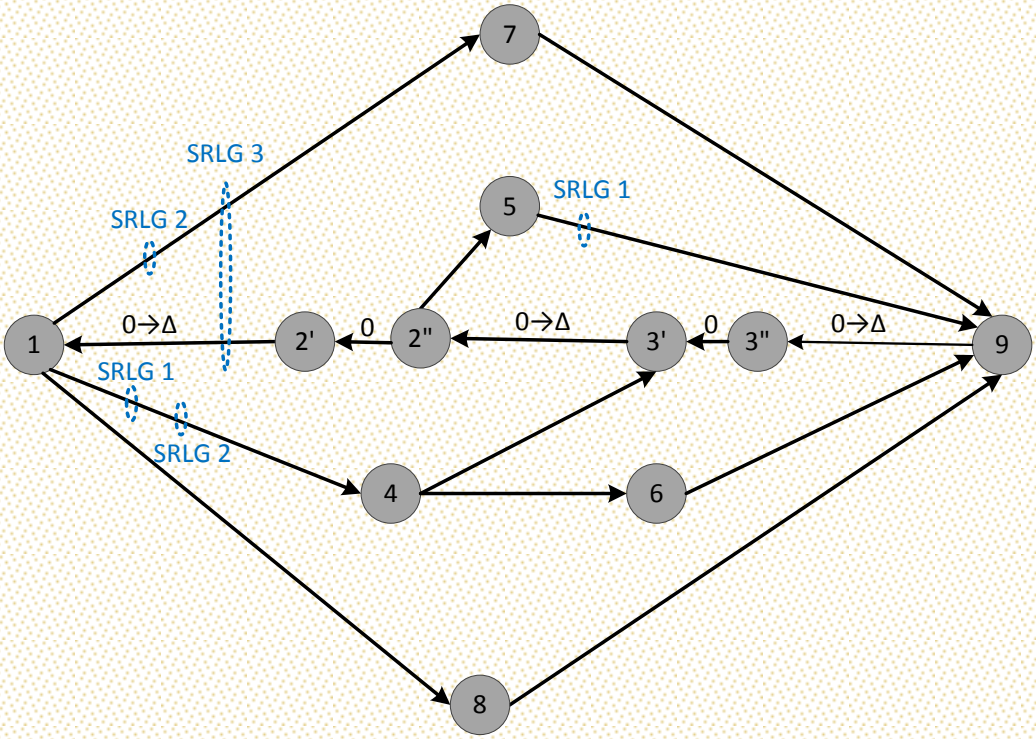


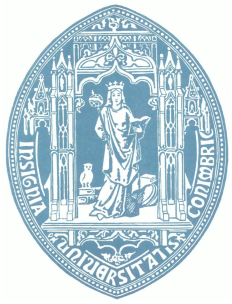

Miguel Filipe Rafael de Morais Soares

# Cálculo de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo, para utilização num PCE

24 de Janeiro de 2012

· U C ·

Universidade de Coimbra



FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

# Cálculo de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo, para utilização num PCE

Miguel Filipe Rafael de Morais Soares

Membros do Júri:
Presidente: Professor Doutor José Manuel Fernandes Craveirinha;
Orientadora: Professora Doutora Teresa Martinez dos Santos Gomes;
Vogal: Professor Doutor Carlos Alberto Henggeler de Carvalho Antunes

24 de Janeiro de 2012

# Agradecimentos

Este trabalho resulta de um esforço conjunto de várias pessoas que contribuíram decisivamente para a sua realização, às quais se manisfesta um reconhecido agradecimento.

À Professora Doutora Teresa Martinez dos Santos Gomes, pelos valiosos conhecimentos que me transmitiu com frontalidade, disponibilidade, dedicação, pela competente e notável orientação que me facultou em todas as fases e momentos do estudo e também pelo excelente apoio e ajuda prestada tanto em termos de procedimentos técnicos como teóricos na elaboração desta dissertação.

À Professora Doutora Luísa Jorge e ao Professor Doutor Paulo Melo, agradeço a disponibilidade que sempre demonstraram para me ajudar nos aspectos mais complexos ao nível informático que foram abordados neste trabalho.

À PT Inovação, e em particular ao Engenheiro Vitor Mirones e ao Engenheiro André Brízido, pela abertura, colaboração e simpatia com que me receberam em Aveiro, para realizar os testes de integração das rotinas desenvolvidas.

Ao INESC Coimbra e à FCT pela atribuição da bolsa de investigação U308 Plurianual 4/2011, que tornou este trabalho possível.

À família mais próxima, pais e irmã, pela compreensão, apoio e confiança com que aceitaram em muitos momentos prescindir da atenção e dedicação que lhes devo.

Aos funcionários do DEEC e do INESC, aos colegas e amigos, que desde sempre me apoiaram e incentivaram a prosseguir com o estudo, auxiliando-me assim a ultrapassar alguma barreiras com que me fui deparando, ao longo da elaboração deste trabalho.

e à Inês, por tudo aquilo que sempre me soube dar nos momentos mais pertinentes.

A TODOS O MEU MUITO OBRIGADO!

Miguel Morais Soares

# Resumo

Os operadores de telecomunicações sentem cada vez mais a necessidade de melhorar a fiabilidade e disponibilidade dos serviços oferecidos. A utilização de mecanismos de proteção, entre os quais se inclui a proteção global ao caminho, contribui para atingir esses objetivos. Essa proteção pode ser conseguida determinando pares (ou conjuntos) de caminhos disjuntos. A versatilidade que o *Generalized Multiprotocol Label Switching* (GMPLS) apresenta ao nível do plano de controlo, permite a implementação de mecanismos automáticos de proteção. A distribuição, pelos protocolos de encaminhamento, de informação relativa aos *Shared Risk Link Group* (SRLG) – grupo de ligações que partilham um risco de falha – torna possível a determinação de caminhos disjuntos nos SRLG, capazes de resistir a falhas múltiplas (desencadeadas pela falha associada a um dado risco). Os *Path Computation Elements* (PCE) que calculam caminhos numa rede GMPLS, podem dispor de poucos recursos (velocidade e memória).

Foram estudados e implementados algoritmos eficientes de determinação de pares de caminhos disjuntos (e um algoritmo de determinação de um conjunto de $K$ caminhos disjuntos) nos nós, de custo aditivo mínimo. Na implementação realizada procurou-se minimizar a utilização da memória sem comprometer a velocidade de execução, definindo uma representação para a rede baseada na *Reverse and Foward Star Form* (RFSF), adequada à transformação da rede requerida por estes algoritmos.

A determinação de um par de caminhos disjuntos nos SRLG é um problema NP-completo. Foram descritas e implementadas duas heurísticas, *Iterative Modified Suurballe's Heuristic* (IMSH) e *Confliting SRLG Exclusion-Min Sum* (CoSE-MS), para a determinação de pares de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo; foram ainda apresentadas duas novas variantes do CoSE-MS que são ligeiramente mais eficientes que a versão original.

Subjacente a este trabalho, esteve sempre o objetivo de obter uma biblioteca partilhada (.so) adequada para utilização num sistema embebido, em particular na placa UNICOM-V5. Os testes realizados demonstraram que as rotinas implementadas para o cálculo de caminhos disjuntos nos nós eram perfeitamente adequadas a este ambiente. As duas novas variantes do algoritmo CoSE-MS mostraram ser um bom compromisso entre eficiência computacional e precisão. A heurística IMSH mostrou ser mais adequada para um ambiente de gestão de rede, dada a sua precisão à custa de um maior tempo de execução.

**Palavras-Chave: proteção, caminhos disjuntos, min-sum, SRLG**

# Abstract

Telecom operators are increasingly feeling the need to improve the reliability and availability of the offered services. One way to achieve this goal is the use of protection mechanisms, among which is global path protection. Such protection can be achieved by determining pairs (or sets) of disjoint paths. The versatility of the *Generalized Multiprotocol Label Switching* (GMPLS) control plan, allows the implementation of automated protection. The distribution by routing protocols of information about *Shared Risk Link Groups* (SRLGs) – a group of links that share a risk of failure – makes it possible to determine SRLG disjoint paths, able to withstand multiple failures (triggered by a failure associated with a given risk). The *Path Computation Elements* (PCEs) which compute paths in a GMPLS network, may have limited resources (CPU and memory).

Efficient algorithms for determining pairs of disjoint paths (and an algorithm for determining a set of disjoint paths $K$), with minimal additive cost, were studied. The implementation of these algorithms sought to minimize memory usage without compromising execution speed, which required the definition of a network representation based on *Forward and Reverse Star Form* (RFSF), suitable for the network transformation required by the algorithms.

The determination of a pair of SRLG disjoint paths is NP-complete. Two heuristics, *Iterative Modified Suurballe's Heuristic* (IMSH) and *Confliting SRLG Exclusion-Min Sum* (CoSE-MS) for determination of SRLG disjoint path pairs, of minimum cost additive, were described and implemented; two new variants of COSE-MS, which are slightly more efficient than the original version, were also presented.

The underlying aim of this work has always been producing a shared library (.so) suitable for using in an embedded system, in particular on the UNICOM-V5 board. The tests showed that the routines implemented for the calculation of disjoint paths were perfectly suited to this environment. The two new variants of CoSE-MS algorithm proved to be a good compromise between computational efficiency and accuracy. The heuristic IMSH was shown to be more appropriate for network management due to its accuracy and higher computacional time.

**Keywords:protection, disjoint paths, min-sum, SRLG**

*"O único lugar onde o sucesso vem antes do trabalho é no dicionário."*

Albert Einstein

# Índice

# Lista de Figuras

# Lista de Tabelas

# Abreviaturas

| | |
|---|---|
| AP | Active Path |
| API | Application Programming Interface |
| BFS | Breadth-First-Search |
| BP | Backup Path |
| CoLE | Confliting Link Exclusion |
| CoSE | Confliting SRLG Exclusion |
| CoSE-MS | Confliting SRLG Exclusion - Min Sum |
| FRSF | Foward and Reverse Star Form |
| FSF | Foward Star Form |
| GMPLS | Generalized Multiprotocol Label Switching |
| IC | Intervalo de Confiança |
| IMSH | Iterative Modified Suurballe's Heuristic |
| ITSA | Iterative Two Step Approach |
| MBH | Modified Bhandari's Heuristic |
| MPLS | Multiprotocol Label Switching |
| MSH | Modified Suurballe's Heuristic |
| PCC | Path Computation Client |
| PCE | Path Computation Element |
| PLI | Programação Linear Inteira |
| RHE | Recovery Head-End |
| RSF | Reverse Star Form |
| RTE | Recovery Tail-End |
| SRLG | Shared Risk Link Group |
| TED | Traffic Engeneering Database |
| TSA | Two Step Approach |

# Capítulo 1

# Introdução

## 1.1 Motivação

Atualmente, devido à elevada largura de banda que as redes óticas apresentam, o volume de dados transportados por estas redes é muito grande. Como tal, uma avaria durante um período de tempo reduzido pode deixar um número muito grande de utilizadores sem serviço. Esta situação pode ter custos muito elevados, pelo que se tem investido no desenvolvimento de tecnologias que permitem prevenir e recuperar falhas, de forma a que estas não se manifestem como avarias.

Um conceito usual em proteção de redes é o conceito de "Shared Risk Link Group" (SRLG). Um SRLG é um grupo de ligações que partilham o mesmo recurso físico (cabo, conduta, nó, etc) cuja falha resulta na falha de todas as ligações desse grupo [17]. Considere-se por exemplo, uma conduta onde passa um conjunto de cabos de transmissão e que cada cabo representa uma ligação. Se essa conduta for acidentalmente cortada, todos os cabos nessa conduta são cortados e consequentemente todas as ligações associadas a esses cabos falham. Assim, o risco dessas ligações falharem é comum ao risco da conduta ser cortada. Logo, todas as ligações que utilizem os cabos dessa conduta estão associadas ao mesmo SRLG.

A determinação de um par de caminhos disjuntos nos SRLG é um problema NP-completo [7]. Em [7] encontra-se uma formalização para este problema. Contudo, a resolução exata deste problema pode requerer um tempo de resolução elevado, pelo que não é compatível com a determinação de caminhos a pedido do plano de controlo numa rede de telecomunicações. Assim, têm sido propostas heurísticas para a determinação de pares de caminhos disjuntos nos SRLG [18, 6, 13, 14, 9, 4].

## 1.2 Objetivos

O principal objetivo deste trabalho foi a implementação de um algoritmo de cálculo de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo, que fosse adequado para utilização de um *Path Computing Element* (PCE) de uma rede *Generalized Multiprotocol Label Switching* (GMPLS) [3]. Para tal, foram previamente selecionados as heurísticas *Iterative Modified Suurballe's Heuristic* (IMSH) [13] e *Conflicting SRLG Exclusion – Min Sum* (CoSE-MS) [4].

O segundo objetivo deste trabalho foi o estudo e a implementação de algoritmos para a determinação de um par de caminhos disjuntos nos nós, de custo aditivo mínimo [2], procurando utilizar o mínimo de memória possível sem comprometer o tempo de execução dos algoritmos.

O terceiro objetivo foi o cálculo de $K$ caminhos disjuntos nos nós, de custo aditivo mínimo, o que foi conseguido à custa de uma extensão de um dos algoritmos de cálculo de pares de caminhos disjuntos [2].

O objetivo final é estudar o desempenho (tempo de execução e qualidade das soluções obtidas) de todos os algoritmos num PCE[1].

Embora em [4] tenha ficado clara a vantagem da utilização, em termos de tempo de execução do CoSE-MS face ao IMSH, procurou verificar-se se esse resultado se mantinha ao limitar o número de iterações do IMSH e o número de problemas resolvidos pelo CoSE-MS.

## 1.3 Enquadramento

Como foi anteriormente referido, este trabalho visa a criação de rotinas que permitam calcular pares de caminhos disjuntos nos nós, ou nos nós e nos SRLG, de custo aditivo mínimo, e conjuntos de $K$ caminhos disjuntos nos nós, de custo aditivo mínimo. Essas rotinas serão integradas em mecanismos de recuperação de uma rede que usa a tecnologia GMPLS. Como tal, esta secção visa descrever (de forma geral) uma rede GMPLS e a sua arquitetura base.

### 1.3.1 Perspetiva geral

O GMLPS, como o nome indica, é uma generalização do MPLS [15]. Assim, o GMPLS faz com que seja possível o MPLS intervir ao nível do plano de controlo (sinalização

[1] O PCE utilizado foi uma placa UNICOM-V5, gentilmente cedida pela PT Inovação.

e encaminhamento) em dispositivos que usem qualquer um dos tipos de comutação: de pacotes, no tempo e no comprimento de onda.

A versatilidade que o GMPLS apresenta ao nível plano de controlo, permite a criação de mecanismos automáticos que: estabeleçam ligações ponto a ponto; administrem os recursos disponíveis numa rede e garantam a qualidade de serviço necessária para que as mais modernas aplicações funcionem sem problemas.

### 1.3.2 Arquitetura baseada em PCE

O cálculo de caminhos em redes de grandes dimensões, com múltiplos domínios ou múltiplas camadas, é complexo e exige uma grande capacidade de computação[2]. Uma forma de resolver este problema é distribuir a capacidade de computação pelos nós da rede. Assim, a arquitetura base usado pelo GMPLS consiste em usar PCE em vários pontos da rede [3].

Um PCE é uma unidade computacional que calcula um caminho a pedido de um "Path Computation Client" (PCC), que é um elemento da rede que pretende determinar um ou vários caminhos no domínio da rede a que pertence. Para determinar um caminho, um PCE tem que recorrer a uma base de dados com a informação relativa ao estado da rede. Essa base de dados é denominada por "Traffic Engeneering Database" (TED).

Neste contexto o cálculo de rotas pode ser realizado recorrendo a um PCE, de forma centralizada ou distribuída. Um PCE centralizado possui, em geral, grande capacidade de processamento, onde o tempo de resposta poderá ser da ordem dos segundos, pois responde a pedidos do sistema de gestão de rede. Contudo, se o PCE em causa estiver localizado num *router* – como pode acontecer num modelo distribuído – em geral o seu poder de cálculo e os seus recursos de memória são limitados, devendo contudo ter uma resposta rápida aos pedidos que lhe são feitos.

As rotinas que foram implementadas no âmbito deste trabalho, visam ser implementadas num PCE que poderá estar residente num *router*.

## 1.4 Conteúdo

No capítulo 2, é apresentada a notação usada neste trabalho, seguida da descrição de conceitos básicos de proteção em redes, sendo igualmente abordado o problema da determinação de caminhos disjuntos.

[2]Por capacidade de computação entende-se a capacidade de processamento juntamente com a memória RAM disponível.

No capítulo 3 são apresentados os algoritmos que permitem calcular um par de caminhos disjunto nos nós de custo aditivo mínimo. É também apresentado um algoritmo que permite calcular um conjunto de $k$ caminhos disjuntos nos nós de custo aditivo mínimo.

No capítulo 4 são descritos os algoritmos que permitem calcular um par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo.

O capítulo 5 é divido em duas partes. A primeira parte apresenta os resultados relativos aos algoritmos que não consideram SRLG e a segunda parte apresenta os resultados relativos aos algoritmos que consideram SRLG.

Finalmente, no capítulo 6, são apresentadas algumas conclusões finais.

# Capítulo 2

# Proteção em redes

## 2.1 Notação, conceitos e definições

- **Grafo dirigido:** O grafo $G = (V,A)$ é definido por um conjunto de nós $V$, $V = \{v_1, \ldots, v_n\}$ e por um conjunto de arcos $A$, $A = \{a_1, \ldots, a_m\}$.

  Um arco liga dois vértice por uma dada ordem, sendo um par ordenado de elementos pertencentes a $V$. Se $a, b \in V$, com $a \neq b$ e $(a,b) \in A$, diz-se que $a$ é a **cauda** (ou origem) do arco e $b$ a sua **cabeça** (ou destino).

- **Grafo não dirigido:** O grafo $G = (V,E)$, é definido por um conjunto de nós $V$, $V = \{v_1, \ldots, v_n\}$ e por um conjunto de arestas $E$, $E = \{e_1, \ldots, e_m\}$, em que uma aresta é um par de nós (por qualquer ordem).

  Se $a, b \in V$, $a \neq b$, a aresta que liga esse dois nós pode ser representada por $(a,b) \in E$ ou $(b,a) \in E$, em que $a$ e $b$ são os **extremos** da aresta.

- **Arco simétrico:** Considere-se o arco $(i, j)$. O simétrico do arco $(i, j)$ é o arco $(j,i)$.

- **Custo de um arco:** Indica o custo de utilização desse arco num caminho. É representada por $l(a,b)$, onde $a$ é o nó de origem e $b$ é o nó de destino do arco $(a,b)$.

- **Rede dirigida:** Uma rede dirigida é representada por um grafo dirigido a cujos arcos foram associados atributos, como por exemplo, o custo de utilizar um dado arco.

- **Rede não dirigida:** Uma rede não dirigida é representada por um grafo não dirigido a cujas arestas foram associados atributos, como por exemplo, o custo de utilizar uma dada aresta.

Uma rede não dirigida pode ser computacionalmente representada por uma rede dirigida, em que cada aresta é substituída por dois arcos, de sentidos opostos, entre o mesmo par de nós (com os mesmos atributos).

- **Caminho:** Sequência contínua de nós (todos diferentes) desde um nó de **origem**, $i$, até um nó de **destino**, $j$, $i, j \in V$. Um caminho é representado por $p = \langle i \equiv v_1, v_2, \ldots, j \equiv v_l \rangle$, onde $(v_k, v_{k+1}) \in A, \forall k \in \{1, \ldots, l-1\}$.

  Seja $A_p$ o conjunto de arcos que formam o caminho $p = \langle i \equiv v_1, v_2, \ldots, j \equiv v_l \rangle$: $A_p = \cup_{\forall k \in \{1,\ldots,l-1\}} (v_k, v_{k+1})$. Seja $V_p$ o conjunto dos nós do caminho $p$.

- **Custo (aditivo) de um caminho:** Somatório do custo dos arcos que constituem o caminho $p$, $C_p = \sum_{(i,j) \in A_p} l(i,j)$, onde $A_p$ é o conjunto dos arcos que constituem o caminho $p$.

  Se um caminho entre um dado par de nós não existe, é representado pelo conjunto vazio ($\emptyset$), e o seu custo é infinito.

- **Par de caminhos:** Conjunto de dois caminhos com o mesmo nó de **origem** e o mesmo nó de **destino**. Um par de caminhos é representado por $(p,q)$, sendo $p$ e $q$ os caminhos constituintes do par.

- **Par de caminhos disjuntos nos nós:** Seja $(p,q)$ um par de caminhos de um nó origem $s$ para um nó de destino $t$. $(p,q)$ é um par de caminhos disjuntos nos nós, se e só se $V_p \cap V_q = \{s,t\}$.

- **Conjunto de $K$ caminhos:** Conjunto de $K$ caminhos com o mesmo nó de **origem** e o mesmo nó de **destino**. Um conjunto de $K$ caminhos é representado por $S$, onde $K = |S|$.

  Os caminhos no conjunto $S$, do nó $s$ para o nó $t$ são todos disjuntos entre si, se e só se: $\cup_{p \in S} V_p = \{s,t\}$

- **Segmento:** Sequência contínua de arcos que constituem parte de um caminho.

- $C_{(p,q)}$**:** Custo (aditivo) de um par de caminhos $(p,q)$, dado pela soma do custo dos caminhos que constituem o par, $C_{(p,q)} = C_p + C_q$.

  Se $(p,q) = (\emptyset, \emptyset)$, o custo do par de caminhos é infinito ($C_{(\emptyset,\emptyset)} = \infty$).

- $C_S$**:** Custo de um conjunto de caminhos $S$, dado pelo somatório do custo dos caminhos nesse conjunto, $C_S = \sum_{p \in S} C_p$.

- $d(i)$**:** Distância do nó $i$ ao nó de origem, ou seja, a soma do custo de todos os arcos que formam o caminho mais curto desde o nó origem até ao nó $i$ (o qual pode ser determinado usando o algoritmo de Dijkstra, desde que os custos sejam não negativos e não hajam custos fixos associados à passagem em nós).

- $P(i)$**:** Nó predecessor do nó $i$ num dado caminho.

- **Nó adjacente:** Um nó, $i$, diz-se adjacente de um outro nó, $j$, quando o nó $i$ se encontra ligado através de um arco ao nó $j$. Desta definição resulta que o nó $i$ é adjacente ao nó $j$ se existir o arco $(i, j)$ ou o arco $(j, i)$ (ou ambos).

- **Nó vizinho:** Um nó, $j$, diz-se vizinho de um outro nó, $i$, se e só se o nó $j$ estiver ligado ao nó $i$ através do arco $(i, j)$.

- $\Gamma_i$**:** Conjunto de nós vizinhos do nó $i$.

- $X$**:** Conjunto dos riscos de falha que podem afetar os arcos da rede. $X = \{x_1, x_2, \ldots, x_r\}$ em que $r$ é o número de riscos na rede.

- $g_i, i = 1, \ldots, r$**:** $g_i$ é o conjunto de arcos da rede que ficam indisponíveis quando ocorre a falha associada ao risco $x_i$ ($x_i \in X$). Ou seja $g_i$ é um SRLG.

- $R'$**:** Conjunto de todos os SRLG existentes na rede. $R' = \{g_1, g_2, \ldots g_r\}$ em que $r$ é o número de SRLG diferentes que existem numa rede.

- $R(i, j)$ **ou** $R(a)$ **com** $a = (i, j)$**:** Conjunto dos SRLG a que o arco $a = (i, j)$ pertence. Das definições anteriores resulta que $R' = \cup_{(i,j)\in A} R(i, j)$, e $r = |R'|$.

- $R$**:** Conjunto dos conjuntos de SRLG de todos os arcos da rede: $R = \{R(i, j) : (i, j) \in A\}$, sendo $G = (V, A)$ o grafo associado a essa rede ($|R| = |A|$).

- $R_p$**:** Conjunto dos SRLG que afetam um caminho $p$, ou seja, $R_p = \cup_{(i,j)\in p} R(i, j)$.

- $\mathscr{P}_{st}$**:** Conjunto de todos os caminhos de $s$ para $t$ na rede.

- $(p^*, q^*)$**:** Par de caminhos disjuntos nos nós, ou nos nós e nos SRLG, dependendo do problema a resolver, de menor custo aditivo, ou seja, uma solução ótima para o problema em causa.

Neste trabalho os termos grafo e rede serão usados indistintamente, sendo o termo grafo utilizado preferencialmente quando se pretende fazer referência à topologia de uma rede.

## 2.2 Esquemas de proteção

### 2.2.1 Introdução

Na secção 1.1 foram referidos os aspectos que motivam a proteção em redes de transmissão.

Assim sendo, é necessário implementar mecanismos de recuperação adequados a cada aplicação. Para tal, é necessário ter em consideração determinados critérios para se optar pelo tipo de mecanismo a utilizar. Esta secção enuncia os aspetos a ter em conta na implementação de mecanismos de recuperação. Os mecanismos de recuperação podem ser divididos em dois grupos: mecanismos de recuperação de *camada única* ou *multi-camada* [15].

No texto que se segue *AP* significa "Active Path" e *BP* "Backup Path". Um *AP* é o caminho ativo entre dois nós de uma rede (usado quando não há falhas). O *BP* é um caminho alternativo que permite que a comunicação continue, havendo uma falha que torne o *AP* inoperacional. São também usados os termos "Recovery Head-End" (RHE) e "Recovery Tail-End" (RTE). Estes termos referem-se ao primeiro nó (RHE) e ao último nó (RTE) do segmento alternativo ao segmento do *AP* onde ocorre a falha. Considere-se o $AP = \langle 1,2,3,4,5 \rangle$. Admitindo que ocorre uma falha que torna inoperacional o segmento $\langle 2,3,4 \rangle$ e que o segmento alternativo é o segmento $\langle 2,6,7,8,4 \rangle$, o RHE é o nó 2 e o RTE é o nó 4.

### 2.2.2 Mecanismo de recuperação de camada única

Os mecanismos de recuperação de camada única podem ser *dedicados* ou *partilhados*. Num mecanismo de proteção dedicado, para um dado *AP*, há um *BP* cujos recursos só são usados caso haja uma falha nesse *AP*. Para implementar este tipo de mecanismos de recuperação é preciso uma grande quantidade de recursos. Para resolver este problema, nos mecanismos de recuperação partilhados, um *BP* pode ser usado para transportar tráfego adicional, enquanto não ocorrer nenhuma falha. Assim que ocorrer uma falha, o *BP* é usado para recuperar dessa falha.

Implementar mecanismos de recuperação partilhados é muito mais complicado do que implementar mecanismos de recuperação dedicados, pois após uma falha é necessário verificar se os recursos reservados para a proteção de um *AP* não estão a ser utilizados na recuperação de nenhum outro *AP*, com o qual partilha recursos de proteção. Por outro lado, os mecanismos de recuperação partilhados são mais eficientes do que os mecanismos de proteção dedicados, dado que são mais flexíveis.

Um outro critério a ter em conta é o modo como o cálculo do *BP* é feito. Há duas alternativas para calcular um *BP*: *pré-planeamento* ou *planeamento dinâmico*. No caso do pré-planeamento, o *BP* é calculado antes de ocorrer a falha. No planeamento dinâmico, o *BP* só é determinado depois de ocorrer a falha.

Os mecanismos de recuperação com pré-planeamento permitem uma recuperação mais rápida do que os mecanismos de recuperação com planeamento dinâmico, dado que os mecanismos de recuperação com planeamento dinâmico demoram algum tempo a calcular o *BP* depois de ocorrer a falha. Por outro lado, devido à sua flexibilidade, os mecanismos de proteção com planeamento dinâmico permitem maior adaptação ao tráfego existente nos vários arcos da rede. Um outro fator que deriva desta flexibilidade é que os mecanismos de recuperação com planeamento dedicado são capazes de encontrar um *BP*, mesmo para situações de falha que à partida não tenham sido consideradas. Já os mecanismos de recuperação com pré-planeamento não têm esta capacidade.

É importante distinguir *proteção* de *reencaminhamento* (ou *restauro*). Ambas as técnicas envolvem sinalização, mas a altura em que essa sinalização é feita não é a mesma. Em proteção, os *BP* são calculados de forma pré-planeada e sinalizados antes de ocorrer uma falha. Já no reencaminhamento os *BP* podem ser calculados de forma pré-planeada ou de forma dinâmica. No entanto, quando ocorre uma falha é necessário sinalização adicional para estabelecer os *BP*.

A grande vantagem da proteção face ao reencaminhamento é a sua rapidez no tempo de recuperação, dado que não há necessidade de gerar a sinalização para estabelecer os *BP*. No entanto, o reencaminhamento apresenta uma maior flexibilidade na recuperação de falhas. As técnicas de reencaminhamento, devido à sua natureza partilhada, necessitam de ter menos recursos reservados para recuperar de uma falha.

