UNIVERSIDADE ESTADUAL PAULISTA

INSTITUTO DE BIOCIÊNCIAS, LETRAS E CIÊNCIAS EXATAS

RELATÓRIO DE BIOINFORMÁTICA

IMPLEMENTAÇÃO E TESTES DOS ALGORITMOS NEEDLEMAN-WUNSCH E SMITH-WATERMAN

Autores

Lorraine Maria Pepe

Everton de Melo Camacho

Prof. Dr. Geraldo F. D. Zafalon

27 de junho de 2019

São José do Rio Preto

1. INTRODUÇÃO

Nas últimas décadas, com a massiva produção de dados biológicos tornou-se necessário o aprimoramento dos algoritmos, estatísticas e outras técnicas matemáticas existentes para obtenção, identificação e análise desses dados.

Entretanto, algumas técnicas mais antigas são utilizadas até os dias de hoje. É o caso dos algoritmos Needleman-Wunsch e Smith-Waterman, para alinhamento global e local, respectivamente, de sequências de aminoácidos ou de proteínas.

Esse algoritmos, por serem determinísticos, garantem que seus resultados sejam os melhores e mais precisos, pois levam em conta toda a extensão das cadeias tomadas como entrada no cálculo do alinhamento.

2. OBJETIVO E METODOLOGIA

Dada a importância desses algoritmos, esse trabalho consiste no estudo, implementação e testes de ambos os algoritmos para três tipos diferentes biossequências, a fim de permitir uma análise de seus funcionamentos.

Para tal, será utilizada a linguagem Scilab, linguagem de alto nível voltada à análise numérica, principalmente para aplicações científica. Foi desenvolvida em 1990 pelos pelos pesquisadores do INRIA *(Institut National de Recherche en Informatique et en Automatique*) e do ENPC (*École Nationale des Ponts et Chaussées*), com sua sintaxe inspirada em Matlab. Atualmente é mantida pelo Scilab Enterprises.

Scilab constitui uma linguagem interpretada, não tipada. Além disso, permite que códigos extensos em linguagens como C, C++ ou Fortran sejam escritas com poucas linhas de código. Também possui uma sofisticadas estruturas de dado com suporte à cálculos matemáticos.

Após a implementação de tais algoritmos, três biossequências serão submetidas como entrada, uma por vez, em cada um deles, e será medido o tempo de execução que os algoritmos levarão para calcular o alinhamento.

3. IMPLEMENTAÇÃO DOS ALGORITMOS

Duas sequências genômicas podem ser comparadas com base em alinhamentos globais e locais. Esses alinhamentos visam confrontar duas ou mais sequências a fim de encontrar trechos homólogos, de forma indicar a localização de regiões conservadas ou mutadas, que auxilie o reconhecimento de padrões e apoie a tomada de decisão dos pesquisadores.

Para a realização de alinhamentos, cada caracter de uma sequência é emparelhado com um caracter da outra sequência ou um espaço nela inserido, de modo conveniente à detecção de maior similaridade entre as cadeias.

Um alinhamento é dito ótimo quando obtém-se a pontuação máxima de similaridade entre as duas sequências analisadas. No alinhamento global, são consideradas ambas as

sequências por completo. Essa forma de alinhamento é considerada útil para análises de sequências com maior grau de similaridade previamente conhecido, por exemplo na comparação de proteínas homólogas.

Já o alinhamento local, alinha pares de subsequências das cadeias das cadeias originais. Logo, é útil em situações em que se deseja identificar regiões altamente conservadas entre os genomas.

Assim, nesta seção será discutida as implementações dos algoritmos Needleman-Wunsch para alinhamento global e Smith-Waterman para alinhamento local. É importante salientar que ambos os algoritmos atuam de acordo com a programação dinâmica, isto é, a solução ótima é procurada com base nos valores anteriormente calculados. Além disso, trabalham com somente duas sequências por vez, o que gera uma complexidade O (n^2).

3.1. NEEDLEMAN-WUNSCH (NW)

Proposto em 1970 por Saul Needleman e Christian Wunsch, o NW calcula uma matriz de pontuação de forma a atribuir pesos maiores quando houver correspondência entre os caracteres (*matches),* pesos menores para não correspondências (*mismatches*) e iguais ou menores ainda para comparações entre caracteres e espaços (*gaps*) inseridos para melhorar o alinhamento.

