Comunicado Técnico 217
ISSN 1678-961X
Dezembro, 2014
Santo Antônio de Goiás, GO

Uma Nova Proposta para se Obter Tandem Repeats em Sequências Genômicas

Marcelo Gonçalves Narciso[1]
Michel Eduardo Beleza Yamagishi[2]

Introdução

O genoma de um organismo pode ser representado por uma grande sequência de nucleotídeos. Esta sequência, comumente chamada de sequência genômica, é composta pelas bases nitrogenadas A, T, C, G (Adenina, Timina, Citosina e Guanina). Com a conclusão de diversos projetos sobre genomas, vieram à tona alguns fatos inesperados como, por exemplo, a constatação que, aproximadamente, 50% dos genomas de mamíferos são compostos por sequências repetitivas (BANNERT; KURTH, 2004), sendo que, nas plantas, essa proporção pode alcançar fenomenais 90% (KAZAZIAN, 2004; SANMIGUEL et al., 1996). As sequências repetitivas podem ser classificadas como Tandem Repeats (TR) ou Interpersed Repeats (IR). São exemplos do primeiro grupo os microssatélites, minissatélites e telômeros; do segundo grupo, os elementos genéticos móveis ou, simplesmente, transposons. Há aplicações instrumentais para os dois grupos. Os microssatélites são utilizados como marcadores moleculares em estudos de melhoramento genético de plantas e de animais para identificação de genes de interesse científico ou econômico (LANDER; BOTSTEIN, 1989; SHARMA et al., 2007), ou, mais comumente, em testes de paternidade (BOTSTEIN et al., 1980; ROCHETA et al., 2007); já os transposons podem ser aplicados em terapia genética (IVICS; IZSVAK, 2006).

Cabe ressaltar que identificar as sequências repetitivas em um genoma pode ser visto como um problema de alinhamento múltiplo de sequências, que é um problema NP-Hard (JIANG et al., 2000; WANG; JIANG, 1994). O simples alinhamento local de duas sequências já é um problema computacionalmente caro (WATERMAN; SMITH, 1981), e exigiu o desenvolvimento de heurísticas como o FASTA (WILBUR; LIPMAN, 1983) e o BLAST (ALTSCHUL et al., 1990) para acelerar, e, consequentemente, tornar viável a busca em bases de dados gigantescas. A identificação de TR é um caso particular da identificação de sequências repetitivas, mas nem por isso menos complexa.

Para facilitar a compreensão sobre TR, nada melhor que um exemplo:

GAGAATCGATCGATCGATCGATCGATCGCTCTCTCTCT

Observe que na sequência acima existem as repetições GAGAGAGAGA, ATCGATCGATCGATCGATCGATCG e CTCTCTCTCT. Cada uma destas sequências é conhecida como TR. Há um padrão que se repete consecutivamente, por exemplo, GA, ATCG ou CT, respectivamente, no caso ilustrado. O número de repetição de cada padrão pode variar bastante de organismo para organismo ou de um indivíduo para outro indivíduo, e por isso podem ser usadas

[1] Engenheiro elétrico, Ph.D. em Bioinformática, pesquisador da Embrapa Arroz e Feijão, Santo Antônio de Goiás, GO, marcelo.narciso@embrapa.br
[2] Matemático, Doutor em Matemática Aplicada, pesquisador da Embrapa Informática Agropecuária, Campinas, SP, michel.yamagishi@embrapa.br

Embrapa

como marcadores moleculares, que podem ser compreendidos como sendo um RG (identificação) de um organismo, isto é, os marcadores moleculares podem ser usados para identificar um indivíduo de uma espécie ou separar uma espécie de outra.

Há uma coleção significativa de softwares para identificar TR. Talvez o mais popular seja o Tandem Repeat Finder (http://tandem.bu.edu/trf/trf.html). E dada a complexidade do problema, a quantidade de novos algoritmos e metodologias cresce a cada ano (KURTZ et al., 2001; PRICE et al., 2005; ZHI et al., 2006), incorporando inclusive conceitos algébricos como o de quaternions (BRODZIK, 2007). Contudo, os algoritmos existentes mais usados não conseguem identificar todos TR, principalmente, na presença de mismatches e Indels que podem ocorrer nessas regiões (MABUCHI et al., 2012). Além disso, esses algoritmos limitam o tamanho máximo da sequência onde serão buscadas as TR (não conseguem executar a aplicação para um genoma de arroz, por exemplo) ou necessitam de grande quantidade de processamento e memória para serem executados (NARCISO; YAMAGISHI, 2011). Enfim, existem limitações que requerem aprimoramentos para obter algoritmos que possam ser executados de forma a identificar todos os TR, por maior que estes sejam e também em um tempo computacional razoável.

Este trabalho apresenta um novo algoritmo, chamado ConvolutionTR, de propósito geral, para se obter todos os TR, qualquer que seja o tamanho, em uma sequência de tamanho arbitrário, mas finito. Este algoritmo se baseia no conceito de convolução (CONVOLUÇÃO, 2013) e é uma nova aplicação para este conceito. Uma das vantagens deste algoritmo é que não necessita de estrutura de dados complexas e também não necessita de uma grande quantidade de memória para ser executado. Além disso, tem tempo de execução um pouco menor que os algoritmos similares, conforme poderá ser visto nos resultados a serem mostrados neste trabalho.

Materiais e Métodos

Foi construído um algoritmo para obter todos os TR possíveis para uma dada sequência de bases nitrogenadas. Entretanto, a motivação de ter também um algoritmo que fosse executado o mais rápido possível levou a vários testes com algoritmos diversos até que se obteve a melhor resposta em termos de obter todos os TR e também que o algoritmo tenha um tempo de execução razoável. Para exemplificar, foi tentada uma metodologia de armazenar partes das sequências em forma de árvore, para posteriormente ter uma busca rápida, porém ficou muito complexo o algoritmo e o rendimento não foi o esperado. Assim, ao se comparar a sequência com ela mesma, e verificar o fato de quanto existe um atraso de k bases (k é um número inteiro) entre as sequências, observou-se que eram obtidos TR de período k. Um detalhe que foi observado é que não há limites para o valor de k e, desta forma, é possível se obter até mesmo TR com períodos de 100, 1.000, 10.000, e assim por diante. Na seção “Descrição do Algoritmo ConvolutionTR” haverá uma melhor explicação para isso. Uma vez que a metodologia de busca de TR foi determinada, então foi construído um programa computacional para executar este algoritmo.

