MAC 5701

Relatório de Estudos

Aluno: Renato Lucindo

Orientadora: Profa. Dra. Yoshiko Wakabayashi

Área de Concentração: Otimização Combinatória e Grafos

— São Paulo, 20 de junho de 2005—

Partições Conexas Balanceadas em Grafos

Sumário

1 Introdução

Grafos são estruturas muito usadas para representar a existência ou não de relações entre elementos de um dado conjunto. Assim, redes de comunicação, fluxos em redes de transporte, mapas geográficos e relações binárias em geral são muito bem representados por grafos, e neste caso, várias questões de interesse prático podem ser investigadas. Por exemplo, qual o seu grau de vulnerabilidade (se a queda de poucas ligações ou vértices causam a perda de conexidade), sua estabilidade (qual o número máximo de vértices independentes que contém), qual o seu diâmetro (maior distância entre quaisquer dois de seus vértices), se podem ser particionados em um certo número de subgrafos conexos, etc.

Essa última questão pode ser de interesse tanto pela sua aplicabilidade direta em diversas situações práticas, mas também no seguinte contexto. Algoritmos para problemas em grafos podem utilizar o seguinte procedimento recursivo: dado um grafo conexo G, particiona G em dois subgrafos conexos de tamanhos aproximadamente iguais, resolve os correspondentes subproblemas para esses subgrafos e de suas soluções obtém uma solução para o grafo original G. A exigência de que a partição seja tão balanceada quanto possível objetiva minimizar a profundidade da recursão e a discrepância dos tamanhos dos grafos usados em qualquer nível do processo de recursão.

Estamos interessados em problemas dessa natureza, onde $G = (V, E)$ é um grafo conexo com pesos associados aos seus vértices. Um dos problemas a ser investigado, chamado *Max q-Partição Conexa Balanceada* (Max-PCB$_q$), é o seguinte: dado um grafo conexo $G = (V, E)$ com uma função peso $w : V \to \mathbb{Z}$ definida sobre seus vértices, encontrar uma q-partição $\{V_1, V_2, \ldots, V_q\}$ de V tal que $G[V_i]$ (grafo induzido por V_i) é conexo para $1 \leq i \leq q$ e o peso do mais 'leve' deles seja o maior possível. Mais formalmente, queremos encontrar uma tal partição que maximiza a função $\min\{w(V_i) : i = 1, \ldots, q\}$, onde $w(S)$ denota o peso de um conjunto S, definido como a soma dos pesos de seus elementos.

Temos interesse no estudo deste problema sob o ponto de vista algorítmico. Neste sentido, é natural começar pela investigação de sua complexidade computacional. Descrevemos brevemente alguns dos resultados conhecidos a este respeito na próxima seção.

1.1 Complexidade Computacional

O caso especial em que $q = 2$, onde se busca uma bipartição conexa balanceada, é um problema de grande interesse, e o mais investigado do ponto de vista algorítmico. Este problema na sua versão sem pesos (ou equivalentemente, no caso em que os pesos são uniformes) pode ser resolvido em tempo polinomial para grafos 2-conexos e algumas outras

classes mais especiais. Porém, este problema já é NP-difícil quando G é simplesmente conexo, conforme mostraram Camerini, Galbiati e Maffiolii [10]. No caso mais geral, com pesos quaisquer, temos que o Max-PCB$_2$ pode ser polinomialmente resolvido para escadas [4] e claramente para árvores [22]. Sabemos também que este problema é NP-difícil para grafos planares e também para grades com pelo menos 3 colunas [3].

No caso sem pesos, Dyer e Frieze [12] provaram que Max-PCB$_q$ é NP-difícil mesmo para grafos bipartidos, para todo q fixo, $q \geq 2$. No caso com pesos arbitrários, Salgado e Wakabayashi [23] provaram que Max-PCB$_q$ é NP-difícil no sentido forte, mesmo para grafos q-conexos (q fixo).

1.2 Algoritmos

Não são muitos os algoritmos que foram desenvolvidos para o Max-PCB$_q$. Quando $q = 2$ e G é um grafo completo o problema é equivalente a uma variante do problema Max Mochila, denominada Max Soma de Subconjuntos, onde os lucros e os pesos são iguais. Como Max Mochila tem um esquema de aproximação completamente polinomial [17], segue que o Max-PCB$_2$ também tem um tal esquema para esta classe de grafos.

