Trabalho Final
Ambientes em Computação

Fabiano Santana, Helena L. Costa, Rossana O. Souza, Thaiane G. Nascimento

[1]Instituto de Ciências Exatas – Universidade Federal de Minas Gerais (UFMG)

{fabianocs7, helenalc, rossanasouza, thaianegn}@ufmg.br

1. Introdução

No contexto da bioinformática o alinhamento de sequências é uma ténica fundamental. Ela permite comparar duas ou mais sequências, o que permite a análise de padrões e homologias entre diferentes organismos. Tais padrões podem estar relacionados a diferenciadas funções dos genes, podem relacionar-se com similaridade entre os organismos, evidenciando relacionamento evolutivo, dentre outros (Durbin et al. 1998).

As sequências podem ser alinhadas de forma local ou global. No caso do alinhamento global, busca-se a maior sub-sequencia comum entre as duas sequências, permitindo inserções e deleções e fazendo com que o alinhamento se extenda para todo o comprimento das sequências.

As sequências biológicas são analisadas na forma de strings e, portanto, tais dados devem ser comparados por meio da função score, que determina as semelhanças ou diferenças entre as sequências. A partir da determinação da estrutura da função score e seus pesos é necessária a utilização de técnicas computacionais para obtenção de um alinhamento entre as sequências que otimize a função score.

Um dos algoritmos mais tradicionais para a finalidade de alinhamento global é o Needleman-Wunsch. Ele utiliza técnicas de programação dinâmica, com custo de ordem $O(n^2)$ (Sedgewick and Wayne 2011), pela sua complexidade quadrática.

Dessa forma, neste documento é abordado o conceito de programação dinâmica e sua aplicação ao algoritmo de Needleman Wunch para resolver o problema de alinhamento de sequências no contexto da bioinformática.

2. Programação Dinâmica e Alimento de sequências

A programação dinâmica é uma técnica de otimização usada para resolver problemas complexos quebrando-os em subproblemas menores e mais simples. A ideia central é evitar a recomputação de subproblemas, armazenando seus resultados para uso futuro. Isso é feito através de uma abordagem de ”memorização”ou ”armazenamento em cache”, onde as soluções para os subproblemas são registradas em uma estrutura de dados, como uma matriz ou tabela. Dessa forma, quando um subproblema é encontrado novamente, o algoritmo pode simplesmente recuperar o resultado armazenado, economizando tempo computacional significativo.

A aplicação da programação dinâmica requer duas propriedades fundamentais: subproblemas sobrepostos e uma estrutura ótima de subproblemas. Subproblemas sobrepostos significam que o problema pode ser decomposto em subproblemas menores, que são repetidamente resolvidos. A estrutura ótima de subproblemas implica que a

solução ótima do problema completo pode ser construída a partir das soluções ótimas de seus subproblemas. A programação dinâmica é amplamente aplicada em áreas como otimização, bioinformática, e teoria dos jogos, entre outras, devido à sua eficiência em resolver problemas complexos de forma prática.

A programação dinâmica também pode ser utilizada em algoritmos de alinhamento de sequencias, uma vez que facilita resolver de forma eficiente programas de alta complexidade. Ao fazer o armazenamento de resultados para resolutação de subproblemas, essa técnica de programação otimiza o processo de alinhamento e pode ser encontrado em algoritmos clássicos, como o de Needleman Wunch (Cormen et al. 2009).

3. Algoritmo de Needleman Wunch

Proposto na década de 1970 por Saul Needleman e Christian Wunsch, realiza um alinhamento global entre um par de sequências. É importante salientar que este algoritmo não foi criado especificamente para esta finalidade, mas foi adaptado e hoje é considerado o algoritmo mais importante da bioinformática.

Para iniciar esse algoritmo é criado uma matriz mxn, onde m é tamanho da primeira sequência e n é o tamanho da segunda sequência que iremos alinhar. Na matriz, cada célula representa a pontuação do alinhamento entre as subsequências das duas sequências que estão sendo comparadas. As bordas da matriz são inicializadas com pontuações de gap. Isso representa o custo de inserir e deletar para alinhar as sequências.

