UNIVERSIDADE D
COIMBRA

Rúben Miguel Rodrigues Gregório

SHORTEST PATHS IN GRAPHS OF CONVEX SETS

Dissertação no âmbito do Mestrado em Matemática, Ramo Estatística, Otimização e Matemática Financeira orientada pelo Professor Doutor João Gouveia e apresentada ao Departamento de Matemática da Faculdade de Ciências e Tecnologia.

Setembro de 2023

Caminhos mais curtos em grafos de conjuntos convexos

Rúben Miguel Rodrigues Gregório

UNIVERSIDADE D
COIMBRA

Mestrado em Matemática

Master in Mathematics

Dissertação de Mestrado | MSc Dissertation

Setembro 2023

Agradecimentos

Em primeiro lugar quero agradecer ao meu orientador Professor Doutor João Gouveia pela disponiblidade, ajuda e dedicação durante estes meses. Foi sem dúvida um privilégio trabalhar consigo.

Aos meus pais um agradecimento especial, pelo apoio incondicional e confiança depositada em mim ao longo destes anos. Festejaram as minhas vitórias como se fossem deles e ergueram-me nos momentos mais difíceis. Ao meu irmão que de uma maneira ou de outra tentava distrair-me do stress e da ansiedade diária.

À minha namorada e melhor amiga que esteve sempre ao meu lado nos bons e maus momentos.

Por fim, aos amigos que Coimbra me deu a conhecer, sem eles este percurso não teria o mesmo significado.

Resumo

O problema do caminho mais curto consiste em, dado um grafo, encontrar uma sequência de arestas de forma a conectar um vértice de origem a um vértice de destino com o menor custo possível. Uma generalização deste problema clássico surge quando se define a posição de cada vértice do grafo como sendo uma variável de decisão contínua que se encontra restrita a um determinado conjunto convexo. Deixa-se assim de ter posições fixas para os vértices do grafo. Posto isto, o comprimento de uma aresta determina-se segundo uma função convexa das posições dos vértices que esta liga. Esta vertente do problema apresenta uma vasta aplicabilidade, nomeadamente, no planeamento do movimento de veículos autónomos e na navegação ao nível da robótica. No entanto, possui uma enorme complexidade, revelando-se **NP**-difícil. A demonstração desta ilação constituirá um dos alvos deste estudo. Além disso, estudar-se-á este problema com o intuito de apresentar uma forte formulação convexa inteira mista baseada em funções perspetiva, que posteriormente será relaxada, tornando assim possível encontrar caminhos globalmente ótimos de forma eficiente em grafos e espaços de grandes dimensões. Para terminar, tirar-se-ão conclusões atendendo aos resultados da implementação computacional da formulação desenvolvida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

No nosso dia-a-dia somos constantemente confrontados com a falta de tempo, causada quer pela falta de organização quer pela carga excessiva de compromissos. De forma a contrariar tal tendência, deparamo-nos, por exemplo, a procurar trajetos com menor duração. Ou seja, todos os dias e na maior parte deles sem nos apercebemos, estamos a resolver intuitivamente problemas de caminho mais curto (PCMC). Daí se tratar de um problema tão fundamental e um dos mais estudados em otimização combinatória.

Na sua forma mais geral, este problema procura, dado um grafo, o caminho mais curto entre dois vértices desse mesmo grafo. Assim, o comprimento do caminho resulta na soma do comprimento das arestas percorridas que, normalmente são escalares fixos previamente definidos.

Contudo, o alvo deste estudo recai numa generalização deste PCMC. Ao contrário do que foi indicado anteriormente, agora temos um problema onde os vértices e os comprimentos das arestas não são fixos. De modo mais preciso, temos um grafo onde cada vértice está emparelhado com um conjunto convexo (denominamos este tipo de grafo por grafo de conjuntos convexos). Assim, as posições de cada vértice passam a ser variáveis de decisão contínua que se encontram restritas pelo respetivo conjunto convexo e, por sua vez, os comprimentos das arestas passam a ser definidos por uma função convexa (por exemplo, a distância euclidiana) das posições dos vértices que estas conectam [6].

Esta versão do PCMC aplica-se em diversas áreas, como por exemplo, no controlo dos voos de drones [6] e no planeamento do movimento de robots [2]. Relativamente ao primeiro cenário, podemos interpretá-lo como o voo de um drone de uma região de partida para uma região de destino tendo, evidentemente, o objetivo de minimizar o comprimento total do voo. De forma a impedir que o drone viage diretamente da região de partida para a região de destino, utilizam-se restrições de autonomia. Assim o drone é obrigado a parar ao longo do caminho em áreas próprias (regiões convexas) para efetuar a recarga. Para evitar transições entre certas regiões basta incorporar tais restrições no grafo correspondente ao problema. No que diz respeito ao planeamento do movimento de robots, tem-se como objetivo posicionar os passos destes com o intuito de caminhar em terrenos irregulares e com obstáculos. Para evitar os obstáculos é necessário decompor o ambiente em regiões convexas livres de obstáculos. Assim, cada passo é atribuído a uma dessas regiões seguras.

Ao contrário do que acontece com o PCMC, para o qual existem algoritmos eficientes para a sua resolução, o problema do caminho mais curto em grafos de conjuntos convexos (PCMCGCC)

é **NP**-difícil, pelo que não se espera encontrar um algoritmo que o resolva um tempo polinomial. Desta forma, ir-se-á relaxar a condição relativa à resolução em tempo polinomial e desenvolver-se-á o PCMCGCC como um programa convexo inteiro misto (PCIM). Para formular o PCIM, recorrer-se-á a uma nova relaxação, tal como ir-se-á ver mais à frente. Este será o tema principal deste estudo, contudo, antes disso começar-se-á pela definição de alguns conceitos básicos relativos às teorias de grafos [4], da convexidade [6, 9] e da complexidade [3]. Para terminar, o último tópico a ser tratado consistirá na realização de testes numéricos atendendo à implementação computacional da formulação desenvolvida.

Capítulo 2

Conceitos Básicos

Ao longo deste capítulo serão abordados alguns conceitos fundamentais para o desenvolvimento do trabalho. Para tal, começar-se-á pela definição de conceitos básicos alusivos à teoria dos grafos, presente em [4], seguindo-se da exposição de noções referentes à teoria da convexidade [6, 9]. Para terminar, tratar-se-á da questão relativa à complexidade dos problemas [3].

2.1 Teoria dos Grafos

Relativamente à Teoria dos Grafos, é fundamental começar com a definição de grafo. Um grafo é um par (V,E), onde V representa um conjunto finito, cujos elementos são denominados vértices e E corresponde um conjunto de pares de vértices a que chamamos arestas. Partindo desta definição, surgem novos conceitos. Dado um grafo, um caminho é uma sequência de vértices $(v_1, v_2, ..., v_t)$ tal que $(v_i, v_{i+1}) \in E$ para $i = 1, ..., t-1$. Se tal caminho não repetir vértices denomina-se de caminho simples. Um grafo diz-se conexo se para todo o par de vértices existir um caminho que os ligue. Um ciclo é nada mais nada menos que um caminho, com pelo menos três vértices, que começa e termina no mesmo vértice. Indo ao encontro da ideia transmitida anteriormente, se tal ciclo não repetir mais nenhum vértice passa a apelidar-se de ciclo simples. Caso não existam ciclos simples num grafo, este é designado de acíclico. De forma a ilustrar as definições mencionadas anteriormente, analisar-se-á o grafo representado no diagrama abaixo.

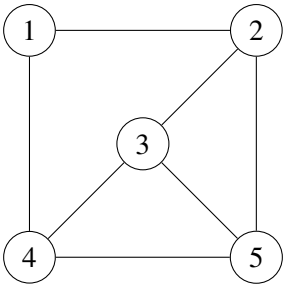


Fig. 2.1 Grafo com cinco vértices e sete arestas.

Exemplo 1. Trata-se de um grafo com $V = \{1,2,3,4,5\}$ e $E = \{\{1,2\},\{1,4\},\{2,3\},\{2,5\},\{3,4\},\{3,5\},\{4,5\}\}$, conexo e não acíclico. Neste, (2,3,4,5) é um caminho simples, contudo o mesmo não

se pode dizer de (2,3,4,5,3) que, apesar de ser um caminho, não é simples. Relativamente aos ciclos, pode-se constatar que (1,2,3,4,1) é um ciclo simples, enquanto que (5,3,4,1,2,3,5) é um ciclo, mas não simples.

Um grafo orientado ou digrafo é um par (*V*,*E*), onde, tal como num grafo, *V* representa um conjunto finito, cujos elementos são denominados vértices. A diferença do digrafo para o grafo reside em *E*. Num grafo orientado este conjunto é constítuido por pares de vértices ordenados, nominados por arcos ou arestas orientadas. Num digrafo, define-se cadeia como sendo uma sequência $(v_1, a_1, v_2, a_2, ..., v_{t-1}, a_{t-1}, v_t)$, onde $v_i \in V$ e $a_i \in E$. É de notar que as arestas pertencentes à sequência têm de ligar, evidentemente, os vértices apresentados à sua esquerda com os da direita. Por outras palavras, $a_i = (v_i, v_{i+1})$ ou $a_i = (v_{i+1}, v_i)$. Se ocorrer a primeira situação, diz-se que o arco orienta-se no sentido direto, caso contrário orienta-se no sentido inverso. Se se falar de circuito está-se a referir a uma cadeia onde o vértice inicial é igual ao final. E mais, se não acontecer a repetição de outro vértice tal circuito denomina-se simples. Denomina-se por caminho uma cadeia onde todos os arcos encontram-se orientados no sentido direto. Indo ao encontro do que definimos para os grafos, num digrafo se não ocorrer a repetição de vértices, diz-se que uma cadeia ou um caminho são simples. A definição de ciclo é idêntica à de circuito, trocando apenas cadeia por caminho. Em jeito de exemplo, proceder-se-á à avaliação do digrafo representado graficamente a seguir.

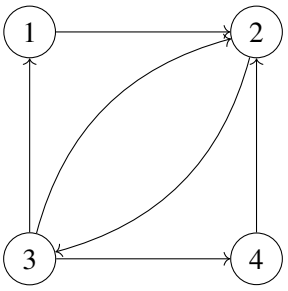


Fig. 2.2 Digrafo composto por quatro vértices e seis arestas.

Exemplo 2. Este é um digrafo com $V = \{1,2,3,4\}$ e $E = \{\{1,2\},\{2,3\},\{3,1\},\{3,2\},\{3,4\},\{4,2\}\}$. Neste, (3,(2,3),2,(4,2),4) é uma cadeia mas não um caminho, enquanto que (3,(3,1),1,(1,2),2) é um caminho e simples. No que diz respeito à definição de ciclo, tem-se que (3,(3,1),1,(1,2),2,(2,3),3) retrata um ciclo simples. Para finalizar, (3,(3,1),1,(1,2),2,(3,2),3) corresponde a um circuito simples.

2.2 Teoria da Convexidade

Quando se fala de Otimização surge, com naturalidade, o conceito de convexidade tanto em conjuntos definidos em $\mathbb{R}^n$ como em funções reais. Deste modo, ir-se-à proceder, primeiramente, à exposição de algumas definições e propriedades de conjuntos convexos, fazendo o mesmo, de seguida, no que diz respeito às funções convexas. Para terminar, serão introduzidas noções referentes à análise convexa.

Antes de apresentar a definição de conjunto convexo é essencial conhecer a noção de segmento linear. Sendo assim, de forma a tornar claro o conceito de conjunto convexo, temos que um segmento linear entre dois pontos (x e y) é o conjunto

$$[x,y] = \{\alpha x + (1-\alpha)y : \alpha \in [0,1]\}$$

tal que, x e $y \in S$, sendo S um subconjunto de $\mathbb{R}^n$. Por conseguinte, quando se considera um conjunto $S \subset \mathbb{R}^n$, este diz-se convexo se e só se $\forall x,y \in S, [x,y] \in S$. Por convenção, admite-se que um conjunto vazio é convexo. Abaixo, encontra-se uma exemplificação do que foi definido.

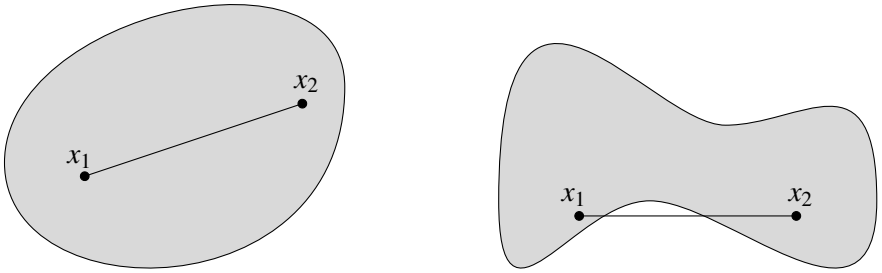


Fig. 2.3 Conjuntos convexo e não convexo, respetivamente.

Consideremos agora, além do conjunto S, $D = \{x_1, x_2, ..., x_k\}$ um conjunto finito de pontos de S. Um ponto $z \in \mathbb{R}^n$ é uma combinação convexa de pontos em D se e só se existem escalares α_i não negativos, verificando $\sum_{i=1}^{k} \alpha_i = 1$ tal que $z = \sum_{i=1}^{k} a_i x_i$. Nas mesmas condições, o invólucro convexo de D é o conjunto de todas as combinações convexas de pontos em D, isto é,

$$conv(\{x_1, ..., x_k\}) = \left\{ \sum_{i=1}^{k} a_i x_i : \sum_{i=1}^{k} a_i = 1 \wedge a_i \geq 0 \right\}.$$

Com o intuito de tornar a compreensão mais clara, apresentam-se os seguintes exemplos.

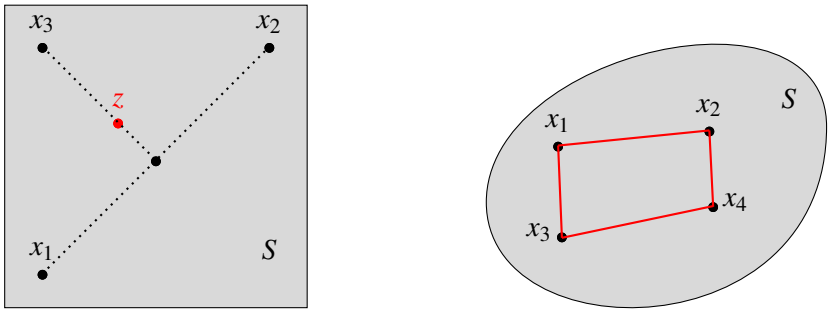


Fig. 2.4 Combinação convexa (à esquerda) e invólucro convexo (à direita).

Da primeira figura retira-se que o ponto z é uma combinação convexa de pontos em $D = \{x_1, x_2, x_3\}$, enquanto que na segunda pode-se constatar que a região delimitada pelas retas a vermelha constitui o invólucro convexo de $D = \{x_1, x_2, x_3, x_4\}$. Relativamente às propriedades, temos que a interseção de conjuntos convexos (finita ou infinita) é um conjunto convexo. Todavia, a reunião destes conjuntos pode não ser convexa. Partindo da definição de invólucro convexo, se S for convexo então $conv(D) \subset S$.

A noção de separação compreende um dos conceitos mais férteis da teoria da convexidade. Nomeadamente, um resultado importante encontra-se transcrito no teorema do hiperplano separador.

Contudo, torna-se permaturo abordar este teorema sem falar do conceito de hiperplano. Um hiperplano H é um conjunto na forma $H := \{x : a^T x = b\}$, onde $a \in \mathbb{R}^n$, $a \neq 0$ e $b \in \mathbb{R}$. Geometricamente, pode-se interpretar este conjunto como um hiperplano com vetor normal a, ficando a constante b responsável pela deslocação do mesmo em relação à origem. Atendendo à descrição deste conjunto, facilmente se constata que um hiperplano em $\mathbb{R}^n$ divide $\mathbb{R}^n$ em duas partes, sendo estas apelidadas de semi-espaços. De forma a ilustrar estas noções segue-se o exemplo 2.5 retirado de [1].

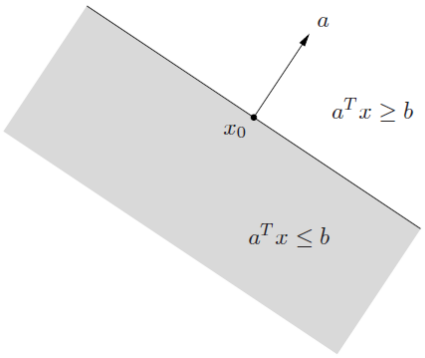


Fig. 2.5 O Hiperplano representado divide $\mathbb{R}^2$ em dois semi-espaços. O semi-espaço que se encontra sombreado é determinado por $a^T x \leq b$ e estende-se segundo a direção $-a$. Por sua vez, o outro semi-espaço, determinado por $a^T x \geq b$, alonga-se na direção a.

Dado um conjunto $S \subseteq \mathbb{R}^n$, a interseção de todos os semi-espaços que contêm S resulta no invólucro convexo deste conjunto.

Relativamente ao teorema do hiperplano separador, cuja prova pode ser consultada em [1], tem-se que este enuncia, de forma sucinta, a existência de um hiperplano que separa dois conjuntos convexos e disjuntos. Por outras palavras, assumindo que C e D representam conjuntos convexos, disjuntos e não vazios, então existem $a \neq 0$ e b tais que $a^T x \leq b, \forall x \in C$ e $a^T x \geq b, \forall x \in D$. Além disso, o hiperplano $\{x : a^T x = b\}$ é apelidado de hiperplano separador para os conjuntos C e D. Isto é ilustrado na figura 2.6, retirada de [1].

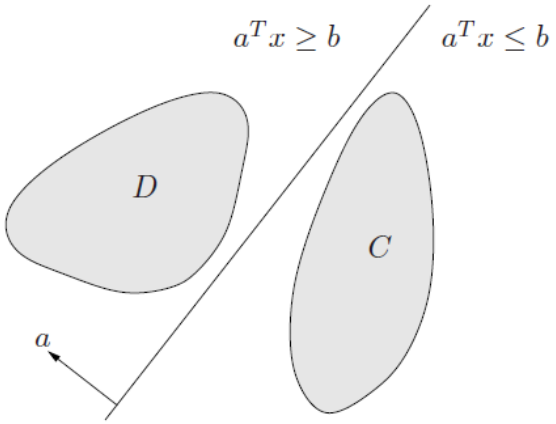


Fig. 2.6 O Hiperplano representado separa os conjuntos C e D.