Ainda no caso $q = 2$, para grafos conexos (e 2-conexos), o melhor resultado conhecido é um algoritmo de aproximação obtido por Chlebíková [11] que resolve o problema dentro de uma razão de $\frac{4}{3}$. No caso $q = 3$. Salgado e Wakabayashi [23] obtiveram uma 2-aproximação para grafos 3-conexos.

No caso em que o grafo de entrada é uma árvore vários algoritmos polinomiais foram desenvolvidos. Destacamos, em particular, o algoritmo devido a Perl e Schach [22] cuja complexidade de tempo é $O(q^2\, r(T) + q\, n)$, onde $r(T)$ é o raio da árvore de entrada T, para qualquer q fixo.

No caso do Max-PCB$_q$ sem pesos, um resultado devido a Lovász [18] (veja também Györi [13]) garante que é possível obter uma q-partição conexa balanceada em tempo polinomial, quando o grafo é q-conexo.

Em suma, para grafos arbitrários e $q \geq 3$, o problema Max-PCB$_q$ ainda foi pouco investigado.

1.3 Aplicações

A grande motivação no estudo de algoritmos para a solução desse problema é a sua aplicabilidade em diversas situações práticas, dentre as quais destacamos as seguintes.

A primeira aplicação vem da área de recuperação de informação. Considere o grafo que modela a classificação automática de termos em classes representando termos mais gerais [24]. Cada termo é representado por um vértice do grafo e dois vértices estão conectados se os termos correspondentes são relacionados. O problema consiste em dividir o conjunto de vértices em conjuntos disjuntos de modo que cada conjunto de vértices irá representar um conjunto de termos para formar um termo mais geral. Tal conjunto de termos deve refletir a interrelação entre os diferentes termos no conjunto. Para a classificação em termos mais gerais ter sentido nenhum subconjunto deve ser muito pequeno. Para particionar o grafo em q termos mais gerais precisamos resolver o Problema Max q-Partição Conexa Balanceada.

Uma outra aplicação interessante é em sistemas operacionais [14, 15, 7]. Considere uma árvore enraizada onde os vértices representam procedimentos; sendo que existe uma aresta de um vértice A para um vértice B se o procedimento A chama o procedimento B. O peso de um vértice representa a quantidade de memória exigida para alocar o procedimento correspondente. Se a soma de memória exigida para todos os procedimentos é maior do que o espaço disponível na memória principal é necessário um sistema de paginação [26]. Um sistema de paginação é uma partição dos procedimentos em conjuntos disjuntos chamados páginas. Somente algumas páginas podem estar na memória simultaneamente, devido às limitações de espaço, as demais são armazenadas em memória secundária. Páginas são trocadas entre as memórias principal e secundária de acordo com a utilização de seus procedimentos na hora da execução. As operações de troca são lentas, devido às operações de entrada e saída que precisam ser feitas na memória secundária. Assim, para reduzir a quantidade destas operações, objetiva-se armazenar procedimentos que são chamados um pelo outro na mesma página. Ou seja, o objetivo é particionar a árvore em q subárvores, e definir para cada página uma tal subárvore.

O tamanho da memória de uma página deve ser suficientemente grande para armazenar todos os seus procedimentos. Visto que páginas diferentes são trocadas com outras (na memória principal), devemos alocar para cada página a quantidade de memória da página de maior tamanho, para permitir trocas entre quaisquer páginas. Neste caso, desejamos minimizar a quantidade de memória da página com o máximo tamanho. Ou seja, buscamos uma q-partição onde a componente com o menor tamanho seja o maior possível.

Uma aplicação [3] que podemos citar é a definição das áreas de responsabilidade de supervisores em uma fábrica. Neste caso, temos unidades individuais (vértices) cujo peso significa o tempo exigido para inspeção e desejamos atribuir porções (componentes) da fábrica para q pessoas distintas de modo que cada parte seja conexa (para garantir concentração espacial da atividade do supervisor). Também busca-se uma atribuição tal que

as cargas de trabalho dos supervisores sejam aproximadamente iguais (balanceadas).

