ALGORITMOS DE ALINHAMENTO DE SEQÜÊNCIAS MOLECULARES

MOLECULAR SEQUENCES ALIGNMENT ALGORITHMS

E. Bilha, E. di Grazia, L.T. Ono, M.R. Cardoso, M.C. Smynniuk, L.C. Rozante

edgar_bilha@yahoo.com.br

RESUMO

Com o advento e o crescimento da bioinformática, em especial, com o seqüenciamento do genoma de vários organismos, inclusive do homem, os bancos de bioseqüências cresceram em tamanho e número. Isto levou à necessidade de novas técnicas métodos de análise e tratamento destas informações, sendo um dos mais importantes, o tratamento do problema de busca e alinhamento de seqüências moleculares. Para isto, existem duas famílias principais de algoritmos, a família FAST e a família BLAST, sendo esta última a mais utilizada. O objetivo dos sistemas baseados nestes algoritmos é fornecer aos pesquisadores em biotecnologia ferramentas de comparação e busca entre diversas seqüências. Neste trabalho, apresentamos um modelo baseado nos algoritmos da família BLAST e são demonstrados, além do funcionamento destes, exemplos de alinhamentos. Descrevemos o funcionamento de algoritmos para o alinhamento de seqüências de DNA, baseados em programação dinâmica, incluindo um algoritmo de alinhamento múltiplo, conhecido como Estrela.

Palavras-chave: similaridade, programação dinâmica, comparação de seqüências, alinhamento múltiplo.

ABSTRACT

The database of bio sequences increased in size and number with the advent and growth of bio computer science, specially the sequence of several genome organisms, including the human genome. This brought the necessity of new techniques, methods of analysis and treatment of these informations, being the treatment of the problem of search and alignment of molecule sequences one of the most important. For this, there are two main families of algorithms: family FAST and family BLAST, being the last one the most used. The objective of the systems based on these algorithms is to supply the researchers in biotechnology tools of comparison and search between several sequences. In this work we present a model based on BLAST family algorithms and, besides their functioning, and examples of alignments. We describe the functioning of algorithms for the alignment of molecule sequences based on dynamic programming, including an algorithm of multiple alignment known as Star.

Keywords: similarity, dynamic programming, sequences comparison, multiple alignment.

1. INTRODUÇÃO

Hoje, sabemos que o DNA é o material genético de toda a vida na Terra. O DNA é formado por três componentes, sendo: um grupo açúcar (desoxirribose), um agrupamento fosfato e uma base nitrogenada, que pode ser de quatro tipos: Adenina (A), Guanina (G), Citosina (C) e Timina (T).

A cada dia, é anunciada na mídia uma nova descoberta, trazendo à tona esperanças em relação à cura de doenças, à melhoria do processo de justiça criminal e à prevenção de problemas ambientais, entre outros. Porém, para a concretização de muitas das expectativas recentemente criadas, ainda há muito estudo e investimentos pela frente, o que poderá consumir décadas de pesquisas.

No entanto, o ritmo das descobertas científicas vem aumentando, pois os cientistas estão usando as ferramentas da bioinformática na busca de soluções para os problemas em todos os ramos da biologia, desde biologia molecular até ecologia e evolução, por exemplo.

A abrangência da bioinformática/biologia computacional é muito extensa, de modo que o foco deste trabalho se restringirá à comparação de bioseqüências e suas aplicações.

Dentre as diversas pesquisas na área, podemos destacar também o desenvolvimento de técnicas e métodos de armazenamento em bases de dados de bioseqüências, em particular as seqüências de ácidos nucléicos e aminoácidos e suas respectivas anotações.

O alinhamento de bioseqüências tem por objetivo comparar uma seqüência a outra(s) obtendo trechos semelhantes entre elas. Isto pode ter várias aplicacões:

- Métodos para determinação de grau de parentesco, tais como testes de paternidade;
- Métodos para identificação de pessoa específica, por exemplo, no caso de identificação criminal;
- Métodos para classificação de espécies, por exemplo, com a descoberta de um novo organismo.

