Algoritmos Paralelos de Granularidade Grossa para Problemas de Alinhamento de Cadeias

Carlos Eduardo Rodrigues Alves[1]*, **Siang Wun Song (orientador do doutorado)**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo
São Paulo, SP

prof.carlos_r_alves@usjt.br

Resumo. *Uma variação do problema de alinhamento de cadeias envolve comparar aos pares a cadeia A com todas as subcadeias de B. São conhecidos algoritmos seqüenciais que resolvem este problema em tempo $O(|A||B|\log(|A|+|B|))$. Se a comparação envolve a determinação do comprimento da mair subseqüência comum, o problema pode ser resolvido em tempo $O(|A||B|)$.*
Propomos novos algoritmos paralelos para este problema, utilizando um modelo próprio para máquinas de memória distribuída, o CGM (Coarse Grained Multicomputers). Os algoritmos aqui propostos apresentam aceleração (speed-up) linear e apenas $O(\log p)$ etapas de comunicação. Não há algoritmos do nosso conhecimento com tais características na literatura.

Abstract. *A variation of the string alignment problem involves the determination of the pairwise similarities of a string A and all the substrings of B. For this variation, sequential algorithms are known that have time complexity $O(|A||B|\log(|A|+|B|))$. If the similarity of two strings is given by the length of the longest common subsequence, the problem can be solved in time $O(|A||B|)$.*
*For this problem, we propose new parallel algorithms under a model suitable to distributed memory systems, the CGM (*Coarse Grained Multicomputers*). The proposed algorithms present linear speed-ups and only $O(\log p)$ comunication rounds. To the best of our knowledge, there is no algorithm for these problems with the suggested properties in the literature.*

1. Introdução

O Problema do Alinhamento de Cadeias está no cerne de muitas aplicações, destacando-se a comparação de seqüências de DNA ou proteínas em Biologia Molecular. Há muitas variações na definição do problema de alinhamento de cadeias [Gusfield, 1997, Setubal and Meidanis, 1997], uma das quais é dada a seguir.

Um alinhamento entre as cadeias A e B é um arranjo de duas linhas, sendo a primeira preenchida com símbolos de A e a segunda com símbolos de B. Os símbolos são colocados no arranjo preservando-se a ordem na cadeia original, com eventuais espaços (aqui representados pelo símbolo " – ") colocados entre eles, de forma que a remoção dos

*Professor da Faculdade de Tecnologia e Ciências Exatas, Universidade São Judas Tadeu

espaços em uma linha leva à cadeia original. Os espaços são inseridos de tal forma que não exista uma coluna com dois espaços no arranjo.

A cada coluna do alinhamento é atribuído um valor, com base nos símbolos presentes e em um certo *esquema de atribuição de valores*. Este esquema pode ser definido por uma função σ: sendo x e y dois símbolos do alfabeto em uso, incluindo " – " neste alfabeto, o valor de uma coluna que contém x e y é $\sigma(x, y)$. O valor de um alinhamento é dado pela soma dos valores de suas colunas.

A seguir temos uma representação de dois alinhamentos possíveis para as cadeias ACTTCAT e ATTCACG (Figura 1). No esquema de atribuição de valores usado neste exemplo, colunas com símbolos coincidentes têm valor 1, outras colunas têm valor 0.

A	A	C	T	T	C	A	–	T		A	A	C	T	T	C	A	–	T	
B	A	T	T	C	–	A	C	G		B	A	–	T	T	C	A	C	G	
val	1	0	1	0	0	1	0	0	3	val	1	0	1	1	1	1	0	0	5

Figura 1: Exemplos de alinhamento.

Podemos então definir o Problema do Alinhamento de Cadeias:

Definição 1 (Problema do Alinhamento de Cadeias) *Dadas as cadeias A e B, encontrar o alinhamento entre elas que apresente valor máximo.*

O valor do alinhamento máximo é chamado também de *similaridade* entre as duas cadeias. Em muitas situações, não estamos interessados no alinhamento em si, mas apenas na similaridade. Algoritmos baseados em programação dinâmica [Gusfield, 1997, Setubal and Meidanis, 1997], podem determinar esta similaridade em tempo $O(n_a n_b)$ e espaço $O(\min\{n_a, n_b\})$. A obtenção do alinhamento propriamente dito pode ser feita mantendo-se a complexidade assintótica quadrática para o tempo e linear para o espaço [Hirschberg, 1975].

