1 Programação Linear

1.1 Otimização

Uma possível formulação computacional genérica para um problema de otimização é:

Dado um conjunto F, que representa o domínio do problema, e uma função $c : F \to \mathbb{R}$, que representa o custo associado a um certo elemento.

Obter um elemento $x \in F$, tal que $c(x)$ é o melhor possível.

Quando o domínio F é um subespaço de $\mathbb{R}^n$ e pode ser definido por equações e inequações lineares e, além disso, a função de custo c também é linear, o problema de otimização pode ser tratado como um **Programa Linear (PL)**.

1.2 Definições de Programação Linear

No âmbito de programação linear, diversos nomes específicos são atribuídos aos conceitos genéricos definidos anteriormente. O domínio F passa a ser definido através de sistemas de inequações através de um vetor $x \in \mathbb{R}^n$ representando as variáveis, uma matriz $A \in \mathbb{R}^{mxn}$ e um vetor $b \in \mathbb{R}^m$ representando respectivamente os coeficientes e os termos independentes do sistema.

Desta forma, o domínio F do problema passa a ser um **politopo**. A definição abaixo formaliza o conceito e um exemplo é descrito na Figura 1.

Definição 1.1. *F é um politopo, se e somente se F pode ser definido da seguinte forma:*

$$F = \{x \in \mathbb{R}^n | Ax \leq b\}$$

Figura 1 – Politopo definido algebricamente

$$\begin{aligned} z + x &\leq 3 \\ x &\leq 2 \\ y &\leq 2 \\ z &\leq 1.5 \\ x, y, z &\geq 0 \end{aligned}$$

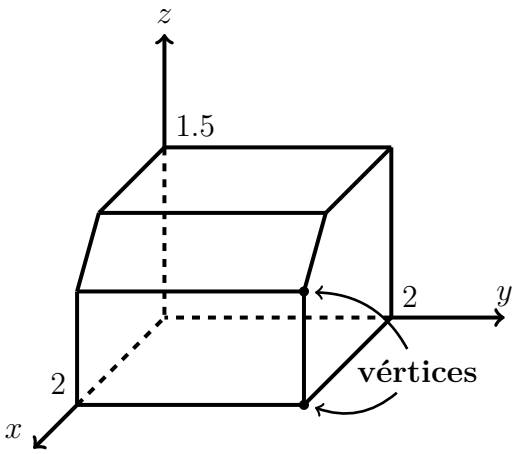
Fonte: o autor

Um politopo também pode ser definido através da combinação convexa de seus **vértices**, como visto na Figura 2.

Definição 1.2. *Um vértice de um politopo F é um ponto $x \in F$ que não é combinação convexa de outros diferentes pontos, ou seja*

$$x = \sum_{i=1}^{k} \lambda_i x_i,\ \lambda \geq 0\ e\ \sum_{i=1}^{k} \lambda_i = 1 \implies x_1 = \cdots = x_k = x$$

Figura 2 – Politopo da Figura 1 definido geométricamente



Fonte: o autor

Informalmente, os vértices de um politopo são as suas *pontas*.

Como a função de custo c é linear, ela pode ser representada com um vetor em $\mathbb{R}^n$, onde o custo de cada variável é calculado através do produto $c^T x$.

Muitas vezes, além de inequações são necessárias também equações para modelar algum aspecto da realidade, com isso em mente, a definição abaixo é apresentada.

Definição 1.3. *Um PL na* ***forma geral*** *é dado por*

$$\begin{array}{ll} min/max & c^T x \\ a_i^T x = b_i & i \in M \\ a_i^T x \geq b_i & i \in \bar{M} \\ x_j \geq 0 & j \in N \\ x_j \lesseqgtr 0 & j \in \bar{N} \end{array}$$

onde a_i são colunas da matriz de restrições $A \in \mathbb{R}^{m \times n}$, x_j são elementos do vetor de variáveis $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $M, \bar{M}$ representam as linhas da matriz de restrições correspondentes a equações e inequações, respectivamente, e $N, \bar{N}$ representam as variáveis positivas e irrestritas, respectivamente.

Porém, para formalizar a representação de um PL e facilitar a aplicação de algumas técnicas, geralmente os problemas são representados usando um formato padrão.

Definição 1.4. *Um PL na* ***forma padrão*** *é dado por*

$$\begin{aligned} &min\ c^T x \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

onde $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ *e* $b \in \mathbb{R}^m$.

É importante ressaltar que apesar de restringir o politopo com igualdades, tal abordagem é equivalente a Definição 1.1. As operações para transformar inequações em equações serão descritas ainda neste capítulo.

Um programa linear é dito **viável** se existir algum ponto viável $x \in F$, ou seja, se $F \neq \phi$, caso contrário ele é dito **inviável**. Um PL também pode ser classificado de acordo com suas limitações: se existe um limitante $z \in \mathbb{R}$ tal que $z > c^T x$, para todo $x \in F$, ele é dito **limitado**, caso contrário é **ilimitado**.

1.3 Transformações em Programas Lineares

A primeira vista, pode parecer que a forma padrão de um programa linear não é abrangente o suficiente para abrigar todos os possíveis sistemas, já que não contém restrições na forma de inequação, além de assumir que todas as variáveis são positivas e que o problema é sempre de minimização. Porem, existem algumas transformações algébricas que podem ser aplicadas em qualquer PL para deixá-lo na forma padrão.

Uma característica necessária destas transformações é que elas não alterem o domínio do problema e, como consequência disto, a solução de um PL antes e depois das transformações é a mesma. Segue uma lista das transformações:

1. Caso o problema se trate de uma maximização, basta multiplicar o vetor, dos coeficientes de custo, c por -1, ou seja

   $$\max c^T x \equiv \min -c^T x$$

   tal operação é feita possível pelo fato da função de custo ser linear.

2. Caso existam restrições na forma

   $$a_i x \leq (\geq)\, b_i$$

   onde a_i e b_i representam linhas como na forma canônica, basta inserir uma nova variável positiva s_i e somá-la (caso $\leq$) ou subtraí-la (caso $\geq$) da inequação:

   $$\begin{aligned} a_i x \leq (\geq)\, b_i \quad \equiv \quad & a_i x + (-)\, s_i = b_i \\ & s_i \geq 0 \end{aligned}$$

3. Caso exista alguma variável x_i que seja negativa, basta substituí-la por uma nova x'_i tal que $x'_i = -x_i$ e inverter o sinal i-ésima coluna de A:

$$\begin{array}{lcl} Ax = b & \equiv & A'x' = b \\ x_i \leq 0 & & x'_i \geq 0 \end{array}$$

onde A' é a matriz de restrições A com os sinais invertidos na i-ésima coluna.

4. Caso exista alguma variável x_i irrestrita, basta criar duas novas variáveis positivas x_i^+ e x_i^-, tal que $x_i = x_i^+ - x_i^-$:

$$\begin{array}{lcl} Ax = b & \equiv & \sum_{j=1}^{i-1} A_j x_j + A_i x_i^+ - A_i x_i^- + \sum_{j=i+1}^{n} A_j x_j = b \\ x_i \lesseqgtr 0 & & x_i^+, x_i^- \geq 0 \end{array}$$

onde A_i representa uma coluna da matriz A.

1.4 Complexidade

Após definidas as estruturas da programação linear surge a questão de quão complexa é a sua resolução.

Todos os problemas computacionais podem ser classificados segundo sua dificuldade, ou especificamente, quanto tempo será necessário para resolve-lo, onde tempo se refere a uma quantidade de operações realizadas pelo computador. A abordagem à complexidade computacional neste trabalho será simplista e definições formais sobre os esquemas de representação e modelos de computação, que seriam necessários para abordar formalmente o tema, não serão apresentados e podem ser encontrados em Papadimitrou e Steiglitz (1998) e Papadimitriou (1994).

Uma notação bastante usada no âmbito de complexidade computacional é a função de limitação superior assintótica O que é formalizada na Definição 1.5.

Definição 1.5. *Sejam duas funções $f, g : \mathbb{Z}^+ \to \mathbb{R}^+$, diz-se que $f(n) = O(g(n))$ quando existe uma constante c tal que, para um n grande o suficiente, $f(n) \leq cg(n)$.*

Com isto em mente, os problemas podem ser postos em duas classes: a classe P, em que os problemas podem ser resolvidos e verificados em *tempo* polinomial no tamanho da entrada e a classe NP onde os problemas precisam apenas ser verificáveis em tempo polinomial no tamanho da entrada, ou seja $P \subset NP$.

A classe dos NP-completos, as vezes referenciada por NPc, é formada por problemas da classe NP que resolvem todos os problemas desta classe, ou seja, se $P \in NPc$, então para todo problema $P' \in NP$ existe uma transformação (redução) de custo polinomial que

possibilita a resolução de P' através de P. Pela definição desta subclasse, conclui-se que todos os problemas pertencentes a ela são redutíveis entre sí.

Outra classe importante é a dos problemas *NP*-difíceis que possui uma definição analoga a *NP*-completa, porém, os problemas são geralmente de otimização, ou seja, não são verificáveis em tempo polinomial.

Teorema 1.1. *A programação linear pertence pode ser resolvida em tempo polinomial, ou seja, pertence a classe P.*

A ideia por trás da prova do Teorema 1.1 é a redução de um PL para um problema de **Inequações Lineares (IL)** através do uso de busca binária e de alguns cálculos referentes ao tamanho e a racionalidade do PL. Usando o princípio da redução, como IL pertence a classe P, então conclui-se que PL pertence a classe P. A prova completa pode ser encontrada em Papadimitrou e Steiglitz (1998, p. 171).

1.5 Dual

O dual de um Programa Linear reflete a ideia de criar limitantes para suas soluções através da combinação linear de suas restrições. Com isso em mente, todo PL tem associado a si outro PL denominado **dual**, cuja função é buscar o melhor limitante possível e, caso tal limitante exista, como será demonstrado nesta sessão, coincidirá com a solução ótima do problema original também referido como **primal**.

A derivação do dual de um PL na forma padrão é caracterizada na Definição 1.6.

Definição 1.6. *A dualização de um PL na forma padrão é dada por*

$$\begin{array}{lcl} min\ c^T x & & max\ b^T \pi \\ Ax = b & \overset{dual}{\Longrightarrow} & A^T \pi \leq c \\ x \geq 0 & & \end{array}$$

onde $\pi \in \mathbb{R}^m$ é o vetor de coeficientes da combinação linear das restrições do primal.

É possível derivar diretamente o dual de um Programa Linear, mesmo que ele não se encontre na forma padrão. De fato, qualquer PL na forma geral pode ser derivado para seu dual diretamente, através da Tabela 1.