Inicialmente o algoritmo insere um gap no começo de ambas a sequências e após cria uma matriz com n caracteres + 1 (do *gap)* linhas e m +1 colunas, tal que n e m são o tamanho de cada sequência.

Posteriormente, é feito o cálculo da primeira linha e primeira coluna que servirão de base para os demais cálculos. Assim, cada posição da linha é preenchida com sua posição vezes o valor de *gap*. O mesmo ocorre para a coluna.

Então, percorre-se a matriz por linhas, preenchendo os demais valores. Para isso, segue-se a seguinte função:

Si,j = max {

S (i-1, j-1) + match/mismatch; //valor da diagonal + match ou mismatch

S (i, j-1) + gap; //linha anterior + gap

S (i-1, j)+ gap; //coluna anterior + gap

}

Após preenchida a matriz, toma-se o último elemento da matriz e faz-se a verificação de qual elemento o originou, de forma a anotar desde o ínicio os caracteres correspondentes a linha e coluna, da direita para a esquerda. Toma-se então o elemento encontrado e faz-se o mesmo até que se chegue à borda da matriz, obtendo com isso o alinhamento requerido. Essa etapa é conhecida como *backtracking* ou *trace back,* que significa retrocesso.

Por exemplo na matriz abaixo, parte-se de 13 e busca qual elemento ao seu redor o originou. Pela função dada, percebemos ser o 8. Assim, anota-se os elemento da seguinte forma:

← (da direita para a esquerda).

A T

C G

_____

- 3 + -3 = -6 , considerando mismatch = -3.

	A	**T**
C	8	4
G	4	13

O próximo passo é atribuir valores de *match, mismatch* e *gap* para as sequências obtidas. Com isso, no exemplo acima obtemos -3 e -3 que, somados, resulta -6 como valor ótimo do alinhamento dado. Note que as sequências obtidas e alinhadas (acima da tabela) também fazem parte da solução, já que representam de fato o alinhamento obtido.

É importante salientar que em caso de empate é necessário seguir todas as possibilidades de caminhos, de forma a exibir como saída todos os alinhamentos, já que para biólogos e pesquisadores esses alinhamentos podem possuir alta relevância. Além disso, caminhos que partem de uma mesma raiz resultam sempre uma mesma pontuação.

No Scilab, para a medição de tempo foram utilizadas as funções tic () e toc(), em que tic() inicia a contagem do relógio e toc() finaliza, como em um cronômetro.

3.2. SMITH-WATERMAN (SW)

Esse algoritmo foi proposto em 1981 por Temple F. Smith e Michael S. Waterman. Foi baseado no algoritmo Needleman-Wunsch, isto é, também com as características de programação dinâmica, complexidade quadrática e alinhamento de apenas duas sequências por vez. Entretanto, esse alinhamento é local, realçando subsequências similares das cadeias.

Inicialmente em sua implementação, primeira linha e coluna são zeradas, não sendo necessário cálculos como no NW.

Após isso, percorrendo a matriz de pontuação por linhas, aplica-se a seguinte função:

Si,j = max {

0 // Inibe valores negativos na matriz.

S (i-1, j-1) + match/mismatch; //valor da diagonal + match ou mismatch

S (i, j-1) + gap; //linha anterior + gap

S (i-1, j)+ gap; //coluna anterior + gap

}

Posterior ao preenchimento, é realizado o *backtracking*, à moldes do NW. Todavia, nesse algoritmo parte-se do maior valor existente na matriz e não do último como no NW. Essa abordagem dá a característica de alinhamento local do algoritmo, já que permite considerar apenas partes relevantes das sequências.

Também nesse algoritmo é necessário dar como saída todos os caminhos possíveis a partir do maior valor da matriz, (ou maiores, pois pode haver mais de um valor igual). Ademais, permanece a propriedade de que caminhos que partem de uma mesma posição devem resultar valores iguais.

Para a medição de tempo foram utilizadas as mesmas funções mencionadas na subseção anterior.

4. TESTES E RESULTADOS

Foram realizados seis testes no total. Três com a execução do NW e o restante com SW. Em cada teste com o NW foi utilizado um dos arquivos contendo as sequências a serem testadas. O mesmo para SW. Com isso, mediu-se o tempo para obtenção da pontuação ótima em cada execução.