O restante da matriz é preenchido calculando a pontuação de cada célula com base em três possíveis movimentos:

- Match: Comparação entre os caracteres correspondentes das duas sequências. Se forem iguais, a pontuação é aumentada.
- Mismatch: Comparação entre os caracteres correspondentes das duas sequências. Se forem diferentes, a pontuação é diminuída.
- Gap: Introdução de um gap (inserção ou deleção) em uma das sequências. Isso aumenta a pontuação.

Dessa forma, a matriz de pontuação é preenchida com valores que representam a similaridade entre as subsequências das duas sequências que estão sendo comparadas. Cada célula contém a pontuação do melhor alinhamento entre as subsequências até essa posição. Após preencher a matriz, rastreamos de volta da célula inferior direita até a célula superior esquerda para encontrar o caminho ótimo. Durante o rastreamento, registramos as operações realizadas (*match, mismatch, gap*) para construir o alinhamento das duas sequências.

A programação dinâmica está aplicada na subpartição do problema principal em problemas menores e reutilização destes, reduzindo o processamento redundante e fazendo uso eficiente da memória. O algoritmo particiona o problema principal em três principais subproblemas: inicialização da matriz, preenchimento da matriz e alinhamento, também chamado de *traceback*.

4. Implementação

O algoritmo de Needleman Wunch, completo, comentado e explicado pode ser encontrado no seguinte repositório do github: `https://github.com/`

rossanaoliveirasouza/ambientes-computacao. O Código 1 a seguir é uma versão sem comentários do algoritmo de Needleman Wunch.

```
def needleman_wunsch(seq1, seq2, igual=5, diferente=-2, lacuna=-6):

    linhas = len(seq1) + 1
    colunas = len(seq2) + 1
    matriz = [[0] * colunas for _ in range(linhas)]

    for linha in range(linhas):
        matriz[linha][0] = linha * lacuna

    for coluna in range(colunas):
        matriz[0][coluna] = coluna * lacuna

    for i in range(1, linhas):
        for j in range(1, colunas):
            alinhar = matriz[i-1][j-1] +
                    (igual if seq1[i-1] == seq2[j-1] else diferente)
            deletar = matriz[i-1][j] + lacuna
            inserir = matriz[i][j-1] + lacuna
            matriz[i][j] = max(alinhar, deletar, inserir)

    alinhamento_1, dalinhamento_2 = "", ""

    linha, coluna = linhas - 1, colunas - 1

    while linha > 0 and coluna > 0:
        if matriz[linha][coluna] == matriz[linha-1][coluna-1] +
        (igual if seq1[linha-1] == seq2[coluna-1] else diferente):
            alinhamento_1 = seq1[linha-1] + alinhamento_1
            dalinhamento_2 = seq2[coluna-1] + dalinhamento_2
            linha -= 1
            coluna -= 1

        elif matriz[linha][coluna] == matriz[linha-1][coluna] + lacuna:
            alinhamento_1 = seq1[linha-1] + alinhamento_1
            dalinhamento_2 = "-" + dalinhamento_2
            linha -= 1

        else:
            alinhamento_1 = "-" + alinhamento_1
            dalinhamento_2 = seq2[coluna-1] + dalinhamento_2
            coluna -= 1


    print("Alinhamento 1:", alinhamento_1)
    print("Alinhamento 2:", dalinhamento_2)

arquivo = open('seq1.txt')
seq1 = arquivo.readline() #TGCTCGTA

arquivo = open('seq2.txt')
seq2 = arquivo.readline() #TTCATA

needleman_wunsch(seq1, seq2)
```

Código 1. Implementação do Algoritmo de Needleman Wunch

5. Conclusão

O Algoritmo de Needleman Wunch é um problema central da bioinformática e foi desenvolvido há mais de 50 anos. Embora surjam diversas alternativas mais recentes para alinhamento de sequências, o fato dele possuir uma abordagem de programação com uma solução ótima, faz ele continuar sendo considerado o principal algoritmo utilizado na bioinformática até os dias atuais. Com a identificação de regiões similares entre duas sequências, nos permite inferir relações funcionais, estruturais e evolutivas em RNA, DNA e proteínas de organismos de diferentes espécies. Por isso, sem dúvidas, o torna uma técnica de extrema importância na bioinformática.

Referências

[Cormen et al. 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT press, Cambridge, MA, 3rd edition.

[Durbin et al. 1998] Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press.

[Sedgewick and Wayne 2011] Sedgewick, R. and Wayne, K. (2011). *Algorithms*. Addison-Wesley, Boston, MA, 4th edition.