Análise de Classificadores de Seqüências Projetados por Aprendizado Computacional Supervisionado e não Supervisionado

Caetano Jimenez Carezzato

Proposta de Dissertação – Mestrado

Ciência da Computação

Orientador: **Professor Doutor Junior Barrera**

Departamento de Ciência da Computação

Instituto de Matemática e Estatística

Universidade de São Paulo

Agradecimentos

Gostaria de agradecer a todos que conviveram comigo durante todo este tempo, auxiliando ou atrapalhando vocês são incríveis.

Gostaria também de agradecer especialmente ao Doutor **Sandro José de Souza** do Institudo Ludwig que atua como Co-orientador deste trabalho, sendo peça chave para a realização do mesmo. Sua equipe também merece elogios, principalmente pela colaboração e eficiência.

Agradeço também ao professor Doutor **Luciano da Fontoura Costa** (USP - São Carlos) pela sua colaboração e pelas idéias que sustentam parte deste trabalho.

Resumo

Nos últimos anos, milhares de seqüências de DNA e proteínas vêm sendo depositadas em bancos de dados públicos em todo o mundo. Atualmente, o principal desafio da Biologia Molecular Computacional é analisar e extrair informações úteis dessa grande quantidade de dados disponíveis. Este trabalho tem por objetivo estudar e comparar diversos métodos computacionais para a realização de busca de homologia e clustering (i.e., reconhecimento de padrões supervisionado e não supervisionado) em seqüências de nucleotídeos (i.e., DNA) e aminoácidos (i.e., proteínas).

Foi realizada uma vasta revisão bibliográfica dos principais métodos utilizados na área de reconhecimento de padrões, principalmente os mais utilizados em Biologia Computacional. É apresentado um resumo dos principais modelos utilizados na área e também alguns resultados preliminares de experimentos realizados com genes humanos utilizando-se uma nova forma de extração de características em seqüência de **DNA**.

Sumário

1 Introdução

Com o crescente número de genomas seqüenciados sendo disponibilizados atualmente [81], inclusive o humano[114, 58], um problema muito importante que surge imediatamente na área de Biologia Molecular é extrair informações desses enormes bancos de dados de seqüências.

A Biologia Molecular Computacional [72] consiste basicamente no desenvolvimento e uso de técnicas matemáticas e de Ciência da Computação para auxiliar a solução de problemas da Biologia Molecular.

Diversos problemas vêm sendo estudados nessa área: a comparação de seqüências de **DNA** [30, 124], montagem de fragmentos de **DNA** [2], mapeamento físico de **DNA** [3], árvores filogenéticas [125], reconhecimento de genes e partes de genes [101, 18], busca de homologia [49], clustering [29, 34], predição da estrutura de proteínas [97] etc.

O objetivo principal deste trabalho é estudar diversos métodos computacionais disponíveis atualmente para clustering e busca de homologia em seqüências de **DNA**, **RNA** e proteínas. Foi realizada uma vasta revisão bibliográfica dos principais métodos utilizados na área de reconhecimento de padrões, principalmente os mais utilizados em Biologia Computacional. É apresentado um resumo dos principais modelos e experimentos realizados com genes humanos.

Este trabalho está vinculado ao Projeto Temático **CAGE** [1] (do inglês, "*Cooperation for Analysis of Gene Expression*") (**FAPESP** nº 99/07390-0), que une esforços do Instituto de Química e do Instituto de Matemática e Estatística da Universidade de São Paulo com o objetivo de estudar os mecanismos de expressão gênica.

O autor recebeu apoio financeiro da **FAPESP** durante a elaboração deste trabalho, processo nº 01/03975-5

Uma versão eletrônica deste documento pode ser encontrada em:

http://www.vision.ime.usp.br/∼caetano/mestrado.html

1.1 Objetivos

O objetivo principal deste trabalho é estudar e analisar técnicas de reconhecimento de padrões aplicadas à análise de seqüências de **DNA**. Nos restringimos à análise de técnicas que têm por

[1] Para mais informações, sobre o grupo, http://www.vision.ime.usp.br/∼cage/

objetivo a classificação de regiões cromossômicas, em especial às técnicas utilizadas na busca de genes humanos.

Não realizamos o estudo de outras técnicas baseadas em seqüências biológicas como, por exemplo, a determinação da estrutura secundária ou terciária a partir da seqüência de uma proteína.

Pretendemos apresentar um resumo comparativo dos principais modelos utilizados pelas principais técnicas na área, bem como exemplos de utilização real de tais modelos.

Pretende-se aprofundar o estudo de algumas técnicas, principalmente de estimação, e implementar alguns testes de uma nova medida para extração de características de regiões gênicas.

Pretende-se realizar também testes que permitam verificar se uma nova técnica de extração de características baseada em freqüência de utilização de bases e dimensão fractal pode ou não ser usada para incrementar os principais algoritmos baseados nos modelos apresentados.

Já foi iniciada tal investigação (vide seção 4) e um cronograma para alcançar os objetivos finais é apresentado na seção 5.

1.2 Organização deste Documento

A próxima seção é uma introdução bem rápida a alguns aspectos da Biologia Molecular que estão relacionados com este trabalho. Na seção 2 há uma revisão dos diversos modelos e técnicas de reconhecimento de padrões utilizados na área de atuação do projeto. Na seção 3 apresentamos uma explicação de técnicas de extração de características que estamos avaliando. Os resultados preliminares de alguns experimentos realizados estão na seção 4. Na seção 5 apresentamos um plano de trabalho a ser realizado até julho de 2003, que é a data prevista para a finalização do mestrado e início da preparação para a defesa da dissertação. Na seção 6 há uma proposta para a estrutura da dissertação.

1.3 Alguns Aspectos da Biologia Molecular

O **DNA**, descoberto em 1953, pode ser descrito como uma fita dupla enrolada ao longo de um eixo em forma de hélice. Tal estrutura possui diversas unidades menores denominadas *nucleotídeos*. Existem diversos tipos de nucleotídeos que são diferenciados pelo tipo de base nitrogenada que possuem: **A** *adenina*, **G** *guanina*, **C** *citosina* e **T** *timina*. As bases adenina e guanina são denominadas *bases púricas*, enquanto a citosina e a timina são denominadas *bases pirimídicas*.

Uma fita de **DNA** é composta por uma seqüência linear de nucleotídeos. Uma molécula de **DNA** possui duas fitas emparelhadas de forma que cada base **A** de uma fita esteja "ligada" a uma base **T** da outra fita. Analogamente, estão ligadas bases **C** com bases **G**.

Um modelo simples de uma molécula de **DNA** é representá-la simplesmente como uma seqüência finita de letras que representam as 4 bases (**A**, **T**, **C** e **G**).

Tal simplificação é muito útil, pois é compacta e permite a análise do **DNA** como se fosse uma seqüência de letras. Além disso, muitos algoritmos já estão disponíveis para lidar com esse tipo de dados [48].

Moléculas de **DNA** compõem os *cromossomos* que por sua vez constituem o *genoma*. Cada organismo possui um genoma. Nos cromossomos estão localizados os *genes*, que são os trechos dos cromossomos responsáveis pela produção de *proteínas*.

Proteínas são macromoléculas muito importante para o organismo. Existem diversos tipos delas, dentre elas *proteínas estruturais* e *enzimas*. Proteínas são formadas por moléculas menores, denominadas *aminoácidos*. São vinte os tipos mais comuns de aminoácidos, raramente um pequeno conjunto de aminoácidos além desses 20 estão presentes. É a seqüência dessas moléculas que define cada tipo de proteína. Essa seqüência é codificada a partir da seqüência do **DNA**, onde cada aminoácido é codificado por uma região de três *nucleotídeos*, denominada *códon*.

Essa seqüência de aminoácidos também é conhecida como *estrutura primária*. Através da iteração dos elementos básicos da proteína, ela forma uma estrutura tridimensional, que pode ser estudada observando-se sua *estrutura secundária*, *estrutura terciária* e *estrutura quaternária*.

A pesquisa em Biologia Molecular Computacional está basicamente voltada para a compreensão da estrutura e função de genes e proteínas. Uma introdução detalhada ao assunto pode ser encontrada em [128].

1.3.1 Comparação de Seqüências

A área de Comparação e Casamento Aproximado de Seqüências é importantíssima na Biologia Molecular Computacional [48, Parte III], por causa dos processos de mutação e da presença de erros nos dados.

Existem diversas formas de se comparar seqüências, *subseqüências* e *subpalavras*[124]. É importante observar a diferença entre subseqüência e subpalavra. Os caracteres numa subpalavra

precisam ser um "pedaço" contínuo da seqüência original, enquanto os caracteres em uma subseqüência não precisam.

Similaridade Similaridade entre duas seqüências pode ser entendido como o quão "uma seqüência se parece com a outra". Para determinar a similaridade entre duas seqüências, utiliza-se algum *critério* para o *alinhamento* das mesmas.

O alinhamento entre duas seqüências é, intuitivamente, uma forma de dispor lado a lado os caracteres de duas seqüências dadas, permitindo tanto erros como acertos e permitindo também que caracteres de uma seqüência possam ser alinhado com espaços inseridos na outra seqüência (i.e., inserção de "buracos"). O critério, em geral, determina pesos para erros, acertos e inserção de espaços. A pontuação de um alinhamento simplesmente é a somatória de todas as penalidades e acertos obtidos.