Antes de se passar à definição do conceito de função convexa, é imprescendível apresentar algumas noções básicas. Posto isto, começar-se-á pela noção de cone. Esta designação refere-se a um conjunto $S \in \mathbb{R}^n$ se, e só se, para todo o escalar positivo α e para todo o x pertencente a S se tem $\alpha x \in S$. No que concerne a este termo, temos que a interseção e reunião de cones forma um cone e que nem todos os cones são convexos. Considerando novamente S, o cone polar deste conjunto representa-se por S° e determina-se da seguinte maneira:

$$S^\circ = \{y \in \mathbb{R}^n : x^T y \leq 0, \forall x \in S\}.$$

S° é sempre convexo e fechado, mesmo que S não o seja. A fim de ilustrar o que foi dito, apresenta-se, de seguida, um exemplo retirado de [8].

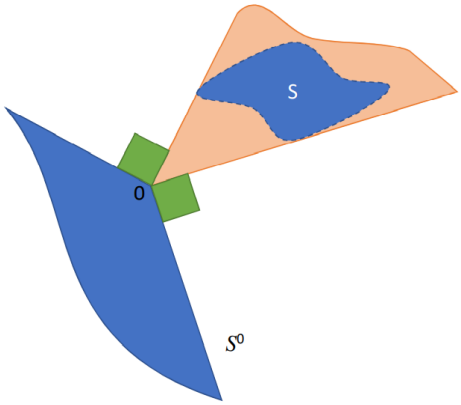


Fig. 2.7 Cone e cone polar de S.

Por sua vez, retomando o conjunto S, entende-se por cone dual de S o conjunto

$$S^* = \{y \in \mathbb{R}^n : x^T y \geq 0, \forall x \in S\}.$$

Naturalmente apercebe-se que o cone dual é igual à negação do cone polar, ou seja, que $S^* = -S^\circ$. Simultaneamente ao cone polar, o cone dual S^* é sempre convexo mesmo que S não o seja.

Neste instante, estão reunidas as condições para se definir o conceito de função convexa num dado conjunto. Posto isto, considerando, de novo, o conjunto S mas desta vez com a particularidade de ser convexo, a função $f : S \to \mathbb{R}$ é convexa em S se e só se

$$\forall \alpha \in [0,1], \forall x, y \in S, f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y).$$

O gráfico 2.8 retrata um exemplo de uma função convexa.

Para terminar, se f é uma função convexa num conjunto S, igualmente convexo, então para todo o $k \in \mathbb{R}$ o subconjunto $S_k = \{x \in S : f(x) \leq k\}$ é também convexo. Esta propriedade descreve uma relação importante entre funções convexas e conjuntos convexos.

No que concerne à análise convexa, é natural formar um cone convexo de $n+1$ dimensões partindo de um conjunto convexo de n dimensões. Este processo é algumas vezes denominado de homogeneização e, por outras palavras, consiste em aplicar a definição de cone mencionada

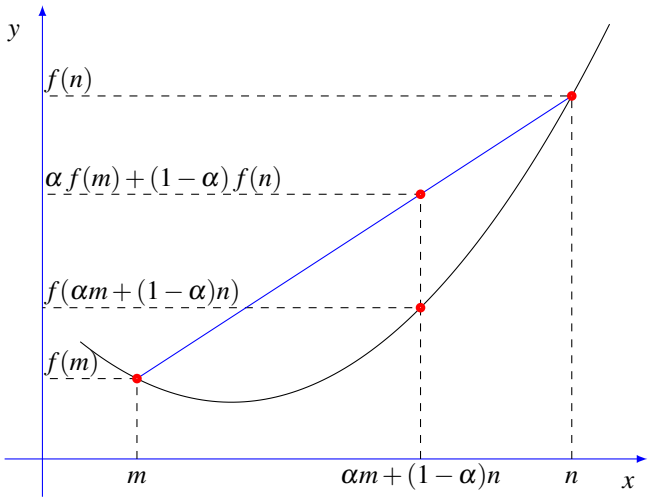


Fig. 2.8 Exemplo de uma função convexa.

anteriormente exigindo apenas que o conjunto a ser trabalhado seja convexo. De forma a seguir a linguagem utilizada em [6], denomine-se este processo de perspetiva. Posto isto, sendo $S \subset \mathbb{R}^n$ um conjunto convexo e compacto, define-se perspetiva do conjunto S como sendo o conjunto

$$\tilde{S} := \{(x,\lambda) \in \mathbb{R}^{n+1} : \lambda \geq 0, x \in \lambda S\}.$$

Esta operação preserva a convexidade, uma vez que facilmente se verifica que $\tilde{S}$ é um cone fechado e convexo. A fim do ilustrar esta definição, considere-se o Exemplo 3.

Exemplo 3. Seja $S = \{x :|| x ||\leq 1\}$. Para $\lambda > 0, \lambda S = \{\lambda x :|| x ||\leq 1\} = \{x :|| x/\lambda ||\leq 1\} = \{x :|| x ||\leq \lambda\}$. Caso $\lambda = 0$, tem-se $0S = \{0\} = \{x :|| x ||\leq 0\}$. Resumidamente, conclui-se que $\tilde{S} = \{(x,\lambda) :|| x ||\leq \lambda\}$.

Aplicando de forma análoga o procedimento abordado mas agora em funções, surge a perspetiva de uma função. Seja $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ uma função convexa e fechada (uma função $f : \mathbb{R} \to \mathbb{R}$ é fechada se para $\alpha \in \mathbb{R}$, $\{x \in domf \mid f(x) \leq \alpha\}$ é um conjunto fechado) e $\bar{x} \in \mathbb{R}^n$ um ponto qualquer tal que $f(\bar{x})$ é finito, define-se a perspetiva da função f como

$$\tilde{f}(x,\lambda) := \begin{cases} \lambda f(x/\lambda), & \text{se} \lambda > 0 \\ \lim_{\tau \to 0} \tau f(\bar{x} + x/\tau), & \text{se} \lambda = 0, \\ \infty, & \text{se} \lambda < 0 \end{cases}$$

onde o valor do limite pode ser mostrado como independente do ponto $\bar{x}$. Tal como a operação anterior, esta também preserva a convexidade, logo a função $\tilde{f}(x,\lambda)$ é conjuntamente convexa em x e λ. Para exemplificar a noção apresentada, atente-se ao Exemplo 4.

Exemplo 4. Seja $f(x) =|| x ||$. Quando $\lambda > 0$, $\tilde{f}(x,\lambda) = \lambda \,||\, x/\lambda \,||= \quad =|| x ||$. No caso de $\lambda = 0$, define-se $\bar{x} = 0$ e assim $\tilde{f}(x,\lambda) = \lim_{\tau \to 0} \tau \,||\, x/\tau \,||= \lim_{\tau \to 0} ||\, x \,||=|| x ||$. Deste modo, a perspetiva

da função f é definida como

$$\tilde{f}(x,\lambda) := \begin{cases} ||\, x\,||, & \text{se}\,\lambda \geq 0 \\ \infty, & \text{se}\,\lambda < 0. \end{cases}$$

Para terminar resta apresentar a noção de cone das desigualdades válidas de um dado conjunto. Por conseguinte, dado um conjunto $S \subset \mathbb{R}^n$, este define-se como sendo o conjunto

$$S^\bullet := \{(a,b) \in \mathbb{R}^{n+1} : a^T x + b \geq 0, \quad \forall x \in S\}.$$

A notação utilizada nesta definição difere da lidada em [6], visto que esta coincidia com a noção de cone polar apresentada anteriormente. Aliás, um facto interessante reside na observação de que o cone das desigualdades válidas está intimamente relacionado com o cone polar e, naturalmente, com o cone dual. No entanto, ao contrário desses conjuntos, o cone das desigualdades válidas vive no espaço de dimensão $n+1$. O lema que se exibe de seguida permite descrever a perspetiva do conjunto S em termos das desigualdades válidas do mesmo.

Lema 1. *Seja S um conjunto convexo e fechado. Pode-se descrever a sua perspetiva da seguinte maneira: $\tilde{S} = \{(x,\lambda) : a^T x + b\lambda \geq 0, \forall (a,b) \in S^\bullet\}$*

Demonstração: Em primeiro lugar, ir-se-á descrever o conjunto S como a interseção das suas desigualdades válidas. Para isso, recorrer-se-á ao Teorema 11.5 de [7], que enuncia que um conjunto convexo e fechado é a interseção dos semi-espaços que o contêm. Tomando em consideração este teorema e a atendendo à definição de $S^\bullet$, conclui-se naturalmente que se pode descrever S como $S = \{x : a^T x + b \geq 0, \forall (a,b) \in S^\bullet\}$. De seguida, determina-se a perspetiva deste conjunto. Para $\lambda \geq 0$:

$$\lambda S = \{\lambda x : a^T x + b \geq 0, \forall (a,b) \in S^\bullet\}$$

$$= \{x : a^T x + b\lambda \geq 0, \forall (a,b) \in S^\bullet\}$$

Ou seja, $\tilde{S} = \{(x,\lambda) : a^T x + b\lambda \geq 0, \forall (a,b) \in S^\bullet\}$, provando-se o pretendido. Note-se que nesta descrição a condição $\lambda \geq 0$ não precisa de ser explicita desde que $(0,1) \in S^\bullet$.

□

Os conceitos referentes à analise convexa constituem as principais ferramentas técnicas que irão ser utilizadas na elaboração e análise do PCIM, como se verá mais à frente.

2.3 Teoria da Complexidade

A Teoria da Complexidade é o ramo da ciência da computação talhado a classificar problemas de acordo com a sua complexidade. A complexidade de um problema é determinada em função do tempo de execução e da memória utilizada por parte do algoritmo capaz de o resolver. Resumidamente, para classificar um problema procede-se com o estudo da eficiência de um algoritmo dotado a solucioná-lo.

Uma vez que a um dado problema podem estar associados vários algoritmos que asseguram a obtenção da sua solução, é conveniente escolher o mais eficiente, isto é, aquele que resolve o problema

de forma mais rápida e com a menor utilização de memória. No entanto, a tarefa de determinar com rigor o tempo de execução de um algoritmo é difícil ou até mesmo impossível. Para contornar este problema, a solução passa por majorar o número de operações elementares que o algoritmo executará. Desta forma, o algoritmo é analisado no seu pior cenário, fornecendo um limite superior para o seu tempo de execução. A métrica descrita corresponde à notação O, também conhecida por "Big O". Esta notação é representada pela letra O seguida de uma função matemática, que descreve o comportamento assintótico do algoritmo, ou seja, como o tempo de execução ou o uso de memória aumenta à medida que o tamanho do problema cresce.

Os exemplos mais comuns desta notação correspondem a $O(1), O(\log(n)), O(n)$ e $O(n^2)$. Estes encontram-se por ordem crescente de "esforço", como se verifica na figura 2.9, sendo assim mais eficiente, por exemplo, um algoritmo da classe $O(\log(n))$ do que um da classe $O(n^2)$. É de notar que um algoritmo resolve o problema em tempo polinomial quando a sua complexidade é $O(n^k)$ para $k \in \mathbb{N}$.

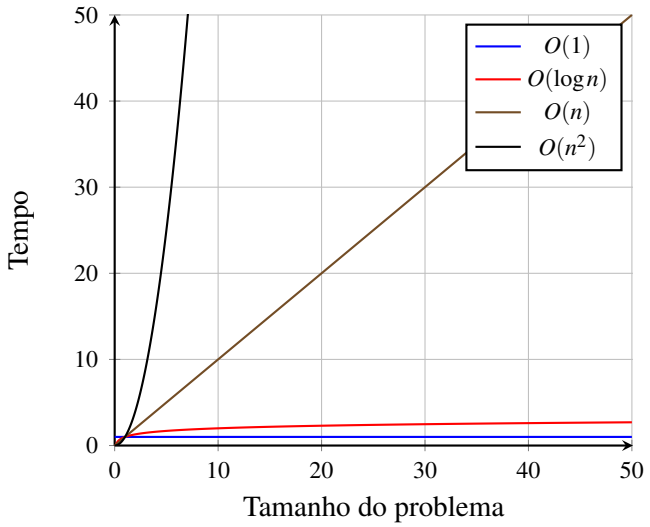


Fig. 2.9 Representação de alguns exemplos da notação O

Atendendo à sua complexidade, os problemas são divididos em diversas classes. Começando com a classe **P**, tem-se que esta engloba os problemas que podem ser resolvidos em tempo polinomial. Ou seja, reúne os problemas cujo tempo necessário para os solucionar cresce de forma razoável à medida que o tamanho do problema aumenta. A soma de números inteiros corresponde a um problema desta natureza. Por sua vez, a classe **NP** é composta pelos problemas que podem ser verificados em tempo polinomial, mas não necessariamente resolvidos no mesmo período de tempo. Por outras palavras, isto significa que dada uma solução de um problema desta classe só é possível verificar a veracidade desta em tempo polinomial. As classes **NP**-completo e **NP**-difícil abrangem os problemas mais difíceis da classe **NP**, isto é, compreendem problemas considerados intratáveis visto que não existem algoritmos conhecidos que os consigam resolver. Mais ainda, um problema é considerado **NP**-completo se for **NP** e se puder ser uma redução de um qualquer problema **NP**. Com isto tem-se que caso seja descoberto um algoritmo para resolver um problema **NP**-completo então todos os problemas **NP** serão solucionados por um algoritmo eficiente. A classe **NP**-difícil contém problemas que são definidos, informalmente, por serem pelo menos tão difíceis quanto os problemas da classe **NP**-completo. No

entanto, esta classe pode não estar absolutamente em **NP**, uma vez que, embora engloba problemas que são pelo menos tão difíceis quanto os problemas da classe **NP**-completo, não tem necessariamente a propriedade de verificação em tempo polinomial.

A fim de cimentar o que foi proferido apresenta-se a figura 2.10 que retrata as relações entre as classes de problemas.

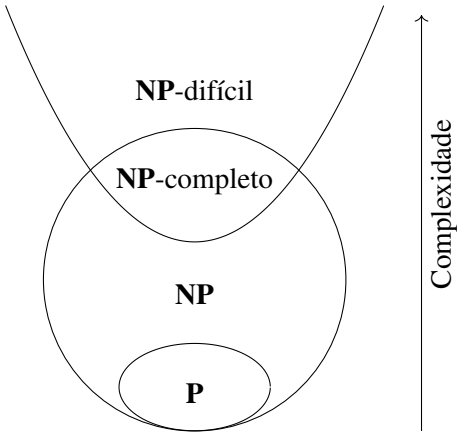


Fig. 2.10 Relação entre as classes de problemas de acordo com a sua complexidade

Um exemplo de um problema que se enquadra na classe **NP**-completo corresponde ao problema 3-SAT [5]. De seguida, efetuar-se-á uma breve descrição deste problema visto que será fundamental mais à frente.

Uma fórmula booleana é construída a partir de variáveis, que podem assumir os valores de 0 ou 1, de parênteses e dos operadores $\wedge, \vee, \neg$ correspondentes à conjunção, disjunção e negação, respetivamente. Por sua vez, o problema de satisfação booleana (SAT), na sua forma mais geral, consiste em decidir se uma dada fórmula booleana é satisfazível, isto é, se existe pelo menos uma atribuição de valores às variáveis de modo a que a fórmula tome o valor 1. Um caso especial do SAT corresponde ao problema 3-SAT. Uma fórmula booleana deste problema é composta por cláusulas, sendo estas formadas por 3 literais (um literal corresponde a uma variável ou à sua negação). Isto significa que considerando as variáveis $x_1, \ldots, x_n$ a fórmula do problema 3-SAT expressa-se da seguinte maneira:

$$\bigwedge_{i=1,\ldots,m} C_i, \quad \text{onde} \quad C_i = (l_{i_1} \vee l_{i_2} \vee l_{i_3})$$

Tal como mencionado anteriormente, os l_{i_j} representam os literais, sendo que a cada um pode corresponder uma variável ou a negação da mesma.

Relativamente à classe **NP**-difícil tem-se que o problema em estudo, ou seja, o PCMCGCC se encaixa nesta categoria. A prova deste resultado será efetuada adiante e contará com a participação da fórmula 3-SAT.

Capítulo 3

Problema do Caminho Mais Curto

O foco deste estudo incide numa generalização do problema do caminho mais curto, onde nem as posições dos vértices nem os comprimentos das arestas são fixos. Não obstante, antes de desenvolver esta variante efetuar-se-á um retrato do clássico PCMC.

3.1 Descrição do Problema Clássico do Caminho Mais Curto

Na sua forma mais geral, o problema do caminho mais curto procura, dado um grafo, o caminho de comprimento mínimo entre dois vértices desse mesmo grafo. O comprimento do caminho resulta assim, na soma do comprimento das arestas percorridas que, normalmente são escalares fixos previamente definidos.

Um aspeto interessante do PCMC traduz-se na solubilidade que possui. Isto significa que este problema pode ser resolvido em tempo polinomial, existindo variadíssimos algoritmos especializados para o fazer. Um dos mais conhecidos consiste no algoritmo de Dijkstra, contudo só pode ser aplicado caso os custos das arestas sejam não negativos.

A fim de ilustrar o que foi proferido, interprete-se o seguinte grafo.

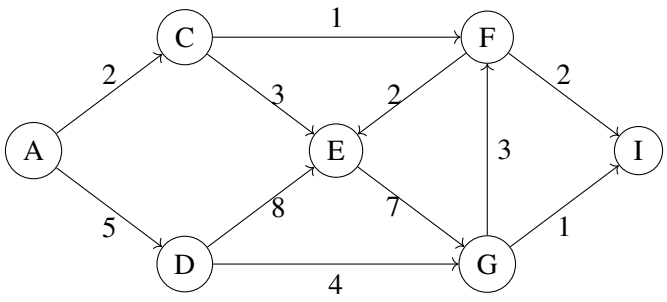


Fig. 3.1 Grafo composto por sete vértices e onze arestas

Exemplo 5. Supondo que se pretende encontrar o caminho mais curto entre A e I e atendendo às características do grafo orientado 3.1, facilmente se apura que a solução pode ser obtida visualmente. Assim, para além de A e I, o caminho mais curto é constituído pelos vértices C e F e apresenta o custo de 5 unidades.

Contudo, nem todos os problemas apresentam este grau de facilidade, impossibilitando a sua resolução a olho nú. Como exemplo serve o típico problema de, dado o mapa de Portugal encontrar o caminho mais curto entre Viseu e Faro, onde a utilização de algoritmos especializados é fundamental. Além disso, o PCMC também consegue ser solucionado usando Programação Linear, como se comprovará de seguida.