O algoritmo proposto, ConvolutionTR, foi implementado com a linguagem C, e pode ser compilado e executado em qualquer sistema operacional. Basta ter algum compilador C instalado, o qual é software livre. Por enquanto, este software, ConvolutionTR, só roda como linha de comando e tem como entrada os parâmetros de busca de TR e também o arquivo no qual serão detectados os TR. Ao executar o programa ConvolutionTR, sem nenhum parâmetro, este mostra todos os parâmetros disponíveis para busca (uma espécie de Help/Ajuda) e também exemplifica como rodar o programa.

Foram feitos vários testes com este algoritmo para se certificar de que este realmente consegue obter os TR de uma sequência e, em seguida, uma vez que este foi validado, optou-se por comparar este algoritmo com outros que são usados correntemente, os quais foram citados na introdução deste trabalho. Assim, procurou-se comparar o convolutionTR com outros algoritmos para TR com período 1, 2, ...,n e ver o rendimento de cada algoritmo para os períodos citados e também qual o período máximo que cada algoritmo podia obter. Desta forma, uma vez que os algoritmos tenham sido comparados, será possível avaliar o desempenho do algoritmo ConvoluionTR e assim, com os resultados obtidos, poder-se-á ver o quão bom são os algoritmos para cada caso. Na seção “Resultados e Discussão” têm-se estas

comparações para verificar se o algoritmo proposto tem qualidade ou não e também o desempenho dos demais algoritmos conforme o tamanho da sequência.

Conforme mencionado anteriormente, este programa, ConvolutionTR, foi comparado com alguns dos mais conhecidos algoritmos para se obter TR. Esses algoritmos/softwares são: TRF (Tandem Repeat Finder), disponível em http://tandem.bu.edu/trf/trf.html; MREPS, disponível em http://bioinfo.lifl.fr/mreps/; Swan (Structure Word Analyser), disponível em http://favorov.bioinfolab.net/swan/tool.html; Phobos, disponível em http://www.ruhr-uni-bochum.de/spezzoo/cm/cm_phobos.htm; e Reputer (KURTZ et al., 2001), que é obtido com os próprios criadores do programa e o endereço de contato está no trabalho citado (KURTZ et al., 2001).

O ambiente computacional no qual foram feitos os testes (comparação) é CentOs linux, v 5.5, kernel 2.6.18-194.3.1.el5, 64 bits, 16 Intel(R) Xeon(R) CPUs, 2.80GHz por CPU, cache size 8192 KB.

Descrição do Algoritmo ConvolutionTR

Em Matemática, a convolução é definida como um operador linear que, a partir de duas funções dadas, f(u) e g(u), resulta numa terceira função, h(u), a qual mede a área subentendida pela superposição de f(u) e g(u) em função do deslocamento existente entre elas. Mais detalhes sobre convolução podem ser vistos em Convolução (2013). A ideia central deste algoritmo, ConvolutionTR, é similar, em parte, ao operador convolução ou ainda à correlação cruzada. Daí o nome de ConvolutionTR.

O algoritmo compara uma sequência com ela mesma defasada de k posições e um de seus objetivos é simplificar a forma de obter TR e também poder obter todos os TR com maior precisão.

Assim, para o caso do DNA, se k for igual a 1, será possível obter as repetições de uma mesma base nitrogenada A, T, C ou G (período da sequência é igual a 1). Se k for igual a 2, será possível obter a repetição de sequências de duas bases (por exemplo, AT, CT, GC, AG, etc.). Se k for igual a 3, será possível obter uma sequência de repetições de 3 bases (pelo menos uma destas é diferente e o período é 3), e assim por diante, valendo para k = n, n é um número inteiro e positivo. Para exemplificar, veja a sequência abaixo.

ATCGGATGATTTTTTATAGGAC

Comparando a mesma sequência com o atraso de uma posição, tem-se o seguinte:

```
A T C G A T G A T T T T T T A T A G G A C
  A T C G A T G A T T T T T T A T A G G A C
  0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0
```

A primeira e a segunda linhas são a mesma sequência, mas defasadas de uma única base entre elas. A terceira linha é composta somente por 0 e 1, onde 0 indica bases diferentes e 1 indica bases iguais. Observe que existe uma sequência TTTTTT, que é facilmente notada na terceira linha pela sequência 111110. Por causa do atraso das duas sequências, o número de valores iguais a 1 é uma unidade a menos que a quantidade da repetição. Neste caso, para seis repetições (TTTTTT), têm-se cinco repetições de 1 (11111).

A ideia é generalizada para qualquer quantidade de bases. Por exemplo, para exemplificar a repetição de uma sequência com duas bases, temos:

ATCGGATGATATATATATATC

Se compararmos esta sequência com ela mesma com o atraso de duas bases, tem-se a seguinte comparação:

```
A T C G A T G A T A T A T A T A T A T C
    A T C G A T G A T A T A T A T A T A T A T C
  0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
```

Observe que se tem uma sequência (ATATATATATAT) com período de duas bases e frequência de seis repetições. Na terceira linha existem dez repetições de 1 (1111111111), que mostram a comparação entre as duas sequências com atraso de duas bases. Como o atraso é de duas bases, e tem-se dez repetições, então basta dividir 10 por 2 (número do atraso, que indica o período da sequência desejada). Assim, tem-se 5, que é igual à frequência da sequência (6) menos 1 (o atraso).

De forma análoga, seja uma sequência de período 3 (três bases que se repetem) e com frequência 7.

