Estágio Obrigatório

Relatório mensal referente ao mês de setembro
Aluno: Josué Crispim Vitorino
Professora Orientadora: Maria Angélica de Oliveira Camargo Brunetto

Sumário

1 Introdução

Em cada célula dos organismos vivos, para que ocorra a transcrição de um gene é necessário a ação conjunta dos fatores de transcrição e dos elementos regulatórios. Os fatores de transcrição são proteínas que se ligam nos elementos regulatórios. Já os elementos regulatórios são pequenos segmentos de DNA localizados em uma região antes do gene que será transcrito. Essa região é chamada de região promotora (ou reguladora).

É de grande importância a identificação dos elementos regulatórios. Uma vez que eles estão ligados com a expressão de um gene. O conhecimento agregado com a identificação desses elementos pode levar a melhoramentos genéticos de organismos importantes para o consumo e a economia mundial, como por exemplo de grãos como o arroz, soja e milho.

No empenho de encontrar elementos regulatórios diversos algoritmos foram propostos. [Das e Dai 2007] os classificaram em três grupos:

- Os baseados em sequências promotoras de genes que são regulados pelos mesmos fatores de transcrição (genes co-regulados); estes métodos se concentram em apenas um único genoma.

  Este ainda é subdividido em : predição probabilística e predição baseada em palavras.

- Os que utilizam sequências promotoras de genes ortólogos, que são sequências de DNA similares a várias espécies, indicando que estas espécies derivaram de um ancestral comum, também chamados de métodos de rastros filogenéticos.

- Os métodos que combinam rastros filogenéticos e sequências promotoras de genes co-regulados.

Também existem métodos que focam na busca de agrupamentos de elementos regulatórios na região promotora, conhecidos como *CRM* (do inglês *Cis Regulatory Modules*). Os CRMs integram a conexão de vários TFs resultando em um controle combinatório, e em um padrão específico da expressão de um gene [Priest, Filichkin e Mockler 2009].

Assim como a identificação dos elementos regulatórios é de grande importância, também é a identificação de fatores de transcrição e a associação da funcionalidade de cada fator de transcrição na expressão e respostas da célula. Estudos como de [Holloway, Kon e DeLisi 2008], [Lan et al. 2007],[Zhang et al. 2005] e [Bigelow et al. 2004], focaram na identificação de genes que são expressados quando há a ação conjunta de determinados fatores de transcrição e elementos regulatórios, ação que só ocorre em casos específicos, como por exemplo em [Wang et al. 2009] onde a ação do fator de transcrição *dehydration responsive element binding proteins* (DREB) e do elemento regulatório identificado como *dehydration responsive element* (DRE) que é formado pela sequencia consenso de bases nitrogenadas (A/GCCGAC), atuam na resposta a estresses abióticos nas plantas, como: alta salinidade, baixa temperaturas e seca. Segundo [Zhang et al. 2005] para que o DRE seja funcional ele tem que estar acompanhado por outros elementos regulatórios, mas a especificidade desses elementos regulatórios é baixa, fazendo que muitos desses varie de gene para gene. O objetivo deste trabalho é a implementação e a utilização de alguns dos métodos para identificação de elementos regulatórios, para identificar o conjunto de elementos que atuam em conjunto com o DRE formando um CRM, e então com essa informação fazer a classificação de genes alvos do DREB, utilizando o método de aprendizado de máquina de vetor de suporte.

No Capitulo 1.1 é descrito como foi obtida as sequências para teste, assim como o pré-processamento das sequência para adequar a entrada nos diversos algoritmos de busca de elementos regulatórios. No Capitulo 2 são apresentados os algoritmos desenvolvidos, que foram utilizados na predição de elementos regulatórios. No capitulo 3 é mostrado detalhes da implementação da máquina de vetor de suporte. Finalizando é descritas as conclusões obtidas após a realização deste estágio.

1.1 Coleta dos dados e pre-processamento

As sequências da *Arabidopsis* utilizadas neste trabalho foram sequências de genes conhecidos por serem expressos mediante estresses abióticos e conter o elemento regulatório DRE, estas sequências foram as mesmas utilizadas por [Wang et al. 2009]. Elas foram baixadas no banco de dados público [Thomas-Chollier et al. 2008].

2 Ferramenta análise de elementos regulatórios

Dentro do conjunto de métodos para busca de elementos regulatórios, foram selecionados os métodos que obtém melhores resultados na identificação de elementos regulatórios e que focam as buscas em genes co-regulados, uma vez que, em busca em genes ortólogos, para fazer a análise computacional é necessário as sequências das espécies relacionadas em uma mesma árvore filogenética, entretanto como muitos organismos ainda não tiveram o genoma decodificado, a árvore filogenética ficaria incompleta, prejudicando a análise. A seguir é detalhado teoricamente cada algoritmo implementado, então é explicado a implementação da ferramenta de analise de elementos regulatórios que está sendo implementada.