Os vários esquemas de proteção, podem ser classificados como:

$1+1$ **(Proteção Dedicada)** Cada *BP* protege exatamente um *AP*. Esta técnica pode ser implementada de duas maneiras. Na primeira, o sinal é duplicado no $RHE$, enviado simultâneamente pelo *AP* e pelo *BP*, e no $RTE$ o sinal maior de qualidade é escolhido e é entregue ao recetor. Uma outra forma de implementar este esquema de proteção é estabelecer um limiar, abaixo do qual o $RTE$ começa a entregar ao recetor o sinal proveniente do *BP*.

$1:1$ **(Proteção Dedicada com Tráfego Adicional)** Cada *BP* protege exatamente um *AP*. No entanto, enquanto não houver nenhuma falha no *AP*, o tráfego é transmitido exclusivamente pelo *AP*. Neste caso, o *BP* fica livre para transmitir tráfego adicional, enquanto não ocorrer nenhuma falha. Logo que ocorra uma falha o *BP* deixa de

transportar tráfego adicional e passa a transportar a informação transmitida pelo *AP*.

$1:N$ **(Recuperação Partilhada com Tráfego Adicional)** Um *BP* é encarregue de proteger *N AP*. Enquanto não houver nenhuma falha, o *BP* pode transportar tráfego adicional. Logo que ocorra uma falha o *BP* deixa de transportar tráfego adicional e passa a transportar a informação transmitida pelo *AP*.

$M:N$ $(M \leq N)$ Um conjunto de *M BP* protegem um conjunto de *N AP*. Enquanto não houver nenhuma falha, os *M BP* podem ser usados para transportar tráfego adicional. Logo que ocorra uma falha os *M BP* deixam de transportar tráfego adicional e passam a transportar a informação transmitida pelos *N AP*.

Os esquemas de recuperação podem ser classificados, em termos de proteção, como sendo *locais*, *globais* ou *ao segmento*. Nos esquemas de recuperação local, só os segmentos do *AP* que são afetados por uma falha é que são evitados. Já num esquema de recuperação global, caso haja uma falha, num segmento de um *AP*, todo o caminho é evitado. Considere-se agora que um segmento pode ser constituído por múltiplos segmentos, denominados sub-segmentos. Num esquema de recuperação ao segmento, caso haja uma falha num sub-segmento do *AP*, todo o segmento a que esse sub-segmento pertence é evitado.

Quanto à escalabilidade, os mecanismos de recuperação podem ser classificados em duas classe: *centralizados* e *descentralizados* (ou *distribuídos*). Numa arquitetura centralizada os mecanismos de recuperação dependem de um sistema central. Caso ocorra uma falha esse sistema deteta-a, recolhe informação sobre o estado da rede e toma as ações necessárias para que a rede recupere da falha. Numa arquitetura distribuída, a capacidade de processamento de uma unidade central é distribuída pelos vários nós da rede. Em caso de falha, cada nó que é afetado pela falha, inicia os seus mecanismos de recuperação. Nesta arquitetura não há um perspetiva geral da rede, mas sim um perspectiva local (e parcial) que cada nó tem da rede. Assim, os nós têm que trocar informação para poderem ter uma visão atualizada da rede para iniciarem os seus mecanismos de recuperação. Há ainda uma outra possibilidade híbrida, em que o cálculo dos *BP* é feito num sistema central, mas cada nó toma as medidas que forem necessárias para iniciar os mecanismos proteção, usando o *BP* calculado no sistema central.

### 2.2.3 Mecanismo de recuperação de camadas múltiplas

Numa arquitetura de múltiplas camadas é necessário coordenar a interação entre as várias camadas. Há duas formas de fazer essa coordenação: *aproximação sequencial* e

*aproximação integrada*.

Na aproximação sequencial, os mecanismos de proteção são iniciados por ordem cronológica, da camada mais baixa (camada cliente) até à camada mais alta (camada servidor). Isto evita situações em que os mecanismos de recuperação das duas camadas sejam iniciados ao mesmo tempo, entrando em competição um com o outro. Há duas possibilidades para implementar esta aproximação. A primeira consiste em introduzir um intervalo de espera ("hold-off time") no mecanismo da camada cliente, para que este só seja iniciado assim que o mecanismo de recuperação da camada servidor esteja terminado. Assim, se for possível resolver a falha recorrendo exclusivamente ao mecanismo de recuperação da camada servidor, a camada cliente não inicia o seu mecanismo de recuperação após o intervalo de espera. Uma forma alternativa de implementar esta aproximação consiste em considerar um alarme ("recovery token sinal") que é enviado do mecanismo de recuperação da camada servidor para o mecanismo de recuperação da camada cliente. Este alarme indica ao mecanismo de recuperação da camada cliente que o mecanismo de recuperação da camada servidor não foi capaz de recuperar da falha, necessitando que o mecanismo de recuperação da camada cliente seja ativado.

Na aproximação integrada, dois mecanismos de recuperação são combinados num esquema de recuperação de camada múltipla. Isto implica que os esquemas de recuperação tenham uma visão global de ambas as camadas, para poderem decidir em qual (ou quais) das camadas devem intervir. Esta aproximação é mais flexível do que a aproximação sequencial.

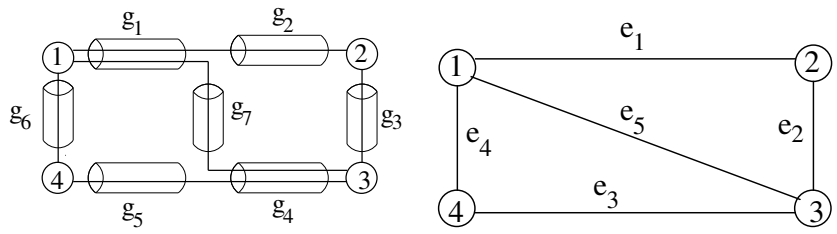
### 2.2.4 Conclusão

As rotinas desenvolvidas para determinação de pares de caminhos disjuntos (considerando ou não SRLG), podem ser classificadas como mecanismos de recuperação dedicados com pré-planeamento que usam técnicas de proteção global.

A rotina de determinação de $K$ caminhos disjuntos nos nós, pode ser utilizada num contexto de proteção partilhada $(1 : K-1)$ ou de proteção dedicada contra $K-1$ falhas simultâneas.

## 2.3 Arquitetura de uma rede ótica

Uma rede ótica pode ser vista como sendo constituída por duas camadas: camada física e camada ótica [7]. A camada física é constituída por nós (por exemplo, switches óticos) e pelas ligações físicas que os unem (por exemplo, cabos ou condutas). A camada




(a) Camada Física (b) Camada Ótica

Figura 2.1: Exemplo de uma rede ótica em duas camadas [4].

ótica é constituída pelas ligações lógicas da rede (uma ligação lógica pode conter uma ou mais ligações físicas).

O conceito de SRLG, introduzido na secção 1.1, veio fazer com que fosse possível à camada ótica determinar caminhos que sejam disjuntos do ponto de vista de falhas que podem afetar a camada física. Na figura 2.1, uma ligação protegida $(1+1)$ entre os nós 1 e 3, contra a falha de um SRLG, não pode usar a aresta $e_5$. Se for desejado um par de caminhos disjuntos no SRLG, a solução é $(\langle 1,2,3\rangle,\langle 1,4,3\rangle)$, onde $R_{\langle 1,2,3\rangle} = \{g_1, g_2, g_3\}$ e $R_{\langle 1,4,3\rangle} = \{g_4, g_5, g_6\}$, obtendo-se $R_{\langle 1,2,3\rangle} \cap R_{\langle 1,4,3\rangle} = \emptyset$.

## 2.4 Determinação de caminhos disjuntos

A determinação de um par de caminhos disjuntos nos arcos (nos nós), de custo aditivo mínimo visa, em geral, minimizar o custo dos recursos utilizados em proteção global dedicada. Este problema, designado por min-sum, resolve-se em tempo polinomial utilizando os algoritmos de Suurballe e Bhandari [2]. Quando se deseja partilhar largura de banda de proteção pode utilizar-se uma aproximação do tipo min-min, ou seja, procura determinar-se para *AP* o caminho mais curto que é possível proteger, e para *BP* o caminho mais curto na rede em que foram removidos os arcos (ou nós) do caminho a proteger. Este problema é NP-completo [16]. No contexto de partilha de largura de banda, a utilização de recursos da rede pelo *AP*, pode ser considerada mais importante do que a do *BP*. Isto pode conduzir ao problema min-sum com pesos assimétricos, em que o custo do *AP* é considerado $\omega$ vezes mais relevante do que o custo do *BP* [8]. Este problema é também NP-completo [16].

Os problemas min-sum e min-min podem ser formulados, considerando também a necessidade dos caminhos serem disjuntos nos SRLG. Neste caso, o problema min-sum passa a ser NP-completo [7]. Assim, têm sido propostas várias heurísticas para a sua resolução. As heurísticas *"Trap Avoidance"* (TA) [18] e CoSE [9] resolvem de forma bastante eficiente o problema min-min, considerando SRLG. Por outro lado, as heurísticas

*"Iterative Two Step Approach"* (ITSA) [6], IMSH [13] e CoSE-MS [4] procuram resolver o problema min-sum de cálculo de um par de caminhos disjuntos nos nós e nos SRLG com custo aditivo mínimo.

# Capítulo 3

# Algoritmos implementados não considerando SRLG

## 3.1 Introdução

Dado uma rede, cuja topologia é representada pelo grafo, $G = (V,A)$, dado $l(i,j)$ o custo associado a cada arco $(i,j) \in A$ e dado um par de nós, origem e destino ($s,t \in V$), deseja obter-se o par de caminhos ($p^*$ e $q^*$) disjuntos nos nós de menor custo aditivo. Este problema pode ser formalizado da seguinte forma:

$$(p^*,q^*) = \arg \min_{(p,q)\in\mathscr{P}_{st}} C_p + C_q \tag{3.1}$$

$$\text{s. a:} \quad V_p \cap V_q = \{s,t\} \tag{3.2}$$

o qual é resolvido em tempo polinomial pelos algoritmos de Suurballe ou Bhandari [2]. Estes algoritmos, com as devidas adaptações, são utilizados nos algoritmos de determinação de pares de caminhos disjuntos nos nós e nos SRLG, descritos no capítulo 4, pelo que se considerou importante descrevê-los aqui em detalhe.

Se for desejada, proteção contra $K-1$ falhas na rede poderá utilizar-se um conjunto de $K$ caminhos disjuntos nos nós. Este problema pode ser formalizado da seguinte forma:

$$S^* = \arg \min_{S\subset\mathscr{P}_{st}} C_S \tag{3.3}$$

$$\text{s. a:} \quad \cap_{p\in S} V_p = \{s,t\} \tag{3.4}$$

$$|S| = K \tag{3.5}$$

em que $S$ é um conjunto de $K$ caminhos disjuntos nos nós contido em $\mathscr{P}_{st}$. Em [2] é apresentado um algoritmo que permite obter um conjunto de $K$ caminhos disjuntos nos

nós e de custo aditivo mínimo, em tempo polinomial.

O resto deste capítulo encontra-se organizado como seguidamente se descreve. Na secção 3.2 é referida a importância da representação da rede para a eficiência computacional das rotinas implementadas. Na secção 3.3 é exemplificada a transformação da rede necessária para obter caminhos disjuntos nos nós. Seguidamente, na secção 3.4, são descritos os algoritmos de Bhandari e de Suurballe, que permitem calcular um par de caminhos disjuntos nos nós de custo aditivo mínimo. Finalmente, na secção 3.5 é apresentado o algoritmo de Bhandari adaptado, para que seja possível determinar um conjunto de $K$ caminhos disjuntos nos nós de custo aditivo mínimo, o qual é aqui designado por $K$-Bhandari.

## 3.2 Representação da rede

Uma parte fulcral deste projeto prende-se com a eficiência ao nível da velocidade de processamento, sem descurar a utilização da memória RAM, dado que o "software" implementado se destina a ser usado num sistema embebido. Daqui advém a importância de escolher uma forma adequada de guardar em memória a representação da rede.

Para obter um bom desempenho ao nível do tempo de processamento é preciso garantir que a estrutura permita um acesso fácil e rápido aos seus elementos. Já ao nível da memória importa garantir que a estrutura ocupe o mínimo espaço possível.

Tendo em vista o compromisso que existe entre a memória e o processamento, a escolha que pareceu ser mais apropriada foi a representação em *"Foward and Reverse Star Form"* (FRSF) [1] – ver secção A.1. Além disso, esta estrutura é especialmente indicada para o algoritmo MPS [10] que irá ser usado no algoritmo de IMSH referido na secção 4.2.

No anexo A é feita uma descrição detalhada da representação da rede e de algumas opções de implementação. A representação em FRSF da rede teve de ser adaptada para permitir a transformação da rede (sumariamente descrita na secção seguinte, e em detalhe na secção A.3 em anexo), de forma eficiente. Note-se que a transformação da rede descrita em A.3 é utilizada por todos os algoritmos implementados.

## 3.3 Transformação da rede

Todos os algoritmos implementados para o cálculo de caminhos disjuntos nos nós, de custo aditivo mínimo, requerem uma transformação da rede que é descrita nesta secção.

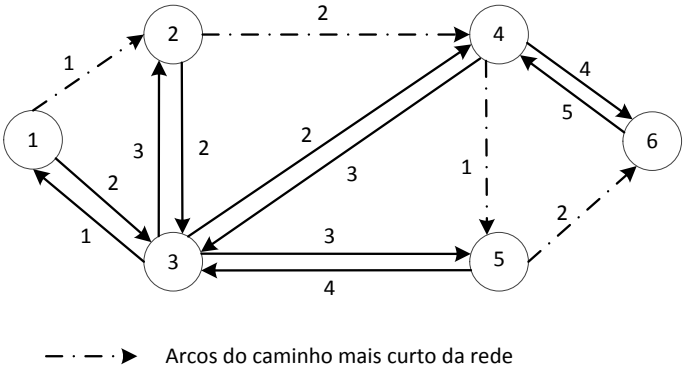



Figura 3.1: Rede com o caminho mais curto do nó 1 para o nó 6 ($\langle 1,2,4,5,6\rangle$), de custo 6.

A figura 3.1 representa a topologia de uma rede dirigida $G = (V,A)$, com $V = \{1,2,3,4,5,6\}$ e $A = \{(1,2),(1,3),(2,3),(2,4),(3,1),(3,2),(3,4),(3,5),(4,3),(4,5),(4,6),(5,4),(5,6),(6,4)\}$, em que os arcos a tracejado indicam o caminho do nó 1 para o nó 6. Esta rede é usada para exemplificar a transformação apresentada nesta secção.

Dado um caminho mais curto numa rede (ou um conjunto de caminhos mais curtos de custo aditivo mínimo), cada nó intermédio[1] desse caminho (ou desse conjunto de caminhos) é divido em dois nós (*Vertex-Splitting*) [2, pág. 79] – tal como é exemplificado na figura 3.2.

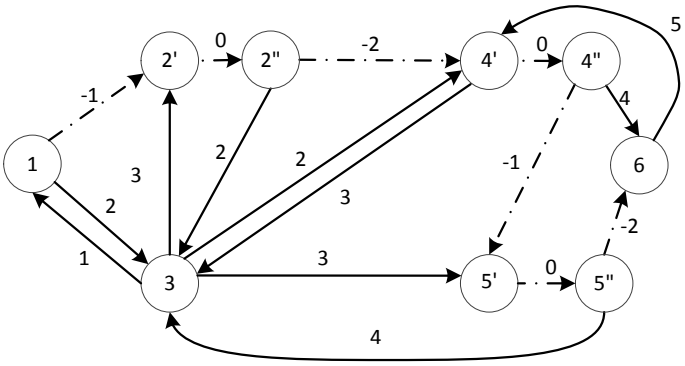



Figura 3.2: Rede da figura 3.1 com os nós divididos.

Cada nó $i$ (nó intermédio do caminho) dividido é agora representado por um *nó de entrada* $i'$ e um *nó de saída* $i''$, ligados por um arco dirigido, $(i',i'')$, de custo nulo. Todos os arcos anteriormente incidentes no nó original passam a ser incidentes no *nó de entrada* $i'$ , e todos os nós emergentes do nó original passam agora a sair do *nó de saída* $i''$.

Segue-se a inversão do sentido de todos os arcos do caminho mais curto determinado (ou dos arcos do conjunto de caminhos mais curtos de custo aditivo mínimo) – o que

[1] Um nó intermédio de um caminho é qualquer nó desse caminho, exceto o nó origem e o nó destino.

implica que o seu custo passa a ser o simétrico do que era antes da inversão – e a eliminação dos arcos paralelos aos agora criados, caso existam. Aplicada esta transformação ao grafo, obtém-se uma rede como a exemplificada na figura 3.3.

Como já foi anteriormente referido, a figura 3.2 ilustra a implementação do *"Vertex-Splitting"* na rede da figura 3.1, depois de calculado o caminho $\langle 1,2,4,5,6 \rangle$. Na figura 3.3 pode observar-se a inversão dos arcos do caminho mais curto (na rede com os nós já divididos).

Devido às restrições de memória da plataforma onde se pretende utilizar as rotinas implementadas, esta transformação da rede não será feita sobre uma cópia da rede original, mas sim modificando a rede existente e repondo-a no seu estado inicial, depois de cessar a utilidade da rede transformada. Para tirar o máximo partido possível da estrutura da rede utilizada (*hybrid foward and reverse star form*) arbitrou-se que após ser aplicado o *"Vertex-Splitting"* os nós $i''$ *coincidiram* com os nós $i$ antes de ser efetuada a referida transformação da rede. Esta decisão encontra-se devidamente explicada na secção A.2.

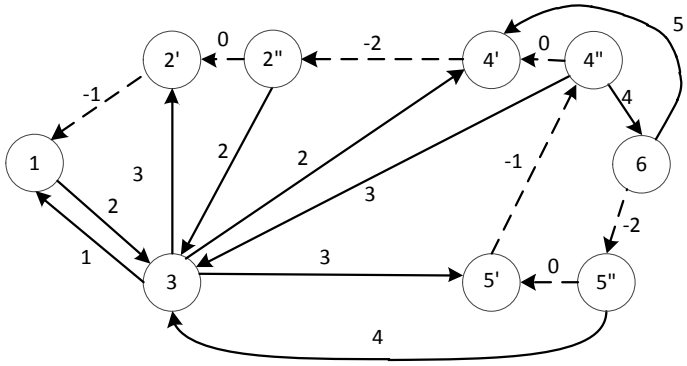



Figura 3.3: Rede da figura 3.1 com os nós divididos e com os arcos do caminho $\langle 1,2,4,5,6 \rangle$ invertidos, e com custo simétrico.

## 3.4 Algoritmos de cálculo de um par de caminhos disjuntos nos nós com custo aditivo mínimo

### 3.4.1 Algoritmo de Bhandari

O algoritmo de Bhandari [2] permite obter um par de caminhos disjuntos nos nós de custo aditivo mínimo. Para tal, o primeiro passo é determinar o caminho mais curto entre dois nós numa rede utilizando o algoritmo de Dijkstra (ou o algoritmo *Breath First Search*–BFS) – revistos em anexo nas secções B.1 e B.3 respetivamente. Seguidamente, e utilizando esse caminho como semente, aplica-se à rede a transformação descrita na

**Algoritmo Bhandari:** Algoritmo de Bhandari que permite calcular um par de caminhos disjuntos nos nós com custo aditivo mínimo [2].

**Dados**: Grafo da rede dirigida $G = (V, A)$; nó origem $s$; nó destino $t$; os custos associados a cada arco $l(i, j)$, $(i, j) \in A$.

**Resultado**: O par de caminhos disjuntos nos nós com custo aditivo mínimo, com origem no nó $s$ e destino no nó $t$, $(p^*, q^*)$.

```
1  Calcular do caminho mais curto de s para t, p1
2  Se p1 não existe então
3  |  Termina                                          // Não existe solução
4  fim
5  Transformar o grafo (ver sub-secção 3.3) com base no caminho mais curto de s
   para t, p1   // Esta transformação pode introduzir arcos de custo negativo
6  Cálculo, na rede modificada, de um caminho mais curto de s para t, utilizando o
   algoritmo de Dijkstra Modificado ou o algoritmo BFS. Seja este caminho p2
7  Se p2 existe então
   |  // Remove os arcos comuns que possam existir entre os dois
   |     caminhos (ver sub-secção 3.4.3 e secção C.1)
8  |  (p*, q*) ← Remove_Interlacing (p1, p2)            // Há solução
9  fim
10 Caso contrário
11 |  (p*, q*) ← (∅, ∅)                                 // Não existe solução
12 fim
```

secção 3.3. Cada arco do caminho mais curto invertido, ou seja, do nó origem para o nó destino, tem custo simétrico do arco que lhe deu origem.

Na rede transformada aplica-se o algoritmo de Dijkstra modificado (ver em anexo B.2) ou o algoritmo BFS (devido à introdução de arcos com custo negativo) para encontrar o caminho mais curto nesta rede. Finalmente, caso esse caminho exista, devem remover-se os arcos em comum (os arcos com os mesmos nós extremos) entre esse caminho e o caminho mais curto obtido na rede original (o caminho semente), para obter um par de caminhos disjuntos nos nós de custo aditivo mínimo.

No algoritmo de Bhandari a função (*Remove_interlacing*), que remove os arcos comuns de um par de caminhos disjuntos é descrita de forma genérica na secção 3.4.3.

### 3.4.2 Algoritmo de Suurballe

O algoritmo de Suurballe [2] também calcula um par de caminhos disjuntos nos nós com custo aditivo mínimo. O princípio é o mesmo que o que é usado no algoritmo de Bhandari, mas fazendo uma transformação adicional à rede, ou seja, criando um grafo $G'$ a partir do grafo original, os quais diferem nos custos. Para tal, é necessário redefinir os

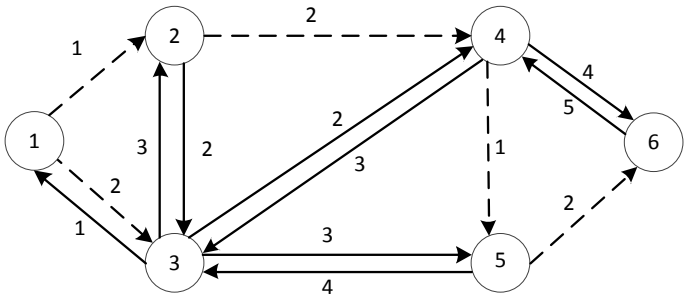



Figura 3.4: Árvore de caminhos mais curtos com raiz no nó A (linhas a tracejado).

custos de cada arco da seguinte forma:

$$l'(i,j) = d(i) + l(i,j) - d(j) \tag{3.6}$$

onde, $d(i)$ é a distância mínima do nó de origem $s$ ao nó $i$ da rede $G$. Os custos $l'(i,j)$ definidos na equação (3.6) são geralmente designados por custos reduzidos.

A figura 3.4 ilustra a transformação anteriormente descrita para a rede dirigida da figura 3.1. As linhas a tracejado representam os arcos que pertencem à árvore de caminhos mais curtos com raiz no nó *1*. Se um arco $(i,j)$ $(i,j \in V)$ pertence à árvore de caminhos mais curtos o seu custo reduzido é nulo. Por esse motivo, a transformação da rede não introduz arcos de custo negativo.

O algoritmo de Suurballe está ilustrado nas figuras 3.5 a 3.7 e consiste em seguir os passos apresentados no algoritmo Suurballe.

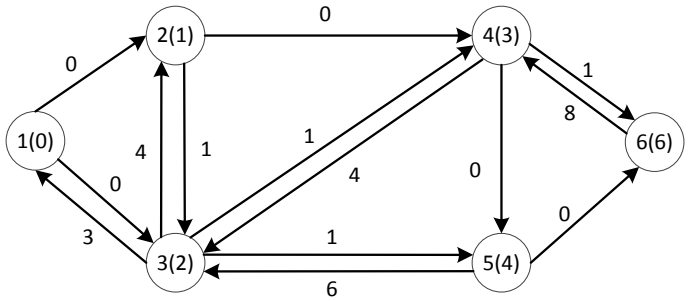



Figura 3.5: Transformação $G \rightarrow G'$ depois de calculados os custos reduzidos para todos os arcos da rede. Os números entre parênteses indicam a distância mínima ao nó 1 (nó de origem). De notar que os arcos na árvore de caminhos mais curtos têm custo nulo.

Na figura 3.5 encontra-se a representação equivalente à da rede dirigida da figura 3.4, em que cada arco tem associado o respetivo custo reduzido; na figura 3.6 é ilustrada a divisão dos nós do caminho mais curto entre 1 e 6, seguindo-se na figura 3.7 a inversão dos arcos que pertencem ao caminho mais curto de 1 para 6 e a eliminação dos arcos paralelos aos arcos invertidos.

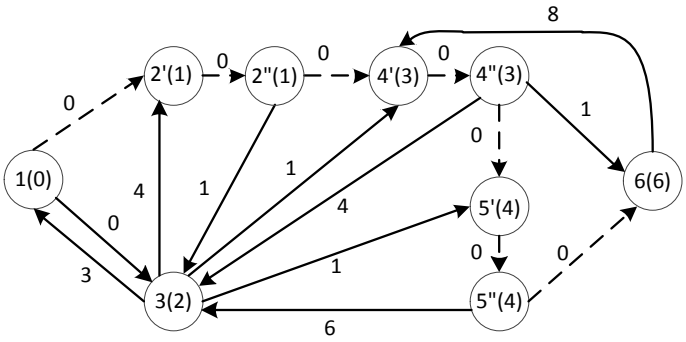



Figura 3.6: Aplicação do *Vertex-Splitting* à rede.

### 3.4.3 Método de remoção dos arcos comuns a um par de caminhos

Nos algoritmos de Bhandari e de Suurballe é utilizada a função "Remove_Interlacing" para remover os arcos comuns que existem entre o par de caminhos inicialmente determinado. Essa função consiste em remover os segmentos que sejam comuns a ambos os caminhos, construindo depois um novo par de caminhos usando os segmentos restantes.

Uma forma de implementar a função "Remove_Interlacing" é apresentada no algoritmo Remove-Interlacing, na secção C.1 em anexo.

### 3.4.4 Observações relativamente aos algoritmos de cálculo de um par de caminhos disjuntos nos nós com custo aditivo mínimo

No algoritmo de Bhandari, ao ser aplicada a transformação referida na sub-secção 3.3 passam a existir arcos de custo negativo, sem contudo surgirem ciclos negativos na rede. O algoritmo de Dijkstra não consegue lidar com este tipo de problema [2]. Assim sendo, para o caso do algoritmo de Bhandari, no cálculo do segundo caminho na rede têm que ser usados os algoritmos de cálculo de caminho mais curto de Dijkstra modificado ou BFS.

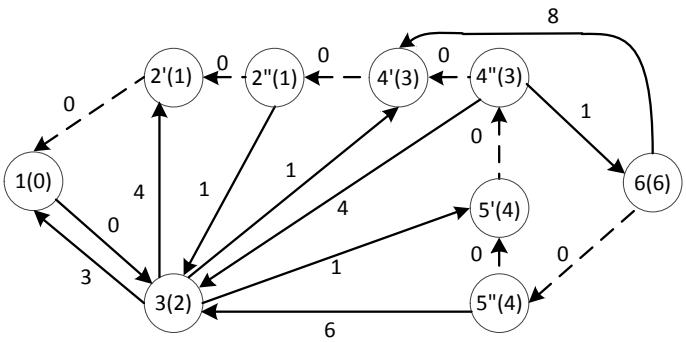



Figura 3.7: Inversão do sentido dos arcos do caminho mais curto ($\langle 1,2,4,5,6 \rangle$) e eliminação dos arcos paralelos aos arcos invertidos.

**Algoritmo Suurballe:** Algoritmo de Suurballe que permite calcular um par de caminhos disjuntos nos nós com custo aditivo mínimo, retirado de [2].

**Dados**: Grafo da rede dirigida $G = (V,A)$; nó origem $s$; nó destino $t$; os custos associados a cada arco $l(i,j)$, $(i,j) \in A$.

**Resultado**: O par de caminhos disjuntos nos nós com custo aditivo mínimo, com origem no nó $s$ e destino no nó $t$, $(p^*, q^*)$.