Outra aplicação que podemos citar para motivar o estudo do Max-PCB$_q$ é a definição de áreas para alocação de recursos numa rede. Neste caso, cada vértice tem um peso representando sua demanda e queremos encontrar uma q-partição da rede para a distribuição de recursos de uma forma balanceada para todas as demandas. É o caso, por exemplo, de se definir onde distribuir *mirrors* de dados na Internet.

Encontrar partições balanceadas em grafos é de grande importância também para a elaboração de algoritmos paralelos/distribuídos em grafos. De fato, em sistemas distribuídos quaisquer, para termos um bom aproveitamento do paralelismo precisamos fazer balanceamento de carga, que consiste encontrar conjuntos (partição) de tarefas (vértices) que trocam informações entre si (arestas) e cujo tempo de computação (peso da componente da partição) seja o mais equilibrado possível.

Existem muitas outras aplicações em diferentes áreas como: processamento de imagens [2, 19, 20], análise de *clusters* [21], simulações científicas [25], projeto de redes de telefonia móvel, projeto de circuitos VLSI [1], biologia computacional, etc.

2 Partições Conexas Balanceadas em Árvores

Investigamos inicialmente vários algoritmos que foram desenvolvidos para o caso especial do Max-PCB$_q$ em que o grafo de entrada é uma *árvore*.

Encontramos diversas referências bibliográficas a respeito de algoritmos polinomiais para esse problema [22, 16, 8, 6, 9, 5], e em particular à uma técnica (de *deslocamento*) desenvolvida por Becker e Perl [7] que permite resolver algumas variantes (outras funções objetivo) desse mesmo problema. Nossa pesquisa consistiu em estudar esses algoritmos e implementar um deles, o algoritmo desenvolvido por Perl e Schach [22].

2.1 Definições

Seja $T = (V, E)$ uma árvore com n arestas, no qual estão associados um peso não-negativo $w(v)$ a cada vértice $v \in V$. Uma partição de T em q componentes conexas $T_1, T_2, \ldots, Tq$ é obtida removendo-se $q-1$ arestas de T. O peso $w(T_i)$ de uma componente T_i é a soma dos pesos dos seus vértices.

O problemas que consideraremos aqui é o seguinte: dada uma árvore $T = (V, E)$ em cujos vértices v estão associados um peso não-negativo $w(v)$, encontrar uma partição de T em q componentes conexas que maximiza a função $\min\{w(T_i) : i = 1, \ldots, q\}$.

Consideraremos aqui que a árvore dada tem uma raiz (um de seus vértices é considerado como tal). Observamos que, para melhorar a eficiência do algoritmo pode-se colocar uma raiz artificial (com peso zero) de modo que a distância dessa raiz aos demais vértices seja basicamente igual ao raio dessa nova árvore. Embora a árvore dada não seja orientada, para facilitar a descrição do algoritmo, consideraremos que a árvore de entrada do algoritm é enraizada, e que suas arestas estão orientadas "fugindo" da raiz (isto é, no sentido do caminho que se inicia na raiz e passa por essa aresta, termina no vértice que é um extremo dessa aresta).

Se $e = (v_1, v_2)$, também denotada por $v_1 \to v_2$, é uma aresta orientada, vamos nos referir ao vértice v_1 como $tail(e)$ e a v_2 como $head(e)$. Para nos referirmos que arestas serão removidas da árvore, vamos nos referir aos *cortes que serão feitos ou atribuídos* às arestas dessa árvore. Por conveniência, se um corte c for atribuído à uma aresta e, então também escreveremos $head(c)$ e $tail(c)$ no lugar de $head(e)$ e $tail(e)$, respectivamente.

Um *deslocamento* de um corte c é a transferência de c de uma aresta e_1 para uma aresta adjacente e_2 que não possui corte atribuído a ela, de maneira que $head(e_1) = tail(e_2)$, ou seja, os deslocamentos são sempre *top-down* (afastando-se da raiz). Note que essa restrição no deslocamento garante que nenhum corte será atribuído a uma mesma aresta mais de uma vez, o que permite encontrar limitantes para a análise da complexidade do algoritmo.

Dizemos que uma subárvore T' de T é uma *subárvore parcial* (*completa*) de T enraizada num vértice v se v é raiz de T' e T' contém um (todos os) filho(s) de v e os seus descendentes.