Os testes e seus respectivos arquivos utilizados se deram da seguinte forma:

a) Teste 1: arquivo Teste1.txt
b) Teste2: arquivo Teste2.txt
c) Teste3: arquivo HomoSapiens_Teste.txt

Desse modo, na Tabela 1 é possível encontrar a pontuação ótima obtida em cada um dos alinhamentos. Já na Tabela 2, encontram-se as medições de tempo de execução de cada um dos testes.

Tabela 1 - Pontuação ótima encontrada para cada teste.

	Teste 1	Teste 2	Teste 3
Needleman-Wunsch	2676	212	1687
Smith-Waterman	2676	238	2377

Tabela 2 - Tempo de execução para cada teste em segundos

	Teste 1	Teste 2	Teste 3
Needleman-Wunsch	5.666895 s	0.646888 s	96.630952 s
Smith-Waterman	7.112449 s	0.767770 s	114.2909566 s

Com os resultados obtidos e levando em conta as propriedades de cada algoritmo, foi possível perceber que quanto maior a sequência, em geral, maior o tempo de processamento e memória utilizada para armazenamento das matrizes.

Todavia, é claro que em sequências de tamanhos similares o tempo para cálculo do *backtracking* pode ser menor na sequência de maior tamanho e com isso resultar tempo de processamento menor.

No caso do Teste 3 as sequências utilizadas possuíam tamanho muito maior que as utilizadas nos dois primeiros testes de cada algoritmo. Por isso o tempo de execução muito maior.

Confirmou-se também que esses algoritmos seriam inviáveis para alinhamentos de múltiplas sequências, visto que a cada outra sequência utilizada causaria um aumento de um grau em suas complexidades.

5. CONCLUSÃO

Mesmo com a evolução dos algoritmos para alinhamento de sequências genômicas, alguns dos primeiros algoritmos propostos ainda são amplamente utilizados ou servem de base para os atuais. É o caso dos algoritmos NW e SW, por serem robustos em seus cálculos, isto é, serem algoritmos determinísticos, considerando toda as sequências.

Entretanto, cada um deles foi proposto para fins diferentes. O NW realiza alinhamentos globais de sequências, enquanto que o SW faz o alinhamento local. Dessa forma, NW calcula o alinhamento ótimo considerando a sequência como um todo, já SW retorna as subsequências mais similares, realçando esses locais ao impedir valores negativos na matriz e ao começar o *trace back* pelo maior valor disponível.

Nesse trabalho, então, foram implementados ambos os algoritmos na linguagem Scilab, no qual foram testados três arquivos em cada um, em execuções distintas, contendo duas biossequências por arquivo. Além disso, mediu-se o tempo de processamento para obtenção da pontuação ótima de cada teste.

Com os resultados obtidos foi possível notar que, em linhas gerais, cadeias maiores demoram mais a serem processadas e consomem maior espaço de memória para armazenar os dados calculados.

De forma geral, as dificuldades enfrentadas para implementação foi conhecer um pouco melhor a sintaxe da linguagem e entender os erros exibidos no ambiente. No

mais, são algoritmos de fácil implementação e entendimento, o que agrega ao fato de ainda serem utilizados.

6. REFERÊNCIAS

NEEDLEMAN, Saul B.; WUNSCH, Christian D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of molecular biology, v. 48, n. 3, p. 443-453, 1970.

SMITH, Temple F. et al. Identification of common molecular subsequences. Journal of molecular biology, v. 147, n. 1, p. 195-197, 1981.

CAMPBELL, Stephen L.; CHANCELIER, Jean-Philippe; NIKOUKHAH, Ramine. Modeling and Simulation in SCILAB. Springer New York, 2006.