2.1 Median String

[Jones e Pevzner 2004] descreve o *Median String Problem* apresentando ele como um algoritmo simples mas eficiente em suas buscas, entretanto a quantidade de elementos regulatórios que ele encontra é muito pequena, identificando apenas elementos regulatórios muito conservados. Este algoritmo foi estudado e implementado para ter um entendimento básico de algoritmos utilizados para busca de elementos regulatórios.

Para fazer as buscas de elementos regulatórios, o algoritmo procura por uma palavra média em todas as sequências de DNA, com o menor número de mutações,os números de mutações m e o tamanho n da palavra escolhido pelo usuário. Então é calculado a distância de Hamming em todas as sequências, o menor valor gerado para cada sequência é somado, conforme a formula :

$$DistnciaTotal = \sum_{i=1}^{L} min(hamming(s)) \tag{2.1}$$

L é a quantidade de sequências.

As palavras que obtiverem a menor soma da distancia serão consideradas candidatas

a elementos regulatórios. A implementação foi feita em C++ na figura 2.1 podemos visualizar os diagramas de classes do sistema e os detalhes de cada classe podem ser visto na seção 2.4.

2.2 Oligo-Analysis

O Oligo-Analysis é um método desenvolvido por [Helden, André e Collado-Vides 1998], foi projetado para detectar oligonucleotídio (pequenos seguimentos de DNA) sobre-representados, que caracterizam elementos regulatórios, na região promotora dentro de um grupo de genes co-regulados, comparando a frequência observada das sequências e as ocorrências esperada com a distribuição binomial. Dentro da classificação feita por [Das e Dai 2007], ele é método focado na predição de genes co-regulados baseado em palavras (ou sub-sequências).

O método primeiramente conta a ocorrência occ de todos possíveis oligonucleotídio b, de tamanho de um até no máximo nove. Este tamanho reflete na ordem da cadeia de Markov utilizada para calcular as frequências esperadas (descrito a seguir), por exemplo, se o tamanho de b for 4 a ordem da cadeia de Markov será 3, este valor é encontrado com $k = m + 1$, onde k é o tamanho de b e m é a ordem da cadeia de Markov. Então é calculada a frequência de cada oligonucleotídio b com a formula :

$$\frac{b}{W} \tag{2.2}$$

Onde W é a soma do tamanho todas as sequências não codificantes do genoma do organismo.

O próximo passo é construir uma modelagem (*background*) para as sequências utilizando cadeias de Markov [Ewens e Grant 2005]. A modelagem consiste, primeiramente, em construir uma matriz de transição a partir das frequências dos oligonucleotídios calculadas utilizando:

$$P(r_i|S_{1...m}) = \frac{F_{bg(r_i|S_{1...m})}}{\sum_{j \in A} F_{bg}(r_j|S_{1...m})} = \frac{F_{bg(S_{1...m}r_i)}}{\sum_{j \in A} F_{bg}(S_{1...m}r_j)} \tag{2.3}$$

Onde F_{bg} é a frequência calculada, r_i é um nucleotídeo na posição i e $S_{1...m}$ é/são o(s) nucleotídeo(s) antecessor(es) a r_i.

A próxima etapa é computar a probabilidade (ou frequência esperada) $F_e\{b\}$ de um oligonucleotídio ser encontrado na região promotora do organismo, a partir do *background* B encontrado na etapa anterior, com a seguinte formula:

$$P(b|B) = P(b_{1,m|B}) \prod_{i=m+1}^{t} P(r_i|b_{i-m}, i-1, B) \tag{2.4}$$

t é o tamanho de S.

Depois de encontrado a frequência esperada é calculado a ocorrência esperada de um oligonucleotídio com :

$$E(occ\{b\}) = F_e\{b\} \times 2 \times \sum_{i=1}^{S} (L_i - w + 1) \tag{2.5}$$

w é a tamanho do oligonucleotídio, S é o numero de sequências no conjunto, L é o tamanho das sequências. O fator 2 é devido a soma de ocorrências que é considerada em ambos os filamentos de DNA.

