Algoritmos e Heurísticas para Sequenciamento de DNA

GERALDO FERREIRA, TIAGO MACAMBIRA
{geraldof,tmacam}@dcc.ufmg.br
Departamento de Ciência da Computação da UFMG

1. Resumo

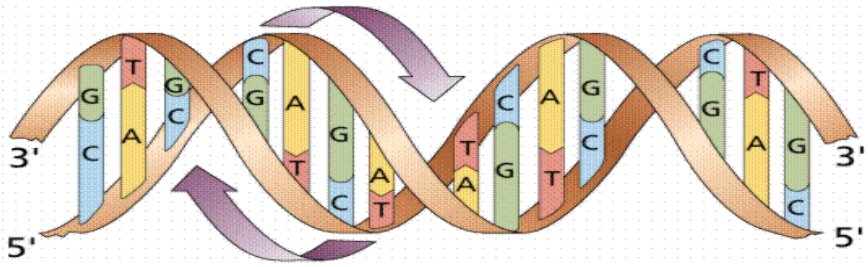
O DNA é considerada a mais importante molécula simples das células vivas e contém todas as informações sobre quem somos e qual a nossa aparência. A composição química do DNA foi descoberta em 1869 por Friederich Miescher na Alemanha. Francis Crick e James Watson descobriram a estrutura de hélice dupla do DNA em 1953 nos Estados Unidos. Desde então o estudo da genética sofreu um grande avanço, principalmente nas 3 últimas décadas quando o computador passou a ser usado como ferramenta na análise das seqüências das bases do DNA. Este texto visa mostrar os conceitos e as técnicas envolvidos na análise de seqüências de DNA. Os 3 artigos nos quais nos baseamos são considerados clássicos na Biologia Computacional. São eles:

[1] escrito por Needleman-Wunsch em 1970 que trata de alinhamento global,
[2] escrito por Smith-Waterman em 1981 que trata de alinhamento local e
[3] escrito por Altschul et al em 1990 que trata do BLAST (Basic Local Alignment Search Tool).

2. Introdução

O corpo humano é composto por bilhões de células. Dentro do núcleo de cada uma destas células está armazenado um conjunto único de 46 cromossomos. Estes cromossomos contém, como uma mola comprimida, uma longa molécula de DNA. Seções do DNA que contém mensagens completas são chamadas genes. Podemos imaginar os genes como sendo "palavras" presentes nas "sentenças" DNA. Um gene armazena a fórmula (código) necessária para a célula produzir uma proteína. Quase todo o corpo humano é composto por proteínas. proteínas estruturais formam os cabelos, os músculos e a pele.

O DNA é composto por pequenas moléculas chamadas nucleotídeos. Os nucleotídeos são 4 em número e podem ser diferenciados por 4 bases: adenina (A), timina (T), guanina (G) e citosina (C). Portanto o alfabeto das bases do DNA pode ser representado por a = {A,T,G,C}. Abaixo a representação em hélice dupla de uma seqüência de DNA, proposta por Francis Crick e James Watson.



Atualmente a comparação de seqüências é um dos mais importantes problemas da biologia computacional dado o número e diversidade de seqüências existentes (não só de DNA mas também de RNA, proteínas e de outras) e dada a frequência que ele precisa ser resolvido diariamente em todo o mundo. Todos os padrões de seqüências que buscamos comparar são baseados na evolução. seqüências homólogas são relacionadas entre si por terem uma linhagem comum nos genes, devido a um organismo ancestral comum. Já a medida da similaridade tenta inferir quão forte é a homologia entre as seqüências. Em biologia computacional entende-se comparação como sendo a comparação aproximada de seqüências. Ou seja, as seqüências comparadas não precisam ser necessariamente iguais, aceita-se um certo número de erros. Isto se deve ao fato de que em seqüências de DNA, alta similaridade implica em alta similaridade estrutural e consequentemente funcional. Em outras palavras, se *s* é muito parecida com *t* e se sei a função de *t*, então com grande probabilidade *s* terá a mesma função. Segundo R. Doolittle, isto se deve ao fato de que o processo evolutivo

continuamente re-utiliza estruturas que dão certo, "*biologia é baseada em enorme redundância*".