Para explicitar algumas das características dos duais, existem três importantes teoremas:

Teorema 1.2. *O dual do dual é o primal.*

Tabela 1 – Transformações duais.

Primal		Dual
$\min c^T x$		$\max b^T \pi$
$a_i x = b_i$	$i \in M$	$\pi_i \lesseqgtr 0$
$a_i x \geq b_i$	$i \in \bar{M}$	$\pi_i \geq 0$
$x_j \geq 0$	$j \in N$	$\pi^T A_j \leq c_j$
$x_j \lesseqgtr 0$	$j \in \bar{N}$	$\pi^T A_j = c_j$

Fonte: (PAPADIMITROU; STEIGLITZ, 1998, p. 84)

Demonstração. Escrevendo o dual de um PL, na forma geral, da maneira abaixo:

$$\begin{aligned} &\min \pi^T(-b) \\ (-A_j^T)\pi &\geq -c_j \quad j \in N \\ (-A_j^T)\pi &= -c_j \quad j \in \bar{N} \\ \pi_i &\geq 0 \qquad i \in \bar{M} \\ \pi_i &\lesseqgtr 0 \qquad j \in M \end{aligned}$$

e aplicando novamente a derivação de dual, como descrito na tabela acima obtêm-se:

$$\begin{aligned} &\max x^T(-c) \\ x_j &\geq 0 \qquad j \in N \\ x_j &\lesseqgtr 0 \qquad j \in \bar{N} \\ -a_i^T x &\leq -b_i \quad i \in \bar{M} \\ -a_i^T x &= -b_i \quad i \in M \end{aligned}$$

resultando no PL inicial na forma geral. □

Teorema 1.3. *Dado um PL na forma padrão e seu dual. Se $x^* \in \mathbb{R}^n$ e $\pi^* \in \mathbb{R}^m$ sao duas soluções para o primal e o dual respectivamente, então $c^T x^* \geq b^T \pi^*$.*

Prova da dualidade fraca. Se x^* é uma solução para o primal, então $Ax^* = b$, logo $(Ax^*)^T = b^T$, ou seja $x^{*T} A^T = b^T$. Multiplicando a solução do dual π^* de ambos os lados da equação temos $x^{*T} A^T \pi^* = b^T \pi^*$. Como $A^T \pi^* \leq c$ então $b^T \pi^* = x^{*T} A^T \pi^* \leq x^{*T} c$, logo $c^T x^* \geq b^T \pi^*$. □

Teorema 1.4. *Dado um PL na forma padrão e seu dual. Se o PL primal for viável e limitado, então existe uma solução viável x^* para o primal e π^* para o dual e $c^T x^* = b^T \pi^*$*

A prova do Teorema 1.4 geralmente envolve o uso da solução providenciada pelo método de resolução de programas lineares chamado simplex, que será apresentado resumidamente ainda neste capítulo. Logo, a prova de tal relação não será explicitada neste trabalho e pode ser encontrada em Papadimitrou e Steiglitz (1998, p. 69).

Decorrente da combinação dos resultados dos Teoremas 1.2, 1.3 e 1.4, existem 4 possíveis combinações de situações de viabilidade e limitações para os problemas primal e dual:

1. **Primal dual viáveis e limitados**: decorre diretamente do Teorema 1.4.
2. **Primal viável e ilimitado** e **dual inviável**: se o primal é ilimitado significa que $\nexists M \in \mathbb{R}$ tal que $c^T x \geq M$ para todas as soluções possíveis do primal. Porém o Teorema 1.3 afirma que $c^T x \geq b^T y$ para toda solução y do dual, logo, o dual não possui soluções viáveis.
3. **Primal inviável** e **dual viável e ilimitado**: como o dual do dual é o primal novamente, a prova do caso 2 se aplica a este caso.
4. **Primal inviável** e **dual inviável**: Tal caso é exemplificado em:

$$\begin{array}{rcr} \max x_1 & & \min \pi_1 + \pi_2 \\ x_1 + x_2 \geq 1 & & \pi_1 - \pi_2 = 1 \\ -x_1 - x_2 \geq 1 & \text{e o dual} & \pi_1 - \pi_2 = 0 \\ x_1 \lneqq 0 & & \pi_1 \geq 0 \\ x_2 \lneqq 0 & & \pi_2 \geq 0 \end{array}$$

   ambos claramente inviáveis. (PAPADIMITROU; STEIGLITZ, 1998, p. 71)

A dualidade forte também oferece a capacidade para os resolvedores de programas lineares oferecerem **certificados** de otimalidade, ou seja, um resolvedor pode devolver a solução ótima dual junto com a primal, para que caso deseje-se conferir a otimalidade da solução primal, basta aplicar a solução dual e verificar a igualdade dos valores.

Um outro importante teorema referente a dualidade de programas lineares é o da **folga complementar** que relaciona as restrições mais restritivas do primal com as menos restritivas do dual.

Teorema 1.5. *Um par variáveis $x^* \in \mathbb{R}^n$, $\pi_i^* \in \mathbb{R}^m$ são soluções ótimas para o primal e o dual, respectivamente, de um problema linear P, se e somente se*

$$u_i = \pi_i^*({a_i}^T x^* - b_i) = 0 \qquad \textit{para } i = 1, \ldots, m$$

ou seja

$$\pi_i^* \neq 0 \implies {a_i}^T x^* = b_i$$

Demonstração. Para o caso em que o problema P está na forma padrão, ou seja, todas suas restrições sao de igualdade, o teorema enunciado é válido, já que ${a_i}^T x = b_i$ para todo x válido.

No caso em que P é definido através de inequações, ou seja, $P = \min \{c^T x \mid Ax \geq b, x \in \mathbb{R}^n\}$, então seu dual é $D = \max \{b^T \pi \mid A^T \pi = c, \pi \geq 0, \pi \in \mathbb{R}^m\}$.

Para provar por contradição, basta supor que existe um índice $1 \leq i \leq m$ tal que $\pi \neq 0$ e ${a_i}^T x^* > b_i$, onde x^* é uma solução ótima de P. Se tal índice existe e seja π^* a solução ótima do dual, então

$$Ax^* > b$$
$$(Ax^*)^T > b^T$$
$$(Ax^*)^T \pi^* > b^T \pi^*$$
$$x^{*T}(A^T \pi^*) > b^T \pi^*$$
$$x^{*T} c > b^T \pi^*$$

ou seja

$$c^T x^* \neq b^T \pi^*$$

o que contradiz o Teorema 1.4. □

1.6 Resolvendo Problemas Lineares

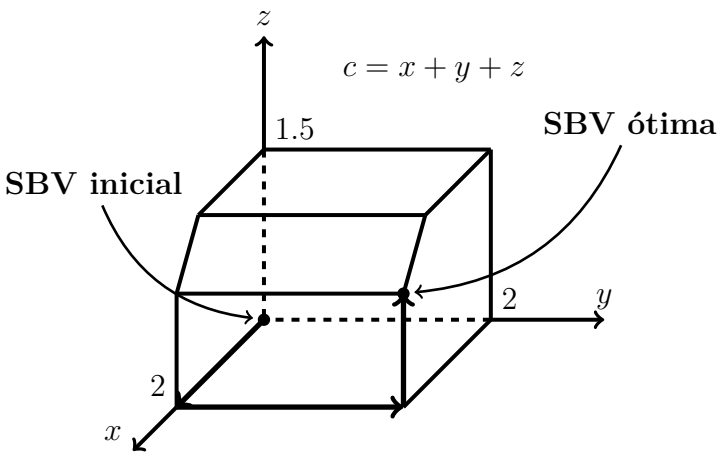
Existem diversas maneiras diferentes de resolver programas lineares, a mais comumente usada e estudada é o método **Simplex**, que baseia-se na topologia do politopo (como na Figura 2) para caminhar entre vértices, sempre buscando um vértice com o custo melhor. Uma importante premissa para este método é a de que caso o problema tenha só uma solução ótima, ela será um vértice e caso tenha múltiplas soluções, alguma delas será um vértice.

Como descrito, o Simplex, utiliza o conceito de **solução básica viável (SBV)** para ser referir aos vértices, que possuem, por sua vez, uma característica de que apenas as **variáveis básicas**, que variam conforme o vértice, possuem valores que podem ser diferentes de 0. Assim, uma SBV pode ser definida através das suas variáveis básicas e seus valores.

A quantidade de variáveis básicas que as SBVs de um PL possuem é igual ao número m de restrições que definem o politopo, com excessão do caso em que $m > n$, onde n é a dimensão da variável x, o que pode ser resolvido eliminando restrições que são combinações lineares das demais.

Para caminhar entre vértices, o método escolhe uma variável x_i que não seja básica e cujo custo parcial represente um ganho para a solução atual. Após isso é escolhido um valor θ que representará quanto a SBV poderá *'se mover'* continuando viável. Após aplicado este valor a solução atual, alguma variáveis básica se torna 0 e deixa de ser básica, cedendo seu lugar para a variável x_i. Ao terminar a substituição de variáveis básicas, o algoritmo recalcula os custos parciais das variávei não básicas e novamente realiza outra atualização da solução atual. Tal processo é exemplificado na Figura 3.

Figura 3 – Execução do simplex para o politopo da Figura 1 com uma função de custo c



Fonte: o autor

Para impedir ciclos na substituição de variáveis, que ocorreriam quando as mesmas variáveis são substituídas para fora e para dentro novamente indeterminadamente, o simplex usa uma regra auxiliar para a escolha da variável não básica chamada **Regra de Bland**, tal processo só é necessário caso o politopo seja degenerado.

O simplex também possui um mecanismo interno que permite detectar o caso em que o PL é ilimitado, encerrando assim a execução. No caso em que o PL é limitado, o algoritmo termina quando nenhuma variável não básica apresenta um custo parcial lucrativo para a solução atual.

Para realizar todas estas operações, o simplex usa um mecanismo de **tábuas**, que são matrizes estruturadas que armazenam as informações necessárias e são atualizadas durante o algoritmo. Inicialmente a tábua contém a matriz de restrições do PL, as variáveis básicas e seus valores, os custos parciais das variáveis não básicas e o custo da solução inicial. Mais informações sobre o Simplex podem ser encontradas em Papadimitrou e Steiglitz (1998, c. 2).

