UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

DANIEL BRUNO LOPES DA SILVA

ESTUDO DO PROBLEMA DO CONJUNTO FECHADO DE PESO MÁXIMO:
aspectos matemáticos, algoritmos e aplicações

Recife
2019

DANIEL BRUNO LOPES DA SILVA

ESTUDO DO PROBLEMA DO CONJUNTO FECHADO DE PESO MÁXIMO: aspectos matemáticos, algoritmos e aplicações

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Produção da Universidade Federal de Pernambuco como parte dos requisitos parciais para obtenção do título de mestre em Engenharia de Produção.

Área de concentração: Pesquisa Operacional.

Orientador: Prof^{o}. PhD. Sóstenes Luiz Soares Lins.

Recife
2019

Catalogação na fonte
Bibliotecária Margareth Malta, CRB-4 / 1198

S586e Silva, Daniel Bruno Lopes da.
Estudo do problema do conjunto fechado de peso máximo: aspectos matemáticos, algoritmos e aplicações / Daniel Bruno Lopes da Silva. - 2019.
82 folhas, il., gráfs., tabs.

Orientador: Prof. Dr. Sóstenes Luiz Soares Lins.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia de Produção, 2019.
Inclui Referências.

1. Engenharia de Produção. 2. Problema do fechado máximo. 3. Corte mínimo. 4. Pseudofluxo. 5. Problema da mineração à céu aberto. I. Lins, Sóstenes Luiz Soares. (Orientador). II. Título.

UFPE

658.5 CDD (22. ed.) BCTG/2019-117

DANIEL BRUNO LOPES DA SILVA

ESTUDO DO PROBLEMA DO CONJUNTO FECHADO DE PESO MÁXIMO:
aspectos matemáticos, algoritmos e aplicações

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Produção da Universidade Federal de Pernambuco como parte dos requisitos parciais para obtenção do título de mestre em Engenharia de Produção.

Aprovada em: 25/02/2019.

BANCA EXAMINADORA

Prof^{o}. PhD. Sóstenes Luiz Soares Lins (Orientador)

Prof^{a}. Dra. Isis Didier Lins (Examinadora Interna)

Prof^{o}. PhD. Nivan Roberto Ferreira Júnior (Examinador Externo)

Aos meus pais, Adelmo e Márcia, aos meus irmãos Priscila, Paulo Wagner e José Henrique, e à minha amiga, companheira e namorada, Rafaela.

AGRADECIMENTOS

Meus sinceros agradecimentos aos membros da banca examinadora, Sóstenes Lins, Nivan Ferreira e Isis Lins, pela atenção e pelo esforço dedicados à avaliação deste trabalho. Pelas críticas, observações e sugestões. Ao PPGEP UFPE pela oportunidade e pelo ambiente dinâmico e fértil de conhecimento. Ao meu orientador, Sóstenes Lins, pelo seu valioso acompanhamento, pelas sugestões, pelas críticas, pelas discussões, mas especialmente pelo exemplo. Ao professor Nivan Ferreira do CIn, pela paciência, solicitude e apoio nas diversas dúvidas que tive com os algoritmos. Aos meus pais Adelmo e Márcia e meus irmãos Priscila, Paulo Wagner e José Henrique por TUDO. À minha namorada, Rafaela, por ser a luz da minha vida. Aos meus amigos do PPGEP Diogo, Francisco (Jr), Dryca, Darío e Augusto pela companhia nos desafios dessa aventura. Aos meus amigos de outras jornadas: Izadora, Daone, Felipe Guilherme e Álvaro. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Por isso, também agradeço imensamente à CAPES.

RESUMO

Em diversas situações, faz-se necessário selecionar um subconjunto de itens de uma ampla coleção, mantendo-se a relação que existe entre alguns dos itens, de tal forma que se um item for selecionado, todos os seus sucessores também devem ser selecionados. A cada item é associado um peso e o objetivo é selecionar o conjunto que maximiza a soma dos pesos dos itens selecionados. Este problema é conhecido como problema do conjunto fechado de peso máximo e apresenta diversas aplicações em áreas como gestão de produção e operações, manutenção, defesa e gerenciamento de projetos. O problema do conjunto fechado de peso máximo se assemelha ao problema da mochila, um conhecido problema NP-completo. No entanto, o problema do conjunto fechado de peso máximo pode ser resolvido através de algoritmos polinomiais. Este trabalho é um estudo de revisão bibliográfica acerca do problema do conjunto fechado de peso máximo com foco em três aspectos: desenvolvimento matemático, algoritmos e aplicações. Em termos matemáticos, os resultados importantes são a unimodularidade total da matriz de restrições, a reducibilidade para o problema do corte mínimo e a relação com outros problemas combinatórios. Analisamos as principais características de três algoritmos para o problema do conjunto fechado de peso máximo, indicando o mais eficiente. Além disso, discutimos brevemente algumas das aplicações do problema, sobretudo em áreas de interesse da engenharia de produção. Acreditamos que este trabalho preenche uma lacuna na literatura, aprofundando o entendimento sobre o problema do conjunto fechado de peso máximo e detalhando suas características mais importantes. Com uma formulação abrangente e flexível, o problema em questão pode ser utilizado em diversas áreas da engenharia de produção. Entendemos que o conteúdo desta pesquisa é uma ferramenta necessária para uma maior utilização do problema e para a formulação de novas aplicações.

Palavras-chave: Problema do fechado máximo. Corte mínimo. Pseudofluxo. Problema da mineração à céu aberto.

ABSTRACT

In a variety of situations, one must select a subset of items from a vast collection, preserving the relationship that may exist among some of these items, in such a way that if an item is selected all of its successors must be chosen as well. To each item is associated a weight and the goal is to select the subset that maximizes the weights' sum of the items selected. This problem is known as the maximum weight closure problem and it has several applications in such areas as production and operations management, maintenance, defense, and project management. The maximum weight closure problem is similar, in a sense, to the knapsack problem, a known NP-complete problem. However, the maximum weight closure problem can be reduced to a minimum cut problem in a special graph and then it can be solved by polynomial algorithms.This work is a bibliographical review study about the maximum weight closure problem focusing in three aspects: mathematical development, algorithms, and applications. In mathematical terms, the important results are the total unimodularity of the constraints matrix, the reducibility to the minimum cut problem, and the relationship with other combinatorial problems. We analyzed the main features of three algorithms for the maximum weight closure problem pointing out the most efficient one. Furthermore, we briefly discuss some applications of the problem, specially in industrial engineering related fields. We believe that this work fills a gap in the literature, deepening the understanding about the maximum weight closure problem and detailing its most important features. With a broad and flexible formulation, this problem can be applied to various industrial engineering domains. It is our understanding that the content of this research constitutes a necessary tool for a broader utilization of the maximum weight closure problem and for formulation of new applications.

Keywords: Maximum closure problem. Minimum cut. Pseudoflow. Open-pit mining problem.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Existem diversas situações nas quais se deseja selecionar um subconjunto de elementos, onde a cada elemento é associado um determinado valor (positivo, nulo ou negativo), e onde existe dependência física ou lógica entre alguns desses elementos. Não há restrições no número de elementos a serem selecionados, mas uma vez que um elemento é selecionado, todos os elementos dependentes devem ser selecionados também. Exemplos dessas situações podem ser encontrados na seleção de terminais de operações em aeropotos ou empresas de frete, seleção de ferramentas para composição de kits de reparo e em problemas de *schedulling*.

Em otimização combinatória, esse problema é chamado de problema do *closure* de peso máximo (*closure* máximo ou *closure* ótimo), onde *closure* significa, nesse contexto, um conjunto fechado de vértices ponderados em um grafo direcionado. Especificamente, nos referimos a um conjunto S de vértices como sendo fechado se a cofronteira dirigida para fora de S é vazia, isto é, não existe arco saindo do conjunto. Dessa forma, nos referimos a esse problema como o problema do conjunto fechado de peso máximo, ou, por simplicidade, problema do fechado máximo ou do fechado ótimo. Não obstante, alguns problemas combinatórios importantes como o problema da seleção (BALINSKI, 1970), o problema dos custos fixos compartilhados (RHYS, 1970) e o problema da provisão (LAWLER, 1976) são todos redutíveis ao problema do fechado máximo. Além disso, o problema denominado *maximum blocking-cut* (RADZIK, 1993) é uma generalização do problema do fechado máximo, de forma que a resolução do último implica na resolução do primeiro.

Em geral, problemas que buscam por uma estrutura específica em um grafo, como um circuito Hamiltoniano, tendem a se concentrar na classe de problemas considerados intratáveis. O problema do fechado máximo é semelhante ao problema da mochila, um conhecido problema NP-completo. Surpreendentemente, no entanto, o problema do fechado máximo pode ser solucionado por algoritmos polinomiais. Isto porque esse problema pode ser reduzido ao problema do corte mínimo, que, por sua vez, é o dual do problema do fluxo máximo.

Problemas de fluxo máximo, em geral, podem ser solucionados por algoritmos rápidos e atrativos. O teorema de Ford e Fulkerson (1956) para o problema do fluxo máximo fornece, além de um algoritmo pseudopolinomial, uma condição de otimalidade de fácil verificação. Com essas características, justificadamente, os trabalhos de Ford e Fulkerson formaram a base para um paradigma de criação de algoritmos para o problema do fluxo máximo. Esses algoritmos são caracterizados por manter um fluxo viável e se aproximar da otimalidade em cada iteração.

Na mesma época, Boldyreff (1955b) apresentou uma heurística para o problema do fluxo máximo cujos requisitos básicos eram: (i) a solução pudesse ser obtida rapidamente, mesmo para uma rede complexa; (ii) o método pudesse ser explicado facilmente para pessoal sem conhecimento técnico especializado; (iii) a validade da solução estivesse sujeita a uma verificação rápida e direta; (iv) o método não dependesse do uso de super computadores ou

outros equipamentos especializados. O método de Boldyreff é a base de outro paradigma para o desenvolvimento de algoritmos para o fluxo máximo. Esses algoritmos atacam o problema pelo lado oposto em relação ao método de Ford e Fulkerson, ou seja, partem de uma condição de superotimalidade e buscam a viabilidade de fluxo.

Os métodos de Ford e Fulkerson e de Boldyreff, juntamente com os trabalhos de Dantzig em programação linear e inteira, firmaram uma base conceitual sólida para resolução eficiente do problema do fluxo máximo. O problema do fechado ótimo, surgido da necessidade de resolução de algumas aplicações práticas a partir da década de 1970, poderia ser resolvido em tempo polinomial como um produto subjacente do problema do fluxo máximo.

Algumas aplicações, especialmente o problema da mineração a céu aberto (*OPMP - Open-pit mining problem*), entretanto, exigiram maior eficiência computacional e também uma maior especialização teórica do algoritmo do fechado máximo. Devido às suas elevadas "dimensões" - o grafo utilizado para modelar o problema pode ultrapassar facilmente 400,000 vértices e 0.7 bilhão de arestas (ver Hochbaum e Chen (2000)) - o OPMP impulsionou o desenvolvimento de diversos algoritmos e heurísticas.

De fato, o problema da mineração a céu aberto levou ao desenvolvimento de um novo algoritmo, o pseudofluxo. Hochbaum (2008) desenvolveu este algoritmo com foco no problema do fechado de peso máximo e baseado em um algoritmo anterior, desenvolvido por Lerchs e Grossmann (1965). O pseudofluxo trouxe uma série de inovações à medida que uniu conhecimentos provenientes da engenharia de minas e da otimização combinatória.

O algoritmo é processado numa estrutura chamada árvore normalizada e é baseado em pseudofluxos, uma generalização do conceito de prefluxo de Goldberg e Tarjan (1988). Um prefluxo viola a restrição de conservação do fluxo nos vértices, mas apenas em uma direção, permitindo que os vértices apresentem excesso de fluxo. Um pseudofluxo viola a restrição da conservação em ambas as direções, de forma que os vértices podem apresentar excesso ou déficit.

Originalmente, o algoritmo não se utilizava do conceito de fluxo, mas sim do conceito de massa. Atualmente, no entanto, o algoritmo resolve também o problema do fluxo máximo e têm se mostrado mais rápido do que o algoritmo *push-relabel* de Goldberg e Tarjan (1988) (CHANDRAN; HOCHBAUM, 2009; FISHBAIN; HOCHBAUM; MUELLER, 2010) que é o *benchmark* até o momento. Além disso, as notações e os conceitos utilizados nesses dois algoritmos são semelhantes, a complexidade é basicamente a mesma e uma variante do algoritmo pseudofluxo também se utiliza de um esquema de rotulação, a exemplo do *push-relabel*.

O problema do fechado máximo tem uma descrição simples, poucos requisitos e ampla flexibilidade. Combinadas essas características com sua resolução polinomial, o problema se torna uma ferramenta importante para resolução de diversos problemas reais, principalmente no contexto industrial. Dessa forma, faz-se necessário um estudo aprofundado do problema do fechado de peso máximo, considerando suas características matemáticas e algorítmicas. Este trabalho busca preencher essa lacuna.

1.1 Justificativa

Os problemas de seleção de terminais de serviço que motivaram a formulação do problema do fechado máximo se concentram na área de planejamento das operações. Com efeito, diversas aplicações do problema do fechado máximo têm sido desenvolvidas no contexto da gestão da produção e das operações.

Além dos problemas de seleção, problemas de *scheduling* também podem ser resolvidos através do problema do fechado máximo. Esses problemas são de especial interesse tanto pela ampla aplicabilidade como pela complexidade. Constantemente, problemas práticos de *scheduling* são resolvidos através de heurísticas ou algoritmos de aproximação. A resolução de alguns tipos de problemas de *scheduling* através de um método polinomial pode revelar formas de desenvolver algoritmos mais eficientes.

O problema do fechado máximo também têm sido usado na resolução de problemas de manutenção, de gerenciamento de projetos, de defesa e mineração, de forma que em cada uma dessas aplicações, o problema do fechado máximo toma uma forma distinta. Apesar de, na essência, todas essas aplicações se utilizarem da modelagem discutida neste trabalho, as especificidades de cada forma do problema podem mostrar estratégias mais adequadas e mais eficientes. O trabalho de Hochbaum e Chen (2000) ilustra essa situação.

Nesse trabalho, as autoras investigaram a fundo o problema da mineração à céu aberto, uma das aplicações do problema do fechado máximo, respondendo à solicitação da indústria de mineração, que enfrentava dificuldades computacionais para resolver o problema mesmo sabendo-se que ele era redutível ao problema do corte mínimo. Elas analisaram características do problema, comparam os dois principais algoritmos disponíveis e propuseram um algoritmo mais adequado, robusto e eficiente para o problema da mineração.

Dessa forma, uma investigação ampla do problema do conjunto fechado de peso máximo, considerando suas características matemáticas estruturais, suas aplicações e os algoritmos disponíveis para sua resolução, faz-se necessária, tendo-se em vista que tal estudo poderá fornecer os subsídios conceituais para um melhor entendimento do tema, bem como para o desenvolvimento de novas aplicações.

1.2 Objetivo Geral

Esta pesquisa busca realizar um estudo do problema do conjunto fechado de peso máximo, focando no seu desenvolvimento, na sua modelagem, nos algoritmos disponíveis para sua resolução e nas aplicações do problema.

1.3 Objetivos Específicos

- Analisar a formulação e a modelagem do problema do conjunto fechado de peso máximo.

- Analisar a relação entre outros problemas combinatórios e o problema do conjunto fechado de peso máximo.
- Avaliar os algoritmos disponíveis para resolução do problema do conjunto fechado de peso máximo.
- Estudar aplicações do problema do conjunto fechado de peso máximo.

1.4 Estrutura do Trabalho

O trabalho está estruturado em seis capítulos. O próximo capítulo, apresenta definições e conceitos iniciais importantes para o desenvolvimento do restante do trabalho. Nesse capítulo, são apresentados elementos da teoria dos grafos, da teoria de algoritmos e complexidade e da otimização combinatória.

O capítulo 3 apresenta o problema do conjunto fechado de peso máximo. Uma breve contextualização é apresentada, seguida pela discussão acerca do desenvolvimento, da formulação matemática e de algumas aplicações do fechado máximo.

O capítulo 4 foca no problema do fluxo máximo e nos principais algoritmos desenvolvidos para resolvê-lo. Os estudos de Ford e Fulkerson recebem uma atenção especial em virtude da formulação do teorema *Max-Flow Min-Cut*.

O capítulo 5 foca no desenvolvimento do algoritmo pseudofluxo. Apesar de resolver também o problema do fluxo máximo, o algoritmo pseudofluxo merece um capítulo à parte porque apresenta algumas implicações teóricas e práticas importantes para o problema do conjunto fechado de peso máximo.

Por fim, o capítulo 6 apresenta as considerações finais, onde são discutidos os aspectos matemáticos, comparados os algoritmos e ressaltadas algumas das características importantes das aplicações do fechado máximo.

2 DEFINIÇÕES E CONCEITOS PRELIMINARES

Neste capítulo são apresentados as definições básicas da teoria dos grafos, bem como os conceitos fundamentais da teoria da complexidade de algoritmos. O capítulo é finalizado por uma breve apresentação da otimização combinatória.

2.1 Teoria dos Grafos

O problema das sete pontes de Königsberg (atualmente Kaliningrado) é o marco inicial da teoria dos grafos. O território que nos anos 1700 era chamado de Königsberg, na Prússia, é banhado pelo rio Prególia, sobre o qual havia sete pontes. Questionava-se, na época, se seria possível percorrer cada uma das sete pontes uma única vez e retornar ao ponto de partida (HARRIS; HIRST; MOSSINGHOFF, 2008).

Em 1736, Euler provou que tal rota era impossível e definiu condições precisas para que se pudesse encontrar uma rota como a desejada num sistema arbitrário de pontes interconectadas. Euler representou essa situação através de um sistema de pontos conectados por linhas formando um diagrama simples, um grafo (FOULDS, 1992).

A partir das definições oferecidas por Euler, a teoria dos grafos começou a ganhar destaque como uma ferramenta eficaz para modelagem e resolução de diversos problemas nas mais diversas áreas do conhecimento como geografia, química e física a partir dos anos 1800. Mais tarde, a teoria dos grafos começou a auxiliar a otimização de sistemas produtivos (HARRIS; HIRST; MOSSINGHOFF, 2008).

A teoria dos grafos foi usada e desenvolvida simultaneamente em diversas áreas do conhecimento como geografia, física, química e genética, e por esse motivo existem diferentes conjuntos de notações. Neste trabalho, utiliza-se a notação apresentada por Bondy e Murty (2008) para as definições de grafos e suas características.

Um grafo G é um par ordenado (V, E) que consiste de um conjunto de vértices (ou nós) $V(G)$ e um conjunto (disjunto de $V(G)$) de arestas $E(G)$, juntamente com uma função incidência ψ_G que associa a cada aresta de G um par **não-ordenado** de vértices, não necessariamente distintos, de G.

A cardinalidade do conjunto dos vértices é normalmente representada por n e a cardinalidade do conjunto das arestas é normalmente representada por m, portanto:

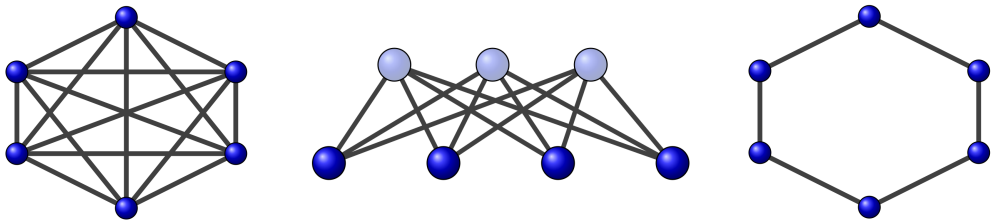
$$n = |V(G)| \qquad \text{e} \qquad m = |E(G)|$$

Determinados tipos de grafos desempenham funções importantes na teoria dos grafos e nas suas aplicações, por isso são classificados em famílias especiais. Essas classificações, em geral, são um reflexo da forma do grafo e, por isso, fáceis de compreender. Algumas das classificações mais importantes são brevemente apresentadas a seguir.

Um grafo que possui apenas um vértice é dito *trivial*, todos os demais são ditos *não-triviais*. Um grafo é *simples* quando não apresenta loops nem arestas paralelas. Muito do estudo de grafos se concentra em grafos simples. Um grafo *vazio* é um grafo no qual não existem vértices adjacentes, ou seja, seu conjunto de arestas é vazio. Um grafo *completo* é um grafo simples no qual quaisquer dois vértices são adjacentes, sendo geralmente representado por K_n. O grafo ilustrado na figura (1a) é um grafo completo em 6 vértices.

Figura 1 – Exemplos de Grafos Especiais

(a) Grafo Completo K_6 (b) Grafo Bipartido Completo (c) Polígono



Fonte: O autor (2018)

Um grafo é *bipartido* se o seu conjunto de vértices pode ser dividido em dois subconjuntos X e Y de tal forma que cada aresta tenha uma extremidade em X e a outra em Y. Essa divisão (X, Y) é chamada uma *bipartição* do grafo. Em geral, denota-se um grafo bipartido G com bipartição (X, Y) por $G[X, Y]$. Se $G[X, Y]$ é simples e se existem arestas ligando cada vértice em X a cada vértice em Y, então G é um *grafo bipartido completo*. O grafo da figura (1b) é um grafo bipartido completo.

Um *caminho*, P, é um grafo simples cujos vértices podem ser organizados em uma sequência linear de tal maneira que dois vértices são adjacentes se eles forem consecutivos na sequência, e não-adjacentes caso contrário. De maneira semelhante, um *polígono*, C, com três vértices ou mais, é um grafo simples cujos vértices podem ser organizados em uma sequência cíclica de forma que dois vértices são adjacentes se eles forem consecutivos na sequência, e não-adjacentes caso contrário. Um polígono com um vértice consite de um vértice com um loop e um polígono com dois vértices consiste em dois vértices ligados por um par de arestas paralelas. O grafo da figura (1c) é o polígono em seis vértices, C_6; removendo-se uma aresta de C_6 na figura (1c) obtem-se um caminho.

Sejam X e Y dois conjuntos não vazios de vértices de G. O conjunto das arestas de G com uma extremidade em X e a outra em Y é representado por $E[X, Y]$. Quando $Y = G \setminus X$, o conjunto $E[X, Y]$ é chamado *corte* de G associado a X ou *cofronteira* de X e é denotado por $\partial(X)$.

Em algumas aplicações, como em problemas de fluxo, o conceito de grafo incorpora um determinado sentido em suas arestas como parte de suas propriedades para poder modelar o problema adequadamente. Nesses casos, define-se o conceito de grafo direcionado (digrafo) e algumas características específicas.

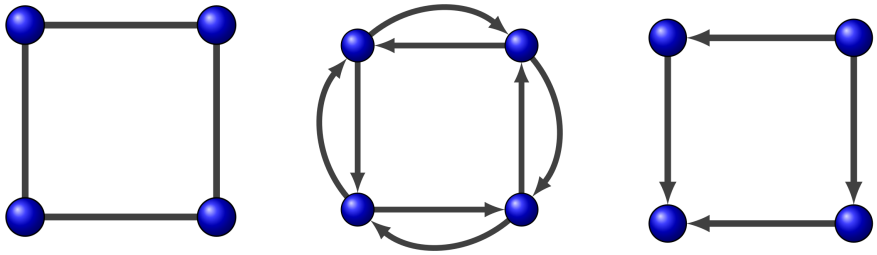
2.1.1 Grafos Direcionados

Um grafo direcionado D é um par ordenado $(V(D), A(D))$ composto por um conjunto de vértices $V := V(D)$ e um conjunto $A := A(D)$, disjunto de V, de arcos, juntamente com uma função incidência ψ_D que associa a cada arco de D um par **ordenado** de vértices (não necessariamente distintos) de D. Se a é um arco de D e $\psi_D(a) = (u, v)$, u é a *cauda* ou *origem* e v, a *cabeça* ou *extremidade* do arco a (diz-se, também, que u domina v).

Grafos e digrafos mantêm uma relação direta. A cada digrafo D, é possível associar um grafo G com o mesmo conjunto de vértices substituindo-se cada arco do digrafo por uma aresta conectando os mesmos vértices. O grafo obtido é chamado *grafo subjacente* de D e é denotado por $G(D)$. Reciprocamente, qualquer grafo pode dar origem a um digrafo substituindo-se cada aresta por dois arcos com orientações opostas entre os mesmos vértices. O digrafo obtido é chamado *digrafo associado* de G, denotado por $D(G)$. É possível, também, obter um digrafo a partir de um grafo substituindo-se cada aresta por apenas um arco em uma das duas direções entre os mesmos vértices; nesse caso, o digrafo obtido é chamado uma *orientação* de G. A figura (2) ilustra esses conceitos.