O objetivo principal deste trabalho é o estudo de algoritmos de alinhamento de bioseqüências, baseados em programação dinâmica. O problema de alinhamento pode ser formulado em termos da pesquisa de similaridade entre uma dada seqüência específica e outras depositadas na base de dados. Isto implica, dependendo do tamanho das entradas, em um razoável custo de computação [Meidanis, 1994]. Esta operação (cálculo de similaridade) tem como função encontrar trechos semelhantes de seqüências moleculares, o que, aparentemente, representa uma tarefa simples. Porém, devemos observar que há vários problemas distintos, com formalizações diversas, em sua maioria, exigindo algoritmos e estruturas de dados próprias para a execução eficiente.

Como o algoritmo básico de alinhamento possui complexidade quadrática [Meidanis, 1994], vários métodos alternativos tipo heurística foram desenvolvidos para obter menor tempo de execução, pois à medida que o volume de dados a ser analisado aumenta, o tempo de execução se torna crítico.

Existem vários algoritmos de alinhamento, que são divididos em famílias; por exemplo, os da família FAST e os da família BLAST. Essas famílias representam métodos largamente utilizados por pesquisadores da área [Meidanis, 1997].

Este trabalho está organizado da seguinte forma: na Seção 2, são apresentadas as definições sobre alinhamento de moléculas e seus principais problemas; na Seção 3, descrevemos alguns algoritmos envolvidos na comparação de biosseqüências em banco de dados, com suas respectivas complexidades; na Seção 4, são apresentadas informações relacionadas à implementação que desenvolvemos, as conclusões e expectativas de trabalhos futuros.

2. DEFINIÇÕES

Definimos como *seqüência* ou *cadeia* uma justaposição de caracteres de um dado alfabeto S. Assim, se $S = \{A,C,T,G\}$, então AAAC e AGC são exemplos de seqüências [Meidanis, 1994].

Dada uma seqüência k, uma *subseqüência* de k é uma seqüência que poderá ser obtida a partir de um fracionamento de k, por exemplo, ACG é uma subseqüência de ACGT e ACT não o é. Uma *subcadeia* é uma seqüência obtida a partir da segmentação de uma cadeia, por exemplo, ACG é uma subcadeia de ACGT e AGC não o é. Toda subseqüência é uma subcadeia, mas, o inverso não o é.

Dadas duas seqüências distintas, k e y, obtemos a concatenação de ambas justapondo os caracteres de k aos de y, por exemplo, dada uma regra na forma $w = k \times y$, dizemos que k é *sufixo* de w, y é *prefixo* de w e x é um *fator* de w [Meidanis, 1994].

Um *buraco* é um símbolo que não pertence ao alfabeto S, que neste trabalho será representado pelo caractere '-' [Meidanis, 1994].

2.1. Dados

Com o avanço das pesquisas e o número de bioseqüências cadastradas crescendo exponencialmente, tornou-se necessário a utilização de sistemas

gerenciadores de bancos de dados (SGBD), que são mais adequados para o gerenciamento de grandes volumes de dados [Doolitle, 1990]. A grande maioria das informações sobre bioseqüências estão armazenadas em bancos de dados relacionais ou sistemas orientados a objeto. Temos como exemplo o GenBank [Benson, 2000], que é um banco de dados público de bioseqüências, que contém as informações biológicas e bibliográficas e é produzido pelo *National Center for Biotecnology Information* [NCBI].

Em um banco de dados bioMolecular (BDBM), o volume de dados que será armazenado tornou-se um ponto critico, já que a massa de dados vem crescendo rapidamente, principalmente com o sequenciamento do genoma de vários organismos, inclusive do homem.