1 Calcular a árvore de caminhos mais curtos com raiz em $s$ (usando o algoritmo de Dijkstra – ver anexo B.1), definindo as distâncias $d(i), i \in V$
2 Seja $p_1$ o caminho mais curto de $s$ para $t$ (contido na árvore de caminhos mais curtos)
3 **Se** $p_1$ *não existe* **então**
4 | Termina `// Não existe solução`
5 **fim**
6 Calcular os custo reduzidos $l'(ij) = d(i) + l(i,j) - d(j), \forall (i,j) \in A$, definindo o grafo $G'$
`// Transformação que não introduz arcos de custo negativo`
7 Transformar o grafo $G'$ (ver sub-secção 3.3 com com base no caminho $p_1$)
8 Determina o caminho mais curto na rede modificada, $p_2$, usando o algoritmo de Dijkstra
9 **Se** $p_2$ *existe* **então**
| `// Remove os arcos comuns que possam existir entre os dois caminhos (ver sub-secção 3.4.3 e secção C.1)`
10 | $(p^*, q^*) \leftarrow$ `Remove_Interlacing` $(p_1, p_2)$ `// Há solução`
11 **fim**
12 **Caso contrário**
13 | $(p^*, q^*) \leftarrow (\emptyset, \emptyset)$ `// Não existe solução`
14 **fim**

No algoritmo de Suurballe, como são usados os custos reduzidos, todos os arcos que pertençam ao caminho mais curto entre dois nós da rede têm custo reduzido nulo. Assim sendo, quando a transformação é aplicada à rede não surgem arcos com custos negativos. A consequência disto é que no cálculo do segundo caminho mais curto pode ser usado o algoritmo de cálculo de caminho mais curto de Dijkstra.

É sabido que o algoritmo de Dijkstra envolve menos operações que os algoritmos de Dijkstra modificado ou BFS. Isto poderá levar à conclusão de que o algoritmo de Suurballe tem um desempenho melhor que o algoritmo de Bhandari. Tal não é verdade, pois o algoritmo de Suurballe envolve o cálculo dos custos reduzidos. Na prática os resultados obtidos (ver tabela 5.1 no capítulo 5) foram muito semelhantes. Para se testar a eficiência computacional de cada um dos algoritmos, foram implementadas as seguintes versões:

1. Algoritmo de Suurballe (que usa o algoritmo de Dijkstra nos dois casos);

2. Algoritmo de Bhandari que usa o algoritmo de Dijkstra no primeiro cálculo de caminho mais curto e o algoritmo de Dijkstra modificado no segundo cálculo de caminho mais curto;

3. Algoritmo de Bhandari, que usa o algoritmo de Dijkstra no primeiro cálculo de caminho mais curto, e o algoritmo BFS no segundo cálculo de caminho mais curto;

4. Algoritmo de Bhandari que usa o algoritmo BFS nos dois casos.

## 3.5 Algoritmo de cálculo de $K$ ($K \geq 2$) caminhos disjuntos nos nós com custo aditivo mínimo

### 3.5.1 Algorimo $K$-Bhandari

Na secção anterior foram apresentados algoritmos para cálculo de um par ($K = 2$) de caminhos com custo aditivo mínimo disjuntos nos nós. Nesta secção é descrito o algoritmo que permite o cálculo de um conjunto de $K$ ($K \geq 2$) caminhos disjuntos nos nós com custo aditivo mínimo. De modo semelhante ao caso de $K = 2$ (cálculo de um par de caminhos), os $K$ caminhos são calculados através de $K$ iterações em que se aplica um algoritmo de cálculo de caminho mais curto apropriado, depois de ser aplicada a devida transformação à rede.

A transformação da rede é efetuada seguindo o modo descrito na secção 3.3, utilizando como semente os $K-1$ caminhos disjuntos nos nós, de custo aditivo mínimo (já determinados).

Esta transformação está ilustrada nas figuras 3.8 e 3.9, onde o trio de caminhos disjuntos nos nós com custo aditivo mínimo é obtido à custa do par de caminhos disjuntos nos nós com custo aditivo mínimo já conhecido ($\langle 1,2,4,8\rangle$, $\langle 1,3,5,8\rangle$). Os nós (exceto os últimos nós de cada caminho) que pertencem ao par de caminhos semente, são ligados aos vértices que não pertencem a esse par de caminhos, de acordo com as regras usadas no algoritmo de cálculo de um par de caminhos disjuntos nos nós com custo aditivo mínimo (secção 3.3). Assim, o nó $i''$ de um dos caminhos liga ao nó $j'$ do outro caminho – por exemplo: $(2'',3')$ e $(3'',2')$. Para determinar um terceiro caminho do nó de origem ao nó de destino basta aplicar um algoritmo de cálculo de caminho mais curto apropriado (Dijkstra modificado ou BFS), coalescer os nós que foram divididos (transformar o nós *i"* e *i'* novamente num nó *i*) e, por fim remover os arcos comuns (arcos de "interlacing").

Este algoritmo será designado por algoritmo de $K$-Bhandari, o qual é uma variante do algoritmo proposto por Suurballe [12].

**Algoritmo $K$-Bhandari:** Algoritmo $K$-Bhandari que permite calcular um conjunto de $k$ caminhos disjuntos nos nós com custo aditivo mínimo [2].

**Dados**: Grafo da rede dirigida $G = (V,A)$; nó origem $s$; nó destino $t$; os custos associados a cada arco $l(i,j)$, $(i,j) \in A$; o número de caminhos disjuntos nos nós $k$ que se pretende obter (caso seja possível).

**Resultado**: O conjunto de $K$ caminhos disjuntos nos nós com custo aditivo mínimo, com origem no nó $s$ e destino no nó $t$.

$S \leftarrow \emptyset$ `// Conjunto de caminhos disjuntos obtidos até ao momento`
$j \leftarrow 0$ `// Número de caminhos mais curtos (semente) obtidos`
Calcular o primeiro caminho mais curto de $s$ para $t$ em $G$, seja este $p_1$, usando o algoritmo de Dijkstra
**Se** $p_1$ *não existe* **então**
    Termina `// Não existe solução`
**fim**
$j \leftarrow j+1$ `// Conjunto` $S$ `formado por` $p_1$
$S \leftarrow \{p_1\}$
`// Tentativa de aumentar o número de caminhos contidos em` $S$
**Enquanto** $|S| \leq K$ **faz**
    Transformar o grafo com base nos caminhos contidos em $S$ (ver sub-secção 3.3)
    Calcular o caminho mais curto entre $s$ e $t$, $p_{j+1}$, na rede transformada, usando o algoritmo de Dijkstra Modificado ou o algoritmo BFS
    **Se** $p_{j+1}$ *existe* **então**
        $j \leftarrow j+1$
        $S \leftarrow K$-`Remove_Interlacing` $(S,p_j)$ `//` $S$ `passa a conter` $j$ `caminhos`
    **fim**
    **Caso contrário**
        Termina
    **fim**
**fim**

No final, o conjunto $S$ contém os caminhos disjuntos nos nós desejados, caso existam. Se $|S| = 0$ não existe nenhum caminho entre os nós $s$ e $t$. Se $|S| = 1$ não existem dois caminhos disjuntos nos nós entre $s$ e $t$. Se $|S| \in [2, K-1]$ existem no máximo $|S|$ caminhos na rede entre $s$ e $t$. Se $|S| = K$ existem pelo menos $K$ caminhos disjuntos nos nós entre $s$ e $t$.

No exemplo apresentado na figura 3.9, os três caminhos resultantes são $(\langle 1,2,7,8\rangle, \langle 1,3,4,8\rangle, \langle 1,6,5,8\rangle)$.

### 3.5.2 Método de remoção dos arcos comuns aos $K$ ($\geq$2) caminhos

Para remover os arcos comuns entre o caminho $p_k$ os $K-1$ caminhos disjuntos nos nós de custo mínimo, anteriormente determinados (conjunto $S$), é criado um conjunto

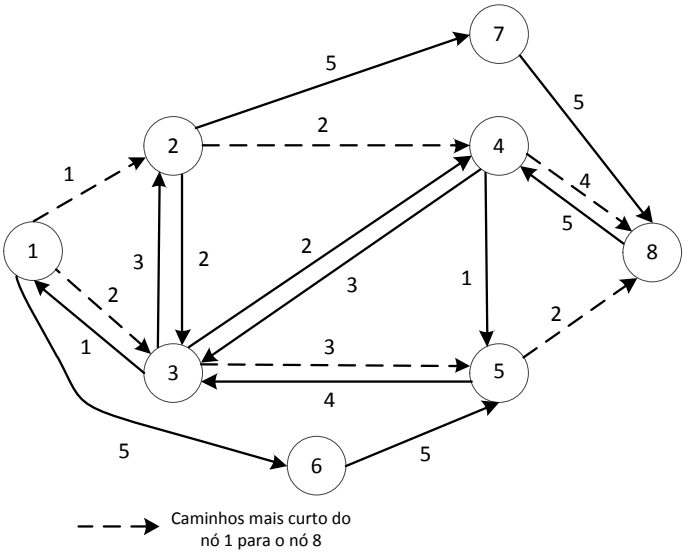



Figura 3.8: Rede usada para exemplificar a transformação da rede, em que ($\langle 1,2,4,8\rangle$, $\langle 1,3,5,8\rangle$) é o par de caminhos disjuntos nos nós com custo aditivo mínimo.

($D_S$) com todos os arcos desse(s) ($K-1$) caminho(s).

De seguida, verifica-se se o simétrico de cada um dos arcos do $K$-ésimo caminho existe no conjunto de arcos criados, $D_S$. Caso exista, é necessário remover esse arco do conjunto $D_S$. Caso contrário, esse arco deve ser inserido no conjunto de arcos correntes, $D_S$.

Note-se que o primeiro e o último arco do $K$-ésimo caminho podem ser automaticamente inseridos. Isto deve-se ao facto de serem necessariamente diferentes dos primeiros e últimos arcos dos ($K-1$) caminhos, dado que estes foram invertidos antes de calcular este caminho.

No caso de não ter havido nenhuma remoção de arcos do conjunto dos arcos que constituem os ($K-1$) caminhos, não existem arcos comuns entre os caminhos. Neste caso, já são conhecidos os $K$ caminhos disjuntos nos nós com custo aditivo mínimo.

Caso existam arcos comuns entre os caminhos, é necessário construir caminhos com os arcos existentes no conjunto dos arcos obtidos após a remoção/inserção de arcos do $K$-ésimo caminho. Para tal, basta remover um dos $K$ arcos com cauda no nó de origem e procurar o arco que esteja no conjunto $D_S$ e que tenha cauda igual ao nó cabeça do arco anterior no caminho a construir, removendo-o. Este processo é repetido até se chegar ao nó de destino, obtendo-se assim um caminho. Todo este processo é repetido $K$ vezes.

A função *K_Remove_interlacing*($S, p$) é apresentada sob a forma de pseudo-código na secção C.2 em anexo.

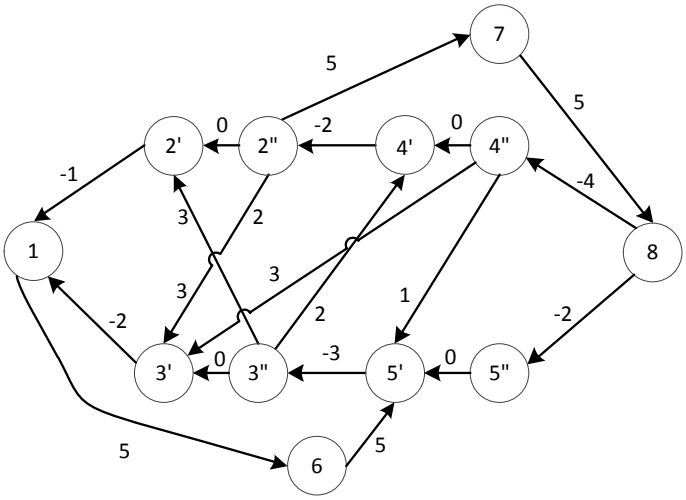



Figura 3.9: Transformação da rede, incluindo o *Vertex-splitting* aplicado à rede da figura 10.

### 3.5.3 Observação relativamente ao algoritmos de cálculo de um conjunto de $K$ caminhos disjuntos nos nós com custo aditivo mínimo

Embora o algoritmo proposto nesta secção permita obter um par de caminhos disjuntos de custo aditivo mínimo, quando for desejado apenas um par, é preferível utilizar os algoritmos propostos em 3.4, pois são mais eficientes. Isso resulta da maior eficiência do algoritmo Remove-Interlacing de remoção de arcos comuns face ao algoritmo $K$-Remove-Interlacing (para $K = 2$). No capítulo 5 são apresentados resultados que sustentam esta observação.

# Capítulo 4

# Algoritmos implementados considerando SRLG

## 4.1 Introdução

Na secção 1.1 foram referidos os aspetos que levaram ao aparecimento do conceito de SRLG no contexto de proteção em redes óticas. Neste capítulo são apresentados algoritmos de cálculo de um par de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo, que podem ser usados para garantir a proteção de um $AP$ (um dos elementos do par determinado).

Considere uma rede, cuja topologia é representada pelo grafo, $G = (V,A)$, sendo $l(i,j)$ e $R(i,j)$, o custo e os SRLG, respetivamente, associados a cada arco $(i,j) \in A$. O problema de cálculo de um par de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo, entre um dado par de nós, origem e destino ($s,t \in V$), pode ser formalizado como se segue:

$$(p^*,q^*) = \arg \min_{(p,q)\in\mathscr{P}_{st}} C_p + C_q \tag{4.1}$$

$$\text{s. a:} \quad V_p \cap V_q = \{s,t\} \tag{4.2}$$

$$R_p \cap R_q = \emptyset \tag{4.3}$$

Hu [7] mostrou que o problema de calcular um par de caminhos disjuntos nos SRLG (assumindo que cada arco pertence a pelo menos um SRLG) é NP-completo e propôs uma formulação de programação linear inteira para a sua resolução. Essa formulação pode ser facilmente estendida para obter pares de caminhos disjuntos nos nós.

Este capítulo descreve os algoritmos "Iterative Modified Suurballe's Heuristic" (IMSH) [13] e "Confliting SRLG Exclusion – Min Sum" (CoSE-MS) [4], nas secções 4.2 e 4.3, respetivamente. O algoritmo IMSH utiliza a heurística "Modified Suurballe's Heuristic"

(MSH), a qual é descrita na sub-secção 4.2.1. Por sua vez o algoritmo CoSE-MS além de utilizar a heurística MSH, utiliza também a heurística "Modified Bhandari's Heuristic" (MBH), que é descrita na sub-secção 4.3.2; o CoSE-MS precisa ainda de calcular um conjunto de SRLG em conflito ("Confliting SRLG Set"), o que é feito recorrendo ao algoritmo SRLG Exclusion, revisto na sub-secção 4.3.4. De notar que na secção 4.3 são apresentadas três versões do algoritmo CoSE–MS, a original proposta em [4] e duas novas versões que diferem da primeira na forma como é obtido o conjunto de SRLG em conflito.

## 4.2 Algoritmo "Iterative Modified Suurballe's Heuristic" (IMSH)

O *"Iterative Modified Suurballe's Heuristic" (IMSH)* [13] é uma heurística iterativa. Em cada iteração é necessário fornecer um caminho do nó origem para o nó destino, por ordem crescente do seu respetivo custo.

Assim sendo, é necessário utilizar um algoritmo de enumeração de caminhos. Em [13] é sugerido que se use o algoritmo de Yen. Um outro algoritmo de enumeração de caminhos é o algoritmo MPS [10]. Dado que este algoritmo permite obter um desempenho melhor no que toca ao tempo de processamento [11], foi o algoritmo de enumeração de caminhos utilizado[1].

### 4.2.1 "Modified Suurballe's Heuristic" (MSH)

Tal como o nome indica o algoritmo *IMSH* é uma versão iterativa da *"Modified Suurballe's Heuristic" (MSH)* [13]. Esta heurística recebe a topologia de uma rede, $G(V,A)$, o custo $l(i,j)$, e os SRLG, $R(i,j)$, associados a cada arco $(i,j) \in A$. Depois, fornecendo um nó de origem $s$, um nó de destino $t$ e um caminho semente $p$ (entre esse par de nós), procura obter um par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo dado o caminho semente utilizado. Caso esse par não exista, devolve um par vazio com custo infinito. No algoritmo MSH é apresentado a heurística *MSH* em pseudo-código.

Os primeiros sete passos do algoritmo MSH consistem em aplicar à rede uma transformação (semelhante à que foi descrita em 3.3) e adicionar $M$ (soma do custo de todos os arcos da rede) aos arcos que estejam em conflito com os SRLG do caminho fornecido

[1] O código utilizado para o MPS foi disponibilizado pela Professora Doutora Teresa Gomes, e apenas pontualmente alterado pelo autor desta dissertação.

**Algoritmo MSH:** Algoritmo MSH que permite obter um par de caminhos disjuntos nos nós e nos SRLG [13].

**Dados**: Grafo da rede dirigida $G = (V,A)$; nó origem $s$; nó destino $t$; o custo $l(i,j)$, e os SRLG, $R(i,j)$, associados a cada arco $(i,j) \in A$; caminho semente $p_c$.
**Resultado**: Par de caminhos disjuntos nos nós e nos SRLG $(p,q)$, obtido usando o caminho semente $p_c$.

1 Remover temporariamente os arcos $(i,j)$ da rede $G, \forall (i,j) \in p_c$
2 $M \leftarrow \sum_{(i,j)\in A} l(i,j)$
3 **Para cada** *arco* $(i,j)$ *da rede tal que* $R(i,j) \cap R_{p_c} \neq \emptyset$ **faz**
  `// O arco (i,j) pertence a pelo menos um dos SRLG que afetam o caminho` $p_c$
4  $l(i,j) \leftarrow l(i,j) + M$
5 **fim**
6 Aplicar à rede uma transformação semelhante à referida em 3.3 com base no caminho $p_c$, em que os arcos simétricos (aos do caminho $p_c$) introduzidos têm custo nulo.
  `// Dado que a rede não tem arcos com custo negativo, pode ser usado o algoritmo de Dijkstra - ver anexo B.1`
7 $p' =$`Dijkstra` $(s,t)$
8 **Se** *existe* $p'$ **então**
9  $(p,q) \leftarrow$`Remove_Interlacing` $(p_c,p')$
10  **Se** $R_p \cap R_q \neq \emptyset$  `// Avalia se (p,q) são disjuntos nos SRLG`
11  **então**
12  $(p,q) \leftarrow (\emptyset,\emptyset)$  `// Não há solução`
13  **fim**
14 **fim**
15 **Caso contrário**
16  $(p,q) \leftarrow (\emptyset,\emptyset)$  `// Não há solução`
17 **fim**

$p$, penalizando a sua utilização. A única diferença da transformação aplicada para a transformação descrita em 3.3 é que após a inversão dos arcos do caminho mais curto, o custo desses mesmos arcos deixa de ser simétrico e passa a ser nulo. Desta forma é criada uma rede modificada $G'$, onde se tentará encontrar um caminho disjunto nos nós e nos SRLG do caminho $p$.

Convém relembrar que $G'$ não tem arcos de custo negativo, o que permite que se use o algoritmo de Dijkstra para o cálculo de um caminho mais curto entre dois nós da rede. Assim, o passo 7 consiste em usar o algoritmo de Dijkstra para procurar um caminho mais curto na rede transformada $G'$.

Caso seja encontrado algum caminho $p'$, avança-se para o passo 9, onde se remove o "interlacing", ou seja, removem-se os arcos comuns, após o coalescer dos nós (caso

seja necessário) e/ou a substituição dos nós $i'$ e $i''$ pelo nó $i$ original, obtendo um par de caminhos disjuntos nos nós. Caso contrário, tem que se avançar para o passo 16, onde se indica que não há solução para os nós de origem $s$ e de destino $t$, considerando o caminho semente $p$.

No passo 10 é feita a verificação da disjunção, relativamente aos SRLG, dos caminhos $p$ e $q$. Caso o par de caminhos seja disjuntos nos SRLG, então esse par é a solução.

Tal como é dito em [13], a complexidade do MSH é $\mathcal{O}(m+r+n\log n)$, onde $n$ é o número de nós da rede, $m$ é o número de arcos da rede e $r$ é o número de SRLG na rede.

O custo do par de caminhos disjuntos nos nós e nos SRLG obtido pelo MSH é sempre menor ou igual ao custo do par de caminhos disjuntos nos nós e nos SRLG obtido pelo "Two-Step Approach" (TSA) [6]. Para se provar esta afirmação basta considerar duas redes, $G'$ e $G''$, que são as redes transformadas para o MSH e para o TSA, respetivamente. Sejam $p'$ e $p''$ os caminhos calculados de $s$ para $t$ nas redes transformadas para o MSH e para o TSA, respetivamente. No caso da rede transformada para o MSH, todos os arcos que constituem o caminho semente $p$ foram removidos e todos os arcos simétricos dos que constituem o caminho semente $p$ têm custo nulo. Já na rede transformada para o TSA, tanto os arcos que constituem o caminho semente $p$, como os arcos simétricos são removidos. Além disso, ainda são removidos da rede os arcos que tenham cauda ou cabeça em nós do caminho mais curto. Como tal podem existir dois casos:

**Caso 1** O caminho determinado, $p'$, não tem nenhum arco que seja simétrico dos arcos do caminho semente $p$. Nesse caso, é fácil de perceber que o caminho $p'$ é igual ao caminho $p''$. Como tal, ambos os caminhos têm o mesmo custo, que é igual a $Custo_{MSH} = Custo_{TSA} = C_p + Custo_{p'} = C_p + C_{p''}$.

**Caso 2** O caminho determinado, $p'$, tem arcos que são simétricos aos arcos que constituem o caminho semente $p$. Nesse caso o caminho $p'$ é diferente do caminho $p''$. Como os arcos simétricos aos arcos do caminho semente $p'$ têm custo nulo, temos que $C_{p'} \leq C_{p''}$. Logo $C_p + C_{p'} \leq C_p + C_{p''} \Rightarrow Custo_{MSH} \leq Custo_{TSA}$, como se queria provar.

### 4.2.2 "Iterative Modified Suurballe's Heuristic" (IMSH)

O algoritmo IMSH recorre iterativamente ao algoritmo MSH, usando um caminho semente diferente a cada iteração. O $i$-ésimo caminho semente é gerado usando o algoritmo MPS. Esta heurística tenta determinar, a cada iteração, um par de caminhos disjuntos nos nós e nos SRLG, usando o $i$-ésimo caminho gerado pelo MPS como caminho semente. Caso não se limite o número de iterações, eventualmente, todos os caminhos entre um par

**Algoritmo IMSH:** Algoritmo IMSH que permite obter um par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo, adaptado de [13].

**Dados**: Grafo da rede dirigida $G = (V, A)$; nó origem $s$; nó destino $t$; o custo $l(i, j)$, e os SRLG, $R(i, j)$, associados a cada arco $(i, j) \in A$.
**Resultado**: O par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo $(p, q)$

```
(p,q) ← (∅,∅)        // Melhor par encontrado até ao momento, C_(∅,∅) = ∞
i ← 0
otimo_nao_encontrado ← true
Enquanto i ≤ MAX_ITERACOES ∧ otimo_nao_encontrado faz
    i ← i + 1
    // Obtém um caminho semente a cada iteração usando o
       algoritmo de MPS
    p_i ←MPS (s,t)                          // O i-ésimo caminho mais curto
    (p'_i, p''_i) ←MSH (G,s,t,l,R,p_i)
    Se C_(p'_i,p''_i) < C_(p,q) então
        (p,q) ← (p'_i, p''_i)                          // Atualiza melhor par
    fim
    Se C_(p,q) − C_p1 ≤ C_pi          // p_1 é o caminho mais curto da rede G
    então
        otimo_nao_encontrado ← false // (p,q) é o par ótimo
    fim
fim
```

de nós (o nó origem $s$ e o nó destino $t$) são explorados. Nesse caso, é possível obter uma solução ótima, caso ela exista. Como o número de caminhos entre um par de nós pode ser muito grande, o número de iterações também poderá ser muito alto, conduzindo a tempos de execução demasiado elevados. Como tal, o número máximo de iterações tem que ser limitado por um valor pré determinado.

O IMSH tem a particularidade de fornecer limite inferior para o custo do par obtido. No entanto, a condição de deteção da otimalidade utilizando $C_{(melhor\_par)} - C_{p_1} \leq C_{p_k}$ (onde $C_{(melhor\_par)}$ é o custo do melhor par obtido até ao momento, $C_{p_1}$ é o custo do caminho mais curto na rede do nó de origem para o nó de destino e $C_{p_k}$ é o custo do caminho semente na iteração $k$) é demasiado exigente. No capítulo 5 são apresentados os resultados que evidenciam este aspeto.

Na secção E.1, em anexo, pode ser encontrado um exemplo detalhado do funcionamento do algoritmo IMSH.

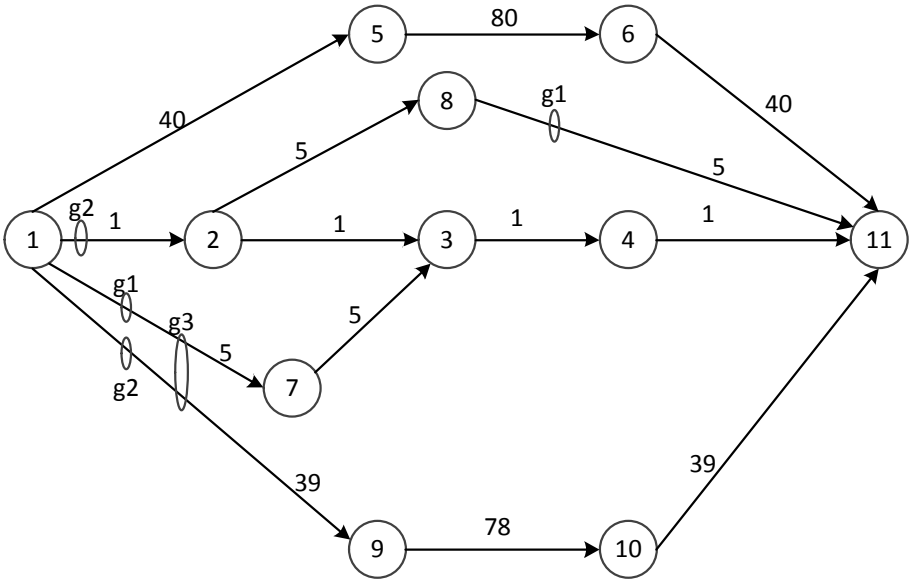



Figura 4.1: Rede usada para o contra-exemplo da condição de otimalidade proposta em [13].

### 4.2.3 Condição de otimalidade do IMSH

Em [13] é dito que o par de caminhos disjuntos nos nós e nos SRLG é ótimo (tem o menor custo aditivo possível) se se verificar a seguinte condição:

$$C_{(melhor\_par)} \leq 2 \times C_{p_i} \tag{4.4}$$

onde $C_{(melhor\_par)}$ é o custo do melhor par de caminhos obtido até ao momento e $C_{p_i}$ é o custo do caminho semente na iteração $i$.

Através do contra-exemplo que se apresenta é possível ver que isto não se verifica. Considere-se a figura 4.1. O IMSH vai realizar as seguintes iterações:

**Iteração 1**

$p_1 = \langle 1,2,3,4,11 \rangle$ ($C_{p_1} = 4$)

$q_1 = \langle 1,7,3',2'',8,11 \rangle$ ($C_{q_1} = 20$)

Coalescendo os nós e removendo o "interlacing" é obtido o seguinte par de caminhos:

$p'_1 = \langle 1,7,3,4,11 \rangle$ ($C_{p'_1} = 12$)

$p''_1 = \langle 1,2,8,11 \rangle$ ($C_{p''_1} = 11$)

Dado que os arcos $(1,7)$ e $(8,11)$ estão associados ao SRLG $g_1$, o par de caminhos encontrado nesta iteração não é disjunto nos SRLG. Assim sendo, o IMSH avança para a próxima iteração.

**Iteração 2**

$p_2 = p'_2 = \langle 1,2,8,11 \rangle$ ($C_{p'_2} = 11$)
$p''_2 = \langle 1,5,6,11 \rangle$ ($C_{p''_2} = 160$)
O par de caminhos encontrado nesta iteração é disjunto nos SRLG e é o melhor par de caminhos encontrado até ao momento, o seu custo é guardado. No entanto, $C_{(melhor\_par)} > 2 \times C_{p_2} \Leftrightarrow 171 > 2 \times 11$. Logo, ainda não foi encontrada a solução ótima, avançando-se para a iteração 3.

**Iteração 3**

$p_3 = p'_3 = \langle 1,7,3,4,11 \rangle$ ($C_{p'_3} = 12$)
$p''_3 = \langle 1,5,6,11 \rangle$ ($C_{p''_3} = 160$)
O par de caminhos encontrado nesta iteração é disjunto nos SRLG e o seu custo ($C_{(p'_3,p''_3)} = 172$) é maior que o custo do melhor par encontrado até ao momento. Assim sendo, o melhor par de caminhos não é atualizado nesta iteração. Como $C_{(melhor\_par)} > 2 \times C_{p_3} \Leftrightarrow 171 > 2 \times 12$, o IMSH continua a pesquisar um par de caminhos melhor.

**Iteração 4**

$p'_4 = \langle 1,9,10,11 \rangle$ ($C_{p'_4} = 156$)
$p''_4 = \langle 1,5,6,11 \rangle$ ($C_{p''_4} = 160$)
O par de caminhos encontrado nesta iteração é disjunto nos SRLG e o seu custo ($C_{(p'_4,p''_4)} = 316$) é maior que o custo do melhor par encontrado até ao momento. Como tal, o melhor par de caminhos não é atualizado nesta iteração.