Finalmente é calculada a distribuição binomial, que calcula a probabilidade de encontrar exatamente n ocorrências do oligonucleotídio b, através da fórmula :

$$P(ooc\{b\} = n) = \frac{T!}{(T-n)! \times n!} \times (F_e\{b\})^n \times (1 - F_e\{b\})^{(T-n)} \tag{2.6}$$

Para que fosse possível a computação a formula foi simplificada para a forma recursiva ficando:

$$P(ooc\{b\} = 0) = (1 - F_e\{b\})^{(T)} \tag{2.7}$$

$$P(ooc\{b\} = n) = \frac{(n+1)(1 - F_e\{b\})}{(T-n)F_e\{b\}} (F_e\{b\}(n+1) \tag{2.8}$$

Também é calculado para n ou mais ocorrências do nucleotídeo b.

$$P(ooc\{b\} \geq n) = \sum_{j=n}^{T} P(ooc\{b\} = j) = 1 - \sum_{j=0}^{n-1} P(ooc\{b\} = j) \tag{2.9}$$

Após calculado a distribuição binomial, então é calculado o limiar para cortar os resultados com abaixo do valor estabelecido. Isto é obtido com as formulas:

$$sig = -log_{10}(P(ooc\{b\} \geq n) \times 4^w - (4^w - 4^{w/2})/2) \tag{2.10}$$

Os oligonucleotídios com maior sig são considerados sobre-representados no conjunto de dados, e consequentemente com uma grande probabilidade de ser um elemento regulató-

rio. Por último, são determinadas nas sequências promotoras as posições que batem com os oligonucleotídios com maior sig encontrados, assim é retornado ao usuário quais foram os elementos regulatórios encontrados e os valores calculados.

Todos as etapas anteriores foram implementadas na linguagem C++, os detalhes da implementação podem ser visto na seção 2.4.

2.3 HexDiff

Neste trabalho [Chan e Kibler 2005] desenvolveu o algoritmo HexDiff. Este algoritmo busca agrupações de elementos regulatórios (CRM), que atuam juntos na regulação de um gene.

O HexDiff é um tipo de algoritmo de aprendizado de máquina, e foi projetado para discriminar dois tipos de sequências de DNA: CRM, e no-CRM(não agrupamento de elemento regulatório). Para fazer a classificação é necessário um conjunto de dados de treinamentos, que é obtido através de conhecidos CRMs, que são colocados no conjunto positivo de treinamento, os não conhecidos, os no-CRMs, são inseridos no conjunto negativo. Após a seleção dos dados é calculada a frequência de cada hexamer (subsequência de nucleotídeos de 6 pb), no conjunto negativo $f_p(h)$ e positivo $f_n(h)$, então é calculo a razão $R(h)$:

$$R(h) = \frac{F_p(h)}{F_n(h)} \tag{2.11}$$

Os hexamers que obtiverem um alto valor de $R(h)$, são colocados no conjunto H_d. Com isto H_d vai ter hexamers que são mais comuns em CRMs do que em no-CRMs.

Depois de gerado o conjunto H_d, ele é usado para classificar cada posição em uma sequência não conhecida como uma sequência CRM e no-CRM. Para fazer a classificação é construído uma janela na sequência, entre 1000-2000 pb, que a cada rodada é movida 1 pb na sequência, e é calculado a pontuação S_i para cada posição i da janela na sequência, pelo produto da razão $R(h)$ e o numero de aparições de um hexamer $n(h_d)$ que está no H_d na janela, qualquer posição que exceder o limite é considerada um CRM.

Este Algoritmo já está desenvolvido, seguindo a descrição feita, ele está agora em uma etapa de calibração com as sequências utilizadas por [Chan e Kibler 2005]. Todo o esquema de desenvolvimento pode ser visto na seção 2.4.

2.4 Documentação básica

Todos os algoritmos implementados estão agrupados em uma única ferramenta para facilitar os usuários a fazerem várias busca em diferentes métodos, para o mesmo conjunto de dados. Atualmente a implementação conta com dez classes (figura 2.1), todas implementadas na linguagem C++.

