Instituto de Matemática, Estatística e Computação Científica
Instituto de Computação
Universidade Estadual de São Paulo

Projeto Supervisionado

Algoritmos para Problemas de Caminhos em Grafos com Classes

Alessandro Andrioni Silva
RA 101324
Orientador: Eduardo Candido Xavier

Neste trabalho estamos interessados em problemas de caminhos em grafos. Tais problemas possuem aplicações em diversas áreas como projeto de circuitos VLSI, fluxo e roteamento em redes, confiabilidade em comunicações e assim por diante.

O problema escolhido para ser abordado é o de Caminho Mínimo com Classes (CMC) e sua generalização, o de Múltiplos Caminhos Mínimos com Classes (MCMC). Dada uma rede óptica, dois nós dela e um requisito de banda, o objetivo do CMC/MCMC é encontrar a rota mais rápida na rede através da qual a banda requisitada possa ser satisfeita [6]. Com a crescente expansão de aplicações de alto desempenho com requisitos de banda extremamente grandes, não só por parte da comunidade científica, mas também no setor privado, como vídeo-conferências, televisão digital de alta definição, as redes ópticas seguem como as candidatas mais promissoras para satisfazerem tais crescentes requisitos de banda, além de apresentarem uma variedade de outras características desejáveis, como imunidade à interferências eletromagnéticas, maior segurança contra grampos e menor custo por capacidade de transmissão[8], portanto há uma forte necessidade de se resolver o CMC/MCMC de forma eficiente.

Supondo que $\mathrm{P} \neq \mathrm{NP}$, sabemos que problemas de otimização NP-difíceis, como o CMC e o MCMC, não possuem algoritmos eficientes que os resolvam de forma exata. Como a busca de soluções ótimas é inaceitável devido à grande quantidade de tempo de execução exigida, é necessário obter algoritmos rápidos mas que consigam gerar soluções de boa qualidade.

0 Introdução

Este capítulo apresenta uma pequena introdução à teoria subjacente a caminhos em grafos com classes. O conteúdo aqui apresentado pode ser encontrado com mais aprofundamento em [2], [3] e [7], com a exceção da parte de grafos com classes.

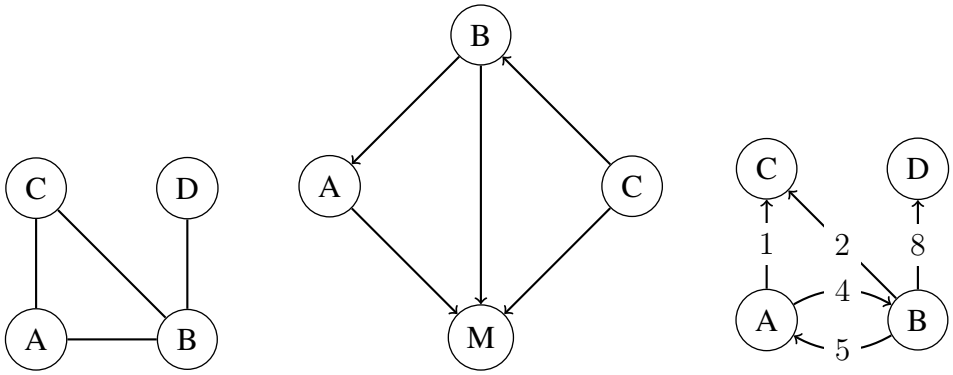
0.1 Grafos

Um *grafo* é um par $G = (V, E)$ onde V é um conjunto de *vértices*, e E é um conjunto de pares $\{u, v\}$ de vértices (isto é, $u, v \in V$), chamados de *arestas*. Um *grafo direcionado* ou *digrafo* é um par $D = (V, A)$ onde V é um conjunto de *vértices*, e A é um conjunto de pares ordenados (u, v) de vértices (isto é, $u, v \in V$), chamados de *arcos*.

Quando estamos trabalhando com grafos direcionados, é normal dizer que o arco (u, v) *vai de* u *até* v, ou que ele *sai de* u e *entra em* v. O conjunto dos arcos que saem de um vértice v é denotado por $\partial^+(v)$, e o conjunto dos que entram por $\partial^-(v)$.