Figura 2 – Grafos e Digrafos

(a) Grafo Subjacente (b) Digrafo Associado (c) Orientação de G



Fontes: O autor (2018)

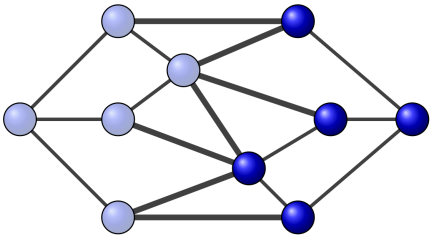
Em um digrafo, pode-se definir dois vértices especiais, a *fonte* e o *sorvedouro*. Um vértice s é uma fonte se não existe arco chegando em s, e um vértice t é um sorvedouro se não existe arco saindo de t (ver figura (3b)).

Para um digrafo $D = (V, A)$, um subconjunto B de A é um corte ou cofronteira se $B = \partial^+(U)$ para algum subconjunto $U \subseteq V$, ou seja, B é o conjunto de arcos de A saindo de vértices em U para vértices em $V \setminus U$. Em particular, $\emptyset$ é um corte e se $\emptyset \neq U \neq V$, então $\partial^+(U)$ é um corte não trivial. Para dois subconjuntos $X, Y \subset V$, (X, Y) é uma outra forma de representar o corte $\partial^+(X)$.

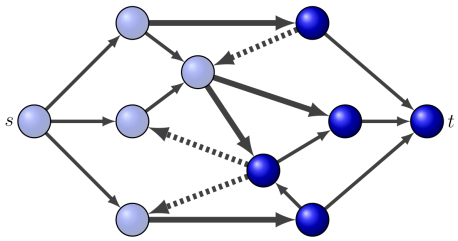
Geralmente, quando se fala de um corte, refere-se a um conjunto X tal que $s \in X$, denominando-se esse conjunto X de *conjunto fonte*. Adicionalmente, um s-t corte refere-se a uma partição em dois subconjuntos de vértices, tal que um subconjunto contém a fonte s e o outro, o sorvedouro, t, tomando-se o corte em relação ao conjunto fonte.

Figura 3 – Corte em grafos e digrafos

(a) Corte em um grafo



(b) Corte em um Digrafo



Fonte: O autor (2018)

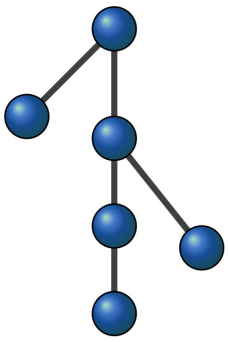
No grafo da figura (3a), o corte $\partial(X)$ é representado pelas arestas em destaque (mais grossas que as demais). No digrafo da figura (3b), o corte $\partial^+(X)$, é representado pelos arcos sólidos mais grossos que os demais, enquanto que $\partial^-(X) = \partial^+(\overline{X})$, isto é, o corte relativo ao conjunto complementar de X, é representado pelos arcos tracejados. Ainda na figura (3b), a fonte e o sorvedouro do digrafo são destacados.

2.1.2 Árvores

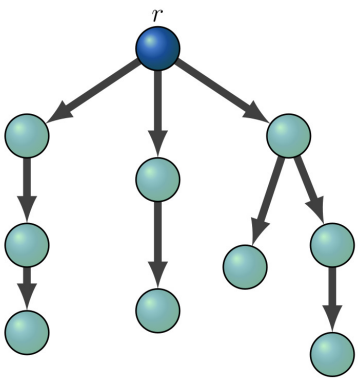
Uma *árvore* é um grafo acíclico e conexo como o grafo da figura (4a). Conforme essa definição, observa-se que cada componente de um grafo acíclico é uma árvore e, por isso, um grafo acíclico é geralmente chamado de uma *floresta*. Um grafo é conexo se existe, no mínimo, um caminho entre quaisquer dois de seus vértices (BONDY; MURTY, 2008).

Figura 4 – Árvores e Arborescências

(a) Árvore T



(b) Arborescência T_r



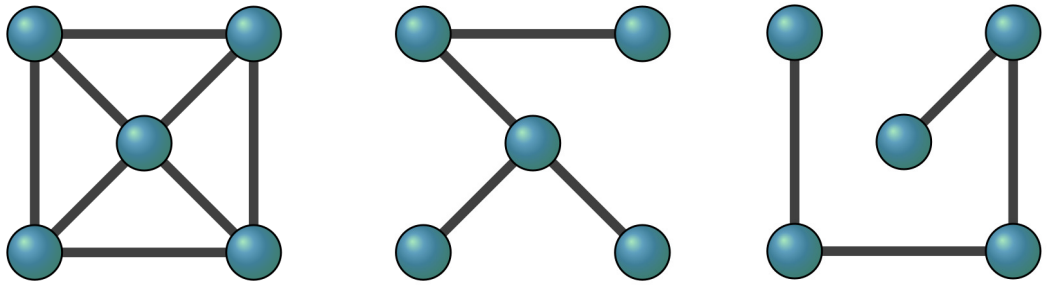
Fonte: O autor (2018)

Em algumas aplicações, faz-se necessário distinguir um vértice especial r de uma árvore T; esse vértice é chamado *raiz* de T. Se a árvore T possui tal vértice, ela é representada por $T(r)$ ou T_r. Uma orientação de uma árvore T_r na qual existe apenas um arco chegando em cada vértice diferente de r, é chamada uma *arborescência*. A figura (4b) mostra uma arborescência.

Considere uma árvore T_r com raiz r. Um vértice u em T_r é um ancestral de um vértice v se o caminho de v a r contém u. Todos os vértices que tem o vértice u como ancestral são descendentes de u. T_v representa um galho de T_r suspenso a partir de v e contendo v e todos os seus descendentes em T_r. Um vértice u é o pai de um vértice v, denotado por $p(v)$, se u é o único vértice imediatamente seguinte a v no caminho de v a r. Portanto, $T_{[v,p(v)]} = T_v$ é a subárvore suspensa a partir da aresta $[v, p(v)]$. Todos os descendentes imediatos de v são seus filhos, denotados por $ch(v)$.

Uma *árvore geradora* é um subgrafo acíclico e conexo obtido por eliminação de arestas. A figura (5) mostra a decomposição do grafo *wheel* W_4 em duas árvores geradoras.

Figura 5 – Duas árvores geradoras do grafo W_4



Fonte: Adaptado de Bondy e Murty (2008)

2.1.3 Representação de Grafos

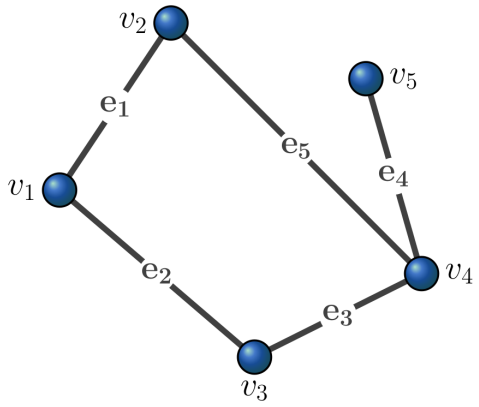
O diagrama de um grafo, arestas ou arcos conectando vértices, é uma representação simples que permite uma visualização concisa da topologia do grafo. Contudo, essa forma de representação não é adequada para o processamento computacional do grafo. Existem diversas estruturas de dados para representação e processamento computacional de grafos, entre as quais a matriz de incidência e a matriz de adjacência, mostradas na tabela (1) a seguir.

Considere o grafo G da figura (6), com conjunto de vértices V e conjunto de arestas E. A matriz de incidência de G é a matriz $\mathbf{M}_G := (m_{ve})$, $n \times m$, onde m_{ve} é o número de vezes em que o vértice v e a aresta e são incidentes, isto é, 0, 1 ou 2 (no caso de *loop*). A tabela (1a) mostra a matriz de incidência do grafo G.

A matriz de adjacência de G, é a matriz $\mathbf{A}_G := (a_{uv})$, $n \times n$, onde a_{uv} é o número de arestas conectando os vértices u e v, considerando cada *loop* contando duas vezes. A tabela (1b) mostra a matriz de adjacência do grafo G.

Naturalmente, digrafos também podem ser representados através da matriz de incidência e da matriz de adjacência. A matriz de incidência representa um digrafo através de uma matriz $n \times m$ que contém uma linha para cada vértice e uma coluna para cada arco. A coluna correspondente a um arco (u, v) apresenta apenas duas entradas diferentes de zero: $+1$ na linha correspondente ao vértice u e -1 na linha correspondente ao vértice v. A matriz de adjacência de um digrafo é uma matriz $n \times n$ com uma linha e uma coluna para cada vértice, de tal forma que seu elemento

Figura 6 – Grafo G



Fonte: O autor (2018)

Tabela 1 – Estruturas de Dados para um Grafo

(a) Matriz de Incidência $\mathbf{M}_G$

	e_1	e_2	e_3	e_4	e_5
v_1	1	1	0	0	0
v_2	1	0	0	0	1
v_3	0	1	1	0	0
v_4	0	0	1	1	1
v_5	0	0	0	1	0

(b) Matriz de Adjacência $\mathbf{A}_G$

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	1	0	0	1	1
v_3	1	0	0	1	0
v_4	0	1	1	0	1
v_5	0	0	0	1	0

Fonte: O autor (2018)

indexado pela linha u e pela coluna v é igual a 1 se o arco (u, v) existe, e é igual a 0 caso contrário (AHUJA; MAGNANTI; ORLIN, 1993).

Considere o digrafo D com conjunto de vértices V e conjunto de arcos A, conforme mostrado na figura (7). As tabelas (2a) e (2b) a seguir apresentam, respectivamente, a matriz de incidência, $\mathbf{M}_D$, e a matriz de adjacência, $\mathbf{A}_D$, relativas ao digrafo D.

Tabela 2 – Estruturas de Dados para um Digrafo

(a) Matriz de Incidência $\mathbf{M}_D$

	a_1	a_2	a_3	a_4	a_5
v_1	1	1	0	0	0
v_2	-1	0	0	0	-1
v_3	0	-1	1	0	0
v_4	0	0	-1	1	1
v_5	0	0	0	-1	0

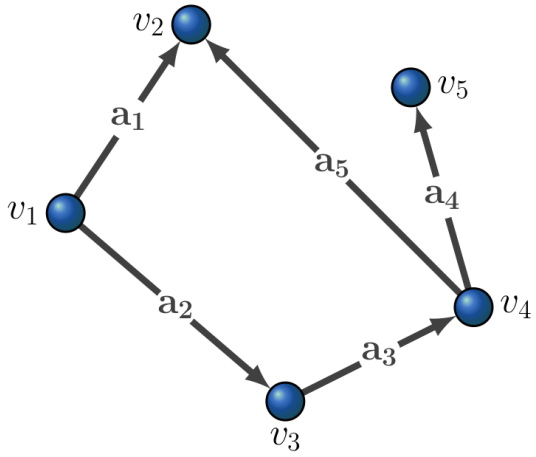
(b) Matriz de Adjacência $\mathbf{A}_D$

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	0	0	0	0	0
v_3	0	0	0	1	0
v_4	0	1	0	0	1
v_5	0	0	0	0	0

Fonte: O autor (2018)

Tanto a matriz de incidência como a matriz de adjacência representam completamente o grafo. Como os grafos geralmente apresentam mais arestas do que vértices, a matriz de adjacência

Figura 7 – Digrafo D



Fonte: O autor (2018)

é geralmente preferida pois frequentemente ocupa menos espaço na memória do computador, uma vez que é indexada nas colunas pelos vértices enquanto que a matriz de incidência é indexada pelas arestas.

2.2 Algoritmos e Complexidade Computacional

O termo algoritmo foi cunhado em homenagem ao árabe Al Khwarizmi, escritor do livro apontado como o mais importante meio de transmissão das ideias estruturais do sistema decimal, isto é, adição, multiplicação, divisão. Os procedimentos apresentados no livro de Al Khwarizmi eram precisos, mecânicos, eficientes e corretos (DASGUPTA; PAPADIMITRIOU; VAZIRANI, 2008).

Daí segue uma noção intuitiva do que seja um algoritmo: um conjunto finito de instruções que executam operações em um conjunto de dados. A observação fundamental aqui é que o conjunto de instruções que constitui o algoritmo é finito e fixo, mas o tamanho do conjunto de dados pode variar e dependerá dos dados de entrada do algoritmo (SCHRIJVER, 2003).

O exemplo a seguir ilustra a ideia primitiva de complexidade de um algoritmo. Considere a famosa sequência de Fibonacci definida da seguinte forma:

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{se } n > 1 \\ 1 & \text{se } n = 1 \\ 0 & \text{se } n = 0 \end{cases}$$

Considere o algoritmo FibExp na figura (8) para calcular o enésimo número de Fibonacci. Este algoritmo é correto pois literalmente implementa a definição da sequência de Fibonacci, ou seja, o termo de índice zero é 0, o termo de índice 1 é 1 e, a partir de então, o termo de índice n é a soma dos termos de índice $n-1$ e $n-2$, de forma que, recursivamente, o algoritmo calcula os

termos de índice $n-1$ e $n-2$ e os soma para fornecer o termo de índice n dado como parâmetro de entrada.

Figura 8 – Algoritmo FibExp

Procedure FibExp(n)

input : n
output : F_n

```
1 if n = 0 then
2 |   return 0
3 end
4 if n = 1 then
5 |   return 1
6 end
7 return FibExp (n − 1) + FibExp (n − 2)
```

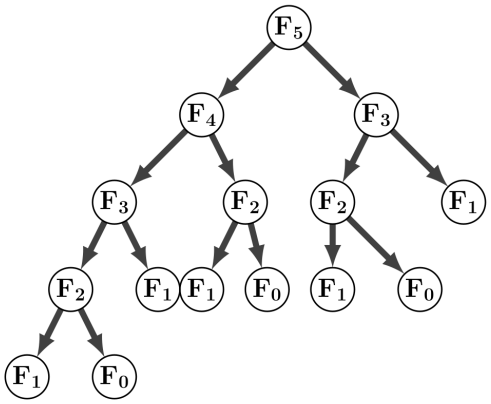
Fonte: Adaptado de Dasgupta, Papadimitriou e Vazirani (2008)

Em termos de desempenho, no entanto, o algoritmo não é desejável conforme n aumenta. Considere, por exemplo, a função $T(n)$ como sendo o número de operações computacionais básicas necessárias para calcular FibExp(n). Para $n = 2$, o algoritmo fará duas invocações recursivas de FibExp(), isto é, FibExp(1) e FibExp(0), checará o valor de n em relação aos dois **if**'s (linha 1 e linha 4), e realizará uma soma (linha 7), ou seja

$$T(n) = T(n-1) + T(n-2) + 3$$

O número de operações básicas cresce mais rapidamente do que os números de Fibonacci. A figura (9) a seguir ilustra a execução do algoritmo FibExp() para $n = 5$.

Figura 9 – Execução do algoritmo FibExp(5)



Fonte: Adaptado de Dasgupta, Papadimitriou e Vazirani (2008)

Esta figura evidencia que o algoritmo é ineficiente para valores moderados de n, realiza cálculos repetidos e desnecessários. Mais que isso, esse algoritmo necessitaria um número

Figura 10 – Algoritmo FibPol

Procedure FibPol(n)

```
   input  : n
   output : F_n
1  if n = 0 then
2  |  return 0
3  end
4  crie uma array f[0 ... n]
5  f[0] = 0, f[1] = 1
6  for i = 2 ... n do
7  |  f[i] = f[i − 1] + f[i − 2]
8  end
9  return f[n]
```

Fonte: Adaptado de Dasgupta, Papadimitriou e Vazirani (2008)

demasiado elevado de operações computacionais para calcular F_{50} ou F_{100}, por exemplo; seria impraticável.

O algoritmo a seguir, no entanto, realiza a mesma tarefa, ou seja, calcula o enésimo número de Fibonacci, mas de forma mais eficiente que o anterior. Ele guarda os cálculos intermediários em uma lista evitando, assim, cálculos repetidos. O loop interno consiste de apenas um passo computacional e é executado apenas $n - 1$ vezes. Portanto, o número de operações básicas usadas nesse algoritmo é linear em n.

Enquanto o primeiro algoritmo, FibExp() varia exponencialmente com n, o segundo algoritmo, FibPol(), varia de forma polinomial com n. Essa é uma distinção fundamental em complexidade computacional e tem sido usada como forma de avaliar a eficiência de algoritmos. De forma geral, considera-se que um algoritmo é uma solução prática útil para um problema computacional somente se sua complexidade cresce polinomialmente com o tamanho do input (PAPADIMITRIOU; STEIGLITZ, 1998; SCHRIJVER, 2003).

Neste exemplo, o design é fundamental para garantir a eficiêcia do algoritmo. Existem problemas, no entanto, para os quais ainda não foram descobertos algoritmos eficientes e esses problemas constituem um dos principais objetos de estudo da complexidade computacional. Mais precisamente, a complexidade computacional é a área da ciência da computação que investiga as razões pelas quais alguns problemas são tão difíceis de serem resolvidos por algoritmos computacionais (PAPADIMITRIOU, 1994).

Analisar um algoritmo significa pensar sobre como os recursos exigidos pelo agoritmo se comportarão com o incremento do tamanho de sua entrada. Encontrar critérios, em termos gerais, que consigam classificar algoritmos quanto à sua eficiência é tarefa difícil. A própria definição de eficiência de algoritmos computacionais está sujeita a diversas variáveis como a plataforma utilizada e a instância do problema em foco. Dessa forma, torna-se desejável uma definição concreta de eficiência algorítmica que seja independente da plataforma e da instância e que possa indicar como o algoritmo se comportará conforme o input aumenta (KLEINBERG;

TARDOS, 2005).

Tipicamente, a eficiência computacional de um algoritmo é mensurada como o número de operações básicas executadas pelo algoritmo como uma função do tamanho de seu input. Mais especificamente, a eficiência computacional de um algoritmo pode ser capturada por uma função $T(n)$ de $\mathbb{N}$ em $\mathbb{N}$ tal que $T(n)$ seja o número máximo de operações básicas que o algoritmo executa para inputs de tamanho n. No entanto, a função $T(n)$ pode se tornar dependente de detalhes inconvenientes decorrentes da definição de operações básicas (ARORA; BARAK, 2009).

Por isso é que se faz uso da notação conhecida como "*big-oh*" como uma forma de estimar a eficiência de um algoritmo. Essa notação carrega a mesma ideia da função $T(n)$ mencionada acima. Com essa forma de analisar os algoritmos, fatores constantes são ignorados e a atenção é voltada para o comportamento assintótico da função $T(n)$.

2.2.1 Notação O

Um determinado algoritmo com função, por exemplo, $T(n) = 5n - 3$, sendo n o tamanho do seu input, apresenta tempo de processamento $O(n)$ que pode ser lido como "*big-oh de* n" ou "*oh de* n" e informalmente significa "*alguma constante vezes* n". Essa noção de uma certa constante vezes n proporciona duas consequências desejáveis: (1) ignora constantes desconhecidas associadas com a máquina e o compilador usados para processar o algoritmo e (2) usa algumas hipóteses simplificadoras (AHO; ULLMAN, 1992).

Considere a seguinte definição precisa do que significa uma função ser "*big-oh*" de outra, segundo Aho e Ullman (1992). Considere $T(n)$ como uma função que representa o tempo de processamento de determinado programa, medida com relação ao tamanho do input, n. Naturalmente, n é um número inteiro não negativo e $T(n)$ é não negativo para qualquer valor de n. Sendo $f(n)$ uma função definida nos inteiros não negativos n, diz-se que "$T(n)$ é $O(f(n))$" se $T(n)$ é, no máximo, uma constante vezes $f(n)$, exceto, possivelmente, para alguns pequenos valores de n. Formalmente, $T(n)$ é $O(f(n))$ se existirem um inteiro n_0 e uma constante $c > 0$ tal que, para todos os inteiros $n \geq n_0$, tem-se $T(n) \leq cf(n)$.

Os princípios básicos da notação "*big-oh*" são:

1. *Fatores constantes não importam*: para qualquer constante positiva d e qualquer função $T(n)$, $T(n)$ é $O(dT(n))$, independentemente do valor de d desde que $d > 0$.

2. *Termos de menor ordem não importam*: suponha que $T(n)$ é um polinômio de forma:

$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_2 n^2 + a_1 n + a_0$$

onde a_k é positivo. Todos os termos com excessão do primeiro (o termo de maior expoente, k) são desconsiderados e, pelo princípio 1, ignora-se também a constante a_0. Portanto, $T(n)$ é $O(n^k)$.

Além disso, considera-se ainda a objetividade e a simplicidade nessa notação para evitar ambiguidades. Dessa forma, a notação "*big-oh*" expressa a complexidade de um algoritmo através de um único termo cujo coeficiente é igual a 1. Então, expressões como $0.5n^3 + 3n$, por exemplo, são representadas por $O(n^3)$ e fala-se que tal algoritmo apresenta um tempo de processamento cúbico, o que permite ter uma ideia concisa de como o tempo de processamento do algoritmo se comporta com grandes valores de input. A tabela (3) a seguir apresenta alguns dos tempos de processamento mais comuns em algoritmos e o nome informal utilizado para descrever tais tempos de processamento.

Tabela 3 – Tempos de processamento mais comuns e seus nomes informais

Big-Oh	Nome Informal
$O(1)$	Constante
$O(\log n)$	Logarítmico
$O(n)$	Linear
$O(n \log n)$	$n \log n$
$O(n^2)$	Quadrático
$O(n^3)$	Cúbico
$O(2^n)$	Exponencial

Fonte: Adaptado de Aho e Ullman (1992)

A análise da eficiência de um algoritmo através da notação $O(n)$ é uma análise do limite superior assintótico do tempo de processamento do algoritmo, ou seja, um tempo de processamento para o pior caso. Existem também notações que capturam outras ideias importantes na análise de complexidade, como as notações $\mathbf{\Omega}$ e $\mathbf{\Theta}$.

2.2.2 Notação Ω

A notação Ω oferece uma análise da eficiência do algoritmo com um olhar para o limite inferior assintótico do algoritmo. Para uma função $g(n)$, denota-se por $\Omega(g(n))$ o conjunto de funções

$$\Omega(g(n)) = \{f(n) : 0 \leq cg(n) \leq f(n)\}$$

para constantes positivas c e n_0 e para qualquer $n \geq n_0$. Portanto, se o tempo de processamento de um algoritmo é $\Omega(g(n))$, isto significa que independentemente do tamanho do input n escolhido, o tempo de processamento do algoritmo para esse input é, no mínimo, uma constante vezes $g(n)$ para um n suficientemente grande. Nesse caso, a notação aponta para o tempo de processamento para o melhor caso (CORMEN et al., 2009; KLEINBERG; TARDOS, 2005).

2.2.3 Notação Θ

Para uma determinada função $g(n)$, denota-se por $\Theta(g(n))$ o conjunto de funções definido por:

$$\Theta(g(n)) = \{f(n) : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

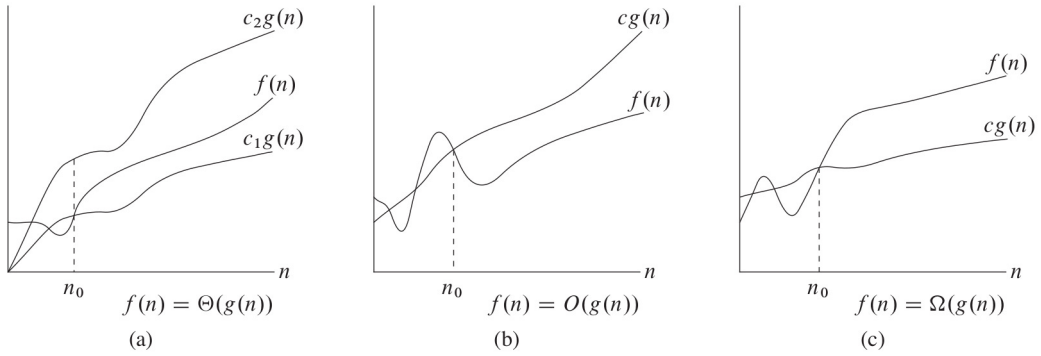
para constantes positivas c_1, c_2 e n_0 e para qualquer $n \geq n_0$. Isto é, uma função $f(n)$ pertence ao conjunto de funções representadas por $\Theta(g(n))$ se existirem constantes positivas c_1 e c_2 tal que $f(n)$ pode ser limitada abaixo por $c_1 g(n)$ e acima por $c_2 g(n)$ para um valor de n suficientemente grande (CORMEN et al., 2009).

Observe que $f(n) = \Theta(g(n))$ implica $f(n) = O(g(n))$ uma vez que Θ é uma noção mais forte do que O. A partir das notações assintóticas, o teorema a seguir é evidente (CORMEN et al., 2009):

Teorema 2.1. *Para quaisquer duas funções $f(n)$ e $g(n)$, tem-se $f(n) = \Theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.*

Os gráficos da figura (11) ilustram o significado das notações apresentadas. Em cada um dos gráficos o valor de n_0 é o menor possível; qualquer valor maior é igualmente aceitável. Em resumo, a notação Θ limita uma função entre fatores constantes, a notação O fornece um limite superior para a função por um fator constante, e a notação Ω fornece um limite inferior para a função por um fator constante.