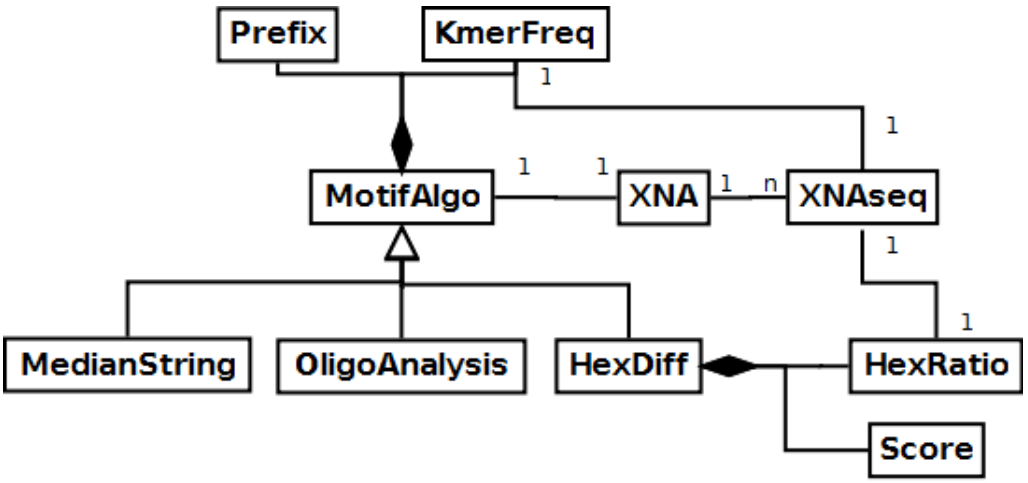

Figura 2.1: Diagrama de classes

As classes responsáveis pela manipulação das sequências são as classes **DNA** e **DNAseq** (figura 2.2). A classe **DNA** o principal objetivo é manter um conjunto de sequências de DNA, este conjunto é armazenado em um vetor de objetos do tipo **DNAseq**, as outras funcionalidades dela são: ler um arquivo de sequências do tipo *fasta* (figura 2.3); escrever um arquivo do tipo *fasta* e adicionar uma nova sequência ao conjunto. A classe **DNAseq**, foi criada para a manipulação de uma sequência, nela é possível: inserir a sequência, inserir as descrições da sequência e o identificador da mesma. Também é possível gerar o complemento da sequência, o reverso complemento e extrair o tamanho da sequência.

Seguindo temos a classe **MotifAlgo** (figura 2.4), está classe implementa funções que são partilhadas entre as classes que implementam os algoritmos de busca. Nela é possível através do método **computeTotalFrequency(DNA,int)**, calcular todas as ocorrências e frequências de todos os possíveis oligonucleotídio em uma sequência, o tamanho dos oligonucleotídios é definido no segundo parâmetro do método, a computação das ocorrências dos oligonucleotídios é feita com o auxilio do software Meryl [Walenz e Florea 2011], ele foi utilizado por ser otimizado para este tipo de tarefa, e também mostrou-se mais rápido do que a implementação feita para este propósito. Com o método **computeMotifFrequency(DNA,DNAseq)**, assim como o método anterior calcula a frequência de oligonucleotídios, a diferença é que ele não

