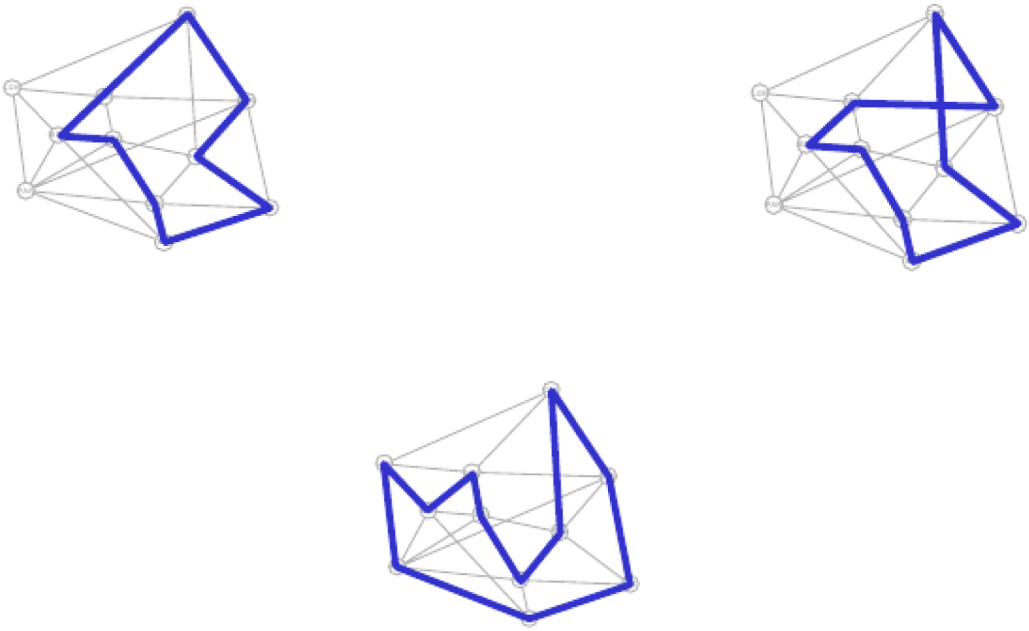

David José Calado Cúrdia Gonçalves

DETERMINAÇÃO DE P-CYCLES

Dissertação de Mestrado Integrado em Engenharia Electrotécnica e de Computadores, especialidade Telecomunicações, orientada pela Professora Teresa Martinez dos Santos Gomes e pela Professora Lúcia Maria dos Reis Albuquerque Martins e, apresentada no Departamento de Engenharia Electrotécnica e de Computadores

Setembro de 2015

· U C ·

UNIVERSIDADE DE COIMBRA

Faculdade de Ciências e Tecnologia da
Universidade de Coimbra

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Determinação de *p-cycles*

Membros do Juri:
Presidente: Maria do Carmo Raposo de Medeiros
Orientadora: Teresa Martinez dos Santos Gomes
Co-orientadora: Lúcia Maria dos Reis Albuquerque Martins
Vogais: Rita Cristina Girão Coelho da Silva e Teresa Martinez dos Santos Gomes

David José Calado Cúrdia Gonçalves

Setembro 2015

Agradecimentos

Em primeiro lugar, como não podia deixar de ser, tenho de agradecer à professora Teresa Martinez dos Santos Gomes e à professora Lúcia Maria dos Reis Albuquerque Martins, por toda a ajuda e disponibilidade que demonstraram durante todo o trabalho.

Quero agradecer também a todos os meus colegas e amigos, pelos momentos vividos e, que vão para sempre ser recordados, ao longo de todo o meu percurso académico.

Por último, quero agradecer aos meus pais por terem garantido todas as condições para que eu conseguisse terminar o meu curso e, pelo apoio e incentivo que sempre me deram, mesmo nos momentos mais difíceis do meu percurso académico.

A todos, um muito obrigado!

Resumo

Estando a sociedade cada vez mais dependente das redes de telecomunicações, é fundamental que estas sejam adequadamente protegidas contra falhas que possam ocorrer. Uma maneira eficiente de se proteger uma rede é usando *p-cycles*. Um *p-cycle* é um ciclo pré-estabelecido que tira partido da capacidade de reserva na rede. Sendo uma estrutura do tipo anel, o *p-cycle* garante um caminho de protecção para cada arco que pertença ao ciclo e dois caminhos de protecção para cada arco *straddling* - arco cujos nós extremos fazem parte do ciclo mas cujo arco não faz parte do ciclo.

Nesta dissertação estudaram-se os *p-cycles* e foram implementados três modelos de programação linear inteira para a determinação de *p-cycles*. O primeiro modelo tem como objectivo minimizar a capacidade de reserva necessária para a formação dos *p-cycles*. O segundo modelo tem como objectivo a minimização do custo da capacidade total necessária para o encaminhamento do tráfego e para garantir protecção. E o terceiro modelo tem como objectivo a maximização da capacidade de trabalho total que é protegida pelos *p-cycles*. Verificou-se que o modelo que optimiza a capacidade total é o melhor no que diz respeito aos custos da capacidade de reserva e aos custos da capacidade total, face ao modelo que optimiza apenas a capacidade de reserva. Verificou-se também que o modelo de optimização da capacidade de trabalho consegue maximizar a capacidade de trabalho total que é protegida pelos *p-cycles*, quando são usadas as capacidades de reserva de cada arco que foram obtidas através do modelo que optimiza a capacidade de reserva.

Um outro estudo feito no âmbito desta dissertação foi usar a métrica da 'eficiência a priori' (AE) nos modelos de optimização da capacidade de reserva e da capacidade conjunta, por forma a escolherem-se os melhores ciclos, de entre os ciclos candidatos, com base na sua eficiência.

Palavras-Chave: protecção, redes de telecomunicações, *p-cycles*, 'eficiência a priori'.

Abstract

Being society even more dependent of telecommunication networks, it is essencial that these networks be properly protected against failures that can occur. An efficient way to protect a network is using *p-cycles*. A *p-cycle* is a pre-planned cycle that takes advantage of the spare capacity in the network. Being a structure of ring type, the *p-cycle* guarantees one protection path to each span that belongs to the cycle and two protection paths to each *straddling* span - span whose end nodes belong to the cycle but the span itself does not belong to the cycle.

In this dissertation *p-cycles* were studied and three different models of integer linear programming were implemented for their creation. The first model aims to minimize the spare capacity necessary to form the *p-cycles*. The second model aims to minimize the cost of the total capacity required for routing traffic and to provide protection. And the third model aims to maximize the total working capacity that is protected by *p-cycles*. The results show that the model that optimizes the combined capacity is the best, when compared to the model which only enhances the spare capacity. The results also show that the third model can maximize the total working capacity which is protected by *p-cycles*, when the spare capacity of each arc obtained by the model that optimizes the spare capacity is used.

The metric of a priori efficiency (AE) was studied and used in the optimization models of spare capacity and joint capacity in order to choose the best cycles, among the candidates cycles, based on their efficiency.

Keywords: protection, telecommunication networks, *p-cycles*, a priori efficiency.

"It does not matter how slowly you go as long as you do not stop"

Confucius

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas

NEPC	node encircling p-cycles
WDM	Wavelength-division Multiplex
SCO	Spare Capacity Optimization
JCO	Joint Capacity Optimization
WCO	Working Capacity Optimization
CIDA	Capacitated Iterative Design Algorithm
DWE	Actual efficiency
HPS	Heuristic p-cycle selection

Capítulo 1

Introdução

1.1 Motivação e Objectivos

Nos dias de hoje, existe uma variedade de serviços de telecomunicações, essenciais na sociedade actual, que dependem do correcto funcionamento das redes de comunicação, como por exemplo, os serviços baseados na Internet, comunicações multimédia, etc. Por isso, é necessário projectar redes resilientes que consigam sobreviver a falhas, como por exemplo, cortes em cabos, avarias, erros humanos, entre outros. A tolerância a falhas é conseguida através do redireccionamento do tráfego a partir dos caminhos afectados pelas falhas para caminhos que estejam livres delas [2]. Os caminhos onde o tráfego é transportado antes de ocorrer uma falha designam-se caminhos activos. Após a ocorrência de uma falha o tráfego vai ser redireccionado para outro caminho, que se designa caminho de protecção. Este caminho de protecção pode ser estabelecido de várias maneiras, podendo ser ou não totalmente disjunto com o caminho activo [2]. A capacidade de reserva, i.e., o acréscimo da capacidade instalada para além da estritamente necessária às comunicações e que é alocada nos caminhos de protecção, introduz redundância na capacidade instalada. A redundância garante assim a sobrevivência a falhas nas redes. Apesar de ser dispendiosa, a redundância é vantajosa, uma vez que os custos resultantes da ocorrência de falhas podem ser ainda mais elevados [2].

A sobrevivência da rede a falhas pode ser conseguida usando mecanismos de protecção e de restauro. No caso da protecção, por exemplo de um caminho, este tem de estar pré-planeado, e a capacidade necessária para a reposição do tráfego afectado deve estar disponível, enquanto que no restauro os caminhos são determinados e estabelecidos no momento em que ocorre uma falha. Quer a protecção quer o restauro podem ser locais, ao segmento, ou globais. Enquanto que na protecção local a protecção é feita ao nível de um arco ou de um nó, na protecção global a protecção é feita extremo a extremo, i.e., existe um caminho pré-estabelecido que fica activo a partir do momento em que é detectada uma falha no caminho que estava activo anteriormente. Na protecção ao segmento o caminho é dividido em segmentos (que podem ter, em geral, um

arco em comum) e cada um tem associado um percurso de desvio para recuperação de qualquer falha nesse segmento.

Nesta dissertação, pretende-se estudar um tipo particular de protecção em redes de telecomunicações, que são os *p-cycles*, ou seja, "ciclos de protecção pré-estabelecidos" [12]. Os *p-cycles* são estruturas em anel que tiram partido da capacidade de reserva na rede. Este mecanismo de protecção pode ser usado para proteger um arco, um nó, um caminho ou parte de um caminho. Neste mecanismo de protecção são apenas necessárias duas acções de comutação, por exemplo, nos nós extremos do arco que falhou, por forma a que o tráfego seja reencaminhado pela parte do ciclo que não foi afectada pela falha. Este mecanismo é semelhante aos anéis, onde um conjunto de nós formam um ciclo fechado e a ligação de um dado nó é feita aos seus dois nós adjacentes [11]. A grande vantagem dos *p-cycles*, face aos anéis, é que eles garantem protecção aos arcos pertencentes ao ciclo, bem como aos arcos *straddling*, ou seja, "arcos enquadrados pelo ciclo" - arco cujos nós extremos fazem parte do ciclo, mas cujo arco não faz parte do ciclo.