3. Alinhamentos e Programação Dinâmica

Um alinhamento pode ser definido como um arranjo mutual de 2 seqüências que mostra onde as 2 seqüências são similares e onde elas se diferem. Por sua vez, um alinhamento ótimo é aquele que exibe as maiores similaridades e as menores diferenças.

Um alinhamento global é aquele que assume que 2 seqüências são basicamente similares uma sobre a outra durante todo o comprimento. Este alinhamento tenta casar fim-a-fim uma seqüência sobre a outra, mesmo existindo partes do alinhamento que não são convincentes. Abaixo um simples exemplo de um alinhamento global.

```
a a a g c g g a a g t c a c a g
| | . | | .| | | | | | . | |
a a g g c t g a  a g t – a t a g
```

Um alinhamento local é aquele que procura por segmentos das 2 seqüências que tenham um bom casamento. Não existe a tentativa de forçar as seqüências completas a se alinharem, somente aquelas partes que aparentam ter uma boa similaridade, de acordo com algum critério. Abaixo um exemplo de alinhamento local, usando a mesma seqüência.

```
a a a g c g g a a g t c a c a g
. . . . . | | | | |  . . . .
a a g g c t  g a a g t – a  t a g
```

Dadas duas seqüências de DNA *s* de tamanho m e *t* de tamanho n e um sistema de pontuação para similaridade, queremos determinar o alinhamento de *s* e *t* com maior similaridade. Um alinhamento é uma maneira de se colocar *s* sobre *t* de modo a estabelecer uma correspondência entre as bases de *s* e as bases de *t*. Um sistema de pontuação é uma maneira de atribuir notas a pareamentos de bases e espaços. Na sua forma mais geral é uma matriz de elementos P tal que i,j pertence a {A,T,G,C,-}, e P(i,j) é um valor numérico. Por exemplo, P(A,A)=+3 diz que A alinhado com A ganha 3 pontos. Para DNA é usual as seguintes atribuições: P(i,i)= +1 e P(i,j)= -1 para i,j bases distintas e P(i, -)=P(- ,i)= -2 para alinhamentos com espaço. Em geral, num alinhamento entre duas seqüências, o pareamento de caracteres distintos é chamado de substituição enquanto o pareamento entre um caracter com espaço é chamado de inserção ou remoção, dependendo da seqüência de referência. O conjunto de inserções ou remoções recebe o nome de *indels*. Vamos considerar as seguintes seqüências:

```
g a – c g g a t  t a g
| |   | | | | .| | |
g a  t c g g a a t a g
```

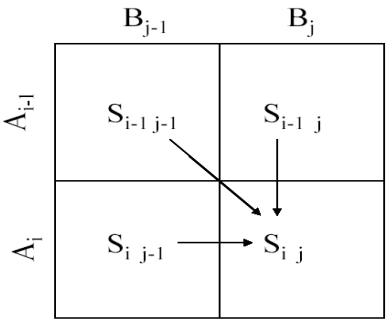
Assumindo as atribuições P(i,i)= +1, P(i,j)= -1 e P(i, -)=P(- ,i)= -2, a pontuação de similaridade para o alinhamento acima seria : [9 x 1] + [1 x (-1)] +[1 x (-2)] = 6.

Uma abordagem que poderia ser usada para calcular a similaridade entre duas seqüências seria gerar todos os possíveis alinhamentos e então pegar o melhor. Entretando, o número de alinhamentos entre 2 seqüências é exponencial, e tal abordagem resultaria num algoritmo intoleravelmente lento. Uma maneira de se contornar este problema é o uso de programação dinâmica. Basicamente a programação dinâmica consiste em resolver uma parte do problema aproveitando-se de soluções já computadas de partes menores do mesmo problema. Dadas 2 seqüências *s* e *t*, ao invés de se calcular a similaridade entre *s* e *t* como somente seqüências completas, construimos a solução determinando-se todas as similaridades entre prefixos arbitrários das 2 seqüências. Começamos com prefixos curtos e usamos os resultados previamente encontrados para resolver o problema com prefixos longos.