Como $C_{(melhor\_par)} < 2 \times C_{p_4} \Leftrightarrow 171 < 2 \times 156$, o IMSH julga já ter encontrado o par ótimo e pára.

É fácil de ver que o par de caminhos $(p'_2, p''_2)$ não é a solução ótima para a rede da figura 4.1, dado que o par de caminhos $(\langle 1,5,6,11 \rangle, \langle 1,2,3,4,11 \rangle)$, com custo 164 também é disjunto nos nós e nos SRLG.

Em [14], a heurística *Iterative Modified Network-flow Heuristic* (IMNH) procura minimizar $C_p\omega + C_q$, garantindo que os caminhos $p$ e $q$ são disjuntos nos SRLG, com $\omega \geq 1$. A heurística IMNH é semelhante IMSH, exceto na função objetivo e na condição de otimalidade. Em [14] provam que a condição de otimalidade para o IMNH é:

$$C_{p_i} \geq \frac{C_{(melhor\_par)} - C_{p_1}}{\omega} \tag{4.5}$$

sendo $p_1$ o caminho mais curto na rede, $p_i$ o $i$-ésimo caminho mais curto utilizado como semente, e $C_{(melhor\_par)}$ o custo do melhor par encontrado até esse momento.

Tomando $\omega = 1$ a condição anterior pode escrever-se:

$$C_{p_i} \geq C_{(melhor\_par)} - C_{p_1} \tag{4.6}$$

e foi essa a condição utilizada no algoritmo IMSH, uma vez que se verificou que a condição proposta em [13] estava errada.

### 4.2.4 Melhoramento da transformação da rede para acelarar a obtenção da solução

O algoritmo de Dijkstra usa uma árvore binária com o objetivo de tornar a pesquisa do nó de custo mínimo mais rápida. Acontece que se existirem dois caminhos entre um nó de origem e um nó de destino com o mesmo custo, o algoritmo de Dijkstra irá devolver um desses dois caminhos, dependendo da forma como a árvore binária está organizada.

Sempre que na rede transformada existem soluções ótimas alternativas para o problema do caminho mais curto, o caminho $p'$ determinado pela heurística MSH no passo 7 irá ser um qualquer desses ótimos. Se um desses ótimos alternativos utiliza arcos simétricos do caminho semente, $p_c$, irá resultar num par de caminhos que pode não ser disjunto nos SRLG (pois o par final é formado por troços de $p'$ e $p_c$). Pelo contrário, se um outro ótimo alternativo não utilizar qualquer arco simétrico do caminho semente $p_c$ será garantido que o par resultante é disjunto nos SRLG. Contudo, numa rede com custos estritamente positivos, este segundo par de caminhos terá sempre custo superior ao anterior.

Uma vez que do ponto de visto do funcionamento do MSH o caminho $p'$ pode ser qualquer uma das soluções ótimas, para o problema do caminho mais curto na rede transformada, optou-se por dar preferência ao caminho que caso exista, resultará num par de caminhos disjuntos nos SRLG, desde que tenha custo inferior a $M$. Assim, o problema que se põe é o de evitar que no cálculo do segundo caminho $p'$, o algoritmo de Dijkstra devolva um caminho que passe pelos arcos simétricos do primeiro caminho $p_c$ (caminho $p_i$ no IMSH), caso exista um outro caminho com o mesmo custo que não use qualquer desses arcos simétricos.

Considere-se o exemplo na figura 4.2, em que o primeiro caminho encontrado foi $p = \langle 1,2,3,9\rangle$. Após a transformação, como se pode ver na figura 4.3, há dois caminhos possíveis de 1 para 9, com o mesmo custo. São eles $q_1 = \langle 1,4,3',2'',5,9\rangle$ e $q_2 = \langle 1,4,6,9\rangle$. Se se optar por $q_1$ obtemos o par de caminhos ($p' = \langle 1,4,3,9\rangle$ , $q' = \langle 1,4,5,9\rangle$) que tem custo 63, mas que não são disjuntos nos SRLG.

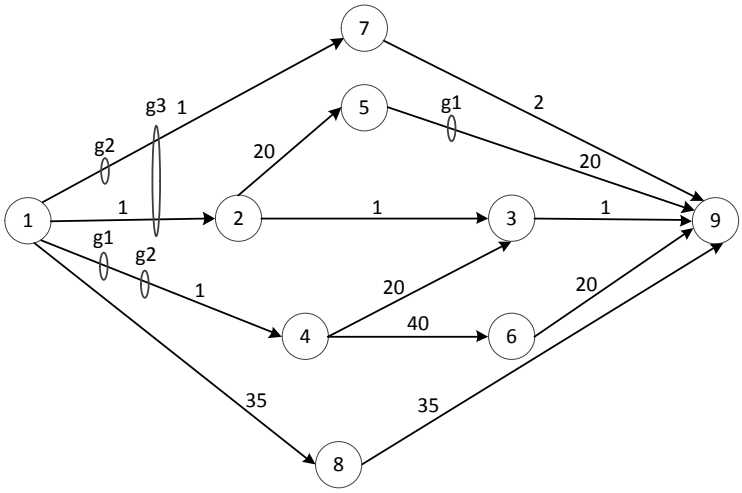



Figura 4.2: Rede que ilustra o melhoramento da transformação da rede para acelerar a obtenção da solução.

Quando se remove o "interlacing" de dois caminhos, os arcos que constituem cada um dos caminhos são utilizados de forma a construir um par de caminhos sem arcos comuns. Se dois arcos de um dos caminhos pertenciam simultaneamente ao mesmo SRLG, e depois de removido o "interlacing" pertencem a caminhos distintos, então o par de caminhos deixa de ser disjunto nos SRLG.

Considere-se o caminho $p = \langle 1,2,3,9 \rangle$ na rede da figura 4.2 e o caminho $q = \langle 1,4,3',2'',5,9 \rangle$ na rede da figura 4.3. O caminho $q$ tem o SRLG $g_1$ associado aos arcos $(1,4)$ e $(5,9)$. Após se remover o "interlacing" e se coalescer os nós, os arcos anteriormente referidos, são distribuídos por cada um dos caminhos do par. Isto conduz a que

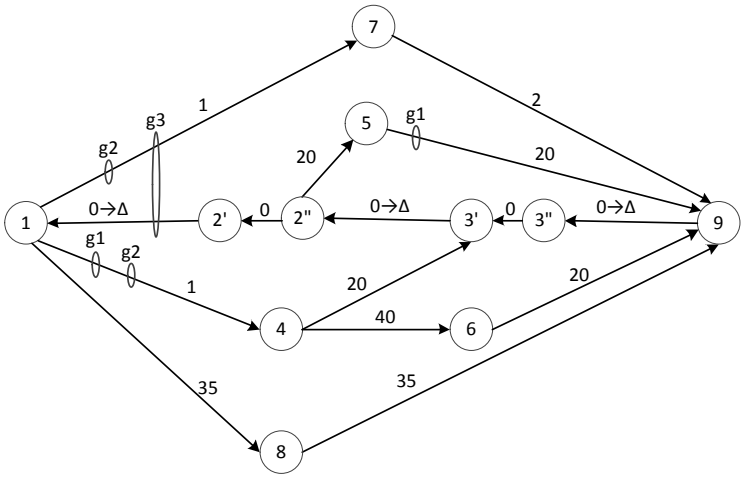



Figura 4.3: Rede que ilustra o melhoramento da transformação da rede para acelerar a obtenção da solução, após ser aplicada a transformação tendo como base o caminho $\langle 1,2,3,9 \rangle$.

não seja encontrado nenhum par de caminhos disjuntos nos SRLG nesta iteração, apesar de ele existir. Caso se escolha $p_2$, o par de caminhos ($p'_2 = \langle 1,2,3,9\rangle, p''_2 = \langle 1,4,6,9\rangle$) obtido é disjunto nos SRLG e tem custo 64. Fica então evidente que na mesma iteração pode ou não haver uma solução, dependendo da forma como a pesquisa dos nós é feita pelo algoritmo de Dijkstra.

Uma vez que o número de iterações máximo considerado foi 2, não vão ser explorados todos os caminhos semente. Nesse caso o último caminho semente a ser explorado é o caminho $p_k = \langle 1,7,9\rangle$, com o qual é obtido o par ($p'_k = \langle 1,7,9\rangle, p''_k = \langle 1,8,9\rangle$) com custo $3+70 = 73$, que é guardado. Como o número de iterações não é suficiente para que se explore o caminho semente $p_k = \langle 1,4,6,9\rangle$, o IMSH não descobre o par de caminhos ($p = \langle 1,4,6,9\rangle, q = \langle 1,2,3,9\rangle$), quando o poderia ter descoberto na primeira iteração.

Assim sendo, há que penalizar a passagem pelos arcos simétricos do primeiro caminho encontrado. Como tal, em vez desses arcos terem custo nulo, passam a ter um custo residual (que depende da gama dos custos), permitindo que em caso de existência de dois caminhos com o mesmo custo seja sempre escolhido o caminho que não passe pelos referidos arcos.

Isto permite que dois caminhos com o mesmo custo, $C$, passem a ter custo diferente. O caminho que não passa por nenhum arco simétrico terá custo $C$. Já o caminho que passe por arcos simétricos terá custo $C + (k \times \Delta)$, onde $k$ é o número de arcos simétricos que são percorridos. Desta forma, independentemente de como a pesquisa é feita pelo algoritmo de Dijkstra, será sempre preferido o caminho que não passe em arcos simétricos, possivelmente conduzindo a uma solução disjunta nos SRLG em menos iterações.

Tudo isto tem especial importância quando o IMSH pára antes de serem explorados todos os caminhos entre o nó de origem 1 e o nó de destino 9. Na rede da figura 4.2, na iteração em que o caminho semente é $p_i = \langle 1,2,3,9\rangle$ o IMSH não encontra um par de caminhos disjuntos nos SRLG. Algumas iterações depois, quando o caminho semente usado é $p = \langle 1,7,9\rangle$, é encontrado o par ($p' = \langle 1,7,9\rangle$,$p'' = \langle 1,8,9\rangle$). Este par é guardado como melhor par até ser atingido o número máximo de iterações. Como o caminho semente $p = \langle 1,4,6,9\rangle$ não chegou a ser inspecionado, a solução obtida tem um custo elevado, porque o IMSH na iteração em que usou o caminho semente $p = \langle 1,2,3,9\rangle$ escolheu um de entre dois caminhos alternativos (com o mesmo custo) na rede transformada.

## 4.3 Algoritmo "Confliting SRLG exclusion – Min Sum" (CoSE-MS)

### 4.3.1 Algoritmo "Confliting SRLG exclusion" (CoSE)

O algoritmo de "Confliting SRLG Exclusion" (CoSE) [9] é uma extensão do algoritmo de "Confliting Link Exclusion" (CoLE) [16], para calcular pares de caminhos disjuntos nos arcos e nos SRLG. Para tal, o CoSE resolve o problema min-min usando a técnica de *"divide-and-conquer"* proposta no CoLE, substituindo o conjunto de arcos em conflito pelo conjunto de SRLG em conflito. Um conjunto de SRLG em conflito, $T$, pode ser definido como o sub-conjunto de SRLG de um caminho ativo, $AP$, que faz com que não seja possível encontrar um caminho de proteção, $BP$, para um caminho que use esse sub-conjunto de SRLG.

A técnica de *"divide-and-conquer"* usado no CoSE consiste em dividir o problema inicial em vários sub-problemas. Cada problema é definido por um conjunto de inclusão ($I$) e por um conjunto de exclusão ($E$). O conjunto $I$ contém os SRLG que podem ser usados pelo $AP$ e que não mais podem ser excluídos em problemas que resultem da sua divisão. Por seu lado os SRLG contidos no conjunto $E$ não podem ser usados pelo $AP$.

Um problema é representado por $P(I,E)$, onde $I$ e $E$ são os conjuntos de inclusão e exclusão, respetivamente. O problema inicial é um problema "vazio", pois $I = \emptyset$ e $E = \emptyset$. O problema inicial é representado por $P(\emptyset,\emptyset)$. Dado um problema $P(I,E)$, o caminho candidato a $AP$ é calculado na rede em que foram removidos os arcos pertencentes aos SRLG em $E$. Seguidamente, numa rede em que foram removidos os SRLG que podem afetar o $AP$ candidato, é determinado o $BP$. Se este existe, o problema corrente foi resolvido. Se o $BP$ não pode ser determinado é preciso determinar o conjunto de SRLG em conflito e gerar novos sub-problemas, como seguidamente se explica.

Se para um determinado $AP$ o conjunto de SRLG em conflito for $T = g_1, g_2, \ldots, g_{|T|}$, então os sub-problemas gerados são $P = (\emptyset, \{g_1\})$, $P = (\{g_1\}, \{g_2\})$,...,$P = (\{g_1, g_2, \ldots, g_{|T|-1}\}, \{g_{|T|}\})$. Cada um destes problemas é resolvido de forma semelhante ao explicado anteriormente, sendo selecionada a melhor solução entre todas as encontradas. Um problema para o qual não é possível determinar um $AP$ é um problema sem solução (que não dá origem a novos problemas).

O CoSE pode facilmente ser adaptado para encontrar pares de caminhos disjuntos nos nós e nos SRLG. Para tal, basta antes de calcular o $BP$, remover da rede, além dos SRLG que afetam o $AP$ candidato, os nós intermédios desse $AP$.

### 4.3.2 "Modified Bhandari's Heuristic" (MBH)

Esta heurística procura determinar um par de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo, numa rede ($G(V,A)$) com custos $l(i,j)$ ($(i,j) \in A$) e os SRLG associados a cada arco. Para tal, recebe o nó de origem, $s$, o nó de destino, $t$, e um caminho semente, $p$.

Tal como no MSH (secção 4.2.1), o MBH [4] começa por aplicar à rede a transformação referida em 3.3 e adicionar $M$ (soma do custo de todos os arcos da rede) aos arcos que estejam em conflito com os SRLG do caminho semente, $p$. Depois todos os arcos que constituem o caminho semente, $p$, são substituídos pelos seus arcos simétricos e com custo simétrico. Nesta rede transformada, é calculado um novo caminho mais curto, $p'$, usando o algoritmo de Dijkstra Modificado. Seguidamente, os nós de $p'$ são coalescidos (os arcos $(i'',i')$ são transformados num nó $i$). Depois é removido o "interlacing" que possa existir entre $p$ e $p'$, originando o par de caminhos $(p,q)$. Por fim, é verificado se o par de caminhos obtidos, $(p,q)$, é disjunto nos SRLG.

O MBH só pode ser usado no caso em que o caminho semente é o caminho mais curto em $G$. Isto deve-se ao facto de ser garantido que não serão criados ciclos negativos na rede transformada, $G'$, neste caso. Uma vez que não há ciclos negativos em $G'$, pode utilizar-se o algoritmo de Dijkstra Modificado ou o algoritmo BFS para se obter um caminho mais curto nesta rede. Já se o MBH fosse usado com um caminho semente que não o caminho mais curto em $G$, ou seja $C_{p_k} > C_{p_1}$, iriam aparecer ciclos negativos. Isto inviabilizaria a utilização do algoritmo de Dijkstra Modificado ou do algoritmo de BFS.

### 4.3.3 Algoritmo "Confliting SRLG exclusion – Min Sum" (CoSE-MS)

O algoritmo "Confliting SRLG exclusion – Min Sum" (CoSE-MS) [4] é uma adaptação do algoritmo CoSE, para a obtenção de pares de caminhos disjuntos no SRLG (assumindo que cada arco pertence pelo menos a um SRLG) de custo aditivo mínimo, ou seja, procura resolver o problema min-sum que o par de caminhos obtidos é disjuntos nos SRLG.

O CoSE-MS é um algoritmo iterativo, em contraste com o algoritmo CoSE que é recursivo. No CoSE-MS cada problema gerado deixa de ter conhecimento do problema que lhe deu origem. Como tal, é necessário considerar um conjunto adicional, o histórico ($H$), que não é mais do que a união de todos os SRLG que pertencem aos conjuntos de exclusão ($E$), de todos os problemas que deram origem ao problema em causa. Assim sendo, um problema passa a ser representado por $P(I,E,H)$, onde $I$ é o conjunto de inclusão, $E$

é o conjunto de exclusão e $H$ é o histórico (união de todos os conjuntos $E$ dos problemas que deram origem ao problema corrente $P$). Os conjuntos $I$ e $E$ têm o mesmo significado que no CoSE.

O primeiro problema a ser resolvido é o problema vazio, que tem $I = E = H = \emptyset$ e que é representado por $P(\emptyset,\emptyset,\emptyset)$.

Para encontrar um caminho mais curto na rede associada a um problema $P$, basta excluir da rede todos os arcos que pertençam a algum SRLG que seja um elemento do conjunto $E \cup H$.

Dado que o algoritmo de CoSE-MS não é recursivo, há necessidade de usar uma pilha de problemas, $S$. A essa pilha de problemas estão associadas as operações: *push(S,P)* (insere um novo problema, $P$, no topo da pilha de problemas), *pop(S)* (remove o elemento que está no topo da pilha $S$), *top(S)* (devolve o problema que está no topo da pilha $S$, sem o remover) e *empty(S)* (devolve "true" se a pilha $S$ estiver vazia, ou "false" caso contrário).

O pseudo-código que permite implementar o algoritmo CoSE-MS adaptado para obter pares de caminhos disjuntos nos nós e nos SRLG é apresentado no algoritmo CoSE–MS. A função *MSH* representa a MSH, apresentada na sub-secção 4.2.1. Já a função *MBH* representa a MBH introduzida na sub-secção 4.2.1.

A função *SRLG_Exclusion* é referente ao algoritmo que permite obter o conjunto $T$ de SRLG em conflito para um dado problema $P$. Essa função é implementada seguindo o algoritmo apresentado na sub-secção 4.3.4.

Fica assim evidente que o CoSE-MS difere do CoSE em alguns aspetos. A primeira diferença e, mais significativa, é o facto de o CoSE ser um algoritmo recursivo, enquanto que o CoSE-MS é um algoritmo iterativo. Além disso, no CoSE é sempre usada o TSA, ao passo que no CoSE-MS é usada o MBH para resolver o primeiro problema ($P(\emptyset,\emptyset,\emptyset)$) e o MSH para os restantes problemas. O MBH é usada em $p_c = p_1$, pois permite obter um par de caminhos disjuntos nos SRLG com um custo menor ou igual ao custo do par obtido usando o MSH [4]. Isto é de extrema importância, pois o algoritmo pára logo na primeira iteração, caso seja encontrado um par de caminhos disjuntos nos SRLG. Caso haja um par de caminhos disjuntos nos SRLG que não seja encontrado na primeira iteração, o algoritmo só termina quando a pilha de problemas $S$ estiver vazia.

No algoritmo de CoSE-MS, quando são criados os novos problemas, o primeiro problema criado tem um conjunto de inclusão vazio ($I_1 = \emptyset$ em $P_1(I_1,E_1,H_1)$) em vez de ser o conjunto de inclusão do problema que lhe deu origem ($I_c$) (como se fazia no algoritmo CoSE). Esta modificação permite que o CoSE-MS encontre um número maior de soluções ótimas. No entanto, o número de problemas gerados vai ser superior (o número de problemas gerados é igual a $|T|$, onde $T$ é obtido usando a função *SRLG_Exclusion*($I_c, p_c$)). Por outro lado, o número de elementos do conjunto de exclusão também aumenta (recorde-se

que a $E_i = E_{i-1} \cup H$), o que garante que o algoritmo termina (após um número de iterações finito).

Na secção E.2 em anexo, pode ser encontrado um exemplo detalhado do funcionamento do algoritmo CoSE-MS.

### 4.3.4 Como encontrar um conjunto de SRLG em conflito (Confliting SRLG Set) para um caminho ativo

Em [9] é proposto um algoritmo, que dado um caminho ativo, um nó de origem $s$ e um nó de destino $t$, permite obter o conjunto de SRLG em conflito. O algoritmo SRLG Exclusion (apresentado na secção C.3 em anexo) foi retirado de [4] e é semelhante ao algoritmo apresentado em [9] adaptado para funcionar para o algoritmo CoSE-MS, excluindo de $T$ os SRLG do conjunto de inclusão $I$.

Este algoritmo vai removendo os arcos associados a cada um dos SRLG do caminho ativo, até não ser possível encontrar outro caminho. Pela definição, esse é o conjunto de SRLG de conflito. Dado que a complexidade do CoSE-MS depende do número de problemas resolvidos, o tamanho do conjunto de SRLG em conflito, $|T|$, influência diretamente o desempenho do algoritmo. Como tal, convém criar um conjunto o mais pequeno possível. Para tal, o algoritmo só adiciona um SRLG ao conjunto de SRLG em conflito, se esse SRLG for comum a todos os caminhos já encontrados, bem como ao $AP$. Isto evita que sejam adicionados SRLG supérfluos ao conjunto de SRLG em conflito.

### 4.3.5 Versão "Empty I" do algoritmo "Confliting SRLG exclusion – Min Sum" (CoSE-MS)

Verificou-se que para determinados pares origem-destino, o algoritmo CoSE-MS demorava muito tempo a convergir (para as redes de teste criadas com base numa rede real fornecida pela PT Inovação), devido à grande quantidade de problemas criados. Como tal, foi efetuada uma alteração no modo como o conjunto de SRLG em conflito ($T$) é calculado.

Caso o problema corrente seja o primeiro problema ($P_1(\emptyset,\emptyset,\emptyset)$) ou caso não seja possível obter um par de caminhos disjuntos nos SRLG, nem tão pouco nos nós, o conjunto $T$ é determinado, usando a função *SRLG_Exclusion*($I_c, p_c$). Caso contrário, $T$ é formado pela intersecção dos SRLG associados ao caminho mais curto determinado para esse problema ($p_c$) e ao par de caminhos disjuntos nos nós que é possível determinar para esse problema ($p$,$q$), excluindo os SRLG contidos no conjunto $I$ do problema corrente ($T = R_{p_c} \cap X \setminus I_c$, $X = R_p \cap R_q$).

**Algoritmo CoSE–MS:** Algoritmo CoSE-MS que permite obter um par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo [4].

**Dados**: Grafo da rede dirigida $G = (V, A)$; nó origem $s$; nó destino $t$; o custo $l(i, j)$, e os SRLG, $R(i, j)$, associados a cada arco $(i, j) \in A$.

**Resultado**: O par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo $(p, q)$

Considere-se uma pilha de problemas vazia, $S$
// $(p,q)$ é o melhor par corrente de custo $C_{(p,q)}$
$(p,q) \leftarrow (\emptyset,\emptyset)$ // Recorde-se que $C_{(\emptyset,\emptyset)} = \infty$
Seja $P_0 \leftarrow (\emptyset,\emptyset,\emptyset)$ o problema vazio
push $(P_0)$
**Enquanto** $\neg$ empty $(S)$ **faz**
    $P_c(I_c,E_c,H_c) \leftarrow$ top $(S)$ // problema corrente
    pop $(S)$ // remove $S$ do topo da pilha de problemas
    $p_c \leftarrow$ caminho mais curto para o problema corrente
    **Se** $C_{p_c} \neq \infty$ // existe um $p_c$
    **então**
        **Se** $I_c = \emptyset \wedge E_c = \emptyset \wedge H_c = \emptyset$ **então**
            $(p',q') \leftarrow$MBH ( $G,s,t,l,R,p_c$)
        **fim**
        **Caso contrário**
            $(p',q') \leftarrow$MSH ( $G,s,t,l,R,p_c$)
        **fim**
        **Se** $C_{(p',q')} \neq \infty$ **então**
            // $q'$ e $p'$ são disjuntos nos SRLG
            **Se** $C_{(p',q')} < C_{(p,q)}$ **então**
                $(p,q) \leftarrow (p',q')$ // Atualiza melhor par corrente
            **fim**
        **fim**
        **Caso contrário**
            $T \leftarrow$SRLG_Exclusion $(I_c, p_c)$
            // Seja $T$ o conjunto $\{A_1, A_2, \ldots A_{|T|}\}$
            $H \leftarrow E_c \cup H_c$
            $P_1(I_1,E_1,H_1) \leftarrow P(\emptyset,\{A_1\},H)$
            push $(S, P_1)$
            $i \leftarrow 2$
            **Enquanto** $i \leq |T|$ **faz**
                $P_i(I_i,E_i,H_i) \leftarrow P(I_{i-1} \cup E_{i-1},\{A_i\},H)$
                push $(S, P_i)$
                $i \leftarrow i+1$
            **fim**
        **fim**
    **fim**
**fim**

Na resolução do problema $P_0$ a função *SRLG_Exclusion*$(I_c, p_c)$ é usada mesmo quando $p_c$ deu origem a um par de caminhos que são disjuntos nos nós (mas não nos SRLG), com o intuito de gerar uma maior variedade de problemas. Nos restantes problemas, quando existe um par de caminhos disjuntos nos nós, mas não nos SRLG, o conjunto $T$ é determinado levando em consideração os SRLG pertencentes ao caminho semente que são comuns a ambos os caminhos do par obtido, pois foram esses SRLG que impediram a obtenção de uma solução no problema corrente – ver linha 26 do algoritmo CoSE–MS EmptyI.

O algoritmo anteriormente descrito, foi designado por *CoSE-MS_EmptyI*, e encontra-se na página 43. Este algoritmo utiliza duas novas versões do MSH e MBH (ver anexo D), as quais devolvem o par de caminhos obtido, caso exista, mesmo quando este não é disjunto nos SRLG; devolve ainda o conjunto de SRLG comum ao par de caminhos calculado (vazio se não existe solução ou se o par obtido é disjunto nos SRLG).

### 4.3.6 Versão "Last I" do algoritmo "Confliting SRLG exclusion – Min Sum" (CoSE-MS)

Para testar a eficiência da implementação com o conjunto de inclusão do primeiro problema criado vazio $I_1 = \emptyset$, quando são adicionados novos problemas à pilha de problemas, foi criado um algoritmo semelhante ao CoSE-MS_EmptyI, mas em que o conjunto de inclusão do primeiro problema criado é igual ao conjunto de inclusão do problema que lhe deu origem ($I_1 = I_c$).

O algoritmo anteriormente referido, foi denominado por *CoSE-MS_LastI* e é praticamente igual ao algoritmo CoSE–MS EmptyI, sendo somente alterada a linha 29 para $P_1(I_1, E_1, H_1) \leftarrow P(I_c, \{A_1\}, H)$.

**Algoritmo CoSE–MS EmptyI:** Algoritmo CoSE-MS_EmptyI que procura obter um par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo, adaptado de [4].

**Dados**: Grafo da rede dirigida $G = (V,A)$; nó origem $s$; nó destino $t$; o custo $l(i,j)$, e os SRLG, $R(i,j)$, associados a cada arco $(i,j) \in A$.

**Resultado**: O par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo $(p,q)$

Considere-se uma pilha de problemas vazia, $S$
// $(p,q)$ é o melhor par corrente de custo $C_{(p,q)}$
$(p,q) \leftarrow (\emptyset,\emptyset)$ // Recorde-se que $C_{(\emptyset,\emptyset)} = \infty$
$P_0 \leftarrow (\emptyset,\emptyset,\emptyset)$, push $(P_0)$
**Enquanto** ¬ empty $(S)$ **faz**
    $P_c(I_c,E_c,H_c) \leftarrow$ top $(S)$ // problema corrente
    pop $(S)$ // remove $S$ do topo da pilha de problemas
    $p_c \leftarrow$ caminho mais curto para o problema corrente
    // Caso exista um caminho para o problema corrente
    **Se** $C_{p_c} \neq \infty$ **então**
        **Se** $P_c = P_0$ **então**
            $(p',q',X) \leftarrow$MBH-v2 $(G,s,t,l,R,p_c)$ // Ver anexo D
        **fim**
        **Caso contrário**
            $(p',q',X) \leftarrow$MSH-v2 $(G,s,t,l,R,p_c)$ // Ver anexo D
        **fim**
        **Se** $X = \emptyset \wedge C_{(p',q')} \neq \infty$ // $q'$ e $p'$ são disjuntos nos SRLG
        **então**
            // não existe par de caminhos disjuntos nos nós nem nos SRLG com $p_c$
            **Se** $C_{(p',q')} < C_{(p,q)}$ **então**
                $(p,q) \leftarrow (p',q')$ // Atualiza melhor par corrente
            **fim**
        **fim**
        **Caso contrário**
            // Se é $P_0$ ou não existe par de caminhos disjuntos nos nós
            **Se** $P_c = P_0 \vee (X = \emptyset \wedge (p',q') = (\emptyset,\emptyset)$ **então**
                $T \leftarrow$SRLG_Exclusion $(I_c, p_c)$
            **fim**
            // Existe par de caminhos disjuntos nos nós mas não nos SRLG
            **Caso contrário**
                $T \leftarrow X \cap R_{p_c} \backslash I_c$
            **fim**
            // Seja $T$ o conjunto $\{A_1,A_2,\ldots,A_{|T|}\}$
            $H \leftarrow E_c \cup H_c$
            $P_1(I_1,E_1,H_1) \leftarrow P(\emptyset,\{A_1\},H)$
            push $(S, P_1)$
            $i \leftarrow 2$
            **Enquanto** $i \leq |T|$ **faz**
                $P_i(I_i,E_i,H_i) \leftarrow P(I_{i-1} \cup E_{i-1},\{A_i\},H)$
                push $(S, P_i)$
                $i \leftarrow i+1$
            **fim**
        **fim**
    **fim**
**fim**

# Capítulo 5

# Análise de resultados

## 5.1 Introdução

Este trabalho foi realizado no âmbito de um contrato de Investigação e Desenvolvimento ente o INESC Coimbra e a PT Inovação. A rede utilizada para teste dos algoritmos, é baseada numa rede SDH, e foi fornecida pela PT inovação.