Seja A uma atribuição de k cortes nas arestas de uma árvore T com raiz r. Uma *árvore de corte* $C = (T, A)$ é uma árvore com raiz r e com $k + 1$ vértices, sendo que cada vértice distinto da raiz corresponde a um dos k cortes de A (veja Figura 1 (b).) Um corte c_1 é *filho* de um corte c_2 se existe um caminho sem cortes de $head(c_2)$ até $tail(c_1)$; e c_1 é filho da raiz r se existe um caminho sem cortes de r até $tail(c_1)$.

Uma *componente inferior* de um corte c em T é asubárvore obtida a partir da subárvore completa de T enraizada por $head(c)$ removendo-se todas as subárvores completas enraizadas por $head(v)$, onde v é filho de c em C, se existirem. A *componente raiz* de T é a subárvore obtida de T removendo-se as subárvores completas enraizadas pelos $head(v)$, para todo vértice v que é filho da raiz r em C.

Algumas das definições dadas estão ilustradas na Figura 1.

No caso em que dois cortes c_i e c_j são atribuídos à uma aresta (como na inicialização do algoritmo, veja a seguir), nos referimos a um dos cortes, digamos c_j, como sendo filho de c_i

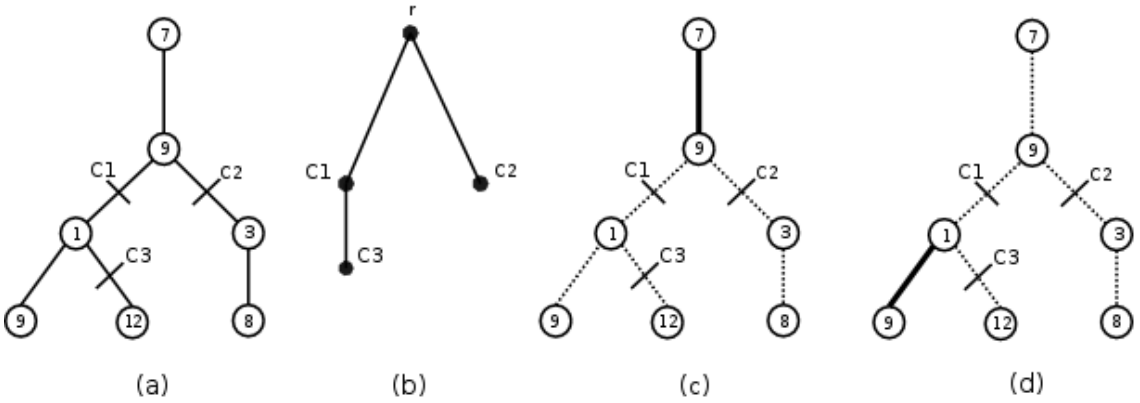

Figura 1: (a) Árvore T. (b) Árvore de corte C. (c) Componente Raiz. (d) Componente Inferior de $C1$

na árvore de corte. A definição de componente inferior implica que a componente inferior de c_i não contém vértices, e portanto tem peso nulo. O caso em que mais de dois vértices são atribuídos à mesma aresta é tratado de maneira similar.

Também dizemos que uma componente T_i é *mais leve* do que uma componente T_j (T_j é *mais pesada* que T_i) se $w(T_i) < w(T_j)$.

2.2 Algoritmo Max-Min (Perl & Schach)

O algoritmo segue uma abordagem *top-down*. Um vértice terminal arbitrário é escolhido como raiz de T, impondo uma direção a partir da raiz a todas as suas arestas. Inicialmente todos os $k = q-1$ cortes são atribuídos à única aresta incidente à raiz. Os cortes são então deslocados, com base em informações locais, de uma aresta para uma aresta adjacente mais afastada da raiz, conseguindo assim melhorias locais até uma solução ótima ser obtida.

Algorithm 1 Algoritmo Max-Min (Perl & Schach)

1: Atribua todos os cortes à única aresta incidente na raiz r
2: $W_{min} \leftarrow$ peso da componente *mais leve*
3: Encontre um deslocamento de um corte c para uma aresta e sem cortes, maximizando o peso RD_c da componente inferior resultante do deslocamento do corte c
4: Se $RD_c \geq W_{min}$ faça o deslocamento do corte c para a aresta e e volte ao passo 2
5: Termine. O peso W_{min} é o peso da componente *mais leve* da $q-$partição *max-min* obtida.

