem-se que $p_{7,1} = 2,3,4,5,6,7 \prec p_{7,1} = 2,3,6,7$.

Entretanto, segundo essa ordenação, do ponto de vista da aplicação que utilizará a rede, a variação o atraso entre o melhor caminho, inicialmente escolhido, $p_{7,1} = 2,1,8,7$ e o segundo melhor caminho $p_{7,1} = 2,3,6,7$ é de 5,4 ms, e a variação do atraso entre o melhor caminho $p_{7,1} = 2,1,8,7$ e o terceiro melhor caminho $p_{7,1} = 2,3,4,5,6,7$ é de -3,4 ms. Assim, para a aplicação, o segundo melhor caminho escolhido pela camada óptica é pior que o terceiro melhor caminho e, portanto, a camada ótica deveria considerar também, como métrica de re-roteamento, a variação relativa do atraso e não somente o atraso absoluto. Observe que esse cenário ocorre quando há uma falha na rede e a sua solução envolve, entre outras coisas, a definição de uma estratégia de restauração, cuja análise foge do escopo desse trabalho e foi deixada para estudos futuros.

Utilizando para a camada de rede a matriz de adjacências $\overline{Adj_r}$; as matrizes $\overline{M_r}[1]$, $\overline{M_r}[2]$, e $\overline{M_r}[3]$ para os respectivos valores das métricas de ocupação (em Mbps), atraso (em ms) e número de saltos, obtidos a partir da escolha dos caminhos selecionados pela camada óptica; o conjunto de sínteses léxicas **SL** = {maximizativa, aditiva, aditiva}; o conjunto de correspondentes abreviações $\mathbf{b} = \{b_1, b_1, b_1\}$; e o conjunto de relações de ordenações léxicas $\preceq_L$ dado por $\mathbf{OL} = \{\geq_L, \geq_L, \geq_L\}$, tal que:

$$\overline{Adj_r} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix},$$

$$\overline{M_r}[1] = \begin{bmatrix} \infty & 1160 & \infty & \infty & 92 \\ 296 & \infty & 1390 & \infty & 39 \\ \infty & 397 & \infty & 962 & \infty \\ \infty & \infty & 723 & \infty & 1230 \\ 163 & 54 & \infty & 1260 & \infty \end{bmatrix},$$

$$\overline{M_r}[2] = \begin{bmatrix} \infty & 6,5 & \infty & \infty & 11,7 \\ 6,5 & \infty & 6,6 & \infty & 18,2 \\ \infty & 6,6 & \infty & 5 & \infty \\ \infty & \infty & 5 & \infty & 29,8 \\ 11,7 & 18,2 & \infty & 29,8 & \infty \end{bmatrix} \text{ e}$$

$$\overline{M_r}[3] = \begin{bmatrix} \infty & 1 & \infty & \infty & 1 \\ 1 & \infty & 1 & \infty & 1 \\ \infty & 1 & \infty & 1 & \infty \\ \infty & \infty & 1 & \infty & 1 \\ 1 & 1 & \infty & 1 & \infty \end{bmatrix},$$

a matriz obtida de "próximos nós", $\overline{NH_r}$, está indicada na expressão 4.6, em que os nós aparecem enumerados conforme segue:

- Nó 1: WA (Seatlle - Washington),
- Nó 2: UT (Salt Lake - Utah),
- Nó 3: KS (Kansas City - Kansas),
- Nó 4: HUTX (Houston - Texas) e
- Nó 5: LACA (California - Los Angeles).

$$\overline{NH_r} = \begin{bmatrix} 0 & 5 & 5 & 5 & 5 \\ 5 & 0 & 5 & 5 & 5 \\ 2 & 2 & 0 & 4 & 2 \\ 3 & 3 & 3 & 0 & 3 \\ 1 & 2 & 4 & 4 & 0 \end{bmatrix}, \tag{4.6}$$

Assim, conforme observado, a flexibilidade obtida pela álgebra de caminhos proposta permite analisar de maneira integrada as diferentes estratégias de otimização adotadas, não só para a mesma camada de rede, mas também para as diferentes camadas de rede existentes, conforme definido na arquitetura de rede ASON/ GMPLS, e, dessa forma, validar uma solução de otimização de rede mais consistente com a disponibilidade total dos recursos da rede e da qualidade final dos serviços prestados por ela.

5 Conclusão e Trabalhos Futuros

Neste trabalho foi apresentada uma nova estrutura matemática para a álgebra de caminhos, que incorpora e flexibiliza as álgebras de caminhos disponíveis na literatura até o momento, permitindo a reanálise da garantia de convergência dos algoritmos de roteamento multi-restritivos. Analisando sob o ponto de vista da engenharia de tráfego e da QoS na arquitetura GMPLS, esse novo arcabouço matemático proposto permite, de maneira confiável, a incorporação de novas métricas de roteamento e novas formas de computar a escolha dos melhores caminhos nos algoritmos de roteamento *hop-by-hop*.

Por meio dessa nova álgebra de caminhos, constatou-se que as propriedades de isotonicidade estrita e liberdade são condições de suficiência para a garantia da convergência dos algoritmos de roteamento *hop-by-hop* e, ao contrário do indicado na literatura até o momento, a propriedade de monotonicidade não é condição necessária e nem suficiente para isso. Sendo assim, esta tese propõe uma nova propriedade denominada coerência, que, no paradigma de roteamento *hop-by-hop*, é demonstrada, por meio de um teorema, ser a condição necessária e suficiente para garantir a convergência dos algoritmos de roteamento mono/ multi-restritivos.

No contexto do roteamento *hop-by-hop*, este trabalho analisa e constata duas formas de definir a escolha do melhor caminho: a escolha feita pelo nó de origem e a escolha feita pelo nó de destino, concluindo que, no segundo caso, sempre se obtém um algoritmo de roteamento *hop-by-hop* com garantia de coerência, independentemente das métricas utilizadas e das formas de computar seus valores ao longo dos caminhos.

Buscando simular os resultados teóricos apresentados nesta tese, foi desenvolvido um programa de simulação denominado ELND e, por meio de um exemplo didático e de dois estudos de casos, pôde ser ilustrada a aplicabilidade da nova álgebra proposta no contexto das redes GMPLS, bem como a garantia da

convergência pela imposição da propriedade de coerência proposta. Como o programa ELND permite definir diferentes tipos de métrica, critérios de ordenação de caminhos e formas de combinação e de síntese dessas métricas ao longo de todos os caminhos simples e elementares, que unem qualquer par de nós origem-destino em uma rede, este trabalho apresenta também uma ferramenta computacional de validação das diferentes estratégias de otimização definidas para uma rede, possibilitando, com isso, a revisão de possíveis equívocos ou inconsistências detectadas antes da implementação dessas estratégias nas redes operacionais.

No desenvolvimento deste trabalho, dentro do contexto das redes GMPLS, detectou-se pelo menos duas necessidades de análise de aplicação, com o uso da álgebra de caminhos proposta, que ficam como sugestão para a continuidade deste trabalho: a reavaliação das estratégias do roteamento multi-restritivo, voltado à qualidade dos serviços finais prestados pela rede GMPLS multi-tecnológica, no paradigma da estratificação da rede em camadas clientes e servidoras, propostos pela arquitetura T-MPLS, tanto no estado normal de operação da rede, quanto nos estados de restauração; o desenvolvimento de uma estratégia de reavaliação dinâmica da ocupação dos recursos configurados na rede, buscando a maximização da sua disponibilidade para a ativação de novos serviços; uma reavaliação das estratégias de roteamento atualmente implementada entre diferentes sistemas autônomos, que utilizam o *Border Gateway Protocol* (BGP) como protocolo de roteamento, não só no contexto das redes IP, como também no contexto das redes GMPLS; e o desenvolvimento de uma ferramenta de simulação com um tempo de processamento mais rápido, não só para a análise do roteamento mono/ multi-restritivo *hop-by-hop* com estratégia de otimização definida pelo nó de origem, como também para a análise do roteamento mono/ multi-restritivo *hop-by-hop* com estratégia de otimização definida pelo nó de destino.

Referências

[1] Miniwatts Marketing Group. *Internet World Stats*. 2008. Disponível em: <http://www.internetworldstats.com/stats.htm>. Acesso em: mar. 2008.

[2] GANESH, A. J. et al. Congestion notification and probing mechanisms for endpoint admission control. *IEEE/ACM Transactions on Networking*, v. 14, p. 568–578, jun. 2006.

[3] HUANG, Y. et al. Supporting excess real-time traffic with active drop queue. *IEEE/ACM Transactions on Networking*, v. 14, p. 965–977, out. 2006.

[4] SMITH, B. R.; GARCIA-LUNA-ACEVES, J. J. *Quality of Service in the Emerging Networking Panorama*. Heidelberg, Alemanha: Springer Berlin / Heidelberg, 2008. 268-277 p. (Lecture Notes in Computer Science).

[5] WANG, X.; SCHULZRINNE, H. Pricing network resources for adaptive applications. *IEEE/ACM Transactions on Networking*, v. 14, p. 506–519, jun. 2006.

[6] CHENG, Y.; ZHUANG, W. Dynamic inter-SLA resource sharing in path-oriented differentiated services networks. *IEEE/ACM Transactions on Networking*, v. 14, p. 657–670, jun. 2006.

[7] SHENKER, S. et al. *RFC 2212 - Specification of Guaranteed Quality of Service*. 1997.

[8] WROCLAWSKI, J. *RFC 2211 - Specification of the Controlled-Load Network Element Service*. 1997.

[9] AWDUCHE, D. et al. *RFC 3272 - Overview and Principles of Internet Traffic Engineering*. 2002.

[10] BERNET, Y. et al. *RFC 2998 - A Framework for Integrated Services Operation over Diffserv Networks*. 2000.

[11] OIF-UNI-01.0 - User Network Interface 1.0 Signaling Specification, Release 2. 2004.

[12] MEF 11 - User Network Interface Requirements and Framework - Technical Specification. 2004.

[13] OSBORNE, E.; SIMHA, A. *Traffic Engineering with MPLS - Design, configuration, and management MPLS TE to optimize network performance*. Primeira edição. Indiana, EUA: Cisco Press, 2003.

[14] AWDUCHE, D. et al. *RFC 2702 - Requirements for Traffic Engineering Over MPLS*. 1999.