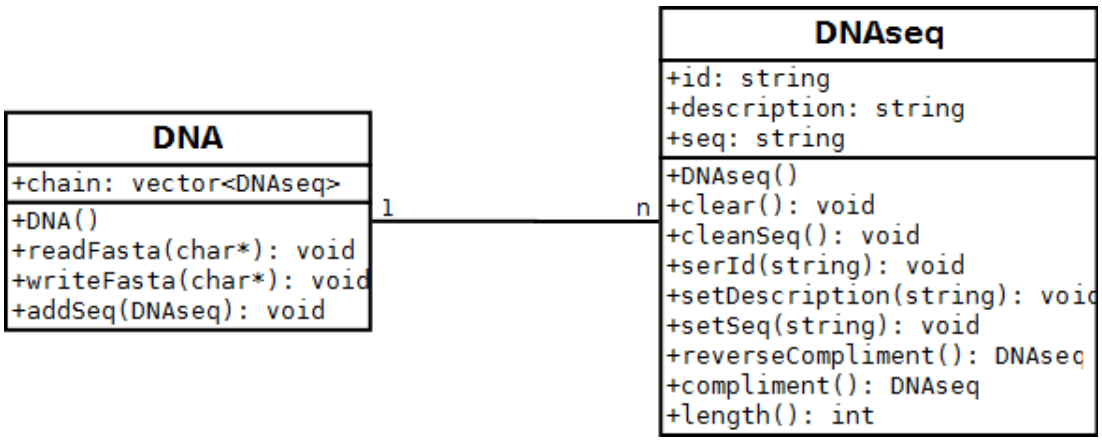

Figura 2.2: Classes DNA e DNAseq

```
>HUMAN
CAGGTTATCAGCAACAACACAGTCATATCCATTCTCAATTAGCTCTACCACAGTGTGTGAACCAATGTATCCAGCACCAC
CTGTAACCAAAACAATTTTAGAAGTACTTTCACTTTGTAACTGAGCTGTCATTTATATTGAATTTTCAAAAATTCTTACTTT
TTTTTTGGATGGACGCAAAGAAGTTTAATAATCATATTACATGGCATTACCACCATATACATATCCATATCTAATCTTACTT
ATATGTTGTGGAAATGTAAAGAGCCCCATTATCTTAGCCTAAAAAAACCTTCTCTTTGGAACTTTCAGTAATACGCTTAAC
GCTCATTGCTATATTGAAGTACGGATTAGAAGCCGCCGAGCGGGCGACAGCCCTCCGACGGAAGACTCTCCTCCGTGC
TCCTCGTCTTCACCGGTCGCGTTCCTGAAACGCAGATGTGCCTCGCGCCGCACTGCTCCGAACAATAAAGATTCTACAAT
CTAGCTTTTATGGTTATGAAGAGGAAAAATTGGCAGTAACCTGGCCCCACAAACCTTCAAATTAACGAATCAAATTAACA
CCATAGGATGATAATGCGATTAGTTTTTTAGCCTTATTTCTGGGGTAATTAATCAGCGAAGCGATGATTTTTGATCTATTA
>RAT
CGGTTTAGCATCATAAGCGCTTATAAATTTCTTAATTATGCTCGGGCACTTTTCGGCCAATGGTCTTGGTAATTCCTTTGCGC
TAGAATTGAACTCAGGTACAATCACTTCTTCTGAATGAGATTTAGTCATTATAGTTTTTTCTCCTTGACGTTAAAGTATAGAG
TATATTAACAATTTTTTGTTGATACTTTTATGACATTTGAATAAGAAGTAATACAAACTGAAAATGTTGAAAGTATTAGTTAAA
>MOUSE
CATTAATTTTGCTTCCAAGACGACAGTAATATGTCTCCTACAATACCAGTTTCGCTGCAGAAGGCACATCTATTACATTTACTG
AGCATAACGGGCTGTACTAATCCAAGGAGGTTTACGGACCAGGGGAACTTTCCAGATTCAGATCACAGCAATATAGGACTAG
```

Figura 2.3: Exemplo de arquivo *fasta*

calcula para todos os possíveis oligonucleotídios, mas sim para um oligonucleotídio definido pelo usuário. O método **buildTransitionalTable()**, ele cria uma matriz de transição que é utilizada no modelo de Markov, para criar a matriz este método utiliza as frequências dos oligonucleotídios calculadas, para utiliza-lo é necessário antes rodar o método **computeTotalFrequency(DNA,int)**, que armazena as frequências na variável **frequencies** esta variável com todas as frequências é usada então pelo **buildTransitionalTable()**. Por último temos o método **scoreSeqMarkov(DNAseq,int)**, ele calcula a probabilidade de um oligonucleotídio aparecer em uma sequência, segundo o *background* calculado com **buildTransitionalTable()**.

As classes **KmerFreq** e **Prefix** (figura 2.5), atuam como auxiliares da classe **MotifAlgo**. Um objeto da **KmerFreq** armazena um oligonucleotídio, as ocorrências deste oligonucleotídio nas sequências e a sua frequência. Um objeto da classe **Prefix**, é utilizado para armazenar um prefixo de um oligonucleotídio as quatro possibilidades de sufixo (A,C,G,T) e a soma das frequências dos oligonucleotídios formados com a concatenação do prefixos com os

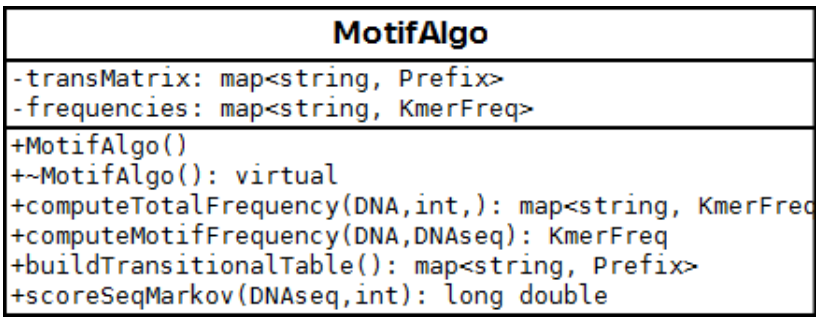

Figura 2.4: Classe MotifAlgo

sufixos, está classe é utilizada na construção da matriz de transição.