Note que, como cortes numa mesma aresta estão em níveis diferentes na árvore de corte, o algoritmo não termina enquanto mais de um corte estiver atribuído à aresta incidente na raiz da árvore.

Aplicando-se o algoritmo acima para o problema de encontrar uma partição *max-min* da árvore apresentada na Figura 1 em quatro componentes conexas, obtemos a seqüência de deslocamentos ilustrada na Figura 2.

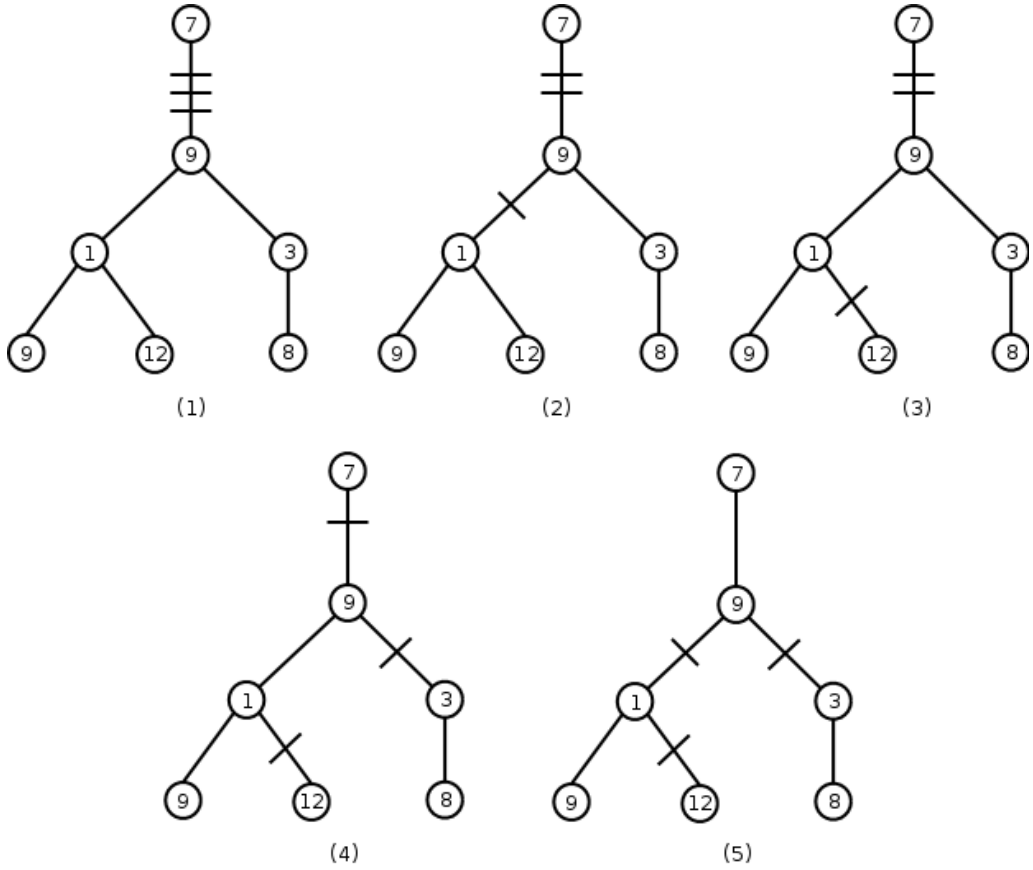

Figura 2: Passos da simulação do algoritmo de *deslocamento*

2.2.1 Implementação

O algoritmo descrito acima foi implementado em C++ usando uma abordagem orientada a objetos. Desenvolvemos classes especícias para trabalhar com cortes e manipulação de arestas de uma maneira eficiente, visto que essas operações são bem exploradas pelo algortimo, e dado que tais recursos não estão facilmente disponíveis em bibliotecas de manipulação de grafos de uso geral. O código pode ser obtido entrando-se em contato diretamente com o autor da implementação (lucindo@ime.usp.br).