Figura 11 – Gráficos das notações Θ, O e Ω



Fonte: Adaptado de Cormen et al. (2009)

Para finalizar esta seção, cabe realizar uma apresentação sucinta de classes de problemas uma vez que a caracterização de problemas desempenha um papel crucial na resolução dos mesmos.

2.2.4 Classes de Problemas

Um problema é uma coleção de "situações" (instâncias), todas geradas de forma semelhante. Problemas de decisão são problemas cuja resposta é do tipo "sim" ou "não" enquanto que problemas de otimização são aqueles cuja resposta é um valor extremo dentro de uma coleção de valores possíveis.

Problemas de decisão são problemas computacionais especiais e a maior parte da teoria da complexidade é baseada em problemas de decisão. Observe, por exemplo, os problemas de decisão correspondentes à programação linear e à programação inteira (KORTE; VYGEN, 2012):

Inequações Lineares:

Instância: Uma matriz $A \in \mathbb{Z}^{m \times n}$ e um vetor $b \in \mathbb{Z}^m$.

Problema de decisão: Existe um vetor $x \in \mathbb{Q}^n$ tal que $Ax \leq b$?

Inequações Lineares Inteiras:

Instância: Uma matriz $A \in \mathbb{Z}^{m \times n}$ e um vetor $b \in \mathbb{Z}^m$.

Problema de decisão: Existe um vetor $x \in \mathbb{Z}^n$ tal que $Ax \leq b$?

Problemas de otimização apresentam uma divisão natural entre aqueles problemas com variáveis contínuas e aqueles com variáveis discretas, estes últimos chamados combinatórios. Em geral, pode-se utilizar as definições a seguir para representar formalmente um problema de otimização (PAPADIMITRIOU; STEIGLITZ, 1998):

Definição 2.1. Uma instância de um problema de otimização é um par (F, c), onde F é um conjunto qualquer, o domínio de pontos viáveis; c é a função custo, um mapeamento

$$c : F \longrightarrow \mathbb{R}^1$$

O problema consiste em encontrar um $f \in F$ para o qual

$$c(f) \leq c(y) \text{ para todo } y \in F$$

Tal ponto f é chamado uma solução ótima global para a instância dada.

Definição 2.2. Um problema de otimização é um conjunto I de instâncias de um problema de otimização.

Um problema de otimização frequentemente pode ser reduzido a um problema de decisão. Considere o problema de otimização: minimize $f(x)$ com $x \in X$, onde X é uma coleção de elementos derivados do input do problema e f é uma função de valores racionais em X. O problema de otimização recém apresentado é equivalente ao problema de decisão:

$$\text{dado um número racional } r, \text{ existe um } x \in X \text{ tal que } f(x) \leq r \ ?$$

Um problema de decisão é completamente descrito pelo input para o qual a resposta é "sim". Considere a seguinte formalização, segundo Schrijver (2003). Fixe um conjunto finito Σ de tamanho, no mínimo, 2 - $\{0, 1\}$, por exemplo. Seja Σ^* o conjunto de todas as *strings* (palavras) finitas de letras de Σ. O tamanho de uma palavra é o número de letras da palavra, contando multiplicidades. Um problema é qualquer subconjunto Π de Σ^* e o correspondente problema informal é definido como:

$$\text{dada um palavra } x \in \Sigma^*, \ x \text{ pertence a } \Pi \ ?$$

A palavra x é o chamado input do problema e, como definido acima, fala-se em uma instância de um problema Π, quando pede-se um input concreto x se x pertence a Π.

Diz-se que um algoritmo é polinomial ou de tempo polinomial se sua execução finaliza após um número de operações limitada por um polinômio em relação ao tamanho do input, ou seja, o número de bits que descrevem o input. Essa noção é importante para a definição das classes de problemas apresentadas a seguir.

P, NP e co-NP são coleções de problemas de decisão. Um problema é chamado polinomial se existe um algoritmo polinomial que decide se uma dada palavra $x \in \Sigma^*$ pertence a Π. A coleção de todos os problemas polinomiais $\Pi \subseteq \Sigma^*$ é denotada por P. NP, por sua vez, é definida como a coleção de todos os problemas de decisão para os quais cada input com resposta positiva apresenta um "certificado", verificável em tempo polinomial, da corretude da resposta. Uma outra forma de caracterizar NP é como a coleção de problemas de decisão que podem ser reduzidas em tempo polinomial ao problema da satisfabilidade, isto é, checar se uma expressão booleana pode ser satisfeita. Por fim, a classe co-NP consiste de todos os problemas Π para os quais o problema complementar $\Sigma \setminus \Pi$ pertecem a NP (SCHRIJVER, 2003).

As definições acima, apesar de superficiais, serão uma introdução suficiente às características fundamentais desses problemas, apresentadas a seguir. O primeiro fato importante é que, apesar de todo esforço dos pesquisadores, matemáticos e cientistas durante todos esses anos, uma questão fundamental persiste, P=NP ? De fato, essa questão é um dos seis problemas do milênio definidos pelo *Clay Mathematics Institute* que continuam sem resposta e cuja resolução vale um prêmio de US$ 1.000.000,00 (JOHNSON, 2012).

A resolução definitiva dessa questão provocaria impactos profundos em diversas áreas da matemática discreta, ciências da computação, criptografia e engenharia de maneira geral. Se P=NP puder ser provado, a prova deverá conter uma novo e revolucionário algoritmo ou, alternativamente, tornará o conceito de tempo polinomial obsoleto e sem significado. Se P $\neq$ NP puder ser provado, a prova deverá revelar algumas das razões pelas quais alguns problemas são mais difíceis que outros o que, por sua vez, levaria os cientistas e designers de algoritmos a atacar o núcleo dessas dificuldades (SCHRIJVER, 2003).

Existe ainda uma categoria especial de problemas contendo milhares de problemas combinatórios, chamados problemas NP-completos. Não são conhecidos algoritmos polinomiais para resolver os problemas NP-completos. Porém, também não existe qualquer prova de que um tal algoritmo não possa existir. Além disso, uma grande parte dos problemas nessa categoria têm sido caracterizados, e resta provado que eles são equivalentes no seguinte sentido: um algoritmo polinomial para qualquer um deles implicaria a existência de um algoritmo polinomial para todos eles (KLEINBERG; TARDOS, 2005).

2.3 Otimização Combinatória

A otimização combinatória tem suas raízes em áreas diversas como combinatória, pesquisa operacional e ciências da computação e tem como principal motivação os milhares de problemas reais que podem ser formulados através dela (KORTE; VYGEN, 2012).

Lawler (1976), em um dos principais clássicos da área, observou, já na década de 1970, que os problemas de otimização combinatória surgem em todas as áreas, especialmente nas áreas de tecnologia e gerenciamento industrial. Observou também que a consciência da importância desses problemas estava sendo acompanhada por uma explosão combinatória de propostas para suas resoluções.

Uma comprovação simples da exatidão das observações de Lawler (1976) é dada pela quantidade de trabalhos e livros publicados em otimização combinatória a partir de então. Com efeito, o livro de Lawler foi o primeiro de uma série de livros com o título "Otimização Combinatória" (e algum subtítulo como *algoritmos e complexidade*, ou *teoria e algoritmos*), como, por exemplo, Papadimitriou e Steiglitz (1998), Cook et al. (1998), Cornuejols (2001) e Schrijver (2003).

Em termos gerais, diversas categorias de problemas de otimização já foram identificadas e caracterizadas na literatura, como mostra Odili (2017). Essa elevada quantidade de tipos de problemas de otimização provém da busca por soluções de problemas reais encontrados em praticamente todos os campos da economia e da sociedade. O autor supracitado, por exemplo, argumenta que dificilmente existe algum campo - da medicina, farmácia, ciências, engenharia aos negócios - que atualmente não exija algum tipo de otimização.

Existe uma divisão natural de todos esses tipos de otimização em duas grandes categorias: otimização contínua e otimização discreta. A otimização contínua foca na minimização ou maximização de funções de números reais contínuos enquanto que a otimização discreta se utiliza de números inteiros na busca pela otimização de funções.

Otimização discreta pode ainda ser subdividida em otimização combinatória e programação inteira. Programação inteira (IP - *integer programming*) trata da maximização ou minimização de uma função de várias variáveis sujeita a restrições na forma de inequações e equações, e restrições de integralidade em algumas ou em todas as variáveis (NEMHAUSER; WOLSEY, 1989). Os casos especiais que admitem frações ou decimais como parte das restrições compõem a programação inteira mista (MIP - *mixed integer programming*). Na maioria das vezes, no entanto, os problemas de IP são tecnicamente classificados como problemas de programação linear inteira (ILP - *integer linear programming*) uma vez que nesses problemas tanto a função objetivo como as restrições são lineares (ODILI, 2017).

Na otimização combinatória, busca-se um objeto ótimo em uma coleção finita de objetos. Em geral, a coleção apresenta uma forma concisa de representação, como um grafo, mas apresenta também uma quantidade enorme de objetos. Especificamente, o número de objetos da coleção cresce exponencialmente de acordo com o tamanho da representação. Dessa forma, a análise individual dos objetos para seleção do ótimo é inviável na maioria das vezes. Métodos mais eficientes devem ser buscados (SCHRIJVER, 2003; ODILI, 2017).

A otimização combinatória, como uma disciplina matemática coerente, é relativamente nova. Schrijver (2005) aponta que o que existia até a década de 1950 era um número de linhas de pesquisas independentes, considerando separadamente problemas como o da designação ótima, da árvore geradora mínima, do transporte e o problema do caixeiro viajante. Apenas

quando a programação linear e inteira se tornou uma ferramenta disponível e a área de pesquisa operacional começou a despertar atenção intensiva é que esses problemas foram postos em uma mesma perspectiva e relações entre eles foram descobertas.

De fato, a programação linear forma o ponto de articulação da história da otimização combinatória. Após a formulação da programação linear como um problema geral e do desenvolvimento do método Simplex em 1947 por Dantzig, praticamente todos os problemas de otimização combinatória foram atacados, muitas vezes com sucesso, através das técnicas de programação linear. Por outro lado, alguns problemas permitem uma resolução mais eficiente do que a resolução através do método Simplex, como o problema do fechado de peso máximo e o problema do fluxo máximo. Métodos mais especilizados de resolução de problemas combinatórios têm se tornado necessários conforme o tamanho da entrada de dados dos problemas aumenta.

3 PROBLEMA DO CONJUNTO FECHADO DE PESO MÁXIMO NUM DIGRAFO

Rhys (1970) e Balinski (1970) estudaram independentemente o problema da seleção de terminais para manuseio de fretes que encontrava aplicações diversas já naquela época e chamou a atenção desses pesquisadores por apresentar uma dificuldade particular de natureza exponencial.

O problema consite em determinar o estabelecimento de ligações entre terminais de serviços de forma ótima. Para uma determinada companhia de transporte poder operar o transporte entre dois pontos (duas cidades, duas regiões, etc) a companhia precisa estabelecer ou construir um terminal de serviços em cada um dos pontos. A operação dos serviços de transporte (de pessoas, de cargas, etc) nessa ligação naturalmente confere à empresa um faturamento enquanto que o estabelecimento dos terminais implica em um custo fixo referente a cada terminal.

A análise tradicional de viabilidade financeira torna-se inviável nesta situação quando se observa que as ligações não são independentes. Caso as ligações fossem independentes, bastaria subtrair os custos do faturamento para determinar a viabilidade financeira do empreendimento. Entretanto, uma vez que um terminal é construído, é possível operar outras ligações a partir deste terminal com qualquer outro ponto onde já exista um terminal de serviços operante. Dessa forma, os custos fixos de construir ou estabelecer um terminal são compartilhados entre as diversas ligações operadas a partir do terminal. Por essa razão, Rhys (1970) generalizou em certa medida o problema da seleção chamando-o de problema dos custos fixos compartilhados.

O problema da seleção (ou dos custos fixos compartilhados) pode ser pensado da seguinte forma: cada ligação determina um subconjunto de dois pontos, cada ponto apresentando um determinado custo. Cada ligação, no entanto, apresenta um benefício e o objetivo é selecionar uma coleção de subconjuntos de pontos de tal forma que o benefício total menos o custo de todos os pontos na coleção seja o máximo possível.

Esse problema pode ser generalizado para comportar subconjuntos de mais de dois pontos. De fato, Lawler (1976) apresentou uma outra versão do problema da seleção, denominado por ele de problema da provisão, incorporando essa possibilidade. O problema da provisão se assemelha ao problema da mochila clássico, um famoso problema NP-completo.

O problema da provisão pode ser apresentado da seguinte forma: pode-se comprar ou selecionar itens de uma lista de n itens, onde cada item j apresenta um custo c_j. Existem m conjuntos de itens, $S_1, S_2, \ldots, S_m$ que conferem um benefício especial. Se todos os itens de um conjunto S_i forem selecionados, um benefício b_i (em termos de utilidade, desconto, etc.) é conferido. Os conjuntos são arbitrários e não precisam ser relacionados de qualquer maneira particular, de forma que um determinado item pode estar em vários conjuntos.

Essa formulação é uma forma um pouco mais sofisticada do problema da mochila uma vez que considera que o benefício atibuído aos itens não é independente. Em diversas situações,

essa hipótese é mais realista do que a hipótese de independência usada no problema da mochila. Uma laterna a pilha, por exemplo, confere pouca utilidade ao usuário se este não dispuser de pilhas. Por outro lado, uma vez que as pilhas são adquiridas, elas podem ser usadas com outros itens (aparelhos de rádio e de comunicação, controles, calculadoras). Não existe restrição com relação ao número de itens que podem ser selecionados e o objetivo é simplesmente maximizar o benefício líquido, ou seja, o benefício total menos o custo total dos itens selecionados.

Lawler observou que, mesmo sem restrições em relação à seleção dos itens, o problema parecia incrivelmente difícil, intratável de fato. Surpreendentemente, porém, ele poderia ser modelado em termos de um problema de corte mínimo e solucionado facilmente através da dualidade max-flow min-cut.

Para formular matematicamente o problema da provisão, define-se dois tipos de variáveis binárias:

$$v_j = \begin{cases} 1, & \text{se o item } j \text{ é selecionado,} \\ 0, & \text{caso contrário.} \end{cases}$$

$$u_i = \begin{cases} 1, & \text{se o conjunto } i \text{ é selecionado,} \\ 0, & \text{caso contrário.} \end{cases}$$

O problema consiste, então, em maximizar a função

$$Z = \sum_{i}^{m} b_i u_i - \sum_{j}^{n} c_j v_j \tag{3.1}$$

sujeita a

$$v_j \geq u_i, \qquad \forall j \in S_i, \qquad i = 1, \ldots, m \tag{3.2}$$

$$u_i, v_j \in \{0, 1\}, \qquad i = 1, \ldots, m, \qquad j = 1, \ldots, n. \tag{3.3}$$

Observe que as restrições (3.2) e (3.3) impedem que o benefício referente a um conjunto i seja conferido sem que todos os itens do conjunto tenham sido selecionados.

Lawler (1976) propõem uma resolução através da introdução de $m + n$ novas variáveis e trabalhando com o problema do corte mínimo. Rhys (1970) e Balinski (1970) também focaram em resolver o problema da seleção através de um problema do corte mínimo, mas com uma abordagem um pouco diferente. A estrutura dos problemas apresentados até aqui, no entanto, é a mesma e todos eles podem ser modelados em termos de um grafo bipartido especial.

Considere os seguintes itens e seus custos, bem como os possíveis conjuntos de itens e os benefícios relativos na tabela (4). A figura (12) esquematiza o exemplo do problema da provisão. Os nós do lado direito, enumerados de 1 a 6, representam os itens juntamente com seus custos. Os nós do lado esquerdo representam os conjuntos juntamente com os respectivos benefícios. Os arcos representam as relações de pertinência, ou seja, existe um arco saindo de um vértice representante de um conjunto para cada um dos vértices representantes dos itens que compõem o conjunto.

Tabela 4 – Itens e Conjuntos

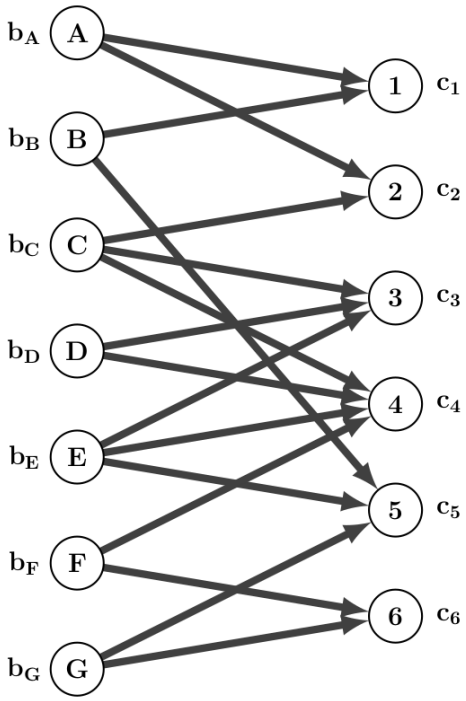
Item	Custo
1	c_1
2	c_2
3	c_3
4	c_4
5	c_5
6	c_6

Conjunto	Benefício
A = 1, 2	b_A
B = 1, 5	b_B
C = 2, 3, 4	b_C
D = 3, 4	b_D
E = 3, 4, 5	b_E
F = 4, 6	b_F
G = 5, 6	b_G

Fonte: Adaptado de Lawler (1976)

Para transformar o problema da seleção em um problema de fluxo a partir da figura (12), dois novos vértices são adicionados, uma fonte s e um sorvedouro t. Um arco saindo da fonte para cada nó representante de um conjunto é criado, e o benefício do conjunto passa a representar a capacidade do arco. Semelhantemente, um arco saindo de cada um dos nós representantes dos itens é adicionado à rede, e o custo do item passa a representar a capacidade do arco. A capacidade dos demais arcos é infinita.

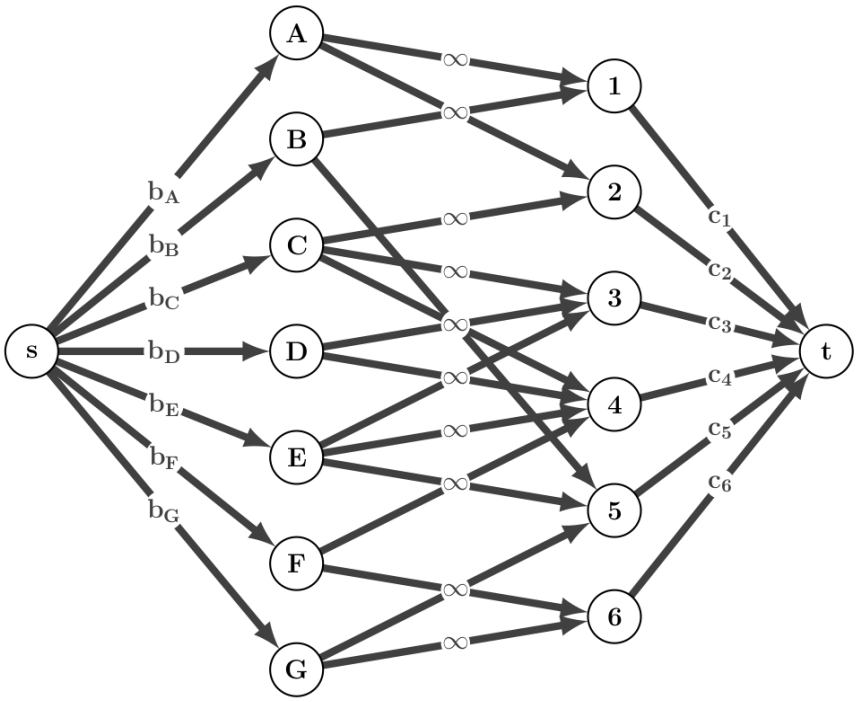
Figura 12 – Ilustração esquemática do problema da provisão



Fonte: O autor (2018)

A figura (13) ilustra esta modelagem. A resolução de um problema max-flow min-cut na rede da figura (13) resolve o problema da provisão, conforme será provado mais adiante neste capítulo.

Figura 13 – Rede do problema da provisão



Fonte: O autor (2018)

Picard (1976) generalizou o problema da seleção para o problema do fechado de peso máximo. Sua principal motivação era na área de engenharia de minas, o problema da mineração a céu aberto (OPMP - *Open Pit Mining Problem*).

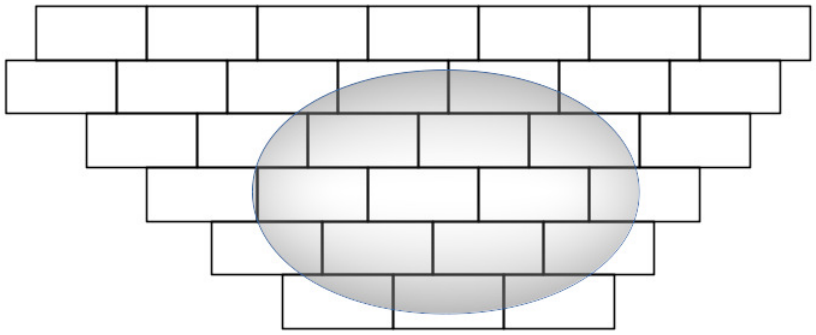
A indústria de mineração naquela época estava desenvolvendo métodos de solução para o OPMP separadamente da comunidade da pesquisa operacional de forma que o grande mérito de Picard foi construir uma ponte entre esses dois campos, permitindo que algoritmos eficientes para problemas de fluxo máximo e corte mínimo fossem usados em problemas de mineração (HOCHBAUM, 2008).

Essa ponte foi benéfica também para a comunidade da pesquisa operacional uma vez que, mais tarde, alguns métodos da engenharia de mineração foram usados como base para o desenvolvimento de soluções para problemas de interesse da pesquisa operacional, como é o caso do algoritmo pseudofluxo.

Antes das operações de escavação serem iniciadas, a região de interesse é particionada em blocos e o valor do mineral em cada bloco, em função do volume de mineral, é estimado por informações geológicas através de perfurações. A figura (14) representa um corte vertical da região de mineração, ilustrando essa situação, onde a região destacada representa o volume de mineral na região. O custo de minerar e processar cada bloco também é estimado e um determinado lucro (ou prejuízo) pode ser atribuído a cada bloco (AMANKWAH; LARSSON; TEXTORIOUS, 2014).

O OPMP consiste em determinar, nesses termos, o contorno ótimo, ou seja, quais blocos

Figura 14 – Região de mineração



Fonte: O autor (2018)

escavar de forma a maximizar o lucro total. As únicas restrições dizem respeito às relações de precedência dos blocos e ao ângulo de segurança a ser mantido. As retrições de precedência obrigam que todos os blocos diretamente acima de um determinado bloco j sejam escavados antes que o bloco j possa ser escavado. As restrições de contorno ou ângulo garantem que as paredes do poço de mineração sejam mantidas com um certo grau de inclinação para garantir a segurança das operações.

O OPMP pode ser representado por um digrafo ponderado nos vértices, onde cada vértice representa um bloco e o peso de um vértice é o resultado que seria obtido com a escavação desse bloco individualmente, ou seja, o valor do metal contido no bloco menos o custo da extração e processamento desse bloco unicamente. Portanto, o peso de um bloco pode ser positivo, nulo ou negativo.

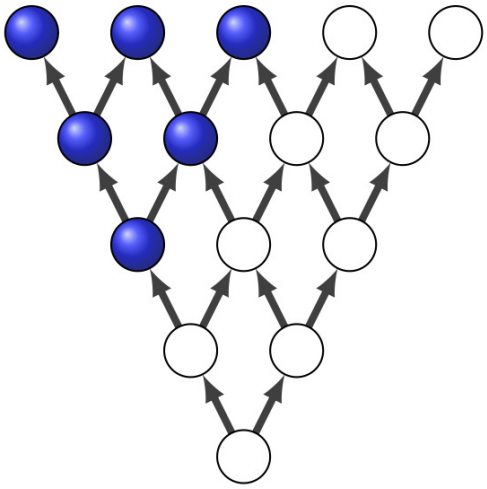
Os arcos representam as restrições de precedência de forma que um arco (i, j) no digrafo significa que o bloco i não pode ser escavado antes da escavação do bloco j. Observe que a restrição de precedência impõe que qualquer conjunto de vértices selecionados no digrafo não apresente arcos com origem no conjunto e extremidade fora do conjunto. Assim, uma solução do problema é um conjunto de vértices fechado em relação a esses arcos e por isso o problema é chamado de problema do fechado máximo. A figura (15) ilustra o grafo característico de um OPMP com os vértices em destaque formando um conjunto fechado.

Formalmente, pode-se definir o problema do fechado máximo da seguinte forma: dado um digrafo $D = (V, A)$ ponderado nos vértices, onde b_i representa o peso (positivo, nulo ou negativo) do vértice i, para todo $i \in V$, encontre um subconjunto fechado de vértices $U \subseteq V$ tal que $\sum_{i \in U} b_i$ é máximo.