A maneira tradicional de representar um grafo é por meio de um diagrama, onde cada vértice é associado a uma ponto, e suas arestas ou arcos a linhas ligando os pontos. A posição dos pontos não é importante, assim como não é o trajeto das linhas, só os vértices que elas conectam (figura 0.1a). Se o grafo é direcionado, a direção do arco é normalmente indicada por uma seta, como na figura 0.1b.

Como nesse trabalho focaremos no estudo de grafos direcionados, todas as menções subsequentes nesta seção a grafo se referirão a grafos direcionados, mas são em sua maioria facilmente adaptáveis para grafos não-direcionados.



(a) Grafo não-direcionado (b) Grafo direcionado (c) Grafo ponderado

Figura 0.1: Exemplos de grafos

$$M_G = \begin{pmatrix} m_{AA} & m_{AB} & m_{AC} & m_{AD} \\ m_{BA} & m_{BB} & m_{BC} & m_{BD} \\ m_{CA} & m_{CB} & m_{CC} & m_{CD} \\ m_{DA} & m_{DB} & m_{DC} & m_{DD} \end{pmatrix} = \begin{pmatrix} 0 & 4 & 1 & 0 \\ 5 & 0 & 2 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 0.2: Matriz de adjacências do grafo da Figura 0.1c

O conjunto de vértices de um grafo D é denotado por $V(D)$, e o seu conjunto de arcos por $A(D)$. Por exemplo, se temos um grafo $H = (A, B)$, teremos que $V(H) = A$, e que $A(H) = B$.

O número de vértices de um grafo D é a sua *ordem*, escrita como $|V(D)|$; seu número de arcos é escrito como $|A(D)|$. O grafo vazio $(\emptyset, \emptyset)$ costuma ser denotado simplesmente por $\emptyset$.

Dizemos que um vértice w é *adjacente* a ou *vizinho* de um vértice v se existe um arco (v, w). O conjunto dos vizinhos de um vértice v em um grafo D é denotado por $N_D(v)$.

Definimos a união de dois grafos $D = (V, A)$ e $D' = (V', A')$ como sendo $D \cup D' := (V \cup V', A \cup A')$, e a intersecção $D \cap D' := (V \cap V', A \cap A')$. Se $D \cap D' = \emptyset$, então D e D' são *disjuntos*. Se $V \in V'$, e $A \in A'$, então D' é um *subgrafo* de D, escrito como $D' \subseteq D$. Caso $D' \subseteq D$ e $D' \neq D$, D' é dito um *subgrafo próprio* de D.

Se $D' \subseteq D$ e D' contém todos os arcos $(x, y) \in A$ com $x, y \in V$, então D' é um *subgrafo induzido* de D; dizemos que V' *induz* ou *gera* D' em D.

Um *caminho* é um grafo não-vazio $P = (V, A)$ da forma $V = \{x_0, x_1, \ldots, x_k\}, A = \{(x_0, x_1), (x_1, x_2), \ldots, (x_{k-1}, x_k)\}$, e o número de vértices de P é o seu *comprimento*. Dizemos que dois caminhos P e P' são *disjuntos de arestas* se $A(P) \cap A(P') = \emptyset$.

Ao trabalharmos com caminhos, é habitual usar "abusos de notação" para facilitar a leitura. Podemos nos referir ao caminho $P = (V, A)$ como uma tupla (sequência ordenada) de vértices de acordo com os arcos presentes. Por exemplo, no grafo G da figura 0.1b, o caminho $P = (\{D, B, A, M\}, \{(D, B), (B, A), (A, M)\}$ pode ser escrito como $P = (D, B, A, M)$, como é feito no Algoritmo 3.1.1.

0.2 Grafos ponderados e matrizes

Dado um grafo G, dizemos que f é uma *função de peso* se f é da forma $f : A(G) \to \mathbb{R}$, isto é, f associa um valor a cada arco de um grafo. Dizemos que um grafo é *ponderado* se há uma função de peso associada a ele. Um exemplo de uma representação comum de um grafo ponderado é a Figura 0.1c.

Podemos associar a um grafo ponderado uma matriz, chamada de *matriz de adjacências*, definida como $M_G := (m_{uv})$, onde u, v são vértices de G e $m_u v = f((u, v))$ caso o arco (u, v) exista, caso contrário, $m_u v = 0$. A matriz de adjacências do grafo da Figura 0.1c pode ser vista em

0.3 Redes e fluxos

Uma *rede* $N = N(x, y)$ é um digrafo D (o *grafo subjacente a* N com dois vértices especiais, uma fonte x e um sorvedouro y, e uma função $c : A(D) \to \mathbb{R}$ chamada de *função de capacidade* de N, e o seu valor em um arco a é chamado de *capacidade* de a.