A pontuação obtida no alinhamento ótimo entre as seqüências define seu grau de similaridade. Existem diversas formas de se escolher o critério, principalmente quando estamos lidando com cadeias de proteínas.

Duas seqüências são similares se "elas se parecem uma com a outra". Para se determinar a similaridade entre duas seqüências, utiliza-se algum critério para alinhamento das mesma e a pontuação obtida no alinhamento ótimo entre as seqüências define seu grau de similaridade. Existem diversas formas de se escolher o critério. Vamos apresentar um exemplo de critério.

Sejam s e t duas palavras. Um alinhamento entre s e t é um par $(s',\ t')$ obtidos de s e t, respectivamente, através da inserção de espaços nas palavras originais. O alinhamento $\alpha = (s', t')$ precisa satisfazer as seguintes regras:

1. $|s'| = |t'|$
2. A eliminação de todos os espaços de s' resulta em s
3. A eliminação de todos os espaços de t' resulta em t
4. $\forall i, 1 \leq i \leq |s'| = |t'|$, ambos $s'[i]$ e $t'[i]$ não podem ser espaços

Desta forma, podemos definir similaridade como sendo a maior pontuação de um alinhamento. Podemos utilizar um sistema de pontuação aditivo, que é composto por uma penalidade g, em geral um número real negativo, e por uma função $p : \Sigma \times \Sigma \to \mathbb{R}$ em que Σ é o alfabeto utilizado nas palavras. Para penalizarmos a insersão de um caracter espaço $\sqcup \notin \Sigma$, construímos $p' : \{\Sigma \cup \sqcup\} \times$

$\{\Sigma \cup \sqcup\} \to \mathbb{R}$ com as mesma regras de p e $\forall \sigma \in \Sigma$, $p'(\sigma, \sqcup) = p'(\sqcup, \sigma) = g$. Dessa forma, o valor da pontuação do alinhamento de α denotado por $score(\alpha)$ pode ser facilmente calculado da seguinte forma: $score(\alpha) = \sum_{i=1}^{|s'|} p'(s'[i], t'[i])$. Note que $p'(\sqcup, \sqcup)$ deve ser negativo ou não definido, de forma a impedir a inserção de espaços na mesma posição das duas seqüências do alinhamento.

Finalmente, definimos similaridade entre duas seqüências s e t de acordo com o nosso sistema de pontuação como sendo $sim(s,t) = \max_{\alpha \in \mathcal{A}(s,t)}(score(\alpha))$, em que $\mathcal{A}(s,t)$ é o conjunto de todos os alinhamentos entre s e t.

Note que o critério apresentado não consegue diferenciar alinhamentos com uma seqüência contínua de espaços de outros com seqüências alternadas de espaços. Isso pode não ser muito bom.

Distância Distância é uma medida de quanto duas palavras "diferem". Uma distância no conjunto E é uma função $d : E \times E \to \mathbb{R}$ de forma que:

1. $\forall x \in E, d(x,x) = 0$ e $\forall x \neq y, d(x,y) > 0$
2. $\forall x, y \in E, d(x,y) = d(y,x)$ (simetria)
3. $\forall x, y, z \in E, d(x,y) \leq d(x,z) + d(y,z)$

Uma distância muito utilizada na área de Biologia Computacional é a *distância de edição*. A idéia principal é transformar (editar) uma palavra na outra através do uso de uma série de *operações de edição* em caracteres individuais. As operações em geral permitidas são: *inserção* de um caractere, *remoção* de um caractere e *substituição* de um caractere. Atribui-se custos para cada uma das operações e a distância é simplesmente dada pela seqüência de transformações que tem a menor somatória de custos.

Vamos apresentar um exemplo de distância clássico em Biologia Computacional.

Podemos definir a distância no conjunto de palavras sob o alfabeto Σ como o "esforço" necessário para transformar uma palavra desse conjunto em outra do mesmo conjunto. A transformação admite dois tipos de operações:

1. Substituição de uma caractere $\alpha \in \Sigma$ por outro $b \in \Sigma, a \neq b$
2. Inserção ou deleção de um caractere arbitrário $\alpha \in \Sigma$

Para cada tipo de operação acima, associamos um custo, dessa forma, podemos definir o custo de um conjunto de operações, denominado *cost* como sendo a somatória dos custos de cada

operação isoladamente. Finalmente, podemos definir a distância entre duas palavras $s, t \in \Sigma^*$ como $dist(s,t) = \min_{\sigma \in \mathcal{S}(s,t)} (cost(\sigma))$, em que $\mathcal{S}(s,t)$ é o conjunto de todas as séries de operações que transformam s em t.

Observe que a *dist* obedece as regras de distâncias mostradas no início da seção se a função *cost* for sempre positiva e simétrica.

Dois problemas interessantes nessa área são justamente escolher as funções de custos tanto para similaridade quanto para distância e desenvolver algoritmos para calculá-las.

Casamento aproximado Diversas técnicas de casamento exato utilizadas para calcular similaridade e distância têm se mostrado inviáveis computacionalmente. Encontrar um alinhamento ótimo de multiplas seqüências, por exemplo, é um problema díficil computacionalmente [48, Capítulo 14], utiliza-se então diversas formas para obter um alinhamento aproximado.

Diversos algoritmos com simplificações como não permitir "buracos" nos alinhamentos foram desenvolvidos e são utilizados principalmente para a busca em grandes bancos de dados. Pretendemos estudar e utilizar diversos desses métodos, entre eles os utilizados pelos programas **BLAST** e **FASTA** para comparação de seqüências e métodos baseados em tabelas e bancos de dados como **PAM**, **PROSITE** e **Pfam** para comparação de proteínas. Alguns métodos muito conhecidos são **BLOCKS**, **BLOSUM** e **COG**.

2 Reconhecimento de Padrões

A área de reconhecimento de padrões [115] tem por objetivo a classificação de objetos num número de categorias ou classes. Essa classificação também pode ser entendida como a obtenção de uma "impressão digital" de cada conjunto de objetos que queremos reconhecer, aonde cada classificador corresponde a uma impressão digital.

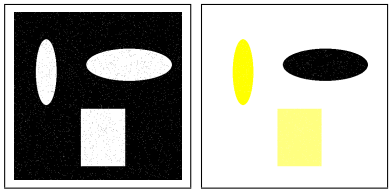


Figura 1: Exemplo de reconhecimento de padrões. Figura original e objetos reconhecidos

Por exemplo, suponha que se queira reconhecer os diversos objetos na Figura 1 que está cheia de ruído. Um modo bem simplista de fazer isso para esse caso é encontrar as componentes conexas da imagem e fazer um *threasholding* a partir do histograma das componentes conexas pegando só as "maiores". O resultado obtido são três regiões distintas como mostrado. Tal procedimento como acima descrito pode ser denominado *classificador*. As medidas utilizadas pelo classificação são chamadas *características* e o conjunto de características é chamado de *vetor de características*. No exemplo acima, o vetor de características simplesmente contém uma única característica que é o número de pixels de uma região conexa e o *classificador* simplesmente encontra vetor que tem os maiores valores de acordo com algum critério que podemos estabelecer como os valores que forem maior que algum x, por exemplo.

Um problema muito importante nessa área de reconhecimento de padrões é encontrar as características certas e portanto um vetor de características "útil". Por exemplo, para projetar um classificador que distingüa homens de mulheres, a característica *cor do cabelo* não é muito interessante. Já a característica *presença de barba* pode ser mais útil. Um vetor de características bom poderia indicar o número de cromossomos X presentes numa célula de pele do indivíduo. Com esse vetor seria bem fácil a classificação.

Na prática, no entanto, nem todas as informações que queremos estão disponíveis. Pode ser que somente uma foto do indivíduo esteja disponível para a classificação, desta forma, teremos que *extrair características* da foto para construir o vetor de características. Após essa primeira etapa, é necessário decidir quais das características geradas devemos utilizar. Em geral um número de características maior do que o necessário acaba sendo gerado e devemos então escolher o vetor de forma que contenha todas as características necessárias para o projeto de um bom classificador.

Tendo escolhido as características apropriadas, é necessário *projetar* o classificador e logo em seguida testá-lo.

Quando a classificação dos vetores de características estão disponíveis, ou seja, eu sei a que *classe* o vetor pertence (no exemplo, podemos ter vetores que pretencem a homens e outros que pertencem a mulheres), esse tipo de reconhecimento de padrões é denominado *reconhecimento de padrões supervisionado* (temos um "professor" para nos "ensinar a verdade"). Existem casos em que tal informação não está disponível. Nesse caso nos é fornecido um conjunto de vetores de caracterísitcas e o objetivo é encontrar similaridades agrupando os vetores por algum critério. Esse procedimento é denominado *reconhecimento de padrões não supervisionado* e é muito usado para agrupar seqüências de **DNA**, por exemplo.

Uma técnica específica muito comum para o projeto supervisionado de classificadores são os *classificadores bayesianos*. Neste modelo, cada classificador, ao invés de responder "sim" ou "não", indica o grau de proximidade do objeto em questão a cada classe que compõe um *conceito*. Dado um conjunto de classes (conceitos), a *decisão bayesiana* consiste simplesmente em escolher a classe que mais se aproxima do objeto sendo classificado.