4. Alinhamento Global por Programação Dinâmica

O melhor alinhamento terminal num par de bases (de DNA) é o melhor alinhamento das seqüências até o ponto anterior a estas duas bases mais a pontuação máxima para o alinhamento destas duas bases adicionais. Numa matriz de pontuação de similaridade, isto pode ser descrito como:

$$S_{ij} = s_{ij} + \max_{\substack{1\le k<i \\ 1\le l<j}} S_{kl} = \max \begin{cases} S_{i-1\,j-1} + s_{ij} \\ \max\limits_{1<x<i} S_{i-x\,j} + W_x \\ \max\limits_{1<y<j} S_{i\,j-y} + W_y \end{cases}$$



Onde:

S_{ij} = pontuação para alinhamento terminal em i na seqüência 1 e j na seqüência 2.

s_{ij} = pontuação para o alinhamento de i com j.

Wx = pontuação (negativa) para criar um "*gap*"de tamanho x na seqüência 1.

Wy = pontuação (negativa) para criar um "*gap*"de tamanho y na seqüência 2.

Vamos ilustrar esta técnica através de um exemplo. Usando as atribuições Match=5, Mismatch= -4, e Gap w= -7 , desejamos obter a pontuação de similaridade entre as seqüências seguintes:

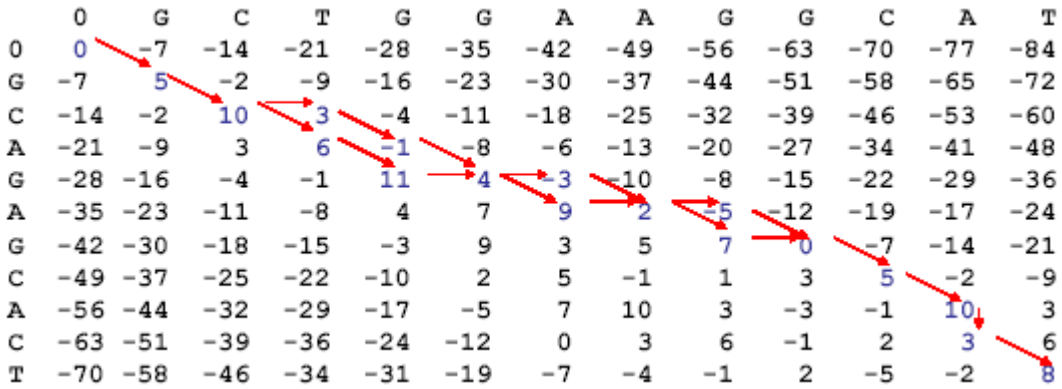
G C T G G A A G G C A T
G C A G A G C A C T

Primeiramente dispomos as seqüências nos eixos x e y, externamente à matriz de similaridade, a partir do canto superior esquerdo. A partir deste mesmo canto, calcula-se os conteúdos (S_{ij}) da matriz de similaridade. Usando os resultados já computados, toda a matriz é preenchida em direção ao canto inferior direito. Abaixo o preenchimento da matriz de similariade até o elemento $S_{2,3}$.

		j=0	1	2	3	4	5	6	7	8	9	10	11	12
i=		-	G	C	T	G	G	A	A	G	G	C	A	T
0	-	0	-7	-14	-21	-28	-35	-42	-49	-56	-63	-70	-77	-84
1	G	-7	5	-2	-9									
2	C	-14	-2	10	$S_{2,3}$									
3	A	-21	-9											
4	G	-28												
5	A	-35												
6	G	-42												
7	C	-49												
8	A	-56												
9	C	-63												
10	T	-70												

$$S_{2,3} = \max \begin{cases} S_{1,2} + s_{C,T} = -2 - 4 = -6 \\ S_{1,3} + w = -9 - 7 = -16 \\ S_{2,2} + w = 10 - 7 = 3 \end{cases} = 3$$

Abaixo a matriz de similaridade totalmente preenchida, quando obtemos o resultado final do alinhamento, pontuação de similaridade S[10,12]=8.