Para formular matematicamente este problema, define-se

$$x_j = \begin{cases} 1, & \text{se o item } j \text{ está no fechado,} \\ 0, & \text{caso contrário.} \end{cases}$$

Figura 15 – Grafo do OPMP



Fonte: O autor (2018)

e b_j o peso do nó j. Dessa forma, tem-se o seguinte problema de otimização:

$$\begin{aligned} \text{max} \quad & \sum_{j \in V} b_j x_j \\ \text{sujeito a} \quad & x_j - x_i \geq 0 \quad \forall (i,j) \in A \\ & 0 \leq x_j \leq 1 \quad inteiro \quad j \in V \end{aligned} \tag{3.4}$$

O problema do fechado máximo, então, é essencialmente uma versão do problema da seleção onde o grafo não é bipartido no sentido de que não há distinção entre itens e conjuntos (HOCHBAUM, 2004). Observa-se que o problema torna-se trivial se todos os vértices tiverem pesos de mesmo sinal. Se todos forem positivos, o fechado ótimo é obviamente todo o grafo. Se todos forem negativos, o fechado ótimo é um conjunto vazio de vértices. Picard (1976) ofereceu uma ampla generalização e provou que também o problema do fechado máximo pode ser solucionado através de um corte mínimo.

Além disso, percebe-se, através dessa formulação, que a matriz de restrições para este problema apresenta, em cada linha, um 1 e um -1. A matriz de restrições é indexada nas linhas pelos arcos e nas colunas pelos vértices e sua transposta é a matriz de incidência do problema. A matriz de incidência de um problema do fechado máximo, portanto, é uma matriz que apresenta um 1 e um -1 em cada coluna e todas as demais entradas da coluna são iguais a 0, o que indica que a matriz é totalmente unimodular (HOCHBAUM, 2001).

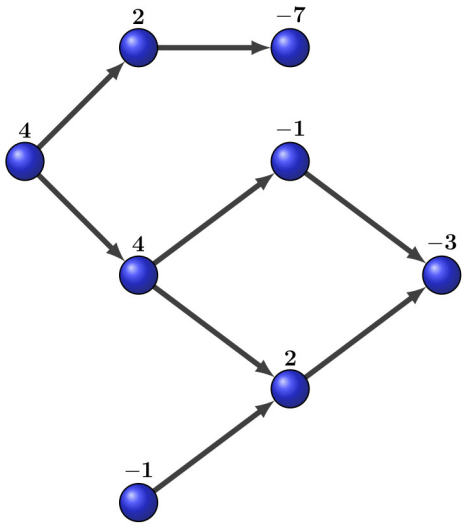
Uma matriz é totalmente unimodular se todos os seus subdeterminantes são iguais a -1, 1 ou 0, o que implica que suas entradas também são iguais a -1, 1 ou 0, uma vez que cada entrada é um subdeterminante. Matrizes totalmente unimodulares são especiais em otimização combinatória porque determinam uma classe de problemas de programação linear com solução ótima inteira. Mas especificamente, qualquer problema de programação linear com entrada

inteira e com matriz de restrições totalmente unimodular possui uma solução ótima inteira (SCHRIJVER, 1999).

3.1 Redução para o Problema do Corte Mínimo

Considere um digrafo $D = (V, A)$ e um peso b_v associado a cada vértice $v \in V$, como na figura (16). Define-se um digrafo modificado $D' = (V', A')$ e um vetor de capacidades c da seguinte forma:

Figura 16 – Digrafo Original



Fonte: Adaptado de Cook et al. (1998)

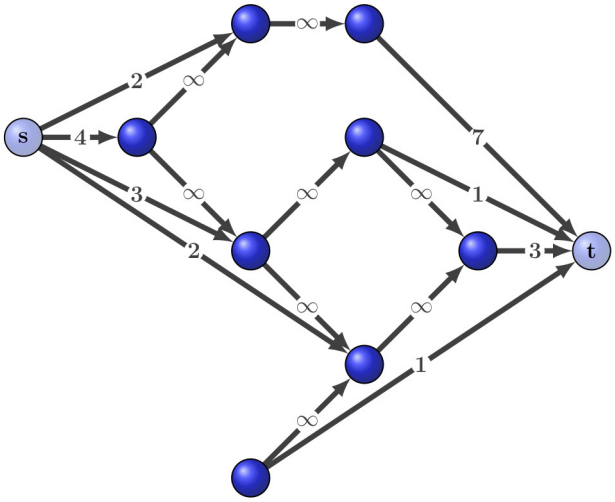
- $V' = V \cup \{s, t\}$;
- Para cada vértice $v \in V$ com $b_v > 0$, D' tem um arco (s, v) com capacidade $c_{sv} = b_v$;
- Para cada vértice $v \in V$, com $b_v < 0$, D' tem um arco (v, t) com capacidade $c_{vt} = -b_v$;
- As capacidades dos arcos restantes (ou seja, os arcos originais de A) são infinitas.

Observe o digrafo modificado a partir desse procedimento na figura (17). Qualquer s-t corte finito $\partial^+(U)$ no digrafo modificado D' será tal que não inclui nenhum arco do digrafo original, uma vez que todos esses arcos, em D', apresentam capacidade infinita. Desta forma, $U = \{s\} \cup S$, onde S é um fechado no digrafo original D. O peso do fechado S, $w(S)$, é dado por:

$$w(S) = \sum_{v \in S} (b_v \mid b_v \geq 0) + \sum_{v \in S} (b_v \mid b_v < 0) \tag{3.5}$$

Reciprocamente, qualquer fechado $S \subseteq V$ determina um s-t corte $\partial^+(S \cup \{s\})$. Dois tipos de arcos fazem parte desse corte: arcos saindo da fonte s para vértices fora do fechado S; e

Figura 17 – Digrafo Modificado



Fonte: Adaptado de Cook et al. (1998)

arcos saindo de vértices do fechado para o sorvedouro t. Portanto, a capacidade do corte $\partial^+(U)$, $c(\partial^+(U))$, é calculada por:

$$c(\partial^+(U)) = \sum_{v \notin S}(b_v \mid b_v > 0) - \sum_{v \in S}(b_v \mid b_v < 0) \tag{3.6}$$

Considere a soma dos pesos não negativos dos vértices em S, $\sum_{v \in S}(b_v \mid b_v \geq 0)$. Adicionando-se e subtraindo-se esta quantidade no termo da direita da equação (3.6), obtém-se:

$$\begin{aligned} c(\partial^+(U)) = &[\sum_{v \notin S}(b_v \mid b_v > 0) + \sum_{v \in S}(b_v \mid b_v \geq 0)] - \\ &[\sum_{v \in S}(b_v \mid b_v < 0) + \sum_{v \in S}(b_v \mid b_v \geq 0)] \end{aligned} \tag{3.7}$$

O lado direito da igualdade na equação (3.7) apresenta dois componentes especiais. O primeiro componente, composto pelos dois somatórios no primeiro par de colchetes, é a soma dos pesos não negativos de todos os vértices do digrafo. Essa quantidade, denotada por W^+, é constante em relação à escolha do fechado, dependendo apenas dos pesos do digrafo original. O segundo componente é precisamente o peso do fechado S. Substituindo essas informações na equação (3.7) e rearranjando-a, obtém-se:

$$w(S) = W^+ - c(\partial^+(U)) \tag{3.8}$$

Como W^+ é constante em relação à escolha do fechado, basta minimizar a capacidade do corte $\partial^+(U)$ para obter um fechado de peso máximo, isto é, basta solucionar o problema do corte mínimo no digrafo modificado. Encontrado o conjunto de vértices U cujo s-t corte apresenta a menor capacidade em D' e observando-se que $U = \{s\} \cup S$, sabe-se que S, em D, é um fechado de peso máximo.

Ford e Fulkerson (1956) estabeleceram que o valor do fluxo máximo é igual ao valor de um s-t corte de capacidade mínima, isto é, estes dois problemas são formulações lineares duais um do outro. As informações apresentadas por esses dois problemas, entretanto, são diferentes pois enquanto o problema do corte mínimo busca apenas uma partição do conjunto de vértices em dois subconjuntos, o problema do fluxo máximo atribui fluxo a cada arco.

O problema do corte mínimo é um dos mais importantes problemas nas ciências da computação e apresenta numerosas aplicações em diversas áreas (DUAN; XU, 2014). Juntamente com o problema do fluxo máximo, forma uma das bases da otimização combinatória (GRANOT et al., 2012).

Hochbaum e Chen (2000) afirmaram que o único método conhecido para resolver o problema do corte mínimo exigia primeiramente a resolução do problema do fluxo máximo e, a partir deste, a determinação do corte mínimo é possível observando os vértices que podem ser alcançados a partir da fonte no grafo residual. Essa afirmação parece continuar válida em certa medida no sentido de que os algoritmos conhecidos atualmente para resolver o problema do corte mínimo são geralmente baseados no problema do fluxo máximo ou, mais raramente, em algum certificado de otimalidade que pode ser usado tanto para o problema do fluxo máximo, como para o problema do corte mínimo.

As assimetrias entre resolver o problema do corte mínimo e o problema do fluxo máximo parecem indicar que a resolução do problema do corte mínimo é mais fácil do que a resolução do problema do fluxo máximo, ainda assim, a maior parte dos algoritmos conhecidos resolvem o problema do corte mínimo através do problema do fluxo máximo. Além disso, dado apenas o corte mínimo, não existe forma eficiente de determinar o fluxo em cada arco (HOCHBAUM; CHEN, 2000).

Em resumo, o problema do conjunto fechado de peso máximo generaliza os problemas da seleção (custos fixos compartilhados), da provisão e da mineração a céu aberto. Além disso, o problema do conjunto fechado de peso máximo resolve também o problema do *maximum blocking-cut* (discutido no capítulo 5). Trata-se, portanto, de uma modelagem flexível e versátil e encontra muitas aplicações na prática. Na próxima seção são apresentadas algumas dessas aplicações.

3.2 Aplicações

As aplicações do problema do conjunto fechado de peso máximo se estendem por uma variedade de domínios, com especial efeito em problemas de engenharia de produção e operações, mas englobando também problemas de defesa e de manutenção, entre outros. Algumas dessas aplicações são brevemente apresentadas nesta seção.

3.2.1 Problemas de Agendamento/Planejamento

Um dos problemas mais comuns na produção industrial e de serviços é o problema do agendamento (*scheduling*) e se refere à alocação de recursos a tarefas considerando um certo

intervalo e buscando otimizar um ou mais objetivos. É um processo decisório rotineiro e com variados níveis de complexidade (PINEDO, 2008).

Faaland e Schmitt (1987) formularam um problema de agendamento de tarefas em um processo de fabricação e montagem baseado no problema da seleção. O modelo formulado por eles apresenta contribuição significante à medida que considera tanto as penalidades decorrentes de atrasos na entrega dos produtos acabados, como o custo de manutenção de estoques para produtos acabados, balanceando os custos de ambas as situações na determinação de tempos viáveis de conclusão de tarefas.

Esse problema de agendamento envolve duas decisões: (1) a ordem na qual cada tarefa deve ser desenvolvida em cada estação de trabalho (expedição); (2) a determinação do tempo de conclusão para as tarefas. As hipóteses do problema são, basicamente:

- cada tarefa é um elemento não divisível;
- o tempo de processamento atribuído a cada tarefa é composto de um tempo de *setup* (preparação) e um tempo de execução/montagem por unidade, multiplicado pela quantidade de itens da batelada;
- todos os tempos de processamento e as datas de entrega dos produtos acabados são conhecidos;
- o tempo de setup de cada tarefa é independente da ordem de expedição;
- depois de iniciada, uma tarefa não pode ser interrompida e após concluída a tarefa, não há atrasos na movimentação dos itens para as próximas estações de trabalho;
- uma tarefa não pode ser iniciada até que todas as tarefas precedentes na fabricação do produto final tenham sido executadas e essas relações entre tarefas são conhecidas;
- duas ou mais tarefas designadas para a mesma estação de trabalho não podem ser processadas simultaneamente.

A estrutura de custos do problema é composta de dois tipos de custos: custos decorrentes de atraso na entrega de produtos acabados e custos de manutenção de estoques. Para o primeiro tipo de custo, o consumidor ou o departamento de vendas da empresa determina uma data para entrega do produto final. Se uma tarefa i é a tarefa final na fabricação de um produto, considere D_i como a data de entrega e L_i a penalidade por unidade de tempo por atraso na finalização do produto. Se o tempo de conclusão x_i excede D_i, atrasando a entrega do produto, uma penalidade $L_i(x_i - D_i)$ é aplicada, refletindo tanto a perda de confiança dos clientes como o custo de oportunidade decorrente do atraso no recebimento do pagamento correspondente ao atraso da entrega do produto.

Cada vez que uma tarefa é realizada, um valor adicional é investido no produto na forma de trabalho e materiais. Esses valores adicionais são retidos até o recebimento do pagamento

pelo produto e constituem o segundo tipo de custos. Se E_i representa o custo de manutenção de estoque por unidade de tempo para tarefa i e r representa a taxa de juros, considere o seguinte exemplo.

Uma determinada tarefa i, executada na estação de trabalho k, utiliza matéria-prima no valor de \$ a_i. Além disso, para um custo de trabalho de b_k por hora na estação k e um tempo de processamento de p_i horas para a atividade i, o custo de manutenção em estoque é dado por $E_i = (a_i + b_k p_i)r$. Se $q(i)$ é a última tarefa na estrutura de produção do produto do qual a tarefa i é componente, então o período entre a finalização da tarefa i e a entrega do produto é dado por $\max(D_{q(i)}, x_{q(i)}) - x_i$, e o custo de manutenção de estoque associado à tarefa i é $E_i[\max(D_{q(i)}, x_{q(i)}) - x_i]$.

Seja F o conjunto de cada uma das últimas tarefas na estrutura de produção de cada produto e $N = \{1, 2, \ldots n\}$ o conjunto de todas as tarefas, o problema do agendamento pode ser formulado como:

$$\min Q(x) = \sum_{i \in N} E_i[\max(D_{q(i)}, x_{q(i)}) - x_i] + \sum_{i \in F} L_i[\max(x_i - D_i, 0)] \tag{3.9}$$

Observe que, diferentemente de outros problemas de agendamento, o problema formulado por Faaland e Schmitt (1987) busca minimizar ambos os componentes de custos, tanto as penalidades oriundas de atraso na entrega dos produtos acabados, como as penalidades refletidas nos custos de manutenção de estoque para produtos finalizados antecipadamente. A solução proposta por eles, chamada de procedimento de seleção, é composta por duas fases.

A primeira fase produz um grafo de precedência especificando toda estrutura de produção e sequências de expedição, além de um agendamento viável inicial. A segunda fase busca melhorar o agendamento inicial resultante da primeira fase. A fase dois termina com uma solução ótima para o grafo de precedência da fase 1.

Na primeira fase, os autores utilizam uma heurística conhecida como *Minimum Slack Time* (MST) para gerar o grafo de precedência representante da ordem de expedição. De maneira geral, essa heurística busca manter uma maior taxa de ocupação das estações de trabalho enquanto mantém a prioridade na fabricação de produtos com datas de entrega mais próximas.

Utilizando-se da heurística MST, organiza-se a expedição das atividades num diagrama de Gantt e, a partir dele, em um grafo direcionado, $G = (V, A)$. Define-se, ainda, a *rede ativa*, $G^* = (V, A^*)$, onde A^* é o conjunto de arcos (i, j) de A tal que $x_j - p_j = x_i$. A obtenção da rede ativa finaliza a fase 1 do procedimento, fornecendo uma sequência de expedição viável, baseada no menor tempo de finalização de cada tarefa. Definidas a sequência de expedição e a estrutura de produção, o problema do agendamento pode ser reduzido ao seguinte problema de minimização:

$$\begin{aligned} \min \quad & Q(x) \\ s.a. \quad & x_i \geq p_i \quad i \in N, \\ & x_j - p_j \geq x_i \quad (i, j) \in A. \end{aligned} \tag{3.10}$$

Na fase 2 do procedimento, cada iteração verifica se a solução pode ser melhorada pelo aumento do valor de alguma variável. Se essa melhoria for possível, o procedimento seleciona a tarefa ou o conjunto de tarefas cujo aumento reduz os custos das penalidades de forma mais rápida. O problema de selecionar o grupo de variáveis apropriadas no grafo é o problema do conjunto fechado de peso máximo. Portanto, o procedimento de Faaland e Schmitt (1987) resolve o problema do agendamento através de um algoritmo que resolve, a cada iteração, o problema do conjunto fechado de peso máximo.

3.2.2 Otimização de Kits de Reparo

Mamer e Smith (1982) formularam um modelo para resolver o problema de otimização de kits de reparo para equipes de trabalho em campo. O modelo é resolvido pelo problema do conjunto fechado de peso máximo e é baseado na taxa de conclusão de tarefas.

Muitas empresas alugam e/ou mantêm equipamentos em diversos locais geograficamente distantes. Empresas especializadas em manutenção, por exemplo, prestam serviços para diversas indústrias em diferentes cidades. Para realizar reparos e intervenções em máquinas e equipamentos, as equipes de trabalho em campo precisam de várias ferramentas e componentes que, em geral, são carregados como kits de reparo pela equipe de campo.

Para uma determinada chamada de reparo, se todos os elementos necessários estiverem no kit, a equipe de campo poderá realizar o reparo. Porém, se nem todos os elementos necessários estiverem presentes no kit, a tarefa não é completada e a ordem de serviço é marcada como incompleta. Ordens de serviço incompletas, além de causarem perda de confiança na empresa e possivelmente quebra de contratos, são custosas do ponto de vista operacional pois aumentam o *downtime* dos equipamentos, exigem deslocamentos extras da equipe de campo, podem exigir parada na produção ou submeter os equipamentos a danos. Por outro lado, carregar mais itens no kit de reparo com a equipe de campo pode aumentar os custos de manuseio e estoque. Assim, o *problema do kit ótimo* busca minimizar os custos de estoque e manuseio, além dos custos decorrentes de ordens de serviço incompletas (AHUJA; MAGNANTI; ORLIN, 1993).

A equipe de campo pode reabastecer os kits de reparos entre tarefas, mas seus itens especificados são fixos. Assim, uma tarefa representa um conjunto de componentes e ferramentas que são necessários antes que o kit possa ser reabastecido. Do ponto de vista do estoque, uma tarefa é um conjunto de componentes e ferramentas de maneira que tarefas que necessitam dos mesmos itens (componentes e ferramentas) são de um mesmo tipo.

Portanto, define-se um conjunto de m tipos de tarefas $J_1, J_2, \ldots, J_m$, que representam todas as possíveis tarefas encontradas pela equipe de campo. De maneira semelhante, existem n, $(1, 2, \ldots, n)$, tipos de itens disponíveis para a equipe. Observe que uma determinada tarefa J_i é definida pelo conjunto B_i de itens necessários para realizar a tarefa J_i. Um kit de reparo consiste do conjunto fixo de itens $M \subseteq \{1, 2, \ldots, n\}$ que será carregado pela equipe de reparo. No contexto do estoque, o conteúdo do kit é sempre M, de forma que as ferramentas, itens que podem ser reutilizados, podem ser incluídas na análise.

Seja H_i o custo de estoque anual para o item i. O custo anual de estoque do kit M é dado por $\sum_{i \in M} H_i$. Com o kit M, todos as tarefas tais que $B_k \subseteq M$ podem ser realizadas e todas as demais serão ordens de serviço incompletas. Para uma tarefa não concluída do tipo J_k, a empresa absorve um custo V_k atribuído às penalidades decorrentes da não conclusão da tarefa.

Para calcular o custo esperado anual associado ao kit M, determina-se o número esperado de tarefas de cada tipo que seriam realizadas durante um ano por um único kit de reparo. Denota-se o número de tarefas do tipo j pela variável aleatória $N_j, j = 1, \ldots, m$. Dessa forma, o custo total anual esperado com as penalidades por kit por ano para o kit M é:

$$\sum_{\{j|B_j \nsubseteq M\}} E[N_j]V_j = \sum_{\{j|B_j \nsubseteq M\}} V_j \lambda_j,$$

onde λ_j é o número esperado de tarefas do tipo j realizado pelo kit durante um ano. O custo esperado total anual por kit para um determinado kit M é:

$$C(M) = \sum_{i \in M} H_i + \sum_{\{j|B_j \nsubseteq M\}} V_j \lambda_j.$$

O problema do kit ótimo, portanto, tem como objetivo selecionar/compor um kit M que minimize $C(M)$. Com essa formulação, para um determinado kit M, um determinado tipo de tarefa poderá ser sempre executada ou nunca poderá ser executada e resultará sempre em ordem de serviço incompleta. A única incerteza envolvida no modelo diz respeito a quais tipos de tarefas serão exigidas em uma determinada ordem de serviço. Observe que, dessa forma, será necessário apenas somar as tarefas que resultarão em ordens de serviço incompletas para determinar o custo esperado das penalidades. Além disso, somente o número de ocorrências λ_j precisa ser estimado para determinação de $C(M)$. Essa estimação pode ser feita com base nos arquivos das operações da empresa (MAMER; SMITH, 1982).

Percebe-se que o problema do kit ótimo pode ser resolvido como um problema de seleção, ou seja, um problema do conjunto fechado de peso máximo em um grafo bipartido. Seja $L_i = V_i \lambda_i$ e considere o grafo da figura (18).

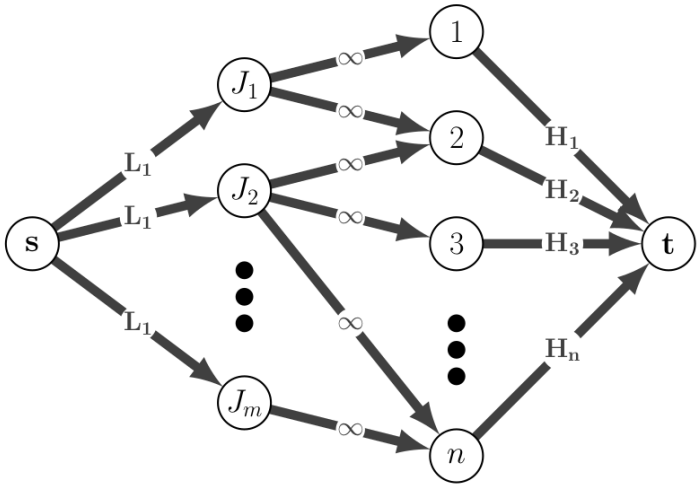
Os vértices $J_1, J_2, \ldots, J_m$ correspondem às m tarefas e os vértices numerados de 1 a n representam os n tipos de itens diferentes, incluindo as ferramentas. Se o item i é necessário para realização da tarefa J_i, existe um arco (J_i, i) no grafo. L_j, $j = 1, 2, \ldots, m$, representa os custos associados a penalidades por ordens de serviço incompletas e H_i, $i = 1, 2, \ldots, n$, representa o custo de estoque.

Portanto, a resolução do problema do conjunto fechado de peso máximo no grafo da figura (18) resolve o problema do kit ótimo fornecendo um conjunto de tarefas, bem como os itens necessários para realização dessas tarefas, que minimiza o custo anual por kit de reparo.

3.2.3 Outras Aplicações

O problema do conjunto fechado de peso máximo encontra aplicações em diversas outras áreas. Ahuja, Magnanti e Orlin (1993) discutem, além do problema do kit ótimo, o problema da

Figura 18 – Problema do Kit Ótimo



Adaptado de Mamer e Smith (1982)

destruição ótima de alvos militares com proteção em camadas e um problema de *data scaling*, ambos usando problema do fechado máximo.

Reiter (1963), antes dos trabalhos de Rhys (1970), Balinski (1970) e Picard (1976), observou a necessidade de um algoritmo mais eficiente do que algoritmos exaustivos em problemas de composição de portfólio com depedência entre projetos. O problema do conjunto fechado de peso máximo pode preencher essa lacuna.

Eisner e Severance (1976) utilizam o problema do fechado máximo num problema de segmentação de arquivos em grandes bases de dados, tornando mais eficiente o tratamento das informações. McGinnis e Nuttle (1978) utilizam o problema do fechado máximo para resolver problemas de gerenciamento de projetos.

A doutora Dorit Hochbaum, da Universidade da Califórnia em Berkeley, tem formulado diversas aplicações do problema do conjunto fechado de peso máximo desde os anos 1990. Algumas dessas aplicações são mostradas na tabela (5) a seguir.

Tabela 5 – Aplicações do Problema do Fechado Máximo

Problema	**Aplicação**	**Referência**
Convex Closure	Estimação Bayesiana	Hochbaum e Queyranne (2003)
Convex s-excess	Segmentação de Imagens	Hochbaum e Singh (2009)
Monot. Int. Program. (IPM)	Cobertura Grafos Bipart.	Hochbaum e Naor (1994)
(Nonmonotone) IP2	Max-Clique, 2-SAT	Hochbaum et al. (1994)

Fonte: Adaptado de Hochbaum (2004)

Os estudos da doutora Hochbaum permitiram que ela desenvolvesse um algoritmo rápido e robusto para resolver o problema do conjunto fechado de peso máximo. O algoritmo resolve também problemas de fluxo máximo. Este algoritmo é apresentado no capítulo 5.