Apêndice A - Códigos

Needleman-Wunsch

```
1ºsemestre/2019
  Implementação do algoritmo Needleman-Wunsch
  Linguagem: Scilab.

  Integrantes: Everton Camacho
               Lorraine Pepe
*/
[nome_seq_1, seq_1, tam_seq_1, nome_seq_2, seq_2, tam_seq_2]= ler_fasta();

function [res_seq_1, res_seq_2, score, trace_back, soma_pontos, tempo_gasto] =
needleman_wunsh();
   //Insere o - nas sequências
   seq_1 = '-' + seq_1;
   seq_2 = '-' + seq_2;

   //Inicializa a matriz toda com zeros
   n = tam_seq_2 + 1; //insere a linha do gap (-)
   m = tam_seq_1 + 1; //insere a coluna do gap (-)
   score = zeros(n,m);
   trace_back = zeros(n,m);

   match = 5;
   mismatch = -3;
   gap = -4;
```

```
//Needleman-Wunsch

tic(); //para medir o tempo gasto
//preenche a primeira linha
k = 1;
for i = 2:n
   score(i,1) = gap * k;
   k = k + 1;
end

//preenche a primeira coluna
k = 1;
for j = 2:m
   score(1,j) = gap * k;
   k = k + 1;
end

//prepara a matriz trace_back
for i = 2:n
   trace_back(i,1) = 3;
end

for i = 2:m
   trace_back(1,i) = 2;
end

seq_1 = mstr2sci(seq_1); //converte a string para um vetor de caracteres
seq_2 = mstr2sci(seq_2); //converte a string para um vetor de caracteres

//preenche a matriz de acordo com o algoritmo Needleman-Wunsch
for i = 2:n
   for j = 2:m
      //percorre as duas sequências e verifica match, mismatch ou gap
      if (seq_1(j) == seq_2(i)) then //ocorreu match
         caso_1 = score(i-1,j-1) + match;
      else //ocorreu mismatch
         caso_1 = score(i-1,j-1) + mismatch;
      end

      caso_2 = score(i,j-1) + gap;
      caso_3 = score(i-1,j) + gap;

      //verifica o máximo entre os três casos e coloca na matriz de pontuação
      [score(i,j),max_index] = max(caso_1, caso_2, caso_3);

      //mapeia cada ponto para saber sua origem (1 se voltou pela diagonal, 2 se
voltou pela esquerda, 3 se voltou por cima).
      trace_back(i,j) = max_index;
```

```
    end
end

nova_sequencia_1 = "";
nova_sequencia_2 = "";

i = n;
j = m;

while(i >= 1 && j >= 1)
    if(trace_back(i,j) == 1) then //volta pela diagonal
       nova_sequencia_1 = nova_sequencia_1 + seq_1(j);
       nova_sequencia_2 = nova_sequencia_2 + seq_2(i);
       i = i-1;
       j = j-1;
    elseif(trace_back(i,j) == 2) then //volta pela esquerda
        nova_sequencia_1 = nova_sequencia_1 + seq_1(j);
        nova_sequencia_2 = nova_sequencia_2 + "-";
        j = j-1;
    elseif(trace_back(i,j) == 3) then //volta por cima
        nova_sequencia_1 = nova_sequencia_1 + "-";
        nova_sequencia_2 = nova_sequencia_2 + seq_2(i);
        i = i-1;
    end
    if(trace_back(i,j) == 0) then //acaba no trace_back(1,1)
       break;
    end
end

//coloca as novas sequências na ordem certa
res_seq_1 = strrev(nova_sequencia_1);
res_seq_2 = strrev(nova_sequencia_2);

i = length(res_seq_2);
j = length(res_seq_1);

res_seq_2 = mstr2sci(res_seq_2);
res_seq_1 = mstr2sci(res_seq_1);

soma_pontos = 0;
//calcula o score
while(i >= 1 && j >= 1)
   if(res_seq_1(j) == res_seq_2(i)) then
      soma_pontos = soma_pontos + match;
   elseif(res_seq_1(j) == "-" || res_seq_2(i) == "-") then
      soma_pontos = soma_pontos + gap;
   else
      soma_pontos = soma_pontos + mismatch;
```

```
      end
      i = i - 1;
      j = j - 1;
   end

   tempo_gasto = toc();
endfunction

clc
[res_seq_1, res_seq_2, score, trace_back, soma_pontos, tempo_gasto] =
needleman_wunsh();

//exibe os resultados
disp(score);
disp(trace_back);

printf("\n:Alinhamento");

disp(res_seq_1);
disp(res_seq_2);

printf("\nPontos: %d\n", soma_pontos);
printf("\nTempo gasto: %f s\n", tempo_gasto);
```