3.2 Formulação do Problema Clássico do Caminho Mais Curto

Com o propósito de cimentar o que foi proferido relativamente ao clássico PCMC, passar-se-á à formulação do mesmo. Dado um digrafo $G := (V,E)$ e os pontos s e t pertencentes a V, segundo [6], ir-se-á parametrizar o caminho de s para t com recurso a variáveis binárias $\{\varphi_e\}_{e\in E}$. Em virtude destas admitirem apenas dois resultados, assumir-se-á que φ_e toma o valor unitário quando a aresta e é atravessada pelo caminho e, incontestavelmente, toma o valor nulo caso tal não se verifique.

A fim de utilizar as variáveis binárias, é indispensável determinar as restrições a que estas devem obedecer. Para tal, com o intuito de tornar essa definição o mais natural possível, considere-se o PCMC como o problema de transferir uma unidade de fluxo de s para t com o menor custo. Denote-se por E_v o conjunto das arestas conectadas ao vértice $v \in V$. Este conjunto pode ser decomposto em dois, onde um é constituído pelas arestas que incidem em v e o outro é formado pelas arestas que saem de v. Sucintamente, representando tais conjuntos por E_v^{in} e E_v^{out}, respetivamente, temos que $E_v = E_v^{in} \cup E_v^{out}$. Além disso, assume-se sem perda de generalidade que $|E_s^{in}| = |E_t^{out}| = 0$, ou seja, que as arestas que incidem em s e que saem de t não são consideradas.

Adaptando a variável φ_e ao presente contexto, vê-se que esta representa as unidades de fluxo transportadas pela aresta e. Por forma a que seja enviada apenas uma unidade do vértice de partida para o vértice de destino, é necessário garantir a conservação de fluxo ao longo de todo o caminho. Esta condição pode ser interpretada matematicamente da seguinte maneira:

$$\sum_{e\in E_t^{in}} \varphi_e = \sum_{e\in E_s^{out}} \varphi_e, \tag{3.1}$$

$$\sum_{e\in E_v^{in}} \varphi_e = \sum_{e\in E_v^{out}} \varphi_e, \quad \forall v \in V \setminus \{s,t\}. \tag{3.2}$$

As igualdades presentes em 3.1 e 3.2 evidenciam, respetivamente, que o fluxo de saída do vértice s é igual ao fluxo de entrada do vértice t e que os fluxos de entrada e saída do vértice v têm de ser idênticos.

Por último, resta assegurar que uma unidade de fluxo é injetada no vértice de partida e ejetada no vértice de destino. Essa imposição pode ser introduzida na igualdade 3.1 do jeito seguinte:

$$\sum_{e\in E_t^{in}} \varphi_e = \sum_{e\in E_s^{out}} \varphi_e = 1, \tag{3.3}$$

Posto isto, assumindo que os custos das arestas c_e são não negativos e finitos tem-se que o PCMC pode ser formulado no PL apresentado a seguir.

$$\begin{aligned} \text{minimizar} \quad & \sum_{e\in E} c_e\varphi_e \\ \text{sujeito a} \quad & \sum_{e\in E_t^{in}} \varphi_e = \sum_{e\in E_s^{out}} \varphi_e = 1, \\ & \sum_{e\in E_v^{in}} \varphi_e = \sum_{e\in E_v^{out}} \varphi_e, \qquad \forall v \in V\setminus\{s,t\}, \\ & \varphi_e \geq 0, \qquad \forall e \in E. \end{aligned} \tag{3.4}$$

Contrariamente ao que foi definido, note-se que a formulação apresentada não exige explicitamente que as variáveis φ_e sejam binárias. Apenas se impõe que estas sejam não negativas. As razões por detrás desta alteração prendem-se ao facto de se poder mostrar que as soluções de 3.4 assumem valores binários e da restrição $\varphi_e \in \{0,1\}$ não afetar o valor ótimo deste programa.

Com o propósito de realçar que o PCMC pode ser resolvido por um programa linear, nomeadamente pelo programa 3.4, considere-se o seguinte grafo.

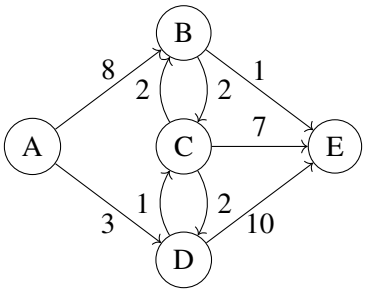


Fig. 3.2 Digrafo composto por cinco vértices e nove arestas

Exemplo 6. Dado o grafo 3.2, a ideia passa por implementar computacionalmente a formulação 3.4 e verificar que é encontrado, efetivamente, o caminho mais curto entre os vértices A e E. Deste modo, procedeu-se à construção de um código em MATLAB, que se encontra em Anexo, e obteve-se o seguinte resultado:

```
Status: Solved
Optimal value (cvx_optval): +7

phiF = phi'

phiF =

    0.0000    1.0000    0.0000    1.0000    1.0000    0.0000    0.0000    1.0000    0.0000

echo off
```

Fig. 3.3 Resultado do código construído em MATLAB

De acordo com o código elaborado, o "phiF" declarado na figura 3.3 retrata o vetor constituído pelo conjunto das arestas que formam o digrafo anterior. Assim, a cada elemento do vetor corresponde uma aresta do digrafo. Os componentes de "phiF" podem assumir o valor 1, caso as respetivas arestas sejam percorridas pelo caminho, ou 0, caso tal não se verifique.

Dito isto, o resultado da implementação da formulação 3.4 coincidiu com o que era de esperar, ou seja, escolher o caminho $(A,(A,D),D,(D,C),C,(C,B),B,(B,E),E)$, cujo comprimento é de 7 unidades.

Claramente que o digrafo apresentado enquadra uma realidade pouco complexa, contudo este serve apenas para referir que o PCMC também consegue ser resolvido através de um PL.

3.3 Descrição do Problema do Caminho Mais Curto em Grafos de Conjuntos Convexos

Prosseguindo, passe-se à generalização do problema do caminho mais curto. O PCMCGCC trata-se de um problema onde os vértices e os comprimentos das arestas não são fixos. De modo mais preciso, tem-se um grafo onde cada vértice está emparelhado com um conjunto convexo (denomina-se este tipo de grafo por grafo de conjuntos convexos). Assim, as posições de cada vértice passam a ser variáveis de decisão contínua que se encontram restritas pelo respetivo conjunto convexo e, por sua vez, os comprimentos das arestas passam a ser definidos por uma função convexa (por exemplo, a distância euclidiana) das posições dos vértices que estas conectam [6].

Com o propósito de ilustar o que foi dito, apresenta-se um exemplo na figura 3.4, retirado de [6].

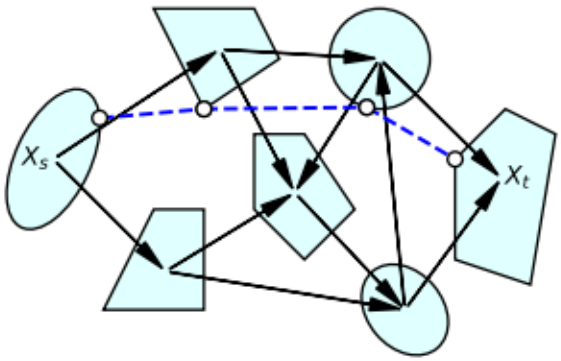


Fig. 3.4 Exemplo de um problema de caminho mais curto num grafo de conjuntos convexos

De acordo com a Figura 3.4, o objetivo consiste em encontrar o caminho mais curto entre os conjuntos X_s e X_t. A escolha dos conjuntos de maneira a constuir um caminho não é aleatória, uma vez que tais transições só são permitidas entre conjuntos conectados por uma aresta (setas pretas). Tal caminho encontra-se sinalizado pela linha tracejada a azul, que acaba por ligar todos os círculos brancos. Estes simbolizam a posição de cada vértice no respetivo conjunto ao longo do caminho. Poderiam ter sido escolhidos outros pontos dentro dos mesmos conjuntos ou selecionados conjuntos diferentes, contudo, se tal acontecesse não iria cumprir com o propósito de descobrir o caminho mais curto, visto que o custo de percorrer uma aresta é uma função convexa das posições dos vértices que esta liga.

Ao contrário do que acontece com o PCMC, o PCMCGCC é NP-difícil, ou seja, não constitui um problema de fácil resolução. Esta ilação será provada na próxima secção, depois de se efetuar a formulação deste problema.

3.4 Formulação do Problema do Caminho Mais Curto em Grafos de Conjuntos Convexos

Com base no retrato efetuado na secção anterior, esta ficará responsável pela formulação do PCMCGCC. Deste modo, considere-se um digrafo $G := (V,E)$. Para cada vértice $v \in V$ tem-se um conjunto convexo, compacto (limitado e fechado) e não vazio, X_v, que contém um ponto x_v. A ponto de simplificar a interpretação, assume-se que todos os conjuntos X_v pertencem ao mesmo espaço, isto é, $X_v \in \mathbb{R}^n$. Tal como referido, o tamanho da aresta $e = (u,v)$ é determinado por uma função convexa da localização dos pontos x_u e x_v. Represente-se tal função por $l_e(x_u,x_v)$, que toma valores em $\mathbb{R}_0^+ \cup \{\infty\}$. Assumir-se-á que esta é própria (atinge um valor finito em pelo menos um ponto), fechada e convexa.

Partindo-se de dois vértices do grafo, s e t, denote-se por π a sequência de vértices distintos que começa em s e termina em t. Por outras palavras, assumindo s e t como vértices de partida e destino, respetivamente, tem-se que $\pi = (v_k)_{k=0}^{K}$, tal que $v_0 = s$, $v_K = t$ e $(v_k, v_{k+1}) \in E$, $\forall k \in 0,\ldots,K-1$. Por fim, designe-se por $E_\pi = \{(v_k, v_{k+1})\}_{k=0}^{K-1}$ o conjunto das arestas atravessadas por π e como Π o conjunto de todos os caminhos de s para t. Desta forma, o problema do caminho mais curto em grafos de conjuntos convexos pode ser formalizado da seguinte maneira:

$$\begin{aligned} \text{minimizar} \quad & \sum_{e \in E_\pi} l_e(x_u,x_v) \\ \text{sujeito a} \quad & \pi \in \Pi, \\ & x_v \in X_v, \qquad \forall v \in \pi. \end{aligned} \tag{3.5}$$

Analisando a formulação obtida em 3.5, facilmente se constata que generaliza o PCMC com arestas de custos não negativos. Para se chegar a tal ilação, basta que a função convexa, correspondente ao comprimento da aresta entre dois pontos (x_u e x_v), tome um valor constante. Por outras palavras, sendo c_e um valor estabelecido de acordo com a aresta e, temos $l_e(x_u,x_v) = c_e$. Um outro cenário que comprova esta dedução sucede-se quando se fixam as posições dos vértices. Se tal acontecer, é evidente que se tem $l_e(x_u,x_v) = d_e$, onde d_e corresponde à distância entre os pontos x_u e x_v. Porventura, caso se fixe π a formulação anterior simplifica numa otimização convexa que pode ser eficientemente solucionada para a maioria dos conjuntos convexos X_v e da função l_e.

Efetuada a formulação do PCMCGCC, resta provar que a mesma é NP-difícil. Para tal, recorrer-se-á ao teorema que se apresenta de seguida, retirado de [6].

Teorema 1. *O PCMCGCC (3.5) é NP-difícil.*

Demonstração: A prova deste resultado reduz-se a mostrar que o Problema do Caminho Hamiltoniano (PCH) é uma redução do PCMCGCC que ocorre em tempo polinomial. Visto que o PCH é NP-completo, segundo [5], então prova-se o pretendido, ou seja, que o PCMCGCC é NP-difícil.

Um PCH resume-se a averiguar se um dado grafo é Hamiltoniano ou não. Por sua vez, um grafo apresenta esta designação caso contenha um caminho que visite todos os seus vértices, tendo este caminho a mesma nomeação. Por outras palavras, utilizando a notação mencionada anteriormente,

tem-se que dado um caminho $\pi = (v_k)_{k=0}^{K}$ este diz-se Hamiltoniano se $K = |V| - 1$ e um grafo que contenha tal caminho, naturalmente, também se designa Hamiltoniano.

Prosseguindo com a prova, a ideia passa por construir uma instância do problema 3.5 que partilha o mesmo grafo que um dado PCH. Deste modo, definem-se os conjuntos de partida e chegada como $X_s = \{0\}$ e $X_t = \{1\}$, enquanto que os restantes conjuntos são formados pelos pontos pertencentes ao intervalo entre os pontos dos conjuntos de partida e chegada. Ou seja, tem-se $X_v := [0,1]$, para todo o $v \in V \setminus \{s,t\}$. Perante estas atribuições, facilmente se constata que o posicionamento ótimo dos vértices ao longo de um caminho π fixo é dado por $x_{v_k} = k/K$ para $k = 0,\ldots,K$. Utilizando a distância Euclidiana para o cálculo do comprimento das arestas, tem-se que o tamanho do caminho é igual a $K(1/K)^2 = 1/K$. Perante este panorama, conclui-se que um caminho ótimo é aquele para o qual o K é maximizado. Além disso, esse caminho é Hamiltoniano se e só se o grafo também o for. Sintetizar esta instância, bem como verificar se $K = |V| - 1$, leva tempo polinomial. □

Esta demonstração evidencia que o PCMCGCC é difícil de resolver, mesmo que os conjuntos convexos X_v sejam simples intervalos unidimensionais.

O resultado subsequente declara que o PCMCGCC continua a ser de difícil resolução, mesmo que sejam admitidas algumas imposições aos dados deste problema.

Teorema 2. *O PCMCGCC (3.5) é NP-difícil, independentemente de o grafo G ser acicíclo, de os conjuntos X_v serem disjuntos e de os comprimentos da arestas l_e serem positivamente homogéneos.*

A prova deste teorema é bastante complicada e por isso, a sua realização requererá um enorme esforço.

Tal como mencionado anteriormente, o problema 3-SAT, relativamente à sua complexidade, pertence à classe **NP**-completo. Além disso, possui a característica de ser a redução de um qualquer problema **NP** em tempo polinomial. Desta forma, se entretanto o problema 3-SAT possuir um algoritmo que o resolva em tempo polinomial, então todos os problemas **NP** também podem ser solucionados em tempo polinomial. Caso um problema **NP** seja intratável, então o problema 3-SAT também o é [3].

A propriedade descrita é utilizada em [5] para provar que o problema do caminho mais curto euclidiano em três dimensões é **NP**-difícil. Para demonstrar **Teorema** 2 utilizar-se-á uma abordagem semelhante. Por outras palavras, ir-se-á reduzir o problema 3-SAT ao PCMCGCC em tempo polinomial. Assim, dados os pontos de partida e chegada, x_s e x_t respetivamente, e uma fórmula 3-SAT composta por n variáveis e m cláusulas, a demonstração consistirá em decidir em tempo polinomial se a fórmula é satisfazível mediante o conhecimento do caminho mais curto entre x_s e x_t.

Um método utilizado para decidir se uma fórmula é satisfazível começa por deduzir todas as atribuições possíveis para as variáveis que a compõe (as variáveis só podem assumir os valores 0 ou 1). Tomando uma fórmula 3-SAT com n variáveis facilmente se constata que serão construídas 2^n atribuições. O próximo passo consiste em verificar se as atribuições para as variáveis fazem com que a fórmula seja satisfazível. Uma vez que a fórmula 3-SAT corresponde a uma conjunção de m cláusulas, para que esta seja satisfazível é necessário as cláusulas que a constituem também o sejam. Assim, ter-se-ão de analisar todas as cláusulas a fim de averiguar se assumem o valor "verdadeiro". Visto que uma cláusula de uma fórmula 3-SAT consiste numa disjunção de três literais, basta verificar se um deles toma o valor 1.

Resumidamente, para cada atribuição gerada, se todas as cláusulas possuírem pelo menos um literal com valor "verdadeiro", então a respetiva atribuição torna a fórmula 3-SAT satisfazível. Caso contrário, a atribuição retorna a fórmula não satisfazível.

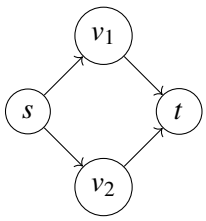
Aplicando o método descrito à demonstração do **Teorema** 2 e retomando a ideia do mesmo, tem-se que a cada atribuição da fórmula 3-SAT irá corresponder um caminho mais curto de x_s para x_t (cada ponto desse caminho encontra-se associado a um conjunto, que se assume ser unidimensional). Deste modo, gerar-se-ão todos os caminhos mais curtos de x_s para x_t e, posteriormente, analisar-se-ão tais caminhos com o intuito de aferir se as respetivas atribuições tornam a fórmula 3-SAT satisfazível.

Observação 1. *De acordo com o que foi exposto e como se verá mais à frente, cada caminho mais curto será codificado por uma sequência binária. Assumir-se-á que as codificações serão geradas de forma ordenada.*

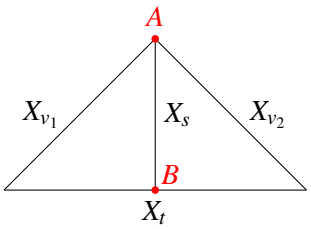
Exemplo 7. A título de exemplo, considere-se uma fórmula 3-SAT constituída por 3 variáveis. O conjunto de todas as atribuições possíveis é composto pelas seguintes sequências binárias: (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1). Cada atribuição representa um caminho mais curto.

De forma a realizar o procedimento traçado, apresentar-se-ão estruturas, definições e alguns resultados imprescendíveis para a elaboração da prova do **Teorema** 2.

Um splitter é um digrafo conforme descrito na Figura 3.5a, que se encontra associado ao conjunto 3.5b. Represente-se este grafo por S.



(a) Digrafo do splitter



(b) Esboço do splitter

Fig. 3.5 Splitter

Relacionado a este grafo orientado surge o seguinte resultado:

Proposição 1. *No splitter, para cada ponto $x_s \in X_s$ existem dois caminhos mais curtos com comprimento igual a $|X_s|$. Tais caminhos terminam em $x_{t_1}, x_{t_2} \in X_t$ com $d(x_s, A) = d(x_{t_1}, B) = d(x_{t_2}, B)$.*

Demonstração: De forma a provar esta proposição considere-se que $X_s = \{0\} \times [0,1]$ e olhe-se apenas para a metade direita do splitter.