2. Alinhamento de Todas as Subcadeias

Uma extensão do Problema do Alinhamento entre Cadeias é o do Alinhamento de Todas as Subcadeias, assim definido:

Definição 2 (Problema do Alinhamento de Todas as Subcadeias - ATS) *Dadas as cadeias A e B, de comprimentos n_a e n_b respectivamente, encontrar o alinhamento entre A e todas as possíveis subcadeias de B: $B_i^j = B_i B_{i+1} \ldots B_{j-1} B_j$, $1 \leq i \leq j \leq n_b$.*

Como há $O(n_b^2)$ subcadeias de B, pode-se esperar uma maior complexidade de tempo e espaço na sua solução do que no caso do alinhamento simples entre A e B. Porém, para algumas versões restritas do problema ATS existem algoritmos mais eficientes, como será comentado.

Há uma série de variações possíveis para o que se considera uma solução para o problema ATS. De maneira geral, espera-se encontrar uma estrutura de dados que permita consultas sobre o *valor* do alinhamento de A com uma determinada subcadeia B_i^j em tempo reduzido, se possível $O(1)$. A determinação do alinhamento propriamente dito leva tempo $\Omega(\max\{n_a, j - i\})$, devido ao próprio comprimento do alinhamento. Dependendo

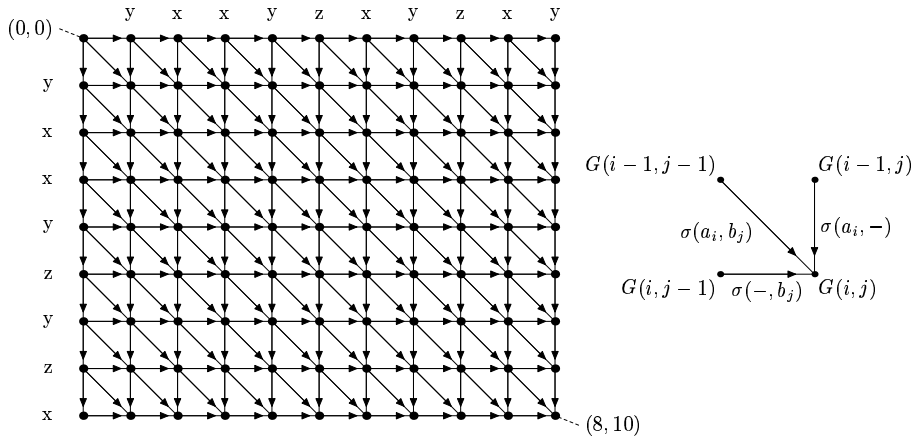

Figura 2: GDAG para os problemas de alinhamento de cadeias e ATS, com cadeias $A =$ yxxyzyzx e $B =$ yxxyzxyzxy.

da aplicação, pode-se tolerar um aumento nestes tempos de consulta desde que o tempo ou o espaço requeridos para construção da estrutura sejam reduzidos.

O problema ATS, com possíveis variações, surge como parte de problemas maiores, dentre os quais podemos citar o problema do alinhamento de uma cadeia com várias outras que possuem cadeias comuns [Landau and Ziv-Ukelson, 2001], o problema do alinhamento circular [Maes, 1990, Schmidt, 1998] e o problema da determinação de repetições aproximadas concatenadas [Schmidt, 1998].

O problema ATS pode ser definido também como sendo a busca pelos comprimentos dos maiores caminhos entre vértices na linha superior e vértices na linha inferior de um GDAG (Grafo Dirigido Acíclico em Grade) como o mostrado na Figura 2. Os pesos nos arcos são dados por σ. O maior caminho entre os vértices $(0, i)$ e (n_a, j) indica a similaridade entre A e B^j_{i+1}.

2.1. Resultados Obtidos

Esta tese contém técnicas paralelas para resolução do problema ATS em duas variações: uma mais geral, em que se considera esquemas genéricos de atribuição de valores, e outra mais específica, em que $\sigma(x, y) = 1$ se e somente se $x = y$, 0 caso contrário. Este último esquema leva à busca pela maior subseqüência comum (*Longest Common Subsequence* – LCS) [Rick, 1994, Setubal and Meidanis, 1997] entre as cadeias. Nos dois casos, o interesse se concentra na obtenção dos valores dos alinhamentos máximos.