Na sequência, considerando a importância do problema do corte mínimo (e, consequentemente, do fluxo máximo) para a formulação e resolução do problema do fechado máximo, o próximo capítulo se dedica a apresentar e brevemente discutir o teorema do fluxo máximo e corte mínimo (Max-Flow Min-Cut, Ford e Fulkerson (1956) e (1962)), bem como os principais algoritmos para resolver estes problemas.

4 FLUXO MÁXIMO

O problema do fluxo máximo, que se tornou amplamente conhecido a partir dos trabalhos de Ford e Fulkerson (1956) e (1962), tem sua origem nos estudos do topólogo Karl Menger sobre curvas nas décadas de 1920 e 1930. Em 1927, Menger publicou um trabalho contendo o seguinte teorema (SCHRIJVER, 1993; SCHRIJVER, 2005):

Teorema 4.1 (Teorema β)**.** *Se K é um espaço unidimensional compacto e regular o qual é n-pontos conexo entre dois conjuntos finitos P e Q, então K contém n curvas disjuntas, cada uma das quais conecta um ponto em P a um ponto em Q.*

Este teorema é uma espécie de precurssor do teorema do fluxo máximo e corte mínimo e pode ser formulado em um grafo. Seja $G = (V, E)$ um grafo (não direcionado) e sejam $P, Q \subseteq V$. Então, o número máximo de caminhos $P - Q$ disjuntos é igual à mínima cardinalidade de um conjunto W de vértices tal que cada caminho $P - Q$ intercepta W (SCHRIJVER, 2005).

O problema estudado por Ford e Fulkerson (1956) era de cunho prático e pode ser descrito da seguinte forma: considere uma rede ferroviária conectando duas cidades através de diversas cidades intermediárias, onde cada link da rede possui uma capacidade; encontre o fluxo máximo de uma das cidades à outra. A descrição do problema do fluxo máximo apresentada por Ford e Fulkerson, apesar de não ter uma relação direta com as curvas estudadas por Menger, se baseia na mesma estrutura matemática.

No seu livro, Ford e Fulkerson (1962) comentam que o problema de encontrar a capacidade máxima de uma rede ferroviária lhes foi proposto por T.E. Harris juntamente com o general F.S. Ross. Atualmente, sabe-se que as motivações e os detalhes por trás do desenvolvimento deste trabalho são mais interessantes do que se supunha, conforme detalhado em Schrijver (2002).

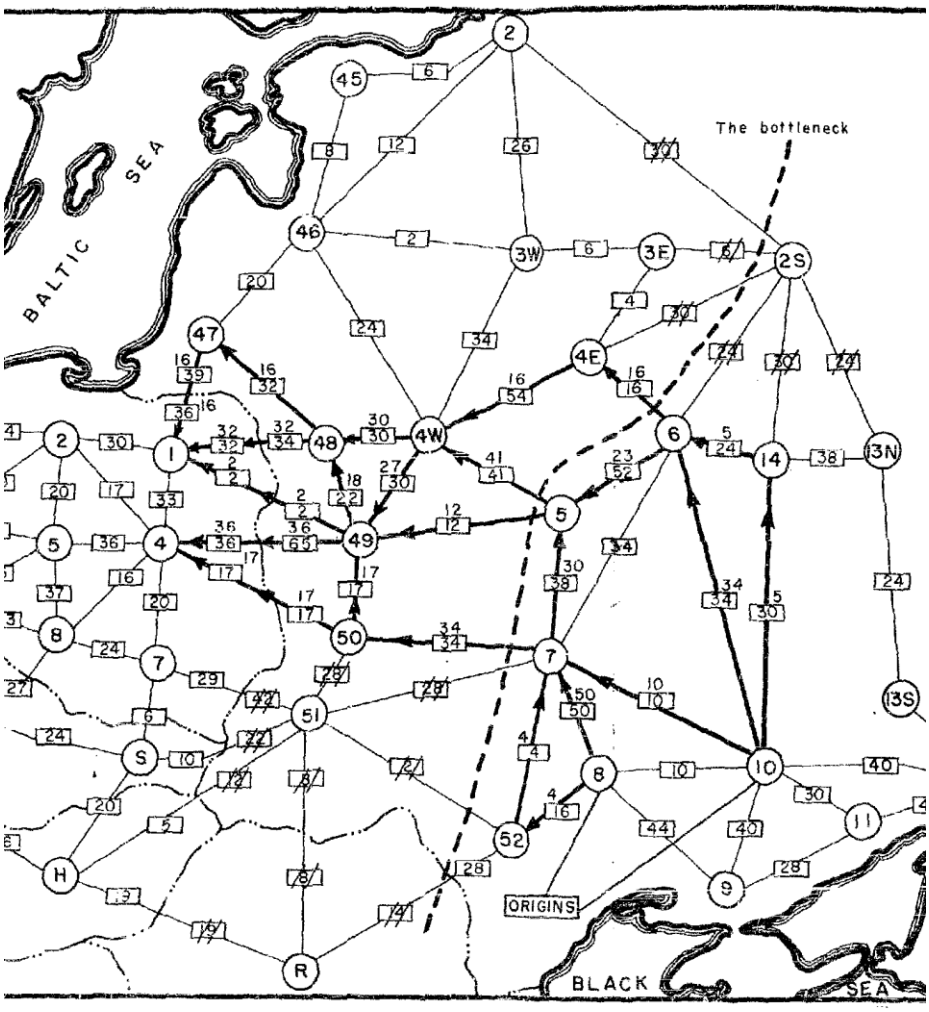
O trabalho seminal de Ford e Fulkerson (1956), "*Maximal Flow Through a Network*", foi originalmente publicado como um relatório secreto da agência *RAND Corporation* para a força aérea americana. O relatório, intitulado *Fundamentals of a Method for Evaluating Rail Net Capacities* (HARRIS; ROSS, 1955) foi desclassificado pelo Pentágono em 1999 através de solicitação feita por Alexander Schrijver.

A rede ferroviária em questão era o sistema ferroviário soviético. O relatório de Harris e Ross apresenta a resolução de um problema de fluxo máximo de larga escala referente a uma rede ferroviária ligando a porção ocidental da União Soviética e os países da Europa oriental. A motivação real do problema, no entanto, não era a obtenção do fluxo máximo, mas sim do corte mínimo, que seria o melhor ponto de interdição do sistema (BILLERA; LUCAS, 1976; HARRIS; ROSS, 1955; SCHRIJVER, 2002).

A imagem (19) a seguir foi publicada no relatório e mostra parte de um diagrama esquemático da rede ferroviária entre a União Soviética oriental e as cidades satélites da Europa

ocidental. O corte mínimo é destacado com uma linha tracejada e a designação "*The bottleneck*", o gargalo em tradução livre, o que seria o melhor conjunto de pontos para ataques aéreos por parte da força aérea americana no sentido de interditar a malha.

Figura 19 – Diagrama da malha ferroviária da União Soviética ocidental



Fonte: Adaptado do relatório de Harris e Ross (1955)

Resultados fundamentais em fluxos em redes são o teorema Max-Flow Min-Cut e o algoritmo do caminho de aumento para encontrar o fluxo máximo, ambos propostos por Ford e Fulkerson. Mais tarde, o trabalho de Goldberg e Tarjan (1988) ofereceu um algoritmo especialmente eficiente em teoria e prática. Trabalhos como os de Dinic (1970) e Edmonds e Karp (1972) também foram fundamentais para o desenvolvimento de melhores algoritmos.

4.1 Teorema Max-Flow Min-Cut

Uma rede é um digrafo $D = (V, A)$ com uma fonte s e um sorvedouro t, juntamente com uma função capacidade de valores reais e não negativos, c, definida nos arcos de A.

Representamos o valor da função capacidade em cada arco a por $c(a)$ ou c_a. Define-se um s-t fluxo da seguinte forma:

Definição 4.1. Uma função $f : A \to \mathbb{R}$ é um fluxo de s para t, ou simplesmente, um s-t fluxo, se:

i. $f(a) \geq 0$ para cada $a \in A$;

ii. $f(\partial^+(v)) = f(\partial^-(v))$ para todo $v \in V \setminus \{s, t\}$.

A condição (i) estabelece que o fluxo é uma função de valores não negativos. A condição (ii) é a chamada lei da conservação do fluxo e estabelece que o fluxo que sai de um vértice v, distinto de s e t, deve ser igual ao fluxo que entra nesse vértice. Representamos o valor do fluxo em um arco a por $f(a)$ ou f_a. O valor do fluxo na rede em qualquer momento é definido como:

Definição 4.2. $value(f) := f(\partial^+(s)) - f(\partial^-(s))$.

Em virtude da lei da conservação do fluxo e das definições de fonte e sorvedouro, o fluxo que sai da fonte se conserva em todos os vértices da rede até chegar em t, portanto:

$$value(f) := f(\partial^+(s)) = f(\partial^-(t))$$

Seja $c : A \to \mathbb{R}_+$ a função capacidade. Um fluxo é dito viável se estiver sujeito à função c, ou seja, se satisfaz à restrição de capacidade dada por

$$f(a) \leq c(a), \qquad \forall\, a \in A.$$

Toda rede apresenta, no mínimo, um fluxo, o fluxo zero, $f(a) := 0$ para todo $a \in A$. Nesses termos, um s-t fluxo máximo, ou simplesmente, um fluxo máximo, é um fluxo de máximo valor sujeito à restrição de capacidade e o problema do fluxo máximo busca encontrar esse valor. Por compacidade e continuidade, um fluxo máximo sempre existe.

É conveniente admitir capacidade infinita em alguns arcos em algumas situações. Obviamente, se todos os arcos possuem capacidade infinita, fluxos arbitrariamente grandes são possíveis. Na prática, no entanto, isto não é comum. Nas aplicações, capacidades infinitas são atribuídas a alguns arcos para garantir que uma determinada relação se mantenha, como, por exemplo, para garantir que vértices que possuem uma dependência se mantenham em um mesmo subconjunto de uma partição. Além disso, uma capacidade infinita pode ser substituída por um valor finito alto, geralmente qualquer valor maior que a soma das capacidades dos outros arcos. Portanto, neste trabalho admite-se que todas as redes possuem um fluxo máximo finito.

Em relação a qualquer função f, define-se uma função excesso em relação à coleção de todos os subconjuntos de V. Para qualquer $f : A \to \mathbb{R}$, a função excesso é definida por:

Definição 4.3. $excess_f : \mathcal{P}(V) \to \mathbb{R}$ tal que

$$excess_f(U) := f(\partial^-(U)) - f(\partial^+(U))$$

para $U \subseteq V$. Além disso, $excess_f(v) := excess_f(\{v\})$ para $v \in V$.

Dessa forma, tem-se o seguinte teorema:

Teorema 4.2. *Seja $D = (V, A)$ um digrafo e $f : A \to \mathbb{R}$ a função fluxo. Além disso, $U \subseteq V$. Então*

$$excess_f(U) = \sum_{v \in U} excess_f(v). \tag{4.1}$$

Demonstração. A prova segue diretamente pela contagem das multiplicidades de $f(a)$ nos dois lados da equação 4.1, para cada $a \in A$. Duas situações são possíveis: (1) os vértices em U são disjuntos, (2) alguns dos vértices em U são adjacentes. Em (1) o termo da direita da equação 4.1 é a própria definição do termo da esquerda, portanto igual. No caso (2), os arcos que ligam dois vértices de U não contribuem para a soma líquida do termo da direita pois somam em um vértice e subtraem no outro. □

A capacidade de um corte é definida por $c(\partial^+(U))$, ou seja, a capacidade de todos os arcos saindo do subconjunto U. O valor do fluxo é sempre menor ou igual a capacidade de um corte. A formalização dessa observação segue na forma do teorema a seguir.

Teorema 4.3. *Seja $D = (V, A)$ uma rede com $c : A \to \mathbb{R}_+$. Então*

$$value(f) \leq c(\partial^+(U)) \tag{4.2}$$

para cada s-t fluxo $f < c$ e cada s-t corte $\partial^+(U)$. Além disso, $value(f) = c(\partial^+(U))$ se e somente se $f(a) = c(a)$ para cada $a \in \partial^+(U)$ e $f(a) = 0$ para cada $a \in \partial^-(U)$.

Demonstração. Observe que $value(f) = -excess_f(s)$ e que $excess_f(U) = excess_f(s)$ uma vez que $s \in U$ e $excess_f(v) = 0$ para $v \neq s$. Portanto

$$value(f) = -excess_f(s) = -excess_f(U) = f(\partial^+(U)) - f(\partial^-(U)) \leq c(\partial^+(U)).$$

Como $f(\partial^-(U)) \geq 0$, tem-se igualdade na equação 4.2 se e somente se $f(\partial^+(U)) = c(\partial^+(U))$ e $f(\partial^-(U)) = 0$. □

Considere um digrafo $D = (V, A)$. Para cada arco $a = (u, v)$ de D, o arco reverso é definido como $a^{-1} = (v, u)$ e o conjunto de todos os arcos reversos pode ser definido como $A^{-1} := \{a^{-1} \mid a \in A\}$. Em problemas de fluxo, os arcos reversos surgem sempre que um fluxo é enviado através de um arco.

Estabelecendo-se uma função para o limite inferior, $l : A \to \mathbb{R}$, do fluxo em cada arco e considerando $c : A \to \mathbb{R}$, como um limite superior para o fluxo em cada arco, tem-se que, para qualquer $f : A \to \mathbb{R}$ satisfazendo $l \leq f \leq c$, define-se o conjunto dos arcos residuais.

Definição 4.4. O conjunto dos arcos residuais com relação ao fluxo f é definido por:

$$A_f := \{a \mid a \in A, f(a) < c(a)\} \cup \{a^{-1} \mid a \in A, f(a) > l(a)\}.$$

Portanto, o conjunto dos arcos residuais é composto pelos arcos originais cujo fluxo é inferior à sua capacidade, chamados arcos diretos, e por um arco reverso para cada arco original onde houver fluxo maior que o limite inferior. A_f depende de f, D, l e c, mas, frequentemente, D, l e c são fixados nas aplicações enquanto que o fluxo varia até que seja atingido seu máximo valor. No contexto de fluxos, $l = 0$ (SCHRIJVER, 2003). Define-se, a seguir, o conceito de grafo residual com relação ao fluxo f, D_f:

Definição 4.5. $D_f := (V, A_f)$.

Dessa forma, o grafo residual em relação a f é formado pelo conjunto de vértices do grafo original e pelo conjunto dos arcos residuais com relação a f. O grafo residual é muito útil no estudo de fluxos pois é nele que se verificam as possibilidades de aumento do fluxo total em cada iteração. No grafo residual é que se estabelece a condição de otimalidade do fluxo:

Corolário 4.3.1. *Seja f um s-t fluxo em D, com $f \leq c$. Suponha que não exista nenhum s-t caminho em D_f. Seja U o conjunto de vértices em D_f alcancáveis a partir de s.*
Então $value(f) = c(\partial^+(U))$. Em particular, f tem valor máximo.

Demonstração. Aplicando o teorema 4.3, para cada $a \in \partial^+(U)$, $a \notin A_f$, portanto $f(a) = c(a)$.
Da mesma forma, para cada $a \in \partial^-(U)$, $a^{-1} \notin A_f$ e portanto $f(a) = 0$.
Logo, $value(f) = c(\partial^+(U))$ e f tem um valor máximo pelo teorema 4.3. □

Retomando-se a definição de caminho em um grafo da seção 2.1, define-se um caminho direcionado como uma orientação de um caminho na qual cada vértice domina seu sucessor na sequência. Qualquer caminho direcionado em D_f fornece um caminho não direcionado em $D = (V, A)$. Considere P como sendo um caminho partindo de s, não necessariamente direcionado. A cada P associa-se um $\epsilon(P)$ definido como:

$$\epsilon(P) := \ min\{\epsilon(a) \mid a \in A(P)\}$$

onde

$$\epsilon(a) = \begin{cases} c(a) - f(a) & \text{se } a \text{ é um arco direto de } P, \\ f(a) & \text{se } a \text{ é um arco reverso de } P. \end{cases}$$

Pela formulação de $\epsilon(P)$, pode-se pensar nele como a "folga" do caminho P, pois é o máximo valor pelo qual o fluxo pode ser aumentado através de P, respeitando-se a restrição de capacidade. Observe que para ambos os casos, arcos diretos ou arcos reversos, $\epsilon(a)$ representa a capacidade residual no arco a. Dessa forma, com relação a um fluxo f, tem-se as seguintes denominações para um caminho P qualquer:

- *saturado:* se $\epsilon(P) = 0$;
- *insaturado:* se $\epsilon(P) > 0$.

Um caminho insaturado, portanto, apresenta "folga" em todos os seus arcos, isto é, cada arco direto de P é insaturado e cada arco reverso de P apresenta um fluxo maior que 0. Claramente, um caminho insaturado é um caminho que não teve toda sua capacidade utilizada. Denomina-se caminho de aumento a um s-t caminho insaturado. Através de um caminho de aumento P_{st} é possível aumentar o fluxo global a partir de um novo fluxo $f' : A \to \mathbb{R}$ tal que:

$$f' := \begin{cases} f(a) + \epsilon(P_{st}) & \text{se } a \text{ é um arco direto de } P_{st}, \\ f(a) - \epsilon(P_{st}) & \text{se } a \text{ é um arco reverso de } P_{st}, \\ f(a) & \text{se } a \text{ não é um arco de } P_{st}. \end{cases} \tag{4.3}$$

Portanto, a existência de um caminho de aumento garante que o fluxo não é máximo. Mais precisamente, o fluxo global pode ser incrementado através da Equação 4.3 por uma quantidade $\epsilon(P_{st})$. Daí segue o teorema de Ford e Fulkerson (1956) e (1962):

Teorema 4.4 (Max-Flow Min-Cut)**.** *Seja $D = (V, A)$ uma rede e seja $c : A \to \mathbb{R}_+$ a função capacidade. Então o valor máximo de um s-t fluxo sujeito à c é igual à mínima capacidade de um s-t corte.*

Demonstração. Seja f um s-t fluxo máximo sujeito à c. Pelo teorema 4.3, será suficiente mostrar que existe um s-t corte, $\partial^+(U)$, com capacidade igual a $value(f)$. Considere o grafo residual D_f e suponha que ele contém um s-t caminho, P_{st}. Então, f' conforme definido na equação 4.3, é também um s-t fluxo sujeito à c com $value(f') = value(f) + \epsilon(P_{st})$, contradizendo a maximalidade de $value(f)$. Portanto, D_f não contém s-t caminho. Seja U o conjunto de vértices em D_f alcancáveis a partir de s. Então, pelo corolário 4.3.1, $value(f) = c(\partial^+(U))$. □

4.2 Algoritmo do Caminho de Aumento

A prova do teorema Max-Flow Min-Cut de Ford e Fulkerson fornece um algoritmo para resolução de problemas de fluxo máximo. Esse algoritmo consiste em encontrar caminhos de aumento no grafo residual e incrementar $value(f)$ de $\epsilon(P_{st})$ a cada iteração. Naturalmente, a condição de parada do algoritmo é a não existência de caminhos de aumento no grafo residual.

Considere $D = (V, A)$ uma rede com $c : A \to \mathbb{Q}_+$, D_f o grafo residual com relação ao fluxo f e P_{st} um s-t caminho direcionado em D_f. O pseudocódigo do algoritmo de caminho de aumento é apresentado na figura (20). O laço (**while**) do algoritmo é usado para construir um caminho de aumento, ou seja, um s-t caminho direcionado no grafo residual D_f. Um vez que um caminho de aumento é verificado, calcula-se a folga do caminho, incrementa-se o fluxo atual através do procedimento apresentado na Equação 4.3 e o algoritmo volta ao passo 1. O algoritmo termina quando $t \notin U$, ou seja, quando o algoritmo não consegue incluir t no conjunto da fonte, indicando que não é mais possível construir um caminho de aumento.

O corolário e o teorema a seguir completam a exposição dos aspectos conceituais do teorema de Ford e Fulkerson.

Figura 20 – Algoritmo caminho de aumento de Ford e Fulkerson

Procedure augmentingPath(D, f)

input : Uma rede D e um fluxo viável f
output : Um fluxo máximo f e um corte mínimo $\partial^+(U)$

1 Inicialize $U := \{s\}$, $p(v) := \emptyset$, $v \in V$;
2 **while** *existe um arco insaturado* $a = (u, v)$ *ou* $a^{-1} = (v, u)$*, com* $u \in U$ *e* $v \in V \setminus U$ **do**
3 substitua U por $U \cup \{v\}$;
4 substitua $p(v)$ por u;
5 **end**
6 **if** $t \in U$ **then**
7 compute $\epsilon(P_{st})$, onde P_{st} é o caminho s-t na árvore cuja função predecessora é p
8 para cada arco direto a de P_{st}, substitua $f(a)$ por $f(a) + \epsilon(P_{st})$;
9 para cada arco reverso a^{-1} de P_{st}, substitua $f(a)$ por $f(a) - \epsilon(P_{st})$;
10 retorne para o passo 1;
11 **end**
12 **return** *(*f*,* $\partial^+(U)$*)*

Fonte: Adaptado de Bondy e Murty (2008)

Corolário 4.4.1 (Teorema da Integridade)**.** *Se* c *é inteira, existe um fluxo máximo inteiro.*

Demonstração. A prova desse corolário segue diretamente da prova do teorema Max-Flow Min-Cut utilizando-se $\epsilon(P_{st}) = 1$. □

O teorema da integridade (FORD; FULKERSON, 1962), garante que para um problema de fluxo máximo cuja função capacidade apresenta valores inteiros, o fluxo máximo é também inteiro. O teorema a seguir garante que se as capacidades dos arcos forem racionais, o algoritmo do caminho de aumento é finito (SCHRIJVER, 2003).

Teorema 4.5. *Se todas as capacidade* $c(a)$ *forem racionais, o algoritmo do caminho de aumento termina.*

Demonstração. Se todas as capacidades forem racionais, existe um número natural K tal que $Kc(a)$ é um inteiro para cada $a \in A$. K pode ser o mínimo múltiplo comum dos denominadores de $c(a)$, por exemplo. Portanto, em cada iteração, cada $f_i(a)$ e cada $\epsilon(P_{st})$ é um múltiplo de $1/K$. Então, em cada iteração, $value(f)$ aumenta de pelo menos $1/K$. Como $value(f)$ não pode ultrapassar o valor de $c(\partial^+(s))$, existe uma quantidade finita de iterações. □

Dantzig e Fulkerson (1956) formularam o problema Max-Flow como um problema de programação linear e deduziram o teorema Max-Flow Min-Cut a partir da dualidade do problema. Ainda neste trabalho, estes autores observaram que uma forma combinatorial do teorema Max-Flow Min-Cut é equivalente ao teorema de Menger sobre grafos lineares.

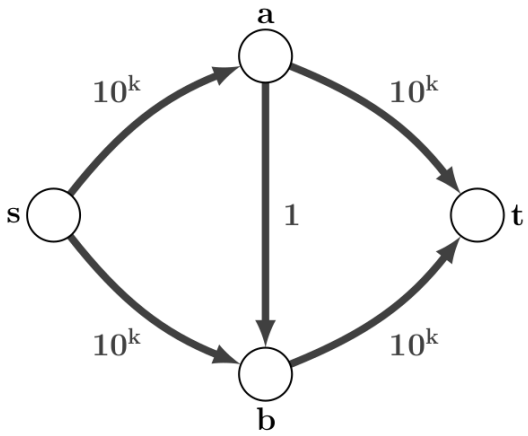
Dantzig, Ford e Fulkerson (1956) foram os primeiros a descrever um algoritmo primal-dual para problemas gerais de programação linear, o que representou um avanço importante. O método primal-dual aplicado ao problema do caminho mínimo levou ao desenvolvimento do

algoritmo de Dijkstra, e aplicado ao problema do fluxo máximo levou ao desenvolvimento do algoritmo de Ford e Fulkerson (PAPADIMITRIOU; STEIGLITZ, 1998).

Como explicam Papadimitriou e Steiglitz (1998), o desenvolvimento desses algoritmos representa um ponto de transição no estudo de algoritmos gerais de programação linear para algoritmos especializados, com um foco mais combinatório. Contudo, o algoritmo inicial de Ford e Fulkerson apresentava uma limitação importante: para capacidades racionais, mesmo sendo finito, o algoritmo pode levar um tempo de processamento demasiado elevado dependendo das capacidades dos arcos do grafo e da escolha do caminho de aumento no grafo residual.