Partindo-se do ponto $x_s = (0,x) \in X_s$, tem-se que o problema de encontrar o caminho mais curto resume-se a descobrir o posicionamento dos pontos x_{v_2} e x_t que asseguram um caminho de comprimento mínimo. Posto isto, tomando-se $x_{v_2} = (1-y, y)$ e $x_t = (1-y, 0)$ o problema 3.6 consiste em

$$\text{minimizar } y+\sqrt{(1-y)^2+(y-x)^2}.$$

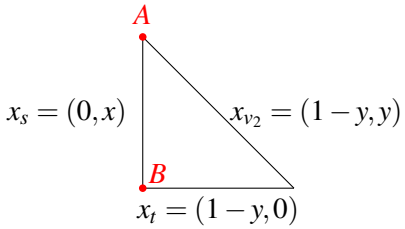


Fig. 3.6 Metade direita do splitter

Para calcular o minimo desta função, deriva-se em ordem a y e obtém-se

$$\frac{\partial}{\partial y}\left(y+\sqrt{(1-y)^2+(y-x)^2}\right)=1+\frac{2y-x-1}{\sqrt{x^2-2xy+2y^2-2y+1}}.$$

Igualando esta expressão a zero alcança-se $y=1$ e $y=x$. Utilizando a segunda solução, $x_s=(0,x), x_{v_2}=(1-x,x), x_t=(1-x,0)$ o que mostra que $d(x_s,A)=d(x_{t_2},B)$. Além disso, verifica-se que o valor ótimo do problema é 1 o que comprova que o comprimento do caminho é igual a $|X_s|$. Analogamente se prova que para a metade esquerda do splitter $d(x_s,A)=d(x_{t_1},B)$ e que o comprimento do caminho é igual a $|X_s|$. □

Para ilustrar o que foi proferido segue o próximo exemplo.

Exemplo 8. Considere-se X_s um intervalo com quatro pontos, equidistantes dois a dois. O splitter de X_s possuirá oito caminhos mais curtos, tal como representado na Figura 3.7.

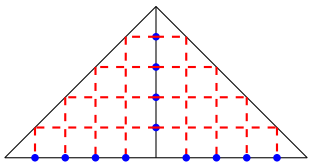


Fig. 3.7 Aplicação de um splitter

Associado ao grafo orientado descrito, surge o digrafo 3.8. Denote-se por S^n e interprete-se como sendo a aplicação de n splitters em cascata. Por outras palavras, tomando um intervalo unidimensional X_{s_1} começa-se por aplicar um splitter. O intervalo X_{t_1} consequente do splitter constitui o novo X_s (ou seja, o X_{s_2}), ao qual será aplicado um novo splitter. Esta diretriz repete-se até serem utilizados n splitters.

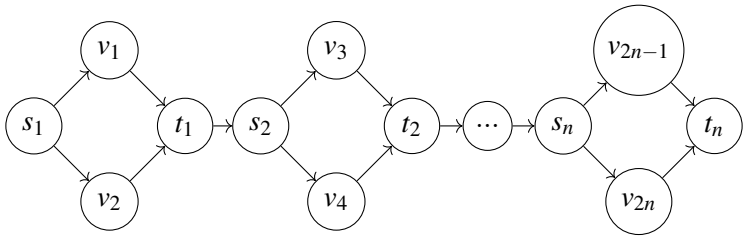


Fig. 3.8 Digrafo de S^n

Atendendo aos resultados expostos para os splitters, naturalmente se chega ao Corolário 1 e à Proposição 2.

Corolário 1. *Em S^n, para cada ponto $x_s \in X_{s_1}$ existem 2^n caminhos mais curtos de comprimento $(2^n-1)|X_{s_1}|$.*

Demonstração: Para a prova deste resultado fixe-se um ponto em X_{s_1}. A Proposição 1 enuncia que num splitter, para $x_s \in X_{s_1}$, existem dois caminhos mais curtos de comprimento $|X_{s_1}|$, que terminam em x_{t_1} e y_{t_1}. Além disso, da mesma, pode-se aferir que $X_{t_1} = 2|X_{s_1}|$. Tomando $X_{s_2} = X_{t_1}$ e aplicando-lhe um splitter, tem-se como resultado a existência de quatro caminhos mais curtos, dois para cada ponto de X_{s_2}, cujos comprimento traduzem-se na soma $|X_{s_1}| + 2|X_{s_1}|$. O intervalo consequente deste splitter, X_{t_2} terá tamanho, em módulo, igual a $2|X_{s_2}| = 2 \times 2|X_{s_1}|$. Considerando agora este intervalo ($X_{s_3} = X_{t_2}$), aplicando-lhe um novo splitter, obtêm-se oito caminhos mais curtos, cujos comprimentos correspondem ao dobro de X_{s_3}, isto é, $2 \times 2^2|X_{s_1}|$. Repetindo este método até serem utilizados n splitters, alcançam-se 2^n caminhos mais curtos. Tais caminhos apresentam tamanho igual a $|X_{s_1}| + 2|X_{s_1}| + 2^2|X_{s_1}| + \ldots + 2^n|X_{s_1}| = (2^n-1)|X_{s_1}|$. Provado para um ponto fixo em X_{s_1}, conclusões idênticas se obtêm para os restantes pontos em X_{s_1}. □

Proposição 2. *Considere-se que $x_s \in X_{s_1}$ é o ponto médio de X_{s_1}. Assumindo que $X_{t_n} = [p_1, p_2] \subseteq \mathbb{R}^2$, com $p_1 < p_2$, os caminhos mais curtos em S^n, identificados com $\{0,1\}^n$, terminam nos pontos*

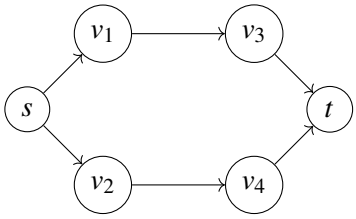
$$X_{t_n}^n = \left\{ p_1 + \left(\frac{1}{2^{n+1}} + \sum_{i=1}^{n} \frac{\alpha_i}{2^i} \right)(p_2 - p_1) : \alpha_i \in \{0,1\} \right\}.$$

Observação 2. *Num splitter, os caminhos que passam pelo lado esquerdo deste apresentam 0 na sua codificação. Caso utilizem o lado direito possuem 1 na sequência binária. Desta forma, os pontos ao longo do intervalo X_{t_n} encontram-se ordenados segundo a sua sequência binária. Por exemplo, considere-se S^2. O caminho que passa no lado esquerdo nos dois splitters apresenta $(0,0)$ como codificação. Caso utilize o lado esquerdo no primeiro splitter e o lado direito no segundo splitter, este possui a sequência binária $(0,1)$. O intervalo X_{t_2} apresenta em primeiro lugar o ponto cuja sequência binária é $(0,0)$ e depois aquela com atribuição $(0,1)$.*

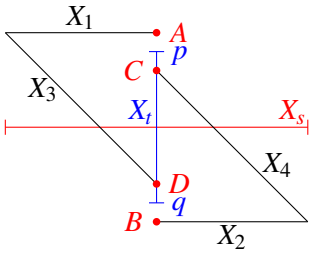
Demonstração: Como referido anteriormente, as sequências binárias que codificam os caminhos mais curtos irão ser geradas de forma ordenada. Para mostrar o pretendido analise-se o caso em que $X_s = [0,1]$ ($x_s = \frac{1}{2}$).

Aplicando um splitter a este intervalo, obtêm-se dois caminhos mais curtos que irão terminar em $\frac{1}{2}$ e $\frac{3}{2}$ no intervalo $X_t = [0,2]$. O caminho que termina em $\frac{1}{2}$ codifica-se por 0 e o que termina em $\frac{3}{2}$ por 1. Recorrendo a um novo splitter ao intervalo sequente do primeiro splitter, passarão a existir quatro caminhos mais curtos que terminam no intervalo $[0,4]$ nos pontos: $\frac{1}{2}$, $\frac{3}{2}$, $\frac{5}{2}$ e $\frac{7}{2}$. Tais caminhos apresentam, respetivamente, as sequências binárias $(0,0)$, $(0,1)$, $(1,0)$ e $(1,1)$. Continuando progressivamente este processo, verifica-se que a fórmula apresentada exprime corretamente o posicionamento dos pontos em S^n. □

Passando ao próximo grafo, defina-se shuffler como sendo o digrafo 3.9a, que se encontra associado ao conjunto 3.9b. Represente-se este grafo por Sh.



(a) Digrafo do shuffler



(b) Esboço do shuffler

Fig. 3.9 Shuffler

De forma a elucidar este grafo, proceder-se-á com a sua descrição detalhada. Deste modo, dado um segmento inicial X_s, o shuffler começa por parti-lo em duas partes. Uma das delas forma X_1, enquanto que a outra compõe X_2. De seguida o shuffler comporta-se como um splitter invertido. Com isto quer se dizer, que o caminho mais curto de X_1 para X_t passa por X_3, ao passo que o caminho de X_2 para X_t atravessa X_3.

Em consequência da descrição explicitada, emerge-se o próximo resultado.

Proposição 3. *Num shuffler, cada ponto $x_s \in X_{t_n}^n$ com sequência binária α, possui um único caminho mais curto de comprimento $\frac{|X_s|}{2}$. Este caminho termina em $y_t \in X_{t_n}^n$ com $d(x_{v_1},A) = d(y_t,D)$ $(x_{v_1} \in X_1)$, se x_s pertencer à metade esquerda de X_s. Caso contrário termina em $y_t \in X_t$ com $d(x_{v_2},B) = d(y_t,C)$ $(x_{v_2} \in X_2)$. Se β for a sequência binária de y_t então esta é descrita da seguinte maneira:*

$$\begin{cases} \beta_1 = \alpha_n, \\ \beta_i = \alpha_{i-1}\, i = 1,\ldots,n-1. \end{cases}$$

Além disso, tem-se que $|X_t| = \frac{|X_s|}{2}$.

Observação 3. *Uma vez que o shuffler, depois de ser dividido em duas partes, atua como um splitter invertido, para que as proposições referentes a este digrado sejam válidas impõe-se que $d(A,p) = d(p,C) = d(D,q) = d(q,B) = \frac{1}{2^{n+1}}|X_t|$.*

Demonstração da Proposição 3: Parte da prova desta demonstração pode ser subentendida visto que se trata da aplicação da Proposição 1, devidamente já provada. □

Com o intuito de ilustrar os caminhos mais curtos num shuffler segue-se o próximo exemplo.

Exemplo 9. Considerem-se os pontos $x_s \in X_{t_3}^3$ com $X_{t_3} = [0,8]$. Tais pontos apresentam, respetivamente, as sequências binárias: (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1). A aplicação do shuffler a este intervalo resume-se à figura 3.10.

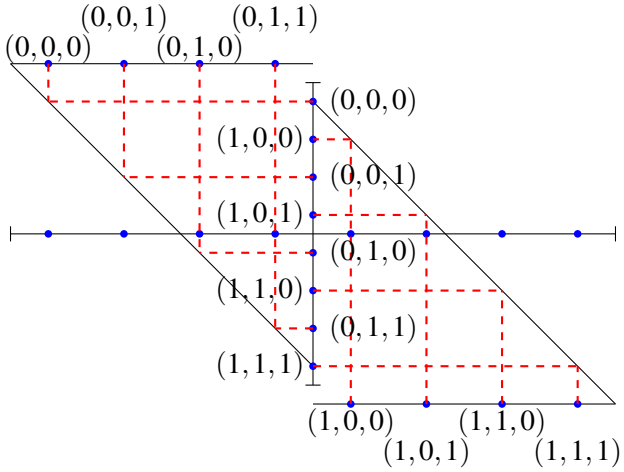
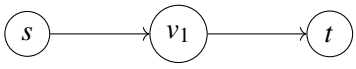
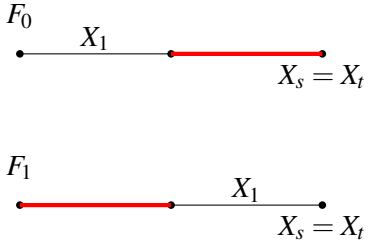


Fig. 3.10 Aplicação de um shuffler

O grafo orientado 3.11a retrata um filtro literal. A este grafo encontram-se associados dois conjuntos, tal como descrito na Figura 3.11b. Represente-se este digrafo por F_i com $i = 0,1$, consoante o conjunto que se pretende admitir.



(a) Digrafo do filtro literal



(b) Esboço do filtro literal

Fig. 3.11 Filtro literal

A especificidade deste digrafo permite tecer a seguinte proposição.

Proposição 4. *O comprimento do caminho mais curto em $F_i, i = 0,1$ é 0. Além disso, o ponto $x_s \in X_{t_n}^n$ com sequência binária α, é a origem de um caminho mais curto em F_i se e somente se $\alpha_1 = i$.*

Demonstração: Em conformidade com a Figura 3.11b, verifica-se que X_s e X_t são idênticos e encontram-se sobrespostos. Por isso, facilmente se consta que o caminho mais curto apresenta tamanho igual a zero.

Para provar o resto, divida-se $X_{t_n}^n$ em duas partes iguais. A lado esquerdo conterá os pontos cujas sequências binárias começam com zero, ou seja, $\alpha_i = 0$. Por sua vez, a parte direita possuirá os pontos

com $\alpha_i = 1$. Desta forma, em F_0 só existirão caminhos mais curtos para os pontos que se encontram no lado esquerdo, uma vez que o lado direito é intransponível. Para F_1, acontecerá o oposto, isto é, os pontos com $\alpha_1 = 1$ que se encontram na parte direita constituíram as origens dos caminhos mais curtos. □

Apesar deste digrafo ser de fácil compreensão, apresenta-se o Exemplo 10 para o clarificar.

Exemplo 10. Retomando os pontos do Exemplo 9, tem-se que para F_0 apenas os pontos com sequências binárias (0,0,0), (0,0,1), (0,1,0) e (0,1,1) são origens de caminhos mais curtos neste filtro. Caso se considere F_1, a conclusão é a oposta, isto é, são os pontos com atribuições (1,0,0), (1,0,1), (1,1,0) e (1,1,1) que compreendem as origens dos caminhos mais curtos.

O digrafo subsequente define um filtro de cláusula. Este filtro fará mais sentido no decorrer da demonstração do **Teorema** 2. Por isso, foque-se, para já, apenas na sua definição.

Dada uma cláusula $C = (l_{i_1} \vee l_{i_2} \vee l_{i_3})$, associa-se esta ao grafo que se encontra representado na Figura 3.12.

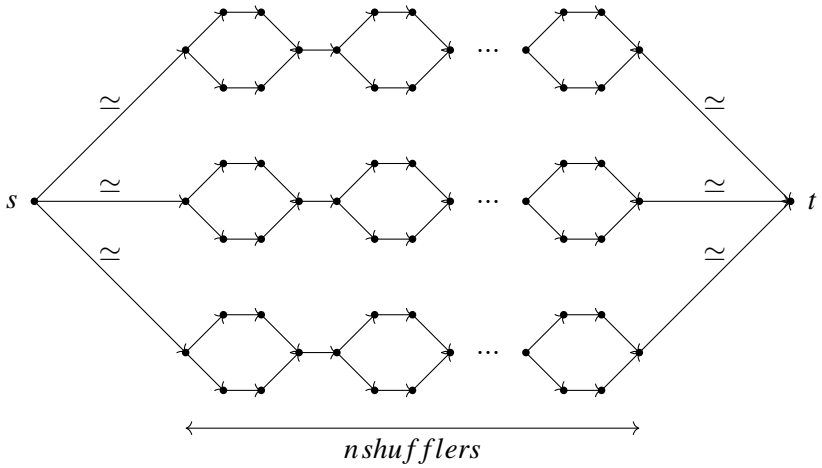


Fig. 3.12 Filtro FC^n

Este grafo descreve parte do filtro de clásula. Para que este esteja completo, ao caminho j é necessário acrescentar o filtro F_0 caso $l_{i_j} = \neg x_{i_j}$ ou F_1 caso $l_{i_j} = x_{i_j}$ antes do shuffler i_j. Esta descrição retrata um filtro de cláusula, que se representará por $FC^n(C)$ e cujo principal resultado encontra-se descrito na próxima proposição.

Proposição 5. *Se $x_s \in X^n_{t_n}$ é um ponto com sequência binária α, existe um caminho mais curto, partindo de x_s, em $FC^n(C)$ com comprimento igual ao do caminho mais curto em FC^n se e somente se α verifica C.*

Demonstração: Uma vez que o filtro $FC^n(C)$ corresponde à adição de filtros literais ao filtro FC^n, naturalmente se deduz que os caminhos mais curtos de $FC^n(c)$ e FC^n apresentam o mesmo comprimento. Esta dedução deve-se pelo facto do caminho mais curto de um filtro literal ter comprimento 0.

Para provar a equivalência, comece-se pelo lado direto da implicação. Note-se que a sequência binária do ponto x_s além de codificar o caminho mais curto, retrata também uma atribuição para as n variáveis da fórmula 3-SAT.

Assim, assumindo-se que existe um caminho mais curto em $FC^n(C)$, partindo de x_s, isto significa que α faz com que existe pelo menos um literal cujo valor é 1. Desta forma, a cláusula C é satisfeita, pois só precisa de um literal com valor 1 para que esta também assuma o valor 1.

Passando ao outro lado da implicação, tem-se que se α verifica C, então pelo menos um dos três literais que a compõe tem valor 1. Isto implica que há um caminho mais curto, partindo de x_s, uma vez que os caminhos não são bloqueados nos filtros em $FC^n(C)$. □

Finalmente, estão reúnidas as condições para se efetuar a prova do **Teorema** 2.

*Demonstração do **Teorema** 2:* Tal como indicado anteriormente, pretende-se reduzir o problema 3-SAT ao PCMCGCC em tempo polinomial. Ou seja, dados os pontos de partida e chegada, x_s e x_t, e uma fórmula 3-SAT com n variáveis e m cláusulas, determinar-se-á se a fórmula é satisfazível através do conhecimento de um caminho mais curto entre x_s e x_t.

Posto isto, o primeiro passo equivale à geração de todos os caminhos mais curtos entre x_s e x_t. Tais caminhos são gerados pela execução de n splitters em cascata (S^n), com início no ponto x_s. Como verificado anteriormente no fim de S^n existirão 2^n caminhos mais curtos de comprimento $(2^n-1)|X_s|$, identificados por $\{0,1\}^n$, que terminam nos pontos $x_{t_n} \in X^n_{t_n}$.