	0	G	C	T	G	G	A	A	G	G	C	A	T
0	0	-7	-14	-21	-28	-35	-42	-49	-56	-63	-70	-77	-84
G	-7	5	-2	-9	-16	-23	-30	-37	-44	-51	-58	-65	-72
C	-14	-2	10	3	-4	-11	-18	-25	-32	-39	-46	-53	-60
A	-21	-9	3	6	-1	-8	-6	-13	-20	-27	-34	-41	-48
G	-28	-16	-4	-1	11	4	-3	-10	-8	-15	-22	-29	-36
A	-35	-23	-11	-8	4	7	9	2	-5	-12	-19	-17	-24
G	-42	-30	-18	-15	-3	9	3	5	7	0	-7	-14	-21
C	-49	-37	-25	-22	-10	2	5	-1	1	3	5	-2	-9
A	-56	-44	-32	-29	-17	-5	7	10	3	-3	-1	10	3
C	-63	-51	-39	-36	-24	-12	0	3	6	-1	2	3	6
T	-70	-58	-46	-34	-31	-19	-7	-4	-1	2	-5	-2	8

Os caminhos mostrados em vermelho na ilustração acima correspondem aos melhores alinhamentos entre as seqüências. O caminho que apresenta as mais altas pontuações leva ao caminho ótimo e ao alinhamento ótimo. As setas diagonais representam um casamento (match) ou um não casamento (mismatch) entre caracteres. Já as setas horizontais ou verticais representam os gaps. Para este nosso exemplo o alinhamento ótimo é mostrado abaixo.

```
G  C  T  G  G  A  A  G  G  C  A  -  T
|  |        |     |     |  |  |     |
G  C  -  A  G  -  A  -  G  C  A  C  T
```

```
8 matches: 8 * 5 = 40
1 mismatch: -4
4 gaps: 4 * -7 = -28
Total: 8
```

Abaixo o pseudo-código para a comparação global de 2 seqüências usando programação dinâmica. Ele apresenta complexidade O (m.n) para calcular $S_{m,n}$. Já para obter o alinhamento (o caminho) precisamos de O (m+n).

```
Align (seq1, seq2)
  Input: string seq1 de tamanho m, string seq2 de tamanho n,
         match e gap scores

      1.    Inicializa S propriamente;
      2.    for i=1 to m // Constroi matriz S linha por linha
      3.      for j=1 to n
                // Inicializa max com o primeiro dos 3 termos
      4.        max = S[i-1][j] + gapScore2 (seq2[j-1])
                // Verifica se o segundo termo é maior
      5.        if max < S[i][j-1] + gapScore1 (seq1[i-1])
      6.          max = S[i][j-1] + gapScore1 (seq2[i-1])
      7.        endif
                // Verifica se o terceiro termo é maior
      8.        if max < S[i-1][j-1] + matchScore (seq1[i-1], seq2[j-1])
      9.          max = S[i-1][j-1] + matchScore (seq1[i-1], seq2[j-1])
      10.       endif
      11.       S[i][j] = max
      12.     endfor
      13.   endfor
      14.   return S[m][n]  // Retorna o valor ótimo do canto inferior direito

  Output: valor do alinhamento ótimo
```

5. Alinhamento Local por Programação Dinâmica

O alinhamento local pode ser encontrado a partir das considerações já feitas para o alinhamento global. As duas pricipais alterações que precisamos implementar são as seguintes:

1. Comece novos alinhamentos. Assim como no alinhamento global, comece no canto superior esquerdo da matriz de similaridades. Entretanto, se um alinhamento tornar-se muito caro descarte o mesmo e recomece um novo.
2. Procure pelo melhor alinhamento local. Assim como no alinhamento global, preencha toda a matriz de similaridade. Entretanto, não assuma o canto inferior direito como o ponto de chegada. Pesquise toda a matriz pelo melhor "canto inferior direito de alinhamento local".