Smith-Waterman

```
/* Trabalho de Bioinformática - 1ºsemestre/2019
  Implementação do algoritmo Smith-Waterman
  Linguagem: Scilab

  Integrantes: Everton Camacho
               Lorraine Pepe
*/

[nome_seq_1, seq_1, tam_seq_1, nome_seq_2, seq_2, tam_seq_2]= ler_fasta();

function [res_seq_1, res_seq_2, score, trace_back, soma_pontos, tempo_gasto] =
smith_waterman();
   //Insere o - nas sequências
   seq_1 = '-' + seq_1;
   seq_2 = '-' + seq_2;

   //Inicializa a matriz toda com zeros
   n = tam_seq_2 + 1; //insere a linha do gap (-)
   m = tam_seq_1 + 1; //insere a coluna do gap (-)
```

```
score = zeros(n,m);
trace_back = zeros(n,m);

match = 5;
mismatch = -3;
gap = -4;

//Smith-Waterman

tic(); //para medir o tempo gasto

seq_1 = mstr2sci(seq_1); //converte a string para um vetor de caracteres
seq_2 = mstr2sci(seq_2); //converte a string para um vetor de caracteres

//preenche a matriz de acordo com o algoritmo Needleman-Wunsch
i_max = 0;
j_max = 0;
valor_max = 0;
for i = 2:n
   for j = 2:m
      //percorre as duas sequências e verifica match, mismatch ou gap
      if (seq_1(j) == seq_2(i)) then //ocorreu match
         caso_1 = score(i-1,j-1) + match;
      else //ocorreu mismatch
         caso_1 = score(i-1,j-1) + mismatch;
      end

      caso_2 = score(i,j-1) + gap;
      caso_3 = score(i-1,j) + gap;
      caso_4 = 0;

      //verifica o máximo entre os três casos e coloca na matriz de pontuação
      [score(i,j),max_index] = max(caso_1, caso_2, caso_3, caso_4);

      if(score(i,j) > valor_max) then
         valor_max = score(i,j);
         i_max = i;
         j_max = j;
      end

      //mapeia cada ponto para saber sua origem (1 se voltou pela diagonal, 2 se
voltou pela esquerda, 3 se voltou por cima, 4 se for zero).
      trace_back(i,j) = max_index;
   end
end

nova_sequencia_1 = "";
nova_sequencia_2 = "";
```

```
   //começa a percorrer a partir do maior valor
   i = i_max;
   j = j_max;
   while(1)
      if(trace_back(i,j) == 1) then //volta pela diagonal
        nova_sequencia_1 = nova_sequencia_1 + seq_1(j);
        nova_sequencia_2 = nova_sequencia_2 + seq_2(i);
        i = i-1;
        j = j-1;
      elseif(trace_back(i,j) == 2) then //volta pela esquerda
         nova_sequencia_1 = nova_sequencia_1 + seq_1(j);
         nova_sequencia_2 = nova_sequencia_2 + "-";
         j = j-1;
      elseif(trace_back(i,j) == 3) then //volta por cima
         nova_sequencia_1 = nova_sequencia_1 + "-";
         nova_sequencia_2 = nova_sequencia_2 + seq_2(i);
         i = i-1;
      end
      if(trace_back(i,j) == 0 || trace_back(i,j) == 4) then //acaba quando encontrar o
primeiro valor zero
         break;
      end
   end

   //coloca as novas sequências na ordem certa
   res_seq_1 = strrev(nova_sequencia_1);
   res_seq_2 = strrev(nova_sequencia_2);

   i = length(res_seq_2);
   j = length(res_seq_1);

   res_seq_2 = mstr2sci(res_seq_2);
   res_seq_1 = mstr2sci(res_seq_1);

   soma_pontos = 0;
   //calcula o score
   while(i >= 1 && j >= 1)
     if(res_seq_1(j) == res_seq_2(i)) then
        soma_pontos = soma_pontos + match;
     elseif(res_seq_1(j) == "-" || res_seq_2(i) == "-") then
        soma_pontos = soma_pontos + gap;
     else
        soma_pontos = soma_pontos + mismatch;
     end
     i = i - 1;
     j = j - 1;
   end
```

```
    tempo_gasto = toc();
endfunction

clc
[res_seq_1, res_seq_2, score, trace_back, soma_pontos, tempo_gasto] =
smith_waterman();

//exibe os resultados
disp(score);
disp(trace_back);

printf("\n:Alinhamento");

disp(res_seq_1);
disp(res_seq_2);

printf("\nPontos: %d\n", soma_pontos);
printf("\nTempo gasto: %f s\n", tempo_gasto);
```