Um *fluxo* em uma rede é uma função de peso f com uma restrição adicional: para todo arco $(u, v) \in A(D)$, $f((u, v)) = -f((v, u))$. Um fluxo é dito *factível* se, para todo arco $a \in A(D)$, temos que $0 \leq f(a) \leq c(a)$.

0.4 Classes em grafos

Dado um digrafo $D = (V, A)$ e um conjunto C de classes, denotamos por $C(a) \subseteq C$ o subconjunto de classes que o arco a contém. Um caminho $P = (V', A')$ em D é *viável* se existe $C' \subseteq C$ tal que $C' \subseteq C(a)$, $\forall a \in A'$ e dizemos que P possui conjunto de classes $C(P) = C'$. Também definimos dois caminhos P_i e P_j como *compatíveis* se $C(P_i) \cap C(P_j) = \emptyset$. Além disso, se eles forem disjuntos de arestas, dizemos que são *absolutamente compatíveis*.

Se para uma classe k temos um subgrafo $D' = (V, A')$ de $D = (V, A)$ tal que $\forall a \in A, a \in A' \iff k \in C(a)$, dizemos que D' é o *subgrafo induzido pela classe* k.

0.5 Programação linear

Um problema de programação linear pode ser apresentado da seguinte forma, como um problema de otimização de uma função linear sujeita a restrições lineares:

$$\text{minimizar} \sum_i c_i x_i \tag{Obj}$$

$$\text{sujeito a} \sum_i a_{ij} x_i = b_i \qquad \forall j \tag{R.1}$$

$$x_i \geq 0, \qquad \forall i \tag{R.2}$$

onde Obj é chamada de *função objetivo*, $x_i \in \mathbb{R}$ são as *variáveis de decisão*, $c_i \in \mathbb{R}$ e $b_i \in \mathbb{R}$ são *vetores de coeficientes* dados, e $a_{ij} \in \mathbb{R}$ são os elementos da *matriz de coeficientes*. Cada linha expressão em R.1 e R.2 é chamada de *restrição* do problema.

0.5.1 Programação linear inteira

Um programa linear inteiro difere de um problema de programação linear por possuir variáveis inteiras, e pode ser apresentado da seguinte forma:

$$\begin{aligned} \text{minimizar } & \sum_i c_i x_i & & \\ \text{sujeito a } & \sum_i a_{ij} x_i = b_i & \forall j & \\ & x_i \in \mathbb{Z} & \forall i & \text{(R.I)} \end{aligned}$$

Dado um programa linear inteiro, podemos achar sua *relaxação* trocando as restrições R.I por restrições semelhantes às presentes em R.2.

1 Estudo sobre algoritmos e métodos para caminhos em grafos

Construção de caminhos em redes é um problema básico em otimização combinatória: dada uma rede e pares de nós nela, desejamos encontrar um ou mais caminhos entre eles com determinada propriedade.

Um exemplo deste tipo de problema é o de encontrar caminhos disjuntos de arestas em grafos (*EDP - Edge-disjoint paths*). No EDP, uma instância de entrada consiste de um grafo $G = (V, E)$ e um conjunto $T = \{(s_i, t_i) : 1 \leq i \leq k\}$ de k pares fonte-sorvedouro. O objetivo é conectar o maior número desses pares por caminhos que não compartilhem arestas. EDP é um dos clássicos problemas NP-difíceis [10] que, além de ter aplicações em projetos de circuitos VLSI, é de grande importância em otimização combinatória e teoria dos grafos.

Cheiura e Khanna [5] mostram que é possível melhorar as taxas de aproximação dos algoritmos existentes para o EDP em grafos orientados e não-orientados. Além do mais, os resultados obtidos podem ser estendidos para o caso em que grafos possuem pesos nas arestas e para o problema do fluxo indivisível com capacidade uniforme.