[15] PAPADIMITRIOU, D.; VERCHERE, D. GMPLS user-network interface in support of end-to-end rerouting. *IEEE Communications Magazine*, v. 43, p. 35–43, jul. 2005.

[16] ITU-T G.8080 - Architecture for the automatically switched optical network (ASON). 2006.

[17] ITU-T G.8110.1 - Architecture of Transport MPLS (T-MPLS) Layer Network. 2006.

[18] ITU-T G.8112 - Interfaces for the Transport MPLS (T-MPLS) hierarchy. 2006.

[19] KUIPERS, F. et al. An overview of constraint-based path selection algorithms for QoS routing. *IEEE Communications Magazine*, v. 40, p. 50–55, dez. 2002.

[20] NEVES, H. D.; MIEGHEM, P. V. Tamcra: A tunable accuracy multiple constraint routing algorithm. *Computer Communications*, v. 23, p. 667–679, mar. 2000.

[21] MIEGHEM, P. V. et al. Hop-by-hop quality of service routing. *Computer Networks*, v. 37, p. 407–423, nov. 2001.

[22] XUE, G. et al. Finding a path subject to many additive QoS constraints. *IEEE/ACM Transactions on Networking*, v. 15, p. 201–211, fev. 2007.

[23] GOUDA, M. G.; SCHNEIDER, M. Maximizable routing metrics. *IEEE/ACM Transactions on Networking*, v. 11, n. 4, p. 663–675, ago. 2003.

[24] SOBRINHO, J. L. Network routing with path vector protocols: Theory and applications. Em: ASSOCIATION FOR COMPUTING MACHINERY. *SIGCOMM 2003 - Special Interest Group on Data Communication*. Karlsruhe, Alemanha, 2003. p. 49–60.

[25] WATTENHOFER, R.; VENKATACHARY, S. The impact of internet policy and topology on delayed routing convergence. Em: IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. *INFOCOM 2001 - 20th Annual Joint Conference of the IEEE Computer and Communications Societies*. Anchorage, Alaska, EUA, 2001. p. 537–546.

[26] MIYAMURA, T. et al. Enhancing the network scalability of link-state routing protocols by reducing their flooding overhead. Em: IEEE. *Workshop on High Performance Switching and Routing*. Torino, Italia, 2003. p. 263–268.

[27] PEI, D. et al. A study of BGP path vector route looping behavior. Em: IEEE COMPUTER SOCIETY. *Proceedings of the 24th International Conference on Distributed Computing Systems*. Tóquio, Japão, 2004. p. 720–729.

[28] LAGOA, C. M. et al. Adaptive control algorithms for decentralized optimal traffic engineering in the internet. *IEEE/ACM Transactions on Networking*, v. 12, p. 415–428, jun. 2004.

[29] JAGGARD, A. D.; RAMACHANDRAN, V. Relating two formal models of path-vector routing. Em: IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. *INFOCOM 2005 - 24th Annual Joint Conference of the IEEE Computer and Communications Societies*. Miami, FL, EUA, 2005. v. 1, p. 619–630.

[30] FLEMING, H. O último trabalho de Mário Schenberg. *Revista USP*, v. 50, jul. 2001. Disponível em: <http://www.hfleming.com/schenberg.shtml>.

[31] ABE, J. M. A noção de estrutura em matemática e física. *Estudos Avançados*, v. 3, p. 113–125, ago. 1989.

[32] HOUAISS, A.; VILLAR, M. S. *Houaiss da língua portuguesa*. Rio de Janeiro, Brasil: Editora Objetiva Ltda, 2007.

[33] WELLS, C. *Mathematical Structures*. jul. 2007. Disponível em: <http://www.abstractmath.org/MM/MMMathStructure.htm>. Acesso em: mar. 2008.

[34] CARRÉ, B. *Graphs and Networks*. Oxford, Inglaterra: Oxford University Press, 1979. (Oxford Applied Mathematics and Computing Science Series).

[35] SOBRINHO, J. L. Algebra and algorithms for QoS path computation and hop-by-hop routing in the internet. *IEEE/ACM Transactions on Networking*, v. 10, n. 4, p. 541–550, ago. 2002.

[36] SOBRINHO, J. L. An algebraic theory of dynamic network routing. *IEEE/ACM Transactions on Networking*, v. 13, n. 5, p. 1160–1173, out. 2005.

[37] Wikimedia Foundation. *Supremo e ínfimo*. 2008. Disponível em: <http://pt.wikipedia.org/wiki/SupremoeInfimo>. Acesso em: mar. 2008.

[38] SILVA, C. M. S. da. *No paraíso dos símbolos: surgimento da lógica e teoria dos conjuntos no Brasil*. 2008. Disponível em: <http://www.ufes.br/circe/artigos/artigo61.htm>. Acesso em: mar. 2008.

[39] BURRIS, S.; SANKAPPANAVAR, H. P. *A Course in Universal Algebra*. California, EUA: Springer-Verlag, 1981.

[40] NETTO, P. O. B. *Grafos: Teoria, Modelos, Algaritmos*. Quarta edição. São Paulo, Brasil: Edgard Blücher, 2006.

[41] CORMEN, T. H. et al. *Introduction to Algorithms*. Massachusetts, EUA: Massachusetts Institute of Technology, 2001.

[42] Wikimedia Foundation. *Algorithm*. 2008. Disponível em: <http://en.wikipedia.org/wiki/Algorithm>. Acesso em: mar. 2008.

[43] KNUTH, D. E. *The Art of Computer Programming*. Terceira edição. Massachusetts, EUA: Addison Wesley Longman, 1997. (Fundamental Algorithms, v. 1).

[44] DIJKSTRA, E. W. A note on two problems in connection with graphs. *Numerische Mathematik*, v. 1, p. 269–271, 1959.

[45] GALLO, G.; PALLOTTINO, S. *Shortest path algorithms*. Holanda: Springer Netherlands, 1998. 1-79 p. (Business and Economics, v. 3).

[46] DIAL, R. B. et al. Algorithm 360: shortest-path forest with topological ordering. *Communications of the ACM*, v. 12, p. 632–633, nov. 1969.

[47] DENARDO, E. V.; FOX, B. L. Shortest-route methods: Reaching, pruning, and buckets. *Operations Research*, v. 27, p. 161–186, jan. 1979.

[48] CHERKASSKY, B. V. et al. Shortest paths algorithms: Theory and experimental evaluation. Em: SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS. *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. San Diego, EUA, 1994. p. 516–525.

[49] CHERKASSKY, B. V. et al. Negative-cycle detection algorithms. Em: EUROPEAN ASSOCIATION FOR THEORETICAL COMPUTER SCIENCE. *Proceedings of the Fourth Annual European Symposium on Algorithms*. Barcelona, Espanha, 1996. p. 349–363.

[50] AHUJA, R. et al. Faster algorithms for the shortest path problem. *Journal of the Association for Computing Machinery*, v. 37, p. 213–223, abr. 1990.

[51] NOSHITA, K. A theorem on the expected complexity of Dijkstra's shortest path algorithm. *Journal of Algorithms*, v. 6, p. 400–408, set. 1985.

[52] RAMAN, R. *Fast Algorithms for Shortest Paths and Sorting*. Londres, Inglaterra, mar. 1996. King's Colledge London Report.

[53] MORET, B. M.; SHAPIRO, H. D. *Algorithms and Data Structures*. Heidelberg, Alemanha: Springer Berlin / Heidelberg, 2006. 400-411 p. (Lecture Notes in Computer Science, v. 519/1991).

[54] GOLDBERG, A. V. *A Simple Shortest Path Algorithm with Linear Average Time*. EUA, 2001. STAR Laboratory, InterTrust Technologies Corp Report.

[55] FORD, J. L. R.; FULKERSON, D. R. *Flows in Networks*. New Jersey, EUA: Princeton University Press, 1962.

[56] FENG, B. et al. *IP-based Routing and Signaling Protocols for the Optical Transport Network*. Heidelberg, Alemanha, abr. 2003. Scalable Optical IP Transport Networks (SCORPION) Report.

[57] WANG, Y.; WANGS, Z. Explicit routing algorithms for internet traffic engineering. Em: IEEE. *Eight International Conference on Computer Communications and Networks - Proceedings*. Boston, Massachusetts, EUA, 1999. p. 582 – 588.

[58] SEOK, Y. et al. Explicit multicast routing algorithms for constrained traffic engineering. Em: IEEE COMPUTER SOCIETY. *Seventh International Symposium on Computers and Communications (ISCC'02)*. Giardini Naxos, Italia, 2002. p. 455.

[59] ROSEN, E. et al. *RFC 3031 - Multiprotocol Label Switching Architecture*. 2001.

[60] YANG, Y. et al. Designing routing metrics for mesh networks. Em: IEEE. *Proceedings of First Workshop on Wireless Mesh Networks*. Santa Clara, CA, EUA, 2005.

[61] RIEDL, A. A versatile genetic algorithm for network planning. Em: TECHNISCHE UNIVERSITÄT MÜNCHEN. *EUNICE-98: Open European Summer School on Network Management and Operations*. Munique, Alemanha, 1998.

[62] NURMINEN, J. K. *Models and Algorithms for Network Planning Tools - Pratical Experiences*. Helsinki, Alemanha, maio 2003. Helsinki University of Technology Report.

[63] LUCIO, G. F. et al. Guided local search as a network planning algorithm that incorporates uncertain traffic demands. *The International Journal of Computer and Telecommunications Networking*, v. 51, p. 3172–3196, ago. 2007.

[64] HASAN, M.; ATI, R. *Management of OIF-UNI, GMPLS, and ASON Enabled Networks*. maio 2004. Disponível em: <http://www-nrc.nokia.com/mail-archive/irtf-smrg/pdf00006.pdf>.

[65] Cisco Systems. *Cisco Transport Manager 8.5 - Product Overview*. 2008. Disponível em: <http://www.cisco.com/en/US/prod/collateral/netmgtsw/ps5730/ps2204/ps7337/product_data_sheet0900aecd80709ff2.>. Acesso em: mar. 2008.

[66] LAMMLE, T. *Cisco Certificated Network Associated - Study Guide*. Segunda edição. California, EUA: SYBEX Inc., 2000.

[67] DOYLE, J. *CCIE Professional Development Routing TCP/IP*. Indianapolis, EUA: Macmillan Technical Publishing, 1998.