Tendo presente que o objetivo deste trabalho era a possível integração do código desenvolvido sob a forma de biblioteca partilhada num PCE, todas as rotinas foram testadas na placa UNICOM-V5. Esta placa tem as seguintes característica: CPU G2_LE *core clock* 330MHz com 128 MB de memória RAM.

Com o objetivo de mostrar claramente a diferença de capacidade de processamento da placa *UNICOM-V5* e de um *desktop*, foram obtidos resultados utilizando um computador pessoal com as seguintes características: Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz com 4 GB de memória RAM.

No decorrer dos testes verificou-se que a criação dos objetos com a estrutura da rede (que permitem a utilização das rotinas desenvolvidas neste trabalho) consumiam grande parte do tempo de processamento (aproximadamente 7,69 vezes o tempo de calcular um par de caminhos disjuntos nos nós e nos SRLG, usando a versão Empty I do algoritmo CoSE-MS, na placa de testes). Dado que a topologia da rede raramente sofre alterações, e quando as sofre são muito simples (consistem somente em alterar o custo ou ativar/desativar alguns arcos), foi desenvolvida uma rotina de atualização da estrutura da rede, permitindo reutilizar o objeto já criado. O tempo que essa rotina leva a atualizar a rede é desprezável face ao tempo de cálculo de um par de caminhos disjuntos nos nós e nos SRLG usando a versão Empty I do CoSE-MS.

Foi implementada uma aproximação do tipo ITSA para a resolução do problema de pares de caminhos. Nessa implementação foi utilizado o MPS com um máximo de 1E6

candidatos. Verificou-se que havia pares sem qualquer solução, para os quais o CoSE-MS obtinha um par de caminhos. Por outro lado, não era possível alargar o número máximo de caminhos candidatos, porque isso faria com que o ITSA ficasse muito lento, pois para pares sem solução seria necessário esgotar sempre esse número de candidatos. Assim, para a verificação da qualidade das soluções foi utilizado o CPLEX[1]. Para avaliar a eficácia das heurísticas num *desktop* face ao CPLEX, foram também obtidos resultados (sem criar bibliotecas partilhadas).

As três variantes do algoritmo CoSE-MS (CoSE-MS versão original [4], CoSE-MS EmptyI, CoSE-MS LastI) serão designadas simplesmente por algoritmos CoSE-MS, para referir as três em conjunto. Para diminuir a largura das colunas das tabelas, a versão EmptyI e LastI foram designadas por CoSE-MS-E e CoSE-MS-L, respetivamente. A mesma estratégia foi utilizada para identificar os algoritmos nas figuras.

Nas figuras apresentadas neste capítulo, as barras de erro em torno do valor médio, indicam a amplitude do intervalo de confiança (IC) a 95%. Nas tabelas será apresentado cada valor médio e a respetiva semi-largura do IC.

## 5.2 Análise de desempenho dos algoritmos implementados não considerando SRLG

A rede inicialmente fornecida pela PT inovação é formada por três componentes. Nestes testes considerou-se a rede correspondente ao maior componente conetado na rede fornecida, com 361 nós e 1022 arcos dirigidos. Foram gerados 1000 pares origem destino, usando 10 sementes diferentes para o gerador de números aleatórios.

Os algoritmos para cálculo de caminhos disjuntos nos nós mostraram ser todos muito eficientes. A implementação do algoritmo de Bhandari utilizando o algoritmo de Dijkstra e o algoritmo de Dijkstra Modificado mostrou ser o mais eficiente na obtenção de um par de caminhos disjuntos nos nós, como se pode observar pela tabela 5.1, no que concerne a placa UNICOM-V5. Os tempos obtidos no *desktop* são aproximadamente 14 vezes inferiores ao obtidos na UNICOM-V5.

Verificou-se que na rede de teste existiam no máximo doze caminhos disjuntos entre os nós da rede. Assim, foi utilizado um valor superior a este, vinte (mas poderia ter sido treze). Verificou-se, mais uma vez, que a utilização do algoritmo de Dijkstra e do algoritmo de Dijkstra modificado conduziu à variante com melhor desempenho na placa

[1]Uma vez que a utilização do CPLEX não estava prevista, a Professora Doutora Teresa Gomes decidiu então desenvolver o código para, utilizando o CPLEX , avaliar se seria possível obter todas as soluções ótimas. O autor desta dissertação apenas incorporou esse código no, já desenvolvido.

| Algoritmo | *UNICOM-V5* (segundos) | *desktop* (segundos) |
|---|---|---|
| Bhandari (Dijkstra + Dijkstra Mod.) | 5.252±0.023 | 0.341±0.002 |
| Bhandari (Dijkstra + BFS) | 6.185±0.051 | 0.359±0.008 |
| Bhandari (BFS + BFS) | 6.508±0.067 | 0.322±0.010 |
| Suurballe (Dijkstra + Dijkstra) | 6.743±0.017 | 0.379±0.007 |

Tabela 5.1: Tabela com os tempos de execução para quatro alternativas de cálculo de um par de caminhos disjuntos nos nós, de custo aditivo mínimo (1000 pares origem destino).

UNICOM-V5, como se pode verificar na tabela 5.2. Os tempos obtidos no *desktop* são aproximadamente 18 vezes inferiores ao obtidos na UNICOM-V5.

## 5.3 Análise de desempenho dos algoritmos implementados considerando SRLG

Nestes testes considerou-se a rede correspondente ao maior componente bi-conetado na rede fornecida (sem informação acerca dos SRLG), com 231 nós e 942 arcos dirigidos. A ausência de soluções ótimas numa rede bi-conetada depende apenas da distribuição dos SRLG. Tendo como objetivo avaliar o desempenho dos algoritmos CoSE e IMSH, foram criadas 10 imagens da rede com diferentes distribuições dos SRLG.

As 10 redes foram criadas tendo como base a maior componente bi-conetada da rede. Para tal, foi adicionada a informação dos SRLG a que cada arco está associado. Cada arco foi associado de forma aleatória a um número de SRLG, que varia entre zero e quatro. Convém também referir que cada SRLG só foi associado no máximo a 4 arcos.

Utilizando as 10 imagens da rede procurou obter-se uma solução para todos os pares origem destino nessa rede, de forma a avaliar a qualidade das soluções obtidas pelas

| Algoritmo | *UNICOM-V5* (segundos) | *desktop* (segundos) |
|---|---|---|
| *K*-Bhandari (Dijkstra + Dijkstra Mod.) | 9.138±0.092 | 0.479±0.012 |
| *K*-Bhandari (Dijkstra + BFS) | 10.567±0.131 | 0.508±0.010 |
| *K*-Bhandari (BFS + BFS) | 10.912±0.137 | 0.478±0.008 |

Tabela 5.2: Tabela com os tempos de execução para três alternativas de cálculo de um conjunto de *K* de caminhos disjuntos nos nós, de custo aditivo mínimo (1000 pares origem destino).

| | *IMSH* (segundos) | *CoSE-MS-E* (segundos) | *CoSE-MS-L* (segundos) | *CoSE-MS* (segundos) |
|---|---|---|---|---|
| $k = 5$ | 75.625± 1.765 | 12.964±1.420 | 12.972±1.423 | 17.956± 2.469 |
| $k = 10$ | 90.363± 1.811 | 17.067±1.995 | 17.086±1.998 | 28.146± 4.343 |
| $k = 20$ | 118.410± 2.067 | 22.194±2.998 | 22.198±2.986 | 46.960± 7.729 |
| $k = 50$ | 197.771± 3.384 | 24.996±3.761 | 24.847±3.705 | 99.247± 16.781 |
| $k = 100$ | 322.100± 6.738 | 25.235±3.893 | 24.992±3.783 | 178.826± 29.706 |
| $k = 200$ | 555.889±15.752 | 25.244±3.902 | 25.001±3.792 | 320.402± 48.519 |
| $k = 500$ | 1217.657±45.937 | 25.242±3.899 | 25.000±3.791 | 688.340±102.383 |
| $k = 1000$ | 2260.720±97.060 | 25.243±3.899 | 25.000±3.792 | 1247.258±177.228 |

Tabela 5.3: Tabela com os tempos de execução para os algoritmos de cálculo de um par de caminhos mais curtos disjuntos nos nós e nos SRLG (IMSH, CoSE-MS EmptyI, CoSE-MS LastI, CoSE-MS), com $k = \{5, 10, 20, 50, 100, 200, 500, 1000\}$, medidos na *UNICOM-V5* usando a biblioteca partilhada (1000 pares origem destino).

heurísticas. Considerando que o tempo de CPU numa aplicação real terá de ser necessariamente limitado, limitou-se o número de número máximo de iterações (algoritmo IMSH) ou número máximo de problemas resolvidos (algoritmos CoSE-MS), de forma a poder ser avaliado o seu desempenho em função deste fator. Nos resultados apresentados, o número máximo de iterações (algoritmo IMSH) ou número máximo de problemas resolvidos (algoritmos CoSE-MS) considerados foram $k = 5, 10, 20, 50, 100, 200, 500, 1000$.

### 5.3.1 Tempo de execução

Foram gerados aleatoriamente 1000 pares, origem destino, para cada uma das 10 redes com diferentes distribuições de SRLG, e registou-se o tempo total para obter um par de caminhos disjuntos nos nós e nos SRLG. São apresentados na tabela 5.3 os valores obtidos na placa UNICOM-V5, usando a biblioteca partilhada desenvolvida. Nessa tabela observa-se que os algoritmos CoSE-MS são consistemente mais rápidos que o IMSH para o mesmo valor de $k$. As duas novas variantes do CoSE-MS são muito semelhante e superiores à versão original. Como se irá ver na seção 5.3.2, os algoritmos CoSE-MS estabilizam, no que toca à percentagem de soluções ótimas, para um número máximo de problemas resolvidos igual a 50. Observando os tempos para esse número de problemas resolvidos, verifica-se que as novas variantes do CoSE-MS para $k = 50$ são aproximadamente 7 vezes mais rápida do que o IMSH para esse mesmo $k$.

Na tabela 5.4 são apresentados os tempos de execução no *desktop*, de forma a ficar evidente a diferença desempenho entre este e a placa UNICOM-V5. Comparando esses

| | *IMSH* (segundos) | *CoSE-MS-E* (segundos) | *CoSE-MS-L* (segundos) | *CoSE-MS* (segundos) |
|---|---|---|---|---|
| $k = 5$ | 3.198± 0.469 | 0.651±0.109 | 0.651±0.109 | 0.883±0.141 |
| $k = 10$ | 3.942± 0.588 | 0.817±0.134 | 0.820±0.137 | 1.368±0.236 |
| $k = 20$ | 5.321± 0.790 | 1.060±0.183 | 1.061±0.181 | 2.290±0.419 |
| $k = 50$ | 9.230± 1.343 | 1.189±0.209 | 1.184±0.211 | 4.788±0.875 |
| $k = 100$ | 15.315± 2.181 | 1.205±0.224 | 1.192±0.218 | 8.485±1.583 |
| $k = 200$ | 26.799± 3.762 | 1.205±0.225 | 1.191±0.218 | 14.960±2.824 |
| $k = 500$ | 59.149± 8.243 | 1.204±0.225 | 1.192±0.219 | 30.025±4.111 |
| $k = 1000$ | 110.135±15.323 | 1.141±0.142 | 1.129±0.139 | 54.743±7.438 |

Tabela 5.4: Tabela com os tempos de execução para os algoritmos de cálculo de um par de caminhos mais curtos disjuntos nos nós e nos SRLG (IMSH, CoSE-MS EmptyI, CoSE-MS LastI, CoSE-MS), com $k = \{5, 10, 20, 50, 100, 200, 500, 1000\}$, medidos num *desktop* usando a biblioteca partilhada.

valores com os da tabela 5.3, observa-se que este são aproximadamente 20 vezes superiores.

No que concerne à utilização da memória verificou-se que o máximo ocupado, nos testes realizados, foi em todos os casos inferior a 2 MiB. Isto pode-se confirmar na tabela 5.5.

### 5.3.2 Eficácia das heurísticas face ao CPLEX

Foram consideradas as seguintes medidas de desempenho, em função do número máximo de iterações (algoritmo IMSH) ou número máximo de problemas resolvidos (algoritmos CoSE-MS):

- A percentagem de soluções ótimas encontradas, usando como referência o valor ótimo obtido pelo CPLEX (versão 12.3).

| | (páginas de memória de 4096 bytes) | |
|---|---|---|
| | *usando a biblioteca partilhada* | *não usando a biblioteca partilhada* |
| Classe *CConstrRede* | 394 | 385 |
| Classe *CMPS* | 488 | 474 |
| Classe *CCoSE_MS* | 485 | 469 |

Tabela 5.5: Tabela com a memória ocupada pelos objetos das classes *CConstrRede*, *CMPS* e *CCoSE_MS*, criados e medidos na *UNICOM-V5*.

- Número médio de iterações (algoritmo IMSH) ou número de problemas resolvidos (algoritmos CoSE-MS).

- Média do erro relativo para as soluções sub-ótimas, usando como referência o valor ótimo obtido pelo CPLEX.

- Média do erro relativo estimado pelo IMSH, para as soluções sub-ótimas, segundo a condição de otimalidade dada na equação (4.6).

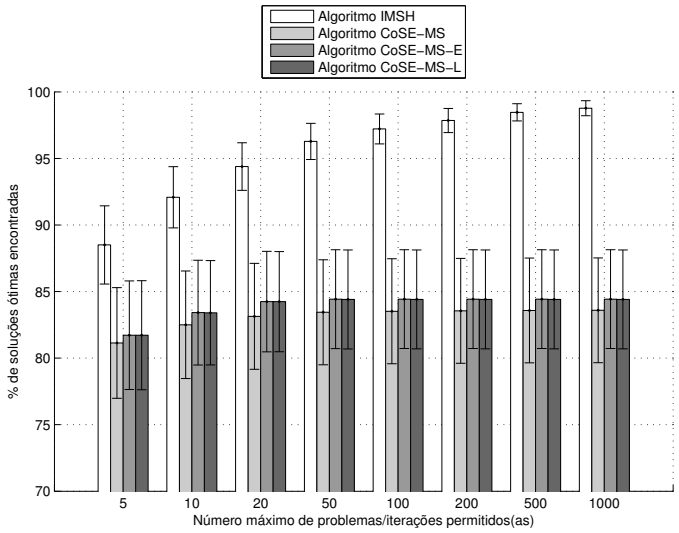



Figura 5.1: Percentagem média de soluções ótimas encontradas pelos algoritmos de IMSH, CoSE-MS EmptyI, CoSE-MS LastI, CoSE-MS, com um número máximo de iterações (IMSH) ou problemas resolvidos (CoSE-MS) $k = 5, 10, 20, 50, 100, 200, 500, 1000$, usando a solução ótima obtida pelo CPLEX como referência.

A percentagem de soluções ótimas encontradas, pode ser observada na figura 5.1. Verifica-se que a percentagem de soluções ótimas obtidas pelo IMSH é consistentemente superior à percentagem de soluções ótimas obtidas por qualquer uma das variantes do algoritmo CoSE-MS. A versão original do algoritmo CoSE-MS apresenta ligeiramente menos soluções ótimas que as duas novas variantes, as quais têm desempenho muito semelhante. O número de soluções sub-ótimas obtidas pelo IMSH aumenta com o número máximo de iterações permitido (88,5%, 96,3% e 98,5%, para $k = 5, 50, 500$ respetivamente); no caso dos algoritmos CoSE-MS LastI ou CoSE-MS EmptyI esse aumento parece estabilizar a partir de um número máximo de problemas permitidos igual a 50 (apenas varia o quarto dígito significativo), sendo o seu valor igual a 81,7%, 83,4%, 84,2% e 84,4%, para $k = 5, 10, 20, 50$ respetivamente.

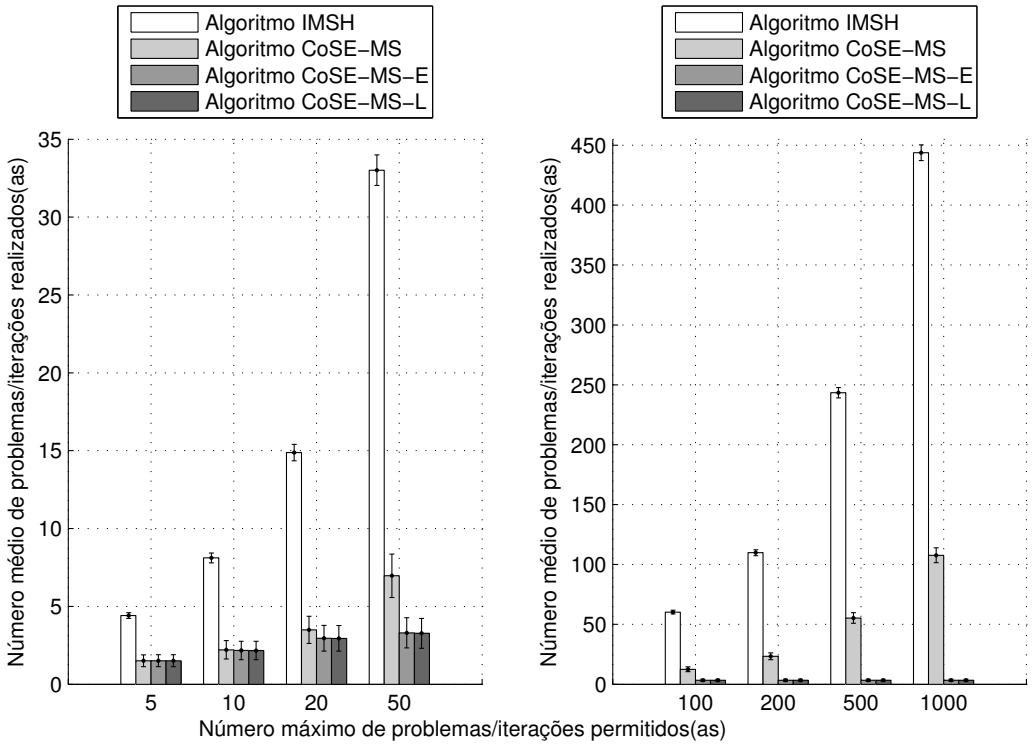



Figura 5.2: Número médio de problemas/iterações resolvidos(as) pelos algoritmos IMSH, CoSE-MS EmptyI, CoSE-MS LastI e CoSE-MS, com um número máximo de iterações (IMSH) ou problemas resolvidos (CoSE-MS) $k = 5, 10, 20, 50, 100, 200, 500, 1000$.

Na secção anterior verificou-se que o tempo de execução das novas variantes do CoSE-MS é significativamente inferior ao tempo de execução do IMSH, o que poderá compensar a sua menor precisão.

Na figura 5.2 pode observar-se o número médio de problemas resolvidos pelos algoritmos CoSE-MS e o número de iterações realizadas pelo algoritmo IMSH. É possível verificar que o IMSH utiliza um número significativamente maior de iterações do que as variantes do algoritmo CoSE-MS. Isto deve-se ao facto de o IMSH tentar atingir a condição de otimalidade para a melhor solução que encontrou até ao momento. Os algoritmos CoSE-MS aumentam ligeiramente o número de problemas resolvidos com um número máximo de problemas permitidos, mas o seu valor pouco varia para $k \geq 50$, o que é consistente com os resultados da figura 5.1.

O erro relativo das soluções sub-ótimas obtidas pelo algoritmo IMSH, é menor do que o valor correspondente para as soluções sub-ótimas obtidas pelos vários algoritmos CoSE-MS, como se pode observar na figura 5.3. Verifica-se ainda uma diminuição do erro relativo com o aumento do número de iterações no caso do algoritmo IMSH. Em contraste, os algoritmos CoSE-MS apresentam apenas uma ligeira diminuição com o aumento de número máximo de problemas permitidos, que mais uma vez estabiliza com $k = 50$.

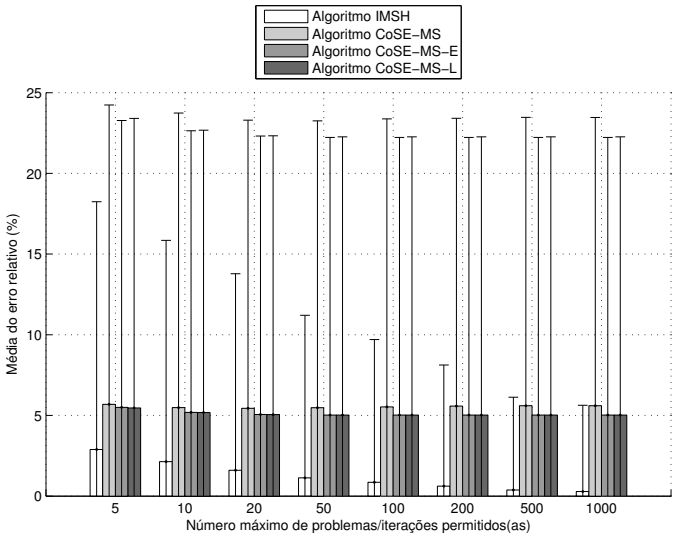



Figura 5.3: Erro relativo médio (tendo como referência as soluções ótimas obtidas pelo CPLEX) para os algoritmos IMSH, CoSE-MS EmptyI, CoSE-MS LastI e CoSE-MS, com um número máximo de iterações (IMSH) ou problemas resolvidos (CoSE-MS) $k = 5, 10, 20, 50, 100, 200, 500, 1000$.

Note-se a grande amplitude dos IC face à média desses mesmos erros, e em particular dos algoritmos CoSE-MS. Novamente, as novas variantes do CoSE-MS apresentam ligeiramente melhor desempenho do que a versão original.

O algoritmo IMSH tem alguma dificuldade em verificar a otimalidade da melhor solução corrente, ou seja, a condição expressa pela equação (4.6) é pouco eficiente. Isto pode ser verificado comparando o erro relativo das soluções sub-ótimas na figura 5.3 (inferior a 3%) e na figura 5.4 (superior a 30%).

Para encontrar as soluções ótimas para cada rede, o CPLEX demorou em média 381,131$\pm$130,080 segundos. Comparando este valor com os tempos de execução das heurísticas apresentados na tabela 5.6, pode concluir-se que as heurísticas CoSE-MS para $k \leq 100$ são duas ordens de grandeza inferiores aos tempos do CPLEX (embora apenas obtenham cerca de 84,4% de soluções ótimas). Por seu lado a heurística IMSH, com $k = 200$, é apenas uma ordem de grandeza mais rápida do que o CPLEX, apresentando uma taxa de soluções ótimas de 97,9%.

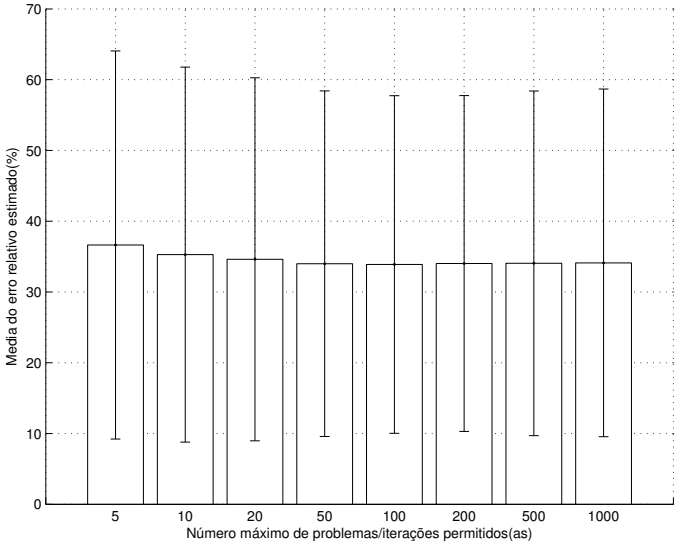



Figura 5.4: Erro relativo estimado médio pelo IMSH, para um número máximo de iterações (IMSH) ou problemas resolvidos (CoSE-MS) $k = 5, 10, 20, 50, 100, 200, 500, 1000$.

| | *IMSH* (segundos) | *CoSE-MS-E* (segundos) | *CoSE-MS-L* (segundos) | *CoSE-MS* (segundos) |
|---|---|---|---|---|
| $k = 5$ | 3.115± 0.496 | 0.654±0.107 | 0.640±0.105 | 0.878±0.150 |
| $k = 10$ | 3.813± 0.601 | 0.828±0.153 | 0.829±0.149 | 1.380±0.249 |
| $k = 20$ | 5.150± 0.818 | 1.078±0.205 | 1.078±0.204 | 2.302±0.424 |
| $k = 50$ | 8.887± 1.385 | 1.213±0.240 | 1.205±0.239 | 4.809±0.881 |
| $k = 100$ | 15.396± 2.198 | 1.226±0.253 | 1.213±0.247 | 8.537±1.591 |
| $k = 200$ | 28.155± 4.274 | 1.226±0.254 | 1.212±0.247 | 15.054±2.923 |
| $k = 500$ | 62.123± 9.484 | 1.226±0.254 | 1.212±0.248 | 31.639±6.566 |
| $k = 1000$ | 115.344±17.924 | 1.272±0.241 | 1.259±0.235 | 57.974±11.247 |

Tabela 5.6: Tabela com os tempos de execução para os algoritmos de cálculo de um par de caminhos mais curtos disjuntos nos nós e nos SRLG (IMSH, CoSE-MS EmptyI, CoSE-MS LastI, CoSE-MS), com $k = \{5, 10, 20, 50, 100, 200, 500, 1000\}$, medidos num *desktop* não usando a biblioteca partilhada.

# Capítulo 6

# Conclusão

Dadas as necessidades de melhoramento dos serviços oferecidos pelos operadores de telecomunicações, no que toca à fiabilidade e à disponibilidade, a proteção em redes de telecomunicações tem especial importância. A proteção em redes de telecomunicações pode ser conseguida através da implementação de mecanismos de recuperação, que utilizam proteção global ao caminho. Uma forma de implementar essa proteção consiste em determinar pares (ou conjuntos) de caminhos disjuntos.

O GMPLS veio permitir a flexibilização ao nível do plano de controlo, dado que tornou possível o estabelecimento de ligações entre dispositivos com diferentes tipos de comutação (comutação de pacotes, comutação no tempo e comutação no comprimento de onda). Esta versatilidade é viabilizada através de um novo modelo baseada num PCE [3], que tem três elementos constituintes (PCE, PCC e TED), distribuídos pela rede (nos nós) e que interagem entre si de forma a estabelecerem ligações ponto a ponto, incluindo ligações protegidas. Devido à arquitetura distribuída do GMPLS, os PCE podem ser integrados em equipamentos com baixa capacidade de processamento e com pouca memória, quando comparados com um *desktop*.

O conceito de SRLG veio fazer com que fosse possível estabelecer uma relação entre as causas de uma falha (riscos) e as ligações lógicas da rede. A determinação de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo, é um problema do tipo min-sum. Quer isto dizer que, em geral, este problema visa minimizar o custo dos recursos utilizados na proteção global dedicada, para um dado pedido de ligação. Se o objetivo for partilhar a largura de banda de proteção, é frequente utilizar-se uma aproximação do tipo min-min. Nesta aproximação, procura determinar-se para *AP* o caminho mais curto que é possível proteger, e para *BP* o caminho mais curto na rede em que foram removidos os arcos (ou nós) do caminho a proteger.

Um dos objetivos deste trabalho foi implementar rotinas de cálculo de um par (ou

um conjunto) de caminhos disjuntos nos nós, de custo aditivo mínimo, para aplicar num PCE. Isto implica cuidados especiais no que concerne à memória utilizada. Como tal, optou por se usar uma estrutura em FRSF. No entanto, o tempo de processamento também é uma característica que importa salvaguardar. Uma vez que todos os algoritmos implementados requerem uma transformação da rede que implica a introdução de novos arcos, foi proposta uma adaptação da FRSF, de forma a permitir realizar rapidamente essa transformação sem duplicar toda a rede.

