Técnicas para Comparação e Visualização de Similaridades entre Seqüências Genéticas

Felipe Fernandes Albrecht (FURB/DSC)
albrecht@inf.furb.br

Resumo. Este artigo apresentará técnicas para comparação e visualização de similaridades de seqüências genéticas. Será exibido as características das principais técnicas, analisado seus funcionamentos, comparado o custo de execução e de espaço e a sua área de aplicação. Por fim, será apresentado um *software* que implementa as técnicas de programação dinâmica para visualização e análise de similaridades entre seqüências genéticas. Para tornar possível uma melhor visualização das similaridades foi desenvolvido um novo algoritmo para matriz de pontos com filtro.

Palavras-chave: Bioinformática, Seqüências Genéticas, Programação dinâmica, Alinhamento de seqüências genéticas, Matriz de pontos, Comparação de seqüências genéticas.

1 Introdução

A comparação e análise de seqüências genéticas é parte de um campo científico chamado bioinformática. Este campo científico é baseado numa interação entre a estatística, biologia molecular e a ciência da computação. Por causa de grandes projetos de seqüênciamento genômicos, por exemplo, o Projeto do Genoma Humano, existe um crescimento exponencial na quantidade de dados disponíveis sobre seqüências genéticas e proteínas. Os métodos tradicionais de laboratório para o estudo da estrutura e função destas moléculas não são capazes de acompanhar a taxa de crescimento de novas informações. Como conseqüência, biólogos moleculares passaram a utilizar métodos estatísticos e computacionais capazes de analisar estas grandes quantidade de dados de forma mais automatizada.

Segundo Mount (2004), alinhamento de seqüências é útil para a descoberta de informações funcionais, estruturais e evolucionárias nas seqüências biológicas. Partindo da premissa de que duas seqüências similares possuem estruturas e comportamentos similares, pode-se inferir informações sobre uma nova seqüência comparando-a a outras já previamente conhecidas.

Existem diversas formas de comparações e alinhamento de seqüências genéticas. Neste artigo, pretende-se dar uma introdução sobre os principais e mais utilizados métodos de comparação de seqüências genéticas. Iniciando do mais básico, o algoritmo de força bruta, utilização de matriz de pontos, programação dinâmica, heurísticas e modelos probabilísticos. Por fim é descrito na seção 3 um software para visualização de similaridades e alinhamentos entre seqüências e um novo algoritmo na seção 3.1 para a realização de tal visualização.

2 Algoritmos e Métodos para Comparação de Seqüências

Esta seção apresentará os principais métodos e algoritmos para comparação de seqüências genéticas.

2.1 Força Bruta

A forca bruta é provavelmente o método mais simples para a comparação de seqüências. Ele é apenas utilizado para determinar se duas seqüências são idênticas ou não, sem reportar qualquer grau de similaridade. O algoritmo é bastante simples: primeiro compara-se os primeiros elementos

```
BRUTE-FORCE-COMPARE(M, N)
if length[M] ≠ length[N]
   then return DIFFERENT

for i ← 0 to length[M]
do if M_i ≠ N_i
       then return DIFFERENT
return EQUAL
```

Quadro 1: Algoritmo força bruta para comparação de seqüências

de ambas seqüenciais, depois os segundos, os terceiros e assim sucessivamente, até encontrar um erro e retornar que as seqüências não são idênticas ou chegar no fim das seqüências e retornar que ambas são idênticas. O algoritmo está especificado no quadro 1.

Um aspecto importante a ser analisado é a quantidade de operações máximas $(O(n))$ que são necessárias para obtenção do resultado neste algoritmo. Se a seqüência é uma lista de elementos aleatórios de um alfabeto de K letras, a probabilidade de acerto, isto é, a probabilidade dos dois elementos das listas que estão sendo comparados serem iguais, em qualquer posição é $1/K$. Em seqüências com alfabetos grandes e com conteúdos aleatórios, a probabilidade de acerto será muito baixa, e normalmente serão necessárias poucas comparações para determinar que as duas seqüências são diferentes. Porem, se as seqüências não são equiprobatoriamente distribuída e o alfabeto possuir um número reduzido de elementos, como no caso do DNA, onde seu alfabeto é constituído de somente quatro letras e os padrões de repetições são comuns, o número de comparações irá aumentar significativamente, desta forma, aumentando o tempo de execução do algoritmo. No pior caso, quando as seqüências são realmente idênticas, o algoritmo de força bruta irá executar N comparações de elementos (N sendo o comprimento das seqüências), ou seja a complexidade do algoritmo é $O(n)$.