[68] LI, B. et al. Routing and wavelength assignment vs. wavelength converter placement in all optical networks. *IEEE Communications Magazine*, v. 41, p. S22–S28, ago. 2003.

[69] KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet*. São Paulo, Brasil: Pearson Addison Wesley, 2006.

[70] RESENDE, M. G. C.; PARDALOS, P. M. *Handbook of Optimization in Telecommunications*. New York, EUA: Springer Science + Business Media, Inc., 2006.

[71] HARARY, R. Z. N. F.; CARTWRIGHT, D. *Structural Models - An Introduction to the Theory of Directed Graphs*. New Jersey, EUA: John Wiley & Sons, Inc., 1965.

[72] Wikimedia Foundation. *Convergence*. 2008. Disponível em: <http://en.wikipedia.org/wiki/Convergence>. Acesso em: mar. 2008.

[73] Wikimedia Foundation. *Operação*. 2008. Disponível em: <http://pt.wikipedia.org/wiki/Operação>. Acesso em: mar. 2008.

[74] BRÁZIO J. L. SOBRINHO, L. M. M.; BYCZEK, L. Destructive effect of acknowledgment traffic in WLANs. Em: IEEE COMMUNICATIONS SOCIETY. *GLOBECOM 2006 - IEEE Global Telecommunications Conference*. California, EUA, 2006.

[75] Wikimedia Foundation. *Metric space*. 2008. Disponível em: <http://en.wikipedia.org/wiki/Metricspace>. Acesso em: mar. 2008.

[76] SEARCÓID, M. O. *Metric Spaces*. Londres, Inglaterra: Springer-Verlag, 2007. (Springer Undergraduate Mathematics Series).

[77] Wikimedia Foundation. *Metrics networking*. 2008. Disponível em: <http://en.wikipedia.org/wiki/Metricsnetworking>. Acesso em: mar. 2008.

[78] MIRAS, D. *Network QoS Needs of Advanced Internet Applications - A Survey*. Londres, Inglaterra, 2002. Computer Science Department University College London - Report.

[79] STRAND, J. et al. Issues for routing in the optical layer. *IEEE Communications Magazine*, v. 39, p. 81–87, fev. 2001.

[80] JUKAN, A.; FRANZL, G. Path selection methods with multiple constraints in service-guaranteed WDM networks. *IEEE/ACM Transactions on Networking*, v. 12, n. 1, p. 59–72, fev. 2004.

[81] DUFFIELD, N.; PRESTI, F. L. Multicast inference of packet delay variance at interior network links. Em: IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. *INFOCOM 2000 - 19th Annual Joint Conference of the IEEE Computer and Communications Societies*. Tel-Aviv, Israel, 2000. p. 1351–1360.

[82] KOBAYASHI, K.; KATAYAMA, T. Analysis and evaluation of packet delay variance in the internet. *IEICE Transactions on Communications*, E85-B, p. 35–42, jan. 2002.

[83] SU, P.; GELLMAN, M. Using adaptive routing to achieve quality of service. *Performance Evaluation*, v. 57, p. 105–119, jun. 2004.

[84] MIEGHEM, P. V.; KUIPERS, F. A. Concepts of exact QoS routing algorithms. *IEEE/ACM Transactions on Networking*, v. 12, p. 415–428, jun. 2004.

[85] LI, Z.; GARCIA-LUNA-ACEVES, J. Loop-free constrained path computation for hop-by-hop QoS routing. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, v. 51, p. 3278–3293, ago. 2007.

[86] SARAVIA, E. et al. Advanced enhancements of the SAm-1 submarine cable system. Em: SUBMARINE TELECOMS FORUM. *SubOptic 2007*. Baltimore, EUA, 2007.

[87] Nortel Networks. *Nortel Optical Cross Connect HDX and HDXc - Product Brief*. 2008. Catálogo.

[88] XU, Y. et al. Generalized MPLS-based distributed control architecture for automatically switched transport networks. *Bell Labs Technical Journal*, v. 6, p. 34–49, jan. 2001.

[89] GUÉRIN, R. A.; ORDA, A. Computing shortest paths for any number of hops. *IEEE/ACM Transactions on Networking*, v. 10, p. 613–620, out. 2002.

[90] RAMASWAMI, R.; SIRVARAJAN, K. N. *Optical Networks - A Practical Perspective*. San Diego, EUA: Morgan Kaufmann Publishers, 1998.

[91] ITU-T G.692 - Optical interfaces for multi-channel systems with optical amplifiers. 1998.

[92] Internet2 Networking Consortium. *Internet2 Network*. 2008. Disponível em: <http://www.internet2.edu/network>. Acesso em: mar. 2008.

Anexo A – Código Fonte do Algoritmo ELND

A.1 Rotina Principal

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rotina principal (main.m): para o cálculo da convergência,   %
% em que:                                                      %
%  - A topologia da rede é modelada em uma Matriz T em que as: %
%     - linhas são a origem das conexões;                      %
%     - colunas são o destino das conexões;                    %
%  - Os valores das metricas para cada enlace é modelado em uma%
%    matriz M em que o array;                                  %
%     - M{m}corresponde a matriz da métrica m onde m pode ser  %
%       1 a 10 por exemplo => 10 métricas;                     %
%  - Nn=length(T) é o número total de nós da rede;             %
%  - Os caminhos obtidos entre "o" e "d":                      %
%  - P{o}{d}={[o, ..., d],....,[o, ..., d]} em que:            %
%     P{o}{d}={[ ]} para o=d;                                  %
%  - Os vetores dos pesos dos enlaces dos caminhos obtidos em  %
%     P{o}{d}{:} é dado por:                                   %
%     - w{m}{o}{d}(:) para m=1,2,..ou., m onde m é a metrica   %
%       escolhida;                                             %
% - F é o array que contém para cada metrica a síntese e rela- %
%   ção de ordem a ser aplicada F{m}={'síntese', 'relação'}:   %
%   Ex: F{1}={'MIN','>'}; F{2}={'MAX','<'}; F{3}={'MUL','>'}   %
%   e F{M}={'ADD','<'}; em que size(F,2)=size(M,2)= número     %
%   de métricas usadas;                                        %
% - A relação de ordem léxica multidimensional é aplicada na   %
%   sequencia em que se define o array F, ou seja:             %
%  - F{1} tem prioridade sobre F{2};                           %
```

```
27  %  - F{2} tem prioridade sobre F{3}, etc.;                          %
28  % - size(M{1})=size(M{2})=size(M{3})=...size(M{m})= size(T)=        %
29  %   size(F,2)=size(M,2)= número de métricas usadas;                 %
30  % - word: é o array com as abreviações utilizadas da palavra-       %
31  %   peso;                                                           %
32  %  - word{m}=l l letras para a palavra-peso da síntese m; e         %
33  %  - word{m}=0 significa palavra-peso sem abreviação.               %
34  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35  [Nn,I] = buscapath(T);
36  [P,w] = separapath(M,I,Nn);
37  [Po,So] = calculapath(F,P,w,Nn,word);
38  [Pfo,Sfo] = ordenapaths(F,Po,So,Nn);
39
40  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41  % Gera array de next hops.                                           %
42  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
43  [Nh,Nht] = geraAnexthop(Pfo,Nn);
44
45  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
46  % Gera matriz de next hop(NH)para os melhores caminho no             %
47  % estado inicial da rede e salva estado inicial da rede.             %
48  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49  [NH] = geraMnexthop(Nh,Nn,1);
50  NHo=NH;
51
52  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53  % Calcula árvore de destino a partir da matriz de next hop NH. %
54  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55  [Cd] = calcdtree(NH,Nn);
56
57
58  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
59  % Entra em busca de loop até a rede convergir, eliminado os          %
60  % loops encontrados.                                                 %
61  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
62  loop=1;
63  E=0;
64  while loop~=0 & E<100
```

```
loop=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Varre nós de destinos para verificar se há loop.         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for d=1:Nn
   C=Cd{d};
   [NC,LO,NA] = buscaloop(C,d,Nh,Nht);
   if ~isempty(NC)

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      % Se houver loop verifica os nós que devem alterados e%
      % evitados.                                           %
      %  size(NC,1)=#total de loops existentes.             %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      for o=1:size(NC,1)
         if NC(o)~=0

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Verifica se loop pode ser eliminado e se pode,%
            % altera a matriz next hop (NH) para o próximo  %
            % estado da rede e elimina da lista de next hop %
            % o nó escolhido.                               %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            NH(NC(o),d)=NA(o);
            i=0;
            for l=1:Nht{NC(o)}{d}
               if Nh{NC(o)}{d}{l}==NA(o)
                  i=l;
               end
            end
            if i==1
               Nh{NC(o)}{d}= Nh{NC(o)}{d}(2:Nht{NC(o)}{d});
            elseif i==Nht{NC(o)}{d}
               Nh{NC(o)}{d}= Nh{NC(o)}{d}(1:Nht{NC(o)}{d}-1);
            else
               a=cell(1,1); b=cell(1,1);
               a=Nh{NC(o)}{d}(1:i-1);
```

```
                  b=Nh{NC(o)}{d}(i+1:Nht{NC(o)}{d});
                  Nh{NC(o)}{d}=[];
                  Nh{NC(o)}{d}=[a b];
               end

               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               % Reduz o número total de nexthop existentes.   %
               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               Nht{NC(o)}{d}=Nht{NC(o)}{d}-1;
               loop=1;
            end
         end
      end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Muda estado da rede e calcula nova árvore de destino a    %
   % partir de NH.                                              %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   E=E+1;
   [Cd] = calcdtree(NH,Nn);
end
if prod(all(NHo==NH))~=1

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Conta as vezes em que foi feita a correção.                %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   Cor=Cor+1;
   if E>Eo
      Eo=E;
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Resultados:                                                     %
% - Pfo {1}{o}{d}{:}: array de caminhos ordenados;               %
% - Sfo {m}{o}{d}{:}: array das correspondentes sínteses; e      %
% - NH matriz de próximo nó.                                      %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