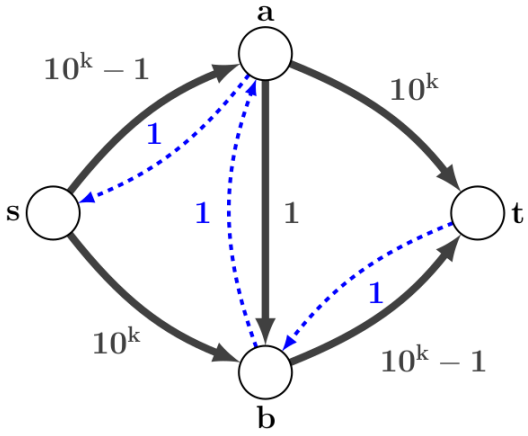
Considere a rede da figura (21) e suponha que, através do algoritmo do caminho de aumento, escolhe-se o caminho $P_{st} = [s, a, b, t]$ como o caminho de aumento da iteração, onde os números próximos aos arcos representam as capacidades dos arcos e k é uma constante.

Figura 21 – Exemplo de Rede



Fonte: Adaptado de Schrijver (2003)

Figura 22 – Grafo Residual



Fonte: O autor (2018)

Para esse caso, $\epsilon(P_{st}) = min\{10^k, 1, 10^k\} = 1$ e o grafo residual resultante é o apresentado na figura (22) a seguir. Para o próximo caminho de aumento, uma escolha possível, apesar de improvável, poderia ser $P_{st} = [s, b, a, t]$, de forma que, seguindo esse padrão, o algoritmo realizaria 2.10^k iterações, ou seja, um tempo exponencial em relação ao tamanho da entrada, $O(k)$ (SCHRIJVER, 2003).

Porém, se o algoritmo tomar sempre o menor caminho, em termos de número de arcos, o número de iterações é de, no máximo, nm (EDMONDS; KARP, 1972). Combinando esse resultado com o fato de que um caminho mínimo pode ser encontrado em $O(m)$, o problema do fluxo máximo pode ser resolvido pelo algoritmo de Ford e Fulkerson em $O(nm^2)$ (SCHRIJVER, 2003).

4.3 Heurística de Boldyreff - *The Flooding Technique*

Ainda no bojo do projeto de Harris e Ross, na *Rand Corporation*, Boldyreff (1955b) apresentou uma heurística para o cálculo do fluxo máximo. As motivações para esse estudo eram as mesmas, a rede ferroviária da união soviética. Contudo, Boldyreff focou em resolver o problema satisfatoriamente mesmo que a otimalidade não fosse garantida, que seu método pudesse ser entendido e utilizado por pessoal sem alto nível de especialização e que não fossem necessários super-computadores para resolver uma instância realista do problema. De fato, a heurística de Boldyreff foi o método escolhido para resolver o problema apresentado no relatório de Harris e Ross (1955), como mostra um outro relatório interno da *Rand Corporation*, Boldyreff (1955a), desclassificado muito tempo depois.

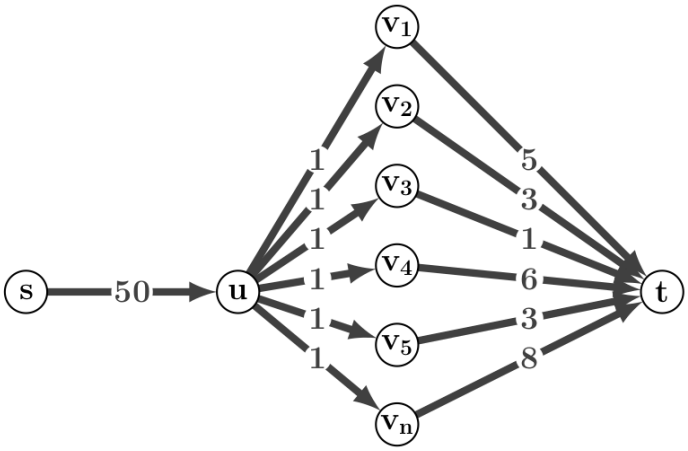
Boldyreff (1955b) inicia seu artigo sugerindo simplificações que eventualmente reduzem consideravelmente a complexidade de certas redes, para depois apresentar sua *flooding technique*. O conceito por trás desse método é, de fato, simples: a ideia central é "inundar" sucessivamente os arcos a cada nível da rede, o primeiro nível sendo a fonte s. Os gargalos, onde vértices recebem um fluxo maior do que o que podem enviar, são eliminados "devolvendo-se" o fluxo excessivo para a origem. O método é empírico, não se utiliza de nenhum tipo de mecanismo *backtrack* e não necessariamente conduz a uma solução ótima (SCHRIJVER, 2003).

Ainda assim, a heurística de Boldyreff fornece uma ideia importante na busca por um algoritmo mais eficiente que o do caminho de aumento. Deixando-se de lado a restrição de viabilidade por um momento, é possível avançar mais rapidamente na construção do fluxo máximo. Considere o digrafo da figura (23) a seguir.

Nesse exemplo, fica evidente que a manutenção de um fluxo viável a todo momento torna o algoritmo lento pois ele terá que retornar à fonte para construir cada novo caminho de aumento. Em cada caminho de aumento, o algoritmo enviará um fluxo de apenas uma unidade através do arco (s, u) para manter a viabilidade do fluxo.

A técnica de Boldyreff, ou qualquer algoritmo que relaxe a restrição de viabilidade, procederia mais rapidamente pois permitiria enviar um fluxo de 50 unidades através de (s, u) e retornaria o fluxo excedente ao final da execução.

Figura 23 – Digrafo D



Fonte: O autor (2018)

Goldberg e Tarjan (1988) utilizaram essa ideia de relaxar a viabilidade do fluxo para desenvolver um dos mais rápidos algoritmos para o fluxo máximo. De fato, o algoritmo *Push-Relabel* é usado para comparação de eficiência de algoritmos para o fluxo máximo por ser muito rápido em teoria e prática.

4.4 Algoritmo Push-Relabel (Preflow)

Goldberg e Tarjan (1988) propuseram um dos mais rápidos algoritmos para o fluxo máximo, tanto no sentido teórico quanto no sentido prático. O algoritmo é baseado em *prefluxo* (*preflow*), o qual é semelhante a um fluxo conforme definido anteriormente, mas permite que o fluxo que chega a um vértice seja maior do que o fluxo que sai do vértice, gerando um excesso de fluxo nesse vértice.

O algoritmo de Goldberg e Tarjan mantém um prefluxo na rede original e envia o excesso de fluxo local para o sorvedouro através do que o algoritmo estima ser um caminho mínimo no grafo residual. Os excessos que não podem ser enviados ao sorvedouro, são retornados para a fonte e apenas quando o algoritmo termina é que o prefluxo se torna um fluxo, o fluxo máximo da rede.

Schrijver (2003) explica que esse algoritmo pode ser pensado como um algoritmo dual em relação a algoritmos baseados em fluxo, no sentido que esses últimos mantêm um fluxo viável e aumentam esse fluxo a cada iteração até que não haja mais nenhum caminho de aumento no grafo residual, enquanto que o primeiro atualiza um prefluxo a cada iteração, mantendo a propriedade de que não haja nenhum caminho de aumento no grafo residual, até que o prefluxo se torne um fluxo. Isto significa que o prefluxo é possivelmente inviável, mas sempre superótimo.

Um prefluxo pode ser definido formalmente da seguinte forma. Seja $D = (V, A)$ uma rede com $c : A \to \mathbb{Q}_+$ (SCHRIJVER, 2003):

Definição 4.6. Uma função $f : A \to \mathbb{Q}$ é chamado um prefluxo, se:

(i) $0 \leq f(a) \leq c(a)$ para todo $a \in A$,

(ii) $excess_f(v) \geq 0$ para todo vértice $v \neq s$.

O algoritmo *push-relabel* verifica, para cada vértice v diferente de s e t com excesso maior que zero, $excess_f(v) > 0$, se é possível enviar uma porção desse excesso para vértices mais próximos de t. A ideia é enviar o máximo possível desse excesso. Se t não puder ser alcançado por um vértice com excesso positivo, então o algoritmo envia esse fluxo para vértices mais próximos de s. Ao final, o algoritmo chega a um estado no qual todos os vértices v diferentes de s e t apresentam $excess_f(v) = 0$. Nesse ponto, a restrição de viabilidade é restabelecida e o prefluxo se torna um fluxo, o fluxo máximo.

Como o algoritmo é baseado em duas operações centrais, *push* e *relabel*, ele é conhecido como algoritmo *push-relabel*. As definições a seguir são necessárias para a exposição do método *push-relabel*.

Definição 4.7. A capacidade residual de uma arco (u, v) é definida como:

- $c^f_{uv} = c_{uv} - f_{uv}$ para um arco direto $(u, v) \in A$;
- $c^f_{vu} = f_{uv}$ para um arco reverso (v, u) tal que $(u, v) \in A$.

Se o vértice u tem excesso positivo e o par (u, v) apresenta capacidade residual positiva, então uma quantidade desse excesso de fluxo, defnida por $\delta = min\{excess_f(u), c^f(u, v)\}$, pode ser enviada de u para v adicionando-se δ a $f(u, v)$ e subtraindo-se δ de $f(v, u)$. Existem duas formas de o par (u, v) apresentar capacidade residual positiva: (1) (u, v) é um arco com fluxo menor que sua capacidade; (2) (v, u) é um arco com fluxo maior que zero.

Definição 4.8. Seja d uma função $d : V \to \mathbb{Z}_+$ tal que:

(i) $d(s) = n$,

(ii) $d(t) = 0$,

(iii) $d(u) \leq d(v) + 1$ para cada $(u, v) \in A_f$.

A função d é criada para estimar a distância de um vértice qualquer para s ou para t. Se $d(v) < n$, $d(v)$ é um limite inferior para a distância real entre v e t no grafo residual, D_f, e se $d(v) \geq n$, então $d(v) - n$ é um limite inferior para a distância entre v e s em D_f. Observe que, conforme esta definição, uma função d só existe quando não existe caminho s-t em D_f. Assim, se uma função d (conforme definição 4.8) existe e se f é um s-t fluxo, então D_f não possui caminho de aumento e, portanto, f é máximo (SCHRIJVER, 2003). Além disso, os vértices ativos desempenham papel crucial no algoritmo.

Definição 4.9. Um vértice v é um vértice *ativo* se:

Figura 24 – Algoritmo Push-Relabel

Procedure pushRelabel(D)

input : Uma rede D
output : Um fluxo máximo f

«Pré-processamento»
$f = 0$
$d(s) = n$, $d(i) = 0, \forall\, i \neq s$
$\forall\, (s, v) \in A$, $f_{sv} = c_{sv}$, $excess_f(v) = c_{sv}$
while *A rede contém um vértice ativo* **do**
    begin
        /* Push */
        Selecione um vértice ativo u
        while $\exists$ *um arco admissível* $a = (u, v)$ **do**
            Seja $\delta = \min\{excess_f(u), c^f(u, v)\}$, onde (u, v) é um arco admissível
            Envie δ unidades de fluxo de u para v
        end
        else
            /* Relabel */
            $d(u) = min_{(u,v)\in A_f}\{d(v) + 1\}$
        end
    end
end
return *(f)*

Fonte: Adaptado de Goldberg e Tarjan (1988)

(i) $v \in V \setminus \{s, t\}$,

(ii) $d(v) < \infty$, e

(iii) $excess_f(v) > 0$.

A versão geral do algoritmo *push-relabel* é apresentada na figura (24). Nessa versão, o algoritmo não calcula o corte mínimo. Goldberg e Tarjan (1988), no entanto, explicam que basta uma pequena modificação para que o método *push-relabel* calcule o corte mínimo: redefinir um vértice ativo como sendo um vértice $v \in V \setminus \{s, t\}$ tal que $excess_f(v) > 0$ e $d(v) < n$. Quando o algoritmo modificado finaliza sua execução, o $excess_f(t)$ é o valor de um fluxo máximo e o corte $(S, \overline{S})$ tal que $\overline{S}$ contém exatamente aqueles vértices a partir dos quais t é alcançável em D_f é um corte mínimo.

O algoritmo *push-relabel* geral encontra um fluxo máximo em tempo $O(n^2m)$. Outras variantes do algoritmo apresentam as seguintes complexidades (GOLDBERG; TARJAN, 1988):

- *First-in, First-out (FIFO):* mantendo uma fila Q, essa variante do algoritmo executa operações de descarga até que Q esteja vazia. Uma operação de descarga consite em remover o vértice v do topo da pilha através da aplicação de operações de *push/relabel*

em v até que $excess_f(v) = 0$ ou até que $d(v)$ aumente, e em adicionar qualquer vértice ativado por essas operações ao final da fila. A complexidade dessa variante é $O(n^3)$.

- *Árvores Dinâmicas:* uma variante do algoritmo *push-relabel* utilizando árvores dinâmicas (SLEATOR; TARJAN, 1983; TARJAN; WERNECK, 2009) encontra um fluxo máximo em $O(mn \log \frac{n^2}{m})$.

Maiores detalhes e as provas das complexidades dos algoritmos podem ser encontrados em Goldberg e Tarjan (1988). O algoritmo *push-relabel* encerra a exposição deste trabalho sobre os algoritmos gerais para calcular o fluxo máximo. Uma apresentação mais completa dos algoritmos, incluindo notas históricas, pode ser encontrada em Schrijver (2002) ou Schrijver (2003). O próximo capítulo apresenta o algoritmo pseudofluxo em detalhes.

A escolha pela criação de um capítulo para tratar especificamente do algoritmo pseudofluxo se deve ao fato de que este algoritmo foi desenvolvido para resolver o problema do fechado máximo apesar de também resolver problemas de fluxo máximo. O desenvolvimento desse algoritmo revela alguns conceitos interessantes e marca a relação de equivalência entre este problema e o problema do *maximum blocking-cut*. Desta forma, um tratamento mais detalhado desse algoritmo se faz necessário.

5 ALGORITMO PSEUDOFLUXO

Conforme discutido no Capítulo 3, o problema da mineração a céu aberto desempenhou um importante papel não apenas na área de engenharia de minas, mas também em pesquisa operacional. O OPMP é um problema do conjunto fechado de peso máximo com características específicas, em particular, conduzindo a redes com grandes quantidades de vértices e de arcos.

Na década de 1990, mesmo com uma teoria robusta e algoritmos eficientes para o problema do fluxo máximo e corte mínimo, a indústria da mineração ainda enfrentava problemas de desempenho computacional na obtenção do contorno ótimo para as operações de mineração. O algoritmo mais usado pela indústria era o algoritmo de Lerchs e Grossmann (1965), o qual resolve o problema do fechado máximo, obtendo a solução ótima sem utilizar o conceito de fluxo.

Hochbaum e Chen (2000) realizaram um estudo extensivo do OPMP com foco na eficiência dos algoritmos existentes na epoca. Nesse estudo, investigaram as características específicas do problema, bem como realizaram uma análise comparativa entre o algoritmo de Lerchs e Grossmann (LG) e um algoritmo baseado no método *push-relabel*, o algoritmo mais rápido para o problema do fluxo máximo na época, constatando que este último era superior ao primeiro em todos os cenários. O algoritmo LG, no entanto, apresentava uma vantagem devido ao uso mais econômico que fazia do espaço na memória. Além disso, o algoritmo LG apresenta uma estrutura, denominada árvore normalizada, de especial interesse, como será discutido neste capítulo.

Hochbaum (2001), dando sequência ao estudo mencionado acima, apresentou um algoritmo especificamente para o problema do fechado máximo e que, portanto, também resolve o problema do corte mínimo, sem a utilização do conceito de fluxo. O algoritmo é baseado no algoritmo LG, resolve o problema do corte mínimo de forma fundamentalmente diferente dos métodos conhecidos até então e apresenta eficiência competitiva em relação aos algoritmos conhecidos.

Um novo algoritmo para o problema do fluxo máximo, o algoritmo pseudofluxo, foi apresentado em Hochbaum (2008). Esse algoritmo resolve diretamente o problema do *maximum blocking-cut*, um problema equivalente ao problema do fechado máximo (e do corte mínimo). A partir dessa solução, a complexidade para se obter um fluxo máximo é de $O(mn \log n)$.

O algoritmo pseudofluxo apresenta uma série de características importantes: o conceito de pseudofluxo generaliza o conceito de prefluxo (preflow) à medida que permite, além de excessos de fluxo, também déficits de fluxo; fonte e sorvedouro não desempenham papel relevante; arcos adjacentes à fonte e ao sorvedouro se mantêm saturados durante toda a execução do algoritmo; inicialmente, o algoritmo não utilizava o conceito de fluxo, mas de massa. A massa de um galho seria a soma dos pesos de todos os vértices no galho, considerando o grafo ponderado nos vértices (HOCHBAUM, 2001).

As seções a seguir apresentam, respectivamente: as relações entre os problemas do fluxo máximo, do *maximum blocking-cut* e do fechado máximo; uma breve apresentação da estrutura chamada árvore normalizada; e uma descrição do algoritmo pseudofluxo e de suas características.

5.1 Relação entre Problemas

O *maximum blocking-cut problem* (MBCP) é definido em um digrafo com capacidades nos arcos e pesos nos vértices, sem fonte nem sorvedouro. O objetivo do problema é encontrar um subconjunto de vértices que maximize a soma dos pesos dos vértices menos a soma das capacidades dos arcos do corte definido pelo subconjunto. Este problema apareceu em diversas formas na literatura e é também chamado de problema do *maximum surplus cut* (HOCHBAUM, 2008).

Este problema é uma generalização do problema do fechado máximo e apresenta uma estreita relação com o problema do corte mínimo, como é demonstrado a seguir. Para essa discussão, é necessário retomar alguns conceitos apresentados durante o trabalho, definir alguns conceitos novos e apresentar três grafos associados.

Seja $D = (V, A)$ uma rede conforme definida previamente e dados dois conjuntos $P, Q \subset V$, seja (P, Q) o corte $\partial^+(P)$. Como fluxo e pseudofluxo não são utilizados no mesmo grafo, utiliza-se a mesma representação e define-se formalmente pseudofluxo:

Definição 5.1. Um pseudofluxo f em uma rede D é uma função que associa a cada arco (u, v) um valor real f_{uv}, tal que $0 \leq f_{uv} \leq c_{uv}$.

Observe que um pseudofluxo é semelhante a um fluxo, mas não exige a restrição da conservação do fluxo de forma que a ideia do algoritmo é trabalhar com pseudofluxos durante a execução e transformá-los em fluxo viável ao final. Para um pseudofluxo f em uma rede simples, a capacidade residual c^f_{uv} de um arco (u, v) é definida da mesma forma que para um fluxo, isto é:

- $c^f_{uv} = c_{uv} - f_{uv}$ para um arco direto $(u, v) \in A$;
- $c^f_{vu} = f_{uv}$ para um arco reverso (v, u) tal que $(u, v) \in A$.

A_f representa o conjunto de arcos residuais, ou seja, os arcos com capacidade residual positiva. Para $P, Q \subset V$, $P \cap Q = \emptyset$, e dado um pseudofluxo f, define-se:

(i) Capacidade do corte (P, Q):

$$C(P, Q) = \sum_{(u,v)\in(P,Q)} c_{uv},$$

(ii) pseudofluxo total de P para Q:

$$f(P, Q) = \sum_{(u,v)\in(P,Q)} f_{uv},$$

(iii) Capacidade residual total do corte (P, Q):

$$C^f(P, Q) = \sum_{(u,v)\in(P,Q)} c^f_{uv}.$$

Utilizaremos três grafos distintos para explicar as relações entre os três problemas de interesse no momento: o MBCP, o problema do corte mínimo e o problema do conjunto fechado de peso máximo. Considere os grafos D, D_{st} e D^{ext} definidos a seguir:

(i) $D = (V, A)$ é um grafo direcionado com pesos $w_i \in \mathbb{R}$ para cada $i \in V$, e capacidades positivas c_{ij} para cada arco $(i, j) \in A$.

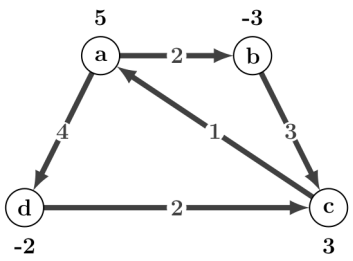
(ii) $D_{st} = (V_{st}, A_{st})$ é uma rede, conforme definido no Capítulo 4. Essa rede é obtida a partir de D conforme o procedimento ilustrado na seção 3.1, ou seja, $V_{st} = V \cup \{s, t\}$; $A_{st} = A \cup A(s) \cup A(t)$, onde $A(s) = \{(s, j) \mid w_j > 0\}$ é o conjunto dos arcos de s a um vértice j de peso positivo, de forma que $c_{sj} = w_j$, e $A(t) = \{(j, t) \mid w_j < 0\}$ é o conjunto de arcos saindo de j, de peso negativo, para t, de forma que $c_{jt} = -w_j = |w_j|$. Vértices de peso zero não se conectam à fonte nem ao sorvedouro.

(iii) A rede estendida D^{ext} é obtida a partir de D_{st} adicionando-se para cada vértice v distinto de s e t, dois arcos de capacidade infinita, (t, v) e (v, s), e fundindo s e t em um só vértice r, chamado raiz. Os arcos anexados de t para cada vértice são arcos de déficit e os arcos anexados de cada vértice a s são os arcos de excesso. O conjunto desses novos arcos é denotado por $A_\infty = \bigcup_{v\in V}\{(v, r) \cup (r, v)\}$ e $D^{ext} = (V \cup \{r\}, A^{ext})$, onde $A^{ext} = A \cup A(s) \cup A(t) \cup A_\infty$.

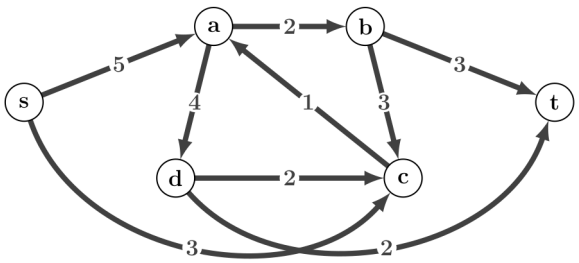
A figura (25a) mostra um exemplo de digrafo D como o definido em (i) e a figura (25b) mostra a rede D_{st} obtida a partir de D de acordo com (ii).

Figura 25 – Grafos Especiais

(a) Digrafo D



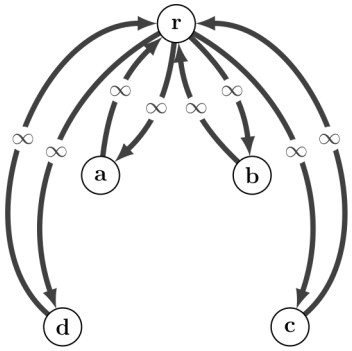
(b) Rede D_{st}



Fonte: (2018)

A figura (26) mostra os arcos do conjunto A_∞ de D^{ext}. Os demais arcos foram retirados da figura para não poluí-la. Os arcos que estão ocultos na figura (26) são os arcos da figura (25b),

Figura 26 – Conjunto de arcos A_∞



Fonte: O autor (2018)

considerando a diferença de que os vértices s e t nesta figura se fundem para formar o vértice r em D^{ext}.

Observa-se que para qualquer pseudofluxo em D_{st}, existe um fluxo viável em D^{ext}, bastando enviar o excesso ou o déficit de cada vértice v de volta para r através dos arcos de excesso (v, r) ou dos arcos de déficits (r, v), respectivamente, em A_∞.

Retomando a função excesso definida no Capítulo 4, agora aplicada a um pseudofluxo no lugar de um fluxo. Assim, dado um pseudofluxo f em D_{st}, seja $inflow(U)$ e $outflow(U)$ o pseudofluxo total entrando em um conjunto de vértices U e saindo de U, respectivamente. Assim

$$\begin{aligned} excess_f(U) &= inflow(U) - outflow(U) \\ &= \sum_{(u,v)\in(V\cup\{s\}\setminus U,U)} f_{uv} - \sum_{(u,v)\in(U,V\cup\{s\}\setminus U)} f_{uv}, \end{aligned}$$

e

$$deficit_f(U) = -excess_f(U).$$

Além disso, nas discussões a seguir, para um conjunto $S \subseteq V$, o complemento de S, $\overline{S} = V \setminus S$, é tomado sempre com relação ao conjunto dos vértices originais, V, mesmo nos grafos D_{st} e D^{ext}.