ATGAATAATAATAATAATAATCGAGGT

Comparando essa sequência com ela mesma com o atraso de três posições, tem-se o seguinte:

```
A T G A A T A A T A A T A A T A A T A A T C G A G G T
    A T G A A T A A T A A T A A T A A T A A T C G A G G T
    1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0
```

Observe que agora se tem uma sequência de 1 (111111111111111111) com 18 repetições. Como o atraso é igual a 3 bases, divide-se então 18 por 3, e tem-se 6, que é o número da frequência de repetições menos 1 (período 3 e frequência 7).

Assim, de forma geral, se tiver uma sequência de período p, e frequência de repetições f, e k for o número do atraso, tem-se que, se o número n de comparações iguais a 1 obtidos for igual a n = (f*k – k) = (f – 1)*k e k = p, então temos uma sequência periódica, com período p (igual ao atraso k) e frequência f. Matematicamente, seria:

f = (n/k) + 1 e p = k

O valor da divisão de n/k deve ser transformado para inteiro, desprezando-se a casa decimal.

Com essa ideia central, é possível descrever um algoritmo rápido para se obter as sequências de repetições desejadas. Pode-se fazer um laço de repetição (loop) com i variando de 1 a S (período máximo de busca) e comparar a sequência com ela mesma com o atraso i. De forma geral, essa comparação, posição por posição, seria:

sequência (j) *versus* sequência (j – i)

j variando de i a F, F é a posição final da sequência (tamanho da sequência). Assim, para cada valor de i, ter-se-iam sequências de 1 ou 0, resultado da comparação, e cada uma das sequências de 1 seria avaliada para ver se a divisão da frequência desta sequência por i estaria exata ou a parte inteira maior que um valor pré-estabelecido. Observe que se procura a sequência de período mínimo i (múltiplos de i não valem).

Um detalhe a ser levado em conta é que se podem ter períodos nos quais existam outros períodos (MATROUD et al., 2012). Por exemplo, veja a sequência abaixo.

AAAAATAAAAATAAAAATAAAAATAAAAAT

A sequência tem um período AAAAAT. Porém, este período tem o período AAAAA. Assim, quando no atraso de uma base (k = 1), o algoritmo vai encontrar a sequência AAAAA. Quando o atraso for igual a 6 (k = 6), o algoritmo vai encontrar a sequência AAAAAT. O algoritmo mostra todos esses períodos e frequências desde que sejam mínimos. Assim, mostraria os períodos A e AAAAAT, com a respectiva posição de início. Fica a cargo de quem vai analisar ver qual o período mínimo que começa na dada posição que vai ser considerado.

Sobre período mínimo, mais um caso particular, veja a sequência CAAAAAAAAAAT. Esta sequência poderia ter os períodos A (frequência 10), AA (frequência 5), AAA (frequência 3), porém, como o período mínimo é 1, então frequência = 10 é o valor que será mostrado pelo algoritmo. Se uma sequência é múltipla de uma que já foi obtida, então esta sequência que se repete será descartada em favor da que tiver um período mínimo (claro que se referem às mesmas bases da sequência em estudo).

O Algoritmo simplificado

O algoritmo para este tipo de análise de sequência para obter TR seria.

Ler a sequência de interesse. Seja F o tamanho da sequência. Cada elemento j da sequência lida é representado por sequência(j). Assim, "sequência" é um vetor e sequência(j) é um elemento do vetor (que pode ser A, C, G, T, N, Y, R).

Tamanho Mínimo da Frequência = 5; // altere aqui se quiser uma frequência maior

```
Repita para i = 1, 2, ...., S,  S é o período máximo desejado
      contador_de_1 = 0;
       Repita para j = i, i+1,i+2, i+3, ...F - i
         Se (sequência(j) = sequência(j – i) )
              então contador_de_1 = contador_de_1 + 1;
              senão
                  contador_de_1_anterior = contador_de_1;
                      contador_de_1 = 0
         Fim_Se
Se (contador_de_1 = 0 e (i+contador_de_1_anterior / i) >= Tamanho Mínimo da Frequência )
      então
          pos = j -  i* contador_de_1 + i;
          //
          // Imprimir resultado parcial
             imprimir ("posicao inicial >>" + pos)
               repita para r = pos, pos +1,...j-1
                   imprima (sequência(r))
                       fim_repita
                  Fim_Se
              Fim_Repita para j
            Fim_Repita para i
```

Acima está o algoritmo, de forma simplificada, que seria usado para obter TR de uma sequência. Obviamente, outros detalhes, como, por exemplo, verificar se a sequência obtida é múltipla de outra sequência já obtida anteriormente, não estão descritos acima para facilitar o entendimento do propósito principal do algoritmo.

Limite de Memória RAM

Geralmente as sequências genômicas são grandes, atingindo tamanhos de centenas de MB a até dezenas de GB. No último caso, a fim de evitar-se a necessidade de computadores com muita memória RAM, é evitada a leitura da sequência em uma única variável para executar o algoritmo. É usada a estratégia de dividir a sequência em partes para efeito de leitura, desde o início até o fim; e para cada parte lida é executado o algoritmo acima descrito. Contudo, é necessário considerar as sequências anterior e posterior, pois um TR pode ter começado no final da primeira sequência lida e terminar na próxima, ou ainda, em alguns casos, continuar na sequência subsequente a atual.

Para ilustrar o procedimento, imagine um arquivo que tenha uma sequência com 1.000.000 de bases. Admita que esse tamanho seja dividido 100 leituras de 10.000 bases. Imagine, sem perda de generalidade, que se deseja obter sequências de até 100 bases como período e que se esteja na leitura do décimo pedaço da sequência total. Assim, tem-se que levar em conta se a sequência anterior, ao ser analisada, terminou em algum atraso k com o valor 1. Se terminou, isso significa que a sequência de 1 ainda pode continuar na sequência atual que está sendo analisada. Então, todas as sequências parciais de 1 que vieram da sequência anterior deverão ser guardadas (aquelas que se referem ao final da sequência anterior e que vão continuar na sequência atual). Esse detalhe tem que ser adicionado ao algoritmo acima, pois ele é para o caso de quando a sequência é pequena suficiente para ser guardada em uma variável. O desempenho do algoritmo será tanto melhor quanto maior for a parcela da sequência total que puder ser armazenada na variável (quanto mais memória, melhor).