As etapas básicas para processo de reconhecimento de padrões são:

1. **Escolha dos Dados**. Dentre todos os exemplos disponíveis, devemos escolher um subconjunto para treinamento, deixando o resto para os testes do classificador.
2. **Extração das Características**. Extraímos o maior número de características de testes a partir do subconjunto para treinamento escolhido.
3. **Seleção das Características**. As características devem ser muito bem escolhidas de forma a minimizar a redundância e maximizar a quantidade de informação "relevante" disponível.
4. **Projeto do Classificador**. O projeto pode ser realizado empiricamente por um operador ou automaticamente por aprendizado computacional.
5. **Testes dos Resultados**. Testes devem ser realizados, tanto com dados simulados como com reais. Testes são importantes para inferir se o processo está resolvendo o problema de forma aceitável.

2.1 Aprendizado Computacional Supervisionado

Projetar classificadores para o caso supervisionado, em geral, não é tarefa simples, exigindo um operador bem treinado e experiente na área, o que nem sempre está disponível. Vamos apresentar uma abordagem genérica para o projeto automático de classificadores para o reconhecimento de padrões supervisionado.

Informalmente, qualquer processo que é capaz de "aprender um conceito", a partir de exemplos que o ilustram, é denominado aprendizado [2]. Um exemplo simples de aprendizado é quando um professor tenta ensinar a um aluno algum conceito como o *sabor adocicado*, por exemplo. O professor pode ter um conjunto de alimentos (exemplos) com diversos sabores, todos em um "saco" e vai sortenado os alimentos e informando ao aluno "este é doce", "este não é doce". Dessa forma, espera-se que o aluno seja capaz de distingüir sabor doce dos demais sabores de alimentos.

Vamos apresentar uma introdução à modelagem do processo de aprendizado apresentado por [117] na qual está baseado o modelo **PAC** básico de Valiant [118] que consiste principalmente do aprendizado de "conceitos" que podem ser modelados por funções booleanas.

Consideremos um domínio D cujos elementos são as representações codificadas de um "mundo real". Seja Σ, denominado *alfabeto*, um conjunto para descrever elementos de D. Denotamos por Σ^n o conjunto de todas as n-uplas de elementos de Σ.

Seja $X \subseteq \Sigma^n$. Um conceito c é uma função definida por $c : X \longrightarrow \{0, 1\}$. X é denominado o *espaço de exemplos* e seus elementos denominados *exemplos*. Um exemplo $x \in X$ é um *exemplo positivo* se $c(x) = 1$ e é um exemplo negativo se $c(x) = 0$. Dado os conjuntos de todos os exemplos positivos, ele determina e é determinado por um conceito c. Por essa razão, um conceito c pode ser também encarado como um subconjunto de X.

O conjunto C de todos os possíveis conceitos é denominado *espaço de conceitos*. O objetivo de um processo de aprendizado é produzir um conceito h que seja uma boa aproximação de um conceito alvo $t \in C$ que queremos aprender. O conjunto de todos os conceitos que podem ser aprendidos é denominado *espaço de hipóteses* e denotado por H.

Uma *amostra de tamanho* m é uma seqüência de m exemplos, ou seja, uma m-upla $x = (x_1, x_2, \ldots, x_m) \in X^m$. Uma *amostra de treinamento* s de tamanho m é um elemento de $(X \times \{0, 1\})^m$, ou seja, $s = ((x_1, b_1), (x_2, b_2), \ldots, (x_m, b_m))$ em que $b_i = t(x_i)$ indica se os exemplos são positivos ou negativos relativamente ao conceito t. Dizemos que s é consistente se $x_i = x_j \Longrightarrow b_i = b_j, 1 \leq i, j \leq m$.

Uma *algoritmo de aprendizado* é uma função L que associa a cada amostra de treinamento s consistente de tamanho m, relativo a um conceito alvo $t \in H$, uma hipótese $h \in H$. Denotamos

[2] É importante distingüir aprendizado de compreensão e conciência. Apesar do computador ser capaz de "aprender" algo, ele não capaz de compreender o mesmo, muito mesmo ter conciência daquilo que está fazendo, como mostra Searle no segundo capítulo de [103] onde ele expõe sua famosa alegoria do "Quarto Chinês"

$L(s) = h$. Uma hipótese $h \in H$ é *consistente com uma amostra* s se $h(x_i) = b_i$ para cada $1 \leq i \leq m$. Um algoritmo de aprendizado é *consistente* se ele é consistente com todas as possíveis amostras de treinamento s consistentes.

O modelo de aprendizado PAC O modelo **PAC** (do inglês, "Problably Approximately Correct") proposto por Valiant considera que $\Sigma = \{0, 1\}$. Além disso, supõe que os elementos do espaço de exemplo X não são contraditórios, ou seja, todas as amostras são consistentes. Vamos também supor que $C = H$.

Suponha que X define um espaço de probabilidades e seja μ a distribuição de probabilidades correspondente a X^m por μ^m. Dado $Y \subset X^m$, interpretamos o valor $\mu^m(Y)$ como sendo a probabilidade de uma amostra aleatória de m exemplos, obtidos de X, segundo a distribuição μ, pertencer a Y. Seja $S(m, t)$ o conjunto de amostras de treinamento de tamanho m. Existe uma bijeção entre X^m e $S(m, t)$ expressa por uma função $\phi : X^m \longrightarrow S(m, t)$, que a cada m-upla de exemplos $x = (x_1, x_1, \ldots, x_m)$ associa a amostras $s = ((x_1 b_1), (x_2, b_2), \ldots, (x_m, b_m))$.

Logo, $\mu^m\{s \in S(m, t) : s$ tem a propriedade $P\}$ pode ser interpretada como $\mu^m\{x \in X^m : \phi(x)$ tem a propriedade $P\}$. Isto permite calcularmos o erro da hipótese $h = L(s)$ e a probabilidade deste erro ser menor que um dado valor ϵ em função da distribuição de probabilidade sobre X.

Definição: Um algoritmo L é **PAC** para um espaço de hipóteses H se, dados um número real $\delta(0 < \delta < 1)$ e um número real $\epsilon(0 < \epsilon < 1)$, então existe um inteiro positivo $m_0 = m_0(\delta, \epsilon)$ tal que para qualquer conceito alvo $ta \in H$ e para qualquer distribuição de probabilidade μ sobre X, sempre que $M \geq m_0$, $\mu^m\{s \in S(m, t) : er_\mu(L(s), t) < \epsilon\} > 1 - \delta$.

O que torna esse modelo interessante, é o fato de que é possível encontrar um algoritmo **PAC** que produz uma hipótese h, tal que, com alta probabilidade $(1 - \delta)$, o erro entre a a hipótese h e o conceito alvo t seja menor do que ϵ. O termo "provavelmente aproximadamente correto" está associado à esta idéia.

Intuitivamente, algoritmos como descrito acima conseguem produzir hipóteses h cada vez melhores em relação ao conceito alvo t conforme o número de exemplos vai aumentando. Ou seja, quanto mais informações sobre t é fornecida, h melhora. Segundo [6], um limitante superior no número de exemplos para garantir que o erro apresentado no parágrafo anterior é $m_0(\delta, \epsilon) = \frac{1}{\epsilon} \ln(\frac{|H|}{\delta})$, em que $|H|$ representa a cardinalidade do espaço de hipóteses.

Esse modelo apresenta diversas restrições, entre elas a suposição de que não existem exemplos

contraditórios no espaço de exemplos X, isto é, dados dois exemplos $x_i, x_j \in X$, se $x_i = x_j$ então $b_i = b_j$. Isso não é muito interessante porque na prática se verifica que é comum encontrar exemplos contraditórios.

Haussler [51] generaliza o modelo **PAC** de forma a permitir exemplos contraditórios e dados em alfabetos que não são binários.

2.2 Aprendizado Computacional não Supervisionado

Quando não estão disponíveis informações sobre a classificação dos objetos sendo estudados (ou seja, não há "professor ensinando o aluno"), utilizamos aprendizado computacional não supervisionado para agrupar os vetores de características por similaridade, processo também conhecido como *clustering*. Uma aplicação muito comum em Biologia Computacional é agrupar seqüências de **DNA** e proteínas de acordo com algum *critério*.

Considere o exemplo de sabores apresentado anteriormente. Se criarmos um clustering usando o critério "cor dos alimentos", vamos provavelmente obter um agrupamento bem diferente de outro agrupamento construído utilizando-se o critério "acidez dos alimentos". Isso mostra que o processo de clustering pode resultar em resultados muito diferentes.

Além das etapas básicas já citados na Seção 2, o desenvolvimento de um processo de clustering requer:

1. **Escolha da medida de proximidade**. É uma medida que mede o quão "similar" ou "diferente" dois vetores de características são.
2. **Escolha do critério de clustering**. O critério define o quão "sensível" vai ser o processo. Em geral expresso por uma função de custo ou outros parâmetros como o número de agrupamentos desejados.
3. **Escolha do algoritmo de clustering**. Cada tipo de algoritmo de clustering produz um tipo de resultado diferente. É necessário descobrir para cada conjunto de dados que tipo de algoritmo é "melhor".
4. **Interpretação dos Resultados**. É necessário investigar os resultados obtidos pelo processo de clustering desenvolvido através de experimentos que comprovem os resultados obtidos para que possamos encontrar as conclusões corretas.