O MBCP pode ser enunciado da seguinte forma: dado um digrafo $D = (V, A)$ ponderado nos vértices com pesos w_i (positivos ou negativos) para todo $i \in V$, e ponderado nos arcos com pesos não-negativos c_{ij} para cada $(i, j) \in A$, o objetivo é encontrar um subconjunto de vértices $S \subseteq V$ tal que

$$surplus(S) = \sum_{i\in S} w_i - \sum_{i\in S,\ j\in\overline{S}} c_{ij} \text{ é máximo.}$$

O conceito de excedente de um conjunto S, $surplus(S)$, apresentado no grafo do problema acima, encontra conceitos equivalentes nos três grafos especiais definidos anteriormente. Para um conjunto $S \subseteq V$,

(i) no grafo D:

$$surplus(S) = \sum_{j \in S} w_j - \sum_{i \in S,\ j \in \overline{S}} c_{ij},$$

(ii) na rede D_{st}:

$$C(\{s\}, S) - C(S, \overline{S} \cup \{t\}),$$

(iii) no grafo D^{ext}:

$$\sum_{j \in S,\ (r,j) \in A(s)} c_{rj} \ - \sum_{j \in S,\ (j,r) \in A(t)} c_{jr} \ - \sum_{(i,j) \in (S, \overline{S})} c_{ij}.$$

Dada a equivalência entre esses três conceitos, as relações entre o MBCP, o problema do corte mínimo e o problema do fechado de peso máximo se tornam claras. Considere o teorema a seguir,

Teorema 5.1 (Radzik (1993), Hochbaum (2008))**.** *Para $S \subseteq V$, $\{s\} \cup S$ é um conjunto fonte de um corte mínimo em D_{st} se e somente se $(S, \overline{S})$ é um maximum blocking-cut no digrafo D.*

Demonstração. Reescrevendo a função objetivo do problema do *maximum blocking-cut* para o grafo D_{st}, tem-se:

$$\begin{aligned} \max_{S \subseteq V}[C(\{s\}, S) - C(S, \overline{S} \cup \{t\})] &= \max_{S \subseteq V}[C(\{s\}, V) - C(\{s\}, \overline{S}) - C(S, \overline{S} \cup \{t\})] \\ &= C(\{s\}, V) - \min_{S \subseteq V}[C(\{s\}, \overline{S}) + C(S, \overline{S} \cup \{t\})]. \end{aligned}$$

Observe que esta última expressão é equivalente à equação (3.8), onde $C(\{s\}, V)$ é igual à constante W^+, isto é, a soma dos pesos positivos do grafo, e $[C(\{s\}, \overline{S}) + C(S, \overline{S} \cup \{t\})]$ é igual à capacidade do corte $(S, \overline{S})$ em D_{st}. Portanto, minimizando-se à capacidade do corte, obtém-se um *blocking-cut* máximo e o teorema está provado. □

Dessa forma, encontrar um *blocking-cut* máximo, S, é equivalente a encontrar o conjunto fonte, $\{s\} \cup S$, de um corte mínimo, e vice-versa. Por outro lado, o MBCP é uma generalização do problema do fechado de peso máximo.

Conforme discutido no Capítulo 3, o problema do fechado máximo consiste em encontrar um conjunto fechado de vértices de peso total máximo em um digrafo ponderado nos vértices. Através do procedimento apresentado naquele capítulo, é possível reduzir o problema do fechado máximo em D a um problema de fluxo em D_{st}, definindo as capacidades dos arcos em A como infinitas e anexando os arcos $A(s)$ e $A(t)$, definidos anteriormente neste capítulo. Em D_{st}, qualquer corte de capacidade finita $C(S, \overline{S} \cup \{t\})$ deve ser tal que $(S, \overline{S}) = \emptyset$, de forma que S é um conjunto fechado, o que implica em $C(S, \overline{S} \cup \{t\}) = C(S, \{t\})$.

O problema do *maximum blocking-cut* generaliza o problema do fechado máximo relaxando a restrição do fechamento do conjunto de vértices, mas aplicando uma penalidade igual à capacidade do arco que viola essa restrição. O algoritmo pseudofluxo resolve o problema do *maximum blocking-cut*, o que fornece um corte mínimo e, portanto, uma solução para o problema do fechado máximo (HOCHBAUM, 2008).

5.2 Árvores Normalizadas

O algoritmo pseudofluxo mantém em cada iteração uma estrutura chamada árvore normalizada, apresentada por Lerchs and Grossmann (1965). Uma árvore normalizada $T = (V \cup \{r\}, E_T)$ é definida em uma árvore geradora no grafo D^{ext} enraizada em r de forma que $E_T \subseteq A \cup \bigcup_{v \in V} \{(v, r) \cup (r, v)\}$.

Os filhos de r nessa árvore geradora são denotados por r_i e são chamados de raízes dos seus galhos ou subárvores. Em uma árvore normalizada, apenas as raízes de cada galho podem apresentar excessos ou déficits diferentes de zero. Dado um pseudofluxo f, um galho T_{r_i} é dito *forte* se $excess(T_{r_i}) = excess(r_i) = f_{r_i,r} > 0$ e *fraco* caso contrário. Todos os nós em um galho forte são fortes e todos os nós em um galho fraco são fracos. Observe que galhos com excesso igual a zero são galhos fracos.

A árvore normalizada T induz uma floresta em $D = (V, A)$ formada pelos arcos em $E_T \cap A$. Como a árvore normalizada desempenha papel fundamental no algoritmo pseudofluxo, os arcos no conjunto $E_T \cap A$ são chamados eventualmente de *arcos da árvore* enquanto que os arcos no conjunto $A \setminus E_T$ são chamados de *arcos fora da árvore*.

Normalmente, árvores são representadas pondo-se a raiz em cima e os galhos emanando da raiz e florescendo para baixo, de forma que, em relação a uma tal árvore, a direção "para baixo" indica um afastamento em relação à raiz, enquanto que a direção "para cima" indica uma aproximação em relação à raiz.

Definição 5.2. Uma árvore geradora T em D^{ext} com um pseudofluxo f em D_{st} é chamada uma árvore normalizada se satisfaz as propriedades a seguir:

Propriedade 1. O pseudofluxo f satura todos os arcos adjacentes à fonte e ao sorvedouro, isto é, satura todos os arcos em $A(s) \cup A(t)$.

Propriedade 2. O pseudofluxo nos arcos fora da árvore é igual à capacidade mínima (zero, geralmente) ou à capacidade máxima dos respectivos arcos.

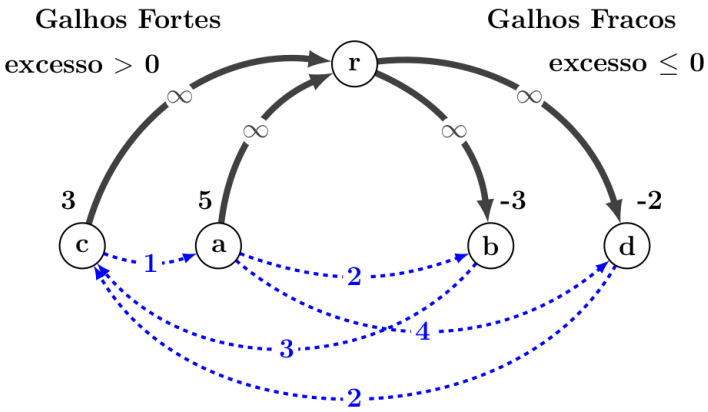
Propriedade 3. Em cada galho, todas as capacidades residuais para baixo são estritamente positivas.

Propriedade 4. Os únicos vértices que não satisfazem a restrição de conservação de fluxo em D_{st} são as raízes dos respectivos galhos.

A propriedade 4 indica que os únicos arcos de déficit ou de excesso permitidos em uma árvore normalizada são os arcos que conectam as raízes dos galhos, r_i, à raiz da árvore r. Essa propriedade torna evidente que o excesso de um galho é igual ao excesso de sua raiz.

Na figura (27), os arcos contínuos indicam uma árvore normalizada inicial construída a partir da rede D^{ext}. As arcos pontilhados são os arcos originais do digrafo D. Do lado esquerdo de r estão os arcos de excesso, com capacidade infinita enquanto que do lado direito estão os arcos de déficit. Os números em cima de cada vértice são os excessos de cada subárvore

Figura 27 – Árvore Normalizada Inicial



Fonte: O autor (2018)

enraizada nesses vértices. Os arcos tracejados são os arcos de A com suas capacidades. Nesse momento, o pseudofluxo em cada arco fora de árvore é igual à capacidade mínima de cada arco, zero, conforme a propriedade 2. Os arcos de $A(s) \cup A(t)$ foram saturados conforme a propriedade 1 e por isso geraram os excessos e déficits mostrados.

Toda árvore normalizada apresenta uma propriedade crucial em relação ao problema do *maximum blocking-cut*, a superotimalidade, enunciada a seguir.

Propriedade 5. (Superotimalidade) O conjunto dos vértices fortes de uma árvore normalizada T é uma solução superotimal do problema do *maximum blocking-cut*. Isto é, a soma dos excessos dos galhos fortes de T é um limite superior para o excedente máximo (HOCHBAUM, 2008).

Para demonstrar a validade dessa propriedade, serão apresentados dois lemas e uma nova definição. Considere o lema a seguir.

Lema 5.1. Para um subconjunto de vértices $S \subseteq V$ tal que $\overline{S} = V \setminus S$ e um pseudofluxo f saturando $A(s) \cup A(t)$, $surplus(S) = excess_f(S) - C^f(S, \overline{S})$.

Demonstração. Para um pseudofluxo f e um conjunto de vértices $U \subseteq V$, a capacidade residual do corte $(U, \overline{U})$, onde $\overline{U} = V \setminus U$, é dada por:

$$C^f(U, \overline{U}) = \sum_{(i,j) \in A \cap (U, \overline{U})} (c_{ij} - f_{ij}) + \sum_{(j,i) \in A \cap (\overline{U}, U)} f_{ji},$$

além disso, com f saturando $A(s) \cup A(t)$, o excesso do conjunto U pode ser escrito como:

$$excess_f(U) = C(\{s\}, U) - C(U, \{t\}) + f(\overline{U}, U) - f(U, \overline{U}).$$

O pseudofluxo saindo de U para $\overline{U}$ pode ser representado pela diferença entre a capacidade dos arcos do corte e a capacidade residual desses arcos:

$$excess_f(U) = C(\{s\}, U) - C(U, \{t\}) + f(\overline{U}, U) - \left(C(U, \overline{U}) - \sum_{(i,j) \in A \cap (U, \overline{U})} c_{ij}^f \right),$$

logo

$$excess_f(U) = C(\{s\}, U) - C(U, \{t\}) - C(U, \overline{U}) + f(\overline{U}, U) + \sum_{(i,j)\in A\cap(U,\overline{U})} c_{ij}^f.$$

Os três primeiros termos do lado direito da equação correspondem ao $surplus(U)$ e os dois últimos termos correspondem à capacidade residual do corte, assim:

$$excess_f(U) = surplus(U) + C^f(U, \overline{U})$$

□

O lema (5.1) acima afirma que o conjunto S é um *maximum blocking-cut* quando a capacidade residual do corte $(S, \overline{S})$ é minimizada. A definição a seguir auxilia na prova da propriedade de superotimalidade.

Definição 5.3. Para um pseudofluxo f saturando $A(s) \cup A(t)$ e uma árvore normalizada T, define-se

$$surplus^T(S) = excess_f(S) - C^f((S, \overline{S}) \cap T).$$

A partir dessa definição e do lema (5.1), segue que $excess_f(S) \geq surplus^T(S) \geq surplus(S)$ e os três são iguais quando $C^f(S, \overline{S}) = 0$, o que ocorre quando $f(S, \overline{S}) = C(S, \overline{S})$ e $f(\overline{S}, S) = 0$.

Lema 5.2. Para uma árvore normalizada T com pseudofluxo f, conjunto de vértices fortes denotado por S e conjunto de vértices fracos denotado por $\overline{S}$,

$$\max_{U\subseteq V} surplus^T(U) = surplus^T(S)$$

Demonstração. O $excess(U)$ é maximizado quando seleciona-se um conjunto U^* contendo todos os vértices com excesso positivo e nenhum vértice com excesso negativo, ou seja, o conjunto U^* deve conter as raízes dos galhos fortes e não deve conter as raízes dos galhos fracos; todos os demais vértices possuem excesso igual a zero. Suponha que para um galho forte $B \subseteq S$, somente um subconjunto de B, B_1, contendo a raiz de B, está contido em U^*. Então, o conjunto de arcos residuais $(B, B_1) \cap T$ é não vazio em virtude da propriedade 3 das árvores normalizadas, e $surplus(B_1) \leq surplus(B)$. Portanto B maximiza o valor de $surplus^T(B)$ para qualquer $U \subseteq B$. Além disso, a inclusão de qualquer subconjunto de vértices fracos que não inclui a raiz do galho fraco não pode aumentar o valor de $surplus^T(U^*)$. Assim, o máximo é obtido com o conjunto dos vértices fortes, S. □

Como $(S, \overline{S}) \cap T = \emptyset$,

$$excess_f(S) = \max_{U\subseteq V} surplus^T(U) \geq \max_{U\subseteq V} surplus(U).$$

Dessa forma, o excesso de S é um limite superior para o valor do conjunto excedente máximo, provando a propriedade de superotimalidade.

Quando a capacidade residual dos arcos no corte $(S, \overline{S})$ é zero, $excess_f(S) = surplus^T(S) = surplus(S)$. Considerando isso juntamente com o lema (5.2), obtém-se um condição de otimalidade:

Condição de Otimalidade: Para uma árvore normalizada T com um pseudofluxo f saturando $A(s) \cup A(t)$ e um conjunto de vértices fortes S, se $C^f(S, \overline{S}) = 0$, então S é um conjunto de excedente máximo e $(S, \overline{S})$ é um *blocking-cut* máximo.

Definição 5.4. Uma árvore normalizada com um pseudofluxo f é ótima se para o conjunto de vértices fortes S na árvore, $(S, \overline{S}) \cap A_f = \emptyset$.

A ideia do algoritmo pseudofluxo é, então, eliminar os arcos que vão de um vértice forte para um vértice fraco na árvore normalizada. A não existência de um tal arco é a condição de parada do algoritmo e o conjunto dos vértices fortes na árvore normalizada é, portanto, um *maximum blocking-cut*, que, por sua vez, dá origem a um conjunto fonte de um corte mínimo em D_{st}, como mostrado na seção anterior.

5.3 Algoritmo

O algoritmo pseudofluxo tem como entrada uma árvore normalizada e um pseudofluxo saturando $A(s) \cup A(t)$. Uma iteração do algoritmo consiste em buscar um chamado *arco de fusão*, ou seja, um arco saindo de um vértice forte para um vértice fraco. Se não houver arco de fusão, a árvore normalizada é ótima.

Se o algoritmo encontrar um arco de fusão, tal arco é anexado à arvore, o arco de excesso do galho forte do arco de fusão é removido, e todo o galho forte é fundido ao galho fraco. Todo o excesso do respectivo galho forte é enviado através do caminho único da raiz do galho forte para a raiz do galho fraco. Qualquer arco neste caminho que não possua capacidade residual suficiente para acomodar a quantidade de excesso enviada é separado e a cauda deste arco se torna a raiz de um novo galho forte com excesso igual à diferença entre a quantidade de excesso enviada e a capacidade residual. O processo de enviar o excesso para a raiz do galho fraco e separar os arcos com capacidade residual insuficiente é chamado *normalização*. A capacidade residual desse novo galho é enviada para cima, até que chega a outro galho também com capacidade residual insuficiente ou ao arco de déficit, adjacente à raiz do galho fraco (HOCHBAUM, 2008).

A versão geral do algoritmo pseudofluxo é apresentada a seguir. Os parâmetros de entrada do algoritmo são um pseudofluxo f em D_{st} saturando $A(s) \cup A(t)$, uma árvore normalizada T associada a f, e os respectivos conjuntos dos vértices fortes, S, e fracos, $\overline{S}$.

Nesta versão, o critério de seleção de arcos de fusão não é especificado. A exposição da corretude do algoritmo e comentários sobre desenvolvimentos e aplicações do mesmo finaliza essa seção. Por uma questão de compacidade, os lemas, definições e provas não serão abordados em detalhes, de forma que para uma exposição mais completa, o leitor deve consultar o artigo seminal de apresentação do algoritmo, Hochbaum (2008).

Figura 28 – Algoritmo pseudofluxo

Procedure pseudofluxo($D_{st}, f, T, S, \overline{S}$)

input : $D_{st}, f, T, S, \overline{S}$
begin
  while $(S, \overline{S}) \cap A_f \neq \emptyset$ **do**
    Selecione $(s', w) \in (S, \overline{S})$
    Seja $r_{s'}$, r_w as raizes dos galhos contendo s' e w, respectivamente
    Seja $\delta = excess_f(r_{s'}) = f_{r_{s'}, r}$
    Merge: $T \leftarrow T \setminus [r, r_{s'}] \cup (s', w)$
    Renormalize: {Envie δ undiades de fluxo através do caminho não direcionado $[r_{s'}, \ldots, s', w, \ldots, r_w, r]$:}
    $i = 1$
    while $v_{r+i} \neq r$ **do**
      Seja $[v_i, v_{i+1}]$ a i-ésima aresta no caminho
      {*Push Flow*}
      if $c^f_{v_i, v_{i+1}} \geq \delta$ **then**
        $f_{v_i, v_{i+1}} \leftarrow f_{v_i, v_{i+1}} + \delta$
      else
        Split$((v_i, v_{i+1}), \delta - c^f_{v_i, v_{i+1}})$
        $\delta \leftarrow c^f_{v_i, v_{i+1}}$
        $f_{v_i, v_{i+1}} \leftarrow c_{v_i, v_{i+1}}$
      end
      $i \leftarrow i + 1$
    end
  end
end

Fonte: Adaptado de Hochbaum (2008)

Figura 29 – função Split

Procedure Split($(a, b), M$)

input : $(a, b), M$
begin
  $T \leftarrow T \setminus (a, b) \cup (a, r)$
  $excess_f(a) = f_{ar} = M$ {a é uma raiz de galho forte }
  $A_f \leftarrow A_f \cup \{(b, a)\} \setminus \{(a, b)\}$
end

Fonte: Adaptado de Hochbaum (2008)

A corretude do algoritmo é expressa na forma do lema a seguir, o qual indica a finitude do algoritmo. Além disso, como a cada iteração o excesso é enviado de volta a r e arcos são separados para se obter uma árvore normalizada, o algoritmo termina com uma árvore normalizada e sem arcos residuais entre vértices fortes e vértices fracos.

Lema 5.3. A cada iteração, ou o excesso total dos vértices fortes é estritamente reduzido, ou ao menos um vértice fraco se torna forte.

A versão geral do algoritmo resolve o problema do *maximum blocking-cut* em $O(nC^*)$ iterações, onde C^* representa a capacidade do corte mínimo. Uma versão do algoritmo baseada em um esquema de *labeling* semelhante ao usado no algoritmo *Push-Relabel* de Goldberg e Tarjan (1988) apresenta complexidade $O(mn \log n)$ (HOCHBAUM, 2008).

Chandran e Hochbaum (2009) realizaram um estudo computacional abragente e compararam, primeiramente, cinco implementações do algoritmo pseudofluxo diferentes em relação ao critério de seleção do arco de fusão, concluindo que a variante *highest-label* é a versão mais eficiente do algoritmo. Em seguida, compararam esta versão do algoritmo pseudofluxo com a versão mais eficiente do algoritmo *push-relabel* em diversas instâncias de problemas de fluxo máximo/corte mínimo. Seus experimentos mostraram que o pseudofluxo é rápido também na prática. De fato, o pseudofluxo se mostrou mais rápido que o *push-relabel* em quase todas as instâncias.

Fishbain, Hochbaum e Mueller (2010) realizaram um estudo comparativo focando no desempenho dos principais algoritmos disponíveis para alguns dos problemas de processamento de imagens mais comuns. Nesses problemas, o problema do corte mínimo desempenha papel crucial. De fato, tal como no problema do fechado de peso máximo, a resolução do problema do fluxo máximo não se faz necessária nesses casos, apenas o corte mínimo determina a resolução desses problemas. Os algoritmos utilizados no estudo foram: (i) *Push-relabel*, (ii) *pseudofluxo*, (iii) algoritmo de Boykov e Kolmogorov (2004), e (iv) *Partial Augmenting-Relabel* de Goldberg (2008).

O estudo de Fishbain, Hochbaum e Mueller (2010) é especialmente importante porque, diferentemente dos trabalhos de Boykov e Kolmogorov (2004) e Goldberg (2008), realiza uma análise dos diferentes estágios do algoritmo em relação ao tempo de execução e ao gerenciamento da memória: inicialização, resolução do problema do corte mínimo e cômputo do fluxo máximo.

Os resultados desse trabalho mostraram que o algoritmo de Boykov e Kolmogorov tem um melhor desepenho em duas situações específicas: (a) pequenas instâncias do problema ($m+n < 1000000$); (b) instâncias caracterizadas por apresentar s-t caminhos curtos. O algoritmo pseudofluxo teve um desempenho superior em todas as outras instâncias além de um melhor gerenciamento da memória, com uma eficiência até 25% maior em relação aos demais algoritmos avaliados.

Hochbaum e Orlin (2013) propuseram uma série de melhorias para o algoritmo pseudofluxo e apresentaram duas novas versões, reduzindo a complexidade teórica do algoritmo.

A versão utilizando árvores dinâmicas apresenta complexidade $O(nm \log(n^2/m))$ enquanto que a versão utilizando estruturas de dados simples apresenta uma complexidade $O(n^3)$.

O pseudofluxo tem sido usado recentemente em problemas de mineração de dados e *machine learning* (HOCHBAUM, 2018), segmentação de imagens (HOCHBAUM; SINGH, 2009), grenciamento de projetos (HOCHBAUM, 2016) e defesa (HOCHBAUM; FIHSBAIN, 2011), entre outros.

6 CONSIDERAÇÕES FINAIS

Alguns aspectos matemáticos do problema do conjunto fechado de peso máximo são particularmente interessantes. Todos já foram tangencialmente abordados durante os capítulos anteriores desse trabalho. A próxima seção discute esses aspectos com maior riqueza de detalhes. A seção seguinte tece alguns comentários sobre os algoritmos analizados neste trabalho. A seção (6.3) dedica-se a uma breve exposição de aspectos importantes de algumas aplicações do problema do fechado máximo. Por fim, a seção (6.4) finaliza este trabalho indicando trabalhos futuros.

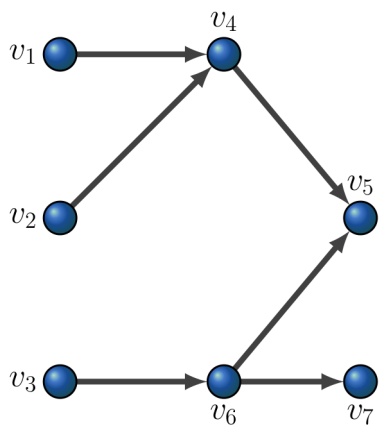
6.1 Aspectos Matemáticos

Naturalmente, um dos aspectos matemáticos mais importantes é o fato de que, apesar de sua semelhança com o problema da mochila, o problema do fechado máximo é um problema polinomial. Conforme discutido no Capítulo 3, através de uma modelagem não trivial, é possível estabelecer uma relação entre um fechado num digrafo e um corte numa rede especial, de forma que a minimização da capacidade do corte maximiza o peso do fechado. Essa modelagem é o que permite resolver o problema do fechado máximo em tempo polinomial.

Outro aspecto matemático importante mencionado no trabalho foi a unimodularidade total da matriz de restrições do problema do fechado máximo. Esse aspecto merece uma breve discussão aqui.

Observe o digrafo na figura (30) a seguir e suponha que a cada vértice é associado um peso b_j, de forma que o vértice v_1 tem peso b_1, o vértice v_2 apresenta peso b_2 e assim sucessivamente.

Figura 30 – Exemplo Fechado Máximo



Fonte: O autor (2018)

Considere novamente a formulação do problema do conjunto fechado de peso máximo,

conforme definido no Capítulo 3, mostrado novamente a seguir por uma questão de conveniência.

$$\begin{aligned} \max \quad & \sum_{j \in V} b_j x_j \\ \text{sujeito a} \quad & x_j - x_i \geq 0 \quad \forall (i,j) \in A \\ & 0 \leq x_j \leq 1 \quad inteiro \quad j \in V \end{aligned} \tag{6.1}$$

A matriz de restrições desse problema de otimização, considerando o digrafo da figura (30), é apresentada na tabela a seguir:

Tabela 6 – Matriz de Restrições

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
(v_1, v_4)	-1	0	0	1	0	0	0
(v_2, v_4)	0	-1	0	1	0	0	0
(v_4, v_5)	0	0	0	-1	1	0	0
(v_6, v_5)	0	0	0	0	1	-1	0
(v_3, v_6)	0	0	-1	0	0	1	0
(v_6, v_7)	0	0	0	0	0	-1	1

Observe que, pela estrutura do problema do fechado máximo, toda linha da matriz de restrições apresenta um 1 e um -1 e todas as demais entradas são 0. Esse fato indica que trata-se de um tipo de matriz totalmente unimodular.

Uma matriz de adjacência é totalmente unimodular se cada um de seus subdeterminantes é igual a 0, 1 ou -1. Em particular, todas as entradas de uma matriz totalmente unimodular são iguais a 0, 1 ou -1, uma vez que cada entrada é, por si só, um subdeterminante de ordem 1 x 1. A relação entre unimodularidade total e programação linear inteira é expressa através do teorema seguinte (SCHRIJVER, 1993).

Teorema 6.1. *Seja A uma matriz totalmente unimodular e seja b um vetor inteiro. Então o poliedro $P := \{x \mid Ax \leq b\}$ é inteiro* [1].