O objectivo deste trabalho foi o estudo de *p-cycles*. Assim, estudaram-se diferentes tipos de *p-cycles* e implementaram-se três modelos baseados em programação linear inteira para criação de *p-cycles* para protecção ao arco. Assim, esta dissertação irá focar-se em três modelos diferentes de optimização. O primeiro modelo tem como objectivo minimizar a capacidade de reserva necessária para a formação dos *p-cycles*. O segundo modelo tem como objectivo a minimização do custo da capacidade total necessária para o encaminhamento do tráfego e para garantir protecção. E o terceiro modelo tem como objectivo a maximização da capacidade de trabalho total que é protegida pelos *p-cycles*. Estes modelos de programação linear inteira vão ser descritos com mais detalhe no capítulo 3.

1.2 Estrutura da Dissertação

Esta dissertação é estruturada da seguinte forma. No capítulo 2 vão ser apresentados os tipos existentes de *p-cycles* assim como algumas abordagens para a sua formação. No capítulo 3 vão ser descritos em pormenor três modelos diferentes para criação de *p-cycles*, para protecção ao arco, baseados na programação linear inteira, que resolvem três problemas distintos associados à formação de *p-cycles* e vão ser apresentados os algoritmos auxiliares implementados neste trabalho. No capítulo 4 vão ser apresentados e analisados alguns resultados obtidos e, por último, no capítulo 5 são apresentadas as principais conclusões deste trabalho.

Capítulo 2

Introdução aos *p-cycles*

Sendo os *p-cycles* um mecanismo de protecção em redes de telecomunicações, neste capítulo, depois de se introduzirem alguns conceitos básicos sobre protecção, são apresentados os diferentes tipos de *p-cycles* conhecidos. Para além disso vão ser ainda descritas diferentes estratégias para obtenção de *p-cycles*, bem como diferentes tipos de protecção conseguidos através de *p-cycles*.

2.1 Conceitos básicos sobre protecção em redes

A protecção, como foi referido anteriormente, pode ser local ou global. Para além disso a protecção pode ainda ser partilhada ou dedicada. Temos assim, os seguintes tipos de protecção [2]:

1. *protecção 1+1* – na protecção global 1+1 o tráfego é simultaneamente transmitido no caminho activo e no caminho de protecção. No caso da protecção local 1+1 o princípio de funcionamento é o mesmo embora apenas um único arco ou nó (não um caminho) seja contornado em caso de falha. Apesar de a protecção ao arco necessitar de mais capacidade de reserva do que a que é necessária na protecção ao caminho, o redireccionamento do tráfego é mais rápido, pois a decisão é tomada no nó localizado mais perto da falha;

2. *protecção 1:1* – na protecção global o tráfego é apenas encaminhado no caminho activo antes da ocorrência de uma falha. Na ausência de falhas no caminho activo o caminho de protecção pode ser usado para tráfego de baixa prioridade (o qual sofre preempção em caso de falha no caminho activo);

3. *protecção partilhada M:N* – na protecção global partilhada M:N os caminhos activos e de protecção (M protecção, N activos, $N \geq M$) são estabelecidos antes de ocorrer qualquer falha. Quando ocorre uma falha num caminho activo, o tráfego é redireccionado para o caminho de protecção. Tendo em conta os custos associados a este tipo de protecção, o método mais comum é a protecção *1:N*. À protecção *1:N* estão associadas também a rapidez e o uso de recursos de forma mais eficiente.

Os dois últimos tipos de protecção apresentados, à semelhança do primeiro tipo, também podem ser locais, com as vantagens e inconvenientes já referidos.

2.2 Os diferentes tipos de *p-cycles*

Os diferentes tipos de *p-cycles* que se conhece são os seguintes:

- *p-cycle Hamiltoniano* – é aquele que passa através de todos os nós da rede uma única vez.
- *p-cycle simples* – é aquele que não passa por um nó ou por um arco mais do que uma vez.
- *p-cycle não simples* – é aquele que passa por um nó ou por um arco mais do que uma vez.
- *p-cycle de protecção ao arco* – é aquele que garante protecção aos arcos.
- *p-cycle de protecção ao nó* – é aquele que passa através dos nós adjacentes ao nó protegido mas que não passa por esse nó. No caso de falha do nó, o tráfego que passa por esse nó é então salvo por este tipo de ciclos.
- *p-cycle de protecção ao caminho* – é aquele que protege um caminho cujo nó origem e o nó destino estejam sobre o ciclo.
- *p-cycle de protecção ao segmento de um caminho* – é aquele que protege uma parte do caminho que esteja sobre o ciclo.

2.3 Estratégias para obtenção de *p-cycles*

Como foi dito anteriormente, a grande vantagem dos *p-cycles* face aos anéis, no caso da protecção ao arco, é garantirem também protecção aos arcos que não pertencem ao ciclo mas cujos nós extremos pertençam ao ciclo. Na figura 2.1 a) é mostrado um exemplo de um *p-cycle*. O que o *p-cycle* mostrado na figura tem de interessante é que, qualquer que seja o arco que falhe na rede, ele está protegido. Como se pode ver na figura 2.1 b), em que está representada a falha do arco A-C pertencente ao ciclo, este garante que o tráfego que era antes encaminhado através deste arco pode ser redireccionado através dos outros arcos do ciclo. No caso de falha do arco *straddling* B-C, como mostra a figura 2.1 c), o *p-cycle* garante dois percursos de protecção, sendo esta a principal vantagem deste mecanismo de protecção face aos anéis.

Os *p-cycles* podem ser determinados por diferentes métodos que podem ser métodos centralizados ou métodos distribuídos. No caso dos métodos centralizados, estes estão divididos em dois grupos. No primeiro grupo temos os modelos de programação linear inteira, que são os estudados no âmbito desta dissertação e, no segundo grupo, temos os métodos baseados em heurísticas.

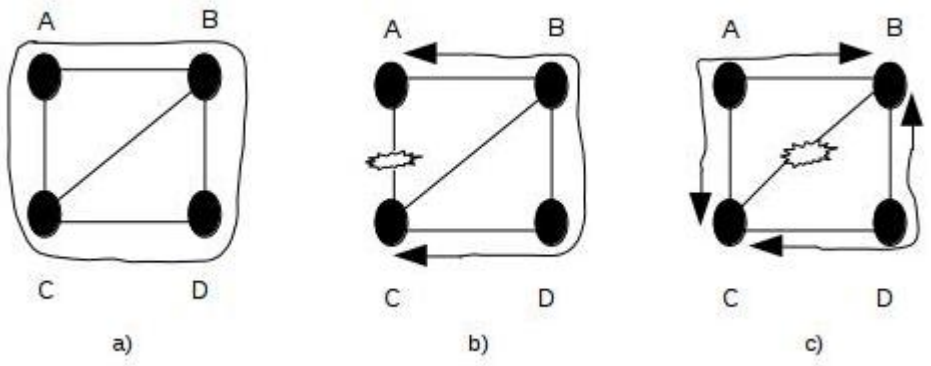

Figura 2.1: a)*p-cycle* A-B-D-C-A, b)falha do arco A-C, c) falha do arco B-C

A vantagem das heurísticas face aos modelos de programação linear inteira é que no caso de redes maiores, o tempo necessário para que seja obtida uma solução é menor, mas não é garantida a optimalidade da solução. Dentro dos métodos heurísticos descrevem-se de seguida alguns dos mais conhecidos. A heurística que se baseia no algoritmo *Capacitated Iteractive Algorithm (CIDA)*, pode usar todos os ciclos candidatos ou apenas um sub-conjunto de ciclos seleccionados [1]. O sub-conjunto de ciclos pode ser seleccionado tendo em conta a métrica *actual efficiency (DWE)*, i.e., 'eficiência real'. Para que a capacidade de trabalho de todos os ciclos candidatos seleccionados seja protegida, a cada iteração do algoritmo, o ciclo com maior eficiência vai ser seleccionado (assume-se que os ciclos seleccionados garantem a protecção de todos os arcos da rede). Em relação ao método heurístico *Heuristic p-Cycle Selection (HPS)* [1], os *p-cycles* candidatos vão ser formados juntando dois dos k caminhos mais curtos determinados entre dois nós de cada um dos arcos da rede. Os *p-cycles* vão depois ser seleccionados, tendo em conta algumas características em particular e a pesquisa irá terminar quando toda a capacidade de trabalho estiver protegida [1]. O último método heurístico ao qual se vai fazer referência baseia-se num algoritmo genético. Nesta heurística, os *p-cycles* candidatos vão ser seleccionados tendo em conta a sua *a priori efficiency (AE)*, i.e. a sua eficiência a priori, que mede a eficiência potencial de se usar um dado ciclo em particular. A relação entre a capacidade de trabalho total da rede e a capacidade de reserva usada pelos *p-cycles* irá ser usada como função objectivo deste método [1].

No que diz respeito aos métodos distribuídos, vai ser feita referência a dois deles. No primeiro método, os *p-cycles* são formados de uma forma distribuída na capacidade de reserva da rede e os caminhos activos vão ser estabelecidos usando um algoritmo que seja adequado. Neste método, os *p-cycles* podem readaptar-se de forma a modificarem-se a eles mesmos em resposta às alterações que possam existir nas capacidades de trabalho [1]. O segundo método distribuído baseia-se em sistemas biológicos. Neste método existe um conjunto de 'agentes', que comunicam entre si de uma forma semelhante ao das formigas. O conjunto de *p-cycles* vai ser formado com

base nas mensagens que os 'agentes' deixam, ou reúnem, nos nós que visitam. Aqui, a comutação do tráfego nos *p-cycles* pode ser realizada de forma totalmente distribuída [1].

2.4 Eficiência dos *p-cycles* em termos de capacidade

A eficiência de capacidade é definida como sendo a razão entre a capacidade de trabalho e a capacidade de reserva, sendo a redundância o inverso da eficiência. Para *p-cycles* Hamiltonianos, i.e., *p-cycles* que passam por todos os nós da rede uma única vez, pode provar-se [10] que a redundância tem como limite inferior $1/(\bar{d}-1)$, onde $\bar{d} = 2A/n$, sendo A e n, respectivamente, o número de arcos (não dirigidos) e o número de nós da rede. Para que a eficiência de capacidade seja óptima, o projecto das redes vai necessitar de uma mistura de ciclos Hamiltonianos e não-Hamiltonianos [1].