A.2 Rotina Buscapath

```
function [Nn,I] = buscapath(T);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [Nn,I] = buscapath(T), em que:                     %
%     - A topologia é modelada em uma Matriz T em que as:      %
%        - linhas são a origem das conexões;                   %
%        - colunas são o destino das conexões;                 %
%     - Nn: é o número total de nós da rede; e                 %
%     - I{n}{:}: é o array de caminhos encontrados com origem  %
%       no nó "n" pela busca pela estensão.                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nn=size(T,1);
C={};
D={};
I={};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Para cada nó existente como origem, busca as possíveis árvo-%
%  res, em que:                                                %
%     - n: é o nó de origem da árvore;                         %
%     - h: é o número de hops;                                 %
%     - D: conexões diretas do nó "n"; e                       %
%     - C: nós com conexão direta ao nó "n".                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for n=1:Nn
  h=1;
  D{h}{n}= T(n,:);
  C{h}{n}= find(D{h}{n}==1);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %   Verifica se o nó "n" tem conexão. Se tiver, para o 1o hop%
  %   busca nós com conexão direta ao nó "n", para cada nó "n" %
  %   contido em C.                                            %
  %      - I{n}{:}: índice de nós de cada árvore com origem    %
```

```
%                 em "n"                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~isempty(C{h}{n})
   for a=1:size(C{h}{n},2),
      I{n}{a}=[n,C{h}{n}(1,a)];
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %   Para o 2o hop busca nós com conexão direta ao nó "n+1"%
   %   para cada nó n+1 contido em C1.                       %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   NI=cell(1,1);
   cp=0;
   while cp~=size(I{n},2)
      h=h+1;
      cp=0;
      for i=1:size(I{n},2),
         if size(I{n}{i},2)>=h
            D{h}{i}= T(I{n}{i}(h),:);
            C{h}{i}= find(D{h}{i}==1);
            if ~isempty(C{h}{i})
               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               %   Elimina enlace de volta (loop)            %
               %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               for j=1:size(I{n}{i},2)
                  k=find(C{h}{i}==I{n}{i}(1,j));
                  if isempty(k)
                  elseif k==1
                     C{h}{i}=[C{h}{i}(1,k+1:size(C{h}{i},2))];
                  elseif k>1
                     C{h}{i}=[C{h}{i}(1,1:k-1) C{h}{i}(1,k+1:
                        size(C{h}{i},2))];
                  end
               end
               if isempty(NI{:})
                  rasc=0;
               else
                  rasc=size(NI{:},2);
```

```
                 end
                 if ~isempty(C{h}{i})
                    for j=1:size(C{h}{i},2)
                       NI{:}{rasc+j}=[I{n}{i} C{h}{i}(1,j)];
                    end
                 else
                    NI{:}{rasc+1}=[I{n}{i}];
                 end
              else
                 if isempty(NI{:})
                    rasc=0;
                 else
                    rasc=size(NI{:},2);
                 end
                 NI{:}{rasc+1}=[I{n}{i}];
                 cp=cp+1;
              end
           else
              if isempty(NI{:})
                 rasc=0;
              else
                 rasc=size(NI{:},2);
              end
              NI{:}{rasc+1}=[I{n}{i}];
              cp=cp+1;
           end
        end
        I{n}=NI{:};
        NI=cell(1,1);
     end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %   Se o nó "n" não tem conexão                             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  else
     I{n}{1}=[n];
  end
end
```

A.3 Rotina Separapath

```
function [P,w] = separapath(M,I,Nn);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [P,w] = separapath(M,I,Nn), em que:                  %
%     - A topologia é modelada em uma Matriz T em que as:        %
%        - linhas são a origem das conexões;                     %
%        - colunas são o destino das conexões;                   %
%     - I{n}{:}: é o array de caminhos encontrados com origem    %
%       no nó "n" pela busca pela estensão;                      %
%     - Nn: é o número total de nós da rede;                     %
%      - M{m}: corresponde a matriz da métrica "m";              %
%     - Os caminhos obtidos entre "o" e "d":                     %
%        - P{o}{d}={[o, ..., d],....,[o, ..., d]} em que:        %
%                P{o}{d}={[]} para o=d;                          %
%      - Os vetores dos pesos dos enlaces dos caminhos obtidos   %
%       em P{o}{d}{:}:                                           %
%        - w{m}{o}{d}(:) para m=1,2,..ou., m onde m é a metrica%
%         escolhida;                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
P={};
w={};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Varre todos os nós como origem e como destino                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for o=1:Nn
   for d=1:Nn
      if o==d

         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         %  Define caminho e peso nulo quando origem = destino.%
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         P{o}{d}{1}=[];
         for m=1:size(M,2)
            w{m}{o}{d}{1}=[];
         end
      else
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Verifica se há pelo menos 1 hop.                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if size(I{o}{1},2)>=2
   k=1;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  Varre os caminhos encontrados para a origem "o" %
   %  e verifica se o caminho tem o nó "d" de destino.%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   for i=1:size(I{o},2)
      if ~isempty(find(I{o}{i}(:)==d))
         P{o}{d}{k}=[];
         f=0;

         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         %  Varre dentro do caminho, buscando o nó de %
         %  destino "d".                              %
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         for j=1:size(I{o}{i},2)
            if f==0
               if I{o}{i}(1,j)~=d
                  P{o}{d}{k}=[P{o}{d}{k} I{o}{i}(1,j)];
                  switch j

                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  %  Pula o primeiro nó pois não há   %
                  %  peso para montar.                %
                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  case 1

                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  %  Monta peso do segundo nó.        %
                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  case 2
                     for m=1:size(M,2)
                        w{m}{o}{d}{k}=[M{m}(P{o}{d}{k}
```

```
75                                  (1,j-1),I{o}{i}(1,j))];
76                               end
77
78                            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
79                            %  Monta peso do enlace.            %
80                            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81                            otherwise
82                               for m=1:size(M,2)
83                                  w{m}{o}{d}{k}=[w{m}{o}{d}{k};
84                                     M{m}(P{o}{d}{k}(1,j-1),
85                                     I{o}{i}(1,j))];
86                               end
87                            end
88                         else
89                            P{o}{d}{k}=[P{o}{d}{k} I{o}{i}(1,j)];
90                            switch j
91                            case 1
92                               'Erro'
93                            case 2
94                               for m=1:size(M,2)
95                                  w{m}{o}{d}{k}=[M{m}(P{o}{d}{k}
96                                     (1,j-1),I{o}{i}(1,j))];
97                               end
98                            otherwise
99                              for m=1:size(M,2)
100                              w{m}{o}{d}{k}=[w{m}{o}{d}{k} M{m}
101                              (P{o}{d}{k}(1,j-1),I{o}{i}(1,j))];
102                             end
103                           end
104
105                           %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
106                           %Verifica se há caminhos duplicados.%
107                           %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
108                           if k~=1
109
110                              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
111                              % Verificar o caso em que os ca- %
112                              % minhos tem diferentes compri-  %
```

```
113                                 % mentos.                          %
114                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
115                                 if all(size(P{o}{d}{k})==size(P{o}
116                                    {d}{k-1}))
117
118                                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119                                    % Verificar o caso em que os  %
120                                    % caminhos são iguais e exclui%
121                                    % caminhos duplicados.        %
122                                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
123                                   if all(P{o}{d}{k}==P{o}{d}{k-1})
124                                       P{o}{d}=P{o}{d}(1:k-1);
125
126                                      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
127                                      % Define peso nulo para o   %
128                                      % enlace do caminho quando  %
129                                      % origem = destino.         %
130                                      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
131                                      for m=1:size(M,2)
132                                          w{m}{o}{d}=w{m}{o}{d}
133                                          (1:k-1);
134                                      end
135                                      k=k-1;
136                                   end
137                                 end
138                              end
139                              k=k+1;
140                              f=1;
141                           end
142                        end
143                     end
144
145                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
146                  %  Se o caminho não tem o nó "d" de destino.    %
147                  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
148                  else
149                     if k==1
150                        P{o}{d}{k}=[];
```

```
                     for m=1:size(M,2)
                        w{m}{o}{d}{k}=[];
                     end
                  end
               end
            end

         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         %  Se não há pelo menos 1 hop. Define caminho e peso  %
         %  nulos.                                             %
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         else
            P{o}{d}{1}=[];
            for m=1:size(M,2)
               w{m}{o}{d}{1}=[];
            end
         end
     end
   end
end
```

A.4 Rotina Calculapath

```
function [Po,So] = calculapath(F,P,w,Nn,word);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [Po,So] = calculapath(F,P,w,Nn,word), em que:      %
%                                                               %
%  - Os caminhos obtidos entre "o" e "d":                       %
%  - P{o}{d}={[o, ..., d],....,[o, ..., d]} em que:             %
%     P{o}{d}={[ ]} para o=d;                                   %
%  - Os vetores dos pesos dos enlaces dos caminhos obtidos em   %
%     P{o}{d}{:} é dado por:                                    %
%     - w{m}{o}{d}(:) para m=1,2,..ou., m onde m é a metrica    %
%       escolhida;                                              %
% - F é o array que contém para cada metrica a síntese e rela-  %
%   ção de ordem a ser aplicada F{m}={'síntese', 'relação'}:    %
%   Ex: F{1}={'MIN','>'}; F{2}={'MAX','<'}; F{3}={'MUL','>'}    %
```

```
%   e F{M}={'ADD','<'}; em que size(F,2)=size(M,2)= número     %
%   de métricas usadas;                                        %
% - A relação de ordem léxica multidimensional é aplicada na   %
%   sequencia em que se define o array F, ou seja:             %
%  - F{1} tem prioridade sobre F{2};                           %
%  - F{2} tem prioridade sobre F{3}, etc.;                     %
% - size(M{1})=size(M{2})=size(M{3})=...size(M{m})= size(T)=   %
%   size(F,2)=size(M,2)= número de métricas usadas;            %
% - Nn: é o número total de nós da rede;                       %
% - word: é o array com as abreviações utilizadas da palavra-  %
%   peso;                                                      %
%  - word{m}=l l letras para a palavra-peso da síntese m; e    %
%  - word{m}=0 significa palavra-peso sem abreviação;          %
%  - Po{m}{o}{d}: contém os caminhos entre 'o' e 'd' ordenados %
%    para a métrica 'm';                                       %
%  - So{m}{o}{d}: contém as correspondentes sinteses desses    %
%    caminhos.                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Po={};
So={};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Varre todas as métrica e todos os nós como origem e como     %
% destino.                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:size(F,2)
   for o=1:Nn
      for d=1:Nn

         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         % Cria a matriz S de sínteses e verifica a síntese a %
         % ser usada.                                         %
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         switch char(F{i}(1,1))
         case 'MIN'
            S{i}{o}{d}=sintesemin(w{i}{o}{d},word{i});
         case 'MAX'
            S{i}{o}{d}=sintesemax(w{i}{o}{d},word{i});
```

```
            case 'ADD'
               S{i}{o}{d}=sinteseadd(w{i}{o}{d},word{i});
            case 'MUL'
               S{i}{o}{d}=sinteseprod(w{i}{o}{d},word{i});
            otherwise
            end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Ordena os caminhos para uma mesma métrica e verifica%
            % a ordenação ser usada.                                %
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            switch char(F{i}(1,2))
            case '>'
               [Po{i}{o}{d},So{i}{o}{d}] = lexmaior(P{o}{d},
               S{i}{o}{d});
            case '<'
               [Po{i}{o}{d},So{i}{o}{d}] = lexmenor(P{o}{d},
               S{i}{o}{d});
            otherwise
            end
         end
      end
   end
```