A precisão de estimação de um processo de clustering depende muito de diversos fatores, entre eles a separação entre as *classes de congruência*, ou seja, a diferença entre as palavras geradas pelas gramáticas sendo utilizadas para os testes.

Um problema muito difícil de resolver nessa área é determinar o número de agrupamentos. Em

geral, somente um subconjunto de todas as possíveis partições do conjunto de exemplos é avaliado. Os resultados dependem muito do algoritmo utilizado e do critério escolhido.

Existem muitos algoritmos para clustering, uma boa introdução ao assunto pode ser encontrada no Capítulo 11 de [115] e no excelente artigo de Jain [57]. Podemos dividir os algoritmos de clustering nas seguintes categorias:

- **Algoritmos seqüênciais**. Em geral são muito rápidos e na maioria dos casos o resultado depende da ordem em que os exemplos são apresentados. Esses algoritmos tendem a produzir agrupamentos compactos [115, Capítulo 12].
- **Algoritmos hierárquicos**. Os algoritmos desta categoria podem ser divididos em *aglomerativos* e *divisores*. O primeiro tipo produz uma seqüência de agrupamentos cujo número dos mesmos é reduzido a cada passo ao juntar um agrupamento com outro. Já o segundo tipo é o oposto. Eles geram uma seqüência de agrupamentos cujo número dos mesmos aumenta a cada passo ao dividir um agrupamento em dois [115, Capítulo 13].
- **Algoritmos baseados em otimização de função de custo**. Em geral tem um número fixo de agrupamentos e tentam maximizar uma dada função. Quando um máximo local é encontrado o algoritmo pára. Em geral os algoritmos probabilísticos e de Fuzzy são dessa categoria [115, Capítulo 14].
- **Outros tipos**. Existem diversos outros tipos como algoritmos para genética, algoritmos estocásticos e algoritmos baseados em técnicas de transformações morfológicas [115, Capítulo 15].

2.3 Abordagens para o Reconhecimento de Padrões

Existem diversas abordagens para se realizar as tarefas necessárias para o reconhecimento de padrões. Dentre elas, podemos destacar:

- Casamento (*template matching*)

  É a abordagem mais simples. Dado um protótipo do conceito a ser reconhecido, simplesmente realiza-se a busca deste conceito simplesmente casando o protótipo com o espaço de busca. Um exemplo clássico de programa que utiliza esta abordagem é o **BLAST**. Uma das grandes desvantagens desta abordagem é que não é muito flexível, não permitindo muitas variações no objeto que se procura.

- Abordagem Sintática

  É uma abordagem inerentemente hierárquica no sentido de que padrões podem conter outros padrões recursivamente, de forma que o padrão principal constitui uma sintaxe. Permite que padrões sejam altamente flexíveis através do aumento da complexidade computacional. Exemplos de técnicas baseadas nesta abordagens são as técnicas baseadas em **HMM** (do inglês, Hidden Markov Models - Modelos Escondidos de Markov) e técnicas baseadas em gramáticas.

- Redes Neurais

É um modelo inspirado nas redes de neurônios, em que cada unidade básica representa um neurônio artificial com suas ligações a outros neurônios. Pode ser usada para representar funções complexas e não lineares. Pode-se facilmente adaptar a rede a novos dados, mas é uma das abordagens em que é mais difícil introduzir informação *a priori*.

- Estatística

  Utiliza análises estatísticas dos vetores de características para desenhar um classificador.

Uma explicação mais detalhada de cada abordagem acima, pode ser encontrada no artigo de Jain [56]. Ao longo deste trabalho, estudamos todas as abordagens acima, dando ênfase para as abordagens sintáticas e estatísticas, em que estão incluídos os modelos de Markov e os modelos baseados em gramáticas. Note que a divisão acima foi apresentada somente para facilitar a visualização, visto que muitas abordagens são utilizadas em conjunto, como é feito neste trabalho.

2.4 Modelos de Markov

Modelos de Markov são modelos estatísticos muito utilizados na área de Biologia Computacional para modelar estocasticamente famílias de seqüências. Pretendemos nesta seção apresentar o modelo básico *Cadeias de Markov* e uma de suas principais extensões conhecida como **HMM**. O objetivo principal destes modelos é fornecer uma forma para modelar eventos que são fisicamente observáveis.

2.4.1 Cadeias de Markov

Seja S um sistema que pode ser descrito em qualquer tempo como estando num estado de um conjunto de n estados $S_1, S_2, \ldots, S_n$.

Em intervalos de tempo regulares, o sistema sofre uma mudança de estado de acordo com um conjunto de probabilidades associadas a cada estado. Denotamos os instantes associados a cada estado como $t = 1, 2 \ldots$ e denotamos o estado no tempo t de q_t.

Um sistema probabilístico completo em geral requer que o próximo estado dependa do estado atual bem como de todos os anteriores.

Definimos então um modelo de Markov de ordem d como dependendo somente dos últimos d estados. Desta forma,

$$P[q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots]$$

$$= P[q_t = S_j | q_{t-1} = S_{i_1}, q_{t-2} = S_{i_2}, \ldots, q_{t-d} = S_{i_d}]$$

Além disso, só consideramos os sistemas em que a parte direita da equação acima não depende do tempo. Por exemplo, para o modelo de ordem 1, demos que $a_{ij} = P[q_t = S_j | q_{t-1} = S_i], 1 \leq i, j \leq n$. Tal definição também é conhecida como *Propriedade de Markov*. Na Figura 2 temos um exemplo de cadeia de Markov com algumas transições de estados.

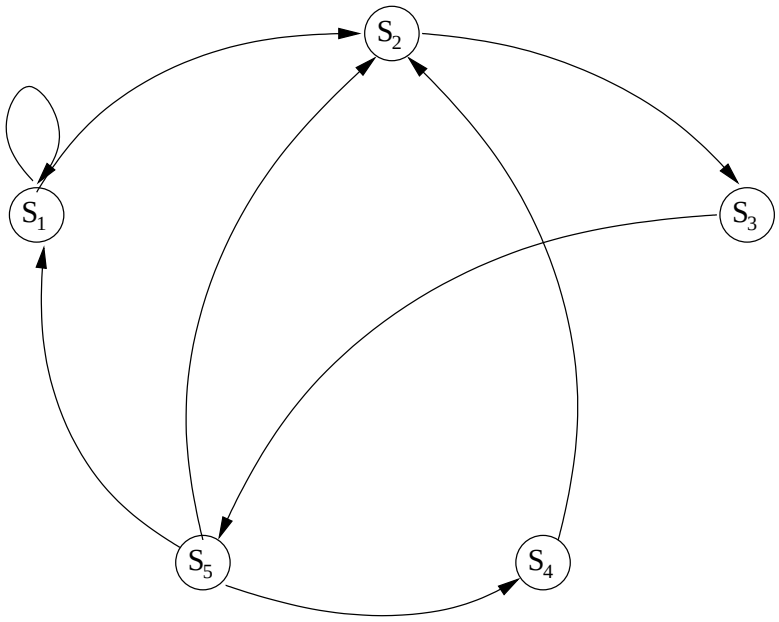


Figura 2: Exemplo de cadeia de Markov com 5 estados

Podemos chamar o processo estocástico acima como um Modelo de Markov observável, porque a saída do processo é simplesmente a cadeia (caminho) de estados percorridos em cada instante de tempo, e cada estado corresponde a um evento fisicamente observável.

Exemplo: Modelando o clima

Podemos utilizar um Modelo de Markov com 3 estados para modelar o clima. Consideramos que as mudanças só acontecem uma vez por dia e as observações são de acordo com os seguintes estados:

- Estado 1: Chuva
- Estado 2: Nublado
- Estado 3: Sol

Podemos escrever uma matriz que descreve o nosso modelo: $A = a_{ij} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$

Esta matriz está representada visualmente na Figura 3.

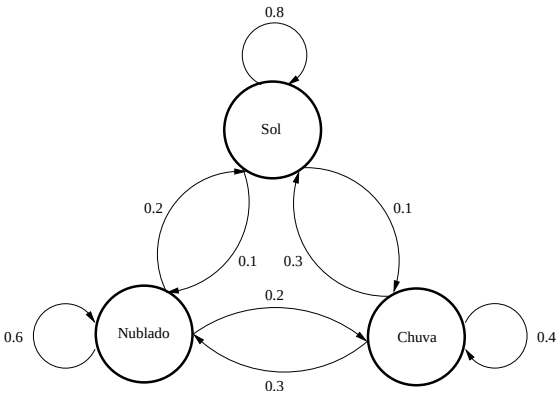


Figura 3: Exemplo de Cadeia de Markov para modelar o clima.

Podemos então fazer uma série de simulações de acordo com o modelo. No exemplo apresentado, é muito mais provável, por exemplo, que se permaneça no estado 3 do que se mude do estado 3 para o estado 2 em 1 passo.