Lidando com Mutações

Uma característica de toda sequência genômica é a presença de mutações. Desde pequenas deleções e/ou inserções até mesmo substituições de um único nucleotídeo (SNPs). Essas mutações também podem ocorrer em regiões repetitivas, em particular, em TR. O algoritmo ConvolutionTR permite a identificação de um número finito de mutações sem perda de sua eficiência computacional. Será descrito a seguir a identificação dessas mutações em TR.

Veja uma sequência tal como a que está a seguir:

ATCATCATCATCAT**G**ATCATCATC

Observe que esta sequência tem uma base G no lugar de uma base C no TR. Assim, ao se fazer a comparação com atraso de k, tem-se o seguinte resultado:

```
A T C A T C A T C A T C A T G A T C A T C A T C
      A T C A T C A T C A T C A T G A T C A T C A T C
      1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1
```

Observe que para o caso da mutação, existem dois zeros na sequência de 1 (linha 3 acima). A distância entre os valores '0' é 3 (posição do último zero menos a posição do primeiro zero), o que é igual ao período do TR. Este caso trata das mutações ditas internas. Se a mutação estiver nos extremos, as considerações são um pouco diferentes. Veja os casos a seguir:

```
C T C A T C A T C A T C A T C A T C A T C
      C T C A T C A T C A T C A T C A T C A T C      (1)
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

A C C A T C A T C A T C A T C A T C A T C
      A C C A T C A T C A T C A T C A T C A T C      (2)
0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

A T A A T C A T C A T C A T C A T C A T C
      A T A A T C A T C A T C A T C A T C A T C      (3)
0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

A T C T T C A T C A T C A T C A T C A T C
      A T C T T C A T C A T C A T C A T C A T C      (4)
0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

A T C A A C A T C A T C A T C A T C A T C
      A T C A A C A T C A T C A T C A T C A T C      (5)
0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
```

Observando os passos de (1) a (5), nota-se que um agrupamento de 1s pode aparecer sem interrupções (sequência (1)), com um zero entre 1s (sequência (2)) e dois zeros entre 1s (sequências (3), (4) e (5)). Estas são as variações possíveis quando uma mutação aparece nas extremidades da sequência. A regra geral para mutação, para qualquer período, é a seguinte:

Se a mutação não estiver nos extremos (ou seja, for interna), a distância entre os 0s da sequência binária de 0s e 1s é igual ao período do tandem.

- Se a mutação estiver nos extremos, em uma das duas bases nitrogenadas do tandem, basta considerar tal como ilustrado nos casos (1) (a primeira base nitrogenada é a mutação) e (2) (a segunda base é a mutação e a primeira base não é).

Estas considerações foram feitas para o caso da mutação ser apenas uma base, que é a proposta deste trabalho. Porém, este algoritmo pode ser expandido para mais bases. A regra geral seria:

"Se a mutação não estiver nos extremos, a distância entre os dois períodos de 0s da sequência de 0s e 1s é igual ao período do tandem, isto é, a distância entre o último 0 da primeira sequência de 0s e o último 0 da segunda sequência de 0s é igual ao período do TR. Por exemplo, veja a sequência:

ATCATCATCATCAT**G**GTCATCATC

A sequência, comparada com ela mesma, considerando o atraso de três bases, tem o seguinte resultado:

```
A T C A T C A T C A T C A T G G T C A T C A T C
      A T C A T C A T C A T C A T G G T C A T C A T C
      1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1
```

Observe que a sequência 00100 tem uma 00 + 1 + 00. A distância do segundo zero do primeiro grupo de 00 até o segundo zero do segundo grupo de 00 é igual a 3. Este trabalho não considera um TR com mutação igual a 3, mas se considerasse, a regra seria a mesma, isto é, haveria um grupo do tipo 000 + 1s + 000 e a distância entre os últimos 0 de cada sequência de 0s é igual ao período do TR e a quantidade de 0s de cada grupo é igual a quantidade de bases que sofreram mutação.

- Se a mutação estiver no início da cadeia, a quantidade de 0s será igual ao período do TR imediatamente antes do início da cadeia de 1s.

- Se a mutação estiver a partir da segunda cadeia, com r bases diferentes, então, a partir do início da cadeia de 1, verificar se, a r posições à esquerda do início da cadeia de 1s tem uma base idêntica a que está no início da cadeia de 1s. Vale a pena ressaltar que esta consideração é apenas em termos de lógica, pois em termos biológicos pode não ser uma mutação e sim uma coincidência.

Se a mutação estiver nos extremos, em uma das duas bases nitrogenadas do TR, basta considerar tal como ilustrado nos casos (1) (a primeira base nitrogenada é a mutação) e (2) (a segunda base é a mutação e a primeira base não é).

Lidando com InDELs

Sobre **InDEL** (**inserção e deleção**), este tópico mostra como pode ser obtida a inserção ou a deleção, a partir deste algoritmo, que considera apenas uma base ter sido removida (deleção) ou inserida (inserção). Veja o caso da sequência abaixo:

ATGGATCATCATCATATCATCATCATCGG

Observe que se tem a sequência de ATC e esta tem um trecho (ATCATCATC**ATATC**ATCATCATC) no qual a base C está ausente. Fazendo o procedimento que foi feito anteriormente, para quando o algoritmo tiver a iteração para k = 3 (atraso de 3 bases), tem-se o seguinte:

```
A T G G A T C A T C A T C A T A T C A T C A T C A T C G G
      A T G G A T C A T C A T C A T A T C A T C A T C A T C G G
      0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0
```