O teorema acima implica que qualquer programa linear com entrada inteira e matriz de restrições totalmente unimodular apresenta solução ótima inteira. Uma caracterização mais abrangente pode ser encontrada em Schrijver (1993). Schrijver (1993) destaca ainda alguns exemplos característicos de matrizes totalmente unimodulares, como a matriz de um grafo bipartido.

A total unimodularidade do problema do conjunto fechado de peso máximo é um indicativo de que o problema pode ser solucionado através do método simplex de maneira satisfatória para instâncias do problema. De fato, o simplex resolve satisfatoriamente o problema do fluxo máximo na rede especial derivada da modelagem apresentada no Capítulo 3. O dual do problema do fluxo máximo é o corte mínimo e determina um fechado de peso máximo. Existem, no entanto, algoritmos combinatórios especializados e que, portanto, oferecem maior eficiência.

[1] A prova desse teorema pode ser encontrada em Schrijver (1993).

Em vertentes do problema do fechado máximo como o problema da mineração à céu aberto ou em aplicações envolvendo quantidades massivas de dados (*Big Data*, *Machine Learning*) esses algoritmos especializados se fazem necessários.

Existe outro aspecto desfavorável à utilização do simplex para resolução do problema do fechado máximo. Conforme discutido neste trabalho, o fluxo máximo não é necessário para resolução do problema do fechado máximo, de forma que a resolução do problema do fluxo máximo implica um certo "desperdício" de processamento. Ainda assim, opta-se pela resolução do problema do fechado máximo como um subproduto do problema do fluxo máximo porque o problema do corte mínimo é frequentemente degenerado, isto é, para esse problema, é possível que o algoritmo simplex entre em uma sequência infinita de iterações sem alterações no valor da função objetivo, conforme definido em Dantzig e Thapa (1997) e Dantzig e Thapa (2003).

Por fim, cabe destacar ainda nos aspectos matemáticos, a abrangência do problema do conjunto fechado de peso máximo. Como mostrado no trabalho, o problema do conjunto fechado de peso máximo generaliza os problemas da seleção, da provisão, dos custos fixos compartilhados e da mineração à céu aberto. Além disso, o Capítulo 5 mostrou que o problema do *maximum blocking-cut* generaliza o problema do fechado máximo, relaxando a restrição de fechamento, mas atribuindo uma penalidade igual à capacidade do arco para cada arco no corte determinado pelo conjunto de vértices. Isto realça a flexibilidade do problema do fechado máximo e suas potencialidades para aplicação nas mais diversas áreas.

6.2 Algoritmos

O algoritmo de Ford e Fulkerson, também chamado de algoritmo do caminho de aumento, desempenha papel fundamental não só para o problema do fechado máximo, mas para problemas de fluxo máximo e problemas de corte mínimo, entre outros. Apesar de não ser o mais rápido, esse algoritmo determinou uma condição precisa de parada para algoritmos de fluxo máximo: a não existência de caminhos de aumento no grafo residual. Essa condição é utilizada na grande maioria dos algoritmos, mesmo que com pequenas alterações.

Diferentemente de outras técnicas como a *flooding technique* de Boldyreff (1955b) e o simplex, o algoritmo de Ford e Fulkerson favorece e evidencia a dualidade entre o fluxo máximo e o corte mínimo, fornecendo ambos ao final da execução. O método primal-dual, como já mencionado no Capítulo 4, representa um avanço importante na consolidação da otimização combinatória como disciplina autônoma.

De fato, a hamornização entre os trabalhos de Ford e Fulkerson e Dantzig firmaram os pilares da otimização combinatória. Com efeito, Dantzig, Ford e Fulkerson (1956) descreveram o primeiro algoritmo primal-dual para problemas gerais de programação linear; o método primal-dual aplicado ao problema do caminho mínimo levou ao desenvolvimento do algoritmo de Dijkstra, e aplicado ao problema do fluxo máximo levou ao desenvolvimento do algoritmo de Ford e Fulkerson (PAPADIMITRIOU; STEIGLITZ, 1998).

Assim, como um dos primeiros algoritmos combinatórios especializados, o algoritmo de Ford e Fulkerson (1956) resolve o problema do fechado máximo com complexidade teórica inferior à complexidade do simplex, resultando em maior eficiência e maior rapidez.

Além disso, o conceito por trás do algoritmo do caminho de aumento é intuitivo e natural, no sentido de que possibilita muitas associações com conceitos físicos, concretos, e facilita sobremaneira o entendimento.

O algoritmo de Ford e Fulkerson recebeu diversas modificações e melhorias durante os 30 anos seguintes ao seu desenvolvimento até que o algoritmo *push-relabel* de Goldberg e Tarjan (1988) foi lançado e tornou-se o *benchmark*, em teoria e prática. Um breve sumário dos algoritmos derivados, em maior ou menor grau, do algoritmo de caminho de aumento é apresentado na tabela a seguir. Uma lista completa dos principais algoritmos de fluxo máximo e suas complexidades pode ser conferida em Schrijver (2003).

Tabela 7 – Algoritmos Fluxo Máximo

Complexidade	**Referências**
$O(n^2mC)$	Dantzig (1955) - *Simplex*
$O(nmC)$	Ford e Fulkerson (1956) - *Augmenting Path*
$O(nm^2)$	Dinic (1970), Edmonds e Karp (1972)
$O(n^2m \log nC)$	Edmonds e Karp (1972)
$O(m^2 \log C)$	Edmonds e Karp (1972) - *Capacity-scaling*
$O(n^2m)$	Dinic (1970)
$O(nm \log n)$	Sleator e Tarjan (1983) - *Dynamic Trees*
$O(nm \log(n^2/m))$	Goldberg e Tarjan (1988) - *Push-relabel + Dynamic Trees*

Fonte: Adaptado de Schrijver (2003)

O algoritmo *push-relabel*, também chamado prefluxo, representou uma mudança no paradigma de desenvolvimento de algoritmos para o problema do fluxo máximo. Em sentido oposto ao da maioria dos algoritmos, o *push-relabel* processa toda sua execução com um fluxo inviável, porém, superotimal. Apenas ao final da execução é que o fluxo se torna viável, e ótimo.

Esse algoritmo tornou-se o algoritmo mais eficiente em teoria e prática, sendo uma espécie de dual do algoritmo de Ford e Fulkerson (1956). À medida que relaxa a condição de conservação do fluxo (em apenas uma direção), o algoritmo de Goldberg e Tarjan (1988) ataca um dos principais problemas do algoritmo do caminho de aumento: a manutenção de um fluxo viável durante toda a execução torna o algoritmo lento. Conforme discutido no Capítulo 4, nos casos em que um arco de grande capacidade é sucedido por diversos arcos de capacidade inferior, o algoritmo *push-relabel* satura o primeiro arco e distribui todo fluxo possível entre os arcos sucessores, retornando o excesso para fonte.

O algoritmo prefluxo incorporou também um esquema de rotulação de vértices que estima o caminho mínimo, em número de arcos, entre cada vértice ativo e o sorvedouro, direcionando o excesso para que este seja enviado o mais "rápido" possível, contribuindo para a eficiência prática do algoritmo.

O algoritmo pseudofluxo de Hochbaum (2008) generaliza, em certos aspectos, o algoritmo *push-relabel*. Isso porque o conceito de pseudofluxo é uma generalização do conceito de prefluxo: enquanto o prefluxo relaxa a restrição de conservação de fluxo em uma direção, permitindo a formação de excessos de fluxo em vértices, o pseudofluxo relaxa a restrição de conservação em ambas as direções, permitindo a formação de excessos e déficits de fluxo nos vértices.

Observe que, à medida que o *push-relabel* satura os arcos conectados à fonte e provoca excesso de fluxo nos vértices do primeiro nível, implicitamente o algoritmo está testando um corte natural; obviamente, a capacidade do corte determinado pelos arcos saindo da fonte é um limite superior para o corte mínimo. O algoritmo pseudofluxo realiza esse "teste" não só para o corte determinado pelos arcos adjacentes à fonte, mas também para o corte determinado pelos arcos adjacentes ao sorvedouro, à medida que satura também estes arcos, provocando déficits nos vértices anteriores ao sorvedouro.

O pseudofluxo apresenta ainda uma outra vantagem em relação ao *push-relabel*: enquanto que o último envia fluxo através de apenas um arco por iteração, o primeiro envia fluxo através de um caminho de um vértice com excesso para um vértice com déficit. Essas são as duas vantagens conceituais do pseudofluxo em relação ao prefluxo.

Mas há que se destacar que essas vantagens são possíveis em razão da utilização da chamada árvore normalizada, utilizada no algoritmo como estrutura principal. Como discutido no Capítulo 5, esta estrutura foi trazida de um algoritmo popular na comunidade de engenharia de mineração e possibilita um gerenciamento mais eficiente da memória do computador durante a execução do algoritmo.

Este trabalho discutiu os principais algoritmos para resolução do problema do conjunto fechado de peso máximo: caminho de aumento de Ford e Fulkerson (1956), *push-relabel* de Goldberg e Tarjan (1988) e pseudofluxo de Hochbaum (2008). O algoritmo *push-relabel* tornou-se o algoritmo mais rápido e eficiente a partir do final da década de 1980 e apenas na última década começou a ser desafiado pelo algoritmo pseudofluxo. Alguns artigos discutidos no final do Capítulo 5 mostram que o pseudofluxo se mostra superior ao *push-relabel* em problemas de fluxo máximo e corte mínimo.

É razoável acreditar que o algoritmo pseudofluxo é o algoritmo mais eficiente conhecido até o momento para resolver o problema do fechado máximo por três razões apresentadas e debatidas ao longo dessa dissertação: (1) o pseudofluxo tem complexidade teórica compatível com a complexidade teórica do *push-relabel*, mas tem se mostrado mais eficiente em problemas gerais de fluxo máximo; (2) o pseudofluxo se mostrou mais eficiente que o *push-relabel* em problemas de visualização de imagens, que apresentam semelhanças importantes com o problema do fechado máximo; (3) o pseudofluxo foi desenvolvido especificamente para resolver o problema da mineração à céu aberto, uma das formas mais gerais do problema do fechado máximo. Contudo, um estudo de desempenho comparativo entre algoritmos especificamente para o problema do fechado máximo parece ser uma necessidade e um campo fértil para trabalhos futuros.

Além disso, nas formas clássicas do problema do fechado máximo, como em problemas de seleção ou da provisão, onde os arcos do digrafo original não apresentam capacidades associadas, o algoritmo pseudofluxo ganha maior velocidade uma vez que não precisa testar se o excesso de um vértice é compatível com a capacidade residual do caminho até um vértice com déficit de fluxo. Nesses casos, a frequência de utilização da função Split() é drasticamente reduzida e, consequentemente, a execução do algoritmo torna-se substancialmente mais rápida.

6.3 Aplicações

Durante este trabalho, diversas aplicações do problema do conjunto fechado de peso máximo foram apresentadas e discutidas. Algumas delas foram discutidas em seu contexto histórico com foco na sua contribuição para a consolidação do problema do fechado máximo e outras foram discutidas por uma questão de completude na exposição.

As aplicações clássicas, como em problemas de seleção/provisão, podem ser encontradas em qualquer campo de atuação porque, após a modelagem discutida neste trabalho, reduzem-se a um problema com requisitos mínimos. Basicamente, o problema consiste em otimizar a seleção de itens mantendo-se as relações existentes entre os itens. Nesses casos, o fato de o digrafo ser bipartido facilita a resolução e permite a utilização de algoritmos eficientes em termos de complexidade teórica e rápidos na prática.

Contudo, uma visão superficial sobre essas aplicações encobre detalhes importantes que são dependentes das especificidades do problema. Um estudo mais aprofundado das características específicas dessas aplicações, como é o estudo de Hochbaum e Chen (2000) sobre o problema da mineração à céu aberto, pode indicar melhorias conceituais e algorítmicas para uma resolução mais eficiente.

O problema da mineração à céu aberto é mais complexo do que as aplicações supracitadas. O seu digrafo característico, em geral, não é bipartido e na prática sua entrada tem dimensões elevadas. Ainda assim, o estudo de Hochbaum e Chen (2000) revelou mecanismos eficazes para melhoria do desempenho computacional dos algoritmos para resolver o problema. Elas mostraram que detalhes como a escolha do padrão de arestas, a reversão do sentido do digrafo e a utilização de um pré-processamento podem reduzir substancialmente o tempo de processamento e a utilização da memória do computador.

O trabalho de Hochbaum e Chen (2000) pode ser usado como um modelo para futuros estudos das aplicações do problema do conjunto fechado de peso máximo. Trabalhos como esse são necessários atualmente para preencher uma lacuna na literatura especializada: apesar de existir uma teoria bem documentada sobre a redução do problema do fechado máximo para um problema de corte mínimo, são necessários estudos mais aprofundados sobre os aspectos específicos das aplicações práticas do problema.

Cabe destacar, ainda em relação às aplicações do fechado máximo, os problemas de *scheduling*. Problemas de *scheduling* são inerentemente complexos. De fato, uma grande parcela dos problemas práticos de *scheduling* são tratados através de heurísticas. A resolução de uma

classe desses problemas através do problema do fechado máximo, como mostrado no trabalho de Faaland e Schmitt (1987), requer uma maior atenção em virtude de suas possibilidades.

6.4 Trabalhos Futuros

Ao realizar uma investigação sobre o problema do conjunto fechado de peso máximo, este trabalho permitiu a identificação da necessidade de estudos futuros acerca de aspectos do problema. Os principais são:

- Um estudo de desempenho comparativo entre os principais algoritmos disponíveis para resolução do problema do conjunto fechado de peso máximo;
- Um estudo das características matemáticas e algorítmicas de problemas de seleção/provisão nos moldes do trabalho de Hochbaum e Chen (2000);
- Um estudo mais aprofundado de problemas de *scheduling* com foco nos problemas que podem ser resolvidos através do problema do fechado máximo.

A realização desses estudos pode revelar implicações teóricas e práticas importantes e permitir uma maior utilização prática dessa teoria, principalmente em contextos de produção e gerenciamento das operações.

REFERÊNCIAS

AHO, A. V.; ULLMAN, J. D. **Foundations of computer science**. New York, NY: Computer Science Press, 1992.

AHUJA, R.; MAGNANTI, T.; ORLIN, J. **Network flow**: theory, algorithms, and applications. New Jersey: Prentice Hall, 1993.

AMANKWAH, H.; LARSSON, T.; TEXTORIOUS, B. A maximum flow formulation of a multi-period open-pit mining problem. **Operations Research International Journal**, v. 2014, n. 14, p. 1–10, 2014.

ARORA, S.; BARAK, B. **Computational complexity**: a modern approach. 1st. ed. New York: Cambridge University Press, 2009.

BALINSKI, M. L. Notes on a selection problem. **Management Science**, v. 17, n. 3, p. 230–231, 1970.

BILLERA, L. J.; LUCAS, W. F. Delbert ray fulkerson: august 14, 1924 - january 10, 1976. **Mathematics of Operations Research**, v. 1, n. 4, p. 299–310, 1976.

BOLDYREFF, A. **Determination of the maximal steady state flow of traffic through a railroad network**. California, 1955. Relatório técnico da Rand Corporation.

BOLDYREFF, A. W. Determination of the maximal steady state flow of traffic through a railroad network. **Journal of the Operations Research Society of America**, The RAND Corporation, California, v. 3, n. 4, p. 443 – 465, 1955.

BONDY, J.; MURTY, U. **Graph theory**. 1st. ed. New York: Springer, 2008.

BOYKOV, Y.; KOLMOGOROV, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 26, n. 9, p. 1124–1137, 2004.

CHANDRAN, B. G.; HOCHBAUM, D. S. A computational study of the pseudflow and push-relabel for algorithms the maximum flow problem. **Operations Research**, v. 57, n. 2, p. 358–376, 2009.

COOK, W. J. et al. **Combinatorial optimization**. New York: John Wiley & Sons, 1998.

CORMEN, T. H. et al. **Introduction to algorithms**. 3. ed. Massachusetts: The MIT Press, 2009.

CORNUEJOLS, G. **Combinatorial optimization**: packing and covering. Philadelphia, PA: SIAM, 2001.

DANTZIG, G. B. Application of the simplex method to a transportation problem. In: KOOPMANS, T. C. (Ed.). **Activity analysis of production and allocation**. New York: Wiley, 1955.

DANTZIG, G. B.; FORD, J. L. R.; FULKERSON, D. R. A primal-dual algorithm for linear programs. In: KUHN, H. W.; TUCKER, A. W. (Ed.). **Linear inequalities and related systems**. New Jersey: Princeton University Press, 1956.

DANTZIG, G. B.; FULKERSON, D. R. On the max-flow min-cut theorem of networks. In: KUHN, H. W.; TUCKER, A. W. (Ed.). **Linear inequalities and related systems**. New Jersey: Princeton University Press, 1956.

DANTZIG, G. B.; THAPA, M. N. **Linear programming 1**: introduction. Berlin: Springer-Verlag, 1997.

DANTZIG, G. B.; THAPA, M. N. **Linear programming 2**: theory and extensions. Berlin: Springer-Verlag, 2003.

DASGUPTA, S.; PAPADIMITRIOU, C. H.; VAZIRANI, U. V. **Algorithms**. 1. ed. New York: McGraw-Hill, 2008.

DINIC, E. A. Algorithm for solution of a problem of maximum flow in a network with power estimation. **Soviet Math. Dokl.**, v. 11, n. 5, 1970.

DUAN, Q.; XU, J. On the connectivity preserving minimum cut problem. **Journal of Computer and System Sciences**, v. 80, p. 837–848, 2014.

EDMONDS, J.; KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. **Journal of the Associaton for Computing Machinery**, v. 19, n. 2, 1972.

EISNER, M. J.; SEVERANCE, D. G. Mathematical techniques for efficiente record segmentation in large shared databases. **Journal of the Association for Computing Machinery**, v. 23, n. 4, p. 619–635, 1976.

FAALAND, B.; SCHMITT, T. Scheduling tasks with due dates in a fabrication/assembly process. **Operations Research**, v. 35, n. 3, p. 378–388, 1987.

FISHBAIN, B.; HOCHBAUM, D. S.; MUELLER, S. Competitive analysis of minimum-cut maximum flow algorithms in vision problems. **arXiv preprint arXiv:1007.4531**, 2010.

FORD, J. L. R.; FULKERSON, D. R. Maximal flow through a network. **Canad. J. Math.**, v. 8, p. 399 – 404, 1956.

FORD, J. L. R.; FULKERSON, D. R. **Flows in networks**. New Jersey: Princeton University Press, 1962.

FOULDS, L. R. **Graph theory applications**. New York: Springer, 1992.

GOLDBERG, A. The partial augmenting-relabel algorithm for the maximum flow problem. **ESA 2008**, p. 466–477, 2008.

GOLDBERG, A. V.; TARJAN, R. E. A new approach to the maximum-flow problem. **Journal of the Association for Computing Machinery**, v. 35, n. 4, p. 921 – 940, 1988.

GRANOT, F. et al. Structural and algorithmic properties for parametric minimum cuts. **Math. Program., Ser. A**, n. 135, p. 337–367, 2012.

HARRIS, J. M.; HIRST, J.; MOSSINGHOFF, M. J. **Combinatorics and graph theory**. Second. New York: Springer, 2008.

HARRIS, T. E.; ROSS, F. S. **Fundamentals of a method for evaluating rail net capacities**. California, 1955. Memorando da RAND Corporation.

HOCHBAUM, D. S. A new-old algorithm for minimum-cut and maximum-flow in closure graphs. **Networks**, v. 37, n. 4, p. 171–193, 2001.

HOCHBAUM, D. S. Selection, provisioning, shared fixed costs, maximum closure, and implications on algorithmic methods today. **Management Sciences**, v. 50, n. 6, p. 709–723, 2004.

HOCHBAUM, D. S. The pseudoflow algorithm: a new algorithm for the maximum-flow problem. **Operations Research**, v. 56, n. 4, p. 992–1009, 2008.

HOCHBAUM, D. S. A polynomial time repeated cuts algorithm for the time cost tradeoff problem: the linear and convex crashing cost deadline problem. **Computers & Industrial Engineering**, v. 95, p. 64–71, 2016.

HOCHBAUM, D. S. **Machine learning and data mining with combinatorial optimization algorithms**. INFORMS, 2018. 109-129 p. Disponível em: <https://pubsonline.informs.org/series/educ/>.

HOCHBAUM, D. S.; CHEN, A. Performance analysis and best implementations of old and new algorithms for the open-pit mining problem. **Operations Research**, v. 48, n. 6, p. 894–914, 2000.

HOCHBAUM, D. S.; FIHSBAIN, B. Nuclear threat detection with mobile distributed sensor networks. **Ann. Oper. Res.**, v. 2011, n. 187, p. 45–63, 2011.

HOCHBAUM, D. S. et al. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. **Math. Programming**, v. 62, p. 69–83, 1994.

HOCHBAUM, D. S.; NAOR, J. Simple and fast algorithms for linear and integer programs with two variables per inequality. **SIAM J. Comput.**, v. 23, n. 6, p. 1179–1192, 1994.

HOCHBAUM, D. S.; ORLIN, J. B. Simplifications and speedups of the pseudoflow algorithm. **Networks**, v. 61, n. 1, p. 40–57, 2013.

HOCHBAUM, D. S.; QUEYRANNE, M. The convex cost closure problem. **SIAM J. Discrete Math.**, v. 16, n. 2, p. 192–207, 2003.

HOCHBAUM, D. S.; SINGH, V. An efficient algorithm for co-segmentation. In: **ICCV**. Kyoto: IEEE, 2009. p. 269–276.

JOHNSON, D. S. A brief history of np-completeness, 1954-2012. **Documenta Mathematica**, Extra Volume ISMP, p. 359–376, 2012.

KLEINBERG, J.; TARDOS, E. **Algorithm design**. Boston: Addison-Wesley, 2005.

KORTE, B.; VYGEN, J. **Combinatorial optimization**: theory and algorithms. 5th. ed. Berlin: Springer, 2012.

LAWLER, E. L. **Combinatorial optimization**: networks and matroids. New York: Holt, Rinehart and Winston, 1976.

LERCHS, H.; GROSSMANN, I. F. Optimum design os open-pit mines. **Trans. Canad. Inst. Mining, Metallurgy, Petroleum**, v. 68, p. 17–24, 1965.

MAMER, J. W.; SMITH, S. A. Optimizing field repair kits based on job completion rate. **Management Science**, v. 28, n. 11, p. 1328–1333, 1982.

MCGINNIS, L. F.; NUTTLE, H. L. W. The project coordinator's problem. **Omega**, v. 6, n. 4, p. 325–330, 1978.

NEMHAUSER, G. L.; WOLSEY, L. A. Integer programming. In: NEMHAUSER, G. L.; KAN, A. R.; TODD, M. (Ed.). **Handbooks in operations research and management science**. The Netherlands: Elsevier, 1989.

ODILI, J. B. Combinatorial optimization in science and engineering. **Current Science**, v. 113, n. 12, 2017.

PAPADIMITRIOU, C. H. **Computational complexity.** NY: Addison-Wesley, 1994.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization**: algorithms and complexity. New York: Dover, 1998.

PICARD, J.-C. Maximal closure of a graph and applications to combinatorial problems. **Management Sciences**, v. 22, n. 11, p. 1268–1272, 1976.

PINEDO, M. L. **Scheduling**: theory, algorithms, and systems. 3rd. ed. Berlin: Springer, 2008.

RADZIK, T. Complexity in numerical optimization. In: ____. NJ: World Scientific, 1993. cap. Parametric flows, weighted means of cuts, and fractional combinatorial optimization, p. 351–386.

REITER, S. Choosing and investiment program among interdependent projects. **The Review of Economic Studies**, v. 30, n. 1, p. 32–36, 1963.

RHYS, J. M. W. A selection problem of shared fixed costs and network flows. **Management Science**, v. 17, n. 3, p. 200–2007, 1970.

SCHRIJVER, A. Paths and flows: a historical survey. **CWI Quarterly**, v. 6, n. 3, p. 169–183, 1993.

SCHRIJVER, A. **Theory of linear and integer programming**. New York: John Wiley & Sons, 1999.

SCHRIJVER, A. On the history of transportation and maximum flow problems. **Math. Progamming**, v. 91, p. 437–445, 2002.

SCHRIJVER, A. **Combinatorial optimization**: polyhedra and efficiency. Berlin: Springer, 2003.

SCHRIJVER, A. On the history of combinatorial optimization (till 1960). In: AARDAL, K.; NEMHAUSER, G. L.; WEISMANTEL, R. (Ed.). **Handbooks in operations research and management science**. Amsterdam: Elsevier, 2005. v. 12.

SLEATOR, D. D.; TARJAN, R. E. A data structure for dynamic trees. **Journal of Computer and System Sciences**, v. 26, n. 3, 1983.

TARJAN, R. E.; WERNECK, R. F. Dynamic trees in practice. **Journal of Experimental Algorithmics (JEA)**, v. 14, n. 5, 2009.