2.4.2 Modelos Escondidos de Markov (HMM)

HMM é uma extensão do modelo de Cadeias de Markov. As bases deste modelo foram publicadas por Baum e seus colegas no final dos anos 60 e início dos anos 70.

Um Modelo Escondido de Markov – **HMM** (*hidden Markov model*) é um processo estocástico, dito oculto, imerso em outro processo estocástico, dito observável, onde cada estado corresponde a um evento observável. Esse processo não observável pode ser visto somente a partir de um outro conjunto de processos estocásticos que produz a seqüência de observações.

Elementos de um HMM

Um **HMM** é uma quíntupla constituída de:

1. Um conjunto $S = \{S_1, S_2, \ldots, S_n\}$ com n estados
2. Um conjunto $V = \{V_1, V_2, \ldots, V_m\}$ com m símbolos
3. A distribuição de probabilidade de transição de estados $A = a_{ij}$
   $a_{ij} = P[q_t = S_j | q_{t-1} = S_{i_1}, q_{t-2} = S_{i_2}, \ldots, q_{t-d} = S_{i_d}], \qquad 1 \leq i, j \leq n.$
4. A distribuição de observação de cada símbolo em cada estado j, $B = \{b_j(k)\}$, $b_j(k) = P[V_k \textbf{ em } t | q_t = S_j]$, $1 \leq j \leq n$, $1 \leq k \leq m$.
5. A distribuição do estado inicial, $\pi = \{\pi_i\}, \pi_i = P[q_1 = S_i]$, $1 \leq i \leq n$.

Dizemos que um **HMM** é de ordem n caso a decisão do próximo estado só dependa dos últimos n estados.

O grande diferencial desta extensão é justamente essa sobreposição de processos estocásticos. Uma das limitações das cadeias de Markov é justamente que cada estado, em geral, corresponde a um evento fisicamente observável. **HMM**s justamente contornam esta limitação permitindo que cada estado emita o evento fisicamente observável de acordo com uma determinada distribuição, passando a ser o processo de escolha das distribuições escondido.

Exemplos para modelar lançamentos de moedas − cara × coroa

Suponha que, dado um conjunto de moedas, lança-se uma de cada vez e informa-se o resultado. O número de moedas não é conhecido, sendo assim, não é informada qual moeda foi lançada. A moeda escolhida não é necessariamente idônea.

Como não sabemos o número de moedas, podemos inicialmente supor que só há uma moeda sendo lançada e podemos facilmente modelar o problema utilizando-se uma cadeia de Markov com dois estados como na Figura 4 em que um estado corresponde a cara e outro a coroa. Basta então estimar as probabilidades de transição.

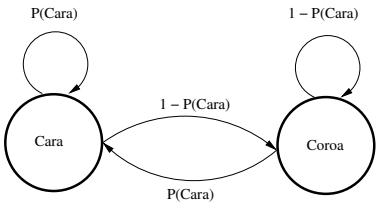
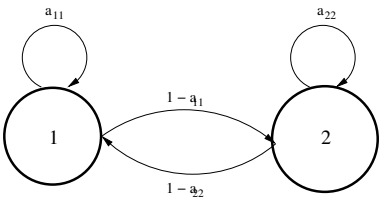


Figura 4: Modelando o problema da moeda utilizando-se cadeia de Markov

Da mesma forma, podemos supor que há duas moedas sendo lançadas, e que as moedas são escolhidas de acordo com um processo estocástico que só depende da última moeda lançada. Desta forma, podemos modelar o problema utilizando um **HMM** de ordem 1 com dois estados como exemplificado na Figura 5 em que cada estado corresponde a uma moeda. Basta então estimar o viés de cada moeda (que corresponde à distribuição do estado) e também o processo estocástico de escolha da moeda, tudo num único passo.

Não satisfeitos com o resultado, podemos supor que há três moedas sendo lançadas e que as moedas são escolhidas de acordo com um processo estocástico que só depende da última moeda



$P(\textbf{Cara}) = P_1$ $\quad$ $P(\textbf{Cara}) = P_2$
$P(\textbf{Coroa}) = 1 - P_1$ $\quad$ $P(\textbf{Coroa}) = 1 - P_2$

Figura 5: Modelando o problema da moeda utilizando-se **HMM** com 2 estados. P_1 e P_2 correspondem às probabilidades de "cara" ser emitida caso se esteja no estado 1 ou 2 respectivamente.

lançada. Desta forma, podemos modelar o problema utilizando um **HMM** de ordem 1 com três estados como na Figura 6 de forma análoga ao caso anterior.

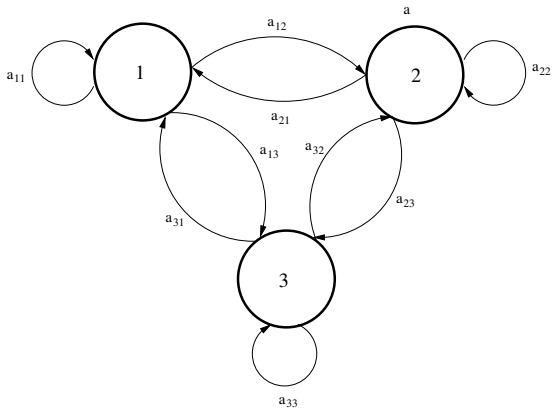


Figura 6: Modelando o problema da moeda utilizando-se **HMM** com 3 estados. Para cada estado temos uma probabilidade de "cara" ser emitida.

Nos casos anteriores, temos 1, 4 e 9 variáveis livres respectivamente. Cabe ao operador determinar qual modelo se adapta melhor aos dados. Note que, caso a escolha das moedas seja feita de forma completamente aleatória, o primeiro modelo conseguirá resultados muitos bons independente do número de moedas.

Exemplo da Urna com Bolas

Suponha que existem n urnas bem largas numa sala. Em cada urna, há um conjunto de bolas coloridas com no máximo m cores. Fazemos um sorteio da seguinte forma: um indivíduo na sala fechada escolhe, de acordo com um processo estocástico qualquer, uma urna inicial e sorteia uma bola, revelando somente sua cor. A bola é recolocada na urna e uma nova urna é escolhida de acordo

com um processo estocático associado à urna atual. Repete-se o processo de sorteio e escolha de uma nova urna.

Uma modelagem óbvia para o problema é a que cada estado representa uma urna e cada cor de bola associada a essa urna é representado pela probabilidade da cor ser emitida no estado. A escolha da urna é ditada pela matriz de transição do **HMM**.

Os três principais problemas em HMM

1. Dados uma observação e um modelo, como calcular eficientemente a probabilidade da observação ser gerada pelo modelo?
2. Dados uma observação e um modelo, como escolher um caminho que seja máximo de acordo com algum critério que tenha algum significado?
3. Como ajustar os parâmetros do modelo de forma a minimizar alguma função de erro

Em geral os dois primeiros problemas podem ser facilmente resolvidos utilizando-se o algoritmo de Viterbi baseado em programação dinâmica, mas a complexidade do problema depende muito dos critérios utilizados.

Vamos dar ênfase na abordagem do último problema.

Resolvendo o último problema

Podemos dividir o problema de ajuste de parâmetros em outros dois subproblemas:

1. Estimação da arquitetura, ou seja, S e o grafo de S.
2. Estimação das probabilidades dado A, B e π dado S.

Em geral, em bioinformática só se realiza a segunda parte, que já é bem complicada, porque o espaço de hipóteses é muito grande e a curva de erro não é conhecida e precisa ser estimada também. Note que a separação do terceiro problema em duas partes é realizada somente para fins didáticos. Não faz sentido realizar somente a primeira parte, visto que como a camada é escondida então a arquitetura não pode ser determinada somente olhando-se os exemplos.

O que em geral se faz é reduzir o espaço de hipótese utilizando-se informação biológica ou mesmo algumas técnicas como as baseadas em reticulados.

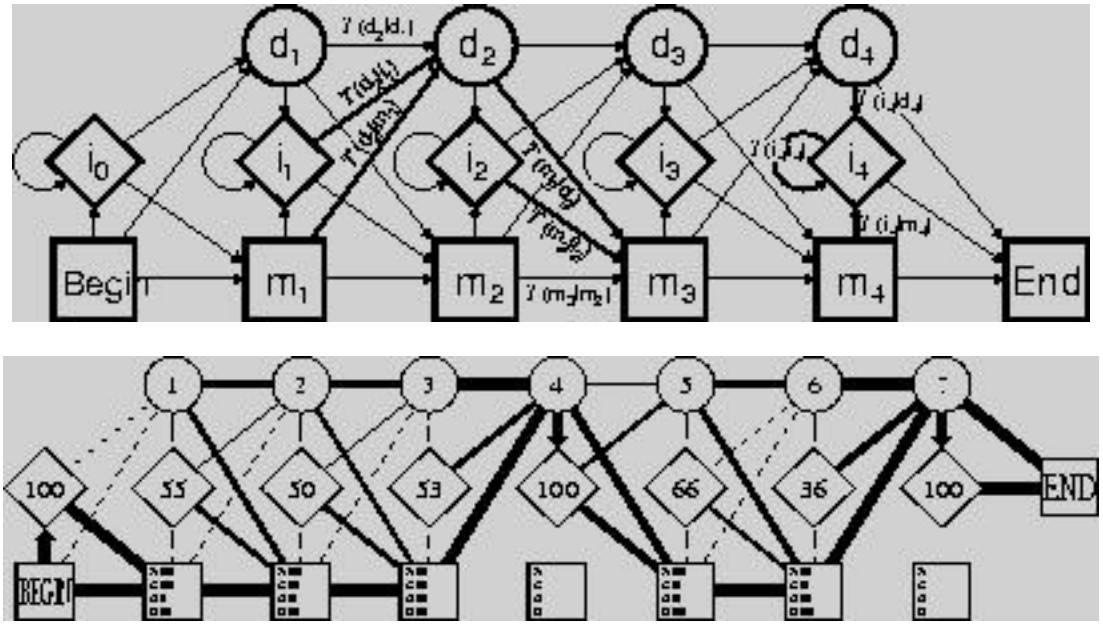


Figura 7: Exemplo de modelo extraído de [55] para modelar alinhamentos e exemplo de inferência do modelo.