Observe que a quantidade de zeros é igual a 3, que é o período da sequência, descontando o evento de deleção. Essa é uma regra geral. Se a sequência tiver algum caractere repetido dentro do período do TR, essa regra também vale. Por exemplo, veja a sequência a seguir:

AATA AATA AATA AATA AATA AAA AATA AATA

Nesta sequência existe uma deleção (retirada de uma base T). Comparando esta sequência com ela mesma, defasada em 4 bases, que é o período do TR, tem-se a seguinte comparação:

```
A A T A A A T A A A T A A A T A A A T A A A A A A T A A A T A
          A A T A A A T A A A T A A A T A A A T A A A A A A T A A A A
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1
```

Nesta sequência, contando-se o número de bases entre um zero e o outro é zero, tem-se que é igual a 4, que é o período da sequência. Ampliando a regra anterior, se entre um zero e o outro zero tiverem k bases e k for o período do atraso, então foi encontrada uma deleção. O primeiro zero é quando ocorre uma deleção na sequência de cima e o último zero é quando ocorre a mesma deleção na sequência de baixo (defasada de período k) com respeito à comparação de uma sequência com ela mesma. Do lado esquerdo do primeiro zero e do lado direito do segundo zero da comparação existem apenas 1s. Assim, qualitativamente, a sequência resultante é do tipo:

11111111110XXX0111111111

O valor de XXX pode ser 000 ou 111 ou 010 ou 101, etc. O que vale é o fato de zeros estarem iniciando e terminando uma sequência de tamanho k e à esquerda do primeiro zero e à direita do segundo zero só existirem 1s.

Para a **Inserção** de uma base em uma dada cadeia repetitiva, tem-se um raciocínio análogo ao que foi feito para a deleção. Para exemplificar, veja a cadeia abaixo:

ATCATCATCATCATCGATCATCATC

Observe que esta cadeia tem período 3 (ATC) e frequência 8. Fazendo uma comparação com o atraso de um período (k = 3), tem-se o seguinte:

```
A T C A T C A T C A T C A T C G A T C A T C A T C
      A T C A T C A T C A T C A T C G A T C A T C A T C
      1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1
```

Tal como foi descrito para o caso da deleção, a inserção tem uma saída da cadeia de 0s e 1s do tipo:

1111111111110XXX0111111111

Na sequência qualitativa acima, a distância do primeiro zero até o último é igual ao período mais 1 (k + 1). Os valores de X podem ser 0 ou 1 (1 aparece quando existem bases repetidas dentro do período do TR).

Assim, para se obter uma deleção, quando da análise de um período k, ao se obter uma sequência do tipo 1111110XXX0111, no qual 0XXX0 tem tamanho do período em questão, então o trecho de sequência de bases nitrogenadas em análise é candidato a deleção. De modo análogo, se o tamanho da sequência 0XXX0 for k + 1, então o trecho considerado de bases nitrogenadas é candidato a inserção.

De forma geral, quando se obter uma sequência de 1s, basta verificar se é uma sequência repetitiva ou ainda se é uma mutação (considerando cadeias de 1s, 0 e 1s) ou deleção (somas das cadeias de 1s dos atrasos k e k – 1 tiverem como resultados uma cadeia de 1s 0 1s) ou inserção (somas das cadeias de 1s dos atrasos k e k +1 tiverem como resultados uma cadeia de 1s 0 1s).

O Algoritmo

Desta forma, um algoritmo geral para se obter um TR com período e frequência com ou sem mutação, deleção e inserção ficaria da seguinte forma:

Ler a sequência de interesse (vetor sequencia). Seja F o tamanho da sequência e sequencia(j) um elemento deste vetor na posição j.

```
TamanhoMinimoDaFrequência = 5; // altere aqui se quiser uma frequência maior
  Repita para i = 1, 2, ...., S, S é o período máximo desejado
    contador_de_1 = 0;
    Repita para j = i, i+1,i+2, i+3, ...F
      Se ( sequência(j) = sequência(j – i) )
          então contador_de_1 = contador_de_1 + 1;
          senão
            contador_de_1_anterior = contador_de_1;
             contador_de_1 = 0
      Fim_Se
          Se sequência tem mutação
            Então imprimir mutação
              senão
                Se sequência tem deleção
                  Então imprimir deleção
                    senão
                      Se sequência tem inserção
                        Então imprimir inserção
                          senão imprimir sequência tandem
                            Fim_Se
                          Fim_Se
                        Fim_Se
                      Fim_Repita para j
                    Fim_Repita para i
```

Este algoritmo não tem os detalhes da busca por indel ou mutação, mas os quadros comparativos de como fazer a busca foi descrito e cada programador poderá fazer da sua maneira. Vale a pena mencionar que este algoritmo vale para apenas sequências com ou sem mutação e indel de apenas uma ou duas bases.

Resultados e Discussão

Nesta seção, serão comparados os algoritmos descritos pa164ra ver o desempenho de cada um quanto à tempo de execução e também quanto à quantidade de TR que cada algoritmo consegue obter. Isto mostra o quanto, na prática, cada algoritmo é eficiente.

As Tabelas 1 e 2 mostram comparações entre a nova proposta e o algoritmo TRF para tempo de execução e também para TR obtidos quanto à sua frequência.

Tabela 1. TRF x ConvolutionTR - tempo de execução de cada algoritmo.

Sequência	TRF (s)	ConvolutionTR (s)
nipponbareCromossomos-1-to12-Sy-Un.fa	1067	570
Oryza_sativa.MSU6.dna.chromosome.1.fa	107	66
TIGR-v6.seq-1-to-12	1039	569
arabThaliana.fa-1-to-5	300	187
arabThalianaCromo5_NC_003076.fa	61	41

Como pode ser observado, ConvolutionTR é mais rápido que o TRF. Os dados usados para fazer esta comparação são o genoma da variedade de arroz Nipponbare (arquivo fasta), que está no arquivo nipponbareCromossomos-1-to12-Sy-Un.fa, citado no primeiro item da coluna "Sequência". O segundo arquivo que contém toda a sequência do cromossomo 1 de Nipponbare está descrito na segunda linha, primeira coluna, o arquivo Oryza_sativa.MSU6. dna.chromosome.1.fa. Outros dados de entrada para comparação entre estes dois algoritmos estão descritos nas demais linhas da primeira coluna. Como pode ser observado, o desempenho do algoritmo ConvolutionTR melhora com o aumento do tamanho da sequência considerada, ou seja, quanto maior for a sequência, maior é a diferença de tempo de execução entre os algoritmos ConvolutionTR e TRF.