A.4.1 Rotina Sintesemin

```
function [S] = sintesemin(C,word);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [S] = sintesemin(C,word), calcula a síntese minimi-%
%           zativa dos caminhos C de mesma origem o destino "d"%
%           composto de enlaces l1,l2,...,ln; em que:          %
%                                                              %
%  - C=w{m}{o}{d}(:) para m=1,2,...,m;                         %
%  - word: é o array com as abreviações utilizadas da palavra- %
%   peso;                                                      %
%     - word{m}=l l letras para a palavra-peso da síntese m;   %
%     - word{m}=0 significa palavra-peso sem abreviação;       %
%  - S{o}{d}{:} é a palavra-peso dos caminhos C=P{o}{d}{:};    %
```

```
%      S(P{o}{d})=[] para P{o}{d}=[] ou o=d;                   %
%      S=S{o}{d}{:}{[Wn, ..., W1],....,[Wn', ..., W1]}.        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~isempty(C)
   for p=1:size(C(:),1)
      S{p}=0;
      i=1;
      for l=size(C{p},2):-1:1,
         S{p}(i)= min(C{p}(1,1:l));
         i=i+1;
      end
   end
   if word~=0
      So={};
      for p=1:size(C(:),1)
         So{p}=S{p}(1:word);
      end
      S={};
      S=So;
   end
else
   S=[];
end

```

A.4.2 Rotina Sintesemax

```
function [S] = sintesemax(C,word);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [S] = sintesemin(C,word), calcula a síntese maximi-%
%           zativa dos caminhos C de mesma origem o destino "d"%
%           composto de enlaces l1,l2,...,ln; em que:          %
%                                                              %
%  - C=w{m}{o}{d}(:) para m=1,2,...,m;                         %
%  - word: é o array com as abreviações utilizadas da palavra- %
%   peso;                                                      %
%     - word{m}=l l letras para a palavra-peso da síntese m;   %
%     - word{m}=0 significa palavra-peso sem abreviação;       %
```

```
12  %  - S{o}{d}{:} é a palavra-peso dos caminhos C=P{o}{d}{:};    %
13  %     S(P{o}{d})=[] para P{o}{d}=[] ou o=d;                    %
14  %     S=S{o}{d}{:}{[Wn, ..., W1],....,[Wn', ..., W1]}.         %
15  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16  if ~isempty(C)
17     for p=1:size(C(:),1)
18        S{p}=0;
19        i=1;
20        for l=size(C{p},2):-1:1,
21           S{p}(i)= max(C{p}(1,1:l));
22           i=i+1;
23        end
24     end
25     if word~=0
26        So={};
27        for p=1:size(C(:),1)
28           So{p}=S{p}(1:word);
29        end
30        S={};
31        S=So;
32     end
33  else
34     S=[];
35  end
```

A.4.3 Rotina Sinteseadd

```
1   function [S] = sinteseadd(C,word);
2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3   %   Função: [S] = sintesemin(C,word), calcula a síntese aditiva%
4   %           dos caminhos C de mesma origem o destino "d"       %
5   %           composto de enlaces l1,l2,...,ln; em que:          %
6   %                                                              %
7   %  - C=w{m}{o}{d}(:) para m=1,2,...,m;                         %
8   %  - word: é o array com as abreviações utilizadas da palavra- %
9   %   peso;                                                      %
10  %     - word{m}=l l letras para a palavra-peso da síntese m;   %
11  %     - word{m}=0 significa palavra-peso sem abreviação;       %
```

```
12  %  - S{o}{d}{:} é a palavra-peso dos caminhos C=P{o}{d}{:};    %
13  %     S(P{o}{d})=[] para P{o}{d}=[] ou o=d;                    %
14  %     S=S{o}{d}{:}{[Wn, ..., W1],....,[Wn', ..., W1]}.         %
15  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16  if ~isempty(C{1})
17     for p=1:size(C(:),1)
18        S{p}=0;
19        i=1;
20        for l=size(C{p},2):-1:1,
21           S{p}(i)= abs(sum(C{p}(1,1:l)));
22           i=i+1;
23        end
24     end
25     if word~=0
26        So={};
27        for p=1:size(C(:),1)
28           So{p}=S{p}(1:word);
29        end
30        S={};
31        S=So;
32     end
33  else
34     S=[];
35  end
```

A.4.4 Rotina Sinteseprod

```
1   function [S] = sinteseprod(C,word);
2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3   %   Função: [S] = sintesemin(C,word), calcula a síntese multi- %
4   %           cativa dos caminhos C de mesma origem o destino "d"%
5   %           composto de enlaces l1,l2,...,ln; em que:          %
6   %                                                              %
7   %  - C=w{m}{o}{d}(:) para m=1,2,...,m;                         %
8   %  - word: é o array com as abreviações utilizadas da palavra- %
9   %   peso;                                                      %
10  %     - word{m}=l l letras para a palavra-peso da síntese m;   %
11  %     - word{m}=0 significa palavra-peso sem abreviação;       %
```

```
12  %  - S{o}{d}{:} é a palavra-peso dos caminhos C=P{o}{d}{:};    %
13  %     S(P{o}{d})=[] para P{o}{d}=[] ou o=d;                    %
14  %     S=S{o}{d}{:}{[Wn, ..., W1],....,[Wn', ..., W1]}.         %
15  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16  if ~isempty(C)
17     for p=1:size(C(:),1)
18        S{p}=0;
19        i=1;
20        for l=size(C{p},2):-1:1,
21           S{p}(i)= prod(C{p}(1,1:l));
22           i=i+1;
23        end
24     end
25     if word~=0
26        So={};
27        for p=1:size(C(:),1)
28           So{p}=S{p}(1:word);
29        end
30        S={};
31        S=So;
32     end
33  else
34     S=[];
35  end
```

A.4.5 Rotina Lexmaior

```
1   function [O,Wo] = lexmaior(p,W);
2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3   %  Função: [O,Wo] = lexmaior(p,W), ordena de forma lexicamente %
4   %     maior o Array p(caminhos) e W{m}(palavra-peso) em que:   %
5   %     - a maior palavra-peso é o primeiro elemento do array    %
6   %       Wo{m}{o}{d}{:} e corresponde ao primeiro elemento do   %
7   %       array O{m}{o}{d}{:};                                   %
8   %     - a menor palavra-peso é o último elemento do array      %
9   %       Wo{m}{o}{d}{:} e corresponde ao último elemento do     %
10  %       array O{m}{o}{d}{:};                                   %
11  %  - p: é o array de caminhos tal que:                         %
```

```
%       - p{o}{d}={[o, ..., d],....,[o, ..., d]}, em que:      %
%       - p{o}{d}={[]} para o=d;                               %
%  - W: é o array de palavras-peso dos enlaces dos caminhos    %
%       referentes a métrica m tal que:                        %
%       - W{m=valor definido para entrar nessa função}{o}{d}(:)%
%         para m=1,2,..ou., m onde m é a metrica escolhida.    %
%       Obs: Funciona para um m definido.                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[O,Wo]=ordenachar(p,W','>',1);
IND=geraindex(Wo);
L=[];
for i=1:length(Wo),
   L(i)=length(Wo{i});
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ordena tamanho dos novos subgrupos com mesmo valor na        %
% primeira letra e varre os grupos.                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:max(L)
   for i=1:size(IND,1)
      if IND(i,1)~=IND(i,2)
         c={Wo{IND(i,1):IND(i,2)}}';
         Oo={O{IND(i,1):IND(i,2)}}';
         [Oo,c]=ordenasize(Oo,c);
         Wo(IND(i,1):IND(i,2))=c;
         O(IND(i,1):IND(i,2))=Oo;
      end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Ajusta índice e ordena subgrupos por caracter.            %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   IND=ajustindexsize(IND,j,Wo);
   for i=1:size(IND,1)
      if IND(i,1)~=IND(i,2)
         c={Wo{IND(i,1):IND(i,2)}}';
         Oo={O{IND(i,1):IND(i,2)}}';
```

```
            [Oo,c]=ordenachar(Oo,c,'>',j);
            Wo(IND(i,1):IND(i,2))=c;
            O(IND(i,1):IND(i,2))=Oo;
         end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Ajusta índice, seleciona novos subgrupos com caracter j   %
   % iguais e ordena por tamanho.                              %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   INDR=[];
   for i=1:size(IND,1)
      INDO=IND(i,:);
      if IND(i,1)~=IND(i,2)
         igual=0;
         for k=1:(IND(i,2)-IND(i,1))
            if Wo{IND(i,1)+k-1}(1,j)~=Wo{IND(i,1)+k}(1,j)
               n=size(INDO,1);
               INDO(n+1,2)=INDO(n,2);
               INDO(n,2)=INDO(n,1)+igual;
               INDO(n+1,1)=INDO(n,2)+1;
               igual=0;
            else
               igual=igual+1;
            end
         end
      end
      INDR=[INDR;INDO];
   end
   IND=INDR;
end
```