Outro problema de caminhos é o de se encontrar caminhos disjuntos de vértices (*VDP - Vertex-disjoint paths*). A instância de entrada é a mesma do EDP, mas estamos interessados em encontrar caminhos disjuntos nos vértices, ou seja, caminhos que não possuem vértices em comum a não ser nos seus extremos. Este problema é fundamental em roteamento, com aplicações no contexto de circuitos VLSI e confiabilidade de rede [1, 13].

Fortune, Hopcropft e Wyllie [9] provaram que mesmo a versão de decisão do VDP na qual queremos testar a existência de dois caminhos vértice-disjuntos é NP-completa em grafos orientados. Para grafos não orientados, a versão de decisão é solucionada em tempo $O(n^3)$ para qualquer quantidade k fixa de caminhos, como mostrado por Robertson e Seymour [14] (onde n é número de vértices no grafo).

Os problemas anteriores assumem $k \geq 2$ pares fonte-sorvedouro. Um problema parecido é o que considera um único par fonte sorvedouro e deve-se encontrar k caminhos disjuntos entre estes, tal que a soma dos pesos das aresta seja mínima. Suurballe [15] desenvolveu um algoritmo capaz de encontrar tais caminhos em k iterações fazendo o uso de um algoritmo de caminho mais curto juntamente com uma técnica similar a de caminhos aumentantes para obtenção de emparelhamentos máximos em grafos.

A importância do estudo de caminhos em grafos vem da sua aplicabilidade. Em [11], Kolliopoulos trata do problema do fluxo em redes que tem por si só um grande número de aplicações e é de grande importância teórica. Kollipoulos foca no desenvolvimento e análise de algoritmos exatos para o problema de caminhos mínimos a partir de um único nó fonte em grafos cujas arestas possuem custos reais. Ele também apresenta algoritmos aproximados para o problema do fluxo indivisível. Nesse problema, são dados uma rede G, um vértice fonte s e k

commodities com sorvedouros t_i e demandas com valores reais ρ_i, $1 \leq i \leq k$, e o objetivo é rotear as demandas ρ_i de cada *commodity* i através de um único caminho $s - t_i$ tal que o fluxo total através de qualquer aresta e da rede seja limitado pela capacidade do arco.

Outra aplicação apresentada em [4] refere-se ao problema de planejamento de linhas de transporte. Esse problema consiste em determinar linhas de transporte, i.e, caminhos em um grafo, e atribuir números inteiros para essas linhas – suas frequências – através das quais um serviço de transporte será oferecido, respeitando as capacidades máximas da rede. O objetivo em planejamento de linhas de transporte é geralmente a minimização do custo operacional ou do desconforto dos passageiros. Büsing e Stiller [4] exploram a complexidade do problema e do seu dual. Apesar das dificuldades de aproximação, são apresentadas classes especiais de grafos como *grids* e árvores para os quais há um algoritmo de tempo polinomial para o problema.

Por fim, Chen *et al.* [6] estudam o problema de roteamento em grafos. Particularmente, propõem aplicar roteamento com caminhos múltiplos em redes ópticas para aplicações de alto desempenho com requisitos de largura de banda extremamente altos, tipicamente maiores do que a capacidade de um comprimento de onda. Eles apresentam um mecanismo de Provisão de Múltiplos Caminhos de Luz e derivam uma solução ótima para o problema baseada em Programação Linear. Esse mecanismo apresenta melhores resultados não só por reduzir a taxa de bloqueio de requisitos de largura de banda, mas também por reduzir a quantidade de tráfego afetada por falhas de *links* isolados. Contudo, esse método ainda tem potencial para ser mais resistente à falhas e heurísticas eficientes precisam ser desenvolvidas para aplicações mais práticas de roteamento por caminhos múltiplos. Uma formulação alternativa à apresentada por Chen *et al.* para o caso da Provisão de Caminho de Luz Único está presente na seção 3.2.

2 Definição dos problemas de Caminhos Mínimos com Classes

Problema do Caminho Mínimo com Classes (CMC)

Definição 2.0.1. Problema do Caminho Mínimo com Classes (CMC).

Uma instância I de entrada consiste de um conjunto $C = \{c_1, c_2, \ldots, c_l\}$ de l classes, um inteiro k, um digrafo $D = (V, A)$, onde cada arco $a_i \in A(D)$ possui um conjunto de classes $C(a_i) \subseteq C$ e pesos $p(a_i) \in \mathbb{Z}$ e vértices $s, t \in V$. O objetivo é encontrar um caminho mínimo P de s a t tal que P é viável e $|C(P)| = k$. Na versão de decisão do problema (CMC$_d$), queremos saber se é possível encontrar um caminho P viável e tal que $|C(P)| = k$, não necessariamente mínimo.