Uma importante extensão deste método é o **Simplex Revisado** cuja principal característica é a de que apenas uma porção chamada **carry** (proporcional ao do número de restrições) da tábua mencionada anteriormente é mantida explicitamente, onde o restante das informações são calculadas quando necessário. Tal abordagem oferece uma significativa economia de memória no caso em que o programa linear possui muitas restrições e variáveis, porém são necessários diversos cálculos para encontrar as variáveis lucrativas, e quando estas forem achadas é necessário gerar as informações referentes a ela para entrar na matriz CARRY. (PAPADIMITROU; STEIGLITZ, 1998)

Outra técnica para auxiliar a resoluções de grandes programas lineares é a **Decom-**

posição de Dantzig-Wolfe, que é baseado na ideia de dividir um PL em diversos PLs menores que estão relacionados através de algumas restrições. Neste método as variáveis são redefinidas como combinações convexas de todos os vértices do politopo.

A decomposição é aplicada em problemas cuja matriz de restrição pode ser organizada, reordenando linhas e colunas, em uma matriz *bloco angular*, como na Figura 4.

Figura 4 – Exemplo de matriz bloco angular

$$R = \begin{pmatrix} D_1 & D_2 & D_3 \\ A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{pmatrix}$$

Fonte: (PAPADIMITROU; STEIGLITZ, 1998, p. 97)

Na Figura 4, D_i, A, B e C representam uma submatrizes de R, onde, no caso em que R é uma matriz de restrições, D_i representa restrições de ligação e as demais representam os subproblemas.

Apesar de reduzir a quantidade de restrições, a decomposição de Dantzig-Wolfe aumenta drásticamente o número de variáveis do problema. Tal problema é contornado através do uso do simplex revisado mencionado acima, porém com a variação de que para encontrar qual variável deve se tornar básica, ao invés de calcular o custo de todas as variáveis não básicas, como é o caso do simplex revisado, na decomposição de Dantzig-Wolfe esta variável é encontrada através da resolução de PLs que representam subproblemas, ou submatrizes, representados na Figura 4. Um possível algoritmo para implementar esta decomposição é encontrado em Papadimitrou e Steiglitz (1998, p. 101) e uma exemplificação interativa é encontrada em Dantzig (1963, p. 455).

Outras importantes extensões do Simplex são: o **Simplex de Duas Fases** que é utilizado quando uma solução inicial não é conhecida ou trivialmente derivada e o **Simplex Dual** que resolve o problema original através do seu dual, o que é especificamente útil quando uma solução primal inviável e dual viável.

O método das elipsóides surge diretamente da prova de que a programação linear possui uma solução polinomial (Teorema 1.1), ou seja, ele é baseado na busca binária nas soluções de diversos sistemas de inequações lineares.

O algoritmo usado pelo método das elipsóides para resolver as ILs é iterativo e baseado na ideia de encolher elipsóides, que sempre contém uma solução para o sistema de inequações (se existir alguma), até que a elipsóide atual seja pequena o suficiente para derivar a solução.

Para apresentação completa do funcionamento das elipsóides são necessários diversas definições e conceitos de algebra que não serão aqui apresentados e, em conjunto com um algoritmo para implementar o método e uma prova de que o mesmo é de custo polinomial, podem ser encontrados Papadimitrou e Steiglitz (1998, s. 8.7).

Apesar da programação linear pertencer a classe P (Teorema 1.1), o simplex, no seu pior caso, que pode ser encontrado em Papadimitrou e Steiglitz (1998, p. 166), não possui tempo de execução polinomial, porém ainda é o método mais utilizado para resolver PLs. Tal fato exemplifica uma controvérsia em que algoritmos de tempo de execução não-polinomiais são mais práticos que os polinomiais, já que mesmo com a criação de um método de custo polinomial (método das elipsóides) , o simplex continua apresentando um melhor desempenho.

2 Programação Inteira

2.1 Definição de Programação Inteira

Uma importante subcategoria dos problemas de otimização é a dos **Programas Inteiros Mistos (PIM)** que, apesar de ser muito semelhante a Programação Linear em sua formulação, demanda um esforço computacional muito maior, como será mostrado neste capítulo. Uma formulação genérica baseada na forma padrão de um PL é:

$$\begin{aligned} &\min c'x \\ &Ax = b \\ &x \geq 0 \\ &x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \end{aligned}$$

onde uma nova variável $p \in \mathbb{N} < n$ representa a quantidade de variáveis discretas. Nota-se que caso $p = 0$ o problema se trata de um Programa Linear usual, porém, se $p = n$ ele é denominado **Programa Inteiro (PI)**.

Analogamente à programação linear, um formato padrão para um programa inteiro é apresentado na Definição 2.1.

Definição 2.1. *Um PI na* ***forma padrão*** *é dado por*

$$\begin{aligned} &min\ c^T x \\ &Ax = b \\ &x \in \mathbb{Z}^+ \end{aligned}$$

onde $A \in \mathbb{R}^{m \times n}$.

Uma característica atípica de um PIM é que caso existam coeficientes irracionais na matriz de restrições, ele pode ser viável e limitado, porém não possuir uma solução ótima (KRUMKE, 2006, p. 3).

Grande parte dos problemas na área de ciência da computação são ainda uma subcategoria de programas inteiros onde as variáveis são restritas a 0 ou 1, tal formulação é denominada **Programação Binária (PB)** e é descrita abaixo:

$$\begin{aligned} &\min c'x \\ &Ax = b \\ &x \in \{0,1\}^n \end{aligned}$$

2.2 Complexidade

Como anunciado no início deste capítulo, a programação inteira demanda um esforço computacional maior do que sua generalização, a programação linear. De fato, o teorema abaixo mostra que a programação inteira se encontra em um conjunto de problemas que acredita-se que não existam algoritmos polinomiais para sua resolução, a classe *NP*-difícil.

Teorema 2.1. *Programação Inteira é NP-difícil.*

Demonstração. Para mostrar que um problema P é *NP*-difícil é necessário provar que existe um algoritmo de tempo polinomial que a partir da resposta de P responda um outro problema P^*, onde P^* é *NP*-Completo. (WILLIAMSON; SHMOYS, 2010, p. 458)

Definindo um novo tipo de problema como Programação Inteira Completa (PIC) como:

$$PIC(v) = \begin{cases} Sim & \text{se } \min\{c'x \mid x \in P \cap \mathbb{Z}\} \leq v \\ Não & \text{caso contrário} \end{cases}$$

É evidente que um PI genérico $P : \min \{c'x \mid x \in P \cap \mathbb{Z}\}$ responde o problema PIC acima, portanto basta provar que PIC é *NP*-completo.

Para mostrar que um problema P^* pertence é *NP*-completo basta apresentar uma redução de tempo polinomial de um problema P', que seja *NP*-completo, para P^*.(WILLIAMSON; SHMOYS, 2010, p. 458)

Seja P' um problema de satisfatibilidade (SAT) na forma normal conjuntiva (CNF) formulado abaixo:

> Dados os conjuntos S de m cláusulas e $X = \{x_1, \ldots, x_n\}$ de variáveis binárias, tais que, $S = \{(x_{i_1} \vee \cdots \vee \bar{x}_{i_k}), \ldots, (x_{j_1} \vee \cdots \vee x_{j_l})\}$, onde $\bar{x}_j$ representa a negação da variável x_j, ou seja, $x_j + \bar{x}_j = 1$ para todo j e $x_i \vee x_j$ representa o *ou* binário. Responder Sim se existe uma valoração das variáveis binárias X tal que todas as cláusulas de S sejam sejam satisfeitas (tenham valor 1), responder Não caso contrário.

SAT é NP-completo. (COOK, 1971)

Seja f a transformação de um SAT P' em um PIC P^* dada por:

$$\begin{array}{lll} P': & \text{Dado um conjunto} & \\ & S = \{C_1, \ldots, C_m\} \text{ de cláusulas.} & \xrightarrow{f} \\ & \text{Verificar se existe alguma} & \\ & \text{valoração que satisfaça } S. & \end{array} \qquad \begin{array}{lll} P^*: & \min 0 & \\ & \sum_{i \in C_j^+} x_i + & \\ & \quad \sum_{i \in C_j^-} (1 - x_i) \geq 1 & \text{para } j = 1, \ldots, m \\ & x_i \in \{0, 1\} & \text{para } i = 1, \ldots, n \end{array}$$

onde a função f divide cada cláusula C_i em uma disjunção de duas cláusulas: C_i^- que contém os índices das variáveis negadas e C_i^+ que contém os índices das demais, ou seja, $C_i = C_i^- \vee C_i^+$.

Para que P^* tenha alguma solução viável, a restrição $\sum_{i \in C_j^+} x_i + \sum_{i \in C_j^-} (1 - x_i)$ tem que ser satisfeita para todas as cláusulas, logo, toda solução viável para P^* satisfaz todas as cláusulas de P', formando assim, uma valoração que satisfaz P'. Caso o problema P^* não tenha solução, então nao é possível satisfazer todas as restrições simultaneamente e portanto P' é insatisfazível.

Como a função de redução f tem tempo de execução polinomial na quantidade e tamanho das cláusulas de P', então, P^* é NP-completo, logo P é NP-difícil. □

2.3 Unimodularidade

Um importante conceito matemático é uma característica chamada **unimodularidade total** que algumas matrizes possuem. Tal conceito é especificamente importante para a programação inteira, pois os PIs que possuem a matriz de restrições nessa forma tem a característica de que, na sua versão linear (sem a restrição de integridade), todos os vértices são inteiros.

Definição 2.2. *Uma matriz B é dita **unimodular** se sua determinante tem valor absoluto 1, ou seja, $det(B) = \pm 1$. Uma matriz A é dita **totalmente unimodular** se todas as suas submatrizes não singulares são unimodulares, ou seja, $det(A') \in \{-1, 0, 1\}$, para toda submatriz A' de A.*

Teorema 2.2. *Se uma matriz $A \in \mathbb{R}^{mxn}$ é totalmente unimodular, então*

$$x \text{ é um vértice de } \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\} \text{ e } b \in \mathbb{Z}^m \implies x \in \mathbb{Z}^n$$

Demonstração. Se $x \in \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ um vértice para um vetor inteiro b, logo x possui uma matriz base B que é formada por m colunas independentes de A, onde as colunas correspondem aos índices das variáveis básicas de x (como descrito na Seção 1.6) e

$$x = B^{-1}b = \frac{B^{adj}b}{\det(B)}$$

Como B^{adj} é a matriz adjunta de B e B é unimodular, então $B^{adj} \in \mathbb{Z}^m$, logo, x é inteiro. □

Claramente o Teorema 2.2 também é válido para politopos que possuem restrições com inequações, pois, adicionando as variáveis de folga (Seção 1.3) resulta em uma matriz de restrições $A' = [A|I]$, que continua totalmente unimodular. Uma prova para esta característica pode ser encontrada em Papadimitrou e Steiglitz (1998, p. 317).