O tempo de execução deste algoritmo varia consideravelmente de acordo com a natureza das seqüências a serem analisadas e os resultados são muito simples, apenas informando se as seqüências são idênticas ou não. Sendo que na comparação de seqüências genéticas, o mais importante é o grau de homogeneidade, ou seja, o quão similares são as seqüências. Mesmo desta forma, este algoritmo não deve ser descartado, pois ele é utilizado dentro de outros algoritmos mais eficientes, principalmente nos baseados em heurísticas. Variações deste algoritmo e outros métodos de comparação exata de seqüências podem ser obtidos em (C. Charras; T. Lecroq, 1996).

2.2 Matriz de pontos

Uma matriz de pontos é primariamente um método para comparação de duas seqüenciais pela observação de possíveis alinhamentos de caracteres entra as seqüências a serem analisadas. Este método também é utilizado para encontrar repetições em seqüências genéticas e para predizer regiões no RNA que são auto complementares e possuem potencial para formar uma estrutura secundária.

Numa matriz de pontos, utiliza-se uma seqüências (M) como eixo horizontal e outra seqüências (N) como eixo vertical. Cada célula da matriz de pontos, $D(i, j)$ é resultado da comparação dos resíduos i da seqüência M com o elemento j da seqüência N. Para todo par de resíduos M_i e

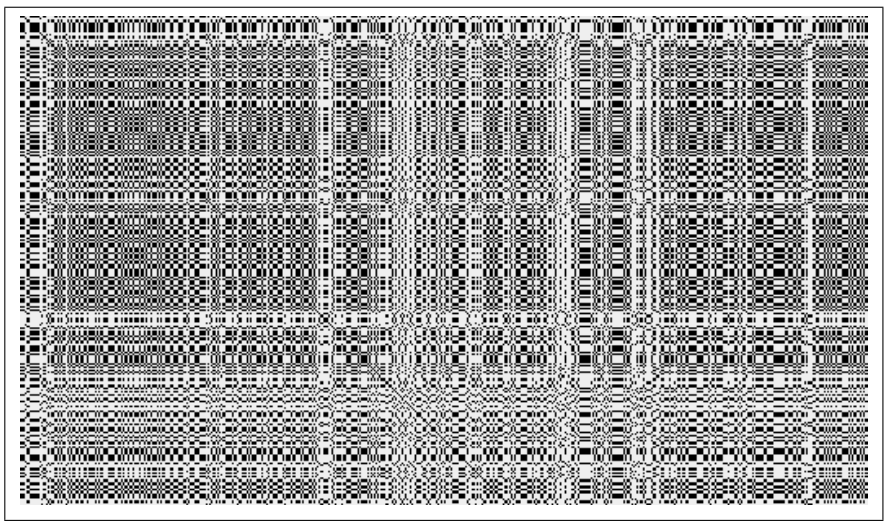

Figura 1: Comparação de duas seqüências sem utilização de filtro

```
SIMPLE-DOT-MATRIX(M, N)
for i ← 0 to length[M]
do for j ← 0 to length[N]
    do if M_i = N_i
            then matrix[i, j] ← 1
            else  matrix[i, j] ← 0
return matrix
```

Quadro 2: Algoritmo simples para matriz de pontos

N_i idênticos, deve-se pintar a célula $D(i, j)$ correspondente de preto. Os alinhamentos locais significantes aparecem na forma de linhas diagonais bem definidas, contra um fundo de pontos sem relação. Um exemplo da sua execução pode ser observado na figura 1 e o algoritmo é especificado no quadro 2.

O tempo de execução deste algoritmo é a multiplicação do comprimento da seqüência M pela comprimento da seqüência N, resultando como tempo de execução $O(mn)$ e o espaço necessário para armazenar tal matriz de pontos é dado pela mesma multiplicação.

Segundo Mount (2004), uma das maiores vantagens do método para encontrar alinhamentos matriz de pontos, é que todos os possíveis encontros de resíduos entre as duas seqüências são exibidos, deixando ao pesquisador a escolha de identificar os mais significantes. O pesquisador irá observar as regiões melhor alinhadas ou que contenham mais repetições e poderá seleciona-las para fazer uma análise mais profunda, utilizando outros métodos de alinhamento de seqüências, como a programação dinâmica, explicada na seção 2.3.