A melhor forma encontrada para verificar se as rotinas desenvolvidas eram adequadas ao uso num PCE foi testá-las num sistema embebido. O sistema embebido utilizado foi uma placa UNICOM-V5. Para testar as rotinas de cálculo de um par (ou um conjunto) de caminhos disjuntos nos nós, de custo aditivo mínimo, foi utilizada um rede de teste com 361 nós e 1022 arcos dirigidos.

Através dos resultados dos testes efetuados na placa da UNICOM-V5, foi possível constatar que as rotinas cumpriram o requisitos impostos à partida, no que toca ao tempo de processamento e à memória utilizada.

O outro objetivo deste trabalho foi implementar uma rotina para calcular um par de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo. Para tal foram implementadas duas heurísticas, a IMSH e a CoSE-MS. No decorrer do trabalho foram criadas duas novas variantes do CoSE-MS, CoSE-MS EmptyI e CoSE-MS LastI. O objetivo inerente à criação destas duas variantes foi diminuir o número de problemas a resolver, tornando o algoritmo mais eficiente, sem comprometer a sua precisão.

Os testes realizados com estas duas heurísticas (para uma rede com 231 nós e 942 arcos dirigidos com 10 distribuições diferentes de SRLG) permitiram confirmar que as duas variantes criadas apresentam uma melhor relação entre eficiência computacional e precisão do que o CoSE-MS. Assim, foi possível concluir que o uso das duas novas versões do CoSE-MS num PCE é viável. Já o IMSH, apesar da sua elevada precisão apresenta tempos de execução demasiado elevados para ser integrado num PCE. No entanto, revelou ser um boa opção para um ambiente de gestão, devido à sua elevada precisão.

# Anexo A

# Descrição da representação da rede e opções de implementação

## A.1 "Foward and Reverse Star Form"

O *"Forward and Reverse Star Form"* (FRSF) [1] armazena a representação da rede numa única tabela (que contém a informação relativa a cada arco da rede) e tem ainda três vetores auxiliares (point, rpoint e trace) para permitir um acesso "quase" direto a cada elemento. Esta representação é obtida através da fusão de duas representações mais básicas. São elas o *"Forward Star Form"* (FSF) e o *"Reverse Star Form"* (RSF) [1].

O FSF permite um acesso por cauda a cada arco contido na tabela. Para tal, os arcos são ordenados por cauda e depois numerados de 1 até $m$, onde $m$ é o número de arcos da rede. Os arcos com a mesma cauda são ordenados de forma arbitrária. No entanto, pode-se escolher ordenar os arcos com a mesma cauda de acordo com um determinado critério, como a ordem crescente das cabeças ou a ordem crescente dos custos de cada arco, caso isso traga alguma vantagem. Depois a informação relativa a cada arco (cauda,

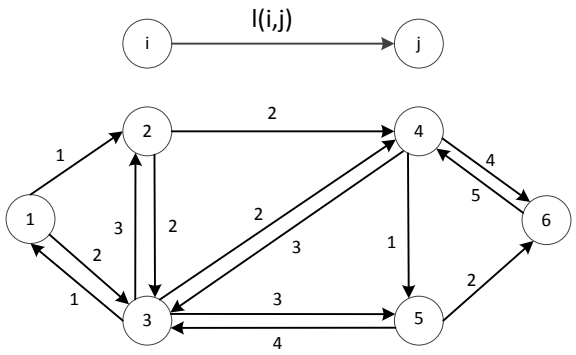



Figura A.1: Rede modelo usada para construir as estruturas.

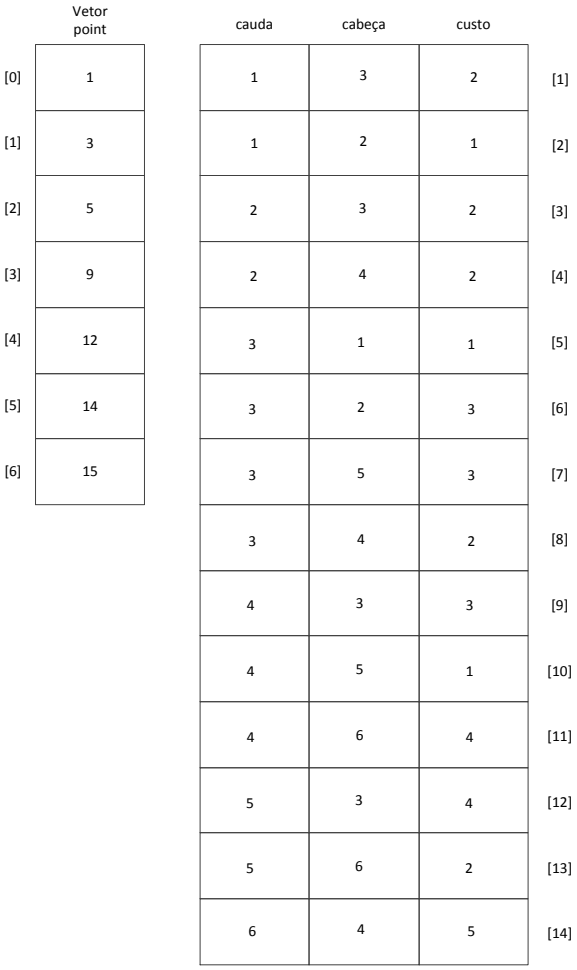



Figura A.2: Representação em "Foward Star Form" da rede da figura A.1.

cabeça, custo, etc) é guardada em cada elemento da tabela.

É importante salientar que como o "software" foi desenvolvido em C++, os índices da tabela começam em zero (tendo sido reservado esse valor para situações anómalas) pelo que o primeiro elemento da tabela de arcos fica vazio. Para se poder aceder diretamente a cada elemento por cauda surge o vetor *point*. Cada elemento deste vetor contém o índice da tabela do primeiro arco do conjunto de arcos com cauda no nó $i = 1,...,n$ (onde $n$ é o número de nós da rede). Como os índices dos vetores em C++ começam em zero, temos que *"point[i-1]"* é o índice do primeiro arco com cauda no nó $i$. A diferença entre dois elementos consecutivos do vetor *point* dá o número de arcos com cauda no nó $i$ ( $point[i] - point[i-1]$ é o número de arcos com cauda $i$). Se $point[i-1] = point[i]$, não há arcos com cauda no nó $i$. Uma vez que a posição 0 da tabela de arcos não foi utilizada, $point[0] = 1$ e $point[n] = m+1$ (onde $m$ é o número de arcos da rede).

A representação em FSF é eficiente para aceder aos arcos sabendo a sua cauda. Para aceder aos arcos pela sua cabeça de forma eficiente é necessário usar uma representa-

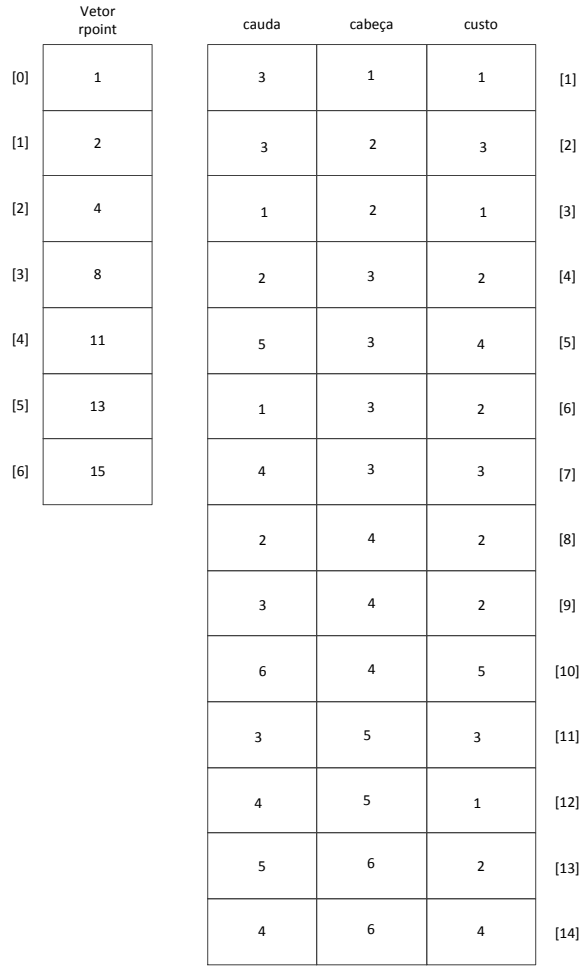



Figura A.3: Representação em "Reverse Star Form" da rede da figura A.1.

ção em RSF. O primeiro passo é ordenar a tabela de arcos pelas suas cabeças de forma crescente. De forma análoga ao vetor *point*, é criado o vetor *rpoint*, ou seja, cada elemento deste vetor contém o índice do primeiro arco, na tabela com os dados dos arcos onde começa cada conjunto de arcos com cabeça num nó $i = 1,\ldots,n$. A diferença entre dois elementos consecutivos do vetor *rpoint* dá o número de arcos com cabeça no nó $i$ ($rpoint[i] - rpoint[i-1]$ é o número de arcos com cabeça em $i$). Se $rpoint[i-1] = rpoint[i]$, não há arcos com cabeça no nó $i$. Tal como na representação em *"Forward Star Form"*, $rpoint[0] = 1$ e $rpoint[n] = m+1$, pois a posição 0 da tabela de arcos não é usada.

Se se guardar em memória simultaneamente as duas representações descritas anteriormente, há informação duplicada. Assim, para aceder rapidamente a um arco sabendo a sua cabeça, utilizando a FSF, precisamos de guardar um vetor auxiliar ($trace$), em articulação com o $rpoint$. Cada um dos elementos do vetor $trace$ é um índice de um elemento da tabela com os dados de cada arco. Assim, dado o nó $j$ é simples localizar na estrutura em FSF todos os arcos com cabeça no nó $j$ (sendo $u$ a posição dos arcos com cabeça no

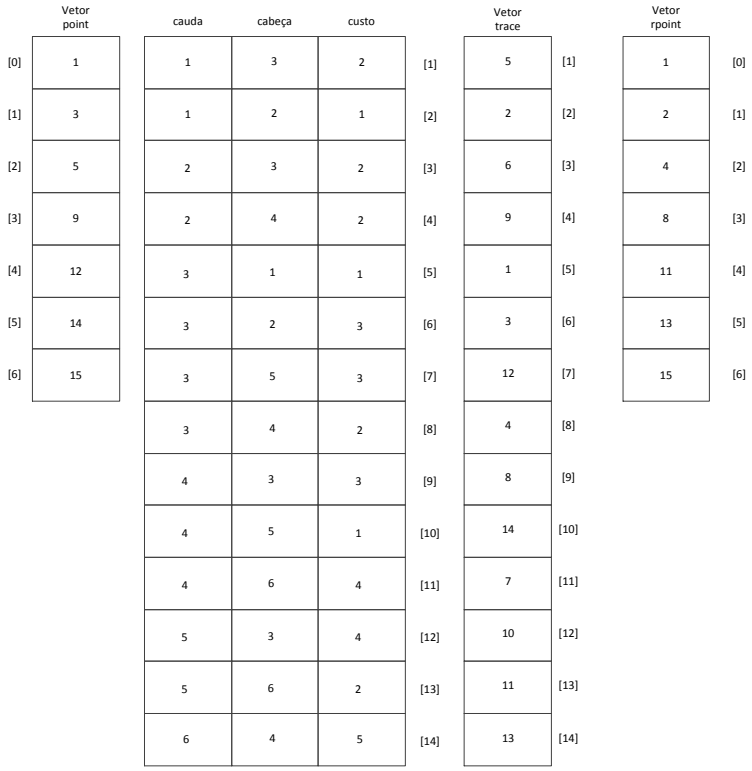



Figura A.4: Representação em "Foward and Reverse Star Form" da rede da figura A.1.

nó $j$ na estrutura em FSF):

$$u = trace[k], k = rpoint[j-1]+1 \ldots rpoint[j]-1 \tag{A.1}$$

Isto simula a ordenação da tabela com a informação dos arcos de acordo com a sua cabeça, mas com a vantagem de necessitar apenas de dois vetores adicionais à FSF, o vetor $rpoint$ de dimensão $n$ e o vetor $trace$ de dimensão $m$.

## A.2 Porquê usar "Foward and Reverse Star Form"?

A "Forward and Reverse Star Form" foi escolhida porque ocupa pouco espaço em memória, em comparação com outras alternativas (como seja a representação em "Adjacency Lists" [1]). A FRSF permite aceder rapidamente a cada arco sabendo a sua chave (posição na tabela) ou localizar facilmente um arco dada a sua cauda, cabeça ou ambos. Esta forma de representar uma rede é fácil de implementar em C++, sobretudo recorrendo a objetos "Standard Template Library". A coluna "cauda" na representação "Foward Star Form" e a coluna "cabeça" na representação em "Reverse Star Form" são redundantes. Contudo, são utilizadas para mais facilmente se identificar a cauda (cabeça) de um arco na posição $k$. Caso contrário, dada a localização de um arco, a sua identificação implica fazer uma

pesquisa binária no vetor *point* (*rpoint*) no caso da "Forwad Star Form" ("Reverse Star Form"), para saber a respetiva cauda (cabeça).

Como em tudo, existem sempre limitações. Neste caso a limitação inerente a este tipo de estrutura é o facto de a inserção e a remoção de arcos no "array" implicar a reconstrução dos vetores *point*, *rpoint* e *trace*. Isto requer uma complexidade da ordem $\mathcal{O}(|m|)$ (onde $m$ é o número de arcos da rede). Este facto é importante no que diz respeito ao *Vertex-splitting* imposto nos algoritmos de cálculo de pares de caminhos com custo aditivo mínimo disjuntos nos nós (ver secção 3.3).

Para contornar o problema anteriormente exposto, criou-se uma estrutura denominada de "Hybrid Foward and Reverse Star Form". Esta estrutura consiste em inserir o espaço para um arco adicional (na realidade insere-se um arco inativo) no fim de cada conjunto de arcos com a mesma cauda. Tal é feito na construção da tabela com os dados de cada arco. Contudo, é necessário marcar na estrutura estes espaços com sendo arcos inativos. Assim sendo, dentro dos dados referentes a cada arco foi adicionado uma máscara binária de oito bits, em que o bit menos significativo (bit 0) indica o estado de atividade do arco (o arco está ativo se esse bit estiver a 1 e inativo caso contrário).

## A.3 Transformação da rede

Os algoritmos de cálculo de pares de caminhos disjuntos de custo aditivo mínimo de um nó origem $s$ para um nó destino $t$, requerem uma transformação da rede, com base num caminho semente. Essa transformação implica a inversão do sentido dos arcos desse caminho (com possível alteração do seu custo). Neste trabalho deseja obter-se pares de caminhos disjuntos nos nós, o que requer a prévia divisão dos nós intermédios ("Vertex-Splitting") nesse caminho. É aqui explicado como essa transformação foi implementada, modificando a FRSF de forma a minimizar o impacto das alterações necessárias na representação da rede, procurando desta forma obter um código eficiente.

Foram intencionalmente inseridos arcos inativos de reserva no fim de cada conjunto de arcos com cauda num nó $i = 1, ..., n$, onde $n$ é igual ao número de nós da rede. Estes arcos inativos servem para implementar a transformação à rede (ver secção 3.3) sem ter que reconstruir os vetores "point", "rpoint" e "trace".

A transformação da rede consiste em dividir os nós do caminho ("Vertex-Splitting") e depois inverter os arcos desse caminho (caso apareçam arcos paralelos é necessário removê-los).

O "Vertex-Splitting" consiste em dividir cada nó, $i$, em dois nós, $i'$ e $i''$. Depois é necessário redirecionar todos os arcos com cabeça no nó $i$ para $i'$ e todos os arcos com

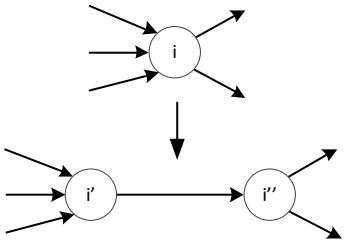



Figura A.5: Exemplo da divisão de um nó.

cauda no nó $i$ para $i''$. A divisão dos nós é ilustrada na figura A.5.

A figura A.6 exemplifica a forma com é feita a inversão dos arcos de um caminho.

A transformação completa, divisão dos nós intermédios de um caminho seguida da inversão do sentido dos arcos desse caminho, pode ser visualizada na figura A.7.

Coloca-se a questão se deveria optar-se por fazer com que depois do *Vertex-splitting* os nós $i$ do caminho mais curto, deveriam coincidir com os nós $i'$ ou $i''$ (na representação da rede, sem duplicação da mesma). Após uma análise cuidada das duas opções, foi possível obter algumas conclusões quanto ao número de operações envolvidas (note-se que não será necessário inverter o arco que liga um nó dividido, pois na implementação será desde logo introduzido com o sentido final):

1. Seja $m_p$ o número de arcos do caminho mais curto, $p$;
   $\delta_p^+$ o número de arcos emergentes de nós intermédios do caminho $p$ (ou seja, todos os nós do caminho exceto a origem e o destino);
   $\delta_p^-$ o número de arcos incidentes em nós intermédios do caminho $p$;
   $\gamma$ o número de arcos do caminho que possuem arcos simétricos.

2. Operações no caso $i = i'$:

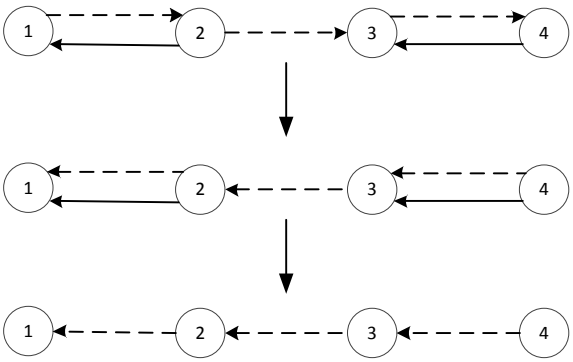



Figura A.6: Exemplo da inversão do sentido dos arcos de um caminho (pode originar arcos paralelos que são removidos).

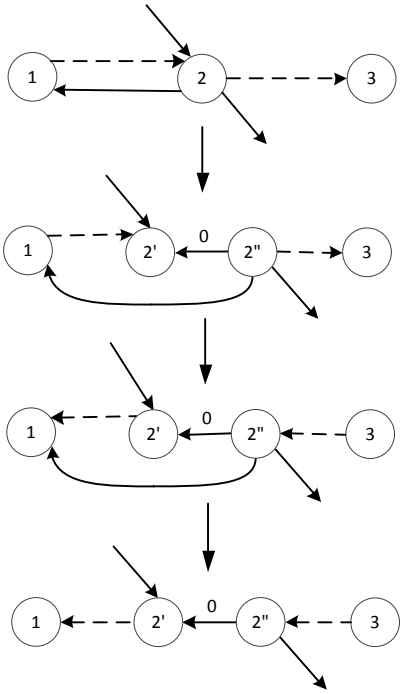



Figura A.7: Exemplo da transformação completa (divisão dos nós intermédios e, inversão dos arcos do caminho $\langle 1,2,3\rangle$, seguido da supressão do arco $(2,1)$).

(a) Para a criação dos arcos $(i',(i-1)'')$ (recorde-se que nó $(i-1)$ representa o nó que antecede o nó $i$ no caminho mais $p$), é necessário alterar a cabeça de um arco de reserva na estrutura, $(i',i'')$, por cada arco do caminho $p$.

O número *total de modificações de cabeças de arcos* da estrutura é $m_p$.

(b) Devido ao "Vertex-Splitting" todos os arcos com cauda no nó $i$ passam a ter cauda no nó $i''$. Como tal é necessário inativar todos os arcos que tenham cauda no nó $i$. Isto equivale a fazer $\delta_p^+$ alterações de estado.

A inversão dos arcos que constituem o caminho $p$ implica não só a modificação de cabeças descrita em 2a, mas também a ativação desses arcos. Assim há que contabilizar $m_p$ mudanças de estado desses arcos.

Para que a transformação da rede esteja completa, é necessário remover os arcos paralelos que apareçam na rede. O número de arcos paralelos que aparecem é igual a $\gamma$. Há então que contar com mais $\gamma$ modificações de estados de arcos.

O número *total de alterações de estados de arcos* da estrutura é $\delta_p^+ + m_p + \gamma$.

(c) É necessário criar no fim da estrutura os arcos com cauda em nós do mais curto. Como já foi visto em 2b, há $\delta_p^+$ arcos que emergem de nós do caminho mais curto.

Para além dos arcos emergentes em nós do caminho mais curto, há ainda que contabilizar os arcos $(i'',i')$ que também têm que ser criados. Dado que para o último arco do caminho $p$ isto não se verifica, há que criar $m_p - 1$ arcos.

O número *total de arcos que têm que ser criados* no fim da estrutura é $\delta_p^+ + m_p - 1$.

3. Operações no caso $i = i''$:

   (a) Devido ao "Vertex-Splitting", todos os arcos com cabeça em nós do caminho $p$, passam a ter cabeça nos nós $i''$. Isto faz com que as cabeças de todos os arcos incidentes em nós do caminho $p$ tenham que ser alteradas, perfazendo um total de $\delta_p^-$ cabeças modificadas.

   O número *total de modificações de cabeças de arcos* da estrutura é $\delta_p^-$.

   (b) Devido à transformação aplicada à rede, são criados arcos $(i'',i')$. Estes arcos são guardados nos arcos de reserva, bastando para tal ativá-los. O número de arcos ativados é igual ao número de arcos do caminho mais curto, $m_p$.

   Com a inversão do sentido dos arcos do caminho $p$ é necessário inativar os arcos que pertençam a este caminho, o que leva a que tenham que ser inativados $m_p$ arcos.

   Após a inversão do sentido dos arcos é necessário desativar eventuais arcos paralelos[1] que possam aparecer. O número de arcos paralelos que podem aparecer é igual a $\gamma$.

   O número *total de alterações de estados de arcos* da estrutura é $m_p + m_p + \gamma = 2 \times m_p + \gamma$.

   (c) Ao dividir os nós ("Vertex-Splitting") é necessário criar os arcos $(i',(i-1'')$. Para o último arco do caminho $p$ não é necessário criar o arco $(i',(i-1)'')$, pois o último nó não é dividido. Assim, é necessário criar $m_p - 1$ arcos.

   O número *total de arcos que têm que ser criados* no fim da estrutura é $m_p - 1$.

As operações em 2a e 2b são equivalentes a 3a e 3b, se considerarmos que $\delta_p^+$ é em média igual a $\delta_p^-$. O número de arcos a adicionar no final da estrutura, deve ser o mais baixo possível, pelo que a opção $i = i''$ é preferível a $i = i'$, como se pode ver comparando o número de operações em 2c e 3c. Assim, a escolha mais vantajosa é fazer coincidir os nós $i$" com os nós $i$.

[1] Consideram-se arcos paralelos os arcos com cauda e cabeça em nós ($i'$ ou $i''$) que representam o mesmo nó dividido ($i$). Por exemplo, o arco $(j'',i')$ é removido por ser considerado paralelo com o arco $(j',i'')$; de facto, após o coalescer dos nós aqueles dois arcos teriam o mesmo nó cauda ($j$) e o mesmo nó cabeça ($i$).

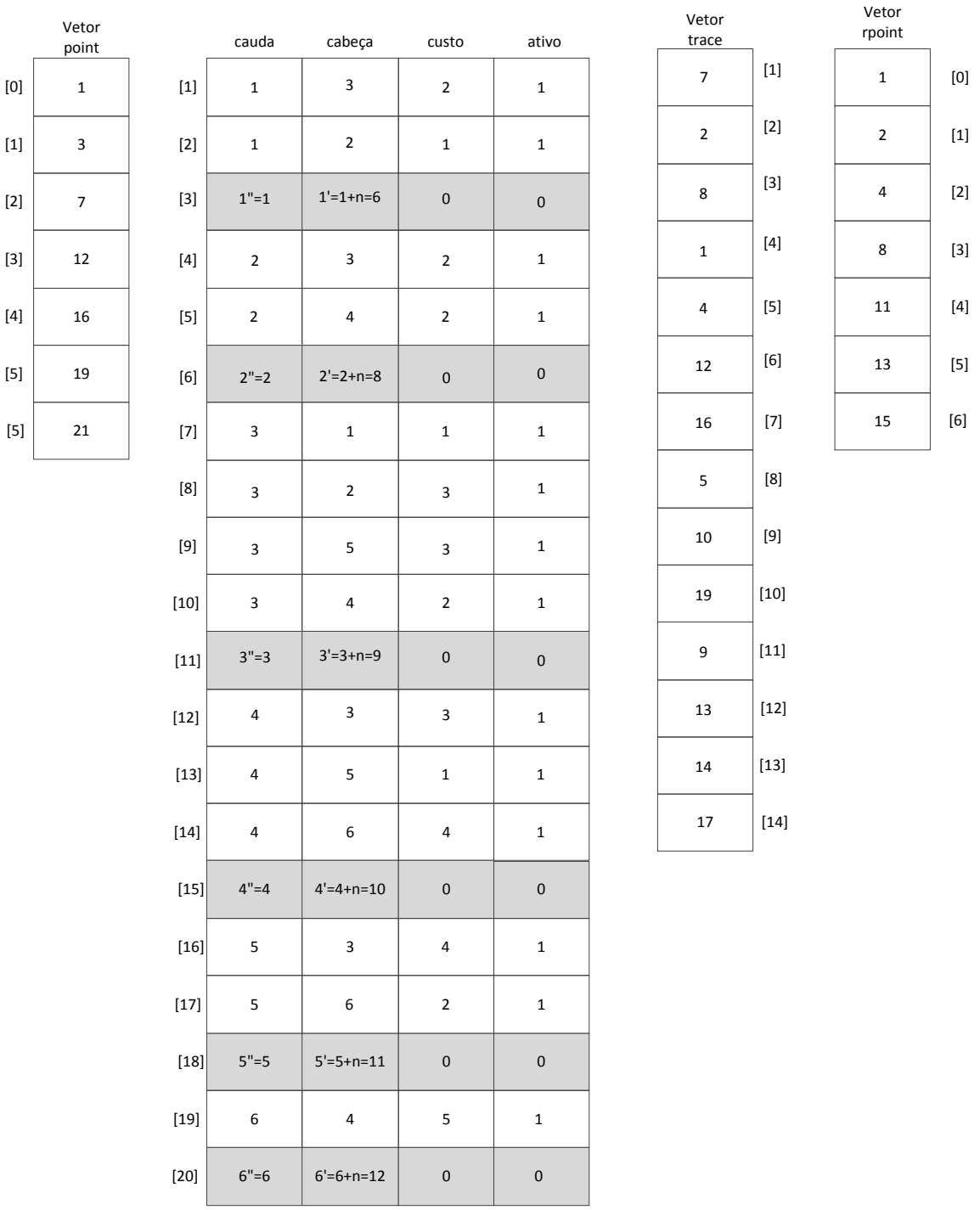
Na implementação realizada, para não ser necessário registar o mapeamento dos nós $i'$ nos nós $i$ que os originaram e vice-versa, optou-se por considerar que $i'$ seria igual a $i+n$. Esta opção torna o código mais eficiente (e simples) requerendo no entanto a duplicação do vetor *point* enquanto se opera na rede transformada. Tendo ainda como objetivo a eficiência do código, optou-se por não sincronizar os vetores *trace* e *rpoint* com a transformação da rede, uma vez que os algoritmos de caminhos mais curto que serão utilizados nessa rede transformada não utilizam a RSF. Isto não cria qualquer problema, porque todas as operações de transformação são seguidas da operação inversa, uma vez cumprida a missão para a qual a mesma foi criada.

A figura A.8 ilustra a estrutura da rede ilustrada na figura A.1, já com espaço reservado para guardar os arcos $(i'',i')$, antes de ser aplicada a transformação. A figura A.9 mostra como fica a estrutura dessa mesma rede após ser aplicada a transformação, admitindo que tem como base o caminho mais curto $\langle 1,2,4,5,6\rangle$, que equivale à rede da figura 3.3. É importante referir que no fim da estrutura são somente acrescentados (por ordem crescente) os arcos $(i',(i-1)'')$ que surjam na rede depois de aplicada a transformação (onde $(i-1)$ representa o nó anterior ao nó $i$ no caminho mais curto). Os blocos que estão preenchidos a cinzento claro são os espaços de reserva que se encontram sempre na estrutura. Os blocos com fundo cinzento escuro indicam os locais onde são colocados os arcos $(i',(i-1)'')$ resultantes da transformação que só ficam em memória até se calcular o segundo caminho mais curto. Depois de se repor a estrutura da rede na sua forma original essa memória é "libertada".

## A.4 Detalhes de implementação

Se esta estrutura fosse apenas utilizada para o cálculo de caminhos disjuntos, o espaço adicional introduzido, ($i''$,$i'$) na figura A.8 não seria estritamente necessário. Nesse caso, era possível reutilizar os arcos do caminho mais curto alterando apenas a sua cabeça. Caso não exista nenhum arco emergente no nó de destino do caminho, isso não causa nenhum problema, pois esse arco nunca faz parte do caminho obtido na rede modificada.