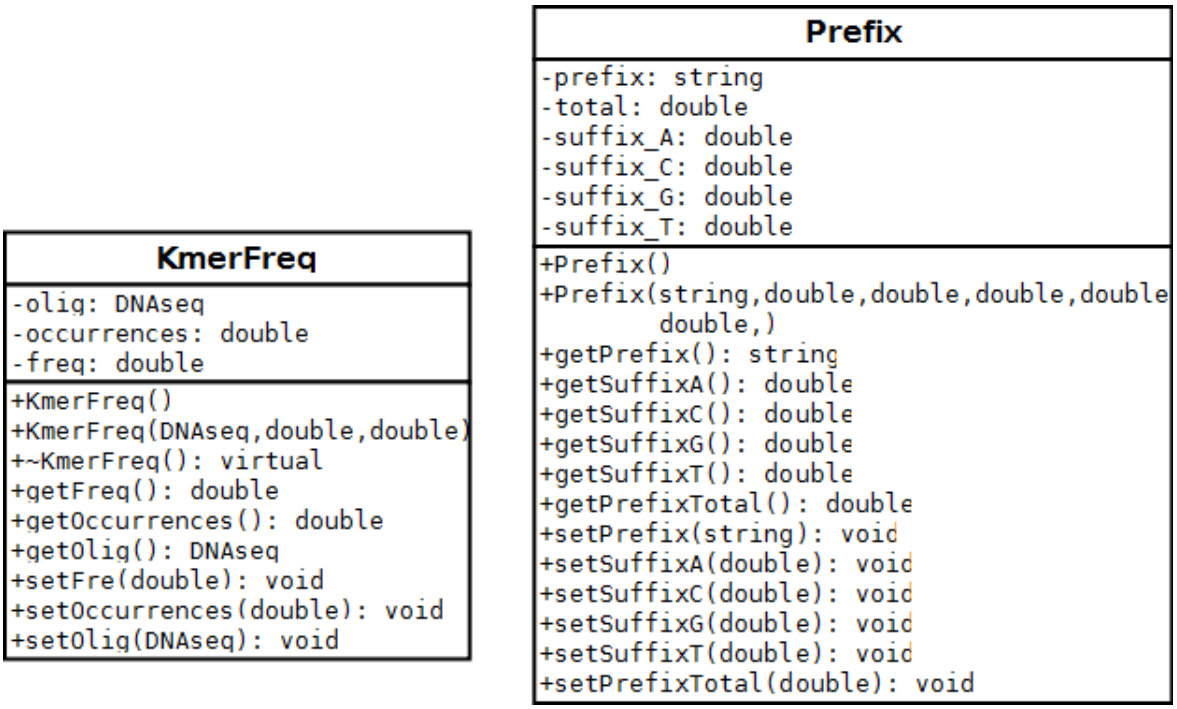

Figura 2.5: Classes KmerFreq e Prefix

A implementação do algoritmo Median String é feita pela classe **MedianString** (figura 2.6). O principal método é o **computeMedianString(DNAseq,int,int)** ele cria a cada rodada um prefixo que começa com apenas uma base e vai incrementando até o tamanho do oligonucleotídio escolhido pelo usuário. Cada prefixo é comparado com todas as sequências procurando a menor distância, isto é feito pelo método **totalDistance(DNAseq,DNA,int)**, este método percorre todas as sequências e extrai fragmentos da sequência do mesmo tamanho do prefixo, então ele faz uma chamada ao método **hammingDistance(string,string)**, este retorna a distância entre o prefixo e o fragmento da sequência, as menores distâncias entre as sequências é somada e retornada. O valor da distância do prefixo é comparado com o melhor valor já encontrado (este valor é iniciado com número grande), se a distancia do

prefixo for maior, é feita uma chamada ao método **bypass(DNAseq,int)** que troca por outra a primeira base do prefixo, . Se o valor for menor, é modificado a última base com o método **nextVertex(DNAseq,int,int)**. Após ser computada todas as possibilidades para o prefixo, a menor distância encontrada substitui o melhor valor já encontrado, então é incrementado mais uma base ao prefixo, que retorna aos passos anteriores, isto é feito até atingir o tamanho do oligonucleotídio. Ao final o é retornado os oligonucleotídios com as menores distâncias.

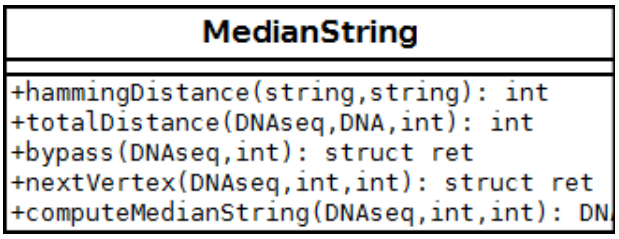

Figura 2.6: Classe MedianString

A classe **OligoAnalysis** (figura 2.7) implementa o algoritmo Oligo-Analysis, ela herda os métodos da classe **MotifAlgo**. O método **performAnalysis(DNA,double,int)** calcula o complemento reverso de todas as sequências e adiciona no conjunto de sequências, então ele faz uma chamada ao método **computeTotalFrequency(DNA,int)** para calcular a frequência e aos métodos **buildTransitionalTable()** e **scoreSeqMarkov(DNAseq,int)**, o retorno do último método são as sequências esperadas, com elas é calculado a distribuição binomial **binomialPValue(double,double,double)** as possibilidade de encontrar ocorrer n ou mais oligonucleotídio na sequência **binomialOccGreater(double,double,double)**, e com o resultado do último método calcular o **sigValue(double,int)** (valor sig).