A visualização de regiões com alinhamentos pode ser melhorada pela remoção dos encontros

```
SLIDING-WINDOW-DOT-MATRIX(M, N, WINSIZE, MIN)
for i ← 0 to length[M]
do for j ← 0 to length[N]
    do sum ← 0
        for w ← 0 to WINSIZE
        do if M_{i+w} = N_{j+w}
                then sum ← sum + 1
        if sum ≥ MIN
          then matrix[i][j] ← 1
          else  matrix[i][j] ← 0
return matrix
```

Quadro 3: Algoritmo para matriz de pontos utilizando janelas deslizantes

aleatórios da matriz de pontos. Esta filtragem pode ser efetuada com o uso de janelas deslizantes *(sliding window)* na comparação das duas seqüências. Ao invés de comparar uma simples posição na seqüência, toda a janela de posições adjacentes ao ponto é comparada ao mesmo tempo e o ponto é exibido apenas se um certo número de encontros exatos ocorrerem. O algoritmo é especificado no quadro 3.

Segundo Mount (2004), uma boa medida para a janela é 15 pares de bases e ser necessário 10 encontros para a exibição do ponto. De qualquer forma estes valores são arbitrários e muitas combinações podem ser utilizadas em busca de melhores visualizações.

Como pode-se observar no quadro 3, este algoritmo tem um custo de execução $O(n^3)$, tornando sua aplicação impraticável para comparações de seqüências longas. Por exemplo, considerando duas seqüências genéticas, cada qual com um comprimento de 1000 pares de bases e utilizando uma janela de 15 pares, serão efetuadas ao todo: $1000 * 1000 * 15$, resultando 150000000 comparações para a criação de uma matriz de pontos utilizando janelas deslizantes. Um algoritmo com menor custo de execução é proposto na seção 3.1.

2.3 Programação dinâmica

Quando um novo gene é descoberto, biólogos normalmente não tem idéia sobre a sua função. Uma técnica é buscar similaridades com genes com funções conhecidas e inferir a função do novo gene com base nas similaridades encontradas.

Segundo Neil C. Jones e Pavel A. Pevzner (2004), Programação Dinâmica provê um *framework* para compreender algoritmos de comparação de seqüências de DNA. A Programação Dinâmica resolve os problemas utilizando soluções previamente computadas, ou seja, dividindo o problema e reaproveitado as soluções previamente calculadas. Ela é utilizada quando as soluções dos problemas não são independentes, isto é, quando sub-problemas compartilham sub-problemas. Segundo Thomas H. Cormen, Charles E. Leiserson e Ronald L. Rivest (1999), um algoritmo de programação dinâmica soluciona todo sub-problema apenas uma vez e então salva sua resposta numa tabela, através disto evita o trabalho da recomputação da resposta toda vez que o sub-problema é encontrado.

Sendo a programação Dinâmica utilizada normalmente para otimização de problemas, no caso de alinhamentos de seqüências, a sua otimização é o alinhamento ótimo entre duas seqüências,

conforme forem especificados os parâmetros para tal. Mount (2004) complementa, o método é muito importante para analise de seqüências porque ele provê o melhor ou ótimo alinhamento entre duas seqüências.

Os algoritmos de programação dinâmica atribuem valores as mutações, deleções e substituições nas seqüências genéticas e então computa um alinhamento entre as duas seqüências que corresponde ao custo do conjunto de tais mutações. Tal alinhamento pode ser tratado como uma minimização da distância evolucionária ou a maximização da similaridades entre as duas seqüências a serem comparadas. Em ambos os casos, o custo deste alinhamento é uma medida de similaridades.

O alinhamento de duas seqüências genéticas pode ter várias soluções ótimas. O valor ótimo depende dos valores de pontuação para a casamento entre dois elementos, para o não casamento e para o *gap* (lacuna). Para o alinhamento de seqüências, há dois algoritmos principais que utilizam programação dinâmica. Para alinhamento global, onde deseja-se alinhas todos os elementos da seqüências M com todos da seqüência N, utiliza-se o algoritmo de Needleman-Wunsch e para alinhamento local, onde dá-se uma ênfase em alinhamentos em uma região, utiliza-se o algoritmo de Smith-Waterman.