Os valores de eficiência mais elevados são obtidos em redes emalhadas onde as capacidades de todos os arcos são idênticas. Para redes homogéneas, ou seja, para redes que tenham as mesmas capacidades de trabalho nos arcos, a redundância é apenas $2/\bar{d}$. A razão para isto acontecer tem a ver com o facto de que, numa rede homogénea, não é possível explorar a totalidade da capacidade de protecção dos *p-cycles*. Como cada um dos arcos tem a mesma capacidade de trabalho, a protecção aos arcos *straddling* é perdida. Devido a este facto, surgiram as redes semi-homogéneas, onde podemos ter o dobro da capacidade de trabalho nos arcos *straddling* face aos arcos que estão sobre o *p-cycle*. Assim, o limite inferior $1/(\bar{d}-1)$ para a redundância pode ser alcançado, usando *p-cycles* Hamiltonianos neste tipo de redes. No entanto, este limite não vai ser obrigatoriamente atingido pois, em redes reais, a solução óptima poderá conter outros *p-cycles*, para além dos *p-cycles* Hamiltonianos [1].

2.5 Diferentes tipos de protecção com *p-cycles*

Inicialmente, o mecanismo de protecção baseado nos *p-cycles*, estava pensado somente para protecção ao arco. Mas, à medida que foi desenvolvido o conceito de *p-cycles*, foram consideradas também a protecção ao nó, protecção ao caminho e protecção ao segmento do caminho.

No que diz respeito à protecção ao nó, os *p-cycles* baseiam-se num conceito que se designa por *node encircling p-cycles (NEPC)*. Para que um *p-cycle* consiga garantir a protecção ao nó, é necessário que ele passe por todos os seus nós adjacentes mas que não passe por esse nó. O problema deste tipo de protecção é que podemos ter *p-cycles* simples mas também, não simples. A grande desvantagem deste tipo de protecção é que, para proteger uma rede com n nós vão ser necessários n *p-cycles*, o que torna este tipo de protecção pouco apelativo [1]. Um modelo de programação linear inteira foi proposto em [6] para este tipo de protecção. Este modelo tem como base um modelo anterior presente em [3], que recorre ao modelo de programação linear

inteira que minimiza a capacidade conjunta. Para além do modelo em [3] poder ser aplicado ao modelo baseado no modelo de optimização da capacidade conjunta, ele também foi aplicado ao modelo de optimização da capacidade de reserva. Para testar este modelo foram usadas três redes com diferente número de nós e arcos. Dos resultados obtidos em [6] verificou-se que o modelo baseado no modelo de optimização da capacidade de reserva, face ao modelo baseado no modelo de optimização da capacidade conjunta, consegue obter melhores resultados no que diz respeito à minimização dos custos da capacidade total das redes. Portanto, o modelo proposto com base no modelo de optimização da capacidade de reserva garante, assim, melhor protecção ao nó, face ao modelo baseado no modelo de optimização da capacidade conjunta.

Os *p-cycles*, além de poderem garantir a protecção de um nó, também podem ser usados para garantir a protecção ao caminho. Para este tipo de protecção vai ser usado um *p-cycle* que é comum a todos os caminhos que sejam mutuamente disjuntos e cujos nós extremos pertençam ao *p-cycle*. Apesar de a comutação ser feita nos nós extremos do caminho onde ocorre a falha, vai ser necessário ter sinalização entre estes nós para que se faça essa comutação, pois o nó fonte e o nó destino do caminho em geral não correspondem aos nós extremos do(s) elemento(s) de rede do caminho que falhou [1]. À medida que as redes se tornam maiores e mais complexas, a probabilidade de ocorrerem múltiplas falhas aumenta, podendo ter consequências desastrosas. Com base nisto, em [4] foi proposto um modelo de programação linear inteira que protege redes WDM (Wavelength-division Multiplex) contra múltiplas falhas usando *p-cycles* de protecção ao caminho, cujo objectivo é minimizar a largura de banda usada para protecção. Dos resultados obtidos verificou-se que este modelo consegue obter resultados bastante bons aquando da ocorrência de múltiplas falhas na rede. Em [7], considerando *p-cycles* de protecção ao caminho no caso da ocorrência de múltiplas falhas, foi feito um estudo para ver quão estáveis são os *p-cycles* em redes WDM perante tráfego dinâmico. O modelo proposto em [7] baseia-se em configurações, onde cada configuração é definida por um ciclo e o conjunto de fluxos que este protege, de maneira a serem usadas diversas técnicas de optimização para se obter uma solução. Neste modelo, apesar de poderem ser geradas diversas configurações para um mesmo ciclo, na solução final, apenas uma configuração é seleccionada para cada ciclo. Os resultados obtidos com base em duas redes mostram que, perante um ambiente com tráfego dinâmico, o número de configurações que é modificada corresponde a uma pequena percentagem, o que prova que o modelo estudado traduz bem o comportamento dos *p-cycles* de protecção ao caminho num ambiente de tráfego dinâmico [7].

Os *p-cycles*, além de oferecerem protecção ao nó e ao caminho, também foram pensados para proteger segmentos do caminho. O segmento do caminho é uma porção do caminho activo que se encontra entre dois nós de intersecção do caminho e do *p-cycle* (Fig. 2.2). Os *p-cycles* de protecção ao segmento do caminho, garantem não só a protecção ao segmento do caminho, mas também protecção ao caminho (caso o nó fonte e o nó destino pertençam ao *p-cycle*), protecção ao nó (para qualquer nó intermédio entre dois nós que intersectem o ciclo) e protecção ao arco.

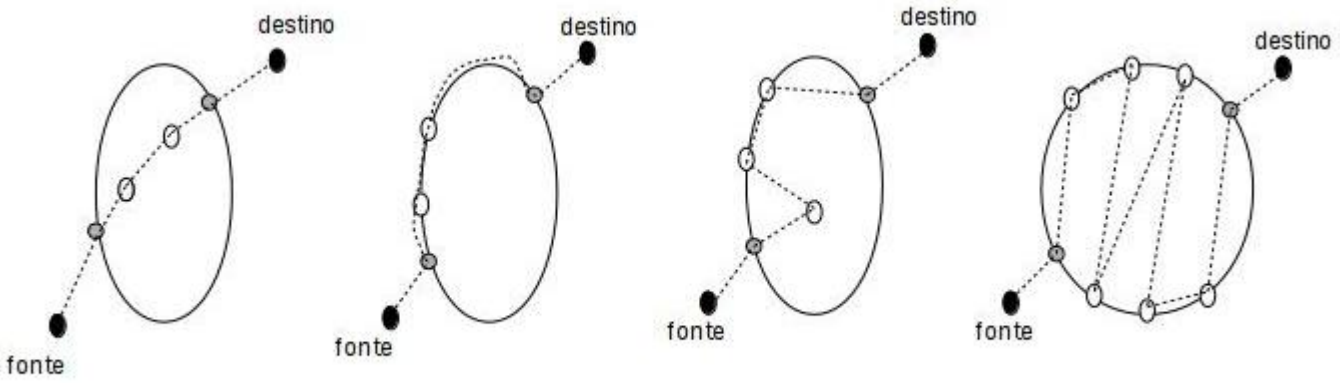

Figura 2.2: *p-cycle* de protecção ao segmento do caminho a proteger vários tipos de fluxos ou segmentos do caminho (adaptado de [1])

Apesar deste tipo de *p-cycles* oferecerem um bom compromisso entre os *p-cycles* de protecção ao arco e os de protecção ao caminho, não oferecem protecção total ao nó. A pensar nisso, é proposta uma nova abordagem em [5], chamada N*p-cycles*, que garante protecção total contra qualquer falha única (nó ou arco), em redes WDM. A ideia dos N*p-cycles* consiste em juntar pares de segmentos consecutivos que pertençam ao mesmo caminho, de forma a que ambos os segmentos sejam protegidos pelo mesmo *p-cycle*. Dos resultados obtidos em [5] verificou-se que, de um modo geral, os N*p-cycles* necessitam de mais capacidade de reserva para protecção aos nós extremos do segmento, face aos *p-cycles* de protecção ao segmento do caminho. Outros resultados que se observaram foi que, independentemente do esquema de protecção, o número de ciclos distintos é pequeno comparativamente ao número de cópias (ou seja, unidades de largura de banda no ciclo, como se explica no próximo capítulo), o que significa uma fácil gestão dos *p-cycles*. Tendo em conta o tamanho dos ciclos, verificou-se que o tamanho médio dos ciclos nunca é o maior nos N*p-cycles* em comparação com os *p-cycles* de protecção ao segmento do caminho. O tamanho dos ciclos pode ser importante pois, em termos práticos, pode ter interesse usar ciclos com o menor tamanho possível, por forma a eliminar a necessidade de regeneração do sinal em redes ópticas [8]. De um estudo feito em [8], relativamente à rede *COST239*, verificou-se que o projecto dos *p-cycles* se torna impraticável para trajectos ópticos muito reduzidos ($\leq 2500km$). Verificou-se também que são necessários mais *p-cycles* quando estes têm um menor tamanho e que o número de *p-cycles* usados se mantém, quando os tamanhos são mais elevados. Apesar de os N*p-cycles* requererem mais largura de banda que os *p-cycles* de protecção ao segmento do caminho, eles usam a largura de banda de forma mais eficiente, o que os torna num esquema de protecção melhor face aos *p-cycles* de protecção ao segmento do caminho.

Capítulo 3

Problemas abordados

Neste capítulo vão ser descritos três modelos para determinação de *p-cycles* que foram implementados e avaliados no âmbito desta dissertação, assim como os algoritmos implementados. Inicialmente vão ser ainda introduzidos alguns conceitos, subjacentes aos métodos referidos.

3.1 Conceitos Básicos

As redes de telecomunicações podem ser representadas por grafos. De seguida, vão ser apresentados alguns conceitos básicos e a notação que irá ser usada ao longo desta dissertação.