Para atingir 1) precisamos:

- Decidir quando um alinhamento local deve ser abandonado (isto é, quando devemos começar um outro). Tipicamente quando a pontuação (score) cai abaixo de zero.
- Modificar a recorrência para evitar pontuações abaixo de zero. Se zero é a maior pontuação em (i,j), um novo alinhamento é inicializado. Obs: gravar os valores (i,j).

$$S_{i,j} = \max (0, S_{i-1,j-1} + \text{exact-match-score}, S_{i-1,j} + \text{string1-gap-score}, S_{i,j-1} + \text{string2-gap-score})$$

Para atingir 2) precisamos:

- Pesquisar pelo $S_{i,j}$ com a maior pontuação de similaridade. Caminhar de volta no caminho até o inicio deste alinhamento local.

6. Alinhamento Múltiplo

Muitas vezes temos que alinhar não duas mas várias seqüências simultaneamente da melhor forma possível. Nesse caso, precisamos construir um **alinhamento múltiplo** dessas seqüências. Isso acontece, por exemplo, quando queremos compararar proteínas que desempenham papéis similares em espécies diferentes. Nesse caso, nosso interesse é de descobrir que partes da proteína são similares e que partes são diferentes. Na verdade, alinhamentos múltiplos são mais comumente realizados entre cadeias protéicas do que com cadeias de DNA.

O problema do alinhamento múltiplo pode ser visto como uma generalização do problema de alinhamento entre duas cadeias. Mais formalmente, buscamos um alinhamento entre um conjunto de cadeias $s_1, \ldots, s_k$ definidas em um alfabeto comum. De forma similar ao problema do alinhamento de um par de cadeias, no alinhamento múltiplo, procuramos construir o alinhamento através da inserção de espaços em algumas cadeias, de forma a torná-las de mesmo tamanho.

Um problema que temos que lidar quando realizamos esse tipo de alinhamento é o de definir uma métrica para a qualidade do alinhamento. A forma mais natural de fazer isso seria extendender a fórmula de pontuação usada no alinhamento entre pares de cadeias para tratar múltiplas cadeias. Existem ainda outras formas de fazermos a pontuação dos alinhamentos, utilizando estruturas baseadas em árvores. A seguir, analisaremos um pouco de cada uma dessas formas.

6.1 Métrica SP

Quando lidamos com o alinhamento de duplas de cadeias

A forma mais intuitiva de fazer o alinhamento entre várias cadeias é extender os algoritmos e estruturas utilizadas no alinhamento entre duas cadeias:

- extender a função de pontuação para tratar não somente dois mas k argumentos;
- extender o algoritmo de programação dinâmica, ampliando a tabela bidimensional para uma tabela k -dimensional.

6.1.1 Extendendo a função de pontuação

O primeiro passo, a extensão da função de pontuação, consiste em criar uma função de pontuação que tenha não 2 mas k argumentos. Uma métrica muito conhecida que define uma função com essas características é a métrica chamada *sum-of-parts* (SP). Essa métrica tem a característica que, dado alinhamento múltiplo α de k cadeias:

$$\text{score-SP}(\alpha) = \sum_{i<j} \text{score}(\alpha_{ij})$$

Onde α_{ij} é o valor do alinhamento da dupla de cadeias i e j. Isso deve-se a forma com que a métrica SP calcula a pontuação de uma coluna do alinhamento global:

$$\text{score-SP}(a, b, c, d) = p(a, b) + p(a, c) + p(a, d) + p(b, c) + p(b, d) + p(c, d)$$

Onde $p(i, j)$ é o valor da pontuação para as bases i e j usando a função de pontuação usada os casamentos de duplas de seqüências.

6.1.2 Extendendo o algoritmo de programação dinâmica

Para realizar o alinhamento de duplas de seqüências utilizamos uma matriz bi-dimensional. Era de se esperar então que no caso do alinhamento múltiplo entre k seqüências utilizássemos uma martriz k -dimensional para armazenar as pontuações obtidas entre os alinhamentos dos prefixos das k cadeias de n caracteres.