2.3.1 Alinhamento global (algoritmo de Needleman-Wunsch)

O alinhamento global de duas seqüências é obtido através de construção de uma matriz de valores, onde cada célula representa um encontro entre uma base ou um gap da seqüência M e uma base ou um *gap* da seqüência N. As pontuações nas células podem ser negativas, positivas ou zero. Os valores nas última célula das linhas e colunas da matriz representam as pontuação de possíveis alinhamento. A recorrências matemática é visto na equação 1 e um possível algoritmo é apresentado no quadro 4. A função δ na equação 1 representa o valor do encontro entre os dois símbolos, retornando o valor de um encontro ou desencontro, conforme o ocorrido.

$$S_{i,j} = max \begin{cases} S_{i-1,j} - GAP, \\ S_{i,j-1} - GAP, \\ S_{i,j} + \delta(m_i, n_j). \end{cases} \tag{1}$$

2.3.2 Alinhamento local (algoritmo de Smith-Waterman)

O alinhamento local é muito similar ao alinhamento global, a diferença é que ele valoriza alinhamentos internos entre as duas seqüências e não o alinhamento completo entre elas. Esta diferença pode ser vista na equação 2, onde percebe-se a inclusão de uma nova opção de valor para as células, o zero. Então nas células da matriz de alinhamento local, os valores variarão de 0 a ∞.

$$S_{i,j} = max \begin{cases} S_{i-1,j} - GAP, \\ S_{i,j-1} - GAP, \\ S_{i,j} + \delta(m_i, n_j), \\ 0. \end{cases} \tag{2}$$

```
GLOBAL-ALIGNMENT(M, N)
for i ← 0 to length[M]
do matrix[i, 0] ← i * GAP_VALUE
for j ← 0 to length[N]
do matrix[0, j] ← j * GAP_VALUE
for i ← 1 to length[M]
do for i ← 1 to length[N]
    do if m_i = n_j
          then matrix[i][j] ← matrix[i − 1][j − 1] + MATCH_VALUE
          else  matrix[i][j] ← matrix[i − 1][j − 1] + DISMATH_VALUE
       if matrix[i][j] < matrix[i, j − 1] + GAP_VALUE
          then matrix[i][j] ← matrix[i][j − 1] + GAP_VALUE
       if matrix[i][j] < matrix[i − 1, j] + GAP_VALUE
          then matrix[i][j] ← matrix[i − 1][j] + GAP_VALUE
return matrix
```

Quadro 4: Algoritmo de alinhamento global utilizando programação dinâmica

2.3.3 Traceback

Para visualizar os alinhamentos, tanto globais como locais, a principal técnica utilizada é o *Traceback*. Onde cada célula possui um ponteiro para a célula que foi utilizado para calcular o seu valor.

Após todo o cálculo do alinhamento, deve-se percorrer este grafo dirigido acíclico e em cada passagem por uma célula, emitir os símbolos que correspondem a ela. O custo de execução deste algoritmo é $O(n + m)$.

Outra possível técnica, é ir na célula de valor ótimo do alinhamento e computar quais foram as células utilizadas para calcular os valores do alinhamento. Estando na célula $V(i, j)$, percorre-se as células vizinhas, $V(i-1, j-1)$, $V(i-1, j)$ e $V(i, j-1)$ e computa-se qual foi utilizada para chegar ao seu valor e então emitir o seu símbolo correspondente e executar a mesma operação nela. Desta forma, deve-se iniciar na ultima célula do alinhamento e percorrer a matriz de valores até chegar na primeira linha ou coluna. A vantagem desta técnica é que os dados requeridos para tal alinhamento já foram computados previamente e não há mais gasto com espaço.

Os algoritmos que utilizam programação dinâmica são muito exatos e retornam os melhores alinhamentos dependendo apenas dos parâmetros. A grande dificuldade deles é o espaço e tempo requerido para a sua computação, $O(mn)$. Por exemplo, para comparação de duas seqüências com dois mil pares de bases e utilizando 1 *byte* para cada par de base, serão necessários aproximadamente 30 *megabytes* para a construção da matriz. Técnicas como Dividir e Conquistar (Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest, 1999; Neil C. Jones; Pavel A. Pevzner, 2004) diminuem o espaço requeridos para $O(min(m, n))$. Para a implementação, utilizar técnicas de compactação, por exemplo, armazenar quatro nucleotídeos em um *byte*, desta forma reduz o espaço requerido em até 16 vezes.