Problema dos Múltiplos Caminhos Mínimos com Classes (MCMC).

Definição 2.0.2. Problema dos Múltiplos Caminhos Mínimos com Classes (MCMC).

A instância de entrada I é a mesma da Definição 2.0.1 mas agora desejamos encontrar x caminhos de s a t, $P_1, \ldots, P_x$, viáveis e compatíveis entre si tal que o custo total dos caminhos é mínimo e

$$\sum_{i=1}^{x} |C(P_i)| = k. \tag{2.1}$$

Na versão de decisão (MCMC$_d$), desejamos determinar a existência de tais caminhos, não necessariamente mínimos.

Ambos os problemas são NP-Difíceis e portanto não espera-se obter algoritmos comprovadamente rápidos e ótimos para estes problemas.

3 Soluções

A seguir são apresentadas duas soluções exatas para o problema do Caminho Mínimo com Classes, uma algorítmica e uma baseada em Programação Linear Inteira.

3.1 A* Adaptado

Algoritmo 3.1.1 Algoritmo exato para o CMC inspirado no A*

Entrada: instância I do CMC.

Saída: caminho $P : (s, v_1, v_2, \cdots, t)$ de s a t usando pelo menos k classes ou $\emptyset$ se tal caminho não existir.

```
P ← (s)
se h(P) = ∞ então
  retorne ∅
fim se
Adicione P à fila FP
enquanto FP ≠ ∅ faça
  P ← primeiro valor de FP
  u é o último vértice de P
  para todo a_i = (u, v) ∈ ∂⁺(u) faça
    se |C(P) ∩ C(a_i)| ≥ k e v ∉ P então
      se v = t então
        retorne P
      senão
        Adicione P + a_i à fila FP
      fim se
    fim se
  fim para
fim enquanto
retorne P
```

O algoritmo A* Adaptado (Algoritmo 3.1.1, na página 10) é, como sugere seu nome, uma adaptação do algoritmo A* ([12, cap. 4]) de caminho mínimo em grafos. O A* atua como uma busca *best-first*, com uma heurística admissível para acelerar a busca.

Uma heurística admissível é uma estimativa do custo necessário para completar um caminho até o vértice sorvedouro (t) limitada superiormente pelo custo real. Caso essa limitação seja eliminada, a heurística deixa de ser admissível e há chance de ignorar a solução ótima.

O algoritmo A* Adaptado se baseia na construção de caminhos parciais, representados por tuplas da vértices, e armazenados numa fila de prioridade, inicialmente populada com a tupla unitária (s). Para avaliar a prioridade é usada a expressão $f(P) + h(P)$, onde $f(P)$ representa o custo total do caminho parcial P, e $h(P)$ é uma heurística que retorna o custo mínimo necessário para completar o caminho P até o vértice t ignorando restrições de classe, portanto $f(P) + h(P)$ representa um limite inferior do custo da solução.

Para cada caminho parcial retirado da fila de prioridade são geradas suas extensões com os vértices adjacentes, desde que sejam satisfeitas duas restrições: 1. o caminho estendido seja viável, isto é, possua o número de classes exigido; e 2. o caminho estendido seja acíclico, isto é, não pode possuir dois vértices idênticos, o que garante que todos os caminhos parciais possam ser parte da solução. Quando é gerado um caminho completo até t, ele já representa a solução ótima, pelo uso da fila de prioridade e da heurística $h(P)$ admissível.

A admissibilidade da heurística $h(P)$ é garantida pelo fato de ser calculada a partir de uma relaxação das restrições de classe do problema original, e proporciona que seja possível encontrar a solução ótima, enquanto o *branch-and-bound* realizado garante que, assim que seja encontrada uma provável solução, não seja checada nenhuma outra de custo superior.

Notação:

$\partial^+(v)$: o conjunto dos arcos saindo do vértice v

$C(P)$: o conjunto das classes que o caminho P contém.

$f(P)$: o custo do caminho P. Consideramos que o custo de um caminho vazio é infinito.