Grafos dirigidos: Um grafo dirigido $G = (N, A)$ consiste num conjunto de nós N e um conjunto de arcos A dirigidos, em que os elementos são pares ordenados de nós distintos. Um arco $b_k = (i, j)$ dirigido (com $k = 1, \ldots, |A|$) tem dois nós associados, o nó i e o nó j. O nó i designa-se por cauda e o nó j designa-se por cabeça. Diz-se que o arco sai do nó i e termina no nó j. Quando um arco $(i, j) \in A$, diz-se que o nó j é adjacente ao nó i. Os nós e/ou arcos podem ter valores associados, tais como custos, capacidades, entre outros.

Grafos não dirigidos: Um grafo não dirigido define-se da mesma forma que um grafo dirigido, com a excepção de que os arcos (não dirigidos) são pares não ordenados de nós distintos. Num grafo não dirigido um arco que junta o nó i ao nó j permite que o fluxo 'circule' tanto do nó i para o nó j como do nó j para o nó i. Assim o mesmo arco pode ser referido por $b_k = (i, j)$ ou $b_k = (j, i)$. Para simplificação de notação, no texto que se segue apenas se refere o arco pelo seu índice.

Na figura 3.1 podemos ver um exemplo de um grafo não dirigido. No decurso desta dissertação foram considerados grafos não dirigidos em representação de redes não dirigidas e, nesse sentido, usar-se-á simplesmente o termo arco para designar arcos não dirigidos.

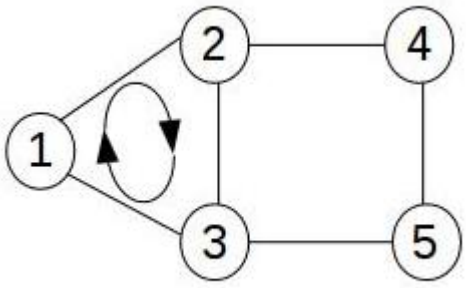

Figura 3.1: Grafo não dirigido e um ciclo

Caminho e Ciclo: Um caminho de i_1 para i_l define-se como sendo uma sequência de nós, i_1 - i_2 - $\cdots$ - i_{l-1} - i_l, onde não haja repetição de nós, e onde $(i_j, i_{j+1}) \in A$, $j = 1, \ldots, l-1$. Um ciclo define-se como sendo um caminho $i_1 - i_2 - \cdots - i_l - i_1$ que inclui o arco (i_l, i_1) ou (i_1, i_l). Na figura 3.1 é mostrado o ciclo $1 - 2 - 3 - 1$ com base no grafo não dirigido representado. Nesta dissertação foi usada a notação $i_1 - i_2 - \cdots - i_l - i_1$ para representar um ciclo. Dado um caminho q o conjunto de arcos (não dirigidos) desse caminho será designado por A_q.

Lista de adjacências: Ao representarmos uma rede, a informação que nos interessa armazenar é a estrutura dos nós e arcos da rede, bem como dados como os custos e capacidades associados aos arcos. A representação escolhida para a rede no âmbito desta dissertação foi uma lista de adjacências. Uma lista de adjacências consiste num conjunto de listas, uma para cada nó do grafo, onde cada lista contém um conjunto de células com um ou mais campos. Na lista correspondente a cada nó, encontra-se a lista de registos com informação de todos os arcos que têm um extremo nesse nó. Assim na lista associada ao nó i, o arco (i, j) (ou (j, i)) irá dar origem a um registo com a seguinte informação: a identificação do nó j, a identificação do arco (i, j), o custo por unidade de capacidade, capacidade de trabalho e a capacidade de reserva do arco (i, j). De forma semelhante na lista associada ao nó j existirá o registo com a representação (j, i) (ou (i, j)), que contém a identificação do nó i, a identificação do arco (i, j), o custo por unidade de capacidade, capacidade de trabalho e a capacidade de reserva do arco (i, j). Esta duplicação de informação tem como objectivo permitir de forma eficiente encontrar a informação associada aos arcos com extremo em qualquer nó da rede.

3.2 Métodos centralizados implementados

Como foi referido no capítulo dois, no âmbito desta dissertação foram implementados três modelos diferentes de programação linear inteira para obtenção de *p-cycles*. No primeiro modelo pretende-se minimizar o custo da capacidade de reserva usada pelos *p-cycles*. No segundo modelo pretende-se minimizar o custo da capacidade total necessária para o encaminhamento do tráfego

e para protecção. E no terceiro modelo pretende-se maximizar a capacidade de trabalho, ou seja, a capacidade dos caminhos activos, que é protegida pelos *p-cycles*. As três abordagens consideradas nesta tese necessitam, como dados de entrada, que lhes sejam fornecidos os ciclos candidatos. Apenas com o conjunto total dos ciclos existentes numa rede, poderá garantir-se que foi encontrada a solução óptima para cada uma das formalizações consideradas. Contudo, como o número de ciclos numa rede é muito elevado, optou-se por utilizar um subconjunto desses ciclos. O algoritmo de geração desses ciclos vai ser descrito na secção 3.3.

3.2.1 Optimização da Capacidade de Reserva (SCO):

Nesta primeira abordagem, para além de ser necessário obter previamente os ciclos candidatos vai ser necessário ter a capacidade de trabalho utilizada em cada arco.

De modo a ter valores realistas para a capacidade de trabalho de cada arco da rede, usou-se a matriz de tráfego existente na SNDlib [9] para cada uma das redes testadas. Para cada arco, a capacidade de trabalho necessária vai ser dada pela soma das capacidades de trabalho necessárias nos arcos do caminho mais curto determinado, entre cada par origem-destino. O que se pretende com este modelo é minimizar a capacidade necessária para a formação dos *p-cycles*. O modelo é dado a seguir.

Conjuntos:

A: Conjunto dos arcos, indexados por j.

P: Conjunto dos *p-cycles*, indexados por p.

Parâmetros:

c_j: Custo do arco j.

w_j: Capacidade de trabalho do arco j.

π_j^p: Igual a 1 se o *p-cycle* p passa sobre o arco j, caso contrário vai ser igual a 0.

x_j^p: Igual a 1 se o *p-cycle* p protege o arco j que passa sobre o ciclo, igual a 2 se o *p-cycle* p protege o arco *straddling* j e 0 caso contrário.

Variáveis:

a_j: Capacidade de reserva necessária no arco j.

n^p: Número de cópias de capacidade unitária do *p-cycle* p na solução, ou seja, número de unidades de largura de banda em cada arco do *p-cycle* p na solução.

O objectivo deste problema de optimização de programação linear inteira é minimizar [1]:

$$\sum_{j \in A} c_j a_j \tag{3.1}$$

Sujeito a:

$$w_j \leq \sum_{p \in P} x_j^p n^p \qquad \forall j \in A, \tag{3.2}$$

$$a_j = \sum_{p \in P} \pi_j^p n^p \qquad \forall j \in A, \tag{3.3}$$

e

$$n^p \geq 0 \qquad \forall p \in P. \tag{3.4}$$

A função objectivo, equação (3.1), minimiza a capacidade de reserva total, ponderada pelos seus custos, usada para a formação dos *p-cycles*. A equação (3.2) garante uma protecção de 100% para a capacidade de trabalho de cada arco no caso de ocorrer uma única falha. E a equação (3.3) garante uma capacidade de reserva suficiente em cada arco para a formação de *p-cycles*.

3.2.2 Optimização da Capacidade Total (JCO):

Enquanto que no modelo anterior apenas a capacidade de reserva é optimizada, este modelo optimiza a capacidade total, i.e, a capacidade de reserva mais a capacidade de trabalho. Aqui, para além dos ciclos candidatos, irá ser necessário um conjunto de caminhos candidatos para cada par origem-destino. Foram considerados 10 caminhos mais curtos entre cada par origem-destino, como o conjunto de caminhos activos candidatos (estes caminhos podem ser obtidos usando, por exemplo, o algoritmo de Yen [13]). A resolução do problema JCO irá seleccionar alguns destes caminhos como caminhos de trabalho em conjunto com a capacidade de reserva, de maneira a minimizar a capacidade total. Ao conjunto dos parâmetros e variáveis usados no modelo anterior, vão ser adicionados outros que são dados de seguida.

Conjuntos:

D: Conjunto de fluxos entre cada par de nós na rede, não nulos, indexados por t.

E^t: Conjunto de caminhos activos candidatos para cada fluxo t, indexado por e.

Parâmetros:

d^t: Valor do fluxo t (inteiro).

$\epsilon_j^{t,e}$: Igual a 1 se o e-ésimo caminho de trabalho para o fluxo t passar através do arco j, 0 caso contrário.

Variáveis:

$u^{t,e}$: Número de unidades do fluxo t, atribuídas ao e-ésimo caminho candidato em E^t.

w_j: Capacidade de trabalho do arco j.

O objectivo é minimizar:

$$\sum_{j \in A} c_j (w_j + a_j) \tag{3.5}$$

Sujeito a:

$$d^t = \sum_{e \in E^t} u^{t,e} \qquad \forall t \in D, \tag{3.6}$$

$$w_j = \sum_{t \in D} \sum_{e \in E^t} u^{t,e} \epsilon_j^{t,e} \qquad \forall j \in A, \tag{3.7}$$

$$a_j = \sum_{p \in P} \pi_j^p n^p \qquad \forall j \in A, \tag{3.8}$$

$$w_j \leq \sum_{p \in P} x_j^p n^p \qquad \forall j \in A, \tag{3.9}$$

$$n^p \geq 0 \qquad \forall p \in P, \tag{3.10}$$

e

$$a_j \geq 0 \qquad w_j \geq 0 \qquad \forall j \in A. \tag{3.11}$$

A função objectivo, equação (3.5), minimiza a capacidade total, ponderada pelos custos, necessária para acomodar os caminhos activos e para garantir protecção. A restrição (3.6) garante que todos os fluxos são encaminhados, e a restrição (3.7) dá-nos a capacidade de trabalho no arco. A equação (3.8) garante que existe capacidade de reserva suficiente no arco necessário por todos os *p-cycles* do conjunto da solução. A restrição (3.9) garante que a protecção fornecida por todos os *p-cycles* do conjunto da solução é suficiente para proteger todas as capacidades de trabalho do arco.

3.2.3 Optimização da Capacidade de Trabalho (WCO):

Neste modelo, para além dos ciclos candidatos, vai ser ainda necessário ter a capacidade de reserva para cada arco. A capacidade de reserva usada para cada arco deste modelo, foi obtida através da resolução do modelo que optimiza a capacidade de reserva. Neste modelo, o que se pretende é maximizar a capacidade de trabalho total que é protegida pelos *p-cycles*. O modelo é dado a seguir.