A.4.5.1 Rotina Ordenachar

```
function [O,Wo]=ordenachar(p,W,R,J)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Função: [O,Wo]=ordenachar(p,W,R,J), ordena os arrays        %
%     p(caminhos) e W{m}(palavra-peso) pelo valor do caracter  %
```

```
5   %     J das palavras.                                          %
6   %     - se R='<' Palavra com caracter J menor fica sobre as    %
7   %       palavras J maior;                                      %
8   %     - se R='>' Palavra com caracter J maior sobre as palavras%
9   %       J menor em que J é a posição do caracter 1o, 2o, etc.. %
10  %       Obs: Funciona para um m definido.                      %
11  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12  if ~isempty(W)
13          l=size(W,1);
14     for i=1:l
15        c(i,1)=W{i}(1,J);
16          c(i,2)=i;
17          end
18          if R=='<'
19          for i=1:l
20          k=find(c(:,1)==min(c(:,1)));
21          co(i,1)=c(k(1),1);
22          co(i,2)=c(k(1),2);
23          c(k(1),:)=[];
24          end
25          elseif R=='>'
26          for i=1:l
27           k=find(c(:,1)==max(c(:,1)));
28           co(i,1)=c(k(1),1);
29           co(i,2)=c(k(1),2);
30           c(k(1),:)=[];
31        end
32     end
33     for i=1:l
34          Wo{i}=W{co(i,2)};
35          O{i}=p{co(i,2)};
36          end
37  else
38     Wo=[];
39     O=[];
40  end
```

A.4.5.2 Rotina Geraindex

```
function [Ind] = geraindex(M)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Função: [Ind] = geraindex(M), gera o vetor de índice para a %
%                  matriz M, agrupando conjuntos de igula pri- %
%                  meiro carcter                               %
%     - Ind=[indice menor, indice maior]                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:length(M)
   c(i,1)=M{i}(1,1);
end
n=1;
anteriorigual=0;
Ind=[1 1];
if length(M)>1
   for m=1:length(c)-1
      if c(m)==c(m+1)
         if anteriorigual==1
            Ind(n,2)=m+1;
         else
            Ind(n,1)=m;
            Ind(n,2)=m+1;
            anteriorigual=1;
         end
      else
         anteriorigual=0;
         n=n+1;
         Ind(n,1)=m+1;
         Ind(n,2)=m+1;
      end
   end
end
```

A.4.5.3 Rotina Ordenasize

```
function [O,Wo]=ordenasize(p,W)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
3   %  Função: [O,Wo]=ordenasize(p,W), ordena o arrays            %
4   %     p(caminhos) e W{m}(palavra-peso) pelo tamanho das       %
5   %     palavras-peso.                                          %
6   %        A palavra de menor tamanho fica sobre as palavras de %
7   %     maior tamanho.                                          %
8   %     - O{m}: array de caminhos ordenado;                     %
9   %     - Wo{m}: array das palavras-pesos ordenadas.            %
10  %       Obs: Funciona para um m definido.                     %
11  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12  if ~isempty(W)
13     l=length(W);
14          for i=1:l
15          c(i,1)=length(W{i});
16           c(i,2)=i;
17          end
18          for i=1:l
19          k=find(c(:,1)==min(c(:,1)));
20                  co(i,1)=c(k(1),1);
21          co(i,2)=c(k(1),2);
22          c(k(1),:)=[];
23     end
24     for i=1:l
25          Wo{i}=W{co(i,2)};
26          O{i}=p{co(i,2)};
27     end
28  else
29     Wo={[]};
30     O={[]};
31  end
```

A.4.5.4 Rotina Ajustindexsize

```
1   function [Indr] = ajustindexsize(Ind,J,M)
2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3   %  Função: [Indr] = ajustindexsize(Ind,J,M), ajusta o vetor de %
4   %          indice para a matriz M, colocando na frente de um   %
5   %          subgrupo x, dado por Ind(x,2)-Ind(x,1), as palavras %
6   %          de tamanho igual a J-1, do ponto de vista do carac- %
```

```
%          ter J em análise.                                     %
%  - Ind=[indice menor, indice maior].                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Indr=[];
for i=1:size(Ind,1)
   Indo=Ind(i,:);
   if Ind(i,1)~=Ind(i,2)
      Mo={M{Ind(i,1):Ind(i,2)}}';
      for k=1:size(Mo,1)
         if length(Mo{k})==J-1
            n=size(Indo,1);
            Indo(n+1,2)=Indo(n,2);
            Indo(n,2)=Indo(n,1);
            if Indo(n,2)+1<=Indo(n+1,2)
               Indo(n+1,1)=Indo(n,2)+1;
            else
               Indo(n+1,:)=[];
            end
         end
      end
   end
   Indr=[Indr; Indo];
end
```

A.4.6 Rotina Lexmenor

```
function [O,Wo] = lexmenor(p,W);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Função: [O,Wo] = lexmenor(p,W), ordena de forma lexicamente %
%     maior o Array p(caminhos) e W{m}(palavra-peso) em que:   %
%     - a menor palavra-peso é o primeiro elemento do array    %
%       Wo{m}{o}{d}{:} e corresponde ao primeiro elemento do   %
%       array O{m}{o}{d}{:};                                   %
%     - a maior palavra-peso é o último elemento do array      %
%       Wo{m}{o}{d}{:} e corresponde ao último elemento do     %
%       array O{m}{o}{d}{:};                                   %
%  - p: é o array de caminhos tal que:                         %
%       - p{o}{d}={[o, ..., d],....,[o, ..., d]}, em que:      %
```

```
%       - p{o}{d}={[]} para o=d;                                %
%  - W: é o array de palavras-peso dos enlaces dos caminhos     %
%       referentes a métrica m tal que:                         %
%       - W{m=valor definido para entrar nessa função}{o}{d}(:)%
%         para m=1,2,..ou., m onde m é a metrica escolhida.     %
%       Obs: Funciona para um m definido.                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[O,Wo]=ordenachar(p,W','<',1);)
IND=geraindex(Wo);
L=[];
for i=1:length(Wo),
   L(i)=length(Wo{i});
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ordena tamanho dos novos subgrupos com mesmo valor na         %
% primeira letra e varre os grupos.                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:max(L)
   for i=1:size(IND,1)
      if IND(i,1)~=IND(i,2)
         c={Wo{IND(i,1):IND(i,2)}}';
         Oo={O{IND(i,1):IND(i,2)}}';
         [Oo,c]=ordenasize(Oo,c);
         Wo(IND(i,1):IND(i,2))=c;
         O(IND(i,1):IND(i,2))=Oo;
      end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Ajusta índice e ordena subgrupos por caracter.             %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   IND=ajustindexsize(IND,j,Wo);
   for i=1:size(IND,1)
      if IND(i,1)~=IND(i,2)
         c={Wo{IND(i,1):IND(i,2)}}';
         Oo={O{IND(i,1):IND(i,2)}}';
         [Oo,c]=ordenachar(Oo,c,'<',j);
```

```
         Wo(IND(i,1):IND(i,2))=c;
         O(IND(i,1):IND(i,2))=Oo;
      end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Ajusta índice, seleciona novos subgrupos com caracter j   %
   % iguais e ordena por tamanho.                               %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   INDR=[];
   for i=1:size(IND,1)
      INDO=IND(i,:);
      if IND(i,1)~=IND(i,2)
         igual=0;
         for k=1:(IND(i,2)-IND(i,1))
            if Wo{IND(i,1)+k-1}(1,j)~=Wo{IND(i,1)+k}(1,j)
               n=size(INDO,1);
               INDO(n+1,2)=INDO(n,2);
               INDO(n,2)=INDO(n,1)+igual;
               INDO(n+1,1)=INDO(n,2)+1;
               igual=0;
            else
               igual=igual+1;
            end
         end
      end
      INDR=[INDR;INDO];
   end
   IND=INDR;
end
```