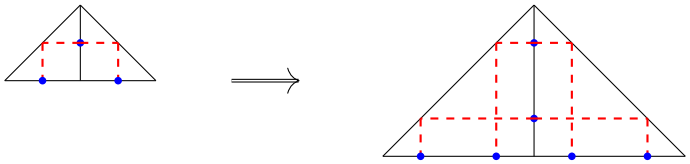


Fig. 3.13 Aplicação de dois splitters em S^n com começo no ponto x_s.

O próximo passo consiste em verificar se as sequências binárias que codificam cada um dos 2^n caminhos mais curtos fazem com que a fórmula 3-SAT seja satisfazível. Uma vez que a fórmula 3-SAT corresponde a uma conjunção de m cláusulas, para que esta seja satisfazível é necessário que as cláusulas que a constituem também o sejam. Assim, ter-se-ão de analisar todas as cláusulas a fim de averiguar se assumem o valor "verdadeiro". Para tal, recorrer-se-á a filtros de cláusulas ($FC^n(C)$), visto que estes garantem que existe um caminho mais curto (em $FC^n(C)$) se e somente se a sequência binária que codifica o ponto de partida verificar uma dada cláusula (Proposição 5). Com o intuito de examinar todas as cláusulas que compõe a fórmula 3-SAT, a ideia passa por colocar m filtros de cláusula em cascata. Após este procedimento, o filtro de cláusula m conterá apenas os caminhos mais curtos cujas atribuições satisfazem a fórmula 3-SAT, isto se existirem.

□

Capítulo 4

Formulações dos Problemas

Tal como referido anteriormente, o PCMCGCC é NP-difícil. De forma a contornar esta situação formular-se-á este problema como um programa convexo inteiro misto (PCIM) que possa ser resolvido eficientemente pelo método branch and bound (B&B baseia-se, de forma abreviada, no desenvolvimento de uma listagem inteligente das soluções candidatas à solução ótima do problema). O desenvolvimento deste PCIM será dividida em três fases. Em primeiro lugar, extender-se-á a formulação linear do clássico PCMC para a realidade do PCMCGCC. A etapa seguinte resume-se na convexificação das restrições do problema obtido no passo anterior. O resultado corresponde à primeira formulação do PCIM. O último ponto restringe-se em reduzir a formulação devido à existência de variáveis de decisão e restrições redundantes. Como consequência, obtém-se a formulação definitiva do PCIM.

4.1 Formulação Bilinear

Tal como referido, começar-se-á pela adaptação da formulação do programa linear do PCMC, obtida em 3.4, ao contexto que se pretende. Com isto quer se dizer que, em primeiro lugar, é fundamental fazerem-se as alterações necessárias à formulação 3.4 para que esta passe a codificar o PCMCGCC. À primeira vista, uma possível abordagem poderia recair na inclusão das posições dos vértices x_v entre as variáveis de decisão, na introdução das restrições convexas $x_v \in X_v, \forall v \in V$ e na substituição da função objetivo pela função não convexa $\sum_{e=(u,v)\in E} l_e(x_u,x_v)\varphi_e$. Deste modo, o problema 3.4 transformaria-se no seguinte:

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{e=(u,v)\in E} l_e(x_u,x_v)\varphi_e & & \\
\text{sujeito a} \quad & \sum_{e\in E_t^{in}} \varphi_e = \sum_{e\in E_s^{out}} \varphi_e = 1, & & \\
& \sum_{e\in E_v^{in}} \varphi_e = \sum_{e\in E_v^{out}} \varphi_e, & \forall v \in V\setminus\{s,t\}, & \\
& \varphi_e \geq 0, & \forall e \in E. & \\
& x_v \in X_v, & \forall v \in V, &
\end{aligned}
\tag{4.1}
$$

Embora se tenha chegado a uma formulação, esta não é valida visto que apresenta duas falhas. Uma delas incide na dificuldade da sua resolução, devido à não convexidade da função objetivo. A outra falha reside na função correspondente à determinação do comprimento das arestas, l_e. Tendo em conta que na sua definição assumiu-se que esta podia tomar valor infinito, quando $l_e(x_u,x_v)=\infty$ e $\varphi_e=0$ o produto $l_e(x_u,x_v)\varphi_e$ é indefinido. Contudo, quando $\varphi_e=0$ seria expectável que o custo do acréscimo da aresta e fosse zero. Com o objetivo de contornar este obstáculo utilizam-se as funções de perspetiva. Definindo duas variáveis auxiliares por aresta, $y_e=\varphi_e x_u$ e $z_e=\varphi_e x_v, \forall e=(u,v)\in E$, e substituindo a função objetivo de 4.1 por $\sum_{e=(u,v)\in E}\tilde{l}_e(y_e,z_e,\varphi_e)$, chega-se ao pretendido. É claro que, quando φ_e é positivo, a expressão apresentada resulta da substituição de x_u e x_v na função objetivo de 4.1 pelas variáveis auxiliares definidas, isto é, $l_e(x_u,x_v)\varphi_e=l_e(y_e/\varphi_e,z_e/\varphi_e)\varphi_e:=\tilde{l}_e(y_e,z_e,\varphi_e)$. Relativamente ao cenário que motivou a utilização da função perspetiva, constata-se que já não constitui um problema. Isto porque, quando $\varphi_e=0$ sabe-se, naturalmente, que $y_e=z_e=0$ e usando a noção da função perspetiva definida anteriormente tem-se

$$\tilde{l}_e(0,0,0):=\lim_{\tau\to 0}\tau l_e(\bar{x}_u+0/\tau,\bar{x}_v+0/\tau)=\lim_{\tau\to 0}\tau l_e(\bar{x}_u,\bar{x}_v)=0,$$

onde $\bar{x}_u$ e $\bar{x}_v$ são dois quaisquer pontos tais que $l_e(\bar{x}_u,\bar{x}_v)$ é finito. Posto isto, conclui-se que quando $\varphi_e=0$, o custo do acréscimo da aresta e é zero (ou seja, encontra-se bem definido), mesmo que $l_e(x_u,x_v)=\infty$. Assim sendo, o PCMCGCC pode-se formular da seguinte maneira:

$$\begin{aligned}
\text{minimizar}\quad & \sum_{e\in E}\tilde{l}_e(y_e,z_e,\varphi_e) && \\
\text{sujeito a}\quad & \sum_{e\in E_t^{in}}\varphi_e=\sum_{e\in E_s^{out}}\varphi_e=1, && \\
& \sum_{e\in E_v^{in}}\varphi_e=\sum_{e\in E_v^{out}}\varphi_e, && \forall v\in V\setminus\{s,t\},\\
& \varphi_e\geq 0, && \forall e\in E\\
. \quad & x_v\in X_v, && \forall v\in V,\\
& y_e=\varphi_e x_u, z_e=\varphi_e x_v, && \forall e=(u,v)\in E,
\end{aligned} \tag{4.2}$$

As variáveis de decisão desta formulação são os fluxos φ_e, as posições dos vértices x_v e as variáveis auxiliares y_e e z_e. As variáveis auxiliares possuem um papel muito importante, uma vez que têm a função de unir os vértices x_u e x_v quando a aresta $e=(u,v)$ é atravessada por uma unidade de fluxo e efetuar o contrário quando $\varphi_e=0$. Este desempenho deve-se às restrições de igualdade bilinear (as penúltimas restrições de 4.2). Relativamente à formulação, estas são as únicas que não são convexas e por isso, proceder-se-á à sua convexificação.

No entanto, antes de avançar com esse procedimento, mostrar-se-á que, tal como mencionado na formulação do clássico PCMC, exigir que as variáveis φ_e sejam binárias não tem qualquer influência no valor ótimo do problema 4.2.

Proposição 6. *Seja $l^\star$ um mínimo local de 4.2. Existe uma atribuição viável para as variáveis deste problema com custo igual a $l^\star$ tal que $\varphi_e\in\{0,1\},\forall e\in E$.*

Demonstração: Sejam $\{x_v^\star\}_{\{v\in V\}}$ e $\{\varphi_e^\star\}_{\{e\in E\}}$ um minimizante local do problema 4.2 com custo $l^\star$. Adicionam-se a 4.2 as restrições $x_v = x_v^\star, \forall v \in V$ e $\varphi_e = 0, \forall e \in E$ desde que $\varphi_e^\star = 0$. Depois de algumas manipulações, este programa é simplificado num PL da forma 3.4 com o conjunto de arestas $E' = \{e \in E : \varphi_e^\star > 0\}$ e o custo das mesmas $c_e := l_e(x_u^\star, x_v^\star), \forall e \in E'$. Estes custos são todos finitos, visto que caso $l_e(x_u^\star, x_v^\star) = \infty$ e $\varphi_e^\star > 0$, isto implicaria que $l^\star = \infty$. O valor ótimo deste problema deve ser igual a $l^\star$, caso contrário, a solução de 4.2 não seria um mínimo local. Além disso, a observação feita após a formulação linear do PCMC permite assumir que os fluxos ótimos deste problema são ótimos. Combinados com as variáveis $x_v^\star$, estes fluxos fornecem uma solução viável de 4.2 com custo $l^\star$. □

4.2 Reformulação Convexa Inteira Mista do Programa Bilinear

Retomando a formulação 4.2, neste momento a ideia passa por convexificar as restrições de igualdade bilinear. Para tal, segundo [6], a conduta a utilizar incide na construção de uma relaxação convexa para as restrições bilineares, o que permitirá reformular 4.2 como um PCIM. Mas antes disso, é preciso descrever as restrições do problema 4.2 em termos de um conjunto não convexo bilinear.

Posto isto, comece-se por estabelecer $\varphi_v = (\varphi_e)_{e\in E_v}$, como sendo o fluxo presente no vértive v e denote-se por $\Phi_v \subset \mathbb{R}^{|E_v|}$, o poliedro definido pelas restrições lineares impostas a φ_v. Isto é, recuperando as condições instituídas às variáveis φ_e na formulação do PCMC (transferência de uma unidade de fluxo do vértice de partida para o vértice de destino, conservação de fluxo ao longo de todo o caminho e fluxo não negativo) prontamente se verifica que:

$$\Phi_s = \{\varphi_s \in \mathbb{R}^{|E_s^{out}|} : \varphi_s \geq 0 \wedge \sum_{e\in E_s^{out}} \varphi_e = 1\},$$

$$\Phi_t = \{\varphi_t \in \mathbb{R}^{|E_t^{in}|} : \varphi_t \geq 0 \wedge \sum_{e\in E_t^{in}} \varphi_e = 1\},$$

$$\Phi_v = \{\varphi_v \in \mathbb{R}^{|E_v|} : \varphi_v \geq 0 \wedge \sum_{e\in E_v^{in}} \varphi_e = \sum_{e\in E_v^{out}} \varphi_e\}, \qquad \forall v \in V \setminus \{s,t\}.$$

Além disso, considere-se a matriz $M_v \in \mathbb{R}^{d\times|E_v|}$, cujas colunas são as variáveis auxiliares z_e para $e \in E_v^{in}$ e y_e para $e \in E_v^{out}$. Deste modo, as restrições bilineares de 4.2 podem ser reescritas como $M_v = x_v\varphi_v^T$. Definindo o conjunto não convexo

$$\Omega_v := \{(x, \varphi, M) : x \in X_v, \varphi \in \Phi_v, M = x\varphi^T\},$$

pode-se exprimir a formulação 4.2 de forma compacta da seguinte maneira:

$$\begin{aligned} &\text{minimizar} && \sum_{e=(i,j)\in E} \tilde{l}_e([M_i]_e, [M_j]_e, \varphi_e) \\ &\text{sujeito a} && (x_v, \varphi_v, M_v) \in \Omega_v, \quad \forall v \in V, \end{aligned} \tag{4.3}$$

onde $[M_i]_e$ corresponde à coluna e de M_i.

Efetuada a descrição das restrições de 4.2 em termos de Ω_v, a próxima etapa corresponde à relaxação convexa deste conjunto. Segundo [6], a técnica de relaxação usualmente utilizada para este tipo de conjuntos bilineares consiste na construção de um invólucro convexo em torno desse conjunto.

De forma a tomar o caso geral e a aliviar a notação, descarte-se a dependência do vértice v e estenda-se a discussão, momentaneamente, a dois quaisquer conjuntos convexos e compactos $X \subseteq \mathbb{R}^d$ e $\Phi \subseteq \mathbb{R}^n$. Por outras palavras, atente-se ao conjunto

$$\Omega := \{(x,\varphi,M) : x \in X, \varphi \in \Phi, M = x\varphi^T\} \subseteq \mathbb{R}^d \times \mathbb{R}^n \times \mathbb{R}^{d\times n}.$$

A ambição era que o invólucro convexo de Ω fosse igual à interseção de todas as restrições válidas de Ω. Contudo, o processo de obter todas as restrições válidas revela-se bastante difícil. A alternativa passa por construir o invólucro convexo em torno de Ω, multiplicando todas as desigualdades válidas $a^T x + b \geq 0$ do conjunto X por todas as desigualdades válidas $c^T\varphi + d \geq 0$ do conjunto Φ.

Observação 4. *Como as desigualdades $a^T x + b \geq 0$ e $c^T\varphi + d \geq 0$ são válidas em X e Φ, respetivamente, tem-se que as desigualdades $(a^T x + b)(c^T\varphi + d) \geq 0$ são válidas em $X \times \Phi$.*

O passo seguinte consiste na linearização dos produtos obtidos através da igualdade bilinear $M = x\varphi^T$. Este procedimento fornece uma família infinita de desigualdades válidas para Ω e produz o resultado que se encontra transcrito no próximo lema.

Lema 2. *O conjunto*

$$\Omega \subseteq \Omega' := \{(x,\varphi,M) : a^T Mc + da^T x + bc^T\varphi + bd \geq 0, \quad \forall (a,b) \in X^\bullet, \forall (c,d) \in \Phi^\bullet\},$$

corresponde à relaxação convexa de Ω.

Demonstração: Concretize-se a metodologia alusiva à construção do invólucro convexo em torno de Ω. Sejam $(a,b) \in X^\bullet$ e $(c,d) \in \Phi^\bullet$, para qualquer ponto $(x,\varphi,M) \in \Omega$ tem-se:

$$\begin{aligned} 0 &\leq (a^T x + b)(c^T\varphi + d) \\ &= (a^T x)(c^T\varphi) + da^T x + bc^T\varphi + bd \\ &= (a^T x)(\varphi^T c) + da^T x + bc^T\varphi + bd \\ &= a^T Mc + da^T x + bc^T\varphi + bd, \end{aligned}$$

provando assim o pretendido. □

Observação 5. *Relativamente à inclusão $\Omega \subseteq \Omega'$, o ideal seria que a relaxação coincidisse com o invólucro convexo de Ω. Contudo, geralmente, isso não pode ser esperado, tal como retratado em [6].*

O facto de Ω' envolver um número infinito de restrições impossibilita a sua implementação computacional. Ou seja, dá-se a necessidade de descrever este conjunto por um número finito de restrições. Perante este cenário, surge o seguinte lema.

Lema 3. *A seguinte descrição é equivalente a* $\Omega^{'}$*:*

$$\left\{(x,\varphi,M): \begin{bmatrix} Mc+dx \\ c^T\varphi+d \end{bmatrix} \in \tilde{X}, \forall (c,d) \in \Phi^{\bullet}\right\}.$$

Demonstração: Utilizando o resultado referido no Lema 1, reescreve-se $\begin{bmatrix} Mc+dx \\ c^T\varphi+d \end{bmatrix} \in \tilde{X}$ como $a^T(Mc+dx)+b(c^T\varphi+d) \geq 0, \forall (a,b) \in X^{\bullet}$. Desenvolvendo os produtos, obtém-se a definição anterior de $\Omega^{'}$, ou seja, $a^TMc+da^Tx+bc^T\varphi+bd \geq 0, \forall (a,b) \in X^{\bullet}, \forall (c,d) \in \Phi^{\bullet}$. □

Segundo [6], a abordagem traçada neste lema difere das técnicas clássicas, visto que a operação de perspetiva permite tratar o conjunto convexo X como uma "black box" e resumir todas as desigualdades válidas deste conjunto por meio de uma restrição de cone convexo. Posto isto, recuperando a dependência do vértice v, a relaxação convexa de 4.2 resulta, simplesmente, na substituição de Ω_v por $\Omega_v^{'}$, conforme descrito no Lema anterior. De forma a tornar mais alusiva esta relaxação, apresenta-se o seguinte Lema obtido exclusivamente pela definição de $\Omega^{'}$ no Lema 3 e pela recuperação da dependência do vértice v.

Lema 4. *Relativamente à formulação 4.2, considere-se uma qualquer desigualdade linear válida escrita da forma*

$$\sum_{e\in E_v} c_e\varphi_e + d \geq 0.$$

Visto que $E_v = E_v^{in} \cup E_v^{out}$*, tem-se que a seguinte restrição também é válida para 4.2:*

$$\begin{bmatrix} \sum_{e\in E_v^{in}} c_e z_e + \sum_{e\in E_v^{out}} c_e y_e + dx_v \\ \sum_{e\in E_v} c_e\varphi_e + d \end{bmatrix} \in \tilde{X}_v$$

.