O objectivo é maximizar:

$$\sum_{\forall j \in A} w_j \tag{3.12}$$

Sujeito a:

$$w_j \leq \sum_{p \in P} x_j^p n^p \qquad \forall j \in A, \tag{3.13}$$

$$a_j \geq \sum_{p \in P} \pi_j^p n^p \qquad \forall j \in A, \tag{3.14}$$

e

$$n^p \geq 0 \qquad \forall p \in P. \tag{3.15}$$

A função objectivo, equação (3.12), maximiza a capacidade de trabalho total que é protegida pelos *p-cycles*. A restrição (3.13) garante que a protecção fornecida por todos os *p-cycles* do conjunto da solução é suficiente para proteger todas as capacidades de trabalho do arco. E a equação (3.14) garante uma capacidade de reserva suficiente em cada arco para a formação de *p-cycles*.

3.3 Algoritmos auxiliares implementados

Como foi referido na secção 3.2, são necessários ciclos candidatos para a resolução dos três modelos implementados nesta dissertação. Para a geração dos ciclos recorreu-se ao algoritmo de Dijkstra e ao algoritmo de Yen [13], com diferentes valores para o número de caminhos a obter (k). O algoritmo de Dijkstra foi implementado no âmbito deste trabalho e encontra-se descrito na sub-secção 3.3.1. No caso do algoritmo de Yen utilizou-se código que pode ser encontrado no sítio https://github.com/yan-qi/k-shortest-paths-cpp-version/tree/master/src[1].

O algoritmo que gera os ciclos candidatos encontra-se descrito na sub-secção 3.3.5 e utiliza como sub-rotinas o algoritmo para encontrar ciclos que passam em um dado arco (algoritmo 3), o algoritmo para encontrar ciclos *straddling* para um dado arco (algoritmo 4), e finalmente o algoritmo 2 (que também é utilizado como sub-rotina pelo algoritmo 4) para obter alguns ciclos adicionais, apresentado na sub-secção 3.3.2.

3.3.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra é um algoritmo que permite determinar o caminho mais curto entre um dado nó e todos os outros nós da rede, cujo custo dos arcos é não negativo. A cada nó atribui-se uma etiqueta que pode ser temporária ou permanente. Uma etiqueta permanente associada a um dado nó representa a verdadeira distância mais curta do nó origem a esse nó da rede. Uma etiqueta temporária representa o comprimento de um caminho do nó origem a esse nó da rede. Uma vez que este caminho pode ou não ser o caminho mais curto, o que a etiqueta temporária representa é um limite superior da distância do nó origem a esse nó, numa dada iteração do algoritmo. Para a implementação deste algoritmo foi usada uma heap binária, pelo que ela é descrita de seguida.

Uma *heap* (ou fila prioritária) é uma estrutura de dados que representa uma forma eficiente de armazenar e manipular um conjunto de elementos ordenados. A estrutura de uma *heap*

[1] A versão utilizada foi modificada pelo Doutor Jaime Silva para remover uma fuga de memória.

binária é semelhante a uma árvore binária, onde cada elemento tem associado uma chave e onde cada um dos arcos dessa árvore representam um par, pai-filho. A árvore tem de satisfazer uma dada propriedade de ordenamento. A raiz contém a menor chave caso a *heap* seja ordenada de forma crescente ou, contém a maior chave, caso seja ordenada de forma decrescente. No âmbito desta dissertação interessa-nos que a *heap* esteja ordenada de forma a que a raiz contenha a menor chave. A figura 3.2 mostra um exemplo de uma *heap* binária e a sua representação do ponto de vista computacional que nada mais é do que um vector. Esta representação tem de

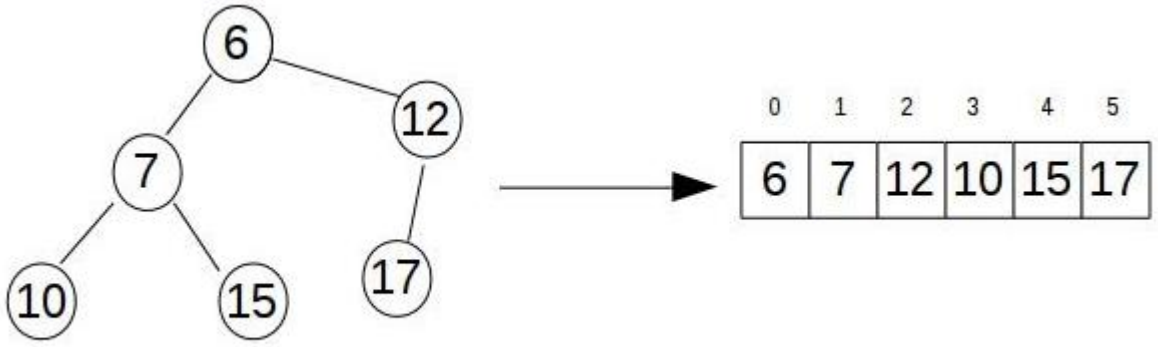

Figura 3.2: Representação de uma *heap* binária

obedecer às seguintes propriedades:

Propriedade 1. Para o elemento na posição de índice k (com $k = 0, \ldots, |N| - 1$):

(a) o pai está na posição $(k-1)/2$, com excepção da raiz ($k = 0$).

(b) o filho da esquerda está na posição $2 * k + 1$.

(c) o filho da direita está na posição $2 * k + 2$.

Propriedade 2. A chave do elemento k na *heap* é sempre menor ou igual às chaves dos filhos. A raiz contém o elemento com a menor chave.

A implementação da *heap* obedece a duas operações básicas: *Inserir* e *Eliminar*.

1. Inserir: O novo elemento é inicialmente anexado no fim da *heap* (como o último elemento do vector). Esse novo elemento vai ser comparado com o pai, e vai trocando de posição com o pai, até o pai ser menor ou igual a esse elemento. Este processo designa-se "*subir na heap*".

2. Eliminar: O elemento de menor valor encontra-se na raiz, que é o primeiro elemento do vector. Depois de retirado o elemento de menor valor da raiz, coloca-se na raiz o último elemento da *heap* e restaura-se a ordenação fazendo-se o "*descer na heap*".
   Descrição do processo:

(a) Colocar na raiz da *heap* o último elemento do último nível.

(b) Comparar a nova raiz com os seus filhos. Caso estejam correctamente ordenados, termina-se o processo.

(c) Caso contrário, troca-se sucessivamente cada elemento com o menor dos seus filhos.

O pseudocódigo do algoritmo de Dijkstra é apresentado no algoritmo 1, onde H é a *heap*, e a função heap_up corresponde ao procedimento "*subir na heap*". O caminho p (saída do algoritmo) é obtido da sequência de predecessores, começando pelo nó destino até atingir o nó origem.

Algoritmo 1: Dijkstra com *Heap*

Data: nó origem (s), nó destino (t), grafo $G(N, A)$
Result: devolve o caminho p de distância mais curta da origem para o destino

```
1  begin
2      Temp ← N                                   // conjunto dos nós temporários
3      H.reserva(|N|)          // reserva espaço (|N| elementos) para a heap binária
4      Pred.reserva(|N|)       // reserva espaço (|N| elementos) para predecessores
5      for i ← 0 to |N| − 1 do                          // valores iniciais na heap
6          Pred[i] ← s                  // predecessor de todos os nós é a origem
7          if i = s then                                  // se i for o nó de origem
8              H[i].dist ← 0                // atribui-lhe uma distância igual a 0
9          else H[i].dist ← ∞          // caso contrário, atribui uma distância ∞
10         H.heap_up(i)                        // coloco o nó i na posição correcta
11     repeat                              // Até o nó destino ter etiqueta permanente
12         i ←H.find_min()               // vai à procura do nó com a distância mínima
13         Temp ← Temp\{i}                    // etiqueta do nó i passa a definitiva
14         if i ≠ t then                                 // Se ainda não chegou ao nó t
15             for cada nó j ∈ Temp adjacente a i do
16                 novo_custo ← H.dist[i] + custo_arco(i, j);
17                 if novo_custo < H.dist[j] then
18                     Pred[j] ← i                // actualiza o nó predecessor do nó j
19                     H[j].dist ← novo_custo          // actualiza a distância do nó j
20                     H.heap_up(j)                       // reposiciona o nó j na heap
21             H.delete_min()                          // Remove a raiz da heap, o nó i
22     until i = t
23     p ← Caminho(s, t, Pred)                          // Constrói caminho mais curto
24     return p                              // Devolve o caminho mais curto de s para t
```

3.3.2 Algoritmo para encontrar ciclos por par origem-destino

O algoritmo 2 recorre ao algoritmo de Dijkstra para encontrar o caminho mais curto do nó origem para o nó destino. De seguida os nós intermédios desse caminho são removidos da rede (ou, caso esse conjunto seja vazio é removido o arco que define o caminho), e é aplicado o algoritmo de Yen [13], que vai calcular os k caminhos mais curtos alternativos para esse par origem-destino. O caminho mais curto calculado inicialmente vai ser concatenado a cada um desses k caminhos, formando k ciclos diferentes.

Formalmente, seja

- q um caminho de i_1 para i_l (i_1 - i_2 - $\cdots$ - i_{l-1} - i_l)
- (i_l, i_k) um arco não dirigido, tal que i_k é diferente de qualquer nó intermédio de q.

A concatenação de q com (i_l, i_k) representa-se por $q \diamond (i_l, i_k)$. O caminho resultante desta concatenação é i_1 - i_2 - $\cdots$ - i_{l-1} - i_l - i_k, que será um ciclo se $i_k = i_1$.

Cada um dos ciclos formados é depois adicionado ao conjunto dos ciclos e, no fim, os nós/arco inicialmente removidos, são novamente repostos na rede. O algoritmo 2 apresenta o pseudocódigo para a determinação dos ciclos para um dado par origem-destino.

Neste trabalho considerou-se que o custo de cada arco (custo de cada unidade de capacidade) é dado pela distância geográfica entre os nós extremos de cada arco. Assim, cada arco será sempre o caminho mais curto entre os seus nós extremos, pelo que a abordagem acima descrita irá, neste caso, resultar em ciclos que coincidem com os obtidos pelo algoritmo 3, pelo que não são gerados.