3. ALGORITMOS

Os algoritmos mais utilizados na atualidade são os da família BLAST (*Basic Local Alignment Search Tool*) [Meidanis, 1994], que estão baseados em programação dinâmica [Cormen, 1990]. Na implementação destes, existem alguns parâmetros que variam de acordo com o banco de dados que está sendo pesquisado (seqüência de proteínas ou de DNA).

O BLAST utiliza como entrada um banco de dados, que nada mais é do que um arquivo texto organizado em um formato chamado FASTA, contendo as seqüências e seus respectivos cabeçalhos; cada cabeçalho possui algumas informações da bioseqüência que o segue.

A seguir, a Figura 1 ilustra um trecho de um arquivo em formato FASTA:

```
>gi|2983882 K+ transport protein homolog
MVKKLNPSRTLLFSFSLLILVGALLLYLPISTTRPISFLDALFTATSAVTVTGLAVLDTYSDFTLFGKLV
ILFLIQVGGLGYMTLSTFFLVLLGRRIGLKERLILAESLEYPSMHGLIRFLKRVFSFVFITELTGAILLS
IYFSLKGVEDPVFNGIFHSVSAFNNAGFSTFKNGLLDFRGDLFVNLVISFLIILGGIGFFVVNDIYLWYT
KKVPRLSVHTKLVMITSVLLILLGTVGLIFTEFGNYKGLWQYDWYERILSSYFMSVSSRTAGFSTVDLID
MSESSQFLLMILMFIGASPGGTGGGIKTTTFFVVILIAVYSFVRGREQSVIFERSVPESTIKKALVILSLS
IFFINFVNLMLDKFENKDFLYTMFEVVSAFSTVGLSIGNPEGLSFCADFSPLGKIVIITMLVGRLGILG
FALALTGRSEVQRIKYPEARILV
```

Figura 1: Exemplo de arquivo FASTA.

Para melhor esclarecer o funcionamento dos algoritmos BLAST, vamos utilizar um exemplo: considere o problema de obter o alinhamento ótimo para o par de seqüências AAAC e AGC. Como pode haver mais de um alinhamento, o propósito básico desses algoritmos é determinar qual o alinhamento ótimo.

```
A A A C      A A A C -      A A A C
A G   C      A G   C        A G C -
```

Figura 2: Possibilidades de alinhamento de última coluna.

Como pode ser visto na Figura 2 existem três possibilidades para a última coluna: alinhar C com C, C com buraco ou buraco com C.

Após identificar essas possibilidades, podemos calcular o *score* para cada uma delas. Poderíamos executar o mesmo procedimento, recursivamente, para cada uma das possíveis subseqüências (de cada uma das seqüências originais) restantes. Este método apresenta o problema de gerar um número exponencial de chamadas recursivas, sendo muitas delas redundantes.

Não há necessidade de calcular mais de uma vez o *score* do alinhamento de duas colunas em duas subseqüências, desde que os resultados sejam guardados de maneira que possam ser consultados posteriormente (e rapidamente). Este é o princípio básico da programação dinâmica. Em geral é usada uma matriz para guardar os resultados parciais.

3.1. Matriz de pontuação

Como dissemos, usamos programação dinâmica para evitar cálculos redundantes; para isso, é necessária uma estrutura de dados que armazene os resultados parciais para consultas posteriores, no caso, uma *matriz de pontuação*.

Para ilustrar, vejamos a matriz da Figura 3: a segunda linha representa a primeira seqüência, *S1* (no exemplo é a AGC), a segunda coluna representa a segunda seqüência, *S2* (no exemplo é a AAAC). Um tamanho genérico *m* é atribuído à seqüência S1 por isso, será representada por *S1[1...m]*, da mesma forma, para a seqüência *S2* é atribuído um tamanho genérico *n* e é representada por *S2[1...n]*. Quando o prefixo de comprimento *i* da seqüência *S1* for considerado, será representado por *S1[1...i]*, da mesma forma em *S2* será representado por *S2[1...j]*. Os valores de *i* e *j* são exibidos na primeira linha e primeira coluna da matriz, sendo que $1 \leq i \leq n$ e $1 \leq j \leq n$.