Como os vértices da versão real do problema são inteiros (Teorema 2.2), então, é garantido que o simplex retornará uma solução ótima para a versão linear que é inteira, já que, para resolver o PL, o simplex caminha entre os vértices, como esta na Seção 1.6.

Esta propriedade de unimodularidade total permite que alguns dos problemas que podem ser formulados como PIs possam ser resolvidos em tempo polinomial, já que, como enunciado no Teorema 1.1, um programa linear pode ser resolvido em tempo polinomial. Não coincidentemente, alguns dos bem conhecidos problemas de grafos que são resolvidos combinatóriamente por algoritmos polinomiais, ao serem transferidos para a forma de programas inteiros tem suas restrições em uma matriz totalmente unimodular. Exemplos destes problemas são o problema do caminho mínimo, fluxo máximo e emparelhamento ponderado máximo em grafos bipartidos. (PAPADIMITROU; STEIGLITZ, 1998, p. 318)

3 Técnicas Auxiliares para Programação Inteira

3.1 Limitantes e Relaxações

Uma importante característica de um problema de otimização é a capacidade de apresentar um certificado com o qual seja possível provar a otimalidade da solução produzida. No caso da programação linear, tal garantia, como discutido, é facilmente providenciada através da dualidade forte (Teorema 1.4).

Porém no contexto de programação inteira, não existe uma maneira simples para provar a otimalidade de uma solução. Para contornar este problema é usado o conceito de limitantes. Se existir um limitante inferior $\underline{z} \leq z^*$ e um limitante superior $\bar{z} \geq z^*$, onde z^* é o custo da solução ótima de um IP e se soubermos que $\bar{z} - \underline{z} \leq \varepsilon$ para algum $\varepsilon \geq 0$, então pode-se dizer que a solução ótima está limitada com acurácia de ε. Se ainda $\varepsilon = 0$, ou seja, $\underline{z} = \bar{z}$, então os limitantes são iguais ao custo da solução ótima.

No contexto de minimização, os limitantes são classificados em duas categorias: **primal**, caso seja um limitante superior e **dual**, caso seja um limitante inferior. Nota-se que qualquer ponto $x \in F$, onde F é o domínio do PI, representa através do custo $c'x$ associado, um limitante primal, pois a solução ótima possui o custo menor ou igual a todos os demais pontos de F.

A ideia de relaxação surge a partir da busca de limitantes para problemas de difícil resolução. Geralmente, no processo de relaxar um problema as características que o tornam mais difícil são contornadas, ou simplesmente eliminadas, resultando em um problema mais fácil porém mais abrangente, fornecendo assim um limitante dual para o problema inicial.

Para que um problema P' seja considerado a relaxação de um problema de minimização P duas características são necessárias: o domínio de P' deve conter o domínio de P, ou seja, $Dom(P) \subseteq Dom(P')$ e que a função de custo f' de P' sempre retorne um valor menor ou igual à função f de P para os pontos do domínio de P, ou seja, se $x \in Dom(P)$, então $f'(x) \leq f(x)$.

No contexto de Programação Inteira uma possível característica a ser relaxada é a discreticidade, ou seja, a necessidade das soluções serem inteiras. Assim é definida a Relaxação Linear:

Definição 3.1. *A **Relaxação Linear** de um programa inteiro $z = min\ \{c'x : x \in P \cap \mathbb{Z}\}$ é o programa linear $z^{LP} = max\ \{c'x : x \in P\}$.*

Como $\{x \in P \cap \mathbb{Z}\} \subseteq \{x \in P\}$ e a função de custo é a mesma para ambos, conclui-se que o processo acima é de fato uma relaxação.

Tal relaxação é especificamente útil quando a matriz de restrições que define o politopo P é totalmente unimodular, pois como foi mencionado na Seção 2.3, a relaxação linear terá uma solução ótima inteira e portanto resolverá o programa inteiro original.

3.2 Relaxação Lagrangeana

O conceito de relaxação lagrangeana é baseado na remoção de restrições específicas de programas lineares, ou particularmente programas inteiros. Além de remover a restrição indesejada, a relaxação lagrangena, procura beneficiar soluções que respeitem a restrição removida através de modificações na função de custo do problema.

Definição 3.2. *Dado um PI na forma*

$$\begin{gathered} min\ c'x \\ x \in P \\ Rx \leq b_R \\ x \geq 0 \end{gathered}$$

onde P é um poliédro qualquer, $R \in \mathbb{R}^{kxn}$ e $b_R \in \mathbb{R}^k$ representam k restrições. Ao aplicar a relaxação lagrangeana na restrição $Rx \leq b_R$, obtêm-se

$$\begin{gathered} min\ c'x - \mu(b_R - Rx) \\ x \in P \\ x \geq 0 \end{gathered}$$

onde $\mu \in \mathbb{R}^k_+$ representa o vetor de taxas arbitrárias.

É importante ressaltar que caso o problema se trate de uma maximização, o termo $\mu(b_R - Rx)$ da Definição 3.2 deve ser somado, e não subtraído, da função de custo.

Para mostrar que a relaxação lagrangeana é de fato uma relaxação, basta mostrar que:

- Todos os pontos do domínio original estão no novo domínio, o que é válido pois remover restrições não invalida pontos, ou seja

$$\{x \in \mathbb{R}^n |\, x \in P, Rx \leq b_R, x \geq 0\} \subseteq \{x \in \mathbb{R}^n |\, x \in P, x \geq 0\}$$

- A função de custo da relaxação sempre retorna um valor menor ou igual (no caso da minimização) ao original, quando aplicada aos mesmos elementos x, o que também é válido, pois se $x \in \{x \in \mathbb{R}^n | x \in P, Rx \leq b_r, x \geq 0\}$, então $b_R - Rx \geq 0$, e como $\mu \geq 0$, então

$$c'x - \mu(b_R - Rx) \leq c'x.$$

Uma característica que torna este tipo de relaxação particularmente atrativo é a de que pode existir um vetor de taxas $\mu^* \in \mathbb{R}^k$ que quando aplicado à relaxação resultara em uma solução ótima para o problema original, o que é mostrado no teorema a seguir.

Teorema 3.1. *Seja $\mu^* \geq 0$ e $x(\mu^*)$ a solução ótima de relaxação PI(μ^*). Se $Rx(\mu^*) \leq b_R$ e x é complementar a μ^*, ou seja, $\mu_i^* = 0$ ou $(Rx)_i = b_{Ri}$, então $x(\mu^*)$ é solução ótima para o PI original.*

Demonstração. Prova Como $x(\mu^*)$ é viável para o PI original, seja z^* o custo da solução ótima deste PI, logo $c'x(\mu^*) \geq z^*$. Como o PI(μ^*) é uma relaxação, então

$$\begin{aligned} z^* &\geq c'x(\mu^*) - \mu^*(b_R - Rx(\mu)) \\ &= c'x(\mu^*) - \textstyle\sum_{i=1}^{k} \mu_i^*(b_{Ri} - (Rx(\mu^*))_i), \quad \mu_i^* = 0 \text{ ou } (Rx)_i = b_{Ri} \\ &= c'x(\mu^*) \end{aligned}$$

logo $z^* = c'x(\mu^*)$.

□

É fácil notar que caso a restrição relaxada seja na forma $Ax = b$ o teorema acima também se aplica, pois se x respeita a restrição $Ax = b$, então $c'x = c'x - \mu(b - Ax)$ e, além disso, a taxa μ não é mais restrita a ser positiva.

Em diversas instâncias de problemas no contexto de programação inteira, os limitantes resultantes de métodos de relaxação lagrangeana são melhores do que os gerados a partir de relaxações lineares. (AHUJA; MAGNANTI; ORLIN, 1993, p. 601)

A principal aplicação desta técnica é em programas inteiros cuja matriz de restrição é quase totalmente unimodular, ou seja, se tratam de problemas que são facilmente resolvidos, com algumas restrições a mais que os tornam difíceis. A relaxação lagrangeana é então usada para remover as restrições que tornam a matriz de restrições não totalmente unimodular, para que a resolução da relaxação seja inteira. É importante ressaltar que isto não reduz a complexidade do programa inteiro original, já que ainda é necessário encontrar a taxa μ^* anunciada no Teorema 3.1, caso ela exista, para que a solução da relaxação respeite as restrições removidas.

3.3 Otimização por Subgradiente

Baseado no cálculo do gradiente de uma função, a otimização por subgradientes é um método iterativo que busca variar as taxas μ da relaxação lagrangeana através de passo de tamanho θ, até que a solução do problema respeite a restrição retirada na relaxação, ou seja, até que a solução ótima para o problema original seja encontrada.

Supondo um PI no formato padrão, seja $\mu^0 \in \mathbb{R}^k$ um valor inicial para a taxa, o método calcula os próximos valores através da fórmula

$$\mu^{p+1} = \mu^p + \theta_p(Ax^p - b)$$

onde $p \in \{0, 1, ...\}$ e x^p é a solução da relaxação lagrangeana com a taxa μ^p.

Caso as restrições relaxadas estejam no formato $Ax \leq b$, pelo Teorema 3.1, a taxa $\mu \in \mathbb{R}^k$ precisa ser positiva, logo, basta forçar a taxa a ser positiva durante a atualização, ou seja, $\mu_i^{p+1} = \max\{\mu_i^p + \theta_p(Ax^p - b)_i, 0\}$ para $i = 1, \ldots, k$, selecionando apenas a parte positiva do vetor de atualização.

Para garantir a convergência do método é necessário que o tamanho do passo θ respeite a propriedade:

$$k \to \infty \implies \theta_k \to 0 \text{ e } \sum_{i=1}^{k} \theta_k \to \infty$$

logo, a escolha direta é $\theta_k = 1/k$. (AHUJA; MAGNANTI; ORLIN, 1993)

Porém existem outras maneiras de calcular o tamanho do passo, como uma derivação do método de Newton para resolução de equações não lineares:

$$\theta_k = \frac{\lambda_k[UB - L(\mu^k)]}{\|Ax - b\|^2}$$

onde UB representa um limitante superior do custo do problema original e λ_k um escalar tal que $0 < \lambda_k \leq 2$. (AHUJA; MAGNANTI; ORLIN, 1993, p. 613)

Em um problema de minimização, conforme a evolução do método, o limitante UB tende a diminuir e portanto, θ_k também diminui. Nota-se também que caso $\|Ax - b\|^2 = 0$ a solução x^k da relaxação já é também solução para o problema original, logo θ_k não precisa ser calculado, pois o método foi concluído.