Demonstração: A desigualdade linear apresentada pode ser escrita como uma condição do Lema 3. Para tal basta interpretar o sumatório como um produto matricial e considerar $\varphi_v = (\varphi_e)_{e\in E_v}$, da seguinte forma:

$$\sum_{e\in E_v} c_e\varphi_e + d \geq 0 \Leftrightarrow c^T\varphi_v + d \geq 0, \forall \varphi_v \in \Phi_v \Leftrightarrow (c,d) \in \Phi_v^{\bullet}.$$

Resta mostrar que a restrição mencionada é equivalente àquela mencionda no Lema 3. Posto isto, tem-se

$$\begin{bmatrix} \sum_{e\in E_v^{in}} c_e z_e + \sum_{e\in E_v^{out}} c_e y_e + dx_v \\ \sum_{e\in E_v} c_e\varphi_e + d \end{bmatrix} \in \tilde{X}_v \Leftrightarrow \begin{bmatrix} \sum_{e\in E_v^{in}} c_e\varphi_e x_v + \sum_{e\in E_v^{out}} c_e\varphi_e x_u + dx_v \\ \sum_{e\in E_v} c_e\varphi_e + d \end{bmatrix} \in \tilde{X}_v \Leftrightarrow$$

$$\begin{bmatrix} \sum_{e\in E_v} c_e\varphi_e x_v + dx_v \\ \sum_{e\in E_v} c_e\varphi_e + d \end{bmatrix} \in \tilde{X}_v \Leftrightarrow \begin{bmatrix} (\sum_{e\in E_v} c_e\varphi_e + d)x_v \\ \sum_{e\in E_v} c_e\varphi_e + d \end{bmatrix} \in \tilde{X}_v \Leftrightarrow \begin{bmatrix} (c^T\varphi_v + d)x_v \\ c^T\varphi_v + d \end{bmatrix} \in \tilde{X}_v \Leftrightarrow$$

$$\begin{bmatrix} (\varphi_v^T c + d)x_v \\ c^T \varphi_v + d \end{bmatrix} \in \tilde{X}_v \Leftrightarrow \begin{bmatrix} x_v \varphi_v^T c + dx_v \\ c^T \varphi_v + d \end{bmatrix} \in \tilde{X}_v \Leftrightarrow \begin{bmatrix} M_v c + dx_v \\ c^T \varphi_v + d \end{bmatrix} \in \tilde{X}_v.$$

Em jeito de conclusão, comprovou-se que a restrição mencionada neste Lema é equivalente à descrição de Ω' no Lema 3. □

Observação 6. *Caso $\sum_{e \in E_v} c_e \varphi_e + d = 0$, então o resultado do Lema 4 simplifica na seguinte igualdade linear:*

$$\begin{bmatrix} \sum_{e \in E_v^{in}} c_e z_e + \sum_{e \in E_v^{out}} c_e y_e + dx_v \\ \sum_{e \in E_v} c_e \varphi_e + d \end{bmatrix} = 0$$

Em jeito de conclusão, partindo da dedução do Lema 4, constata-se que este também produz um invólucro convexo em torno das restrições bilineares, mediante um número reduzido de condições convexas. Ou seja, este Lema transforma qualquer restrição linear válida dos fluxos presentes no vértice v numa restrição de perspetiva que envolve as igualdades bilineares. Assim, o PCIM resulta da aplicação do Lema 4/Observação 2 a cada uma das três primeiras restrições de 4.2 (também expressas em Φ_v).

Observação 7. *Note-se que o resultado do Lema 4, também pode ser atingido multiplicando $x_v \in X_v$ no lado esquerdo da desigualdade $\sum_{e \in E_v} c_e \varphi_e + d \geq 0$ e utilizando as restrições bilineares. Por isso, a restrição $x_v \in X_v$ não irá constar explicitamente na formulação do PCIM, no entanto será verificada.*

Por forma a que o Lema seja aplicado corretamente é primordial definir c_e e d para cada uma das restrições. Para tal, defina-se $C_v = (c_e)_{e \in E_v}$, o vetor de dimensão $\mathbb{R}^{|E_v|}$. Assim, a c_e corresponderá um valor do vetor C_v, que será estabelecido de acordo com a restrição em causa.

Começando pela restrição $\sum_{e \in E_t^{in}} \varphi_e = \sum_{e \in E_s^{out}} \varphi_e = 1$, apura-se que esta pode ser desdobrada em duas condições:

$$\sum_{e \in E_t^{in}} \varphi_e = 1 \quad \wedge \quad \sum_{e \in E_s^{out}} \varphi_e = 1.$$

Em ambas as situações, $d = -1$. Relativamente a c_e, verifica-se que na primeira condição $C_t = (1_{|E_t^{in}|})$ enquanto que na segunda $C_s = (1_{|E_s^{out}|})$, sendo que 1_n denota um vetor com n entradas iguais a 1. Recorrendo à Observação 2 obtém-se, respetivamente, as seguintes restrições:

$$\begin{bmatrix} \sum_{e \in E_t^{in}} z_e - x_t \\ \sum_{e \in E_t^{in}} \varphi_e - 1 \end{bmatrix} = 0 \quad \wedge \quad \begin{bmatrix} \sum_{e \in E_s^{out}} y_e - x_s \\ \sum_{e \in E_s^{out}} \varphi_e - 1 \end{bmatrix} = 0. \tag{4.4}$$

No que diz respeito à restrição $\sum_{e \in E_v^{in}} \varphi_e = \sum_{e \in E_v^{out}} \varphi_e$, tem-se que $C_v = (1_{|E_v^{in}|}, -1_{|E_v^{in}|})$ e $d = 0$. Posto isto, servindo-se novamente da Observação 2 tomam-se as seguintes condições:

$$\begin{bmatrix} \sum_{e \in E_v^{in}} z_e - \sum_{e \in E_v^{out}} y_e \\ \sum_{e \in E_v^{in}} \varphi_e - \sum_{e \in E_v^{out}} \varphi_e \end{bmatrix} = 0. \tag{4.5}$$

Por fim, resta a condição $\varphi_e \geq 0$. Nesta, a C_v corresponde um qualquer vetor de base canónica de $\mathbb{R}^{|E_v|}$ e d assume o valor de 0. Assim, o resultado da aplicação do Lema 4 equivale a:

$$(y_e, \varphi_e) \in \tilde{X}_u, \forall e \in E_v^{in} \quad \wedge \quad (z_e, \varphi_e) \in \tilde{X}_v \forall e \in E_v^{out}. \tag{4.6}$$

Substituindo as restrições de 4.2 pelas condições obtidas, define-se o PCIM como:

$$\begin{aligned} \text{minimizar} \quad & \sum_{e \in E} \tilde{l}_e(y_e, z_e, \varphi_e) \\ \text{sujeito a} \quad & 4.4 \\ & 4.5 \qquad \forall v \in V, \\ & 4.6 \qquad \forall e \in E, \\ & \varphi_e \in \{0,1\}, \qquad \forall e \in E. \end{aligned} \tag{4.7}$$

Esta formulação apresenta as mesmas variáveis de decisão que o problema 4.2 e possui um número elevado de restrições. Posto isto, seria interessante analisar as restrições com o intuito de compactar o problema.

4.3 Formulação Convexa Inteira Mista Reduzida

A fim de condensar a formulação 4.7, comece-se pela análise das restrições 4.4. Espontaneamente verifica-se que estas traduzem as igualdades:

$$x_s = \sum_{e \in E_s^{out}} y_e \quad \wedge \quad x_t = \sum_{e \in E_t^{in}} z_e, \tag{4.8}$$

$$\sum_{e \in E_t^{in}} \varphi_e = \sum_{e \in E_s^{out}} \varphi_e = 1. \tag{4.9}$$

Utilizando as restrições 4.9 e 4.6, consegue-se observar que os valores de x_s e x_t calculados segundo 4.8 pertencem sempre aos conjuntos X_s e X_t, respetivamente. Logo, face à formulação 4.7, as igualdades 4.8 são redundantes, pelo que podem ser retiradas desta formulação. Assim, o cálculo das posições dos vértices de partida e destino é resolvido fora do problema e as variáveis x_s e x_t são eliminadas da formulação.

No que toca aos vértices $v \in V \setminus \{s,t\}$ nota-se que as variáveis de decisão x_v referentes a estes vértices não constam na formulação 4.7 e, neste seguimento, podem ser também eliminadas deste problema. O cálculo do posicionamento dos vértices $v \in V \setminus \{s,t\}$ será definido no próximo teorema e, claro está, será efetuado fora do problema.

Sucintamente, à formulação obtida anteriormente ir-se-ão retirar as variáveis de decisão $x_v, \forall v \in V$, bem como as restrições 4.8. Assim, conclui-se que o PCIM formula-se da seguinte maneira:

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{e\in E} \tilde{l}_e(y_e, z_e, \varphi_e) \\
\text{sujeito a} \quad & \sum_{e\in E_t^{in}} \varphi_e = \sum_{e\in E_s^{out}} \varphi_e = 1, \\
& \sum_{e\in E_v^{in}} z_e = \sum_{e\in E_v^{out}} y_e, && \forall v \in V\setminus\{s,t\}, \\
& \sum_{e\in E_v^{in}} \varphi_e = \sum_{e\in E_v^{out}} \varphi_e, && \forall v \in V\setminus\{s,t\}, \\
& (y_e,\varphi_e) \in \tilde{X}_u, (z_e,\varphi_e) \in \tilde{X}_v, && \forall e=(u,v)\in E, \\
& \varphi_e \in \{0,1\}, && \forall e \in E.
\end{aligned}
\tag{4.10}
$$

Obtida a formulação do PCIM, provar-se-á agora que esta é equivalente ao PCMCGCC 3.5 e reconstruir-se-á o posicionamento dos vértices para $v \in V\setminus\{s,t\}$.

Teorema 3. *O valor ótimo do PCIM 4.10 é igual ao do PCMCGCC 3.5. A solução ótima π do PCMCGCC é recuperada a partir da solução do PCIM através da relação $E_\pi = \{e \in E : \varphi_e = 1\}$. Além disso, as posições ótimas dos vértices são obtidas por 4.8, para os vértices de partida e destino e por:*

$$x_v = \frac{\sum_{e\in E_v^{out}} y_e}{\sum_{e\in E_v^{out}} \varphi_e}, \quad \forall v \in V\setminus\{s,t\}.$$

Caso o divisor tome o valor 0, assume-se que x_v é um qualquer ponto em X_v.

Demonstração. Partindo-se de uma solução ótima do PCIM 4.10, as restrições de fluxo juntamente com a restrição $\varphi_e \in \{0,1\}$ deste problema, garantem que o conjunto de arestas $E_\pi = \{e \in E : \varphi_e = 1\}$ identifica um caminho π, solução de 3.5.

De forma a provar isto, considerem-se as restrições $(y_e,\varphi_e) \in \tilde{X}_u, (z_e,\varphi_e) \in \tilde{X}_v, \forall e=(u,v)\in E$. Estas podem ser reescritas como $y_e \in X_u\varphi_e, z_e \in X_v\varphi_e, \forall e=(u,v)\in E$. Tomando apenas as arestas que são percorridas por uma unidade de fluxo, ou seja, as arestas $e=(u,v)\in E_\pi$, tem-se que as restrições anteriores se resumem a $y_e \in X_u, z_e \in X_v$. Para as restantes arestas $e \in E\setminus E_\pi$, como têm fluxo nulo ($\varphi_e = 0$), as mesmas restrições simplificam a $y_e = z_e = 0$.

Para todas as arestas $e=(u,v)$ e $f=(v,w)$ ao longo do caminho mais curto, constata-se que as restrições referentes à conservação de fluxo em torno do vértice v podem ser expressas como $z_e = y_f$. Atendendo às definições mencionadas para a determinação das posições dos vértices, verifica-se que $x_u = y_e$ e $x_v = z_e, \forall e \in E_\pi$, o que mostra que as restrições do PCIM são equivalentes às do PCMCGCC.

Relativamente à função objetivo, verifica-se que as arestas atravessadas por uma unidade de fluxo possuem um custo não negativo $\tilde{l}_e(y_e,z_e,1) = l_e(\frac{y_e}{1},\frac{z_e}{1})1 = l_e(y_e,z_e) = l_e(x_u,x_v)$. As restantes arestas, evidentemente, apresentam custo $\tilde{l}_e(0,0,) = 0$. Posto isto, a presença de ciclos pode ser excluída e confirma-se que o valor ótimo do PCIM é igual ao do PCMCGCC. □

A relaxação convexa do PCIM 4.10 resulta simplesmente da eliminação da restrição $\varphi_e \in \{0,1\}, \forall e \in E$. Deste modo, os fluxos podem assumir valores fracionários, originando um valor ótimo inferior ao do PCIM 4.10. A fim de ilustrar o que foi dito, considerem-se os próximos

problemas, resolvidos computacionalmente e cujo código, elaborado no MATLAB, se encontra em Anexo.

Observação 8. *Para os exemplos que irão ser apresentados, os tamanhos das arestas foram calculados de acordo com a distância Euclidiana. Além disso, assumiram-se que os conjuntos X_v eram polítopos representados por $\{x : A_v x \leq b_v\}$.*

Observação 9. *Caso os conjuntos X_v se reduzam a um ponto, o PCMCGCC resume-se ao PCMC. Ou seja, a formulação 4.10 simplifica para a formulação 3.4.*

Exemplo 11. Considere-se um grafo orientado G com $V = \{s,1,2,t\}$ e $E = \{(s,1),(s,2),(2,1),(2,t)\}$. Definam-se os conjuntos $X_s = [0,2]$, $X_1 = [4,5]$, $X_2 = [7,10]$ e $X_t = [12,14]$. Para este grafo, o valor ótimo do PCIM 4.10 coincide com o valor ótimo da sua relaxação, pelo que apenas se apresenta um dos outputs.

```
Status: Solved
Optimal value (cvx_optval): +10


phiF = phi'

phiF =

     0     1     0     1
```

Fig. 4.1 Resultado da implementação efetuada para o PCIM e para a sua relaxação

Conforme o código elaborado, o "phiF" declarado na figura acima retrata o vetor constituído pelo conjunto das arestas que formam o digrafo anterior. A cada elemento do vetor corresponde uma aresta do digrafo. Os componentes de "phiF" podem assumir o valor 1, caso as respetivas arestas sejam percorridas pelo caminho, ou 0, caso tal não se verifique. Deste modo, o output 4.1 indica que o caminho mais curto é $(s,(s,2),2,(2,t),t)$ com comprimento 10. Aplicando as fórmulas mencionadas anteriormente para o cálculo do posicionamento dos vértices, obtém-se o caminho mais curto sinalizado a vermelho na figura 4.2.

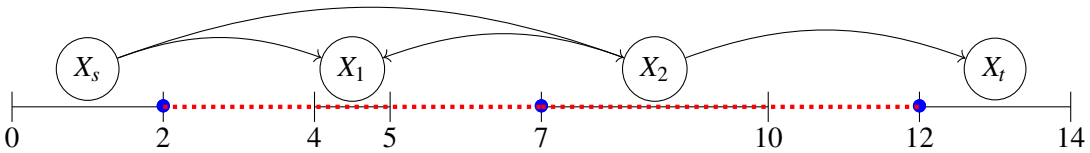


Fig. 4.2 Os pontos que se encontram sinalizados a azul representam as posições ótimas. À reta tracejada a vermelho corresponde o caminho mais curto entre s e t.

Exemplo 12. Atente-se agora ao digrafo G com $V = \{s,1,2,3,t\}$, $E = \{(s,1),(s,2),(1,3),(2,3),(3,t)\}$ e os conjuntos $X_s = \{(1,4)\}$, $X_1 = \{(2,y) : y \in [7,8]\}$, $X_2 = \{(2,y) : y \in [0,1]\}$, $X_3 = \{(4,y) : y \in [2,6]\}$, $X_s = \{(6,4)\}$. Aplicando este problema ao código referente à formulação do PCIM 4.10 obteve-se o seguinte output.

```
Status: Solved
Optimal value (cvx_optval): +8.16228


phiF = phi'

phiF =

     1     0     1     0     1
```

Fig. 4.3 Resultado da implementação efetuada para o PCIM

Deste resultado tira-se que o caminho mais curto corresponde a $(s,(s,1),1,(1,3),3,(3,t),t)$. Determinando as posições dos vértices ao longo do caminho obtém-se o caminho descrito na figura 4.4.

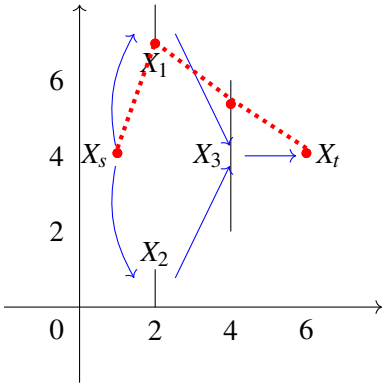


Fig. 4.4 O caminho mais curto encontra-se sinalizado pelo tracejado a vermelho. As setas azuis representam as arestas do digrado descrito anteriormente.

Passando à execução deste problema com o código alusivo à relaxação do PCIM 4.10, alcança-se o seguinte resultado:

```
Status: Solved
Optimal value (cvx_optval): +7.39835


phiF = phi'

phiF =

    0.5000    0.5000    0.5000    0.5000    1.0000
```

Fig. 4.5 Output da implementação efetuada para a relaxação do PCIM

Como se pode verificar, o valor ótimo da relaxação (7.39835) é inferior ao valor ótimo do PCIM (8.16228). Isto deve-se ao facto de os fluxos assumirem valores fracionários, tal como já referido anteriormente. De acordo com este output, partindo-se de s, o fluxo é dividido em duas partes, indo uma delas para o vértice 1 e a outra para o vértice 2. De seguida os fluxos são reúnidos no vértice 3 e posteriormente é enviado num todo para o vértice de destino t. Contudo, este procedimento não se enquadra na definição de caminho mais curto.

Para terminar, resta referir que a formulação 4.10 foi moldada para digrafos acíclicos. Para grafos orientados cíclicos é necesário introduzir a restrição:

$$\sum_{e \in E_v^{out}} \varphi_e \leq 0, \forall v \in V \setminus \{s,t\}.$$

Esta restrição, denominada restrição de grau, define um limite para o fluxo que sai do vértice v, evitando assim que a solução ótima contenha ciclos. Por forma a mostrar a veracidade deste apontamento considere-se o Exemplo 13.

Exemplo 13. Seja G o digrafo com $V = \{s,1,2,t\}$ e $E = \{(s,1),(1,2),(1,t),(2,1)\}$. Definam-se os conjuntos como $X_s = \{-1\}$, $X_1 = [-1,1]$, $X_2 = \{0\}$, $X_t = \{1\}$ e calculem-se os comprimentos das arestas segundo a distância Euclidiana ao quadrado. Alterando a função objetivo no código construído para a resolução da formulação 4.10 tem-se o seguinte resultado:

```
Status: Solved
Optimal value (cvx_optval): +1


phiF = phi'

phiF =

     1     1     1     1
```

Fig. 4.6 Output da implementação do PCIM quando o grafo é cíclico

Tendo em conta este output, verifica-se que todas as arestas são percorridas, ou seja, todas elas fazem parte do caminho mais curto deste problema. No entanto, isto induz a presença de um ciclo, nomeadamente o ciclo $(1,(1,2),2,(2,1),1)$. Além disso, retira-se que o valor ótimo deste é 1. Contudo, o valor ótimo deste problema é 2, tal como observado no output 4.7.

```
Status: Solved
Optimal value (cvx_optval): +2


phiF = phi'

phiF =

     1     0     1     0
```

Fig. 4.7 Output da implementação do PCIM com a restrição de grau, quando o grafo é cíclico

O resultado 4.7 deveu-se à introdução da restrição de grau na formulação 4.10. Neste pode-se observar que o caminho mais curto é $(s,(s,1),1,(1,t),t)$.

Assim, conclui-se que o acréscimo da restrição de grau à formulação 4.10 assegura a ausência de ciclos em grafos cíclicos.