Na Tabela 2 a seguir, foi usado o arquivo nipponbareCromossomos-1-to12-Sy-Un.fa para se obter todos os TR possíveis conforme o algoritmo usado. Pode-se observar que o software ConvolutionTR identifica mais TR quando comparado aos algoritmos existentes. Assim, ConvolutionTR é mais eficiente do que o algoritmo TRF tanto em tempo de execução (Tabela 1) quanto a quantidade de TR obtidos. Observa-se que o software Phobos é também muito eficiente e foi o que mais se aproximou do software ConvolutionTR em termos de quantidade de TR obtidos.

Tabela 2. TRF x ConvolutionTR – Frequência de tandem x número de sequências repetidas. Os 12 cromossomos pertencem à variedade de arroz Nipponbare e o período de tandem é igual ou maior que 2.

Frequência de TR	TRF	Convolution TR	Phobos	MREPS	Swan
4	60312	238004	233819	13282	11987
5	10912	57941	57117	7403	5671
6	3624	20359	19825	4420	4321
7	1661	9924	9510	2049	1987
8	941	5154	4932	1174	1087
9	626	3179	3046	742	762
10	424	1997	1941	436	359
11	319	1310	1281	273	271
12	223	960	944	273	325
13	167	744	730	157	170
14	140	631	616	150	172
15	146	579	579	144	164
16	118	467	458	107	97
17	103	414	406	80	78

As Tabelas 3 e 4, a seguir, mostram uma comparação de tempo de execução e número de TR obtidos por todos os algoritmos. De todos os algoritmos, o MREPS é, sem dúvidas, o mais rápido; entretanto, o número de TR identificado é muito inferior ao número dos outros algoritmos. Por exemplo, o ConvolutionTR e o Phobos identificam quase uma ordem de grandeza a mais de TR que o MREPS. Em estudos voltados exclusivamente a sequências repetitivas do tipo tandem, essa diferença pode significar muito.

Tabela 3. Tempo de execução para obter todos os tandem repeats de período variando de 1 a 100 e frequência maior do que 8 para DNA da variedade Nipponbare.

Software	Tempo de execução (s)	Quantidade de Tandem
ConvolutionTR	548	149014
MREPS	30	18460
Phobos	1013	146591
TRF	964	82663
Swan	13912	27061

A maior quantidade de TR para um dado período é para a repetição de uma base (AAAAAAAAAAAA ou TTTTTTT, por exemplo). Ao observar as Tabelas 3 e 4, pode ser notado que a quantidade de TR de período igual a 1 é bem maior do que 50% (para os testes feitos e resumidos nas Tabelas 3 e 4, está maior do que 80%).

Tabela 4. Tempo de execução para obter todos os tandem repeats de período variando de 2 a 100 e frequência maior do que 8 para DNA da variedade Nipponbare.

Software	Tempo de execução (s)	Quantidade de Tandem
Phobos	985	15347
Swan	1391	4358
ConvolutionTR	542	15691
TRF	964	3781
MREPS	19	3448

Tabela 5. Todos os períodos, a partir de 2, e com todas as frequências abaixo (4 a 17).

Frequência de TR	TRF	Convolution TR	Phobos	MREPS	Swan
4	19849	68717	67710	8055	7855
5	3430	14131	13978	4375	4076
6	1011	4478	4433	2726	2534
7	493	2213	2188	1383	1234
8	283	1444	1433	885	899
9	170	950	942	582	576
10	106	633	621	391	343
11	87	436	436	267	217
12	48	312	310	189	189
13	37	219	211	148	139
14	28	212	212	135	127
15	24	175	174	105	98
16	23	158	155	101	89
17	15	103	103	74	71

Na Tabela 5, e também nas demais tabelas, pode-se observar que o resultado do Phobos é bem próximo ao do ConvolutionTR, em termos de quantidade de TR obtidos, perdendo no tempo de execução do

algoritmo. O que se pode notar até aqui é que o *ConvolutionTR* tem um bom resultado perante aos demais algoritmos.

Outro software muito usado e reconhecido na literatura é o Reputer (KURTZ et al., 2001). Este software é composto por três programas: repfind, repvis e repselect.

O programa repfind usa uma implementação eficiente e compacta de árvores (estrutura de dados em árvore) com a finalidade de localizar repetições exatas em um tempo que cresce linearmente com o tamanho da entrada (n) e, portanto, de ordem O(n). A saída é ordenada por e-value, do menor para o maior, e os melhores valores (mais próximos de zero) correspondem aos TR principais e os piores valores são os TR já obtidos anteriormente, isto é, tandem repeats que compõem outros tandem repeats e assim já foram computados. É necessária assim uma filtragem para obter apenas os tandem repeats não repetidos.

O programa repselect permite selecionar repeats de interesse, conforme algum critério do usuário, a partir da saída produzida por repfind. A saída produzida por repselect é uma lista de TR com o comprimento de cada sequência e seu respectivo e-value.

O programa repvis mostra graficamente a saída produzida por repfind. Um código de cores indica a significância e uma barra de rolagem controla a quantidade de dados que são mostrados. Uma função de "zoom" mostra uma visão do genoma todo, assim como pode mostrar também uma representação detalhada de um pedaço do genoma.

Existe um limite para o comprimento da sequência de entrada a ser analisada. Se as opções r (quantidade máxima de repeats reversos), c (quantidade máxima de repeats complementares), e p (quantidade máxima de repeats palíndromos) não são fixados, então o comprimento máximo é 134.217,724. Se as opções r, c, ou p são desejadas, então o comprimento máximo é 67.108,860. Se a sequência de entrada for maior que estes valores, o programa não será executado e emitirá uma resposta de saída ou parada do programa.