3.4 Branch and Bound

O branch and bound é uma técnica que segue a ideia de divisão e conquista e é amplamente usada na área de computação, ela consiste basicamente de duas etapas: branching, que é a divisão do problema em diversos subproblemas menores e bounding, que é a eliminação de alguns dos subproblemas menores decorrente de algumas de suas propriedades que o caracterizam como infrutífero.

Desta forma, um problema na forma

$$z = \min\{c'x \,|\, x \in S\}$$

pode ser dividido (*branching*) em diversos subproblemas. Sejam $S_1 \dots S_k$ conjuntos tais que $S_1 \cup \cdots \cup S_k = S$, definindo $z_i = \min\{c'x \mid x \in S_i\}$, podemos obter a resposta do problema original através de $z = \min\{z_i \mid i \in \{1, \dots, k\}\}$.

Porém, sem um mecanismo de poda (*bounding*), a solução de todos os subproblemas teria complexidade equivalente a solução do problema original. Para resolver este problema são usados limitantes para as soluções dos subproblemas, tais limitantes são usados para remover problemas cujas ramificações não apresentarão soluções melhores do que as já encontradas.

Por exemplo suponha que um problema de minimização, como o descrito acima, foi particionado em duas subdivisões S_1 e S_2 e cada um possui os limitantes inferiores $\underline{z}_1$ e $\underline{z}_2$ e superiores $\bar{z}_1$ e $\bar{z}_2$. Agora supondo que $\bar{z}_1 \leq \underline{z}_2$, como, pela definição de limitantes, $c'x_1 \leq \bar{z}_1$ e $\underline{z}_2 \leq c'x_2$, para todo $x_1 \in S_1$ e $x_2 \in S_2$, têm-se que $c'x_1 \leq c'x_2$, logo, é matemáticamente impossível que o segundo subproblema gere uma solução melhor do que o primeiro, portanto S_2 pode ser cortado, poupando o esforço computacional requerido para expandi-lo.

Um algoritmo genérico para executar branch and bound é representado na figura abaixo

Algorithm 1 Branch and Bound genérico para Programação Inteira de Minimização

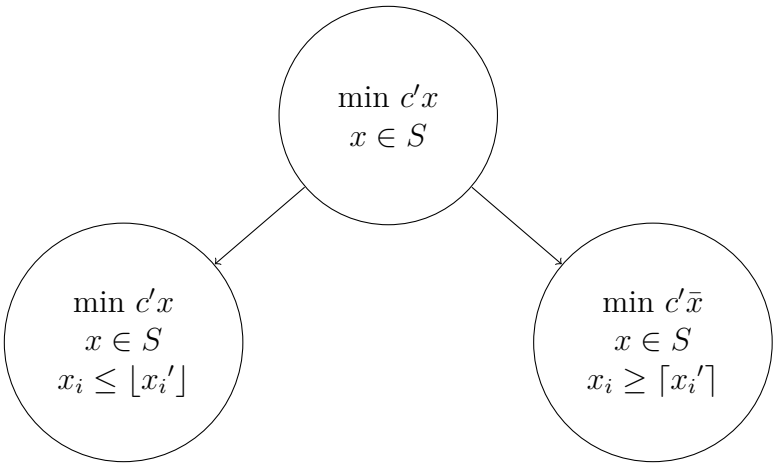
```
1:  procedure BBPI
2:      L ← Programa Inteiro Inicial PI
3:      x* ← null
4:      v* ← ∞
5:  laço:
6:      while L ≠ φ do
7:          Selecione e remove um problema P de L.
8:          x,v ← resolveRelaxação(P).
9:          if x = null or v ≥ v* then
10:             goto laço;
11:         if x ∈ Z^n then
12:             x* ← x.
13:             v* ← v;
14:         S ← geraSubproblemas(P).
15:         L ← L ∪ S;
```

onde a variável v^* representa um **limitante superior** que é usado para podar soluções infrutíferas e x^* representa a melhor solução encontrada durante a execução do algoritmo.

No escopo da Programação Inteira, o método Branch and Bound é especificamente útil pois permite realizar a divisão do problema em dois, através da escolha de uma variável que possuía, na solução relaxada do problema, um valor fracionário. Tal processo é feito através da inclusão de duas restrições diferentes, uma em cada subproblema, limitando a variável inferiormente pelo seu chão ou superiormente pelo seu teto.

Suponha um problema de minimização $P : \min\{c'x \mid x \in S \cap \mathbb{Z}\}$, seja $z' = \min\{c'x \mid x \in S\}$ a solução da relaxação linear de P. Escolhendo uma variável x_i, cujo valor na solução da relaxação é o valor fracionário x_i', pode-se criar dois diferentes subproblemas, um deles com uma nova restrição $x_i \leq \lfloor x_i' \rfloor$ e outro com $x_i \geq \lceil x_i' \rceil$. É fácil perceber que nenhuma possível solução para o problema original foi cortada nesta divisão, já que não existe um inteiro $l \in \mathbb{Z}$ tal que, $\lfloor x_i' \rfloor < l < \lceil x_i' \rceil$. O processo pode ser representado como na Figura 5.

Figura 5 – Divisão (*Branch*) em Programação Inteira



Fonte: o autor

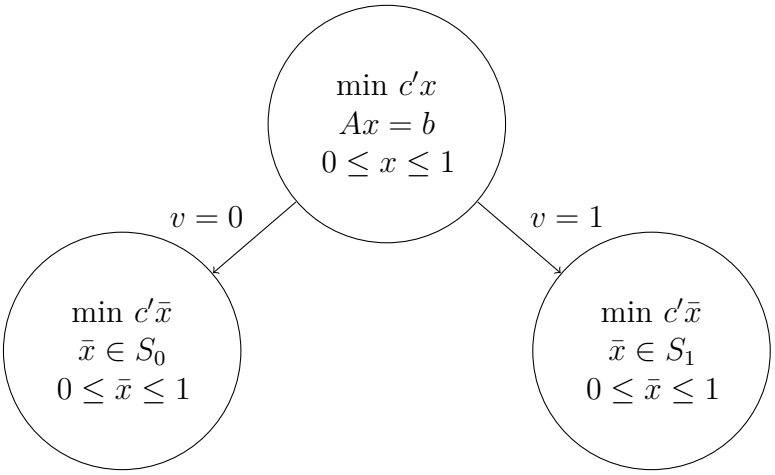
No caso de Programação Binária, o processo de divisão essencialmente elimina uma variável do problema. Executando um processo semelhante ao descrito na Figura 5, as restrições criadas serão $x_i = 0$ e $x_i = 1$, já que que a variável x_i é binária. Definindo uma nova variável $\bar{x} \in \mathbb{R}^{n-1}$ e um novo politopo $\bar{S}_v = \{\bar{x} \mid \bar{A}\bar{x} = b - A_i v\}$, onde $\bar{A}$ representa a matriz de restrições sem a i-ésima coluna, $v = 0$ para o primeiro subproblema e $v = 1$ para o segundo. Tal processo essencialmente diminui uma dimensão das relaxações geradas e pode ser visto na Figura 6.

Existem diversas heurísticas usadas para escolher qual variável será usada no processo de divisão, especialmente no caso de programação binária, onde escolher a variável representa usá-la ou não na solução, como por exemplo, usar ou não a aresta no menor caminho entre dois vértices.

3.5 Planos de Corte

Outra família de métodos usados para auxiliar a busca por soluções inteiras em um PI são os **planos de corte**, tais técnicas são comumente inseridas como um passo final

Figura 6 – Divisão (*Branch*) em Programação Binária



Fonte: o autor

em cada iteração do branch and bound para adicionar restrições que excluem partes do poliédro do PI que não contém soluções inteiras.

Nota-se que os métodos de planos de cortes não devem ser usados nos casos em que o PI em questão tenha a matriz de restrições totalmente unimodular, já que, ao adicionar uma restrição, a matriz pode perder esta propriedade e portanto, a relaxação linear deixaria de produzir uma solução inteira.

Os mais usados destes métodos são os **cortes de Gomory** cujo funcionamento é baseado na criação uma restrição, a partir da tábua final do simplex, que exclua, do politopo, a maior porção possível de soluções fracionárias que inclua a solução ótima da relaxação linear e não contenha soluções inteiras. Tal restrição é denominada **plano de corte**.

Uma importante característica dos cortes de Gomory, é que após a inserção da restrição criada, a solução ótima da relaxação linear antes do corte deixa de ser primal viável (como era o objetivo do corte) porém continua sendo dual viável, já que a inserção de uma nova restrição no problema primal corresponde a apenas uma nova variável no dual, o que não invalida a solução dual atual.

Com esta característica em mente, um algoritmo simples que use os cortes de Gomory pode também usar o método simplex dual para otimizar a solução dual, que possivelmente encontra-se próxima ao novo ótimo e pela dualidade forte (Teorema 1.4) possui o mesmo custo da solução ótima primal. Caso a inserção de uma nova restrição resulte em um problema dual ilimitado, como demonstrado na Seção 1.5, o problema primal é de fato inviável e portanto, não há nenhum ponto inteiro em seu poliédro. Tal algoritmo, assim como o método para criar as restrições, pode ser encontrados em Papadimitrou e

Steiglitz (1998, c. 14).

O algoritmo mencionado acima apresenta uma desvantagem de que caso ele precise ser suspenso antes do término, como é comum em problemas cujas resoluções não tem custo (em tempo) previsível e podem levar muito tempo, ele não possuirá uma solução inteira intermediária. Para contornar este problema, os planos de cortes são geralmente usados em conjunto com algoritmos de branch and bound, onde no final de cada iteração do mesmo um novo corte é criado e passado para os novos subproblemas. Tal abordagem é denominada **branch and cut**.

Existem outros tipos de cortes denominados **planos de cortes primais inteiros**, que mantém soluções subótimas intermediárias, permitindo assim, uma execução parcial dos algoritmos. (PAPADIMITROU; STEIGLITZ, 1998, p. 339)

4 Caminho Mínimo Restrito

4.1 Definições Básicas de Grafos

Informalmente, um **grafo** pode ser definido como um conjunto de elementos que possuem ou não relações dois a dois. Tal conceito pode ser aplicado em diversas situações, como por exemplo, tratando esquinas como os elementos do grafo, podemos dizer que dois elementos estão conectados se as esquinas são imediatamente adjacentes, ou seja, caso estejam na mesma rua e não exista nenhuma outra esquina entre ambos.