Cada célula *(i, j)* da matriz representa a pontuação (*score*) do alinhamento de melhor pontuação entre dois prefixos de *S1[1...i]* e *S2[1...j]*; veja na Figura 3 as subseqüências consideradas para cada célula *(i, j)* da matriz de pontuação correspondente ao exemplo da Figura 2.

	i	0	1	2	3
j	S1[1...m] S2[1...n]	—	A	G	C
0	—	- | - S1[0] S2[0]	A | - S1[1] S2[0]	AG | - S1[1...2] S2[0]	AGC | - S1[1...3] S2[0]
1	A	- | A S1[0] S2[1]	A | A S1[1] S2[1]	AG | A S1[1...2] S2[1]	AGC | A S1[1...3] S2[1]
2	A	- | AA S1[0] S2[1...2]	A | AA S1[1] S2[1...2]	AG | AA S1[1...2] S2[1...2]	AGC | AA S1[1...3] S2[1...2]
3	A	- | AAA S1[0] S2[1...3]	A | AAA S1[1] S2[1...3]	AG | AAA S1[1...2] S2[1...3]	AGC | AAA S1[1...3] S2[1...3]
4	C	- | AAAC S1[0] S2[1...4]	A | AAAC S1[1] S2[1...4]	AG | AAAC S1[1...2] S2[1...4]	AGC | AAAC S1[1...3] S2[1...4]

Figura 3: Matriz indicando em cada célula as subseqüências que estão sendo comparadas.

O *score* de A | AAA é calculado da seguinte forma na matriz:

- Soma-se o *score* (-1) de A | AA a (-2). Isto porque o alinhamento da última coluna que foi feito em A | AAA para se obter A | AA foi A | -, cujo *score* é -2.
- Soma-se o *score* (-6) de - | AAA a (-2). Isto porque o alinhamento da última coluna que foi feito em A | AAA para se obter - | AAA foi - | A, cujo *score* é -2.
- Soma-se o *score* (-4) de - | AA a (+1). Isto porque o alinhamento da última coluna que foi feito em A | AAA para se obter - | AA foi A | A, cujo *score* é +1.

A maior das somas acima será o *score* de A | AAA. No caso, teríamos os valores -3, -8 e -3, sendo assim o *score* de A | AAA é -3.

O pseudo código da Figura 4 descreve com maior precisão o funcionamento deste algoritmo no preenchimento de cada célula. Os cálculos devem ser efetuados, de modo que os valores de células anteriores, necessários para determinar o valor de uma célula, estejam disponíveis no momento certo. Este algoritmo calcula a pontuação entre *S1[1...m]* e *S2[1...n]*, dependendo de um parâmetro *g* que indica o peso de um buraco (no caso o valor -2) e de uma função *p* para pares de caracteres (no caso foi usado $p(x,y) = 1$ se $x = y$ e $p(x,y) = -1$ se $x \neq y$).

```
Sim(S1,S2) /* |S1| = m, |S2| = n */
Para i ← 0 até m faça
  a[i,0] ← i * g
Fim para
Para j ← 0 até n faça
  a[0,j] ← j * g
Fim para
Para i ← 1 até m faça
Para j ← 1 até n faça
  a[i, j] ← Max{a[i-1,j] + g, a[i-1,j-1] + p(i, j) , a[i,j-1] + g}
Fim para
Fim para
```

Figura 4: Algoritmo básico de programação dinâmica para comparação de seqüências.