2.4 Heurísticas

Conhecendo um problema, pode-se utilizar algumas aproximações que facilitarão a resolução deste problema. Utiliza-se informação e intuições a respeito da instância do problema e da sua estrutura para resolvê-lo de melhor forma. Sendo uma aproximação, nem toda heurística tem uma razão de qualidade comprovada matematicamente ou uma prova formal de convergência.

Para comparação de seqüências as duas principais famílias de algoritmos que utilizam heurística são FAST(Lipman, D.J.; Pearson, W.R, 1985), e BLAST (Basic Local Alignment Search Tool)(Altschul, S. F. et al., 1990).

2.5 Algoritmos probabilísticos

O principal algoritmo probabilístico para o alinhamento e busca de semelhanças entre seqüências é o Modelos Ocultos de Markov (Hidden Markov Models, HMM). Koski (2001) diz que os Modelos Ocultos de Markov foram primeiramente aplicados no reconhecimento de fala e atualmente são amplamente usados na biologia computacional e na bioinformática. Koski complementa que uma importante característica da modelagem dos modelos ocultos de Markov é a flexibilidade e adaptabilidade.

Os Modelos Ocultos de Markov tendem a funcionar num tempo de execução e de espaço menor que a programação dinâmica, porem, a qualidade dos resultados depende fortemente dos dados que foram utilizados para treinar os modelos. Deste modo ganha-se em custo do tempo de execução e espaço, mas a certeza da qualidade dos resultado é inferior.

3 Software para visualização e comparação de seqüências

Esta seção apresenta um *software* que tem o objetivo de dar ao usuário a opção de visualizar similaridades entre seqüências genéticas através de uma matriz de pontos e poder selecionar as subseqüências que o convém para realizar alinhamentos entre elas.

Um dos principais requisitos não funcionais, é a que ele seja simples de operar, isto é, sem menus e muitas opções de configurações. O usuário deve apenas abrir os arquivos com as seqüências genéticas e selecionar a seqüência que lhe convém. Outro requisito, é a necessidade de poder comparar seqüências com até dois mil pares de bases cada uma num tempo hábil.

Para o desenvolvimento foi o utilizado a Linguagem Java Versão 1.5, junto com a biblioteca BioJava(BIOJAVA, 2005). Esta biblioteca foi utilizada para facilitar a leitura dos dados no formato FASTA, comparação de nucleotídeos e exibição de dados.

3.1 Algoritmo para matriz de pontos

Uma dificuldade encontrada na exibição das matrizes de pontos com filtros descrita na seção 2.2, foi o seu alto custo computacional. Conforme visto na seção 2.2, o algoritmo de Matriz de Pontos com Filtros de Janelas Deslizantes tem o custo de execução $O(n^3)$. Com tal custo de execução, a execução deste algoritmo é impraticável com seqüências longas. Para resolver este problema foi criado um novo algoritmo, chamado de "Matriz de Pontos com Pontuação Mínima".

O algoritmo de Matriz de Pontos com Pontuação Mínima é baseado na programação dinâmica. Nele cria-se uma matriz onde são armazenados os valores do casamento ou não casamento entre os símbolos das posição M_i e M_j correntes. É utilizado no algoritmo um limite superior e um inferior para a pontuação das células. Estes limites são utilizados para que um trecho de duas subseqüência onde ocorra vários encontros ou desencontros não interfira nas pontuações seguintes. Um terceiro limite é utilizado para a exibição do ponto, ou seja, para a exibição de tal ponto, um número

```
MINIMUM-PONTUATION-DOT-MATRIX(M, N, LIM)
for i ← 0 to length[M]
do if m_j = n_0
      then matrix[i, 0] ← 1
      else  matrix[i, 0] ← 0
for i ← 0 to length[N]
do if m_0 = n_i
      then matrix[0, i] ← 1
      else  matrix[0, i] ← 0
for i ← 1 to length[M]
do for j ← 1 to length[M]
   do if m_i = n_j
         then if matrix[i − 1][j − 1] + 1 < UPPER_LIMIT
                  then matrix[i][j] ← matrix[i − 1][j − 1] + 1
                  else  matrix[i][j] ← matrix[i − 1][j − 1]
         else  if matrix[i][j] > 0
                  then matrix[i][j] ← matrix[i − 1][j − 1] − 1
                  else  matrix[i][j] ← 0
return matrix
```