Capítulo 5

Resultados Numéricos

Nesta secção efetuar-se-ão testes numéricos com o intuito de avaliar a formulação do PCIM 4.10, desenvolvida anteriormente, bem como a sua relaxação. Para tal, considerar-se-ão duas estruturas para a geração de problemas.

Em primeiro lugar, atente-se a problemas cuja descrição se assemelha à que se encontra na próxima figura.

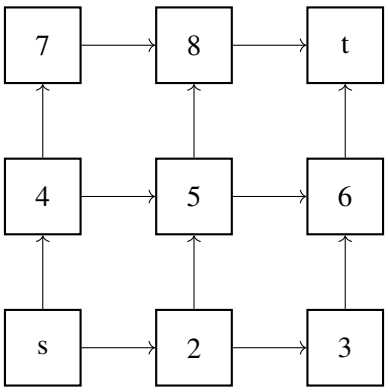


Fig. 5.1 Grafo orientado composto por nove vértices e doze arestas.

Atendendo a que a intenção passa por encontrar o caminho mais curto de s para t, acordo com a Figura 5.1, verifica-se que a maioria dos vértices que constituem o grafo oferecem duas possibilidades de prosseguir o caminho em direção a t (para a direita ou para cima). Os vértices que estabelecem a exceção situam-se no limite superior (o caminho apenas pode prosseguir para o vértice à direita) e na margem direita da malha (nestes o caminho só pode continuar para o vértice acima).

Considerando esta especificidade de problemas, testaram-se diferentes valores para a largura e altura da malha e para o espaçamento vertical e horizontal entre os conjuntos (assumiu-se que os conjuntos seriam quadrados e, por isso, variou-se a sua dimensão), com a intenção de avaliar a discrepância entre os valores ótimos da formulação 4.10 e da sua relaxação. Por outras palavras, variou-se o número de vértices na horizontal e na vertical, a largura dos conjuntos, sendo eles quadrados, e por fim a distância horizontal e vertical entre os conjuntos. Além disso, considerou-se a distância Euclidiana para o cálculo do comprimento das arestas.

Observação 10. *Para contextualizar, o grafo descrito na Figura 5.1 apresenta três vértices na horizontal e três vértices na vertical. Além disso, o espaçamento vertical e horizontal é idêntico à largura dos conjuntos.*

Para isso, construiu-se um código em Matlab que pode ser consultado no Anexo. Os resultados desses testes encontram-se descritos na tabela seguinte.

Malha		Conjuntos	PCIM		Relaxação	
Alt.	Larg.	Larg., Esp. Vert. e Hor.	V. Ótimo	Tempo(CPU)	V. Ótimo	Tempo(CPU)
10	10	1 , 1 , 1	24.6274	0.47	24.6274	0.22
10	10	1 , 1 , 2	32.2023	0.33	32.2023	0.23
10	10	1 , 2 , 1	32.2023	0.47	32.2023	0.19
10	10	2 , 1 , 1	35.9411	0.53	35.9411	0.39
20	10	1 , 1 , 1	41.9629	3.86	41.9309	0.45
10	20	1 , 2 , 1	49.3596	4.09	49.3276	0.28
20	20	1 , 1 , 1	52.9117	3.53	52.9117	0.36
20	20	2 , 2 , 1	93.0000	50.28	93.0000	0.55
20	25	1 , 1 , 1	61.3182	2056.86	60.4928	0.44

Tabela 5.1 Resultados dos testes em grafos "malha".

Da tabela anterior pode-se verificar que para grande parte dos testes, o valor ótimo do PCIM 4.10 é idêntico ao da sua relaxação. Quando não o é, a diferença em todos os casos não chega a uma unidade. No entanto, o que salta à vista é a diferenca de tempos de CPU. Inicialmente ambos apresentam tempos reduzidos, contudo à medida que os tamanhos dos grafos aumentam, o tempo de CPU para a formulação 4.10 aumenta drasticamente quando comparado com o da sua relaxação.

Para terminar, testar-se-á uma estrutura completamente diferente da anterior, com o intuito de tornar o grafo o mais aleatório possível. Para isso desenvolveu-se um código em Matlab, que se encontra em Anexo. De seguida, proceder-se-á com a descrição detalhada do mesmo.

Dados os números de vértices e de arestas de um grafo, começou-se por determinar o conjunto das arestas. Para isso, construíram-se caminhos de forma aleatória de s para t, de modo a que todos os vértices $v \in V \setminus \{s,t\}$ fossem percorridos por exatamente um caminho. Por outras palavras, gerou-se o número de vértices que um caminho iria ter e de seguida os vértices que este caminho iria dispor. Repetiu-se este processo até que todos os vértices $v \in V \setminus \{s,t\}$ fossem atravessados exatamente por um caminho. Terminada esta etapa, o próximo passo traduziu-se na adição aleatória de arestas ao grafo até que o número de arestas estivesse satisfeito. Isto é, geraram-se vértices de forma aleatória e analisou-se se a aresta que os ligaria já existia ou se consistia um caso proíbitivo. Os casos que se definiram proíbitivos correspondem às arestas que terminam em s, que começam em t ou que unem um vértice a ele próprio.

Para finalizar, resta definir os conjuntos. Definiu-se $X_s = \{x \in \mathbb{R}^2 : x_1 = x_2 = 0\}$ e $X_t = \{x \in \mathbb{R}^2 : x_1 = x_2 = 1\}$. O procedimento para a definição dos restantes conjuntos, X_v com $v \in V \setminus \{s,t\}$, foi também feito de modo aleatório, tendo-se gerado números uniformemente distribuídos no intervalo

$[0,1]^2$. Assumindo que os conjuntos apresentam a forma de um quadrado, estes números representam os centros dos vários conjuntos. Dado um valor para a área dos conjuntos, fixaram-se os limites dos conjuntos atendendo ao seu centro.

Tendo como objetivo clarificar o que foi dito, segue-se um grafo obtido deste processo. Este teve como input 5 vértices, 7 arestas e área dos conjuntos igual a 0,01.

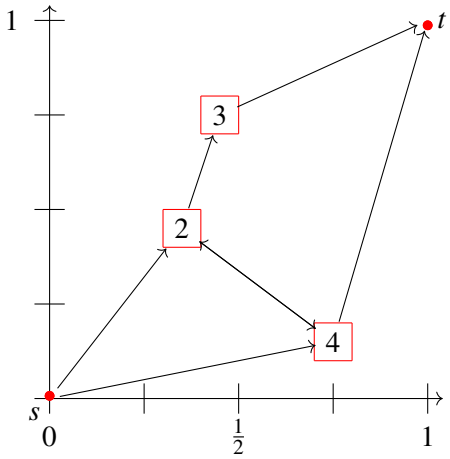


Fig. 5.2 Grafo composto por 5 vértices, 7 arestas e cuja área dos conjuntos é 0,01.

Com a finalidade de avaliar a performance do PCIM 4.10 e da sua relaxação neste tipo de grafos, geraram-se 100 grafos partindo dos mesmos parâmetros e calculou-se a média das diferenças entre os valores ótimos do PCIM e os valores ótimos da sua relaxação. Além disso, registou-se a maior diferença entre os valores ótimos do PCIM e da sua relaxação entre os 100 grafos gerados (isto, se existir). Estes resultados encontram-se descritos na próxima tabela. Note-se que se utilizou novamente a distância Euclidiana.

Grafo	Conjuntos	PCIM - Relaxação	
Nº de vértices e de arestas	Área	Média das diferenças	Diferença máx.
15 , 20	0.01	0.0000	0.0000
15 , 30	0.05	0.0000	0.0016
50 , 100	0.01	0.0000	0.0000
50 , 100	0.05	0.0002	0.0216
50 , 200	0.01	0.0001	0.0120
100 , 200	0.01	0.0000	0.0000
100 , 200	0.05	0.0003	0.0302
250 , 500	0.01	0.0000	0.0000

Tabela 5.2 Resultados dos testes em grafos gerados de modo aleatório - valor ótimo.

Na tabela abaixo encontram-se registados os tempos médios e máximos para o PCIM e para a sua relaxação, associados aos testes efetuados anteriormente.

Grafo	Conjuntos	PCIM - Tempo(CPU)	Relaxação - Tempo(CPU)
Nº de vértices e de arestas	Área	Médio | Máximo	Médio | Máximo
15 , 20	0.01	0.2316 | 2.3906	0.2520 | 1.7813
15 , 30	0.05	0.1416 | 1.3906	0.1448 | 0.5938
50 , 100	0.01	3.5938 | 2.9219	0.2606 | 1.4219
50 , 100	0.05	0.5128 | 1.6094	0.3572 | 1.4844
50 , 200	0.01	0.7872 | 3.9688	0.4948 | 1.7188
100 , 200	0.01	0.8584 | 2.6094	0.6053| 2.1094
100 , 200	0.05	0.7061 | 2.1250	0.4789 | 3.3906
250 , 500	0.01	2.4739 | 10.0625	1.7072 | 6.2656

Tabela 5.3 Resultados dos testes em grafos gerados de modo aleatório - tempo(CPU).

A análise das Tabelas 5.2 e 5.3 permite tirar conclusões semelhantes às que foram citadas para o primeiro tipo de problemas. Isto é, com o uso da distância Euclidiana, a relaxação do PCIM 4.10 é muito precisa para a maioria dos grafos gerados. Além disso, verifica-se que esta apresenta tempos de execução mais baixos que os da formulação do PCIM.

Os resultados obtidos neste tipo de grafos levantou alguma curiosidade, principalmente pela eficácia da relaxação em problemas de maiores dimensões. Isto incentivou à realização de alguns testes num ambiente de três dimensões. Assim, considerou-se $X_s = \{x \in \mathbb{R}^3 : x_1 = x_2 = x_3 = 0\}$ e $X_t = \{x \in \mathbb{R}^3 : x_1 = x_2 = x_3 = 1\}$. A geração dos X_v com $v \in V \setminus \{s,t\}$ realizou-se de forma análoga, adaptando à realidade do ambiente tridimensional.

O código responsável por este tipo de testes resultou da adaptação do código desenvolvimento para o ambiente bidimensional e pode ser consultado no Anexo. Os outputs deste apresentam-se nas Tabelas 5.4 e 5.5. Note-se que se voltou a utilizar a distância Euclidiana.

Grafo	Conjuntos	PCIM - Relaxação	
Nº de vértices e de arestas	Área	Média das diferenças	Diferença máx.
15 , 30	0.01	0.0000	0.0000
50 , 100	0.05	0.0002	0.0150
50 , 200	0.01	0.0000	0.0036
100 , 200	0.05	0.0000	0.0038
250 , 500	0.01	0.0000	0.0010

Tabela 5.4 Resultados dos testes em grafos gerados de modo aleatório num ambiente tridimensional - valor ótimo.

Grafo	Conjuntos	PCIM - Tempo(CPU)	Relaxação - Tempo(CPU)
Nº de vértices e de arestas	Área	Médio | Máximo	Médio | Máximo
15 , 30	0.01	0.2808 | 1.7500	0.3059 | 1.7656
50 , 100	0.05	0.5102 | 0.7969	0.3000 | 0.6406
50 , 200	0.01	0.6927 | 1.3906	0.3320 | 0.8750
100 , 200	0.05	0.8177 | 1.2813	0.4709 | 0.7813
250 , 500	0.01	2.6206 | 7.0313	1.6578 | 4.4375

Tabela 5.5 Resultados dos testes em grafos gerados de modo aleatório num ambiente tridimensional - tempo(CPU).

Mais uma vez, os dados retirados dos testes mostram que os valores ótimos do PCIM e da sua relaxação são maioritariamente iguais, ou seja, a relaxação é precisa. Além disso, à medida que o tamanho do problema aumenta, a tendência do tempo de execução é manter-se relativamente baixa.

Em jeito de conclusão, esta secção serviu para evidenciar que o PCIM 4.10 e a sua relaxação não se limitam a problemas de pequena dimensão e que permitem a resolução de problemas **NP**-difíceis.

Capítulo 6

Conclusão

O foco deste estudo incidiu numa das generalizações de um dos problemas mais estudados em otimização combinatória, o problema do caminho mais curto em grafos de conjuntos convexos (PCMCGCC).

Neste problema, os vértices e os comprimentos das arestas não são fixos. Ou seja, tem-se um grafo onde cada vértice está emparelhado com um conjunto convexo. Assim, as posições de cada vértice passam a ser variáveis de decisão contínua que se encontram restritas pelo respetivo conjunto convexo e, por sua vez, os comprimentos das arestas passam a ser definidos por uma função convexa (por exemplo, a distância euclidiana) das posições dos vértices que estas conectam.

A especificadade deste problema torna-o **NP**-difícil. De forma a contrariar esta situação desenvolveu-se, ao longo deste estudo, uma formulação convexa inteira mista e esta revelou-se bastante positiva, tendo em conta os resultados obtidos na secção 5. Ou seja, os testes numéricos mostraram que a relaxação da formulação desenvolvida é geralmente muito precisa e possibilita a resolução de problemas de grandes dimensões em tempos reduzidos.

Bibliografia

[1] Boyd, S. P. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[2] Deits, R. and Tedrake, R. (2014). Footstep planning on uneven terrain with mixed-integer convex optimization.

[3] Garey, M. R. (1979). A guide to the theory of np-completeness. *Computers and intractability*.

[4] Gouveia, J. (2017). Programação linear e otimização combinatória. Departamento de Matemática da Universidade de Coimbra.

[5] Karp, R. M. (1972). *Reducibility among combinatorial problems, in Complexity of computer computations*. Springer.

[6] Marcucci, T., Umenberger, J., Parrilo, P. A., and Tedrake, R. (2021). Shortest paths in graphs of convex sets. *arXiv preprint arXiv:2101.11565*.

[7] Rockafellar, R. T. (1970). Convex analysis, volume 28 of.

[8] Santos, J. L. (2019). Conceitos básicos sobre otimização. Departamento de Matemática, Faculdade de Ciências e Tecnlogia da Universidade de Coimbra.

[9] Santos, J. L. (2021). Conceitos básicos sobre otimização e sua contextualização em finanças. Métodos de Otimização em Finanças.

Anexo A

Nesta secção encontram-se todos os códigos elaborados em MATLAB. O primeiro foi construído especialmente para o Exemplo 6.

```
1   echo on
2
3   nv = 5; ne = 9;
4   E =
        [1,2,1;1,4,2;2,3,3;2,5,4;3,2,5;3,4,6;3,5,7;4,3,8;4,5,9];

5   c = [8;3;2;1;2;2;7;1;10];
6   somaS = zeros(ne,1); somaT = zeros(ne,1);
7   soma2in = zeros(ne,1); soma2out = zeros(ne,1);
8   soma3in = zeros(ne,1); soma3out = zeros(ne,1);
9   soma4in = zeros(ne,1); soma4out = zeros(ne,1);
10
11  for i = 1:1:ne
12          if(E(i,1) == 1)
13                  somaS((E(i,3)),1) = 1;
14          end
15          if(E(i,2) == 5)
16                  somaT((E(i,3)),1) = 1;
17          end
18          if(E(i,2) == 2)
19                  soma2in((E(i,3)),1) = 1;
20          end
21          if(E(i,1) == 2)
22                  soma2out((E(i,3)),1) = 1;
23          end
24          if(E(i,2) == 3)
25                  soma3in((E(i,3)),1) = 1;
26          end
27          if(E(i,1) == 3)
28                  soma3out((E(i,3)),1) = 1;
```

```
                end
                if(E(i,2) == 4)
                        soma4in((E(i,3)),1) = 1;
                end
                if(E(i,1) == 4)
                        soma4out((E(i,3)),1) = 1;
                end
        end

        cvx_begin
        variable phi(ne)
        minimize(c' * phi)
        subject to
        somaS' * phi == somaT' * phi == 1;
        soma2in' * phi == soma2out' * phi;
        soma3in' * phi == soma3out' * phi;
        soma4in' * phi == soma4out' * phi;
        phi >= 0;
        cvx_end
        phiF = phi'
        echo off
```

De forma a resolver os PCIM, de acordo com a formulação 4.10, procedeu-se com a construção do código apresentado de seguida. O input deste código consiste no número de vértices que compõe o grafo, da matriz de adjacência do mesmo e por fim dos limites inferiores e superiores dos conjuntos (visto que se assumiram ser quadrados ou retângulos). Como output tem-se o valor ótimo do problema, bem como um vetor que identifica as arestas que foram utilizadas pelo caminho ótimo.

```
nv = input('Introduza o numero de vertices do grafo: ');
M = zeros(nv,nv);
M = input('Introduza a matriz de adjacencia: ');

%Contagem do numero de arestas
ne = 0;
for i=1:nv
        for j=1:nv
                if(M(i,j) == 1)
                ne = ne+1;
                end
        end
end

```

```
15          %Definicao da matriz E, onde a primeira coluna
               corresponde ao vertice de
16          %partida, a segunda ao vertice de destino e a terceira
               identifica a etiqueta referente
17          %a aresta
18          E = zeros(ne,3);
19          k = 1;
20          for i=1:nv
21                  for j=1:nv
22                          if(M(i,j) == 1)
23                                  E(k,1) = i;
24                                  E(k,2) = j;
25                                  E(k,3) = k;
26                                  k = k+1;
27                          end
28                  end
29          end
30          %Definicao da matriz soma que contem a informacao
               relativa as arestas que entram
31          %e saem de cada vertice
32          soma = zeros(ne,(nv-2)*2 + 2);
33          k = 2;
34          for i = 2:nv-1
35                  for j = 1:ne
36                          if(E(j,2) == i)
37                                  soma((E(j,3)),k) = 1;
38                          end
39                          if(E(j,1) == i)
40                                  soma((E(j,3)),k+1) = 1;
41                          end
42                  end
43          k = k + 2;
44          end
45
46          for j = 1:ne
47                  if(E(j,1) == 1)
48                          soma((E(j,3)),1) = 1;
49                  end
50                  if(E(j,2) == nv)
51                          soma((E(j,3)),((nv-2)*2 + 2)) = 1;
52                  end
53          end
```

```
54
55          Av = [-1, 0; 1, 0; 0,-1,; 0, 1];
56          b = zeros(4,nv);
57
58          for j = 1:nv
59                  fprintf('Introduza os intervalos dos conjuntos
                        para x e y do conjunto %d: \n', j);
60                  b(1,j) = (input('X-limite inferior: \n'))*-1;
61                  b(2,j) = input('X-limite superior: \n');
62                  b(3,j) = (input('Y-limite inferior: \n'))*-1;
63                  b(4,j) = input('Y-limite superior: \n');
64          end
65          echo on
66          cvx_begin
67                  obj=0;
68                  variable phi(ne) binary
69                  variable z(ne,2)
70                  variable y(ne,2)
71                  for i=1:ne
72                          %Distancia Euclidiana
73                          obj=obj+norm(z(i,:)-y(i,:));
74
75                          %Distancia Euclidiana ao quadrado
76                          %obj=obj+pow_pos((norm(z(i,:)-y(i,:)))
                                ,2);
77                  end
78                  minimize obj
79                  subject to
80                          p = 1;
81                          for i = 1:ne
82                                  Av * (y(i,:))' <= b(:,E(p,1)) *
                                        phi(i);
83                                  Av * (z(i,:))' <= b(:,E(p,2)) *
                                        phi(i);
84                                  p = p + 1;
85                          end
86                          soma(:,(nv-2)*2 + 2)' * phi == soma(:,1)
                                ' * phi == 1;
87                          l = 2;cout=0;
88                          while(cout~=nv-2)
89                                  %Restricao para grafos ciclicos
90                                  %soma(:,l+1)' * phi <=1;
```

```
91                                  soma(:,l)' * phi == soma(:,l+1)'
                                        * phi;
92                                  soma(:,l)' * z == soma(:,l+1)' *
                                        y;
93                                  l = l + 2;
94                                  cout = cout + 1;
95                          end
96          cvx_end
97
98          phiF = phi'
99          echo off
```

Caso se pretenda utilizar a distância Euclidiana ao quadrado basta retirar a expressão correspondente ao seu cálculo do comentário e comentar a expressão anterior. Se o grafo for ciclico necessita-se de retirar a expressão que se encontra devidamente comentada.