Todavia, essa extensão trivial dos algoritmos de programação dinâmica utilizados anteriormente não resulta em um problema computacionalmente atraente. Na verdade, o problema do alinhamento múltiplo ótimo é $\mathcal{NP}$-completo. Isso é fácil de ser observado. A matriz k-dimensional requer um espaço de armazenamento da ordem de $O(n^k)$. Como cada item dessa tabela é preenchido, temos um custo inical de pelo menos $O(n^k)$. Somados os custos necessários para calcular o valor que vai em cada item da tabela, o custo de acesso aos items, de calcular o valor da função de pontuação SP, dentre outros, chegamos a um custo total de $O(k^2 2^k n^k)$.

Devido a abordagem usando programação dinâmica pura não ser atraente para o uso geral, fazemos uso de heurísticas para diminuir o tempo de processamento e o espaço necessário.

6.2 Alinhamento Estrela

Uma heurística que pode ser utilizada para o alinhamento múltiplo e que possui tempo polinomial é conhecida como *alinhamento estrela*. Essa heurística utiliza uma abordagem que foge do modelo utilizando programação dinâmica pura. A idéia nessa heurística é a de utilizar os algoritmos para alinhamento de pares de cadeias, que são ótimos e possuem tempos polinomiais, para realizar o alinhamento entre uma seqüência fixa e as demais.

Sejam $s_1, ..., s_k$ as seqüências que desejamos alinhar. Escolhemos dentre essas k seqüências uma seqüência s_c para ser o *centro* da estrela. Realizamos então o alinhamento ótimo entre s_c e as todas as demais seqüências, realizando entre os sucessivos alinhamentos uma agregação dos resultados anteriores e os novos resultados obtidos. Ao final do processo teremos um alinhamento entre as k seqüências que possivelmente não

será o ótimo mas será, na maioria dos casos, um alinhamento razoável, a um custo total de $O(kn^2 + k^2l)$, onde l é um limite superior para o tamanho final do alinhamento.

7. BLAST

A evolução das técnicas de seqüênciamento de DNA e de proteínas permitiram a criação de vastos bancos de dados de material genético e protéico seqüenciado. Como decorrência disso, surgiu a necessidade da criação de mecanismos eficientes para pesquisar esses bancos da dados por seqüências similares a uma dada requisição. Todavia, dada a grandeza desses repositórios, os algoritmos lineares conhecidos para realizar busca por similaridade e por alinhamentos ótimos, mesmo tendo complexidade de tempo $O(n^2)$, são proibitivos. Na busca por algoritmos mais rápidos e eficientes para lidar com esse problema, um alternativa muito atraente é o uso de heurísticas.

A despeito da falta, em alguns casos, de um embasamento teórico consistente para modelar a complexidade de tempo e de espaço desses algoritmos, programas baseados neles tornaram-se importantes ferramentas na área e, devido a isso, esturaremos um desses algoritmos, o BLAST [3]

7.1 Terminologia

Antes de darmos início ao estudo do funcionamento do BLAST, descreveremos alguns dos termos e expressões usados as sessões seguintes.

Definimos *segmento* como uma substring de uma seqüência qualquer.

Para uma dada métrica de pontuação de similaridade, um MSP, *Maximal Segment Pair,* é o par de segmentos de tamanhos iguais escolhidos entre duas seqüências com maior pontuação. O BLAST calcula heuristicamemte essa métrica para inferir informações de similaridade local entre duas seqüências.