3.2 Algoritmo *traceback*

Quando as pontuações de todas as células forem calculadas, é então possível obter o alinhamento ótimo; para isso, é necessário percorrer as células da matriz. Neste percurso, cada movimento de uma célula para outra corresponde a uma coluna do alinhamento. Genericamente, estando na célula *a[i, j]*, é possível ir para as células *a[i-1,j]*, *a[i,j-1]* ou *a[i-1,j-1]*. Se o passo for de *a[i, j]* para *a[i,j-1]*, significa que está sendo considerado o alinhamento de um buraco em *S2[1...j]* e um caractere em *S1[1...i]*. Se o passo for de *a[i, j]* para *a[i-1,j]*, significa que está sendo considerado o alinhamento de um buraco em *S1[1...i]* e um caractere em *S2[1...j]*. Se o passo for de *a[i, j]* para

a[i-1,j-1], significa que está sendo considerado o alinhamento de um caractere em *S2[1...j]* e um caractere em *S1[1...i]*.

Seguindo este raciocínio, é construído o alinhamento ótimo percorrendo a matriz, calculada anteriormente, a partir da célula da posição *[m, n]*.

Nesta etapa, estando em uma célula *(i,j)*, precisamos descobrir como foi obtido o score da célula *(i,j)*, segundo as regras descritas. Para isto, basta fazer as somas dos *scores* das células vizinhas (*a[i-1,j]*, *a[i,j-1]* ou *a[i-1,j-1]*) com *p(i, j)* ou *g* (dependendo do caso) e encontrar qual das somas é igual ao *score* procurado. Descobrindo a célula vizinha responsável pelo *score* da célula que está sendo tratada - *(i,j)*, descobre-se também o alinhamento de uma determinada coluna, o próximo passo é fazer a mesma análise para esta célula vizinha escolhida, desta forma a matriz vai sendo percorrida.

É possível que mais de uma célula vizinha possa ser a responsável pelo *score* da célula que está sendo tratada, neste caso, qualquer uma delas é válida, fica a critério do programador determinar o caminho a seguir. Para ilustrar, apliquemos esta idéia ao exemplo da Figura 2, para verificarmos o alinhamento ótimo de AGC e AAAC.

S1	-	A	G	C
S2	A	A	A	C

Figura 5: Vetor com resultado do alinhamento.

Siga as setas na matriz apresentada na Figura 6 e os passos a seguir:

- A análise se inicia na última célula da matriz, que corresponde às seqüências AGC e AAAC. Testa-se como o *score* desta célula é -1. A soma da vizinha acima seria (-1-2) = -3, da diagonal esquerda seria (-2+1) = -1, e da esquerda seria (-4-2) = -6. Então o *score* é igual à soma da vizinha da diagonal esquerda, que corresponde ao alinhamento da última coluna de AGC | AAAC, que é C | C (lembrando que os outros casos tratados eram C | - e - | C), passando a analisar a vizinha da diagonal esquerda;
- A análise continua na célula das seqüências AG | AAA, vendo que o *score* desta célula é -2. A soma da vizinha acima seria (0-2) = -2, da diagonal esquerda seria (-1-1) = -2, e da esquerda seria (-3-2) = -5. Então o *score* é igual à soma da vizinha da diagonal esquerda (ou da vizinha acima, mas será escolhido aqui a diagonal esquerda), que corresponde ao alinhamento da última coluna de AG | AAA, que é G | A (lembrando que os outros casos tratados eram G | - e - | A), passando a analisar a vizinha da diagonal esquerda;
- A análise continua na célula das seqüências A | AA, testa-se que o *score* desta célula é -1. A soma da vizinha acima seria (+1-2) = -1, da diagonal esquerda seria (-2+1) = -1, e da esquerda seria

	i	0	1	2	3
j	S1[1...m] S2[1...n]	—	A	G	C
0	—	0 - | -	-2 A | -	-4 AG | -	-6 AGC | -
1	A	-2 - | A	+1 +1 A | A	-1 -1 AG | A	-1 -3 AGC | A
2	A	-4 - | AA	+1 -1 A | AA	-1 0 AG | AA	-1 -2 AGC | AA
3	A	-6 - | AAA	+1 -3 A | AAA	-1 -2 AG | AAA	-1 -1 AGC | AAA
4	C	-8 - | AAAC	-1 -5 A | AAAC	-1 -4 AG | AAAC	+1 -1 AGC | AAAC

Figura 6: Cálculo do alinhamento ótimo pela matriz A.