3.3.3 Algoritmo para encontrar ciclos que passam em um dado arco

O cálculo de ciclos que passam num dado arco (não dirigido), pode ser obtido removendo esse arco da rede e em seguida calculando caminhos entre os nós extremos desse arco. Seguidamente a concatenação de cada um desses caminhos com o arco removido dará origem a ciclos (que obviamente passam nesse arco).

O algoritmo 3 descreve o procedimento de criação dos ciclos que passam num dado arco.

3.3.4 Algoritmo para encontrar ciclos *straddling* para um dado arco

Este algoritmo, calcula ciclos *straddling* para um dado arco da rede. Dado um arco, a sua cauda e cabeça são colocadas num vector caminho. De seguida, o arco é removido e vão ser encontrados os ciclos *straddling* com recurso ao algoritmo 2. Depois de os ciclos *straddling* estarem encontrados para um dado arco, esse arco é reposto na rede e o algoritmo devolve os ciclos *straddling* encontrados para o arco em causa. O algoritmo 4 apresenta o pseudocódigo para a determinação dos ciclos *straddling* para um dado arco.

Algoritmo 2: Encontra ciclos para um par (s,t)

Data: nó origem (s), nó destino (t), grafo $G(N, A)$, número de caminhos alternativos k considerados

Result: devolve o conjunto dos ciclos gerados

```
1  begin
2      ciclos ← ∅                                   // nenhum ciclo foi encontrado
3      q ← Dijkstra(s,t)              // caminho mais curto de origem para o destino
4      if q = s - t then                    // se o caminho mais curto for um arco
5          sai sem ciclos               // estes ciclos são obtidos pelo algoritmo 3
6      A_r ← { arcos incidentes nos nós intermédios de q}
7      N_r ← { nós intermédios de q}
8      A' ← A \ A_r
9      N' ← N \ N_r
10     G ← (N', A')                     // novo grafo sem nós intermédios do caminho
11     i ← 0                             // contador de caminhos mais curtos gerados
12     repeat                               // gera k ciclos se existirem k caminhos
13         i ← i + 1                        // vai contando os caminhos alternativos
14         p ← q                  // o ciclo p é inicialmente o caminho mais curto
15         b_i ← i-ésimo caminho de s para t em G obtido pelo algoritmo de Yen
16         if b_i ≠ ∅ then                                  // se o caminho existe
               // concatena o caminho q com o caminho b_i criando o ciclo
17             pilha ← cria pilha com os nós de b_i  // começando em s e terminando em t
18             pilha.pop()        // extrai o último elemento da pilha (nó destino)
19             while pilha não estiver vazia do      // junta b_i ao caminho mais curto
20                 j ← último nó do caminho no ciclo p
21                 p ← p ⋄ (j,pilha.top())              // junta mais um nó ao ciclo p
22                 pilha.pop()                    // removendo-o de seguida da pilha
23             ciclos ← ciclos ∪ p  // coloca o ciclo p gerado no conjunto de ciclos
24     until (q = ∅ ∨ i = k)
25     G ← (N ∪ N_r, A' ∪ A_r)                 // volta a repor arco/nós intermédios
26     return ciclos                                    // Devolve os ciclos gerados
```

Algoritmo 3: Encontra ciclos que passam num dado arco

Data: arco (s, t), grafo $G(N, A)$, número de caminhos alternativos k considerados
Result: devolve o conjunto dos ciclos gerados

```
1  begin
2      A' ← A \ {(s,t)}                          // remove o arco não dirigido
3      G ← (N, A')                               // novo grafo sem o arco removido
4      i ← 0                                     // número corrente de caminhos é zero
5      repeat                                    // gera k ciclos se existirem k caminhos
6          i ← i + 1                             // vai contando os caminhos alternativos
7          q_i ← i-ésimo caminho de s para t em G (algoritmo de Yen)
8          if q_i ≠ ∅ then                       // se o caminho existe
9              p ← q_i ◇ (t, s)                  // cria ciclo
10             ciclos ← ciclos ∪ p   // coloca o ciclo p gerado no conjunto de ciclos
11         else
12             p ← 0
13     until (p = ∅ ∨ i = k)
14     G ← (N, A' ∪ {(s,t)})                     // volta a repor o arco (s,t)
15     return ciclos                             // Devolve os ciclos gerados
```

Algoritmo 4: Encontra ciclos *straddling* para um dado arco

Data: cauda (s), cabeça (t), grafo $G(N, A)$, número de caminhos alternativos k considerados
Result: devolve em ciclos_straddling o conjunto dos ciclos *straddling* gerados

```
1  begin
2      A' ← A \ {(s,t)}                          // remove o arco (s,t) da rede
3      G ← (N, A')                               // novo grafo sem o arco (s,t)
       // encontra os ciclos straddling com base no algoritmo 2
4      ciclos_straddling ← algoritmo 2(s, t, G, k)
5      G ← (N, A' ∪ {(s,t)})                     // volta a repor o arco (s,t)
6      return ciclos_straddling                  // Devolve os ciclos straddling
```

3.3.5 Algoritmo de geração de ciclos candidatos

Este algoritmo vai encontrar ciclos candidatos, *straddling* e não *straddling*, de uma rede. Em primeiro lugar, para encontrar os ciclos que passam num dado arco, este algoritmo vai percorrer todos os arcos da rede aplicando de seguida o algoritmo 3. Em seguida gera os ciclos *straddling*, utilizando o algoritmo 4 para cada arco da rede. Finalmente gera mais alguns ciclos para todos os pares origem-destino da rede cujo caminho mais curto não seja o arco que os liga directamente (caso existam). No fim, junta os ciclos *straddling* e não *straddling* ao conjunto de todos os ciclos e devolve-os. O pseudocódigo correspondente encontra-se no algoritmo 5.

Algoritmo 5: Encontra ciclos candidatos

Data: nó origem (s), nó destino (t), grafo $G(N, A)$, número de caminhos alternativos k considerados para cada sub-algoritmo gerador de ciclos
Result: devolve em conj_ciclos todos os ciclos encontrados

```
1  begin
2      foreach (s,t) ∈ A do
           // encontra k ciclos que passam no arco (s,t)
3          ciclos_pass ← algoritmo 3(s,t,G,k)
4      foreach (s,t) ∈ A do
           // encontra k ciclos straddling para o arco (s,t)
5          ciclos_strad ← algoritmo 4(s,t,G,k)
6      ciclos_st ← ∅
7      foreach s ∈ N do
8          foreach t ∈ N do
9              if s < t then              // para |N|(|N|−1)/2 pares origem-destino
                   // k ciclos contendo o caminho mais curto de s para t
10                 ciclos_st ← ciclos_st ∪ algoritmo 2(s,t,G,k)
       // conjunto de todos os ciclos gerados
11     conj_ciclos ← ciclos_pass ∪ ciclos_strad ∪ ciclos_st
12     return conj_ciclos                  // Devolve todos os ciclos gerados
```

Capítulo 4

Análise de Resultados

De forma a serem analisados os resultados, foram usadas 3 redes da SNDlib [9].

Rede	$\|N\|$	$\|A\|$	$2\|A\|/\|N\|$
polska	12	18	3
nobel-germany	17	26	3.06
nobel-eu	28	41	2.93

Tabela 4.1: Redes da SNDlib testadas

As simulações foram executadas num PC com Ubuntu 14.04 LTS, processador Intel Core 2 Quad CPU Q8200 2.33GHz×4, com 3.9 GB de RAM. Para a resolução dos modelos de programação linear inteira, foi usado o software IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.0.0.

A qualidade de um ciclo pode ser definida em função do seu comprimento e do número de nós que estão presentes nesse ciclo. Assim, foi calculado o tamanho dos ciclos através da expressão [8]:

$$circunferencia\_ciclo[p] = \sum_{j \in A: x_j^p = 1} (80km + c_j), \qquad \forall p \in P \tag{4.1}$$

onde se assume que para além do custo dos arcos, cada nó adiciona $80km$ equivalentes às perdas ópticas.

Foi feito um estudo com k (parâmetro do algoritmo 5) que determina o número total de ciclos candidatos utilizados nos modelos de optimização. Considerou-se $k = 3, 5, |N|$ para as redes polska, nobel-germany e nobel-eu. Note-se que a inclusão de um número elevado de ciclos aumenta a dimensão do problema, podendo levar a tempos de execução elevados, em particular, em redes de maior dimensão.

4.1 Análise dos resultados para $k = 3, 5, |N|$

Nas tabelas seguintes usa-se a seguinte notação:

- $\sum_{\forall j \in A} w_j$ – capacidade de trabalho total. No modelo WCO representa a função objectivo.
- $\sum_{\forall j \in A} a_j$ – capacidade de reserva total.
- $\sum_{j \in A} c_j a_j$ – custo da capacidade de reserva total. No modelo SCO representa a função objectivo.
- $\sum_{j \in A} c_j w_j$ – custo da capacidade de trabalho total.
- $\sum_{j \in A} c_j (a_j + w_j)$ – custo da capacidade total. No modelo JCO representa a função objectivo.
- p_g – número de ciclos gerados.
- p_u – número de ciclos usados pelos modelos de optimização.
- CPU – tempo de resolução dos modelos.