A.5 Rotina Ordenapaths

```
function [Pfo,Sffo] = ordenapaths(F,Po,So,Nn);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [Pfo,Sffo] = ordenapaths(F,Po,So,Nn), ordena cami- %
%           nhos iguais para uma mesma métrica, ou seja ordena %
```

```
5   %           entre métricas, em que:                          %
6   %                                                            %
7   % - A relação de ordem léxica multidimensional é aplicada na  %
8   %   sequencia em que se define o array F, ou seja:           %
9   %     - F{1} tem prioridade sobre F{2};                      %
10  %     - F{2} tem prioridade sobre F{3}, etc.;                %
11  % - size(M{1})=size(M{2})=size(M{3})=...size(M{m})= size(T)=  %
12  %   size(F,2)=size(M,2)= número de métricas usadas;          %
13  % - Nn: é o número total de nós da rede;                     %
14  % - Po{m}{o}{d}: contém os caminhos entre "o" e "d" ordenados %
15  %    para a métrica 'm';                                     %
16  % - So{m}{o}{d}: contém as correspondentes sinteses desses    %
17  %    caminhos;                                               %
18  % - Pfo{o}{d}: contém os caminhos entre "o" e "d" ordenados  %
19  %              entre as métricas;                            %
20  % - Sfo{o}{d}: contém os caminhos entre "o" e "d" ordenados  %
21  %              entre as métricas;                            %
22  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23  Nm=size(F,2);
24  Pfo={};
25  Sfo={};
26  if Nm>1
27
28     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29     %  Varre nós de origem e destino buscando caminhos iguais  %
30     %  para uma mmesma métrica.                                %
31     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
32     for o=1:Nn
33        for d=1:Nn
34           m=1;
35
36           %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37           %  Toma como referencia a ordem da primeira métrica.  %
38           %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39           Pfo{1}{o}{d}=Po{m}{o}{d};
40           Sfo{1}{o}{d}=So{m}{o}{d};
41
42           %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
43          %  Indica que todos os caminhos podem ser reordenados %
44          %  porque  Pfo{2} são iguais entre si.                 %
45          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
46          g=0;
47          for i=1:size(Po{m}{o}{d},2)
48             Pfo{2}{o}{d}{i}=[g];
49             Sfo{2}{o}{d}{i}=[g];
50             Pfo{3}{o}{d}{i}=[];
51             Sfo{3}{o}{d}{i}=[];
52          end
53
54          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55          %  Verifica se há mais de um caminho e se há, rever   %
56          %  quando m=2 é igual, mas m=1 é diferente, e deixa   %
57          %  valer m=2 e não m=1.                                %
58          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
59          if size(Pfo{1}{o}{d},2)>1
60             while m<Nm
61                i=1;
62                while i<=(size(Pfo{1}{o}{d},2)-1)
63
64                   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
65                   % Varre os caminhos em busca dos que têm     %
66                   % pesos m=1 iguais                           %
67                   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
68                   if size(So{m}{o}{d}{findpos(Po{m}{o}{d},
69                         Pfo{1}{o}{d}{i})},2)== size(So{m}{o}{d}
70                      {findpos(Po{m}{o}{d},Pfo{1}{o}{d}{i+1})},2)
71                      if So{m}{o}{d}{findpos(Po{m}{o}{d},
72                            Pfo{1}{o}{d}{i})}==So{m}{o}{d}
73                         {findpos(Po{m}{o}{d},Pfo{1}{o}{d}{i+1})}
74
75                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76                         % Se encontra um, seleciona grupo de   %
77                         % caminhos de pesos m iguais que se    %
78                         % pode subagrupar, pois os "m" anterio-%
79                         % res eram iguais também.              %
80                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
81                          if Pfo{2}{o}{d}{i}==Pfo{2}{o}{d}{i+1}
82                             fim=0;
83                             PP={};
84                             SS={};
85                             g=g+1;
86                             PP{1}=Pfo{1}{o}{d}{i};
87                             SS{1}=Sfo{1}{o}{d}{i};
88                             PP{2}=Pfo{1}{o}{d}{i+1};
89                             SS{2}=Sfo{1}{o}{d}{i+1};
90                             Pfo{3}{o}{d}{i}=g;
91                             Sfo{3}{o}{d}{i}=g;
92                             Pfo{3}{o}{d}{i+1}=g;
93                             Sfo{3}{o}{d}{i+1}=g;
94                             cg=i;
95                             fg=i+1;
96                             i=i+2;
97                             j=3;
98                             for p=i:size(Pfo{1}{o}{d},2)
99                                if fim==0
100                                  if size(So{m}{o}{d}
101                                     {findpos(Po{m}{o}{d},
102                                        Pfo{1}{o}{d}{p-1})},2)==
103                                     size(So{m}{o}{d}
104                                     {findpos(Po{m}{o}{d},
105                                        Pfo{1}{o}{d}{p})},2)
106                                     if So{m}{o}{d}
107                                        {findpos(Po{m}{o}{d},
108                                           Pfo{1}{o}{d}{p-1})}==
109                                        So{m}{o}{d}
110                                        {findpos(Po{m}{o}{d},
111                                           Pfo{1}{o}{d}{p})}
112                                        if Pfo{2}{o}{d}{p-1}==
113                                           Pfo{2}{o}{d}{p}
114                                           PP{j}=Pfo{1}{o}{d}{p};
115                                           SS{j}=Sfo{1}{o}{d}{p};
116                                           Pfo{3}{o}{d}{p}=g;
117                                           Sfo{3}{o}{d}{p}=g;
118                                           j=j+1;
```

```
                  i=i+1;
                  fg=fg+1;
               else
                  fim=1;
               end
            else
               fim=1;
            end
         else
            fim=1;
         end
      end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Odena caminhos segundo m=m+1.     %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   [PP,SS]=ordenam(PP,SS,Po,So,m+1,o,d);
   Pfo{1}{o}{d}(cg:fg)=PP;
   Sfo{1}{o}{d}(cg:fg)=PP;
   if fg~=size(Pfo{1}{o}{d},2)
      g=g+1;
      Pfo{3}{o}{d}{i}=g;
      Sfo{3}{o}{d}{i}=g;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Caso, em que:                        %
%   Pfo{2}{o}{d}{i}==Pfo{2}{o}{d}{i+1}.%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
else
   g=g+1;
   Pfo{3}{o}{d}{i}=g;
   Sfo{3}{o}{d}{i}=g;
   g=g+1;
   Pfo{3}{o}{d}{i+1}=g;
   Sfo{3}{o}{d}{i+1}=g;
   i=i+1;
```

```
                    end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Caso, em que:                           %
                %   So{m}{o}{d}{i}~==So{m}{o}{d}{i+1}.    %
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                else
                    g=g+1;
                    Pfo{3}{o}{d}{i}=g;
                    Sfo{3}{o}{d}{i}=g;
                    g=g+1;
                    Pfo{3}{o}{d}{i+1}=g;
                    Sfo{3}{o}{d}{i+1}=g;
                    i=i+1;
                end

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Caso, em que:                              %
            % size(So{m}{o}{d}{i},2)~=                   %
            %                       size(So{m}{o}{d}{i+1},2).%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            else
                g=g+1;
                Pfo{3}{o}{d}{i}=g;
                Sfo{3}{o}{d}{i}=g;
                g=g+1;
                Pfo{3}{o}{d}{i+1}=g;
                Sfo{3}{o}{d}{i+1}=g;
                i=i+1;
            end
        end
        m=m+1;
        Pfo{2}=Pfo{3};
        Sfo{2}=Sfo{3};
        for i=1:size(Po{m}{o}{d},2)
            Pfo{3}{o}{d}{i}=[];
            Sfo{3}{o}{d}{i}=[];
        end
```

```
            end
         else
            Po{m}{o}{d}=[];
            So{m}{o}{d}=[];
         end
      end
   end
elseif Nm==1

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Caso, em que só há uma métrica:                               %
   %  - toma como referencia a ordem da unica métrica;             %
   %  - toma como referencia a ordem da unica métrica.             %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   Pfo{1}=Po{1};
   Sfo{1}=So{1};
else
   'Erro: Não há nenhuma métrica definida'
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Busca valores das sínteses dos camimhos ordenandos em Pfo:  %
%  - Po{m}{o}{d}{:}: entrada array de caminhos não ordenados;  %
%  - So{m}{o}{d}{:}: entrada array das sinteses correspondentes%
%                    do Po;                                    %
%  - Pfo {m}{o}{d}{:}: entrada array de caminhos ordenados;    %
%  - Sffo {m}{o}{d}{:}: saída array das sinteses corresponden- %
%                       tes do Pfo.                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Sffo={};
Pos=[];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Varre todas as métricas, nó de origem e destino buscando    %
%  caminhos iguais para uma mmesma métrica.                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m=1:size(F,2)
   for o=1:Nn
```

```
233         for d=1:Nn
234            for i=1:size(Pfo{1}{o}{d},2)
235               acho=0;
236               j=1;
237               Indice=[1:1:size(Pfo{1}{o}{d},2)];
238               Indicen=[];
239               while j<=size(Pfo{1}{o}{d}{i},2)& acho==0
240                  z=1;
241                  for k=1:size(Indice,2)
242                     if    j<size(Pfo{1}{o}{d}{i},2)
243                        if size(Po{m}{o}{d}{Indice(k)},2)>=j
244                           if Pfo{1}{o}{d}{i}(j)==
245                              Po{m}{o}{d}{Indice(k)}(j)
246                              Indicen(z)=Indice(k);
247                              z=z+1;
248                           end
249                        end
250                     elseif j==size(Pfo{1}{o}{d}{i},2)
251                        if size(Po{m}{o}{d}{Indice(k)},2)>=j
252                           if Pfo{1}{o}{d}{i}(j)==
253                              Po{m}{o}{d}{Indice(k)}(j)
254                              if size(Pfo{1}{o}{d}{i}(j))==
255                                 size(Po{m}{o}{d}{Indice(k)}(j),2)
256                                 Indicen(z)=Indice(k);
257                                 z=z+1;
258                              end
259                           end
260                        end
261                     end
262                  end
263                  if acho==0
264                     if size(Indicen,2)==1
265                        Sffo{m}{o}{d}{i}=So{m}{o}{d}{Indicen(1)};
266                        acho=1;
267                     end
268                  end
269                  j=j+1;
270                  Indice=Indicen;
```

```
               Indicen=[];
            end
         end
      end
   end
end
```

A.5.1 Rotina Findpos

```
function [p] = findpos(A,B);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [p] = findpos(A,B), encontra em A a primeira       %
%                 posição p do array de caminhodado em B.      %
%     - B=n1 n2 n3 ... nn                                      %
%     - A: é um array A{c}={n11  n12   n13   ...   n1n         %
%                          n21  n22   n23   ...   n2n         %
%                          ...........................}      %
%     Retorna p=0 se não encomtrar B em A                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nA=size(A,2);
nB=size(B,2);
p=0;
ok=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Varre os caminhos en A e os nós de B                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:nA
   if size(A{i},2)==nB
      if ok==0
         l=0;
         for j=1:nB
            if  A{i}(1,j)==B(1,j)
               l=l+1;
            else
               l=0;
            end
         end
```

```
         if l==nB
            p=i;
            ok=1;
         end
      end
   end
end
```

A.5.2 Rotina Ordenam

```
function [PPo,SSo]=ordenam(PP,SS,Po,So,m,o,d);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [PPo,SSo]=ordenam(PP,SS,Po,So,m,o,d), ordena o     %
%           array PP segundo a métrica "m"                     %
%                                                              %
% - PP: contém os caminhos entre "o" e "d" a serem ordenados   %
% - SS: contém as respectivas sínteses dos caminhos PP         %
% - Po{m}{o}{d}: contém os caminhos entre "o" e "d" ordenados  %
%    para a métrica 'm';                                       %
% - So{m}{o}{d}: contém as correspondentes sinteses desses     %
%    caminhos;                                                 %
% - m: métrica utilizada para ordenação;                       %
% - o: nó de origem;                                           %
% - d: nó de destino;                                          %
% - PPo{o}{d}: contém os caminhos entre "o" e "d" parcialmente %
%              ordenados;                                      %
% - SSo{o}{d}: contém os caminhos entre "o" e "d" parcialmente %
%              ordenados.                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
posPP=[];
for i=1:size(PP,2)
   posPP(i)=findpos(Po{m}{o}{d},PP{i});
end
posPPo=[];
for i=1:size(PP,2)
   posPPo(i)=min(posPP);
   a=find(posPP==posPPo(i));
   if a==1
```

```
      posPP=posPP(2:size(posPP,2));
   elseif a==size(posPP,2)
      posPP=posPP(1:(size(posPP,2)-1));
   else
      posPP=[posPP(1:a-1) posPP(a+1:size(posPP,2))];
   end
end
for i=1:size(PP,2)
   PPo{i}=Po{m}{o}{d}{posPPo(i)};
   SSo{i}=So{m}{o}{d}{posPPo(i)};
end
```