Contudo, pretende-se que esta estrutura permita (no futuro) a obtenção de pares de caminhos disjuntos nas avarias (ou seja, caminhos que podem partilhar arcos protegidos e os nós extremos desses arcos), em que tal como é dito em [5], os arcos pertencentes ao caminho mais curto (no sentido da origem para o destino) não podem ser removidos. Assim sendo, é de facto necessário introduzir este espaço de reserva, quando a estrutura é criada.



| | Vetor point |
|---|---|
| [0] | 1 |
| [1] | 3 |
| [2] | 7 |
| [3] | 12 |
| [4] | 16 |
| [5] | 19 |
| [5] | 21 |

| | cauda | cabeça | custo | ativo |
|---|---|---|---|---|
| [1] | 1 | 3 | 2 | 1 |
| [2] | 1 | 2 | 1 | 1 |
| [3] | 1"=1 | 1'=1+n=6 | 0 | 0 |
| [4] | 2 | 3 | 2 | 1 |
| [5] | 2 | 4 | 2 | 1 |
| [6] | 2"=2 | 2'=2+n=8 | 0 | 0 |
| [7] | 3 | 1 | 1 | 1 |
| [8] | 3 | 2 | 3 | 1 |
| [9] | 3 | 5 | 3 | 1 |
| [10] | 3 | 4 | 2 | 1 |
| [11] | 3"=3 | 3'=3+n=9 | 0 | 0 |
| [12] | 4 | 3 | 3 | 1 |
| [13] | 4 | 5 | 1 | 1 |
| [14] | 4 | 6 | 4 | 1 |
| [15] | 4"=4 | 4'=4+n=10 | 0 | 0 |
| [16] | 5 | 3 | 4 | 1 |
| [17] | 5 | 6 | 2 | 1 |
| [18] | 5"=5 | 5'=5+n=11 | 0 | 0 |
| [19] | 6 | 4 | 5 | 1 |
| [20] | 6"=6 | 6'=6+n=12 | 0 | 0 |

| Vetor trace | |
|---|---|
| 7 | [1] |
| 2 | [2] |
| 8 | [3] |
| 1 | [4] |
| 4 | [5] |
| 12 | [6] |
| 16 | [7] |
| 5 | [8] |
| 10 | [9] |
| 19 | [10] |
| 9 | [11] |
| 13 | [12] |
| 14 | [13] |
| 17 | [14] |

| Vetor rpoint | |
|---|---|
| 1 | [0] |
| 2 | [1] |
| 4 | [2] |
| 8 | [3] |
| 11 | [4] |
| 13 | [5] |
| 15 | [6] |

Figura A.8: Representação em “Hybrid Forward and Reverse Star Form” da rede da figura A.1 (em que apenas se apresenta o valor do bit ativo da máscara associada a cada arco).

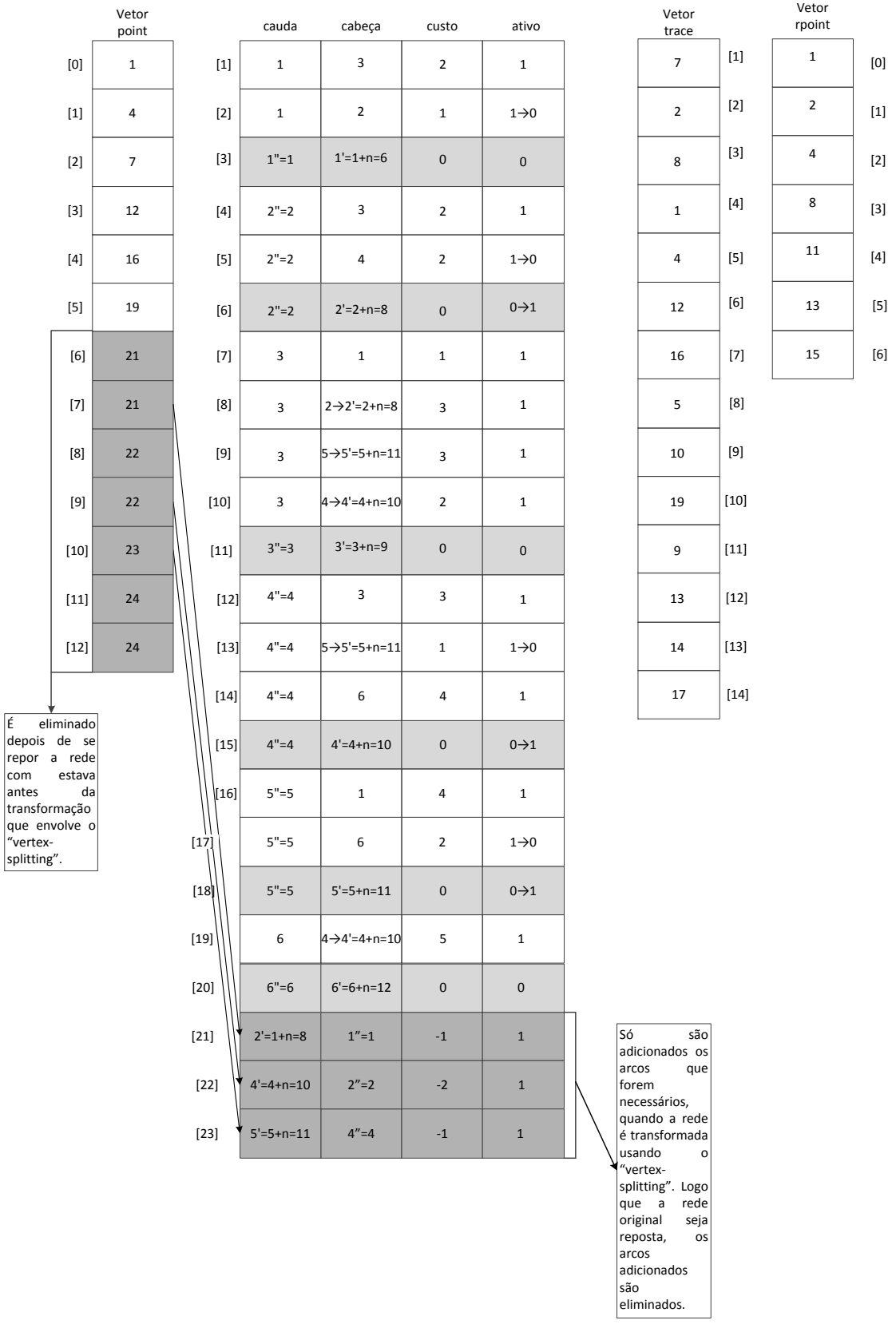



Figura A.9: Representação em "Hybrid Forward and Reverse Star Form" da rede da figura A.1 após ser aplicada a transformação tendo como base o caminho $\langle 1,2,4,5,6 \rangle$. Até a reversão desta transformação, *rpoint* e *trace* não podem ser utilizados.

Caso contrário, seria necessário introduzir posteriormente alguns desses elementos, o que comprometeria a eficiência computacional desses algoritmos.

Os arcos do caminho que são invertidos, terão o seu custo alterado face ao arco que lhe deu origem, dependendo do algoritmo que utiliza a transformação da rede descrita na figura A.3.

# Anexo B

# Algoritmos de cálculo de caminho mais curto

Neste anexo são descritos os algoritmos auxiliares de cálculo de caminho mais curto utilizados neste trabalho.

## B.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra [2] determina o caminho mais curto (caso ele exista) entre dois nós de uma dada rede, representada pelo grafo $G = (V, E)$. Este algoritmo só termina quando é encontrado o caminho mais curto ou quando se verifica que o caminho não existe. No texto que se segue assumiu-se que o nó de origem do caminho é sempre o nó $s$ e o nó de destino é sempre o nó $t$.

O algoritmo de Dijkstra, mostra como se pode determinar o caminho de $s$ para $t$. Neste algoritmo, $d(s) = 0$, dado que a distância de um nó a ele mesmo é zero, e $\infty$ é um número suficientemente grande que seja superior à soma dos custos de todos os arcos da rede.

A primeira parte do algoritmo limita-se a etiquetar todos os nós à exceção do nó $s$ com uma distância inicial infinita. Na realidade não é uma distância infinita, mas uma distância suficientemente grande, tal que seja maior do que $\sum_{(i,j)\in E} l(i,j)$.

Após a execução da primeira parte, o algoritmo entra num ciclo "While". Na linha 9, as distâncias (ou etiquetas temporárias) dos nós são continuamente atualizadas, percorrendo os vizinhos dos nós selecionados (ou permanentemente etiquetados) na linha 7. A pesquisa é limitada aos vizinhos que pertencem ao conjunto de nós não etiquetados. Quando a etiqueta de um nó é atualizada na linha 9, então o seu predecessor também é atualizado.

**Algoritmo Dijkstra:** Algoritmo de Dijkstra que permite calcular um caminho mais curto, $p$, numa rede dirigida, retirado de [2].

**Dados**: Grafo da rede dirigida $G = (V,A)$; nó origem $s$; nó destino $t$; custos associados a cada arco $l(i,j)$, $(i,j) \in A$.

**Resultado**: O caminho de caminhos mais curtos na rede $p$.

1 $d(s) \leftarrow 0$
2 $d(i) \leftarrow l(s,i)$, se $i \in \Gamma_s$ `// Distância provisória dos nós adjacentes a` $s$
3 $d(i) \leftarrow \infty$, se $i \notin \Gamma_s$ e $i \neq s$
4 $S \leftarrow V - \{s\}$ `// O nó` $s$ `já tem etiqueta definitiva`
5 $P(i) \leftarrow s, \forall i \in S$ `// Inicialmente` $s$ `é predecessor de todos os nós`
6 **Enquanto** $i \neq t$ **faz**
7 Procurar um $j \in S$, tal que $j \leftarrow \arg\min_{i \in S} d(i)$
8 $S \leftarrow S \backslash \{j\}$ `// O nó` $j$ `fica etiquetado permanentemente`
`// Atualizar as etiquetas dos nós em` $S$ `adjacentes a` $j$
9 **Para todo** $i \in \Gamma_j \wedge i \notin S$ **faz**
10 **Se** $d(j) + l(i,j) < d(i)$ **então**
11 $d(i) \leftarrow d(j) + l(i,j)$
12 $P(i) \leftarrow j$
13 **fim**
14 **fim**
15 **fim**

Na linha 7 quando um nó é permanentemente etiquetado, o caminho mais curto do nó $s$ para esse nó está encontrado. O algoritmo de Dijkstra procura o primeiro nó que estiver mais perto de $s$, o segundo nó mais perto de $s$ e assim sucessivamente até encontrar o nó de destino $t$. Quando o nó de destino é encontrado, o caminho mais curto foi encontrado e o custo desse caminho é igual a $d(t)$. Percorrendo os predecessores de cada nó, começando no nó $t$ até se encontrar o nó $s$, pode determinar-se a sequência de nós que constituem o caminho mais curto (por ordem inversa).

É possível usar este algoritmo para calcular a árvore de caminhos mais curtos com raiz num dado nó da rede. Para tal basta alterar a condição de paragem do ciclo "While" na linha 6 para $S \neq \emptyset$.

A implementação original do algoritmo de Dijsktra tem uma complexidade $\mathcal{O}(n^2)$, onde $n$ é o número de nós da rede. A implementação realizada utiliza uma *binary heap* [1], permite melhorar o desempenho do algoritmo. A implementação do algoritmo de Dijsktra, usando uma *binary heap* – foi esta a estrutura utilizada[1] no código desenvolvido – tem uma complexidade $\mathcal{O}(m \log n)$, em que $m$ é o número de arcos da rede. Este valor é muito menor do que $\mathcal{O}(n^2)$ se a rede não for densa. De notar que se for utilizada uma *heap de Fibonacci* a complexidade do algoritmo de Dijkstra é $\mathcal{O}(m + n \log n)$.

[1] A *binary heap* utilizada faz parte do código do algoritmo MPS.

## B.2 Algoritmo de Dijkstra modificado

O algoritmo de Dijkstra modificado [2] é uma variante do algoritmo de Dijkstra que permite calcular caminhos em redes com arcos de custo negativo sem ciclos negativos. Este algoritmo é usado em algoritmos de cálculo de um par de caminhos disjuntos (nos nós e nos arcos) com custo aditivo mínimo. Quando a rede só tem arcos de custo não-negativo este algoritmo comporta-se como o algoritmo de Dijkstra. No texto que se segue assumiu-se que o nó de origem do caminho é sempre o nó $s$ e o nó de destino é sempre o nó $t$.

Para determinar o caminho de $s$ para $t$ basta seguir o algoritmo de Dijkstra Modificado, em que tal como no algoritmo de Dijkstra, $d(s) = 0$ e $\infty$ é um número suficientemente grande que seja superior à soma do custo de todos os arcos da rede.

**Algoritmo Dijkstra Modificado:** Algoritmo de Dijkstra Modificado que permite calcular um caminho mais curto, $p$, numa rede dirigida, retirado de [2].

**Dados**: Grafo da rede dirigida $G = (V, A)$; nó origem $s$; nó destino $t$; custos associados a cada arco $l(i, j)$, $(i, j) \in A$.
**Resultado**: O caminho de caminhos mais curtos na rede $p$.

1 $d(s) \leftarrow 0$
2 $d(i) \leftarrow l(s,i)$, se $i \in \Gamma_s$ `// Distância provisória dos nós adjacentes a` $s$
3 $d(i) \leftarrow \infty$, se $i \notin \Gamma_s$ e $i \neq s$
4 $S \leftarrow V - \{s\}$ `// o nó` $s$ `já tem etiqueta definitiva`
`// Inicialmente todos os nós em` $S$ `têm o nó` $s$ `como nó predecessor`
5 $P(i) \leftarrow s, \forall i \in S$
6 **Enquanto** $i \neq t$ **faz**
7 Procurar um $j \in S$, tal que $j \leftarrow \min_{i \in S} d(i)$
8 $S \leftarrow S - \{j\}$ `// o nó` $j$ `fica etiquetado permanentemente`
`// atualizar as etiquetas dos nós em` $S$ `adjacentes a` $j$
9 **Para todo** $i \in \Gamma_j$ **faz**
10 **Se** $d(j) + l(i, j) < d(i)$ **então**
11 $d(i) \leftarrow d(j) + l(i, j)$
12 $P(i) \leftarrow j$
13 $S \leftarrow S \cup \{i\}$
14 **fim**
15 **fim**
16 **fim**

A diferença entre este algoritmo e o algoritmo de Dijkstra reside no ciclo "For" da linha 9 (do algoritmo de Dijkstra) onde um nó permanentemente etiquetado na linha 8 (do algoritmo de Dijkstra) pode voltar a ser novamente inserido no conjunto $S$. Isto faz

com que na iteração seguinte, este nó volte a fazer parte da pesquisa, podendo ser re-etiquetado. Esta modificação faz com que o algoritmo de Dijkstra modificado consiga lidar com redes que têm arcos de custo negativo, desde que não haja ciclos negativos.

## B.3 Algoritmo Breath-First-Search (BFS)

O algoritmo Breath-First-Search (BFS) [2] é uma alternativa ao algoritmo de Dijkstra modificado, que também pode ser aplicado a redes com arcos de custo negativo. No texto que se segue assumiu-se que o nó de origem do caminho é sempre o nó $s$ e o nó de destino é sempre o nó $t$.

Neste algoritmo, para cada iteração, a pesquisa é feita para todos os nós etiquetados na iteração anterior. Do conjunto de nós já etiquetados são excluídos os nós com etiquetas iguais ou superiores à etiqueta do nó $t$. Isto é o mesmo que dizer que só são considerados os nós que tiverem uma distância ao nó $s$, inferior à distância do nó $t$ ao nó $s$. Isto garante que o novo caminho encontrado seja mais curto do que o caminho que já existe de $s$ para $t$.

No algoritmo BFS é apresentado, em pseudo-código, a forma de implementar este algoritmo. Mais uma vez, $d(A) = 0$ e $\infty$ é um número suficientemente grande que seja superior à soma do custo de todos os arcos da rede.

A primeira parte do algoritmo BFS é a inicialização do mesmo, entrando depois num ciclo "For" da linha 6. Nesse ciclo "For" são adicionados ao conjunto $\Gamma^I$ todos os nós que sejam re-etiquetados com uma nova etiqueta menor e a qual seja menor que a etiqueta do nó $t$. A partir do momento em que a etiqueta do nó $t$ deixa de ser infinita, começa a haver um processo de exclusão dos nós que são re-etiquetados com uma nova etiqueta maior que a etiqueta do nó $t$ de $\Gamma^I$. Assim, o conjunto de nós a serem pesquisados nas iterações seguintes irá diminuir e o algoritmo tende a convergir. Quando não existir nenhum nó no conjunto $\Gamma^T$ com etiqueta menor que a etiqueta de $t$, o caminho mais curto para o nó $t$ está encontrado. Dependendo das ligações que existam entre o nó $s$ e o nó $t$ na rede em causa, a convergência deste algoritmo pode ser bastante rápida. Isto faz com que este algoritmo possa ser mais rápido que o algoritmo de Dijkstra modificado.

O algoritmo de BFS tem a particularidade de para dois caminhos com o mesmo custo (soma do custo dos arcos que constituem o caminho) escolher sempre o caminho com menos nós.

**Algoritmo BFS:** Algoritmo BFS que permite calcular um caminho mais curto, $p$, numa rede dirigida, retirado de [2].

**Dados**: Grafo da rede dirigida $G = (V, A)$; nó origem $s$; nó destino $t$; os custos associados a cada arco $l(i, j)$, $(i, j) \in A$.

**Resultado**: O caminho de caminhos mais curtos na rede $p$.

```
1  d(s) ← 0
2  d(i) ← ∞, ∀i ∈ V (i ≠ s)
3  P(i) ← s, ∀i ∈ V            // Inicialmente todos os nós têm o nó s como nó
   predecessor
   // Conjunto de nós a partir dos quais a pesquisa tem início a
      cada iteração
4  Γ^T = {s}
   // Conjunto de nós cuja etiqueta foi atualizada na iteração
      corrente
5  Γ^I = ∅
6  Para todo j ∈ Γ^T faz
7      Para todo i ∈ Γ_j faz
8          Se d(j) + l(j,i) < d(i) ∧ d(j) + l(j,i) < d(t) então
9              d(i) ← d(j) + l(i,j)
10             P(i) ← j
11             Γ^I ← Γ^I ∪ {i}
12         fim
13         Γ^T ← Γ^I \ {t}
14         Se Γ^T = ∅ então
15             Termina
16         fim
17     fim
18 fim
```

## B.4 Observações relativamente aos algoritmos de cálculo de caminhos mais curtos entre dois nós da rede

O algoritmo de Dijkstra não permite o cálculo de caminhos mais curtos numa rede com arcos de custo negativo, enquanto que o algoritmo de Dijkstra modificado e BFS permitem.

No que respeita a redes com arcos de custo não-negativo o algoritmo BFS é em média mais rápido do que o algoritmo de Dijkstra modificado. No entanto, para estas redes, o algoritmo de Dijkstra é o mais rápido [2].

# Anexo C

# Algoritmos Auxiliares

## C.1 Algoritmo de remoção dos arcos comuns a um par caminhos

É aqui apresentado o algoritmo, denominado de Remove-Interlacing, que permite remover os arcos comuns que existem entre um par de caminhos.

**Algoritmo Remove-Interlacing:** Algoritmo "Remove_Interlacing" que remove os arcos comuns entre um par de caminhos.

**Dados**: Um par de caminhos $(p,q)$; o nó de origem $s$ (do par de caminhos $(p,q)$); o nó de destino $t$ (do par de caminhos $(p,q)$).

**Resultado**: Um par de caminhos disjuntos nos nós com origem no nó $s$ e destino no nó $t$: par $(p,q)$ modificado pelo algoritmo.

1 **Enquanto** *o par de caminhos (p,q) não for disjunto nos nós* **faz**

2 $h \leftarrow$ segmento de $q$ que é comum ao par de caminhos $(p,q)$

`//` Se o segmento comum encontrado em $q$, for $h = \langle a,b,\ldots,x,z\rangle$, então o segmento correspondente em $p$ é $h = \langle z,x,\ldots,b,a\rangle$

3 $p = \langle p^1,h,p^2\rangle$; $q = \langle q^1,h,q^2\rangle$

`//` Apagar o segmento correspondente ao segmento $h$ no par de caminhos $(p,q)$ e reconstruir o par de caminhos com os segmentos que restam

4 $p \leftarrow \langle q^1,p^2\rangle$

5 $q \leftarrow \langle p^1,q^2\rangle$

6 **fim**

## C.2 Algoritmo de remoção dos arcos comuns aos $K$ ($\geq$2) caminhos

O algoritmo apresentado, denominado de $K$-Remove-Interlacing, nesta secção serve para remover os arcos comuns a $K$ ($\geq$2) caminhos.

**Algoritmo $K$-Remove-Interlacing:** Algoritmo "$K$-Remove_Interlacing" que permite remover os arcos comuns entre um conjunto de caminhos [2].

**Dados**: Um conjunto $S$ com $u = |S|$ caminhos; um caminho $p$; o nó de origem $s$ (dos caminhos contidos em $S$ e de $p$); o nó de destino $t$ (dos caminhos contidos em $S$ e de $p$).

**Resultado**: O conjunto $S$ com $u+1$ caminhos disjuntos nos nós com origem no nó $s$ e destino no nó $t$.

`// Cria o conjunto dos arcos dirigidos de` $s$ `para` $t$ `que pertencem aos caminhos em` $S$

$D_S \leftarrow$ os arcos dirigidos de $s$ para $t$ que pertencem aos caminhos em $S$

`// Cria o conjunto dos arcos dirigidos de` $s$ `para` $t$ `que pertencem ao caminho em` $p$

$A_S \leftarrow$ os arcos dirigidos de $s$ para $t$ que pertencem ao caminho $p$

$flag \leftarrow false$ `// Assinala que não houve nenhuma remoção`

`// Remove os arcos comuns (`*interlacing*`)`

**Para todo** $(i,j) \in A_S$ **faz**
- **Se** $(j,i) \in D_S$ **então**
  - $D_S \leftarrow D_S \backslash \{(j,i)\}$ `// Remove os arcos comuns`
  - $flag \leftarrow true$ `// Assinala que houve remoção`
- **fim**
- **Caso contrário**
  - $D_S \leftarrow D_S \cup \{(i,j)\}$ `// Insere novo arco`
- **fim**

**fim**

`// Caso não haja arcos comuns (‘‘interlacing’’), os caminhos finais são os caminhos contidos em` $S$ `mais o caminho` $p$

**Se** *flag*=*false* **então**
- $S \leftarrow S \cup \{p\}$

**fim**

`// Caso haja arcos comuns (‘‘interlacing’’), é necessário reconstruir os caminhos com os arcos contidos em` $D_S$ `(já sem os arcos comuns)`

**Caso contrário**
- $S \leftarrow \emptyset$
- Usando os arcos contidos em $D_S$, construir os $u+1$ caminhos disjuntos nos nós e armazená-los em $S$.

**fim**

## C.3 Algoritmo para encontrar um conjunto de SRLG em conflito (Confliting SRLG Set) para um caminho ativo

Nesta secção é apresentado o algoritmo SRLG Exclusion, que permite encontrar um conjunto de SRLG em conflito (Confliting SRLG Set) para um caminho ativo.

É importante notar que a função *empty* devolve "true" se o conjunto $X$ estiver vazio, ou "false" caso contrário, e a função *erase* remove um elemento do conjunto $x$ de SRLG em conflito, devolvendo o seu valor.

**Algoritmo SRLG Exclusion:** Algoritmo "SRLG_exclusion" que permite calcular um conjunto de SRLG da rede que estejam em conflito com o conjunto de SRLG de um caminho ativo $p_c$ [4].

**Dados**: Grafo da rede dirigida $G = (V, A)$; nó origem $s$; nó destino $t$; o custo $l(i, j)$, e os SRLG, $R(i, j)$, associados a cada arco $(i, j) \in A$; caminho ativo $p$; conjunto de inclusão, $I$, do problema corrente.

**Resultado**: O conjunto de SRLG em conflito para o caminho ativo $p$

```
1  T ← ∅                          // O conjunto de SRLG está inicialmente vazio
2  X ← conjunto de SRLG que afetam o caminho ativo p
3  X ← X\I                        // Remove de X os SRLG contidos em I
4  stop ← false
5  Enquanto ¬ empty (X) ∧ ¬stop faz
6      g ←erase (X)
7      T ← T ∪ {g}
8      Remover da rede, G, todos os arcos pertençam ao SRLG g
9      p_g ← Dijkstra (s,t)
10     Se C_{p_g} ≠ ∞ então
11         X_{p_g} ← conjunto de SRLG que afetam o caminho ativo p_g
12         X ← X ∩ X_{p_g}        // Para reduzir o tamanho de T
13     fim
14     Caso contrário
15         stop ← true            // T já está calculado
16     fim
17 fim
```

## C.4 "Modified Bhandari's Heuristic" (MBH)

Nesta secção é apresentada a *Modified Suurballe's Heuristic* (MSH), que é utilizada pelo algoritmo de CoSE-MS e pelas suas outras duas variantes.

**Algoritmo MBH:** Algoritmo MBH que permite obter um par de caminhos disjuntos nos nós e nos SRLG [4].

**Dados**: Grafo da rede dirigida $G = (V,A)$; nó origem $s$; nó destino $t$; o custo $l(i,j)$, e os SRLG, $R(i,j)$, associados a cada arco $(i,j) \in A$; caminho semente $p_c$.

**Resultado**: O par de caminhos disjuntos nos nós e nos SRLG $(p,q)$, obtido usando o caminho semente $p_c$.

1 Remover temporariamente os arcos $(i,j)$ da rede $G, \forall (i,j) \in p_c$
2 $M \leftarrow \sum_{(i,j)\in A} l(i,j)$
3 **Para cada** *arco* $(i,j)$ *da rede tal que* $R(i,j) \cap R_{p_c} \neq \emptyset$ **faz**
  `// O arco` $(i,j)$ `pertence a pelo menos um dos SRLG que afetam o caminho` $p_c$
4  $l(i,j) \leftarrow l(i,j) + M$
5 **fim**
6 Aplicar à rede uma transformação semelhante à referida em 3.3 com base no caminho $p_c$, introduzindo na rede arcos de custo negativo
  `// Dado que a rede tem arcos com custo negativo, usa o algoritmo de Dijkstra Modificado - ver anexo B.2`
7 $p' \leftarrow$`Dijkstra_Modificado` $(s,t)$
8 **Se** *existe* $p'$ **então**
9  $(p,q) \leftarrow$`Remove_Interlacing` $(p_c,p')$
10  **Se** $R_p \cap R_q \neq \emptyset$  `// Avalia se` $(p,q)$ `são disjuntos nos SRLG`
11  **então**
12  $(p,q) \leftarrow (\emptyset,\emptyset)$  `// Não há solução`
13  **fim**
14 **fim**
15 **Caso contrário**
16  $(p,q) \leftarrow (\emptyset,\emptyset)$  `// Não há solução`
17 **fim**

# Anexo D

# Novas versões de MSH e MBH

As novas versões de MSH e MBH aqui introduzidas diferem das originais no seguinte:

- Se o caminho semente der origem a um par de caminhos, este é sempre devolvido.
- Se o caminho semente der origem a um par de caminhos, é devolvido o conjunto de SRLG comuns a esse dois caminhos: $X = R_p \cap R_q$. Se os caminhos obtidos forem disjuntos nos SRLG ou caso o par não exista, $X = \emptyset$.

Tal como nas versões originais, o custo do par obtido será infinito se $X \neq \emptyset$.

**Algoritmo MSH–v2:** Algoritmo MSH que procura obter um par de caminhos disjuntos nos nós e nos SRLG, e devolve o conjunto de SRLG comuns ao par obtido em caso de insucesso.

**Dados**: Grafo da rede dirigida $G = (V,A)$; nó origem $s$; nó destino $t$; o custo $l(i,j)$, e os SRLG, $R(i,j)$, associados a cada arco $(i,j) \in A$; caminho semente $p_c$.

**Resultado**: Par de caminhos disjuntos nos nós $(p,q)$, obtido usando o caminho semente $p_c$; custo do par de caminhos, $C_{(p,q)}$ ($\infty$ se $R_p \cap R_q \neq \emptyset$); conjunto dos SRLG comuns a $p$ e $q$, $X$.