Os resultados obtidos podem ser vistos nas tabelas seguintes.

polska	$\sum_{\forall j \in A} w_j$	$\sum_{j \in A} c_j a_j$	$\sum_{j \in A} c_j w_j$	$\sum_{j \in A} c_j (a_j + w_j)$	p_g	p_u	CPU (s)
$k = 3$	21315	2.8931×10^6	3.7044×10^6	6.5976×10^6	53	8	0
$k = 5$	21315	2.8907×10^6	3.7044×10^6	6.5951×10^6	63	8	0
$k = \|N\|$	21315	2.8907×10^6	3.7044×10^6	6.5951×10^6	65	8	0

Tabela 4.2: Resultados rede polska para o modelo SCO

polska	$\sum_{j \in A} c_j a_j$	$\sum_{j \in A} c_j w_j$	$\sum_{j \in A} c_j (a_j + w_j)$	Nº caminhos	p_g	p_u	CPU (s)
$k = 3$	2.51360×10^6	3.7968×10^6	6.3104×10^6	660	53	8	0.04
$k = 5$	2.51357×10^6	3.7966×10^6	6.3102×10^6	660	63	7	0.03
$k = \|N\|$	2.51357×10^6	3.7966×10^6	6.3102×10^6	660	65	7	0.03

Tabela 4.3: Resultados rede polska para o modelo JCO

polska	$\sum_{\forall j \in A} a_j$	$\sum_{\forall j \in A} w_j$	p_g	p_u	redundância (%)	CPU (s)
$k = 3$	15826	29638	53	10	53.4	0
$k = 5$	15762	29574	63	8	53.3	0
$k = \|N\|$	15762	29574	65	9	53.3	0

Tabela 4.4: Resultados rede polska para o modelo WCO

nobel-germany	$\sum_{\forall j \in A} w_j$	$\sum_{j \in A} c_j a_j$	$\sum_{j \in A} c_j w_j$	$\sum_{j \in A} c_j (a_j + w_j)$	p_g	p_u	CPU (s)
$k = 3$	1552	2.2010×10^5	2.0165×10^5	4.2175×10^5	91	7	0
$k = 5$	1552	2.1861×10^5	2.0165×10^5	4.2026×10^5	109	8	0
$k = \|N\|$	1552	2.1861×10^5	2.0165×10^5	4.2026×10^5	132	8	0.01

Tabela 4.5: Resultados rede nobel-germany para o modelo SCO

nobel-germany	$\sum_{j \in A} c_j a_j$	$\sum_{j \in A} c_j w_j$	$\sum_{j \in A} c_j (a_j + w_j)$	Nº caminhos	p_g	p_u	CPU (s)
$k = 3$	1.2536×10^5	2.2419×10^5	3.4954×10^5	1210	91	6	0.13
$k = 5$	1.2668×10^5	2.2237×10^5	3.4905×10^5	1210	109	8	0.11
$k = \|N\|$	1.2588×10^5	2.2234×10^5	3.4822×10^5	1210	132	8	0.04

Tabela 4.6: Resultados rede nobel-germany para o modelo JCO

nobel-germany	$\sum_{\forall j \in A} a_j$	$\sum_{\forall j \in A} w_j$	p_g	p_u	redundância (%)	CPU (s)
$k = 3$	1776	3128	91	7	56.77	0
$k = 5$	1731	3037	109	8	57	0
$k = \|N\|$	1731	3037	132	10	57	0

Tabela 4.7: Resultados rede nobel-germany para o modelo WCO

nobel-eu	$\sum_{\forall j \in A} w_j$	$\sum_{j \in A} c_j a_j$	$\sum_{j \in A} c_j w_j$	$\sum_{j \in A} c_j (a_j + w_j)$	p_g	p_u	CPU (s)
$k = 3$	5812	2.3345×10^6	1.9939×10^6	4.3284×10^6	262	15	0.01
$k = 5$	5812	2.3103×10^6	1.9939×10^6	4.3042×10^6	380	14	0.01
$k = \|N\|$	5812	2.2668×10^6	1.9939×10^6	4.2607×10^6	1103	12	0.03

Tabela 4.8: Resultados rede nobel-eu para o modelo SCO

nobel-eu	$\sum_{j \in A} c_j a_j$	$\sum_{j \in A} c_j w_j$	$\sum_{j \in A} c_j (a_j + w_j)$	Nº caminhos	p_g	p_u	CPU (s)
$k = 3$	1.7491×10^6	2.0801×10^6	3.8291×10^6	3780	262	8	0.25
$k = 5$	1.7162×10^6	2.0880×10^6	3.8041×10^6	3780	380	7	0.25
$k = \|N\|$	1.5770×10^6	2.1075×10^6	3.6845×10^6	3780	1103	8	0.65

Tabela 4.9: Resultados rede nobel-eu para o modelo JCO

nobel-eu	$\sum_{\forall j \in A} a_j$	$\sum_{\forall j \in A} w_j$	p_g	p_u	redundância (%)	CPU (s)
$k = 3$	6031	8684	262	20	69.45	0.01
$k = 5$	5900	8746	380	15	67.46	0.01
$k = \|N\|$	5714	9235	1103	17	61.87	0.04

Tabela 4.10: Resultados rede nobel-eu para o modelo WCO

Comparando o modelo de optimização SCO com o modelo de optimização JCO para $k = 3, 5, |N|$, podemos ver que o custo da capacidade de reserva e o custo da capacidade total no caso do modelo JCO é menor em comparação com o modelo SCO, como seria esperado. Isto deve-se ao facto de o modelo de optimização JCO optimizar conjuntamente a capacidade de trabalho e a capacidade de reserva distribuindo, assim, as capacidades da melhor forma possível

Rede	Nº nós	Nº arcos	Nº ciclos gerados	Ciclo menor(Km)	Ciclo maior(Km)	Tamanho médio ciclos (Km)
polska	12	18	53	747.4	3358.16	2034.85
nobel-germany	17	26	91	493.226	3078.12	1882.2
nobel-eu	28	41	262	1454.86	9505.91	5142.99

Tabela 4.11: Tamanho dos ciclos gerados considerando $k = 3$

Rede	Nº nós	Nº arcos	Nº ciclos gerados	Ciclo menor(Km)	Ciclo maior(Km)	Tamanho médio ciclos(Km)
polska	12	18	63	747.4	3358.16	2126.51
nobel-germany	17	26	109	493.226	3185.1	2011.37
nobel-eu	28	41	380	1454.86	9752.93	5535.41

Tabela 4.12: Tamanho dos ciclos gerados considerando $k = 5$

Rede	Nº nós	Nº arcos	Nº ciclos gerados	Ciclo menor(Km)	Ciclo maior(Km)	Tamanho médio ciclos(Km)
polska	12	18	65	747.4	3358.16	2157.11
nobel-germany	17	26	132	493.226	3531.81	2160.18
nobel-eu	28	41	1103	1454.86	12806	7135.98

Tabela 4.13: Tamanho dos ciclos gerados considerando $k = |N|$

de maneira a reduzir os custos da capacidade total. No caso do modelo de optimização SCO, a capacidade extra que vai ser necessária adicionar para protecção vai depender do encaminhamento previamente seleccionado e da distribuição da capacidade de trabalho resultante, pelo que os custos da capacidade de reserva são mais elevados neste modelo. Em relação aos custos da capacidade de trabalho estes são mais elevados no modelo de optimização JCO em comparação com o modelo SCO para todas as redes testadas.

No que diz respeito aos tempos de CPU destes dois modelos podemos ver que o modelo de optimização JCO é aquele que demora mais tempo até que seja obtida uma solução. Esta é a grande desvantagem que este modelo tem face ao modelo de optimização da capacidade de reserva pois, à medida que as redes se tornam maiores, o tempo de CPU para que seja obtida uma solução pode ser bastante elevado.

Comparando o modelo SCO para os diferentes valores de k considerados, para as redes polska e nobel-germany, podemos ver que com $k = 3$ os custos da capacidade de reserva são os mais elevados e que, para $k = 5$ e $k = |N|$ os valores do custo da capacidade de reserva é o

mesmo, para estas mesmas redes. No caso da rede nobel-eu, quando $k = |N|$ o valor do custo da capacidade de reserva é ligeiramente menor, relativamente aos valores de $k = 3$ e $k = 5$ o que resulta do elevado (em termos relativos) número de ciclos candidatos quando $k = |N|$.

Relativamente ao modelo JCO para os diferentes valores de k considerados, podemos ver que os custos da capacidade de reserva e da capacidade total são menores quando é usado um $k = |N|$ nas redes nobel-germany (menos de 0.5%, em termos relativos) e nobel-eu (menos de 4%). No caso da rede polska, os custos são praticamente os mesmos para os diferentes valores de k considerados. Isto pode dever-se ao facto de o conjunto de ciclos gerados ser bastante próximo, o que não acontece com as redes restantes.

Dos resultados obtidos para o modelo de optimização WCO para $k = 3, 5, |N|$ podemos ver que este modelo consegue maximizar a capacidade de trabalho total que é protegida pelos *p-cycles*, quando é dada a capacidade de reserva total obtida através do modelo de optimização SCO. Isto torna o modelo WCO mais eficiente relativamente ao modelo SCO, como seria de esperar. Os tempos de CPU do modelo de optimização WCO são semelhantes relativamente ao modelo SCO.

Analisando o tamanho dos ciclos podemos ver que a rede nobel-eu é a rede onde o tamanho médio dos ciclos é superior, para todos os valores de k considerados, face às outras redes testadas. Por outro lado, a rede cujo comprimento médio dos ciclos é menor é a rede nobel-germany, no caso de $k = 3, 5$. Quando o $k = |N|$ a rede polska é a rede que apresenta um comprimento médio menor face às outras redes. Em termos práticos, no caso da rede nobel-eu como tem ciclos maiores, pode ser necessário usar mais repetidores que no caso das outras redes, de forma a ultrapassar as perdas por atenuação na fibra óptica e distorção do sinal. Dos resultados, podemos ainda verificar que o número de ciclos usados na solução dos problemas abordados, representa uma pequena percentagem face ao número de ciclos gerados inicialmente, em todas as redes testadas.

De toda a análise até aqui realizada, conclui-se que o número total de ciclos candidatos correspondente a $k = 5$ é um valor, em geral, suficientemente elevado uma vez que os resultados obtidos são muitas vezes superiores aos obtidos com $k = 3$ e raramente inferiores aos obtidos para $k = |N|$. Assim na secção seguinte os resultados são apresentados apenas para $k = 5$.

4.2 Escolha dos ciclos com base na sua eficiência a priori

A eficiência a priori $AE(p)$ de um *p-cycle* p, define-se como

$$AE(p) = \frac{\sum\limits_{\forall j \in A} x_j^p}{\sum\limits_{\forall j \in A | \pi_j^p = 1} c_j}, \tag{4.2}$$

onde se recorda que x_j^p é igual a 1 se o *p-cycle p* protege o arco j que passa sobre o ciclo, igual a 2 se o *p-cycle p* protege o arco *straddling* j e 0 nos restantes casos. O parâmetro c_j representa o custo do arco j. O numerador da equação mede o número total de relações de protecção que o *p-cycle p* consegue garantir e, o denominador dá-nos o custo total dos arcos que estão sobre o *p-cycle p*. Desta forma é possível calcular a eficiência potencial de se usar um *p-cycle p* para protecção.