Neste trabalho será usado o programa repfind para obter TR e também será comparado com o algoritmo proposto neste trabalho, *ConvolutionTR*, quanto a tempo de execução e número de TR obtido. O comando repfind usado foi o que está descrito a seguir.

repfind -f -l 4 -lw 4000 -allmax -s inputFile.fasta > outputFile.txt

As opções usadas para executar o comando repfind são:

-f = > mostra a máxima quantidade de repeats em sentido direto.

-l 4 = > especifica o mínimo de componentes de um repeat igual a 4.

-lw 4000 = > O resultado de cada TR deverá ter até 4000 caracteres por linha.

-allmax = > mostra todos os TR à medida que estes são encontrados.

-s = > mostra a cadeia que compõe o TR.

Os arquivos usados por repfind, *inputFile.fasta* e *outputFile.txt*, são arquivos de dados de entrada e arquivo de saída de dados respectivamente. Os resultados obtidos têm uma série de redundâncias e foram filtrados por e-value, posição do TR tal que não seja repetida e tamanho da sequência pelo menos maior que 4. As Tabelas 6 e 7 contém os resultados sobre repfind e *ConvolutionTR.*

Tabela 6. Tempo de execução, em segundos, para obter TR.

Espécie	Cromossomo/Locus	Reputer (s)	Convolution TR (s)
Drosophila melanogaster	4	97	2
Arabdopsis lyrata	Locus NW_003302546	10	1
Arabdopsis lyrata	Locus NW_003302545	8	1
Arabdopsis lyrata	Locus NW_003302543	5	1
Arabdopsis lyrata	Locus NW_003302544	3	1
Arabdopsis lyrata	Locus NW_003302542	4	1
Homo sapiens	human_T1 - chromosome 7 - (115977709-117855134)	2	1
Escherichia coli	Locus NC_008253	984	8
Arabdopsis lyrata	Locus NW_003302547	183	2
C. elegans	2	138	14
Arabidopsis thaliana	4	171	29
Arabdopsis lyrata	Locus NW_003302554	175	25

Tabela 7. Número de TR obtidos por Reputer/repfind e ConvolutionTR.

Espécie	Cromossomo/Locus	Reputer (s)	Convolution TR (s)
Drosophila melanogaster	4	29899	35528
Arabdopsis lyrata	Locus NW_003302546	3478	5756
Arabdopsis lyrata	Locus NW_003302545	3329	4970
Arabdopsis lyrata	Locus NW_003302543	2341	3401
Arabdopsis lyrata	Locus NW_003302544	3412	4120
Arabdopsis lyrata	Locus NW_003302542	2134	3021
Homo sapiens	human_T1 - chromosome 7 - (115977709-117855134)	4031	4753
Escherichia coli	Locus NC_008253	54891	68128
Arabdopsis lyrata	Locus NW_003302547	39871	43653
C. elegans	2	376927	518065
Arabidopsis thaliana	4	297613	421015
Arabdopsis lyrata	Locus NW_003302554	318975	414123

A partir das Tabelas 6 e 7, pode-se observar que o enfoque a respeito do *ConvolutionTR* tem valores melhores quanto a tempo de execução e também quanto ao número de TR obtidos. É importante mencionar que os valores de TR obtidos pelo algoritmo proposto neste trabalho teve uma filtragem para evitar valores repetidos e assim haver uma quantidade maior de TR obtidos.

O Reputer possui mais opções de execução do que o ConvolutionTR. O Reputer já tem um amadurecimento e assim, com o passar do tempo, as opções foram sendo adicionadas. Por exemplo, o ConvolutionTR não considera ainda TR em sentido reverso, o qual é opção do Reputer.

Outras versões do ConvolutionTR deverão conter mais opções no sentido de se equiparar aos softwares já existentes, os quais já têm alguns anos de amadurecimento e assim possuem um conjunto maior de opções de busca. Por enquanto, o que se pretende neste trabalho é expor um novo enfoque para busca de TR que seja confiável, rápido e que contenha o maior número possível de TR não repetidos. De acordo com os testes feitos, esta proposta apresentada é muito promissora.

Conclusões

Com base nos resultados apresentados, fica claro que o ConvolutionTR é um algoritmo rápido e eficiente quanto a identificação de TR. Sua flexibilidade permite a identificação desde microssatélites até minissatélites sem limites teóricos ao número de repetições ou ao tamanho do período. Além disso, a possibilidade de lidar com mutações (mismatches e Indels) também torna o ConvolutionTR mais atrativo em estudos de sequências repetitivas.

Com o ConvolutionTR, o trabalho de identificação de TR se tornou factível em escala genômica num tempo razoável. Por outro lado, esta proposta ainda pode ser melhorada e ter mais opções de busca de TR com o passar dos anos e também ter uma opção de busca pela web. As propostas apresentadas na literatura já têm amadurecimento e assim possuem mais opções de busca do que o ConvolutionTR. Porém, a metodologia apresentada é promissora, conforme mostrado nos resultados.

Este trabalho não tem como objetivo mostrar que o ConvolutionTR é o melhor algoritmo, mas mostrar o potencial desta metodologia e contribuir para outros desenvolvedores terem mais uma opção de busca de TR, não importando a frequência e o período do TR, bem como o tamanho da sequencia considerada (seja de alguns kBytes a vários GBytes).

Referências

ALTSCHUL, S. F.; GISH, W.; MILLER, W.; MYERS, E. W.; LIPMAN, D. J. Basic local alignment search tool. **Journal of Molecular Biology**, London, v. 215, n. 3, p. 403-410, Oct. 1990.

BANNERT, N.; KURTH, R. Retroelements and the human genome: new perspectives on an old relation. **Proceedings of the National Academy of Sciences of the United States of America**, Washington, v. 101, n. 2, p. 14572-14579, Oct. 2004.