Formalmente um grafo é tratado como na Definição 4.1.

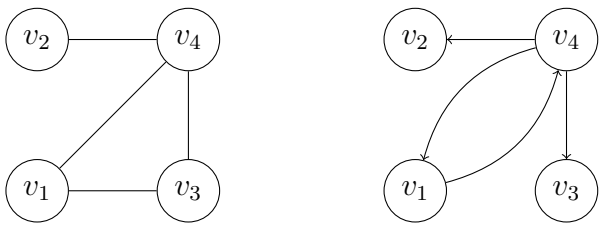
Definição 4.1. *Um grafo $G = (V, E)$ é formado por um conjunto de **vértices** V e um conjunto de **arestas** E, onde cada aresta é um conjunto de dois vértices, ou seja, $E \subseteq \binom{V}{2}$.*

Em alguns casos, é necessário que as relações entre os vértices não sejam necessariamente simétricas. Em tais casos a Definição 4.2 é usada.

Definição 4.2. *Um **grafo direcionado** ou **dígrafo** $G = (V, A)$ é formado por um conjunto de **vértices** V e um conjunto de **arcos** A, onde cada arco é um **par ordenado** de vértices, ou seja, $A \subseteq V \times V$.*

Usualmente, um grafo pode ser visualizado gráficamente representando os vértices como círculos, as arestas como linhas e, no caso em que o grafo é direcionado, arcos como setas. Um exemplo desta representação é mostrado na Figura 7.

Figura 7 – Exemplo de grafo e grafo direcionado, respectivamente



Fonte: o autor

Tendo em vista a praticidade, as definições subsequentes nesta seção serão referentes apenas a grafos direcionados, porém, todas elas podem ser facilmente estendidas para o contexto de grafos não-direcionados.

Existem outras maneiras de se representar um dígrafo, sendo as principais a **matriz de adjacência** e a **matriz de incidência**, que são explicitadas nas Definições 4.3 e 4.4, respectivamente.

Definição 4.3. *A matriz de adjacência MA de um dígrafo $G = (V, A)$ é definida da seguinte maneira:*

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \text{ , onde } a_{ij} = \begin{cases} 1 & \text{se } (v_i, v_j) \in A \\ 0 & \text{caso contrário} \end{cases}$$

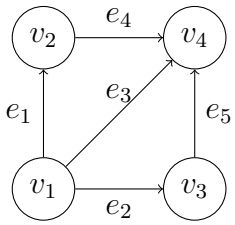
onde n é a quantidade de vértices, ou seja, $n = |V|$.

Definição 4.4. *A matriz de incidência MI de um dígrafo $G = (V, A)$ é definida da seguinte maneira:*

$$M = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \text{ , onde } a_{ij} = \begin{cases} 1 & \text{se } \exists\, v_k \in V \text{ tal que } e_j = (v_i, v_k) \\ -1 & \text{se } \exists\, v_k \in V \text{ tal que } e_j = (v_k, v_i) \\ 0 & \text{caso contrário} \end{cases}$$

onde $n = |V|$, $e_j \in A$ representa um arco e m é a quantidade de arcos, ou seja, $m = |A|$.

Figura 8 – Exemplo de uma matriz de adjacência e uma de incidência de um dígrafo



$$MA = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad MI = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & -1 \end{pmatrix}$$

Fonte: o autor

Todas as matrizes de incidência de dígrafos possuem a importante característica de serem totalmente unimodulares e, segundo o Teorema 2.2, todos os vértices dos poliédros definidos por elas são inteiros, o que, como já foi mencionado, possibilita uma resolução através de programação linear. Tal aspecto é provado no Teorema 4.1.

Teorema 4.1.

Demonstração. Suponha que a matriz MI não seja totalmente unimodular, logo, existem submatrizes quadradas de MI cujos determinantes não são -1, 0 ou 1. Seja B a menor (em

dimensão) destas submatrizes, logo, $det(B) \notin \{-1, 0, 1\}$. Seja c a coluna de B com a menor quantidade de valores diferentes de 0, podemos representar c em três casos possíveis: c tem todos os valores iguais a 0, c tem um valor diferente de 0 e c tem dois valores diferentes de 0. Nota-se que c não pode ter mais de dois valores diferentes de 0 pela definição da matriz de adjacência.

Caso 1 c tem todos os valores iguais a 0, logo, pela propriedade das determinantes, $det(B) = 0$, o que contradiz a suposição de que $det(B) \notin \{-1, 0, 1\}$.

Caso 2 c tem um valor $v \neq 0$, seja B' a sumbatriz de B sem a coluna c e a linha em que v se encontrava, como B' é uma dimensão menor do que B, temos que $det(B') \in \{-1, 0, 1\}$. Pelo teorema de Laplace, temos que $det(B) = \pm v det(B')$, logo $det(B) \in \{-1, 0, 1\}$, contradizendo a suposição.

Caso 3 c tem dois valores $v, v' \neq 0$, logo, todas as colunas de B tem exatamente dois valores diferentes de 0. Como, pela construção da matriz de adjacência, os dois valores são 1 e -1, temos que a soma de todas as linhas da matriz B resulta no vetor nulo, logo, as linhas de B são linearmente dependentes e, portanto, $det(B) = 0$, o que contradiz a suposição.

Logo, por absurdo, temos que MI é totalmente unimodular. □

Para denotar a quantidade de arcos entrando e saindo de um vértice a Definição 4.5 é usada.

Definição 4.5. *O grau de entrada de um vértice v, denotado por $\delta^+(v)$, é a quantidade de arcos cujo extremo final é v, ou seja, $\delta^+(v) = |\{(i, v) \,|\, (i, v) \in A\}|$. Analogamente o grau de saída, denotado por $\delta^-(v)$, é a quantidade de arcos cujo extremo inicial é v, ou seja, $\delta^+(v) = |\{(v, i) \,|\, (v, i) \in A\}|$.*

Nota-se que caso o grafo seja não-direcionado, o grau de entrada e o de saída são iguais e o seu valor é denotado por $d(v)$.

Diversos problemas da teoria de grafos envolvem o conceito de partir de um vértice e chegar em outro, através dos arcos (ou arestas) do grafo. A Definição 4.1 formaliza este conceito.

Definição 4.6. *Um **passeio** C em um dígrafo $G = (V, A)$ é uma sequência de vértices $(v_1, \ldots, v_n)$ tal que $(v_i, v_{i+1}) \in A$ para todo $1 \leq i \leq n - 1$. Caso $v_i \neq v_j$ para todo $1 \leq i, j \leq n$, então C é um **caminho** e diz-se, ainda, que C é um caminho de v_1 para v_n.*

Diversos problemas em grafos envolvem o conceito de montar um caminho cujo primeiro e último vértice coincidem, o que consiste de uma extrapolação da Definição , já que ela formaliza que todos os vértices precisam ser diferentes. O conceito é apresentado na Definição .

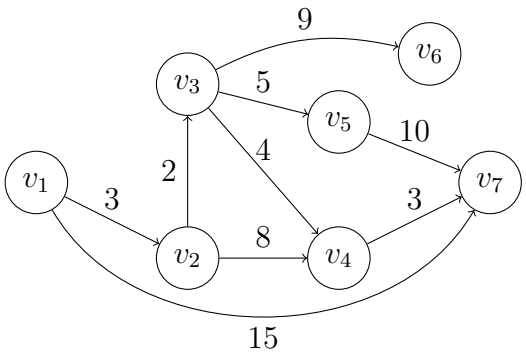
Definição 4.7. *Um* ***ciclo*** C *em um dígrafo* $G = (V, A)$ *é um caminho* $C' = (v_1, \ldots, v_n)$ *concatenado com o vértice* v_1*, caso o arco* $(v_n, v_1) \in A$*, ou seja,* $C = (v_1, \ldots, v_n, v_1)$*, com* $v_i \neq v_j$ *para todo* $1 \leq i, j \leq n$*.*

Muitas vezes é necessário atribuir valores, ou pesos, aos arcos (ou arestas) de um grafo, como mostrado na Definição .

Definição 4.8. *Um* ***dígrafo valorado*** *é um dígrafo* $G = (V, A)$ *associado a uma função de valoração* c*, onde a função de valoração pode estar relacionada aos vértices* $(c : V \to \mathbb{R})$ *ou aos arcos* $(c : A \to \mathbb{R})$*.*

Um dígrafo valorado é exemplificado na Figura 9, onde os valores das arcos são mostrados próximos a eles.

Figura 9 – Exemplo um grafo direcionado valorado nos arcos



Fonte: o autor

A Definição 4.8 pode ser extrapolada e aplicada a um passeio $P = (v_1, \ldots, v_k)$, ou seja, no caso da função referente aos vértices $c(P) = \sum_{i=1}^{k} c(v_i)$ e, quando referente aos arcos, $c(P) = \sum_{i=1}^{k-1} c(v_i, v_{i+1})$. Por exemplo, na Figura 4.8, o valor do caminho $C = (v_1, v_2, v_3, v_6)$ é $c(C) = 3 + 2 + 9 = 14$.

4.2 Caminho Mínimo

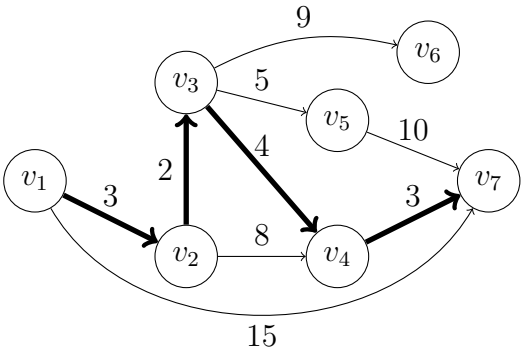
O problema de encontrar caminhos mínimos entre dois pontos sempre foi muito interessante para a humanidade, desde tempos pré-históricos, quando homens precisavam chegar até sua comida o mais rapidamente possível até o roteamento de pacotes na Internet. Inicialmente, o caminho mínimo era calculado somente para distâncias unitárias, porém, no decorrer de apenas uma década, métodos mais eficientes para distâncias mais genéricas foram desenvolvidos. (SCHRIJVER, 2010)

No contexto de grafos direcionados, o problema de encontrar o **caminho mínimo** é formalizado na Definição 4.9.

Definição 4.9. *Dado um dígrafo valorado $G = (V, A)$, uma função de valoração $c : A \to \mathbb{Z}$ e um par de vértices v_i e v_j. Encontrar um caminho mínimo de v_i para v_j, ou seja, encontrar um caminho $C = (v_i, \ldots, v_j)$ tal que $c(C)$ é mínimo.*

Um exemplo de caminho mínimo é mostrado na Figura 10, onde os arcos mais espessa representam o caminho mínimo.