7.2 Funcionamento

O BLAST, *Basic Local Alignment Search Tool*, procurar alinhar uma seqüência de busca com cada seqüência existente no banco de dados, procurando por segmentos com alto grau de similaridade. Ele tenta listar apenas ocorrências que contenham um segmento tão similar com o padrão de busca tal que a similiaridade entre as duas seqüências sejam estatisticamente significativas. Em outras palavras, ele retira do seu resultado ocorrências onde o casamento tenha acontecido mais por forças do acaso do que por razões evolutivas. Seu funcionamento pode ser sumarizado nas 3 etapas de seu algoritmo:

1. Compilar uma lista de palavras com alta pontuação
2. Buscar casamentos entre e essas palavras e seqüências no banco de dados
3. Extender esses casamentos para formar MSPs

Dependendo do tipo de seqüências comparadas (DNA ou proteínas) algumas etapas do algoritmo serão modificadas. Isso deve-se a características particulares de cada uma delas.

Em seqüências de DNA, por exemplo, dado o limitado alfabeto de 4 letras, o algoritmo comprime cada nuceotídeo em seqüencias de 2 bits para acelerar o processo de busca e o espaço de armazenamento. Outra característica desse tipo de seqüências é que elas não são muito randômicas, apresentado áreas onde encontramos uma forte concentração de um determinado conjunto de bases (áreas ricas em bases A e T) e de repetição de subcadeias (seqüências *Alu*). Isso tem importantes conseqüências para qualquer algoritmo de busca por padrões em cadeias de DNA. O BLAST leva em esse fato em consideração nas suas heurísticas.

7.2.1 Compilação da lista de palavras

O primeiro passo do algoritmo consiste em gerar uma lista de palavras de tamanho m (*w-mers*). No caso de seqüências de proteínas essa lista de *w-mers* é composta por todos os *w-mers* que tenham uma pontuação de similaridade com a seqüencia de busca de pelo menos T, onde T é um parâmetro do algoritmo.

7.2.2 Busca por casamentos

O BLAST possui duas abordagens para realizar a busca por casamentos entre a lista de palavras construídas no passo anterior e as seqüencias armazenadas no banco de dados:

- A primeira utiliza uma hash table para rapidamente mapear cada palavra (*w-mers*) da seqüencia pesquisada em uma palavra da lista.
- A segunda usa uma abordagem baseada em autômatos finitos determinísticos (DFA). Essa é a abordagem preferida dos autores.

7.2.3Extendendo os casamentos

Uma vez encontrado um casamento, o BLAST procura efetuar a extensão desse casamento para formar um MSP local. Para economizar tempo, o processo é encerrado assim que a pontuação dessa extensão for inferior à melhor pontuação de um segmento de tamanho inferior. Isso não garante de maneira alguma a qualidade dos "MSPs" encontrados mas, segundo os autores do BLAST, essa heurística possui um erro *negligenciável*.

8. Considerações Finais

Neste texto apresentamos um estudo sobre as principais técnicas e algorítmos utilizados em biologia computacional. Esses algorítmos são ferramentas básicas utilizadas nos mais variados campos dessa área do conhecimento: de seqüenciamento à busca por estruturas morfologicamente homólogas em espécies distintas.

Apesar de existirem algoritmos polinomiais para tratar alguns problemas, vários outros possuem apenas algoritmos exponenciais. Muito do trabalho teórico nessa área concentrará seu esforço no desenvolvimento de novas e mais eficientes heurísticas e algorítmos aproximativos para tratrarem desses problemas, tanto sobre o aspecto da complexidade de tempo quanto na melhoria das garantias oferecidas pelas heurísticas. O uso de paralelização nesse contexto apresenta-se como um alternativa atraente para o tratamento desses problemas.

9. Referências

[1] Needleman, S.B., Wunsch, C.D.: "*A general method applicable to the search for similarities in the amino acid sequence of two proteins."* J. Mol. Biol. Vol.48, pp.443-453 (1970).

[2] Smith, T.F., Waterman, M.S.: *"Identification of common molecular subsequences."* J. Mol. Biol. Vol.147, pp.195-197 (1981).

[3] Altschul, S. F., Gish, W, Miller, W., Myers, E. W, and Lipman,D. J.;. *"Basic Local Alignment Search Tool"*. Journal of Molecular Biology, 215:403--410, 1990.

[4] Setubal, J and Meidanis, J. *"Introduction to computational molecular biology."* PWS Pub., 1997. ISBN: 0534952623