Modelando *domains*, *motifs* e alinhamentos

Para a aplicação de **HMM** em Biologia Computacional, é importante observar que até então falamos de emissão e geração. Queremos reconhecer padrões e não gerar, mas no fundo isso tudo é equivalente.

Dizemos que um **HMM** reconhece uma palavra, se e somente se ele gera essa palavra.

Como o espaço de busca é muito vasto, em geral é necessário introduzir informação biológica nos algoritmos. Além disso, é necessário supor que o padrão que queremos modelar é computável pelo modelo.

A idéia básica em ambos os problemas é modelar inserções, deleções e *matches* utilizando estados para cada situação.

Na Figura 7 temos um exemplo de modelo baseado em **HMM** para modelar inserções (estados nomeados com i), deleções (estados nomeados com d) e emparelhamento (estados nomeados com m). A grossura da aresta entre os estados é diretamente proporcional à probabilidade de se mudar de um estado para o outro. Em estados de emparelhamento temos a distribuição aceita para cada letrinha. Em estados de inserção temos valores proporcionais à probabilidade de continuar inserindo e nos estados de deleção temos o número do passo.

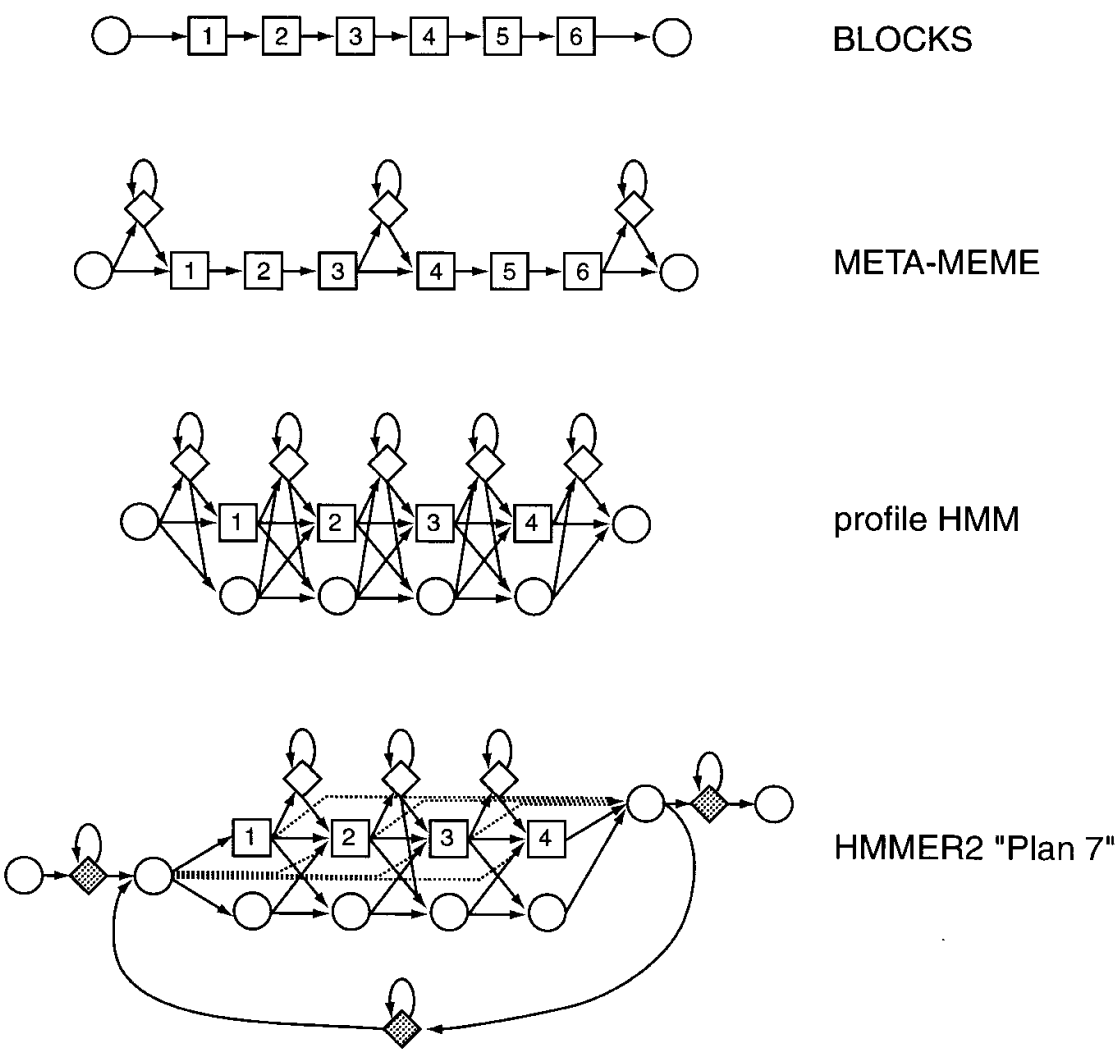


Figura 8: Exemplo de modelos extraídos de [33] utilizados em diversas aplicações conhecidas em Biologia Computacional

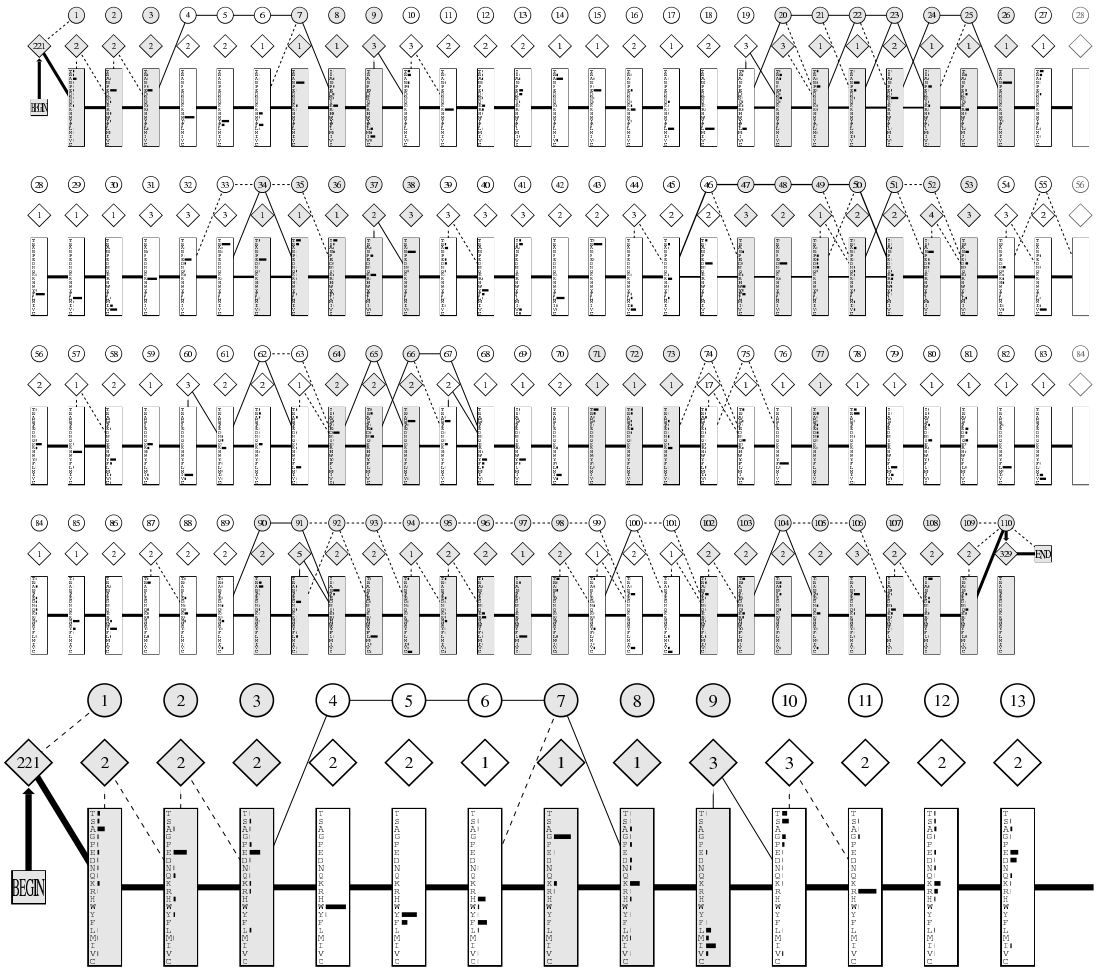


Figura 9: Exemplo de modelo para modelar o domínio SH2 extraído de [55]. A seção inicial está apresentada em baixo do modelo completo utilizando-se uma resolução maior. Os estados não sombreados representam elementos da estrutura secundária.