Figura 10 – Exemplo de camínho mínimo de v_1 para v_7



Fonte: o autor

Em alguns casos é permitido que o caminho mínimo contenha um cíclo, tais problemas são ditos **caminho mínimo não-elementares**, em oposto ao problema da Definição 4.9 que é dito **elementar**, que é o escopo do presente trabalho.

Nota-se que na Definição 4.9, a função de valoração foi restrita aos inteiros, o que não consiste uma restrição muito forte, pois não é comum o uso de números irracionais no escopo de grafos e, caso os números sejam racionais, basta multiplicar todos os valores pelo mínimo divisor comum, resolver o problema e dividir o resultado pelo mesmo valor.

A maioria dos algoritmos para resolução do problema do caminho mínimo funcionam baseados em passeios e não caminhos, ou seja, não checam se há repetições de vértices (ou arcos). Como consequência disto, a maioria dos métodos exigem que não existam ciclos de valor negativo alcançáveis pelo vértice de partida, pois caso exista algum, não existirá um passeio mínimo, já que percorrendo o ciclo de valor negativo constrói-se um passeio de valor menor. Um dos algoritmos que funcionam desta maneira é o método de Bellman-Ford, que executa em tempo $O(|V|\,|A|)$.

Alguns dos algoritmos desenvolvidos para resolver o problema do caminho mínimo restringem a função de valoração c a ser positiva, ou seja, $c : A \to \mathbb{Z}^+$, o que é mais restritivo que a proibição de ciclos de valor negativo, pois sem arcos de valor negativo, tais

ciclos não existirão. O principal exemplo destes algoritmos é o método de Dijkstra, que quando implementado usando uma estrutura de dados chamada *heap*, que é usada para facilitar a manutenção da ordem de um conjunto de vértices, possui tempo de execução $O(|A| \, log \, |V|)$.

Uma consequência imediata da existência destes métodos de tempo de execução polinomial é o Teorema 4.2.

Teorema 4.2. *O problema do caminho mínimo pertence a classe P.*

4.3 Caminho Mínimo Restrito

Algumas vezes é interessante que cada arco do dígrafo tenha, além de um custo, um consumo de um certo recurso e uma restrição referente a este consumo que os novos caminhos viáveis devem comprir. Claramente esta ideia é mais propícia para modelar algumas das situações reais, em que custo e consumo não são diretamente proporcionais, como é o caso na maioria dos exemplos que envolvem o problema puro do caminho mínimo, como busca de rotas simples ótimas para veículos, em que o combustível gasto é, em geral, diretamente proporcional a distância percorrida.

Dentro deste ramo de caminhos mínimos restritos existem diversas subcategorias que variam em estrutura e complexidade e, apesar de aparentemente serem apenas uma variação deste problema, englobam diversos problemas clássicos da teoria dos grafos, como por exemplo o problema do caixeiro viajante.

A variação mais tradicional é denominada **caminho mínimo restrito por recursos (CMRR)**, em que cada aresta possui, além do custo tradicional, um consumo específico para cada recurso que está no escopo do problema. Além disso, os recursos possuem limites inferiores e superiores de consumo. Diversas pesquisas foram feitas neste escopos, algumas delas considerando apenas um recurso e apenas limites superiores, o que permite uma forte simplificação ao resolver o problema através de técnicas de programação inteira, como será visto ainda neste capítulo.

Outra categoria dentro das variações do problema de caminho mínimo é a **caminho mínimo restrito por vértices (CMRV)**, onde os caminhos viáveis passam, obrigatoriamente, por alguns vértices específicos, ou algumas vezes, por uma quantidade de vértices dentro de um subconjunto específico. Tal abordagem é um caso especial do CMRR, pois basta associar um recurso diferente para cada vértice, associar um consumo de 1 deste recurso para todos as arcos cujos destino é o vértice e impor um limite inferior de 1 unidade para cada um dos recursos. (BEASLEY; CHRISTOFIDES, 1989)

Ainda outra variação é o **caminho mínimo restrito no tempo (CMRT)**, onde o custo e os consumos dos arcos são dependentes do tempo em que são percorridos e

os vértices possuem uma janela de tempo em que podem ser acessados. Um exemplo de aplicação para o CMRT é a distribuição de produtos em lojas, onde o tráfego nas rodovias (e consequentemente o custo para percorrê-las) varia conforme o horário do dia e as lojas só abrem em um dado intervalo de tempo.

Uma característica interessante destes problemas é que apesar de aparentemente representarem apenas uma variação do caminho mínimo, estes consumos de recursos permitem a redução de diversos problemas clássicos da teoria dos grafos para o caminho mínimo restrito. Um exemplo direto de redução é referente ao problema do caixeiro viajante aberto para um CMRV forçando a passagem por todos os vértices. (BEASLEY; CHRISTOFIDES, 1989)

De fato, ao adicionar a restrição por recursos ao problema do caminho mínimo, o mesmo deixa de pertencer a classe P, como é mostrado no Teorema 4.3.

Teorema 4.3. *O problema do caminho mínimo restrito por recursos é NP-difícil.*

Demonstração. Basta encontrar uma redução de custo polinomial de um problema *NP*-difícil para o CMRR.

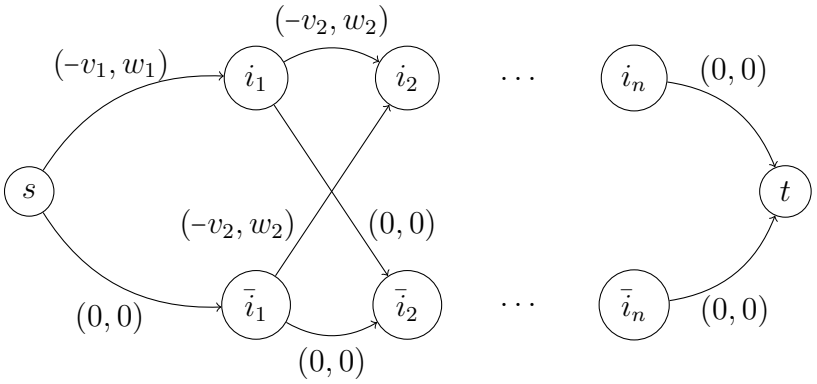
Seja P' o problema da **Mochila** formulado a seguir:

> Dado uma capacidade W e um conjunto de n itens $\{(v_1, w_1), \ldots, (v_n, w_n)\}$, onde v_i e w_i representam o valor e o peso, respectivamente, do i-ésimo item.
> Devolver o conjunto de itens cuja soma dos valores é máximo e a soma dos seus pesos não excedem a capacidade W.

O problema da Mochila é *NP*-difícil.

Basta montar um dígrafo acíclico como o da Figura 11 e achar um caminho mínimo restrito por recurso entre s e t, onde o recurso representa a W capacidade da mochila.

Figura 11 – Problema da Mochila em um grafo



Fonte: o autor

Seja um caminho $C = (s, v_1, v_2, \ldots, v_n, t)$ no dígrafo da Figura 11. Claramente, $v_j \in \{i_j, \bar{i}_j\}$, onde o vértice i_j e $\bar{i}_j$ representam a escolha de colocar o item j ou não, respectivamente, na mochila. Assim, limitando o consumo do recurso w_i por W, a solução resultante é viável para a mochila e mínima no inverso dos valores v_i dos itens, logo, é uma solução ótima para o problema da mochila.

Como montar o dígrafo a partir da instância do problema da mochila e transformar o caminho mínimo em uma resposta para o problema da mochila claramente são de custo polinomiais no tamanho da instância, a redução é válida e, portanto, o problema do caminho mínimo restrito é *NP*-difícil. □

4.4 Resolvendo problemas de caminhos mínimos restritos

Por ser um problema de dificil resolução (*NP*-difícil), o problema do caminho mínimo restrito, foi explorado em diversos trabalhos e possui diversas abordagens de resolução diferentes. Esta seção tem como propósito delinear algumas destas técnicas e ponderar sobre suas respectivas eficiências e peculiaridades.

Um algoritmo baseado no problema de caminho com múltiplas restrições (CMR), que é a *versão NP-completa* do CMRR (semelhante ao processo apresentado no Teorema 2.1), é apresentado em Jaffe (1984). O CMR consiste em responder se existe um caminho, entre dois vértices dados, com o custo no máximo L e o consumo de um recurso no máximo W, tal problema pode ser usado para resolver o CMRR através de uma busca binária no limitante L.

Para resolver este problema foi adotada uma abordagem baseada em programação dinâmica, através da manutenção de tabelas (uma tabela por vértice) que contém o caminho, de menor custo para um dado consumo máximo, até todos os demais vértices. Tal tabela é atualizada através da troca de mensagens entre os vértice, ao detectar um caminho de custo menor do que o armazenado em sua tabela, o vértice envia uma mensagem contendo o custo, o consumo e o destino do caminho a todos os outros vértices.

Ao receber uma mensagem, um vértice verifica se o valor do caminho é menor do que o armazenado em sua tabela e, caso seja menor, atualiza a entrada com do destino e consumo recebido na tabela, além de buscar entradas com consumos máximos maiores e, caso o valor recebido seja menor, atualiza-as também. Após esta etapa o vértice, caso necessário, envia mensagens para os demais, e assim sucessivamente.

Uma importante característica deste método é que ele permite a minimização (ou maximização) multiobjetivo através de uma combinação linear entre valor e consumo, o que é particularmente útil no âmbito de redes de computadores, onde busca-se um balanço entre entregas de mensagens e custo monetário de manutenção do canal. (JAFFE, 1984)

A resolução de Jaffe possui tempo de execução pseudo-polinomial e é, inerentemente, propício para paralelização através da troca de mensagens. Como consequência disto, possui a necessidade de sincronizações entre os vértices e, pelo fato de ser baseado em programação dinâmica, necessita também de uma memória potencialmente grande para armazenar as tabelas de cada vértice.

4.5 Formulação em Programação Inteira

Como provado no Teorema 2.1, a programação inteira é *NP*-difícil, logo, como todo problema da classe P pode ser reduzido aos problemas *NP*-completos, existe uma redução de custo polinomial que transforma um problema de caminho mínimo em um PI. Tal processo é descrito na Definição 4.10 que foi adaptada de Beasley e Christofides (1989).