Quadro 5: Algoritmo para matriz de pontos utilizando programação dinâmica

mínimo de casamentos devem ter ocorrido anteriormente. A recorrência matemática desde algoritmo é exibido na equação 3 e o algoritmo especificado no quadro 5.

$$S_{i,j} = min \begin{cases} UPPER_LIMIT, \\ max \begin{cases} S_{i-1,j-1} + \delta(m_i, n_j), \\ 0. \end{cases} \end{cases} \qquad \delta(m_i, n_j) = \begin{cases} 1\, if\ m_i = n_j, \\ -1\, if\ m_i \neq m_j. \end{cases} \tag{3}$$

A fórmula matemática apresenta uma recorrência, onde toda a célula $V[i, j]$ necessita do valor da célula $V[i-1, j-1]$ para computar o seu valor. O algoritmo primeiramente inicializa os valores da linha 0 e coluna 0, sendo que estes não possuem células anteriores para o cálculo do seu valor. Após, para cada célula $V[i, j]$, o algoritmo checa se ocorreu um casamento entre os pares de bases M_i e N_j, caso não ocorra, o valor da célula $V[i, j]$ será o valor da célula $V[i-1, j-1] - 1$. Porém, como observado na equação 3, há um limite inferior de 0 para cada célula. Caso ocorra um casamento entre os pares de bases, o valor da célula $V[i, j]$ será o valor da célula $V[i-1, j-1]+1$. Conforme a equação 3, há um limite superior, determinando que o valor de cada célula nunca poderá ser superior a UPPER_LIMIT.

Como este algoritmo percorre todos os elementos da seqüência M e compara com todos os elementos da seqüência N, seu custo de execução é $O(MN)$ e por utilizar uma matriz de tamanho M por N para armazenar os sub-resultados, seu custo de espaço é $O(MN)$. Desta forma, o custo de tempo de execução e de espaço do algoritmo possui um crescimento de ordem quadrática (n^2), inferior ao custo de execução do algoritmo de janelas deslizantes da seção 2.2, que é $O(n^3)$.

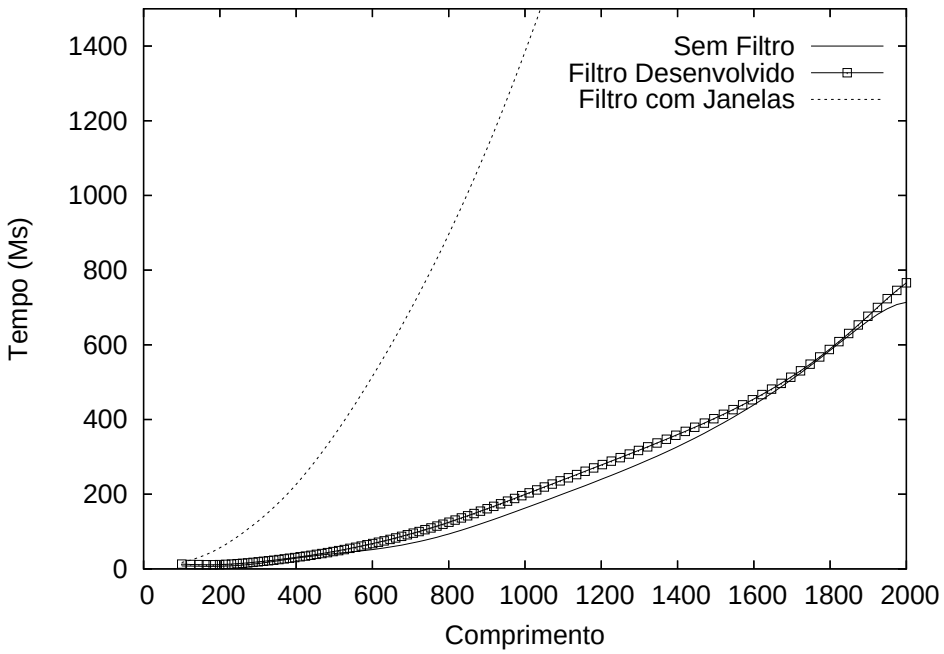

Figura 2: Comparação do tempo de execução dos Algoritmos.