2.2.2 Resultados Computacionais

Alguns testes preliminares foram feitos para atestar que o código que temos devolve corretamente a solução exata garantida pelo algoritmo. Testes para verificar o desempenho do algoritmo que implementamos estão sendo realizados. Estamos agora implementando algoritmos para gerar árvores aleatórias com pesos aleatórios, de modo que possamos fazer mais testes computacionais.

Referências

[1] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: The VLSI Journal*, 19:1–80, 1995.

[2] E. Aparo and B. Simeone. Un algoritmo di equipartizione e il suo impiego in un problema de contrasto ottico. *Ricerca Operativa*, 3:31 – 42, 1973.

[3] R. Becker, I. Lari, M. Lucertini, and B. Simeone. Max-min partitioning of grid graphs into connected components. *Networks*, 32(2):115 – 125, 1998.

[4] R. Becker, I. Lari, M. Lucertini, and B. Simeone. A polynomial-time algorithm for max-min partitioning of ladders. *Theory of Computing Systems*, 34(4):353 – 374, 2001.

[5] R. I. Becker and Y. Perl. Shifting algorithms for tree partitioning with general weighting functions. *J. Algorithms*, 4(2):101–120, 1983.

[6] R. I. Becker and Y. Perl. A shifting algorithm for constrained min-max partition on trees. *Discrete Applied Mathematics*, 45(1):1–28, 1993.

[7] R. I. Becker and Y. Perl. The shifting algorithm technique for the partitioning of trees. *Discrete Appl. Math.*, 62(1-3):15–34, 1995.

[8] R. I. Becker, S. R. Schach, and Y. Perl. A shifting algorithm for min-max tree partitioning. *J. ACM*, 29(1):58–67, 1982.

[9] R.I. Becker and Stephen R. Schach. A bottom-up algorithm for weight- and height-bounded minimal partitions of trees. In *CAAP*, pages 63–72, 1984.

[10] P. Camerini, G. Galbiati, and F. Maffioli. On the complexity of finding multi-constrained spanning trees. *Discrete Applied Mathematics*, 5:39–50, 1983.

[11] J. Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60:225 – 230, 1996.

[12] M. Dyer and A. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10:139 – 153, 1985.

[13] E. Györi. On division of graphs to connected subgraphs. In *Combinatorics (Proc. Fifth Hungarian Combinatorial Coll.,1976,Keszthely)*, pages 485 – 494. Bolyai – North-Holland, 1978.

[14] B. W. Kernighan. *Some Graph Partitioning Problems Related to Program Segmentation*. Ph.d. dissertation, Princeton Univ., 1969.

[15] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Sys. Tech. J.*, 49(2):291–308, 1970.

[16] S. Kundu and J. Misra. A linear tree partitioning algorithm. *SIAM J. Comput.*, 6(1):151–154, 1977.

[17] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339 – 356, 1979.

[18] L. Lovász. A homology theory for spanning trees of a graph. In *Acta Math. Acad. Sci. Hunger*, volume 30, pages 241 – 251, 1977.

[19] M. Lucertini, Y. Perl, and B. Simeone. Image enhancement by path partitioning. In V. Cantoni and S. Levialdi, editors, *Recent Issues in Image Analysis*, Lectures Notes in Computer Science. Springer, 1989.

[20] M. Lucertini, Y. Perl, and B. Simeone. Most uniform path partitioning and its use in image processing. *Discrete Applied Mathematics*, 42:227 – 256, 1993.

[21] A. Maravalle, B. Simeone, and R. Naldini. Clustering on trees. *Computational Statistics and Data Analysis*, 24:217 – 234, 1997.

[22] Y. Perl and S. R. Schach. Max-min tree partitioning. *J. ACM*, 28(1):5–15, 1981.

[23] L. R. B. Salgado and Y. Wakabayashi. Approximations results on balanced connected partitions of graphs. *Eletronic Notes in Discrete Mathematics*, 18:207–212, 2004.

[24] G. Salton. *Automatic Information Organization and Retrieval*. McGraw-Hill, 1968.

[25] K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high performance scientific simulations. In *The Sourcebook of Parallel Computing*, pages 491–538. Morgan Kaufmann, 1978.

[26] D. C. Tsichritzis and P. A. Bernstein. *Operating Systems*. Academic Press, 1974.