BOTSTEIN, D.; WHITE, R. L.; SKOLNICK, M.; DAVIS, R. W. Construction of genetic-linkage map in man using restriction fragment length. **American Journal of Human Genetics**, Chicago, v. 32, n. 3, p. 314-331, May 1980.

BRODZIK, A. K. Quaternionic periodicity transform: an algebraic solution to the tandem repeats detection problem. **Bioinformatics**, Oxford, v. 23, n. 6, p. 694-700, Feb. 2007.

CONVOLUÇÃO. Disponível em: <http://pt.wikipedia.org/wiki/Convolução#cite_note-2>. Acesso em: 10 abr. 2013.

IVICS, Z.; IZSVAK, Z. Transposons for gene therapy!. **Current Gene Therapy**, Amsterdam, v. 6, n. 5, p. 593-607, Oct. 2006.

JIANG, T.; KEARNEY, P.; LI, M. Some open problems in computational molecular biology. **Journal of Algorithms**, San Diego, v. 34, n. 1, p. 194-201, Jan. 2000.

KAZAZIAN, H. H. Mobile elements: drivers of genome evolution. **Science**, Washington, v. 303, n. 5664, p. 1626-1632, Mar. 2004.

KURTZ, S.; CHOUDHURI, J. V.; OHLEBUSCH, E.; SCHLEIERMACHER, C.; STOYE, J.; GIEGERICH, R. REPuter: the manifold applications of repeat analysis on genomic scale. **Nucleic Acids Research**, Oxford, v. 29, n. 22, p. 4633-4642, Nov. 2001.

LANDER, E. S.; BOTSTEIN, D. Mapping mendelian factors underlying quantitative traits using RFLP linkage maps. **Genetics**, Austin, v. 121, n. 1, p. 185-199, Jan. 1989.

MABUCHI, K.; SONG, H. Y.; TAKESHIMA, H.; NISHIDA, M. A set of SNPs near or within STR regions useful for discriminating native Lake Biwa and introduced "Eurasian" strains of common carp. **Conservation Genetics Resources**, v. 4, n. 3, p. 649-652, Sept. 2012.

MATROUD, A. A.; HENDY, M. D.; TUFFLEY, C. P. NTRFinder: a software tool to find nested tandem repeats. **Nucleic Acids Research**, Oxford, v. 40, n. 3, p. e17, Feb. 2012.

NARCISO, M. G.; YAMAGISHI, M. E. B. Uma nova proposta de busca por tandem repeats. In: CONGRESSO NACIONAL DE CIÊNCIAS BIOLÓGICAS, 1.; SIMPÓSIO DE CIÊNCIAS BIOLÓGICAS, 4., 2011, Recife. **Biodiversidade e floresta**: desafios e perspectivas: anais. Recife: Universidade Católica de Pernambuco, 2011. p. 760-767.

PRICE, A. L.; JONES, N. C.; PEVZNER, P. A. De novo identification of repeat families in large genomes. **Bioinformatics,** Oxford, v. 21, Suppl. 1, p. i351-i358, June 2005.

ROCHETA, M.; DIONISIO, F. M.; FONSECA, L.; PIRES, A. M. Paternity analysis in excel. **Computer Methods and Programs in Biomedicine**, Amsterdam, v. 88, n. 3, p. 234-238, Dec. 2007.

SANMIGUEL, P.; TIKHONOV, A.; JIN, Y. K.; MOTCHOULSKAIA, N.; ZAKHAROV, D.; MELAKEBERHAN, A.; SPRINGER, P. S.; EDWARDS, K. J.; LEE, M.; AVRAMOVA Z.; BENNETZEN, J. L. Nested retrotransposons in the intergenic regions of the maize genome. **Science**, Washington, v. 274, n. 5288, p. 765-768, Nov. 1996.

SHARMA, P. C.; GROVER, A.; KAHL, G. Mining microsatellites in eukaryotic genomes. **Trends in Biotecnology**, Amsterdam, v. 25, n. 11, p. 490-498, Nov. 2007.

WANG, L.; JIANG, T. On the complexity of multiple sequence alignment. **Journal of Computational Biology**, New York, v. 1, n. 4, p. 337-348, 1994.

WATERMAN, M. S.; SMITH, T. F. The identification of common molecular subsequences. **Journal of Molecular Biology**, London, v. 147, n. 1, p. 195-197, Mar. 1981.

WILBUR, W. J.; LIPMAN, D. J. Rapid similarity searches of nucleic acid and protein data banks. **Proceedings of the National Academy of Sciences of the United States of America**, Washington, v. 80, n. 3, p. 726-730, Feb. 1983.

ZHI, D.; RAPHAEL, B. J.; PRICE, A. L.; TANG, H.; PEVZNER, P. A. Identifying repeat domains in large genomes. **Genome Biology**, London, v. 7, n. 1, p. R7, Jan. 2006.

Comunicado Técnico, 217

Exemplares desta edição podem ser adquiridos na:
Embrapa Arroz e Feijão
Endereço: Rod. GO 462 Km 12 Zona Rural, Caixa Postal 179 75375-000 Santo Antônio de Goiás, GO
Fone: (62) 3533 2123
Fax: (62) 3533 2100
www.embrapa.br/fale-conosco/sac
1ª edição
Versão online (2014)

Comitê de publicações

Presidente: *Pedro Marques da Silveira*
Secretário-Executivo: *Luiz Roberto R. da Silva*
Membros: *Camilla Souza de Oliveira, Luciene Fróes Camarano de Oliveira, Flávia Rabelo Barbosa Moreira, Ana Lúcia Delalibera de Faria, Heloisa Célis Breseghello, Márcia Gonzaga de Castro Oliveira, Fábio Fernandes Nolêto*

Expediente

Supervisão editorial: *Luiz Roberto R. da Silva*
Revisão de texto: *Camilla Souza de Oliveira*
Normalização bibliográfica: *Ana Lúcia D. de Faria*
Editoração eletrônica: *Fabiano Severino*

CGPE: 11241