Um gráfico comparando execuções dos algoritmos num computador *Atlhon XP 1700+* com 380 *megabytes* de memória e executando *Fedora Core 4* é exibido na figura 2. Nele é exibido o tempo de execução do algoritmo sem filtro, com filtro de janelas deslizantes apresentado na seção 2.2 e com o filtro desenvolvido, que utiliza o algoritmo de Matriz de Pontos com Pontuação Mínima. Observa-se que o tempo de execução do novo algoritmo aproximasse em muito do algoritmo sem filtro e conseqüentemente fica muito abaixo do tempo de execução do algoritmo com Filtro de Janelas Deslizantes. Enquanto uma matriz que contem duas seqüências com o comprimento de 1000 nucleotídeos demora aproximadamente $1,5$ segundos para serem filtradas no algoritmo de com Filtro de Janelas Deslizantes, no algoritmo de Matriz de Pontos com Pontuação Mínima a filtragem é executado em aproximadamente 200 milésimos de segundo. Para filtrar uma matriz com duas seqüências, cada uma com o comprimento de 2000 nucleotídeos, o algoritmo de janelas deslizantes executa em aproximadamente $4,6$ segundo, o novo algoritmo efetua a operação em aproximadamente 800 milésimos de segundo.

A qualidade dos resultados do Algoritmo de Matriz de Pontuação Mínima é igual ou superior ao do Algoritmo de Janelas Deslizantes. Porque enquanto as janelas deslizantes analisam apenas o conteúdo da janela atual, o Algoritmo de Matriz de Pontuação Mínima utiliza todos os dados calculados da diagonal dos pares de bases atuais.

O algoritmo retorna uma matriz de valores, que necessitam ser filtrados antes de serem exibidos. Este filtro é feito através de um limite mínimo que é utilizado no algoritmo especificado no quadro 6. Conforme descrito na seção 3.1, o algoritmo de exibição da matriz com o resultado do filtro serve para que apenas pontos que procedam um certo número de encontros sejam exibidos. Caso o valor mínimo do filtro seja 1, todos os encontros serão exibidos, como se a matriz não tivesse sido filtrada pelo algoritmo de Matriz de Pontos de Pontuação Mínima.

```
SHOW-PONTUATION-DOT-MATRIX(M, N, MIN)
for i ← 0 to length[M]
do for i ← 0 to length[N]
    do if matrix[i][j] ≥ MIM
          then PRINT_DOT(i, j)
```

Quadro 6: Algoritmo para exibição da matriz de pontos

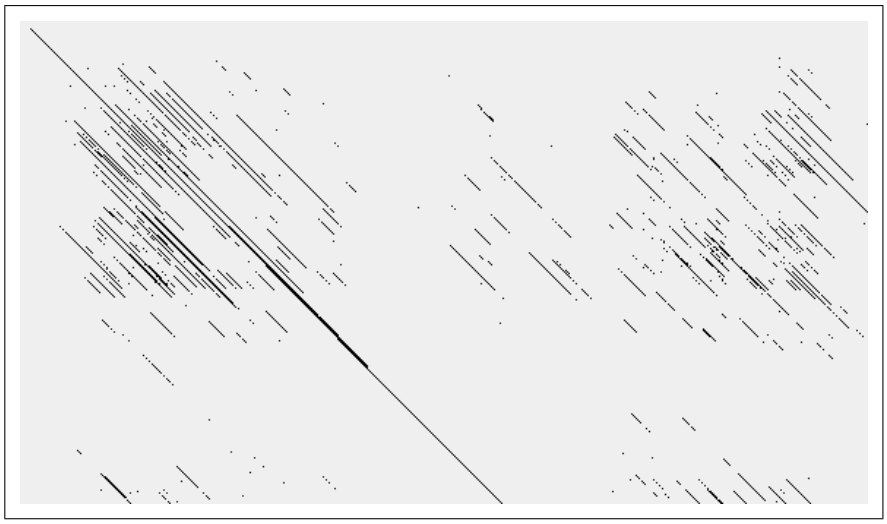

Figura 3: Comparação de duas seqüências utilizando o algoritmo de pontuação mínima