O modelo de máquina paralela usado é o BSP/CGM, ou simplesmente CGM [Dehne et al., 1993, (Ed.), 1999]. Este modelo permite o desenvolvimento de algoritmos para máquinas de memória distribuída com previsão satisfatória do seu desempenho. Supõe-se, para fins de análise, que o processamento é realizado em várias rodadas, intercalando-se rodadas de computação local (sem comunicação entre processadores) e rodadas de comunicação (sem processamento). Os principais objetivos no desenvolvimento de algoritmos BSP/CGM são a obtenção de uma aceleração (*Speed Up*) linear no número de processadores p e a minimização do número de rodadas de comunicação. Além disso, a quantidade total de dados recebidos ou enviados por um processador em uma rodada deve ser avaliada, assim como a memória local necessária. Em geral, a análise

de um algoritmo CGM/BSP é feita considerando-se p baixo com relação ao tamanho da instância do problema. Quão alto este p pode ser indica quão boa é a escalabilidade do algoritmo.

Para os dois problemas mencionados foram obtidos algoritmos com aceleração linear e apenas $O(\log p)$ rodadas de comunicação (o número de rodadas não depende do tamanho das cadeias envolvidas, embora a quantidade de dados transferida dependa). No segundo caso, o algoritmo obtido resolve também o problema LCS básico, obtendo a subseqüência comum mais longa entre A e B com aceleração linear.

Estes resultados foram publicados no *14th Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA 2002) [Alves et al., 2002] e no *17th International Parallel and Distributed Processing Symposium* (IPDPS 2003) [Alves et al., 2003].

3. Algoritmo CGM para o Problema ATS

O algoritmo CGM para o ATS é baseado em um algoritmo PRAM, originalmente proposto para o problema de alinhamento simples de cadeias [Apostolico et al., 1990]. Supondo $n_a \leq n_b$, uma versão serializada deste algoritmo apresenta tempo de execução $\Theta(n_a n_b \log n_a + n_b^2)$. Este problema também pode ser resolvido em tempo $\Theta(n_a n_b \log n_b)$ pelas técnicas apresentadas em [Schmidt, 1998].

Para cada par de vértices no GDAG característico do problema, sendo o primeiro vértice da borda superior ou esquerda e o segundo vértice da borda inferior ou direita, deseja-se obter os comprimentos do maior caminho do primeiro para o segundo vértice. Os resultados devem ser armazenados em uma matriz denominada AL_G, sendo G o GDAG em questão. Deve-se notar que os resultados obtidos vão além do exigido no problema ATS, por envolverem as bordas laterais do GDAG.

Assim, como o algoritmo PRAM, o algoritmo CGM é baseado em divisão-e-conquista, dividindo-se cada uma das cadeias A e B em $\sqrt{p}$ trechos (por simplicidade, vamos supor que p é uma potência par de 2). O GDAG G fica assim dividido em p GDAGs menores, que compartilham bordas (linhas ou colunas de vértices) com os vizinhos. Nestes p GDAGs aplicamos o algoritmo seqüencial para o ATS, obtendo assim as matrizes de distância correspondentes.

Seguem-se $\log p$ etapas de união de GDAGs, em que os GDAGs são unidos aos pares para formação de GDAGs maiores. Estas etapas intercalam uniões de GDAGs alinhados verticalmente com uniões de GDAGs alinhados horizontalmente. Cada união envolve a determinação de uma matriz de distâncias para um novo GDAG, baseada nas matrizes de distância dos dois GDAGs que foram unidos.

A cada etapa, o número de GDAGs envolvidos cai pela metade. As matrizes se tornam maiores e devem ser distribuídas nas memórias locais de vários processadores. No final da última etapa, a matriz AL_G estará distribuída nas memórias de todos os processadores.

O processo de união de dois GDAGs deve ser realizado em paralelo por diversos processadores. Este processo é bastante elaborado, para garantir que o número total de rodadas de comunicação seja o mais baixo possível. No caso, são 6 rodadas por etapa de união, totalizando $O(\log p)$ rodadas para o algoritmo completo.