A.6 Rotina GeraAnexthop

```
function [Nh,Nht] = geraAnexthop(Pfo,Nn);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [Nh,Nht] = geraAnexthop(Pfo,Nn), gera um array com %
%           os next hop para cada par oringem-destino a partir %
%           dos caminhos listados em Pfo{1}, em que:           %
%     - Nh{o}{d}{:}=[] [] []: next hop dos caminhos;           %
%     - Nht{o}{d}=[]: indica o número total de Nh diferentes   %
%              existentes                                      %
%     - Pfo{1}{o}{d}{:}=[] [] [] caminhos ordenados a partir do%
%                       melhor para o pior;                    %
%     - Nn: é o número total de nós da rede;                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nh={};
Nht={};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Varre os caminhos em Pfo{1} e os nós de origem e destino.    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for o=1:Nn
   for d=1:Nn

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      % Verifica se há caminho e conta o número de next hops   %
```

```
24          % diferentes para o mesmo par origem destino.            %
25          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26          if ~isempty(Pfo{1}{o}{d})
27             if ~isempty(Pfo{1}{o}{d}{1})
28                n=1;
29                for c=1:size(Pfo{1}{o}{d},2)
30                   if ~isempty(Pfo{1}{o}{d}{c})
31
32                      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
33                      % Verifica se não é o primeiro next hop a   %
34                      % ser salvo.                                %
35                      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36                       if n>1
37                         flag=0;
38                         for i=1:n-1
39                            if Nh{o}{d}{i}==Pfo{1}{o}{d}{c}(1,2)
40                               flag=1;
41                            end
42                         end
43                         if flag==0
44                            Nh{o}{d}{n}=Pfo{1}{o}{d}{c}(1,2);
45                            n=n+1;
46                         end
47                      else
48                         Nh{o}{d}{n}=Pfo{1}{o}{d}{c}(1,2);
49                         n=n+1;
50                      end
51                   end
52                end
53                Nht{o}{d}=size(Nh{o}{d},2);
54
55             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
56             % Se não há caminhos                                   %
57             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
58             else
59                Nh{o}{d}=[];
60                Nht{o}{d}=[];
61             end
```

```

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Se não há caminhos                                    %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        else
           Nh{o}{d}=[];
           Nht{o}{d}=[];
        end
     end
  end
```

A.7 Rotina GeraMnexthop

```
function [NH] = geraMnexthop(Nh,Nn,E);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [Nh,Nht] = geraMnexthop(Pfo,Nn), a matriz de NH    %
%        para o estado da rede para cada par (o,d) a partir de %
%        Nn e do estado E da rede, em que:                     %
%     - Nh{o}{d}{:}=[] [] []: next hop dos caminhos;           %
%     - Nn: é o número total de nós da rede;                   %
%     - E: variável do estado da rede. E=1: estado inicial.    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
NH=zeros(Nn);
for o=1:Nn
   for d=1:Nn
      if ~isempty(Nh{o}{d})
         NH(o,d)=Nh{o}{d}{E};
      end
   end
end
```

A.8 Rotina Calcdtree

```
function [Cd] = calcdtree(NH,Nn);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Função: [Cd] = calcdtree(NH,Nn), calcula o array c{n} para %
%           cada nó de destino em que:                         %
```

```
5   %     - Cd{n}: é a matriz de conexõa direta para a arvore de   %
6   %              destino "n";                                     %
7   %     - NH: é a matriz de next hop da rede;                     %
8   %     - Nn: é o número total de nós da rede.                    %
9   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10  for d=1:Nn
11     Cd{d}=zeros(Nn);
12     for o=1:Nn
13        if NH(o,d)~=0
14           Cd{d}(o,NH(o,d))=1;
15        end
16     end
17  end
```

A.9 Rotina Buscaloop

```
1   function [NC,LO,NA] = buscaloop(C,d,Nh,Nht);
2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3   % Função:[NC,LO,NA] = buscaloop(C,d,Nh,Nht), em que:            %
4   %   - C: é matriz de conexão vista pelo ponto de vista do nó   %
5   %          "d", em que:                                         %
6   %      - linhas são a origem das conexões pto-a-pto para chegar%
7   %        ao nó "d";                                             %
8   %      - colunas são o destino das conexões pto-a-pto para     %
9   %        chegar ao nó "d";                                      %
10  %   - d: é o nó de destino de referencia para a matriz C;      %
11  %   - Nh{o}{d}{:}: é a lista de next hops por par de nós       %
12  %     origem-destino;                                          %
13  %   - Nht{o}{d}{1}: é a lista do número total de next hop por  %
14  %     par origem-destino                                       %
15  %                                                              %
16  %  A soma na vertical indica a quantidade de nós com conexão   %
17  %  direta que chega a um destino:                              %
18  %     - nós que cuja soma vertical é zero não pertence a nenhum%
19  %       loop pois nenhuma outra conexão chega a ele.           %
20  %       OBS: o nó de destino a ser analizado deve ter essa soma%
21  %            maior ou igual a 1. Se for zero, nenhum nó chega  %
```

```
22  %            ao destino.                                          %
23  %     - como a topologia é observada a partir de um nó de des- %
24  %       tino "d", as conexões sempre chegam a ele e, portanto, %
25  %       o nó "d" não pertence a nenhum loop;                     %
26  %                                                                %
27  %  A soma na horizontal sempre tem de ser igual a 1 para os nós%
28  %  que não são o de destino "d", em que:                        %
29  %  - II: vetor con os nós a serem evitados (nós conectados a    %
30  %        algun ciclo a qualquer distancia dos ciclos e eo nós   %
31  %        dos ciclos);                                           %
32  %        OBS: não importa a correlação dos nós ligados a algum %
33  %        ciclo con o ciclo pois estes nós devem ser evitados    %
34  %        por qualquer nó da rede;                               %
35  %  - NC: nó que deve alterar a sua rota;                        %
36  %  - ID: vetor com os nós que não não pertencem a nenhum ciclo;%
37  %  - IC: vetor com os nós que pertencem a algun ciclo;          %
38  %  - LO: lista de ciclos existentes;                            %
39  %  - NA: nó de next hop escolhido para NC.                      %
40  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41  NCC=[];
42  CR=C;
43  I=[1:size(C,2)]';
44  IC=I;
45
46  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
47  % Reduz matriz de ciclo extraindo o destino e extraindo os nós %
48  % ligados diretamente ao destino.                                %
49  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50  CR=[CR(1:(d-1),:); CR((d+1):size(CR,1),:)];
51  IC=[IC(1:(d-1),:); IC(d+1:size(IC,1),:)];
52  CR=[CR(:,1:(d-1)),CR(:,(d+1):size(CR,2))];
53  D0=find(C(:,d)~=0)';
54  ID=[d; D0'];
55  for j=1:size(D0,2),
56     i=find(IC==D0(j));
57     CR=[CR(1:(i-1),:); CR((i+1):size(CR,1),:)];
58     IC=[IC(1:(i-1),:); IC((i+1):size(IC,1),:)];
59     CR=[CR(:,1:(i-1)),CR(:,(i+1):size(CR,2))];
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Encontra os destino que não tem nenhuma conexão chegando a  %
%  ele e, portanto, não pertencem a nenhum ciclo. Reduz matriz %
%  de ciclo. CR e IC são respectivamente a matriz de ciclo e   %
%  os respctivos nós (indices).                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while ~isempty(find(sum(CR)==0))& ~isempty(CR)
   D=find(sum(CR)==0);
   ID=[ID; IC(D(1))];
   CR=[CR(1:(D(1)-1),:); CR((D(1)+1):size(CR,1),:)];
   IC=[IC(1:(D(1)-1),:); IC(D(1)+1:size(IC,1),:)];
   CR=[CR(:,1:(D(1)-1)),CR(:,(D(1)+1):size(CR,2))];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Verfica se há ciclos.                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
NC=[];
if ~isempty(CR)

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  Encontra nós que conectados a algun ciclo e a qualquer   %
   %  distancia dos ciclos II, dadoa os nós que:               %
   %  - não pertencem a nenhum ciclo: ID;                      %
   %  - pertencem a algun ciclo: IC.                           %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   II=IC;
   IDR=ID;
   I=1;
   while ~isempty(I)
      MIDv=zeros(size(C,1),1);
      MICv=zeros(1,size(C,1));
      for i=1:size(IDR,1)
         MIDv(IDR(i),1)=1;
      end
      for i=1:size(II,1)
```

```
      MICv(1,II(i))=1;
   end
   MID=[];
   MIC=[];
   for i=1:size(C,1)
      MID=[MID MIDv];
      MIC=[MIC;MICv];
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Indica nós (I) que se conectam a um ciclo.               %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   [I,J]=find(((MID&MIC)&C));
   II=[II;I];
   for i=1:size(I,1)
      IDR(find(IDR==I(i)))=[];
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define os ciclos e os coloca em ordem, em que:             %
%  - LO: array com os ciclos.                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LO=cell(1);
nc=1;
n=1;
while n<=size(CR,1)
   i=find(IC);
   i=i(1,1);
   LO{nc}=[IC(i,1)];
   IC(i,1)=0;
   n=n+1;
   nnc=find(CR(i,:)==1);
   while IC(nnc,1)~=0
      LO{nc}=[LO{nc};IC(nnc,1)];
      IC(nnc,1)=0;
      n=n+1;
      nnc=find(CR(nnc,:)==1);
```

```
      end
      nc=nc+1;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  Encontra os nós a serem notificados para trocarem seus   %
   %  caminhos ao destino d, levando em consideração que eles  %
   %  tem opções.                                              %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   for j=1:size(II,1)
      if Nht{II(j,1)}{d}>1

         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         % Verifica se os nós têm mais que um next hop e se há %
         % mais que um next hop, verificar entre os next hops  %
         % os nós que têm que ser evitados e os salva em NA.   %
         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         Na(j)=0;
         f=0;
         for k=1:Nht{II(j,1)}{d}
            if isempty(find(II==Nh{II(j,1)}{d}{k}))& f==0
               Na(j)=Nh{II(j,1)}{d}{k};
               f=1;
            end
         end
      else
         Na(j)=[0];
      end
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Varre todos os nós a serem evitados e escolhe o melhor Na,%
   % ou seja, o Na do nó que tem maior número total de conexões%
   % que chegam nele.                                          %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   if sum(Na)~=0
      VS=sum(C);
      rasc=find(Na);
```

```
      NA=[Na(rasc(1,1))];
      NC=[II(rasc(1,1),1)];
      for j=1:size(II,1)
         if (VS(II(j,1))>VS(NC))& Na(j)~=0
            NC=[II(j,1)];
            NA=[Na(j)];
         end
      end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Se não há opções de next hop que não pertença ao conjunto %
   % de nós a serem evitados.                                  %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   else
      NA=[0];
      NC=[0];
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Se não há loop.                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
else
   II=[];
   LO=cell(1,1);
   NC=[];
   NA=[];
end
```