Definição 4.10. *Dado um dígrafo $G = (V, A)$ e uma função de valoração $c : A \to \mathbb{Z}$, o problema do caminho mínimo entre os vértices v_1 e v_n, possui um programa inteiro correspondente no formato abaixo:*

$$
\begin{aligned}
&min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
&s.a. \sum_{i \in V} x_{ij} = \sum_{i \in V} x_{ji} && \forall j \in V && (4.1)\\
&\sum_{j \in V} x_{1j} = 1 && && (4.2)\\
&\sum_{i \in V} x_{in} = 1 && && (4.3)\\
&\sum_{i \in V} x_{ij} \leq 1 && \forall j \in V && (4.4)\\
&\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 && \forall S \subseteq V && (4.5)\\
&x_{ij} \in \{0, 1\}
\end{aligned}
$$

onde a variável x_{ij} é definida como:

$$
x_{ij} = \begin{cases} 1 & \text{se o arco } (i,j) \in A \text{ está no caminho ótimo} \\ 0 & \text{caso contrário} \end{cases}
$$

As restrições (4.1) do PI na Definição 4.10 garantem que o grau de entrada e de saída de cada vértice é o mesmo e, caso a reposta não contenha ciclos, o grau varia entre 0 e 1. As restrições (4.3) e (4.4) garantem que o caminho comece no vértice v_1 e termine no vértice v_n, respectivamente. Já a inequação (4.2) garante que não existam ciclos com vértices em comum com o caminho mínimo entre v_1 e v_2.

Claramente as restrições (4.5) da Definição 4.10 são um problema, já que a quantidade de subconjuntos de um conjunto é exponencial em seu tamanho. Porém, como a

função destas inequações é a de impedir ciclos desconexos do caminho mínimo entre v_1 e v_n ela pode ser desconciderada em casos que não existam ciclos cujo valor é negativo no dígrafo de entrada, o que não representa um grande problema, pois, a existência deste ciclo implicaria de certo modo na inexistência de um caminho mínimo.

A abordagem de eliminar as restrições (4.5) ainda permite que a reposta do PI contenha ciclos desconexos de valor 0, porém, um processamento, de custo linear na quantidade de arestas, pode ser aplicado à resposta original para eliminar estes ciclos, o que não impactará no custo e na otimalidade da solução (no caso em que não existam ciclos de valor negativo). Um raciocinio analogo pode ser aplicado para eliminar a restrição (4.4).

Com estas simplificações e a exigência de não existência de ciclos negativos chega-se na Definição 4.11.

Definição 4.11. *Dado um dígrafo $G = (V, A)$* ***sem ciclos de valor negativo****, cuja matriz de incidência é MI e uma função de valoração $c : A \to \mathbb{Z}$, o problema do caminho mínimo entre os vértices v_1 e v_n, possui um programa inteiro correspondente no formato abaixo:*

$$\begin{aligned} min \ & \sum_{i=1}^{|A|} c_i x_i \\ s.a. \ & MIx = b \\ & x_i \in \{0, 1\} \end{aligned} \tag{4.6}$$

onde a variável x_i é definida como:

$$x_i = \begin{cases} 1 & \text{se o arco } e_i \in A \text{ está no caminho ótimo} \\ 0 & \text{caso contrário} \end{cases}$$

e o vetor de termos independentes $b \in \mathbb{Z}^n$ como:

$$b = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix}$$

Como provado no Teorema 4.2, o problema do caminho mínimo pode ser resolvido em tempo polinomial, logo, transformá-lo em um PI, que segundo o Teorema 2.1 é *NP*-completo, não seria prudente. Porém, como a matriz de restrições do problema da Definição 4.11 é a matriz de incidência de um dígrafo que, segundo o Teorema 4.1, é totalmente unimodular, então o PI definido pode ser resolvido diretamnete através de sua relaxação linear em tempo polinomial.

Para gerar o programa inteiro corresponde ao problema do caminho mínimo restrito por recursos basta adicionar as restrições referentes aos recursos na formulação da Definição 4.10. O resultado é mostrado na Definição 4.12, adaptada de Beasley e Christofides (1989).

Definição 4.12. *Dado um digrafo $G = (V, A)$, uma função de valoração $c : A \to \mathbb{Z}$, uma função de consumo de recursos nos arcos $r : A \to \mathbb{Z}_+^K$, uma função de consumo de recursos nos vértices $q : V \to \mathbb{Z}_+^K$ e os limitantes inferiores e superiores $L \in \mathbb{Z}_+^K$ e $U \in \mathbb{Z}_+^K$, respectivamente, onde $K \in \mathbb{N}$ representa a quantidade de recursos diferentes. O problema do caminho mínimo restrito por recursos entre os vértices v_1 e v_n no dígrafo mencionado pode ser formulado como:*

$$
\begin{aligned}
&\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
&\text{s.a.} -\sum_{(i,j) \in A} r(i,j)\, x_{ij} - \sum_{i \in V} q(i) \Big(\sum_{m \in V} x_{mi} + \sum_{j \in V} x_{ij} \Big) \Big/ 2 \leq -L && (4.7)\\
&\quad \sum_{(i,j) \in A} r(i,j)\, x_{ij} + \sum_{i \in V} q(i) \Big(\sum_{m \in V} x_{mi} + \sum_{j \in V} x_{ij} \Big) \Big/ 2 \leq U && (4.8)\\
&\quad \sum_{i \in V} x_{ij} = \sum_{i \in V} x_{ji} \qquad \forall j \in V && (4.9)\\
&\quad \sum_{j \in V} x_{1j} = 1 && (4.10)\\
&\quad \sum_{i \in V} x_{in} = 1 && (4.11)\\
&\quad \sum_{i \in V} x_{ij} \leq 1 \qquad \forall j \in V && (4.12)\\
&\quad \sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq V && (4.13)\\
&\quad x_{ij} \in \{0, 1\}
\end{aligned}
$$

As restrições adicionais (4.7) e (4.8) são referentes as limitações inferiores e superiores, respectivamente, dos K recursos. Como foi mencionado anteriormente, as restrições de consumo $q(i)$ de recurso em vértices pode ser reduzidas para consumos em arcos, basta adicionar o consumo $q(i)/2$ a todos os arcos que chegam e saem do vértice i. Logo as segundas somatórias das restrições adicionais podem ser desconsideradas.

Outra importante observação referente à Definição 4.12 é que o fato do consumo de recursos ser limitado inferiormente tem um impacto na importância de ciclos (conectados ao caminho ou não) na solução ótima, já que a remoção de um ciclo da solução pode inviabilizá-la. Como consequência disto, caso algum consumo seja limitado inferiormente, as restrições nao podem ser representadas (como na Definição 4.11) apenas pela matriz de incidência do dígrafo.

Com isso em mente, o problema que será tratado no restante deste trabalho será o caminho mínimo restrito superiormente por recursos (**CMRSR**), que é abordado mais

concisamente na Definição 4.13.

Definição 4.13. *Dado um dígrafo $G = (V, A)$* ***sem ciclos de valor negativo****, cuja matriz de incidência é MI e uma função de valoração dos arcos representada através do vetor $c \in \mathbb{Z}^n$, uma função de consumo de recursos nos arcos, representada pela matriz $R \in \mathbb{Z}_+^{K \times n}$ e um limitante superior para o consumo de cada recurso representado através do vetor $U \in \mathbb{Z}^K$, onde $n = |V|$ e K é o número de recursos diferentes.*

O problema do caminho mínimo restrito superiormente por recursos entre os vértices 1 e n, possui um programa inteiro equivalente, no formato abaixo:

$$
\begin{aligned}
& min\ c^T x \\
& s.a.\ Rx \leq U && (4.14)\\
& \quad MIx = b && (4.15)\\
& \quad x_i \in \{0, 1\}
\end{aligned}
$$

onde x_i e b são estruturados como na Definição 4.11.

Nota-se que, analogamente ao programa inteiro do problema do caminho mínimo, a solução ótima do PI da Definição 4.13 pode conter ciclos, porém os mesmos podem ser removidos, em tempo linear, sem impactar na viabilidade e otimalidade da solução. Tal característica é feita possível pelo fato de existirem apenas restrições superiores ao consumo de recursos. É importante também destacar que as restrições (4.14) torna o PI não totalmente unimodular.

Como o CMRSR é *NP*-completo (Teorema 4.3), o fato do PI associado a ele não ser totalmente unimodular é esperado, já que uma solução polinomial deste PI (através da relaxação linear) implicaria em uma solução polinomial para o CMRSR e, consequêntemente, em $P = NP$.

Referências

AHUJA, R. K.; MAGNANTI, T. L.; ORLIN, J. B. *Network Flows*. 1. ed. [S.l.]: Prentice-Hall, Inc., 1993. ISBN 0-13-617549-X. Citado 2 vezes nas páginas 18 e 19.

BEASLEY, J. E.; CHRISTOFIDES, N. An algorithm for the resource constrained shortest path problem. *Networks*, Wiley Subscription Services, Inc., A Wiley Company, v. 19, n. 4, p. 379–394, 1989. ISSN 1097-0037. Disponível em: <http://dx.doi.org/10.1002/net.3230190402>. Citado 4 vezes nas páginas 29, 30, 32 e 34.

COOK, S. A. The complexity of theorem-proving procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 1971. (STOC '71), p. 151–158. Disponível em: <http://doi.acm.org/10.1145/800157.805047>. Citado na página 13.

DANTZIG, G. B. *Linear Programming and Extensions*. 1. ed. [S.l.]: Princeton University Press, 1963. Citado na página 10.

JAFFE, J. M. Algorithms for finding paths with multiple constraints. *Networks*, Wiley Subscription Services, Inc., A Wiley Company, v. 14, n. 1, p. 95–116, 1984. ISSN 1097-0037. Disponível em: <http://dx.doi.org/10.1002/net.3230140109>. Citado na página 31.

KRUMKE, S. O. Integer programming: Polyhedra and algorithms. 2006. Citado na página 12.

PAPADIMITRIOU, C. M. *Computational complexity*. Reading, Massachusetts: Addison-Wesley, 1994. ISBN 0201530821. Citado na página 4.

PAPADIMITROU, C. H.; STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. 1. ed. [S.l.]: Englewood Cliffs, N.J., 1998. Citado 10 vezes nas páginas 4, 5, 6, 7, 9, 10, 11, 14, 15 e 23.

SCHRIJVER, A. *Documenta Math. 155 On the History of the Shortest Path Problem*. 2010. Citado na página 27.

WILLIAMSON, D. P.; SHMOYS, D. B. *The Design of Approximation Algorithms*. 1. ed. [S.l.]: Cambridge University Press., 2010. Citado na página 13.