O código correspondente à relaxação do PCIM é praticamente idêntico, sendo a única nuance o facto de a variável "phi" não apresentar a instrução para ser uma variável binária.

Na secção 'Resultados Numéricos' foram desenvolvidos dois códigos. O primeiro, correspondente ao grafo "malha" e à formulação 4.10, apresenta-se de seguida. Caso se pretenda passar para a relaxação, basta retirar a instrução *binary* em $phi(ne)$.

```
1           cvx_solver mosek
2           l = input('Introduza a largura da malha: ');
3           a = input('Introduza a altura da malha: ');
4
5           %Numero de vertices
6           nv = l * a;
7           %Numero de arestas
8           ne = a*(l-1) + l*(a-1);
9
10          %Matriz de adjacencia
11          M = zeros(nv,nv);
12
13          h = 0;
14          %Linhas horizontais
15          for i=1:a
16                  for j=1:l-1
17                          M(j+h,j+1+h) = 1;
18                  end
19                  h = h + l;
20          end
21          p = 0;
22          %Linhas verticais
```

```
23          for i=1:a-1
24                  for k=1:l
25                          M(k+p,k+l+p) = 1;
26                  end
27                  p = p + l;
28          end
29
30          E = zeros(ne,3);
31          k = 1;
32          for i=1:nv
33                  for j=1:nv
34                          if(M(i,j) == 1)
35                                  E(k,1) = i;
36                                  E(k,2) = j;
37                                  E(k,3) = k;
38                                  k = k+1;
39                          end
40                  end
41          end
42
43          soma = zeros(ne,(nv-2)*2 + 2);
44          k = 2;
45          for i = 2:nv-1
46                  for j = 1:ne
47                          if(E(j,2) == i)
48                                  soma((E(j,3)),k) = 1;
49                          end
50                          if(E(j,1) == i)
51                                  soma((E(j,3)),k+1) = 1;
52                          end
53                  end
54                  k = k + 2;
55          end
56
57          for j = 1:ne
58                  if(E(j,1) == 1)
59                          soma((E(j,3)),1) = 1;
60                  end
61                  if(E(j,2) == nv)
62                          soma((E(j,3)),((nv-2)*2 + 2)) = 1;
63                  end
64          end
```

```
65
66          Av = [-1, 0; 1, 0; 0,-1,; 0, 1];
67          b = zeros(4,nv);
68          lc = input('Introduza a largura dos conjuntos: ');
69          ac = lc;
70          espH = input('Introduza o espacamento horizontal entre
               os conjuntos: ');
71          espV = input('Introduza o espacamento vertical entre os
               conjuntos: ');
72          %Definicao da malha
73          k = 1;
74          n2 = 0;
75          m2 = 1;
76          for i=1:a
77                  n1 = 0;
78                  m1 = 1;
79                  for j=1:l
80                          b(1,k) = (n1*lc + n1*espH)*-1;
81                          b(2,k) = m1*lc + n1*espH;
82                          b(3,k) = (n2*ac + n2*espV)*-1;
83                          b(4,k) = m2*ac + n2*espV;
84                          k = k+1;
85                          n1 = n1 + 1;
86                          m1 = m1 + 1;
87                  end
88                  n2 = n2 + 1;
89                  m2 = m2 + 1;
90          end
91
92          echo on
93          cvx_begin
94                  obj=0;
95                  variable phi(ne) binary
96                  variable z(ne,2)
97                  variable y(ne,2)
98                  for i=1:ne
99                          obj=obj+norm(z(i,:)-y(i,:));
100                 end
101                 minimize obj
102                 subject to
103                         p = 1;
104                         for i = 1:ne
```

```
                                    Av * (y(i,:))' <= b(:,E(p,1)) *
                                        phi(i);
                                    Av * (z(i,:))' <= b(:,E(p,2)) *
                                        phi(i);
                                    p = p + 1;
                            end
                            soma(:,(nv-2)*2 + 2)' * phi == soma(:,1)
                                ' * phi == 1;
                            q = 2;cout=0;
                            while(cout~=nv-2)
                                    %soma(:,q+1)' * phi <=1;
                                    soma(:,q)' * phi == soma(:,q+1)'
                                        * phi;
                                    soma(:,q)' * z == soma(:,q+1)' *
                                        y;
                                    q = q + 2;
                                    cout = cout + 1;
                            end
        cvx_end

        phiF = phi'
        echo off
```

Por fim, o código que se apresenta a seguir corresponde ao que foi elaborado para problemas gerados aleatoriamente, conforme as instruções dadas na secção 'Resultados Numéricos'.

```
cvx_solver mosek

nv = input('Introduza o numero de vertices do grafo: ');
na = input('Introduza o numero de arestas do grafo: ');
A = input('Introduza a area dos conjuntos: ');
lArea = sqrt(A);

val_ot = zeros(100,2);
t_ot = zeros(100,2);
kOT = 1;
for i=1:100
        M = zeros(nv,nv);
        et = zeros(nv,2);
        et(1,2) = 1; et(nv,2) = 1;
        for i = 1:nv
                et(i,1) = i;
        end
```

```
18
19          count = 2;
20          nv_aux = nv;
21          na_aux = 0;
22
23          while count ~= nv
24                  nC = randi(nv_aux-2);
25                  p = 1;
26                  for j = 1:nC
27                          cj = 1;
28                          while et(cj,2) == 1
29                                  cj = randi([2,nv]);
30                          end
31                          M(p,cj) = 1; et(cj,2) = 1;
32                          p = cj;
33                          count = count +1;
34                          nv_aux = nv_aux - 1;
35                          na_aux = na_aux +1;
36                  end
37                  M(p,nv) = 1;
38                  na_aux = na_aux +1;
39          end
40
41          while (na_aux ~= na)
42                  x=0;y=0;
43                  while (x==y) || (M(x,y)==1) || (x==nv) || (y==1)
44                          x = randi(nv);
45                          y = randi(nv);
46                  end
47                  M(x,y) = 1;
48                  na_aux = na_aux + 1;
49          end
50
51          Av = [-1, 0; 1, 0; 0,-1,; 0, 1];
52          b = zeros(4,nv);
53          b(1,nv) = -1; b(2,nv) = 1;
54          b(3,nv) = -1; b(4,nv) = 1;
55          for j = 2:nv-1
56                  bx = rand(1,nv-2); by = rand(1,nv-2);
57                  b(1,j) = bx(1,j-1) - lArea/2;
58                  b(2,j) = bx(1,j-1) + lArea/2;
59                  b(3,j) = by(1,j-1) - lArea/2;
```

```
60                b(4,j) = by(1,j-1) + lArea/2;
61        end
62
63        E = zeros(na,3);
64        k = 1;
65        for i=1:nv
66                for j=1:nv
67                        if(M(i,j) == 1)
68                                E(k,1) = i;
69                                E(k,2) = j;
70                                E(k,3) = k;
71                                k = k+1;
72                        end
73                end
74        end
75
76        soma = zeros(na,(nv-2)*2 + 2);
77        k = 2;
78        for i = 2:nv-1
79                for j = 1:na
80                        if(E(j,2) == i)
81                                soma((E(j,3)),k) = 1;
82                        end
83                        if(E(j,1) == i)
84                                soma((E(j,3)),k+1) = 1;
85                        end
86                end
87                k = k + 2;
88        end
89
90        for j = 1:na
91                if(E(j,1) == 1)
92                        soma((E(j,3)),1) = 1;
93                end
94                if(E(j,2) == nv)
95                        soma((E(j,3)),((nv-2)*2 + 2)) = 1;
96                end
97        end
98
99        tStart = cputime;
100       echo on
101       cvx_begin
```

```
102                 obj=0;
103                 variable phi(na) binary
104                 variable z(na,2)
105                 variable y(na,2)
106                 for i=1:na
107                         obj=obj+norm(z(i,:)-y(i,:));
108                 end
109                 minimize (obj)
110                         subject to
111                         p = 1;
112                         for i = 1:na
113                                 Av * (y(i,:))' <= b(:,E(p,1)) *
                                       phi(i);
114                                 Av * (z(i,:))' <= b(:,E(p,2)) *
                                       phi(i);
115                                 p = p + 1;
116                         end
117                         soma(:,(nv-2)*2 + 2)' * phi == soma(:,1)
                               ' * phi == 1;
118                         l = 2;cout=0;
119                         while(cout~=nv-2)
120                                 %Restricao para grafos ciclicos
121                                 soma(:,l+1)' * phi <=1;
122                                 soma(:,l)' * phi == soma(:,l+1)'
                                        * phi;
123                                 soma(:,l)' * z == soma(:,l+1)' *
                                        y;
124                                 l = l + 2;
125                                 cout = cout + 1;
126                         end
127         cvx_end
128         tEnd = cputime - tStart;
129
130         phiF = phi';
131         echo off
132
133         tStartR = cputime;
134         echo on
135         cvx_begin
136                 objR=0;
137                 variable phiR(na)
138                 variable zR(na,2)
```

```
139                 variable yR(na,2)
140                 for i=1:na
141                         objR=objR+norm(zR(i,:)-yR(i,:));
142                 end
143                 minimize (objR)
144                 subject to
145                         p = 1;
146                         for i = 1:na
147                                 Av * (yR(i,:))' <= b(:,E(p,1)) *
                                        phiR(i);
148                                 Av * (zR(i,:))' <= b(:,E(p,2)) *
                                        phiR(i);
149                                 p = p + 1;
150                         end
151                         soma(:,(nv-2)*2 + 2)' * phiR == soma
                               (:,1)' * phiR == 1;
152                         l = 2;cout=0;
153                         while(cout~=nv-2)
154                                 %Restricao para grafos ciclicos
155                                 soma(:,l+1)' * phiR <=1;
156                                 soma(:,l)' * phiR == soma(:,l+1)
                                        ' * phiR;
157                                 soma(:,l)' * zR == soma(:,l+1)'
                                        * yR;
158                                 l = l + 2;
159                                 cout = cout + 1;
160                         end
161         cvx_end
162         phiFR = phiR';
163         echo off
164
165         tEndR = cputime - tStartR;
166         val_ot(kOT,1) = obj;
167         val_ot(kOT,2) = objR;
168         t_ot(kOT,1) = tEnd;
169         t_ot(kOT,2) = tEndR;
170         kOT = kOT + 1;
171 end
172
173 vFINAL = round(val_ot(:,1) - val_ot(:,2),4);
174 vMax = max(vFINAL);
175 tFINAL = sum(t_ot(:,1))/100;
```

```
tFINALR = sum(t_ot(:,2))/100;
tMax = max(t_ot(:,1));
tMaxR = max(t_ot(:,2));
```

Para os testes efetuados no ambiente tridimensional adaptou-se o código anterior, tendo-se obtido o seguinte:

```
cvx_solver mosek

nv = input('Introduza o numero de vertices do grafo: ');
na = input('Introduza o numero de arestas do grafo: ');
A = input('Introduza a area dos conjuntos: ');
lArea = sqrt(A);

val_ot = zeros(100,2);
t_ot = zeros(100,2);
kOT = 1;
for i=1:100
        M = zeros(nv,nv);
        et = zeros(nv,2);
        et(1,2) = 1; et(nv,2) = 1;
        for i = 1:nv
                et(i,1) = i;
        end
        count = 2;
        nv_aux = nv;
        na_aux = 0;

        while count ~= nv
                nC = randi(nv_aux-2);
                p = 1;
                for j = 1:nC
                        cj = 1;
                        while et(cj,2) == 1
                                cj = randi([2,nv]);
                        end
                        M(p,cj) = 1; et(cj,2) = 1;
                        p = cj;
                        count = count +1;
                        nv_aux = nv_aux - 1;
                        na_aux = na_aux +1;
                end
                M(p,nv) = 1;
```

```
37              na_aux = na_aux +1;
38          end
39
40          while (na_aux ~= na)
41              x=0;y=0;
42              while (x==y) || (M(x,y)==1) || (x==nv) || (y==1)
43                  x = randi(nv);
44                  y = randi(nv);
45              end
46              M(x,y) = 1;
47              na_aux = na_aux + 1;
48          end
49
50          Av = [-1,0,0;1,0,0;0,-1,0;0,1,0;0,0,-1;0,0,1];
51          b = zeros(6,nv);
52          b(1,nv) = -1; b(2,nv) = 1;
53          b(3,nv) = -1; b(4,nv) = 1;
54          b(5,nv) = -1; b(6,nv) = 1;
55          for j = 2:nv-1
56              bx = rand(1,nv-2); by = rand(1,nv-2);
57              bz = rand(1,nv-2);
58              b(1,j) = bx(1,j-1) - lArea/2;
59              b(2,j) = bx(1,j-1) + lArea/2;
60              b(3,j) = by(1,j-1) - lArea/2;
61              b(4,j) = by(1,j-1) + lArea/2;
62              b(5,j) = bz(1,j-1) - lArea/2;
63              b(6,j) = bz(1,j-1) + lArea/2;
64          end
65
66          E = zeros(na,3);
67          k = 1;
68          for i=1:nv
69              for j=1:nv
70                  if(M(i,j) == 1)
71                      E(k,1) = i;
72                      E(k,2) = j;
73                      E(k,3) = k;
74                      k = k+1;
75                  end
76              end
77          end
78
```

```
79          soma = zeros(na,(nv-2)*2 + 2);
80          k = 2;
81          for i = 2:nv-1
82                  for j = 1:na
83                          if(E(j,2) == i)
84                                  soma((E(j,3)),k) = 1;
85                          end
86                          if(E(j,1) == i)
87                                  soma((E(j,3)),k+1) = 1;
88                          end
89                  end
90                  k = k + 2;
91          end
92
93          for j = 1:na
94                  if(E(j,1) == 1)
95                          soma((E(j,3)),1) = 1;
96                  end
97                  if(E(j,2) == nv)
98                          soma((E(j,3)),((nv-2)*2 + 2)) = 1;
99                  end
100         end
101
102         tStart = cputime;
103         echo on
104         cvx_begin
105                 obj=0;
106                 variable phi(na) binary
107                 variable z(na,3)
108                 variable y(na,3)
109                 for i=1:na
110                         obj=obj+norm(z(i,:)-y(i,:));
111                 end
112                 minimize (obj)
113                 subject to
114                         p = 1;
115                         for i = 1:na
116                                 Av * (y(i,:))' <= b(:,E(p,1)) *
                                        phi(i);
117                                 Av * (z(i,:))' <= b(:,E(p,2)) *
                                        phi(i);
118                                 p = p + 1;
```

```
119                         end
120                         soma(:,(nv-2)*2 + 2)' * phi == soma(:,1)
                               ' * phi == 1;
121                         l = 2;cout=0;
122                         while(cout~=nv-2)
123                         %Restricao para grafos ciclicos
124                                 soma(:,l+1)' * phi <=1;
125                                 soma(:,l)' * phi == soma(:,l+1)'
                                        * phi;
126                                 soma(:,l)' * z == soma(:,l+1)' *
                                        y;
127                                 l = l + 2;
128                                 cout = cout + 1;
129                         end
130         cvx_end
131         tEnd = cputime - tStart;
132
133         phiF = phi';
134         echo off
135
136         tStartR = cputime;
137         echo on
138         cvx_begin
139                 objR=0;
140                 variable phiR(na)
141                 variable zR(na,3)
142                 variable yR(na,3)
143                 for i=1:na
144                         objR=objR+norm(zR(i,:)-yR(i,:));
145                 end
146                 minimize (objR)
147                 subject to
148                         p = 1;
149                         for i = 1:na
150                                 Av * (yR(i,:))' <= b(:,E(p,1)) *
                                        phiR(i);
151                                 Av * (zR(i,:))' <= b(:,E(p,2)) *
                                        phiR(i);
152                                 p = p + 1;
153                         end
154                         soma(:,(nv-2)*2 + 2)' * phiR == soma
                               (:,1)' * phiR == 1;
```

```
155                             l = 2;cout=0;
156                             while(cout~=nv-2)
157                             %Restricao para grafos ciclicos
158                                     soma(:,l+1)' * phiR <=1;
159                                     soma(:,l)' * phiR == soma(:,l+1)
                                            ' * phiR;
160                                     soma(:,l)' * zR == soma(:,l+1)'
                                            * yR;
161                                     l = l + 2;
162                                     cout = cout + 1;
163                             end
164         cvx_end
165         phiFR = phiR';
166         echo off
167
168         tEndR = cputime - tStartR;
169         val_ot(kOT,1) = obj;
170         val_ot(kOT,2) = objR;
171         t_ot(kOT,1) = tEnd;
172         t_ot(kOT,2) = tEndR;
173         kOT = kOT + 1;
174 end
175
176 vFINAL = round(val_ot(:,1) - val_ot(:,2),4);
177 vMax = max(vFINAL);
178 tFINAL = sum(t_ot(:,1))/100;
179 tFINALR = sum(t_ot(:,2))/100;
180 tMax = max(t_ot(:,1));
181 tMaxR = max(t_ot(:,2));
```