(-4-2) = -6. Então o *score* é igual à soma da vizinha da diagonal esquerda (ou da vizinha acima, mas será escolhido aqui a diagonal esquerda), que corresponde ao alinhamento da última coluna de A | AA, que é A | A. Mas é igual também à soma da vizinha acima, que corresponde ao alinhamento de A | -. Neste momento qualquer um dos casos poderia ser seguido. Suponha que o algoritmo implementado dê prioridade para a análise da vizinha da diagonal esquerda;

- A análise continua na célula das seqüências - | A, esta é uma célula que corresponde à base da recursão, verificando que o *score* desta célula é -2, neste caso só existe a célula vizinha acima, e a soma dela deve ser igual à -2. Como esperado, a matriz mostra que a soma é (0-2) = -2, portanto, retorna-se a base da recursão, que é o alinhamento - | A;
- Finalizando, construímos as tabelas *S1* e *S2* esquematizadas na Figura 3.

O algoritmo do alinhamento está representado na Figura 7.

```
alin(i, j, t)
  Se i = 0 e j = 0 então
     t ← 0
  Se i > 0 e a[i, j] = a[i-1,j] + g então
     alin(i-1,j,t)
     t ← t + 1
     S1[t] ← S1[i]
     S2[t] ← S2[j]
  Se i > 0 and j > 0 e a[i, j] = a[i-1,j-1] + p(i, j) então
     alin(i-1,j-1,t)
     t ← t + 1
     S1[t] ← S1[i]
     S2[t] ← S2[j]
  Se j > 0 e a[i, j] = a[i,j-1] + g então
     alin(i,j-1,t)
     t ← t + 1
     S1[t] ← '-'
     S2[t] ← S2[j]
```

Figura 7: Algoritmo de alinhamento.

3.3. Complexidade do algoritmo

O algoritmo que preenche a matriz com os valores das pontuações possui quatro laços de repetição; os dois primeiros consomem tempo *O(m)* e *O(n)*, respectivamente. Os dois últimos laços de repetição são encaixados e consomem *O(mn)* unidades de tempo, ou seja, esta é a parte do algoritmo que domina assintoticamente. O espaço utilizado também é proporcional ao tamanho da matriz, logo, o algoritmo possui complexidade de espaço *O(mn)*. O algoritmo que calcula o alinhamento a partir da matriz consome tempo *O(t)*, onde *t* é o tamanho do alinhamento retornado. Neste algoritmo há essencialmente uma chamada recursiva por coluna de alinhamento e cada chamada consome tempo *O(1)*.

3.4. Alinhamento Múltiplo

Os algoritmos descritos tratam do alinhamento entre duas seqüências. Em muitos casos, é de interesse promover o alinhamento de *k* seqüências *S1,S2,...Sk* conforme ilustrado na Figura 8.

S1 =	A	A	G	...	T
S2 =	A	A	A	...	C
...	...	...	...	...	...
Sk =	G	T	C	...	A

Figura 8: Alinhamento múltiplo.

Algoritmos exatos para alinhamento múltiplo têm complexidade exponencial. Uma alternativa que é freqüentemente usada é o desenvolvimento de heurísticas, que apesar de não garantirem a optimalidade do alinhamento resultante, podem fornecer respostas rápidas e razoavelmente boas.