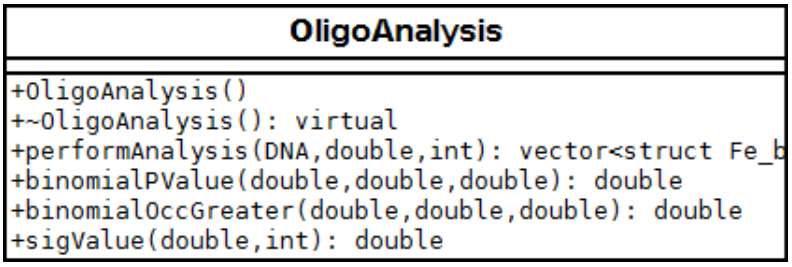

Figura 2.7: Classe OligoAnalysis

O algoritmo HexDiff é implementado pela classe **HexDiff** (figura 2.8), ela também herda as funcionalidades da classe **MotifAlgo**. No método **performFrequency()** é caculado a frequência de todos os possíveis oligonucleotídios de tamanho seis nos dos dados de entrada positivo e negativo, também é calculada a razão, então os oligonucleotídios e a razão calculada para eles são armazenados em um objeto da classe **HexRatio**, que é uma classe auxiliar a **HexDiff** utilizada apenas para este fim. Todos os objetos da classe **HexRatio** são alocados

em um vetor Hd, os n oligonucleotídios com maior razão são mantidos no Hd o restante é apagado, isto é feito com o método **chooseHexamers()**. Por último, com **computeScore()** é computado a pontuação de todas as possíveis janelas de tamanho de tamanho definido pelo usuário de 1000-2000 pares de bases, as janelas que excederem um limite determinado pelo usuário são consideradas possíveis CRMs, e são alocadas em um vetor de objetos da classe **Score**, esta classe também é uma auxiliar onde é armazenado a posição da janela encontrada e a pontuação obtida.

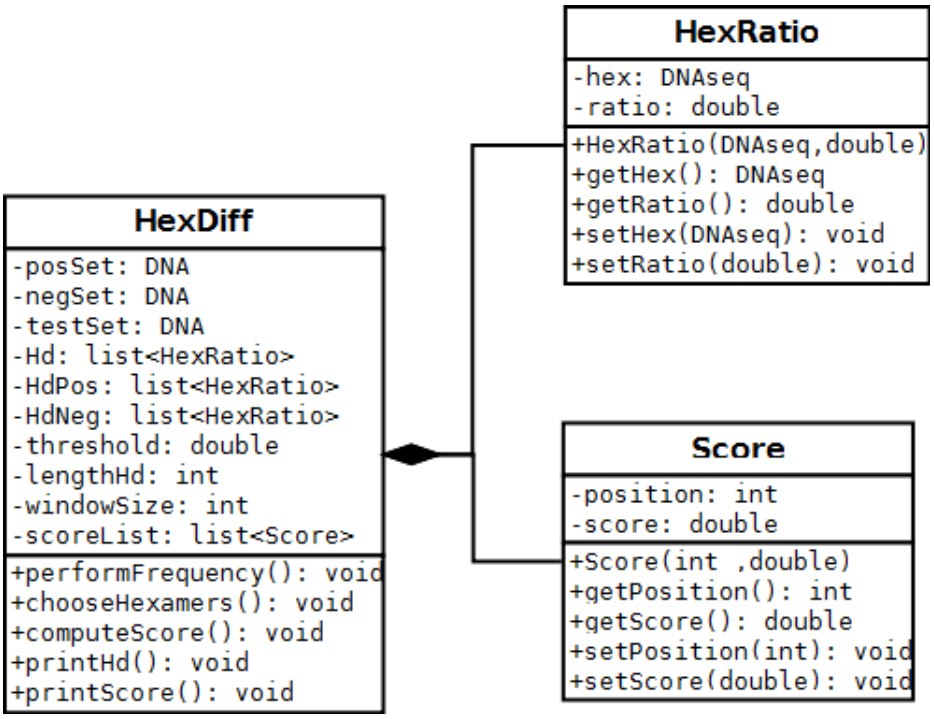

Figura 2.8: Classes HexDiff, HexRatio e Score

Referências Bibliográficas

[Bigelow et al. 2004]BIGELOW, H. et al. CisOrtho: A program pipeline for genome-wide identification of transcription factor target genes using phylogenetic footprinting. *BMC Bioinformatics*, v. 5, n. 1, p. 27+, 2004. ISSN 1471-2105. Disponível em: <http://dx.doi.org/10.1186/1471-2105-5-27>.