O processo de união é baseado em princípios semelhantes aos usados em [Apostolico et al., 1990]. Supondo que dois GDAGs alinhados verticalmente S (superior) e I (inferior) serão unidos para formar um GDAG U, a matriz AL_U deve ser construída a partir de AL_S e AL_I. De fato, uma parte da matriz AL_U é diretamente copiada das outras matrizes. A parte que precisa ser determinada se refere aos caminhos que se originam em S e terminam em I.

Como os maiores caminhos em S e I já estão determinados, para se determinar o melhor caminho entre um vértice na borda esquerda (por exemplo) de S e outro na borda direita de I é preciso apenas escolher o vértice na borda comum a S e I por onde o caminho deve passar.

Esta determinação deve ser feita para todos os pares de vértices das bordas de S e I de maneira eficiente. Para isto, exploram-se certas propriedades de *monotonicidade total* [Aggarwal et al., 1987] do problema, que podem ser resumidas da seguinte maneira: dois caminhos no GDAG que possuem uma mesma origem ou um mesmo destino *não podem se cruzar*. Assim, cada novo caminho determinado restringe a pesquisa que precisa ser feita para os próximos caminhos. O algoritmo seqüencial para a união de GDAGs é eficiente porque envolve uma boa ordem de determinação dos caminhos.

Na união paralela de GDAGs, supondo que q processadores inicialmente contém a matriz AL_S e outros q contém AL_I, a matriz AL_U é dividida em $2q \times 2q$ blocos de tamanhos iguais a serem calculados em paralelo pelos $2q$ processadores, num total de $4q^2$ subproblemas. Cada subproblema envolve a busca pelos melhores caminhos de um *trecho* da borda de S para um *trecho* da borda de I.

Cada subproblema requer uma parte das matrizes AL_S e AL_I, sendo que as partes usadas por cada subproblema não se sobrepõem (a não ser nas bordas, que são irregulares). Isto permite que os dados de AL_S e AL_I sejam redistribuídos entre os processadores, de forma que cada processador tenha apenas os dados referentes aos problemas alocados para ele. No entanto, os detalhes envolvidos são complexos [Alves et al., 2002]. A seguir, temos uma visão geral do procedimento:

- as bordas dos subproblemas devem ser determinadas, o que envolve a busca pelos melhores caminhos que envolvem alguns pontos-chave das bordas dos GDAGs. Esta etapa consome três rodadas de comunicação.
- estimativas do tempo de execução e do espaço requerido por cada subproblema são calculadas e os problemas são alocados entre os processadores, o que requer mais uma rodada de comunicação.
- Os dados são transferidos entre os processadores, os cálculos são realizados e os resultados são redistribuídos, levando mais duas rodadas de comunicação.

Demonstra-se que o algoritmo CGM completo apresenta aceleração linear no número de processadores, sendo que a memória utilizada por cada processador é $O((n_a + n_b)^2/p)$, não mais do que o requerido para armazenamento da resposta.

4. Algoritmo CGM para o Problema ALCS

O Problema ALCS (de *All-Substrings Longest Common Subsequence*) é uma variação do problema ATS, mas os pesos nos arcos são apenas 0 ou 1, como comentado anteriormente.

Um algoritmo seqüencial de tempo $O(n_a n_b)$, baseado em resultados de [Schmidt, 1998], é apresentado na tese. O algoritmo CGM desenvolvido na tese leva tempo $O(n_a n_b/p)$ (aceleração linear) e requer $O(C \log p)$ rodadas de comunicação, para alguma constante inteira $C \geq 1$ comentada a seguir.

As técnicas de paralelização envolvidas são bastante distintas das usadas no caso do ATS geral, tendo como base um algoritmo PRAM apresentado em [Lu and Lin, 1994]. O uso de representações e estruturas de dados especialmente elaboradas para o algoritmo CGM são essenciais para manter o espaço em memória local $O(n_b\sqrt{n_a})$ e a quantidade de informações transferidas por processador a cada etapa de comunicação praticamente linear, $O(n_a p^{1/C} + n_b)$. A constante C surge para permitir *broadcasts* eficientes em sistemas que não suportam este tipo de operação.