Através de testes, chegou ao valor mínimo para exibição de 7 pontos e um valor máximo de 11 pontos. Uma exibição de parte de uma comparação de seqüências genéticas utilizando o filtro de Matriz de Pontos com Pontuação Mínima com estes valores especificados é exibido na figura 3. Na imagem dois trechos de duas seqüências de animais da família das Abelhas são exibidos. Pode-se visualizar várias retas formadas na imagem. Cada reta desta imagem significa que há um alto grau de similaridade entre as seqüências naquele trecho. Por exemplo, se a reta começa na posição 40 da seqüência M (seqüência horizontal) e termina na posição 90 e começa na posição 70 da seqüência N (seqüência vertical) e termina na posição 120, signifi/ca que a sub-seqüência 40 a 90 da seqüência M e sub-seqüência 70 a 120 da seqüência$N$ são muito similares.

4 Conclusão

Muitos são os métodos e algoritmos para comparação de seqüências genéticas, compara-los e escolher um melhor é uma tarefa difícil. Cada técnica possui suas vantagens e desvantagens e principalmente sua área de aplicação. As heurísticas apresentadas na seção 2.4 são mais rápidas e requerem menos espaço para comparar e encontrar seqüências similares do que programação dinâmica, porém elas são menos sensíveis a pequenas diferenças. Estas características fazem delas úteis em

busca em banco de dados de seqüências por similaridades. Por exemplo, uma nova versão(Altschul, S. F.; Scheffer, A. A., 1997) do algoritmo BLAST, é utilizando num dos maiores banco de dados de seqüências genéticas do mundo, o NCBI(NCBI, 2005). O Modelo Oculto de Markov é útil quando se tem um bom conjunto de seqüências para treinar o modelo e deseja predizer algumas informações sobre a seqüência, por exemplo, se ela é codificante, ou qual será a família da proteína.

Para a comparação de duas seqüências que possuem alguma similaridade previamente conhecida, é interessante uma comparação mais detalhada, neste caso uma matriz de pontos para a visualização das similaridades e alinhamentos que utilizam os algoritmos vistos na seção 2.3 são úteis e funcionais para este tipo de comparação. O grande problema das comparações genéticas é o grande volume de dados, e isto torna a utilização de algoritmos de crescimento experiências difíceis de serem aplicados, tornando necessário a utilização de heurísticas ou modelos probabilísticos para a busca de similaridades.

O algoritmo e o *software* [1] apresentados proporcionam ao usuário a opção de visualizar similaridade de seqüências num tempo hábil e tendo a sua disposição o melhor detector de padrões existente, o cérebro. Conforme visto na figura 3, o algoritmo de Matriz de Pontos com Pontuação Mínima da ao usuário a opção de visualizar as partes mais semelhantes entre as seqüências e então o usuário pode seleciona-las para alinha-las utilizando o algoritmo de alinhamento global visto na seção 2.3.1.

Referências

Altschul, S. F. et al. Basic local alignment search tool. *Journal of Molecular Biology*, v. 215, p. 403–410, 1990.

Altschul, S. F.; Scheffer, A. A. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, v. 25, n. 17, p. 3389–3402, 1997.

BIOJAVA, Comunidade. *BioJava*. 2005. Disponível em: <http://www.biojava.org>. Acesso em: 1 Ago. 2005.

C. Charras; T. Lecroq. *Exact string matching algorithms*. 1996. Disponível em: <http://www-igm-.univ-mlv.fr/~lecroq/string/>. Acesso em: 20 Ago. 2005.

KOSKI, Timo. *Hidden Markov Models For Bioinformatics*. Dordrecht: Kluwer Academic Publishers, 2001.

Lipman, D.J.; Pearson, W.R. Rapid and sensitive protein similarity searchers. *Science*, v. 227, p. 1435–1441, 1985.

MOUNT, David W. *Bioinformatics*. Cold Spring Harbor: Cold Spring Harbor Laboratory, 2004.

NCBI. *National Center for Biotechnology Information*. 2005. Disponível em: <http://www.ncbi-.nlm.nih.gov>. Acesso em: 1 Ago. 2005.

Neil C. Jones; Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms*. Cambridge: The MIT Press, 2004.

Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest. *Introduction to Algorithms*. Cambridge: The MIT Press, 1999.

[1] O *download* do *software* pode ser feito em *http://www.inf.furb.br/ albrecht/interesses/bioinformatica/seminco2005/*