Foram considerados p_g ciclos por ordem decrescente de AE(), entre os ciclos candidatos gerados como descrito anteriormente com $k = 5$, por forma a garantir que todos os arcos da rede estão protegidos. Este conjunto de ciclos foi então usado como o conjunto de ciclos candidatos nos modelos SCO e JCO cujos resultados podem ser vistos nas tabelas seguintes.

polska, $k = 5$	$\sum_{j\in A} c_j a_j$	$\sum_{j\in A} c_j w_j$	$\sum_{j\in A} c_j(a_j + w_j)$	p_g	p_u	CPU (s)
SCO	2.8907×10^6	3.7044×10^6	6.5951×10^6	63	8	0
SCO AE	3.8245×10^6	3.7044×10^6	7.5289×10^6	2	2	0

Tabela 4.14: Rede polska: comparação de resultados do modelo SCO sem e com métrica AE

polska, $k = 5$	$\sum_{j\in A} c_j a_j$	$\sum_{j\in A} c_j w_j$	$\sum_{j\in A} c_j(a_j + w_j)$	Nº caminhos	p_g	p_u	CPU (s)
JCO	2.5136×10^6	3.7966×10^6	6.3102×10^6	660	63	7	0.03
JCO AE	2.8309×10^6	3.7286×10^6	6.5595×10^6	660	2	2	1.1

Tabela 4.15: Rede polska: comparação de resultados do modelo JCO sem e com métrica AE

nobel-germany, $k = 5$	$\sum_{j\in A} c_j a_j$	$\sum_{j\in A} c_j w_j$	$\sum_{j\in A} c_j(a_j + w_j)$	p_g	p_u	CPU (s)
SCO	2.1861×10^5	2.0165×10^5	4.2026×10^5	109	8	0
SCO AE	2.7924×10^5	2.0165×10^5	4.8089×10^5	5	3	0

Tabela 4.16: Rede nobel-germany: comparação de resultados do modelo SCO sem e com métrica AE

nobel-germany, $k = 5$	$\sum_{j\in A} c_j a_j$	$\sum_{j\in A} c_j w_j$	$\sum_{j\in A} c_j(a_j + w_j)$	Nº caminhos	p_g	p_u	CPU (s)
JCO	1.2668×10^5	2.2237×10^5	3.4905×10^5	1210	109	8	0.11
JCO AE	1.3411×10^5	2.2183×10^5	3.5594×10^5	1210	5	3	0.16

Tabela 4.17: Rede nobel-germany: comparação de resultados do modelo JCO sem e com métrica AE

nobel-eu, $k = 5$	$\sum_{j\in A} c_j a_j$	$\sum_{j\in A} c_j w_j$	$\sum_{j\in A} c_j(a_j + w_j)$	p_g	p_u	CPU (s)
SCO	2.3103×10^6	1.9939×10^6	4.3042×10^6	380	14	0.01
SCO AE	2.7840×10^6	1.9939×10^6	4.7779×10^6	73	9	0

Tabela 4.18: Rede nobel-eu: comparação de resultados do modelo SCO sem e com métrica AE

nobel-eu, $k = 5$	$\sum_{j \in A} c_j a_j$	$\sum_{j \in A} c_j w_j$	$\sum_{j \in A} c_j (a_j + w_j)$	Nº caminhos	p_g	p_u	CPU (s)
JCO	1.7162×10^6	2.0880×10^6	3.8041×10^6	3780	380	7	0.25
JCO AE	2.0111×10^6	2.0606×10^6	4.0716×10^6	3780	73	6	0.29

Tabela 4.19: Rede nobel-eu: comparação de resultados do modelo JCO sem e com métrica AE

Rede	Nº nós	Nº arcos	Nº ciclos gerados	Ciclo menor(Km)	Ciclo maior(Km)	Tamanho médio ciclos (Km)
polska	12	18	2	2841.52	3161.74	3001.63
nobel-germany	17	26	5	493.226	3078.12	2462.56
nobel-eu	28	41	73	2126.23	8226.73	5000.91

Tabela 4.20: Tamanho dos ciclos gerados considerando a métrica AE

Dos resultados obtidos para um valor de $k = 5$, tendo em conta a métrica da eficiência a priori, verificou-se que os resultados dos modelos SCO e JCO são piores, relativamente ao caso em que não era usada qualquer métrica para seleccionar os ciclos candidatos. Uma possível razão para esta pioria de resultados, pode dever-se ao facto de a métrica AE() estar a escolher os melhores ciclos de um subconjunto restrito de ciclos, i.e., a métrica não está a escolher os melhores ciclos do conjunto de todos os ciclos candidatos para as redes testadas. Além disso, provavelmente o valor p_g utilizado precisaria de ser aumentado para incluir mais ciclos candidatos escolhidos sempre por ordem decrescente de AE(). A determinação de um possível valor adequado para p_g é deixado para trabalho futuro.

No que concerne ao tamanho dos ciclos, a métrica da eficiência a priori usa, em geral, um tamanho médio de ciclos superior face aos resultados observados anteriormente, quando não é usada qualquer métrica. Isto acontece porque a métrica da eficiência a priori (AE) vai escolher o menor número de ciclos candidatos de forma a proteger todos os arcos da rede, por ordem de eficiência, pelo que poderá escolher ciclos com o maior número de nós o que leva a que o tamanho dos ciclos neste caso seja, em geral, maior comparativamente ao caso em que não é usada qualquer métrica. A rede em que o tamanho médio dos ciclos é menor é na rede nobel-germany e, em contrapartida, a rede que tem o tamanho médio de ciclos maior é a rede nobel-eu. Contudo, verifica-se que no caso da rede nobel-germany o menor ciclo é o mesmo que no caso em que não é aplicada qualquer métrica para $k = 5$.

Capítulo 5

Conclusão

Nesta dissertação foi estudado o mecanismo de protecção em redes de telecomunicações baseado nos *p-cycles* de protecção ao arco, onde se implementaram três modelos de programação linear inteira que resolvem três problemas distintos. A abordagem feita a estes problemas teve em consideração um número limitado de ciclos pois, à medida que as redes se tornam maiores, os modelos de programação linear inteira podem gerar ficheiros muito longos, que demorariam mais tempo a ser resolvidos. Estes problemas fazem, portanto, parte de problemas NP-hard.

Dos resultados obtidos verificou-se que o modelo que optimiza a capacidade conjunta (JCO) é o modelo que melhor distribui a capacidade de reserva e de trabalho para todas as redes testadas comparativamente ao modelo de optimização da capacidade de reserva (SCO), como seria de esperar. Em relação ao modelo que optimiza a capacidade de trabalho (WCO) que é protegida pelos *p-cycles*, este consegue maximizar a capacidade de trabalho total que é protegida pelos *p-cycles*, quando são dadas as capacidades de reserva de cada arco obtidas pelo modelo de optimização SCO.

Ao aplicar a métrica da eficiência à priori aos modelos de optimização SCO e JCO, verificou-se que eram obtidos resultados piores relativamente aos custos das capacidades, quando foram considerados p_g ciclos por ordem decrescente de AE() entre os ciclos candidatos com $k = 5$. A eficiência à priori não mostrou ser um bom critério de escolha dos ciclos, quando a sua escolha está limitada pelo número p_g e está restringida a um subconjunto restrito de ciclos gerados.

Do trabalho desenvolvido pode concluir-se que optimizar conjuntamente a capacidade de trabalho e a capacidade de reserva é melhor, uma vez que os custos relativos à capacidade de reserva e à capacidade total são menores em comparação com o caso em que apenas é optimizada a capacidade de reserva. O inconveniente é que à medida que o conjunto de ciclos candidatos aumenta, o tempo para que seja obtida uma solução é maior. Por outro lado, o custo da capacidade de trabalho é mais elevado no modelo que optimiza a capacidade conjunta (JCO). Mas, o que se pretende quando se quer proteger uma rede, é gastar o menos possível em protecção pelo que, o modelo que optimiza a capacidade conjunta é o modelo com mais

interesse dos três modelos implementados.

Relativamente ao trabalho futuro, uma possibilidade seria encontrar um valor adequado para p_g de forma a aumentar o número de ciclos candidatos para serem usados na métrica AE.

Bibliografia

[1] R. Asthana, Y.N. Singh, and W.D. Grover. p-cycles: An overview. *Communications Surveys Tutorials, IEEE*, 12(1):97–111, First 2010.

[2] P. Cholda, A. Mykkeltveit, B.E. Helvik, O.J. Wittner, and A. Jajszczyk. A survey of resilience differentiation frameworks in communication networks. *Communications Surveys Tutorials, IEEE*, 9(4):32–55, Fourth 2007.

[3] J. Doucette, P.A. Giese, and W.D. Grover. Combined node and span protection strategies with node-encircling p-cycles. In *Design of Reliable Communication Networks, 2005. (DRCN 2005). Proceedings.5th International Workshop on*, pages 9 pp.–, Oct 2005.

[4] Hai Anh Hoang and B. Jaumard. A new flow formulation for fipp p-cycle protection subject to multiple link failures. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2011 3rd International Congress on*, pages 1–7, Oct 2011.

[5] B. Jaumard and Honghui Li. Segment p-cycle design with full node protection in wdm mesh networks. In *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, pages 1–6, Oct 2011.

[6] A.Z. Kasem, R. Gallardo, and J. Doucette. An enhanced ilp design model for node-encircling p-cycle networks. In *Design of Reliable Communication Networks (DRCN), 2014 10th International Conference on the*, pages 1–7, April 2014.

[7] A. Metnani and B. Jaumard. Stability of fipp p -cycles under dynamic traffic in WDM networks. *Networking, IEEE/ACM Transactions on*, 21(2):413–425, April 2013.

[8] D.P. Onguetou and W.D. Grover. p-cycle network design: From fewest in number to smallest in size. In *Design and Reliable Communication Networks, 2007. DRCN 2007. 6th International Workshop on*, pages 1–8, Oct 2007.

[9] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski. Sndlib 1.0—survivable network design library. *Networks*, 55(3):276–286, 2010.

[10] D. Stamatelakis and W.D. Grover. Theoretical underpinnings for the efficiency of restorable networks using preconfigured cycles (ldquo;p-cycles rdquo;). *Communications, IEEE Transactions on*, 48(8):1262–1265, Aug 2000.

[11] Jean-Philippe Vasseur, Mario Pickavet, and Piet Demeester. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[12] Bin Wu, K.L. Yeung, and Pin han Ho. Ilp formulations for p -cycle design without candidate cycle enumeration. *Networking, IEEE/ACM Transactions on*, 18(1):284–295, Feb 2010.

[13] J.Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, pages 712–716, 1971.