O algoritmo faz uso de uma representação diferente para os comprimentos dos caminhos de um GDAG. No caso, o interesse se concentra apenas nos caminhos da linha superior para a linha inferior. Sendo $C_G(i, j)$ o comprimento do maior caminho entre o vértice i no topo do GDAG G e o vértice j no fundo do mesmo GDAG, $0 \leq i < j \leq n_b$, temos que $C_G(i, j) - C_G(i, j-1)$ é sempre 0 ou 1, bastando então registrar os valores de j para os quais tal diferença é 1.

A matriz D_G armazena estes "limiares", sendo $D_G(i, k)$ ($0 \leq i < n_b$, $0 \leq k \leq n_a$) o valor do k-ésimo limiar referente ao vértice i no topo do GDAG ($D_G(i, 0) = i$ e alguns valores em D_G podem assumir valores infinitos). A linha i da matriz D_G será denominada D_G^i. Uma propriedade interessante de D_G é que linhas consecutivas são *1-variantes*, ou seja, pode-se obter uma linha a partir da anterior pela remoção do primeiro elemento e inserção de um novo elemento. Esta propriedade permite a representação dos resultados do problema ALCS em espaço linear, usando apenas a primeira linha de D_G (D_G^0) e um vetor com os elementos que são adicionados a cada linha (V_G).

O algoritmo CGM começa pela divisão horizontal do GDAG em p GDAGs menores que serão chamadas *faixas*, sendo cada uma deles entregue a um processador. Em cada faixa, o algoritmo seqüencial para o ALCS é usado e a representação linear (D_G^0 e V_G) dos resultados é gerada, em tempo $O(n_a n_b/p)$. A seguir, as faixas são unidas duas a duas em $\log p$ etapas.

Sejam S e I duas faixas que serão unidas em uma faixa U. Em [Lu and Lin, 1994] é mostrado como o processo de união pode fazer uso das matrizes D_S e D_I, através da construção de novas matrizes que têm propriedades de *monotonicidade total*. O algoritmo PRAM resultante é muito interessante e é ótimo no que diz respeito ao trabalho total efetuado, mas as estruturas de dados usadas são adequadas apenas para o modelo PRAM, por exigirem muito espaço compartilhado.

Para o algoritmo CGM apresentado na tese, foi desenvolvida uma estrutura de dados adequada para a representação de uma matriz D_G $n \times m$, que pode ser criada a partir da representação linear em tempo e espaço $O(n\sqrt{m})$ e consultada em tempo constante. Diversas outras modificações foram necessárias, incluindo novas estruturas para a representação das matrizes t.m. (totalmente monotônicas) envolvidas e uma alteração do algoritmo de [Aggarwal et al., 1987] para resolução do Problema dos Mínimos das Colunas nestas matrizes.

Uma breve descrição do procedimento de união das faixas S e I em uma faixa U é

dada a seguir. S e I são GDAGs de altura m e largura $n = n_b$. O número de processadores envolvidos é q.

Considere-se a determinação de uma única linha de D_U, D_U^i ($0 \leq i \leq n$). Naturalmente, os valores de D_S^i serão necessários, pois indicam os vértices na fronteira entre S e I que são candidatos a pertencer aos maiores caminhos que partem do vértice i no topo de U. Pode-se provar que

$$D_U(i,k) = \min_{0 \leq l \leq m} \{D_I(D_S(i,l), k-l)\}. \quad (1)$$

Para determinar D_U^i, pode-se construir uma matriz cujas linhas são cópias deslocadas de linhas de D_I, usando-se D_S^i para escolher que linhas de D_I tomar. Pode-se mostrar que a matriz assim construída é t.m., havendo um algoritmo eficiente para encontrar os mínimos de todas as suas colunas [Aggarwal et al., 1987].

Os q processadores recebem os dados de D_S e D_I nas suas representações lineares e constroem localmente as estruturas de acesso em tempo constante, o que requer tempo e espaço local $O(n\sqrt{m})$. Cada processador irá determinar um conjunto de linhas contíguas de D_U, ocupando-se assim de um trecho contíguo dos vértices do topo de U.

Como linhas contíguas de D_S são muito semelhantes (1-variantes), a determinação de linhas contíguas de D_U envolve uma série de matrizes t.m. também muitos semelhantes, com partes que são comuns a todas elas. Cada processador trabalha com um bloco de aproximadamente $\sqrt{m}$ linhas contíguas de D_U de cada vez, para aproveitar as semelhanças entre as matrizes. As partes comuns a todas as matrizes t.m. envolvidas são determinadas e "contraídas" (cada uma é substituída por uma linhas que contém os mínimos de suas colunas), usando um algoritmo adaptado de [Aggarwal et al., 1987].