1 Remover temporariamente os arcos $(i,j)$ da rede $G, \forall (i,j) \in p_c$
2 $M \leftarrow \sum_{(i,j)\in A} l(i,j)$
3 **Para cada** *arco* $(i,j)$ *da rede tal que* $R(i,j) \cap R_{p_c} \neq \emptyset$ **faz**
  `// O arco (i,j) pertence a pelo menos um dos SRLG que afetam o caminho` $p_c$
4  $l(i,j) \leftarrow l(i,j) + M$
5 **fim**
6 $(p,q) \leftarrow (\emptyset,\emptyset)$
7 $C_{(p,q)} \leftarrow \infty$ `// Até prova em contrário não há solução`
8 $X \leftarrow \emptyset$
9 Aplicar à rede uma transformação semelhante à referida na secção 3.3 com base no caminho $p_c$, em que os arcos simétricos (aos do caminho $p_c$) introduzidos têm custo nulo.
  `// Dado que a rede não tem arcos com custo negativo, pode ser usado o algoritmo de Dijkstra (ver anexo B.1)`
10 $p' =$`Dijkstra` $(s,t)$
11 **Se** *existe* $p'$ **então**
12  $(p,q) \leftarrow$`Remove_Interlacing` $(p_c,p')$
13  $X \leftarrow R_p \cap R_q$
14  **Se** $X = \emptyset \wedge (p,q) \neq (\emptyset,\emptyset)$ **então**
15  $C_{(p,q)} \leftarrow$custo do par de caminhos $(p,q)$
16  **fim**
17  **Caso contrário**
18  $C_{(p,q)} \leftarrow \infty$
19  **fim**
20 **fim**

**Algoritmo MBH–v2:** Algoritmo MBH que procura obter um par de caminhos disjuntos nos nós e nos SRLG, e devolve o conjunto de SRLG comuns ao par obtido em caso de insucesso.

**Dados**: Grafo da rede dirigida $G = (V, A)$; nó origem $s$; nó destino $t$; o custo $l(i, j)$, e os SRLG, $R(i, j)$, associados a cada arco $(i, j) \in A$; caminho semente $p_c$.

**Resultado**: Par de caminhos disjuntos nos nós $(p, q)$, obtido usando o caminho semente $p_c$; custo do par de caminhos, $C_{(p,q)}$ ($\infty$ se $R_p \cap R_q \neq \emptyset$); conjunto dos SRLG comuns a $p$ e $q$, $X$.

1 Remover temporariamente os arcos $(i, j)$ da rede $G, \forall (i, j) \in p_c$
2 $M \leftarrow \sum_{(i,j)\in A} l(i, j)$
3 **Para cada** *arco* $(i, j)$ *da rede tal que* $R(i, j) \cap R_{p_c} \neq \emptyset$ **faz**
  `// O arco` $(i, j)$ `pertence a pelo menos um dos SRLG que afetam o caminho` $p_c$
4  $l(i, j) \leftarrow l(i, j) + M$
5 **fim**
6 $(p, q) \leftarrow (\emptyset, \emptyset)$
7 $C_{(p,q)} \leftarrow \infty$ `// Até prova em contrário não há solução`
8 $X \leftarrow \emptyset$
9 Aplicar à rede uma transformação semelhante à referida na secção 3.3 com base no caminho $p_c$, introduzindo na rede arcos de custo negativo
  `// Dado que a rede tem arcos com custo negativo, usa o algoritmo de Dijkstra Modificado (ver anexo B.2)`
10 $p' \leftarrow$`Dijkstra_Modificado` $(s,t)$
11 **Se** *existe* $p'$ **então**
12  $(p, q) \leftarrow$`Remove_Interlacing` $(p_c,p')$
13  $X \leftarrow R_p \cap R_q$
14  **Se** $X = \emptyset \wedge (p, q) \neq (\emptyset, \emptyset)$ **então**
15  $C_{(p,q)} \leftarrow C_p + C_q$
16  **fim**
17  **Caso contrário**
18  $C_{(p,q)} \leftarrow \infty$
19  **fim**
20 **fim**

# Anexo E

# Exemplos de funcionamento de algoritmos que consideram SRLG

Neste anexo são apresentados exemplos de funcionamentos dos algoritmos de cálculo de pares de caminhos disjuntos nos nós e nos SRLG, IMSH e CoSE-MS.

## E.1 Exemplo de funcionamento do algoritmo IMSH

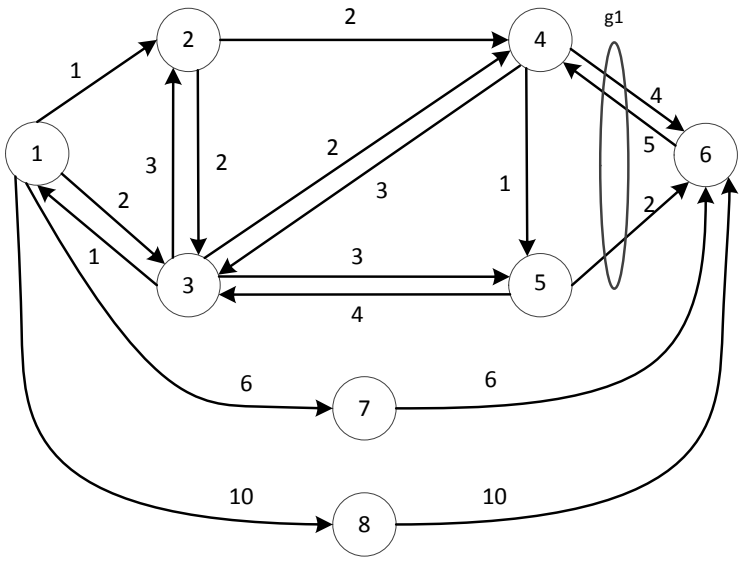



Figura E.1: Rede usada no exemplo de funcionamento do algoritmo IMSH.

Nesta sub-secção é apresentado um exemplo do funcionamento do algoritmo IMSH para a rede que se apresenta na figura E.1, em que cada arco define um SRLG e para além disso os arcos $(4,6)$, $(5,6)$ e $(6,4)$ pertencem ao SRLG $g_1$. Pretende-se calcular um par

de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo entre o nó 1 e o nó 6, com um número máximo de iterações igual a 100.

No que se segue, $p_k$ é o caminho semente na iteração $k$ (com custo $C_{p_k}$), $(p,q)$ indica o melhor par obtido até ao momento com custo $C_{melhor\_par}$. Para esta rede o algoritmo IMSH realiza as seguintes iterações:

**Inicialização**

$(p,q) \leftarrow (\emptyset,\emptyset)$

$C_{(melhor\_par)} = C_{(p,q)} = C_p + C_q = \infty$

**Iteração 1**

$p_1 \leftarrow \langle 1,2,4,5,6\rangle$ $(C_{p_1} = 6)$

$(p'_1, p''_1) \leftarrow \text{MSH}(G,1,6,l,R,p_1)$

$(p'_1 = \langle 1,2,4,5,6\rangle, p''_1 = \langle 1,7,6\rangle)$, $C_{(p'_1,p''_1)} = C_{p'_1} + C_{p''_1} = 6+12 = 18$

Como o par $(p'_1,p''_1)$ é disjunto nos SRLG e $C_{(p'_1,p''_1)} < C_{melhor\_par} \Leftrightarrow 18 < \infty$, então $(p,q) \leftarrow (p'_1,p''_1)$ e $C_{melhor\_par} = C_{(p'_1,p''_1)} = 18$

$C_{melhor\_par} - C_{p_1} > C_{p_1} \Leftrightarrow 18-6 > 6 \Leftrightarrow 12 > 6$, então a solução ótima ainda não foi encontrada.

**Iteração 2**

$p_2 = \langle 1,3,5,6\rangle$ $(C_{p_2} = 7)$

$(p'_2, p''_2) \leftarrow \text{MSH}(G,1,6,l,R,p_2)$

$(p'_2 = \langle 1,3,5,6\rangle, p''_2 = \langle 1,7,6\rangle)$, $C_{(p'_2,p''_2)} = C_{p'_2} + C_{p''_2} = 7+12 = 19$

O par $(p'_2,p''_2)$ é disjunto nos SRLG, mas $C_{(p'_2,p''_2)} > C_{melhor\_par} \Leftrightarrow 19 > 18$, então $(p,q)$ não é atualizado.

$C_{melhor\_par} - C_{p_1} > C_{p_2} \Leftrightarrow 18-6 > 7 \Leftrightarrow 12 > 7$, então a solução ótima ainda não foi encontrada.

$(\cdots)$

Como se escolheu o número máximo de iterações igual a 100, o algoritmo vai percorrer todos os caminhos que existem na rede, do nó 1 para o nó 6, até que encontre uma solução ótima ou até serem examinados todos os caminhos semente da rede[1]. Para nenhum desses caminhos semente irá ser obtido um par de caminhos com custo melhor do que o que foi obtido na iteração 1.

**Iteração 12**

$p_{12} = \langle 1,7,6\rangle$ $(C_{p_{12}} = 12)$

$(p'_{12}, p''_{12}) \leftarrow \text{MSH}(G,1,6,l,R,p_{12})$

[1] Para a rede da figura E.1 existem 13 caminhos possíveis.

$(p'_{12} = \langle 1,7,6 \rangle, p''_{12} = \langle 1,2,4,5,6 \rangle)$, $C_{(p'_{12},p''_{12})} = C_{p'_{12}} + C_{p''_{12}} = 12+6 = 18$
O par $(p'_{12},p''_{12})$ é disjunto nos SRLG, mas $C_{(p'_{12},p''_{12})} = C_{melhor\_par} \Leftrightarrow 18 = 18$, então $(p,q)$ não é atualizado.
$C_{melhor\_par} - C_{p_1} = C_{p_{12}} \Leftrightarrow 18-6 = 12 \Leftrightarrow 12 = 12$, então a solução ótima já foi encontrada e o algoritmo termina.

Para o caso em que o número máximo de iterações é igual a 100, a condição de ótimalidade é verificada na iteração 12 (sem se gerar todos os caminhos). Caso se altere o número máximo de iterações para 5, então o algoritmo pára porque foi atingido o número máximo de iterações sem se ter verificado a condição de otimalidade. Nesse caso a solução é considerada sub-ótima pelo IMSH.

## E.2 Exemplo de funcionamento do algoritmo CoSE-MS

Nesta sub-secção é apresentado um exemplo do funcionamento do algoritmo CoSE-MS para a rede que se apresenta na figura E.2. Pretende-se calcular um par de caminhos disjuntos nos nós e nos SRLG de custo aditivo mínimo entre o nó 1 e o nó 5.

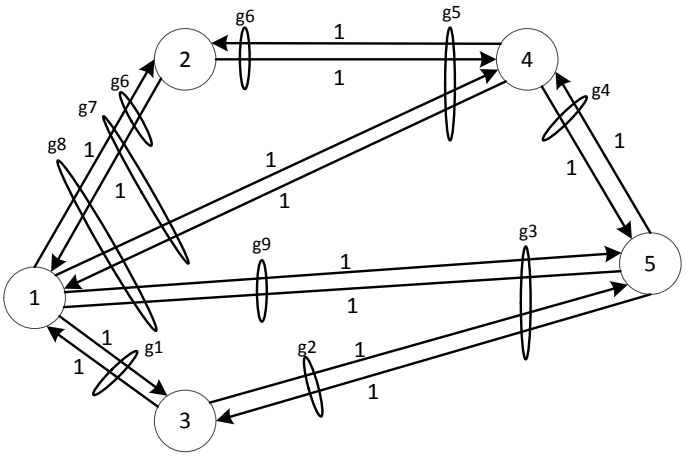



Figura E.2: Rede usada para exemplificar o algoritmo de CoSE-MS, onde $M = 14$.

O algoritmo começa por criar e colocar na pilha de problemas o problema vazio $(P(\emptyset,\emptyset,\emptyset))$. A partir deste momento, o algoritmo começa a resolver os problemas que estão na pilha até esta estar vazia. A ordem e a forma como esses problemas são resolvidos é a seguinte:

$P_0(\emptyset,\emptyset,\emptyset)$

Quando o problema $P_0$ é resolvido obtém-se o caminho semente $p_c = \langle 1,5 \rangle$. Como se pode ver pela figura E.3, usando a MBH com o caminho semente $p_c$, obtém-se o

par de caminhos ($p' = \langle 1,5 \rangle$,$q' = \langle 1,3,5 \rangle$). Os arcos $(1,5)$ e $(3,5)$ estão associados ao SRLG $g_3$, logo o par de caminhos obtidos não é disjuntos nos SRLG. Assim sendo, é criado um conjunto de SRLG em conflito, $T = \{g_3, g_8\}$ usando o algoritmo referido na sub-secção 4.3.4[2]. Como tal, são criados e colocados (por esta ordem) na pilha de problemas os problemas $(\emptyset, \{g_3\}, \emptyset)$ e $(\{g_3\}, \{g_8\}, \emptyset)$.

$P_1(\{g_3\}, \{g_8\}, \emptyset)$

Segue-se a resolução do problema, $(\{g_3\}, \{g_8\}, \emptyset)$, que se segue no topo da pilha. Na figura E.5 é ilustrada a rede da figura E.2, depois de removidos os arcos associados aos SRLG de $E_1$. O SRLG $g_8$ está a tracejado, indicando que pertence a $E_1$. Os arcos removidos encontram-se a ponteado. Nesta rede, o caminho semente obtido é $p_c = \langle 1,3,5 \rangle$. Usando este caminho semente a MSH determina o par de caminhos ($p' = \langle 1,3,5 \rangle$,$q' = \langle 1,4,5 \rangle$), como se pode ver na figura E.6. Dado que este par de caminhos é disjunto nos SRLG e ainda não foi encontrado nenhum par, este par é guardado em $(p,q)$, juntamente com o seu custo ($C_{(p,q)} = 4$).

$P_2(\emptyset, \{g_3\}, \emptyset)$

Por fim falta resolver o último problema que está na pilha de problemas. Esse problema conduz ao caminho semente $p_c = \langle 1,4,5 \rangle$. Usando o MSH com o caminho semente anteriormente determinado, é possível obter o par de caminhos disjuntos nos nós e nos SRLG ($p' = \langle 1,4,5 \rangle$,$q' = \langle 1,3,5 \rangle$). Dado que o custo deste par ($C_{(p,q)} = 4$) não é inferior ao custo do melhor par obtido até ao momento ($C_{melhor\_par} = 4$), o par ($p' = \langle 1,4,5 \rangle$,$q' = \langle 1,3,5 \rangle$) é descartado.

Após a resolução do problema $P_1$ a pilha de problemas fica vazia e o CoSE-MS termina, devolvendo o par de caminhos ($p = \langle 1,2,5 \rangle$,$q = \langle 1,4,5 \rangle$) com custo 4.

[2] A figura E.4 permite ver que quando são removidos os SRLG $g_3$ e $g_8$ não há nenhum caminho do nó 1 para o nó 5.

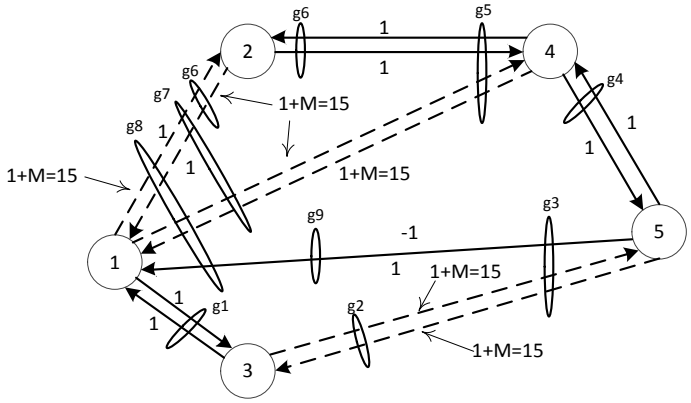



Figura E.3: Topologia da rede ilustrada na figura E.2, depois de aplicada a transformação pelo MBH, usando como caminho semente ($p_c = \langle 1,5 \rangle$), para o problema $P_0(\emptyset,\emptyset,\emptyset)$. Os arcos a tracejado são os arcos afetados pelos SRLG do caminho semente.

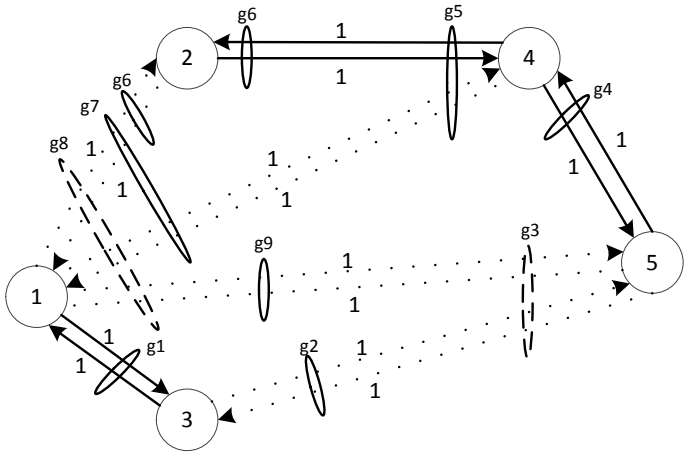



Figura E.4: Exemplo de funcionamento do algoritmo SRLG Exclusion na rede da figura E.2. Os arcos a ponteado são os arcos que foram removidos da rede.

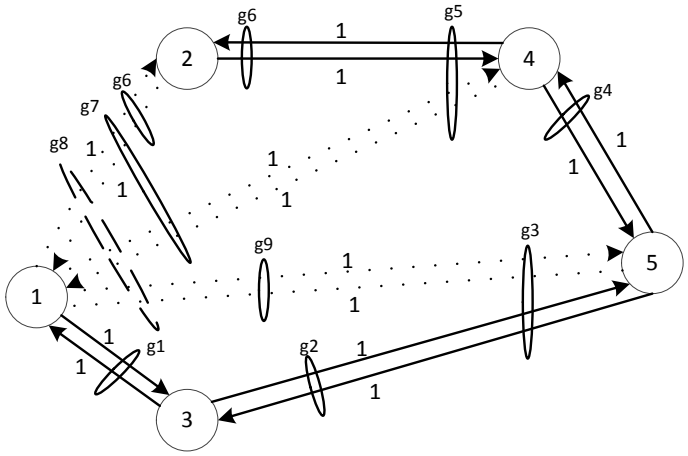



Figura E.5: Topologia da rede da figura E.2 para o problema $P_1 = (\{g_3\},\{g_8\},\emptyset)$ – relembre-se que os arcos a ponteado são afetados por $g_8$. Os arcos a tracejado são os arcos afetados pelos SRLG do caminho semente.

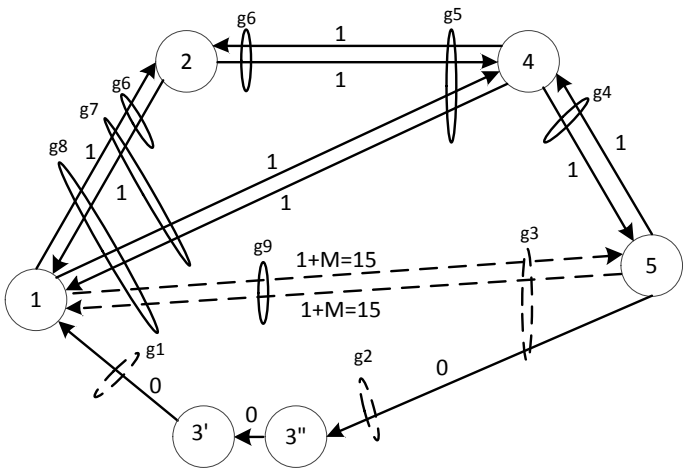



Figura E.6: Topologia da rede ilustrada na figura E.2, depois da transformação pela MSH, usando o caminho semente $p_c = \langle 1,3,5 \rangle$, para o problema $P_1 = (\{g_3\},\{g_8\},\emptyset)$.

# Anexo F

# Biblioteca partilhada (.so)

Hoje em dia verifica-se uma ampla utilização de bibliotecas partilhadas (ficheiros .so em UNIX e .DLL em Windows). Desta forma evita-se o uso de cada um dos ficheiros *objeto* no momento de fazer a *ligação* (*linking*), o que permite que o *software* desenvolvido seja partilhado por várias aplicações. Bibliotecas de grande dimensão não precisam de fazer parte integrante de um executável, uma vez que apenas são carregadas quando este executa. Estas bibliotecas permitem igualmente criar uma API (*Application Programming Interface*) que acedem aos recursos disponibilizados numa biblioteca.

Neste contexto, as rotinas para cálculo de caminhos disjuntos, foram implementadas recorrendo a uma biblioteca dinâmica.

Para criar uma biblioteca partilhada seguem-se os seguintes passos:

1. Criar os ficheiros com o código objeto (.o);

2. Criar a biblioteca partilhada usando os ficheiros com código objeto (.o) anteriormente criados;

3. O passo seguinte é colocar a biblioteca partilhada (.so), juntamente com os devidos *links* simbólicos, na pasta do sistema operativo apropriada para esse efeito.

   Em sistemas UNIX, para além da biblioteca partilhada (.so), é necessário criar dois outros *links* simbólicos. O *link simbólico* com extensão `.so` permite que seja feita a compilação na altura da criação do ficheiro executável que usa a biblioteca dinâmica (.so). O *link simbólico* com extensão `.so.1` permite a ligação do ficheiro executável com a biblioteca partilhada (.so) no momento da execução do programa.

4. Compilar o ficheiro executável, ligando-o com a biblioteca criada no passo 2.

Para mais informação acerca da criação de uma biblioteca partilhada, aconselha-se o leitor a consultar o sítio http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html.

# Anexo G

# Formalização do problema de cálculo de um par de caminhos disjuntos nos nós e nos SRLG, de custo aditivo mínimo

Neste anexo, é apresentada a formalização em PLI para o problema de cálculo de um par de caminhos disjuntos nos nós e nos SRLG adaptada de [7]. Nesta formalização é considerada uma rede dirigida, $G(V,A)$, onde para cada caminho o nó de origem será designado por $s$ e o nó de destino por $t$.

Primeiro é introduzida a notação adicional que permite formalizar o problema em PLI.

- $\delta(i)^+$ : conjunto dos arcos emergentes do nó $i \in V$.
- $\delta(i)^-$ : conjunto dos arcos incidente do nó $i \in V$.
- $h_{g,(i,j)}$, com $g \in R'$ e $(i,j) \in A$, indica se o SRLG $g$ contém o arco $(i,j)$

$$h_{g,(i,j)} = \begin{cases} 1 & \text{se } g \in R(i,j), \\ 0 & \text{caso contrário;} \end{cases} \tag{G.1}$$

- $x_{(i,j),k}$ é a variável binária de decisão do arco $(i,j) \in A$ associado ao caminho $p_k$ ($k = 1,2$), onde,

$$x_{(i,j),k} = \begin{cases} 1 & \text{se o arco } (i,j) \text{ pertence ao caminho } p_k, \\ 0 & \text{caso contrário;} \end{cases} \tag{G.2}$$

- $z_{g,k}$ é a variável binária de decisão dos SRLG que afetam o caminho $p_k$ ($k = 1,2$), onde,

$$z_{g,k} = \begin{cases} 1 & \text{se } g \text{ está associado ao caminho } p_k, \text{ ou seja se } g \in R_{p_k}, \\ 0 & \text{caso contrário;} \end{cases} \tag{G.3}$$

É agora introduzida a formalização do problema de cálculo de um par de caminhos disjuntos nos nós e nos SRLG em PLI.

$$\text{min} \quad \sum_{(i,j)\in A} l(i,j)(x_{(i,j),1} + x_{(i,j),2}) \tag{G.4}$$

$$\text{s. a:} \quad \sum_{(i,j)\in\delta(i)^+} x_{(i,j),k} - \sum_{(j,i)\in\delta(i)^-} x_{(j,i),k} = \begin{cases} 1 & : \quad i = s, \\ -1 & : \quad i = t, \\ 0 & : \quad i \in V\backslash\{s,t\} \end{cases}, \quad i \in V, k = 1,2 \tag{G.5}$$

$$\sum_{(i,j)\in A} h_{g,(i,j)} x_{(i,j),k} \leq |A| z_{g,k}, \quad g \in R', k = 1,2 \tag{G.6}$$

$$z_{g,1} + z_{g,2} \leq 1, \quad g \in R' \tag{G.7}$$

$$\sum_{k\in\{1,2\}} \sum_{(i,j)\in\delta(i)^+} x_{(i,j),k} \leq 1, \quad i \in V\backslash\{s\}, \tag{G.8}$$

$$\sum_{k\in\{1,2\}} \sum_{(j,i)\in\delta(i)^-} x_{(j,i),k} \leq 1, \quad i \in V\backslash\{t\}, \tag{G.9}$$

$$x \text{ e } z \text{ são variáveis binárias de decisão} \tag{G.10}$$

- A restrição G.5 garante que os arcos $(i,j)$ selecionadas com base em $x_{(i,j),k}$, constituem o caminho $p_k$ ($k = 1,2$) de $s$ para $t$.

- A restrição G.6 implica que se $g$ está contido no caminho $p_k$, então algum arco do conjunto desse SRLG tem que ser usado. Se um arco está associada a vários SRLG, então só pode ser escolhido para o caminho $p_k$, se e só se os elementos contidos em $R'$ estiverem contidos em $p_k$. O coeficiente $|A|$ é usado, pois $p_k$ pode conter vários arcos que estejam associados a um SRLG.

- A restrição G.7 garante que nenhum dos elementos do conjunto $R'$ está contido em ambos os caminhos. Assumindo que cada arco pertence a pelo menos um SRLG ($\forall(i,j) \in A : R(i,j) \neq \emptyset$), a disjunção nos SRLG garante a disjunção nos arcos.

- As restrições G.8 e G.9 garantem que os caminhos são disjuntos nos nós.

# Bibliografia


[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms and applications*. Prentice Hall, 1993.

[2] R. Bhandari. *Survivable Networks, Algorithms for Diverse Routing*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1999.

[3] A. Farrel, J.-P. Vasseur, and J. Ash. A path computation element (PCE)-based architecture. RFC 4655, Agosto 2006.

[4] T. Gomes, C. Simões, and L. Fernandes. Resilient routing in optical networks using SRLG-disjoint path pairs of min-sum cost. *Telecommunication Systems Journal*, pág. 1–13. Em impressão.

[5] T. M. Gomes, J. Silva, J. Craveirinha, and C. Simões. SRLG failure-disjoint path pair of min-sum cost in GMPLS networks. In *8th International Workshop on the Design of Reliable Communication Networks (DRCN 2011)*, pág. 196–203, Krakow, Poland, Outubro 2011.

[6] P.-H. Ho and H. T. Mouftah. Shared protection in mesh WDM networks. *IEEE Communications Magazine*, 42(1):70–76, Janeiro 2004.

[7] J. Q. Hu. Diverse routing in optical mesh networks. *IEEE Transactions on Communications*, 51(3):489–494, Março 2003.

[8] P. Laborczi, J. Tapolcai, P.-H. Ho, T. Cinkler, A. Recski, and H. T. Mouftah. Algorithms for asymmetrically weighted pair of disjoint paths in survivable networks. In T. Cinkler, editor, *Proceedings of Design of Reliable Communication Networks (DCRN 2001)*, pág. 220–227, Budapest, Hungary, Outubro 7-10 2001.

[9] M. J. Rostami, S. Khorsandi, and A. A. Khodaparast. CoSE: A SRLG-disjoint routing algorithm. In *Proceedings of the Fourth European Conference on Universal Multiservice Networks (ECUMN'07)*, Toulouse, France, 2007.

[10] E. Martins, M. Pascoal, and J. Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10(3):247–263, 1999.

[11] E. Q. V. Martins, M. M. B. Pascoal, and J. L. Santos. A new algorithm for ranking loopless paths, CISUC, Maio 1997. Existe uma versão posterior: "An algorithm for ranking loopless paths", Technical Report 99/007, CISUC, (1999).

[12] J. W. Suurballe. Disjoint paths in networks. *Networks*, 4:125–145, 1974.

[13] A. Todimala and B. Ramamurthy. IMSH: An iterative heuristic for SRLG diverse routing in WDM mesh networks. In *13th International Conference on Computer Communications and Networks, ICCCN'2004*, pág. 199–204, Outubro 2004.

[14] A. Todimala and B. Ramamurthy. A heuristic with bounded guarantee to compute diverse paths under shared protection in WDM mesh networks. In *IEEE Globlecom 2005, November 28 - December 2, 2005, St. Louis, MO, USA*, pág. 1915–1919, 2005.

[15] J.-P. Vasseur, M. Pickavet, and P. Demeester. *Network Recovery – Protection and Restoration of optical, SONET-SDH, IP, and MPLS*. Elsevier, 2004.

[16] D. Xu, Y. Chen, Y. Xiong, C. Qiao, and X. He. On finding disjoint paths in single and dual link cost networks. In *IEEE INFOCOM 2004*, Hong Kong, 2004.

[17] D. Xu, Y. Xiong, and C. Qiao. Protection with multi-segments (PROMISE) in netorks with shared risk link groups (SRLG). *IEEE/ACM Transactions on Networking*, 11:248–258, Abril 2003.

[18] D. Xu, Y. Xiong, C. Qiao, and G. Li. Trap avoidance and protection schemes in networks with shared risk link groups. *Journal of Lightwave Technology*, 21(11):2683–2693, Novembro 2003.