$h(v_i)$: o custo mínimo de v_i até t.

$h(P)$: o custo mínimo para o completar o caminho P até t ignorando restrições de classe.

$P + v_i$: a tupla formada por todos os elementos do caminho P mais o vértice v_i no final.

FP : uma fila de prioridade de caminhos A ordenada por $f(A) + h(A)$.

3.2 Formulação do CMC como programa linear inteiro

O seguinte modelo é uma formulação em Programação Linear Inteira (PLI) do problema de Caminho Mínimo com Classes:

Notação:

p_{ij} : peso do arco entre o vértice i e o vértice j.

C_{ijw} : disponibilidade da classe w no arco entre o vértice i e o vértice j.

x_{ij} : variável binária indicando o uso do arco entre o vértice i e o vértice j na solução.

y_w : variável binária indicando o uso de classe w na solução.

$$\text{minimizar} \sum_{i,j} p_{ij} x_{ij} \tag{Obj}$$

$$\text{sujeito a} \sum_{j} x_{ij} - \sum_{j} x_{ji} = \begin{cases} 1, & \text{se } i = s \\ -1, & \text{se } i = t \\ 0, & \text{c.c.} \end{cases} \qquad \forall i \tag{R.1}$$

$$\sum_{k} y_w = k \tag{R.2}$$

$$\sum_{w} C_{ijw} y_w \geq k x_{ij}, \qquad \forall i, j \tag{R.3}$$

$$x_{ij}, y_w \in \{0, 1\} \qquad \forall i, j, w$$

O objetivo do programa (Obj) é encontrar um caminho de custo mínimo satisfazendo um conjunto de três restrições:

1. A restrição R.1 é a maneira tradicional (vide [2, cap. 9]) de se representar um problema de caminhos em grafo, garantindo que só haja um par fonte-sorvedouro;

2. A restrição R.2 seleciona as classes utilizadas pela solução e garante que sejam exatamente k, como desejado;

3. A restrição R.3, a mais importante, garante que as classes escolhidas pela variável y_k estejam disponíveis em todo o caminho.

A vantagem desta formulação em comparação com a presente em [6] é uma grande economia tanto de variáveis como restrições. Enquanto a função objetivo Obj e as restrições R.1 e R.2 são análogas às respectivas [6, 1a], [6, 1b] e [6, 1c], a restrição [6, 1d] força a criação de uma variável binária adicional na resolução do problema ([16, p. 9], e obriga o uso do parâmetro w na variável x, o que cria $|C||V|$ restrições do tipo [6, 1b], em oposição às $|V|$ presentes na formulação aqui apresentada. Isso representou nos testes realizados (utilizando o CPLEX 12.4 como *solver*) uma diminuição no tempo necessário para encontrar a solução ótima e no menor uso de memória.

Referências Bibliográficas

[1] A. Aggarwal, J. Kleinberg, and D. P. Williamson. Node-disjoint paths on the mesh and a new trade-off in vlsi layout. *SIAM J. Comput.*, 29:1321–1333, February 2000.

[2] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear programming and network flows*. Wiley-Interscience, 2005.

[3] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008.

[4] C. Büsing and S. Stiller. Line planning, path constrained network flow and inapproximability. *Networks*, 57:106–113, January 2011.

[5] C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 628–637, 2003.

[6] X. Chen, A. Jukan, A. C. Drummond, and N. L. S. Da Fonseca. A multipath routing mechanism in optical networks with extremely high bandwidth requests. In *Proceedings of the 28th IEEE Conference on Global Telecommunications*, GLOBECOM'09, pages 2065–2070, Piscataway, NJ, USA, 2009. IEEE Press.

[7] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[8] H. J.R. Dutton. *Understanding Optical Communications*. IBM, 1998.

[9] S. Fortune, J. E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[11] S. G. Kolliopoulos. Exact and approximation algorithms for network flow and disjoint-path problems, 1998.

[12] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2004.

[13] T. Ohtsuki. The two disjoint path problem and wire routing design. In *Proceedings of the 17th Symposium of Research Institute of Electric Communication on Graph Theory and Algorithms*, pages 207–216, London, UK, 1981. Springer-Verlag.

[14] N. Robertson and P. D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[15] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[16] L.A. Wolsey. *Integer Programming*. Wiley, 1998.