Na Figura 8 temos alguns modelos clássicos utilizados por algumas aplicações importantes em Biologia Computacional.

Um domínio muito conhecido é o SH2. Na Figura 9 temos um exemplo de arquitetura que modela tal domínio com técnicas parecidas com as do exemplo anterior, só que usando aminoácidos.

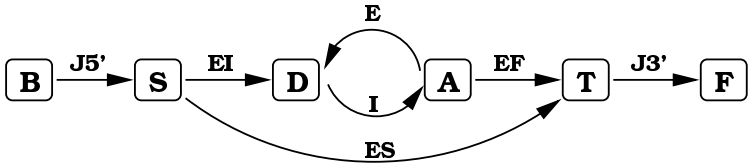
2.4.3 Modelo de Markov Escondido Generalizado

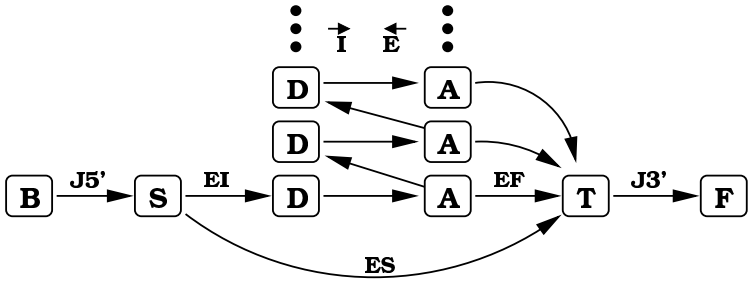
GHMM é o Modelo de Markov Escondido Generalizado, ou seja, em cada estado, ao invés de se ter uma distribuição para as letras do alfabeto, se tem distribuições para palavras compostas de letras do alfabeto. Este é o modelo em geral utilizado para a especificação dos principais algoritmos de predição em Biologia Computacional, principalmente os de predição de genes. A grande vantagem é que se pode utilizar outros modelos para se estimar a distribuição de probabilidade de cada estado.

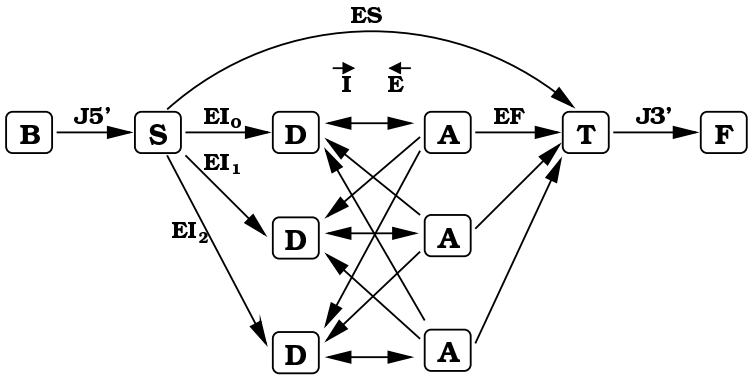
Predição de Genes

A idéia básica da maioria dos softwares é usar **GHMM** como uma camada multiresolução. Para tal, define-se estados que simbolizem alguma informação biológica, como a distribuição do tamanho dos éxons, a distribuição do tamanho dos íntrons, a freqüência de utilização de bases de cada região entre outras propriedades.

A seguir temos três exemplos extraídos de [64] de modelos de complexidade crescente utilizados para se procurar genes humandos.







2.5 Linguagens Formais e Modelos Estocásticos

Nesta seções vamos introduzir alguns conceitos básicos necessários para modelar seqüências de **DNA** e proteínas utilizando-se gramáticas estocásticas.

2.5.1 Conceitos Básicos

Seja Σ um conjunto, e seja Σ^* o conjunto de seqüências finitas de elementos de Σ. Se $\sigma_1, \sigma_2, \ldots, \sigma_n$ forem elementos de Σ, para algum $n \geq 1$, a seqüência $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ será denotada por: $\sigma_1\sigma_2 \ldots \sigma_n$,

$n \geq 1$. A seqüência vazia () será denotada por λ. Os elementos de Σ^* serão chamados de *palavras*, sendo λ a *palavra vazia*. Σ será chamado de *alfabeto* e seus elementos de *letras*.

Dada duas palavras não vazias em Σ^*, $s = \sigma_1\sigma_2 \ldots \sigma_n$ e $t = \tau_1\tau_2 \ldots \tau_m$, $(\sigma_i, \tau_j \in \Sigma)$, a sua concatenação st é a palavra $st = \sigma_1\sigma_2 \ldots \sigma_n\tau_1\tau_2 \ldots \tau_m$. Para qualquer $s \in \Sigma^*$, $s\lambda = \lambda s = s$.

Dada uma palavra não vazia, $s = \sigma_1\sigma_2 \ldots \sigma_n$, o inteiro n é o seu comprimento denotado por $|s|$. O comprimento $|\lambda|$ da palavra vazia é zero. Dessa forma, para s e t em Σ^*, $|st| = |s| + |t|$. De forma análoga, denotamos por $|s|_\sigma$ o número de ocorrências da letra σ em s.

Uma *linguagem sobre* Σ (ou simplesmente linguagem) é um subconjunto de Σ^*.

Dois exemplos simples de linguagens sobre o alfabeto $\Sigma = \{\mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G}\}$ são: $L_1 = \{s \in \Sigma^* : |s| = 3\}$ e $L_2 = \{s \in \Sigma^* : |s|_{\mathbf{A}} = 2\}$.

L_1 é uma linguagem finita com 84 palavras não vazias. L_2 é uma linguagem com um número infinito de palavras.

A concatenação de duas linguagens L_1 e L_2 é a linguagem $L_1L_2 = \{st \in \Sigma^* \mid s \in L_1, t \in L_2\}$.

2.5.2 Gramáticas

Uma gramática G consiste de:

- Um alfabeto finito não vazio V
- Um subconjunto próprio Σ de V
- Um subconjunto finito P de $V^*(V \setminus \Sigma)V^* \times V^*$
- Um elemento S de $V \setminus \Sigma$

A gramática G acima será denotada por (V, Σ, P, S). O alfabeto Σ é chamado de *alfabeto terminal* cujo símbolos são denominados *símbolos terminais*, enquanto o conjunto $N = V \setminus \Sigma$ é chamado de *alfabeto não terminal* cujo símbolos são denominados *símbolos não terminais*. Os elementos do conjunto P são chamados de *produções*. Uma produção (s, s') de P costuma ser denotada por $s \longrightarrow s'$. s e s' são chamados *lado esquerdo* e *lado direito* respectivamente da produção (s, s'). Note que o lado esquerdo de cada produção contém sempre algum símbolo não terminal. O símbolo S em N é chamado de *símbolo inicial*.

Uma *derivação* na gramática G é uma seqüência finita não vazia $d = (s_0, s_1, \ldots, s_n)$ de elementos de V^*, tal que para cada $0 \leq i \leq n$, existem palavras t_i, t'_i, u_i e u'_i em V^*, tais que $s_i = t_iu_it'_i$,

$s_{i+1} = t_i v_i t' i$ e $u_i \longrightarrow v_i$ é uma produção de G. O natural n é chamado *comprimento da derivação* d e é denotado por $|d|$.

Uma derivação $d' = (s_0, s_1)$ de comprimento um em G é denotada por $d' : s_0 \Longrightarrow_G s_1$ e a derivação d acima é denotada por $d : s_0 \Longrightarrow_G s_1 \Longrightarrow_G \ldots \Longrightarrow_G s_n$ ou por $d : s_0 \Longrightarrow_G^* s_n$.

A *linguagem gerada* pela gramática G é denotada por $|G|$ e é dada por $|G| = \{s \in \Sigma^* \mid S \Longrightarrow_G^* s\}$.

O tipo de gramática acima descrito, também conhecido como gramáticas do tipo 0 ou sem restrições, gera uma linguagem se e somente se a linguagem é *recursivamente enumerável* como provado no Teorema 4.6.1 de [67].

A classe de linguagens recursivamente enumeráveis é muito grande [85, Capítulos 3 e 20], inclusive, decidir se uma palavra dada pertence a uma linguagem qualquer dessa classe, pode ser indecidível ou inviável computacionalmente.

Atualmente, não somos capazes de computar eficientemente linguagens que estão além da classe PTIME [22, Capítulo 36], ou seja, problemas que não conseguimos decidir usando algum modelo clássico de computação como Máquinas de Turing determinísticas em tempo no máximo polinomial.

Tendo em mente essa limitação, vamos introduzir a hierarquia de Chomsky [21] que restringe o conjunto de produções das gramáticas. O objetivo principal é obtermos classes de gramática que geram linguagem "eficientemente" computáveis.

Definição: G é uma **gramática sensível ao contexto** ou do tipo 1 se suas produções são da forma $S \longrightarrow \lambda$ ou $uAv \longrightarrow usv$, com $u, v \in V^*, A \in V \setminus \Sigma$, $s \in V^+$.