Uma heurística bastante conhecida e utilizada é o método de alinhamento estrela [Meidanis, 1997], que está baseada na fixação de uma seqüência, denominada *centro da estrela*, cuja determinação está descrita nos pseudo códigos das Figuras 10,11 e 12. Para estabelecer qual das seqüências contidas no banco será fixada como centro, são calculadas as pontuações entre os pares de seqüências (*Si,Sj*), $1 \leq i, j \leq k$, $i \neq j$. Estas pontuações são armazenadas em uma matriz. Depois, as colunas desta matriz são analisadas de modo a obter a coluna de maior somatório, que terá o seu índice utilizado para fixar o centro da estrela, e as demais seqüências passarão a ser as pontas. O algoritmo que calcula o centro da estrela (centro) é definido em três passos:

Passo 1: Calcula a pontuação do alinhamento ótimo de cada par de seqüência (*Si,Sj*), $1 \leq i, j \leq k$, $i \neq j$, onde *a[n,m]* representa a pontuação do alinhamento ótimo das subseqüências *Si[n]* e *Sj[m]*. O pseudo código da Figura 9 descreve melhor esse passo.

```
j ← 0
Enquanto i < n faça
   j ← i + 1
Enquanto j < n faça
   Sim (Si[1...n],Sj[1...m])
   Matriz[i,j] ← a[n,m]
   Matriz[j,i] ← a[n,m]
   j ← j + 1
Fim-enquanto
   i ← i + 1
Fim-enquanto
```

Figura 9: Algoritmo para cálculo de pontuações entre cada par de seqüências.

Passo 2: Calcula, para cada seqüência *Si*, *i* 1 $\leq i \leq k$, a soma das pontuações de cada seqüência (representadas pelas colunas de *Matriz*), obtidas no passo anterior. A Figura 10 descreve melhor esse passo.

```
j ← 0
Enquanto j < n faça
  soma [j] = 0
  i ← 0
  Enquanto i < n faça
    Se (i != j)
      soma [j] ← soma [j] + Matriz [i,j]
    Fim se
    i ← i + 1
  Fim-enquanto
    j ← j + 1
Fim-enquanto
```

Figura 10: Algoritmo para cálculo das somas das pontuações das colunas de *Matriz*.

Passo 3: Seleciona o centro da estrela. A Figura 11 descreve melhor esse passo.

```
max ← soma [0]
c ← 0
i ← 1
Enquanto i < n faça
  Se (max < soma [i])
    max ← soma [i]
    c ← i
  Fim se
  i = i + 1
Fim-enquanto
```

Figura 11: Algoritmo para selecionar o centro.

A seguir, é observado como estes algoritmos funcionam sobre um conjunto seqüências hipotéticas. Sejam as seqüências abaixo:

S1 = A T T G C C A T T
S2 = A T G G C C A T T
S3 = A T C C A A T T T T
S4 = A T C T T C T T
S5 = A C T G A C C

Na Figura 12, é mostrada a Matriz com as pontuações obtidas nas comparações entre estas seqüências.

	S1	S2	S3	S4	S5
S1		7	-2	0	-3
S2	7		-2	0	-4
S3	-2	-2		0	-7
S4	0	0	0		-3
S5	-3	-4	-7	-3	

Figura 12: Matriz de pontuação.

A coluna *S1* obteve o maior somatório, desta forma, passará a ser o centro da estrela, enquanto *S2*, *S3*, *S4* e *S5* passarão a ser as pontas.