[Chan e Kibler 2005]CHAN, B.; KIBLER, D. Using hexamers to predict cis-regulatory motifs in Drosophila. *BMC Bioinformatics*, v. 6, n. 1, p. 262+, out. 2005. ISSN 1471-2105. Disponível em: <http://dx.doi.org/10.1186/1471-2105-6-262>.

[Das e Dai 2007]DAS, M.; DAI, H. K. A survey of DNA motif finding algorithms. *BMC Bioinformatics*, v. 8, n. Suppl 7, p. S21+, 2007. ISSN 1471-2105. Disponível em: <http://dx.doi.org/10.1186/1471-2105-8-S7-S21>.

[Ewens e Grant 2005]EWENS, W. J.; GRANT, G. R. *Statis[t]ical methods in bioinformatics : an introduction*. 2nd. ed. Springer, 2005. Hardcover. ISBN 0387400826. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0387400826>.

[Helden, André e Collado-Vides 1998]HELDEN, J. van; ANDRé, B.; COLLADO-VIDES, J. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies1. *Journal of Molecular Biology*, Centro de Investigación sobre Fijación de Nitrógeno, Universidad Nacional Autónoma de México, AP565A Cuernavaca, Morelos, 62100, México. jvanheld@ebi.ac.uk, v. 281, n. 5, p. 827–842, set. 1998. ISSN 00222836. Disponível em: <http://dx.doi.org/10.1006/jmbi.1998.1947>.

[Holloway, Kon e DeLisi 2008]HOLLOWAY, D.; KON, M.; DELISI, C. Classifying transcription factor targets and discovering relevant biological features. *Biology Direct*, v. 3, n. 1, p. 22+, maio 2008. ISSN 1745-6150. Disponível em: <http://dx.doi.org/10.1186/1745-6150-3-22>.

[Jones e Pevzner 2004]JONES, N. C.; PEVZNER, P. *An introduction to bioinformatics algorithms*. MIT Press, 2004. ISBN 9780262101066. Disponível em: <http://www.worldcat.org/isbn/9780262101066>.

[Lan et al. 2007]LAN, H. et al. Combining classifiers to predict gene function in Arabidopsis thaliana using large-scale gene expression measurements. *BMC Bioinformatics*, v. 8, n. 1, p. 358+, set. 2007. ISSN 1471-2105. Disponível em: <http://dx.doi.org/10.1186/1471-2105-8-358>.

[Priest, Filichkin e Mockler 2009]PRIEST, H. D.; FILICHKIN, S. A.; MOCKLER, T. C. cis-Regulatory elements in plant cell signaling. *Current Opinion in Plant Biology*, v. 12, n. 5, p. 643–649, out. 2009. ISSN 13695266. Disponível em: <http://dx.doi.org/10.1016/j.pbi.2009.07.016>.

[Thomas-Chollier et al. 2008]THOMAS-CHOLLIER, M. et al. RSAT: regulatory sequence analysis tools. *Nucleic Acids Research*, v. 36, n. suppl 2, p. W119–W127, jul. 2008. ISSN 1362-4962. Disponível em: <http://dx.doi.org/10.1093/nar/gkn304>.

[Walenz e Florea 2011]WALENZ, B.; FLOREA, L. *Meryl*. jul. 2011. Disponível em: <http://sourceforge.net/projects/kmer/>.

[Wang et al. 2009]WANG, S. et al. An <i>in silico</i> strategy identified the target gene candidates regulated by dehydration responsive element binding proteins (DREBs) in <i>Arabidopsis</i> genome. *Plant Molecular Biology*, Springer Netherlands, v. 69, n. 1, p. 167–178, jan. 2009. ISSN 0167-4412. Disponível em: <http://dx.doi.org/10.1007/s11103-008-9414-5>.

[Zhang et al. 2005]ZHANG, W. et al. Cis-regulatory element based targeted gene finding: genome-wide identification of abscisic acid- and abiotic stress-responsive genes in Arabidopsis thaliana. *Bioinformatics*, Oxford University Press, v. 21, n. 14, p. 3074–3081, jul. 2005. ISSN 1367-4803. Disponível em: <http://dx.doi.org/10.1093/bioinformatics/bti490>.