Uma vez contraídas as partes comuns das matrizes, a determinação de cada linha de D_U^i envolverá uma matriz t.m. com menos linhas (apenas $O(\sqrt{m})$). Nesta matriz, um novo procedimento é usado para que se determine apenas que elemento está presente na linha D_U^i e não está na linha D_U^{i-1}, o que é suficiente para determinar $V_U(i)$.

O resultado das etapas anteriores é o vetor V_U, que pode ser enviado a todos os processadores que irão usar D_U na próxima etapa de união. Um dos processadores determina e envia D_U^0 para todos os outros processadores (o já mencionado *broadcast*), completando a distribuição da representação linear de D_U.

Após as $\log p$ etapas de união, a representação linear de D_G estará completa nas memórias de todos os processadores, podendo ser estendida para a representação de tamanho $O(n_b\sqrt{n_a})$ para permitir consultas rápidas. Mantendo-se as representações lineares obtidas ao longo de todas as etapas do processo, pode-se ao final obter qualquer caminho no GDAG G. Em particular, isto permite a obtenção da maior subseqüência comum entre A e B, resultando em um algoritmo CGM para o problema LCS clássico, com aceleração linear e apenas $O(\log p)$ rodadas de comunicação.

5. Conclusões

Foram aqui apresentados algoritmos CGM para dois problemas importantes, o ATS e o ALCS, sendo que o algoritmo para o ALCS é adequado também para o problema

LCS clássico. Estes algoritmos apresentam aceleração linear, número de rodadas de comunicação independente do tamanho da instância do problema e boa escalabilidade: pode-se mostrar que as características dos algoritmos são mantidas se a quantidade de processadores empregada for inferior a $\sqrt{n}$, onde n é o tamanho da menor cadeia envolvida. Estes resultados indicam a aplicabilidade prática destes algoritmos.

Referências

Aggarwal, A., Klawe, M. M., Moran, S., Shor, P., and Wilber, R. (1987). Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208.

Alves, C. E. R., Cáceres, E. N., Dehne, F., and Song, S. W. (2002). Parallel dynamic programming for solving the string editing problem on a CGM/BSP. In *Proceedings of the 14th Symposium on Parallel Algorithms and Architectures (ACM-SPAA)*, pages 275–281. ACM Press.

Alves, C. E. R., Cáceres, E. N., and Song, S. W. (2003). A BSP/CGM algorithm for the all-substrings longest common subsequence problem. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IEEE-IPDPS)*. IEEE Computer Society Press.

Apostolico, A., Atallah, M. J., Larmore, L. L., and Macfaddin, S. (1990). Efficient parallel algorithms for string editing and related problems. *SIAM J. Comput.*, 19(5):968–988.

Dehne, F., Fabri, A., and Rau-Chaplin, A. (1993). Scalable parallel computational geometry for coarse grained multicomputers. In *Proc. 9th Annual ACM Symp. Comput. Geom.*, pages 298–307.

(Ed.), F. D. (1999). Coarse grained parallel algorithms. *Special Issue of Algorithmica*, 24(3/4):173–176.

Gusfield, D. (1997). *Algorithms in Strings, Trees and Sequences. Computer Science and Computational Biology*. Cambridge University Press.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communication of ACM*, 18:341–343.

Landau, G. M. and Ziv-Ukelson, M. (2001). On the common substring alignment problem. *Journal of Algorithms*, 41:338–359.

Lu, M. and Lin, H. (1994). Parallel algorithms for the longest common subsequence problem. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):835–848.

Maes, M. (1990). On a cyclic string-to-string correction problem. *Information Processing Letters*, 35:73–78.

Rick, C. (1994). New algorithms for the longest common subsequence problem. Technical Report 85123–CS, Institut für Informatik, Universität Bonn.

Schmidt, J. (1998). All highest scoring paths in weighted graphs and their application to finding all approximate repeats in strings. *SIAM J. Computing*, 27(4):972–992.

Setubal, J. and Meidanis, J. (1997). *Introduction to Computational Molecular Biology*. PWS Publishing Company.