Ao selecionarmos o centro, estaremos escolhendo um índice *c*, *1* $\leq c \leq k$,. Para cada índice *i* diferente de *c*, podemos obter um alinhamento ótimo - utilizando os algoritmos de programação dinâmica padrão discutidos anteriormente - entre as seqüências *Si* e *Sc*. Por exemplo, para o conjunto de seqüências citado, são obtidos os seguintes alinhamentos para cada par de seqüências:

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

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

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

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

Devido ao tamanho de S3, para demonstrar adequadamente o resultado deste alinhamento, é necessário incluir buracos no final de cada seqüência resultante, assim, não será alterado o resultado do alinhamento; o algoritmo da Figura 13 descreve esse passo:

```
T ← 0
Enquanto não for fim do arquivo Faça
  Se T < Tamanho registro
    T ← Tamanho do registro
  Fim Se
Fim-enquanto

Enquanto não for fim do arquivo Faça
  K ← Tamanho do registro
  Se K < T
    Para i ← K + 1 até T Faça
      S[i] ← "-"
    Fim Para
  Fim Se
Fim-enquanto
```

Figura 13: Algoritmo de ajuste das seqüências alinhadas.

O resultado da execução do algoritmo pode ser observado abaixo:

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

4. IMPLEMENTAÇÃO E CONCLUSÕES

Na implementação dos algoritmos de alinhamento, foi criado um aplicativo com interface visual desenvolvido em Visual Basic Professional versão 6.0, onde o usuário pode escolher entre três tipos de alinhamento, conforme descrição abaixo:

- Alinhamento 1 X 1 (um para um), que compara duas seqüências individualmente, demonstrando o alinhamento e a pontuação entre as duas seqüências;
- Alinhamento 1 X N (um para N), que compara uma seqüência selecionada com N seqüências contidas em um arquivo, demonstrando os alinhamentos e as pontuações obtidas, um de cada vez, conforme parâmetros fornecidos pelo usuário;
- Alinhamento N X N (N para N), opção que realiza um alinhamento múltiplo de N seqüências, segundo o método de alinhamento estrela. Lê as seqüências de um arquivo (do tipo FASTA) e demonstra em tela o alinhamento obtido.

Durante as simulações foi observado que para um micro com 64 MB de memória principal, pode-se alinhar seqüências de até três mil caracteres. Para micros com 128 MB de memória, alinhamentos de até cinco mil caracteres e, para micros com 256 MB de memória alinhamentos de até dez mil caracteres. Para seqüências acima de três mil caracteres em micros com 64 MB e acima de cinco mil caracteres em micros com 128 MB e acima de dez mil caracteres em micros com 256 MB não obtivemos sucesso, pois ocorreu *stack overflow*. Não efetuamos medidas precisas de desempenho de tempo, porém, para as entradas acima executadas com sucesso, nas três máquinas consideradas, observamos tempos entre 1 e 5 minutos.

O propósito geral deste trabalho foi estudar, descrever e implementar um dos algoritmos básicos de alinhamento de seqüências moleculares. Para atingir este objetivo estudamos uma técnica geral de desenvolvimento de algoritmos (programação dinâmica) e métodos de alinhamento baseados nesta técnica, além de implementar um algoritmo de alinhamento global entre duas seqüências e o método de alinhamento múltiplo estrela.

Também foi necessário fazer um estudo relativo a conceitos básicos de biologia molecular a fim de melhor compreender o contexto de aplicações destes métodos.

REFERÊNCIAS BIBLIOGRÁFICAS

BENSON, D.; KARSCH-MIZRACHI, I.; LIPMAN, D. J.; OSTELL, J.; RAPP, B. A.; WHEELER, D.L. **Genbank**. Nucleic Acids Research, 2000.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to Algorithms**. MIT Press, 1990.

DOOLITLE, R. F. **Molecular Evolution: Computer Analisys of Protein and Nucleic Acid Sequences.** Methods in Enzimology. Academic Press, 1990.

NCBI – The National Center for Biotecnology Information. http://www.ncbi.nlm.nih.gov.

NHGRI – The National Human Genome Research Institute. http://www.nhgri.nih.gov.

MEIDANIS, J.; SETÚBAL, J. C. **Uma Introdução à Biologia Computaciona**. Escola de Computação. Recife, 1994.

MEIDANIS, J.; SETÚBAL, J. C. **Introdution in to Computational Molecular Biology**. PWS Publishing Company, 1997.