Definição: G é uma **gramática livre de contexto** ou do tipo 2 se suas produções são da forma $A \longrightarrow a$, com $A \in V \setminus \Sigma$ e $s \in V^*$.

Definição: G é uma **gramática regular** ou do tipo 3 se suas produções são da forma $A \longrightarrow s$, com $A \in V \setminus \Sigma$ e $s \in \Sigma^*$ ou $A \longrightarrow sB$, com $A, B \in V \setminus \Sigma$ e $s \in \Sigma^*$.

Seja G uma gramática do tipo $i, i = 1, 2, 3$. Dizemos que a linguagem gerada por G é do tipo i. As famílias de linguagens $L_i, 0 \leq i \leq 3$ constituem a hierarquia de Chomsky. Importante observar que toda gramática G do tipo $i, i > 0$ é também uma gramática do tipo $i - 1$ e que o conjunto de linguagens de gramáticas do tipo i está propriamente contido no conjunto de linguagens do tipo $i - 1$.

Para decidir se uma palavra s dada pertence a uma gramática G do tipo 1, gastamos espaço

linear em relação ao tamanho da palavra [108, Capítulo 3]. O tempo necessário para decidir se s pertence à G pode vir a ser exponencial, o que torna inviável a sua utilização com os equipamentos disponíveis atualmente.

Para as linguagens livres de contexto e regulares podemos decidir se uma dada palavra s pertencem à linguagem eficientemente. Esse é o principal argumento para utilizarmos as linguagens do tipo 2 e 3 nos nossos trabalhos apesar de serem modelos bem simples em relação aos fenômenos que já se conhecem como os que Searls [105, 106] apresenta. Tudo isso motivou Searls a estudar uma classe de gramática intermediária chamada *string variable grammar* [104, 107, 105] que não é tão difícil de reconhecer quanto as linguagens sensíveis ao contexto mas é mais genérica que as livres de contexto.

2.5.3 Gramáticas Estocásticas

Quando modelamos somente alguns aspectos do **DNA**, diversos eventos demonstram uma natureza estocástica, entre eles, a freqüência de determinados trechos das palavras (tal fato muito importante não pode ser modelado pelas gramáticas comuns). Além disso, quando modelamos seqüências de **DNA** ou proteínas, devemos levar em conta as diversas mutações podem ocorrer no código genético e os diversos erros que podem ocorrer no processo de leitura das moléculas, como por exemplo erros de seqüenciamento e montagem. Tudo isso nos sugere uma abordagem estocástica para o problema.

Uma extensão simples que podemos fazer ao modelo de gramáticas apresentado anteriormente é utilizar gramáticas estocásticas capazes de lidar com linguagens estocásticas.

Uma gramática estocástica G consiste de:

- Um alfabeto finito não vazio V
- Um subconjunto próprio Σ de V
- Um subconjunto finito P de $V^*(V \setminus \Sigma)V^* \times V^*$
- Um conjunto finito de probabilidades em que para cada produção é associada uma probabilidade de forma que a somatória de todas as probabilidade de produções que tem o mesmo lado esquerdo sempre resulte em 1
- Um elemento S de $V \setminus \Sigma$

O aprendizado computacional pode ser utilizado para gerarmos modelos a partir de um conjunto finito de elementos de uma linguagem. O modelo gerado nada mais é do que uma estimação da

linguagem sendo treinada e portanto uma linguagem também. Na Seção 2.5.2 vimos diversos tipos de linguagens que podem ser expressas por gramáticas. Tendo em vista a natureza estocástica do problema e os limites computacionais já apresentados na Seção 2.5.2, decidimos representar os classificadores usando gramáticas estocásticas livres de contexto e regulares.

2.5.4 Treinamento de Gramáticas Estocásticas

Para treinar as gramáticas descritas na Seção 2.5.3, são necessários dois passos principais: estimar quais são as produções da gramática e estimar a probabilidade das produções

Para estimar as produções de uma gramática, podemos considerar dois tipos principais de algoritmos: enumeração e construção. Enumeração consiste em enumerar todo o espaço de busca e aplicar cada elemento desse espaço aos dados de treinamento eliminando as gramáticas inconsistentes e escolhendo as restantes de acordo com algum critério. Tal abordagem se mostra inviável porque o espaço de busca é exponencial em relação ao tamanho dos exemplos, que são grandes. Construção consiste em construir a gramática desejada a partir das amostras num processo *botton-up*. Existem diversos algoritmos polinomiais para esse problema.

Tendo as produções sido estimadas, devemos estimar as probabilidades das produções. Para tal, temos que assumir que a amostra de treinamento tem a mesma distribuição de probabilidades da linguagem estocástica desconhecida, hipótese básica para um algoritmo ser **PAC**. Dessa forma, podemos utilizar a freqüência das cadeias ou partes de cadeias para estimar as probabilidades.

Existem diversos algoritmos para o treinamento de gramáticas, como o *inside-outside* [74] e o *Tree Grammar EM* [66], entre outros.

2.6 Outros Modelos

Existem diversos outros modelos que também são utilizados na área, como Redes Neurais, SVM (Support Vector Machines) e modelos baseados em *kernel*. Tais modelos são muito úteis em problemas em que não se conhece muita informação *a priori*, tais como determinação de fronteiras íntron/éxon e exón/íntron. Um outro exemplo é a determinação de estruturas secundárias e terciárias de proteína.

Ao longo deste trabalho não estudamos a fundo tais métodos. Foi dada ênfase principalmente em modelos sintáticos.

2.7 Relação entre os Modelos

A essência de qualquer algoritmo de aprendizado está na sua capacidade de:

- Generalizar o conjunto de exemplos fornecidos
- Permitir a introdução de informação *a priori* a respeito do problema
- Representar de forma eficiente o conceito aprendido

Existem diversos modelos como os apresentados, justamente porque cada um tem diferentes graus de facilidade para a generalização, inclusão e representação de informação dependendo do problema abordado.

Por exemplo, no caso de introdução de informação *a priori*, se sabemos que um grande número de éxons possui um determinado padrão conhecido e queremos introduzir esta informação, será mais fácil fazer isso se estivermos utilizando algoritmos baseados em **HMM** ou **EG** do que redes neurais.

Em contrapartida, se sabemos que uma determinada função que queremos aprender é crescente, este tipo de informação é mais facilmente introduzido utilizando-se o modelo ISI ou redes neurais.

De forma análoga, há limitações computacionais quanto ao tipo de conhecimento que pode ser introduzido em cada modelo. No caso de gramáticas, não é possível representar, utilizando-se gramáticas regulares, repetições do tipo xy, em que $x = y$ podendo x ser de tamanho qualquer. Em contrapartida, tal expressão é facilmente representável utilizando-se gramáticas livre de contexto.

No caso de representação, uma gramática representa mais facilmente famílias com seqüências com tamanho variável do que uma rede neural, por exemplo, enquanto que uma equação booleana é mais facilmente representada utilizando-se redes neurais.

Um aspecto mais sutil e importante é a capacidade de generalização que se obtém ao se escolher um modelo. Em geral, quando se utiliza um modelo, é esperado que tal modelo seja suficientemente bom para representar o fenômeno sendo estudado sem ser excessivamente custoso do ponto de vista computacional.

Cada modelo define uma forma com a qual uma superfície de separação entre classes é construída, em geral baseadas em medidas de erros e distâncias.

De um ponto de vista mais teórico, podemos ver qualquer algoritmo de aprendizado computacional supervisionado como *clustering* utilizando-se regras impostas pelo modelo escolhido seguido da definição de uma função de erro ou distância que se adapte bem ao modelo.

Por exemplo, no treinamento de uma rede neural, aparentemente não há *clustering*, mas, em sua essência, a definição da superfície de decisão que vai sendo construída conforme o algoritmo utilizado, nada mais é do que a definição de regiões (agrupamentos), ou seja, um *clustering*. Além disso, a superfície de decisão em si é a própria função de erro ou distância. Note que os pontos agrupados não são necessariamente os pontos dados como entrada, podem ser pontos produzidos a partir dos pontos de entrada como se faz numa redução de um problema a outro. Qualquer algoritmo de Aprendizado Computacional podem ser reduzido utilizando técnicas como essas.

Ao estudarmos *clustering*, estamos estudando também Aprendizado Supervisionado. Uma compreensão mais profunda de Aprendizado, principalmente supervisionado, só é alcançada estudando-se técnicas não supervisionadas. Tais estudos permitem entender e compreender melhor as relações entre as diversas áreas de reconhecimento de padrões.

HMM $\times$ EG

O modelo que apresentamos é equivalente à Gramáticas Estocásticas Regulares. A maioria dos artigos na área, prefere se referir a **HMM** ao invés de gramáticas regulares principalmente por questões de notação e bagagem estatística sólida do **HMM**.

Existem gramáticas mais poderosas, mas em geral a estimação é muito custosa, então em geral é necessário a realização da poda dos espaço de busca utilizando-se conhecimento *a priori*.

Muitos artigos se referem à **HMM** mas utilizam máquinas de estados numa forma "dualizada" o que dificulta a compreensão para quem não está acostumado.

3 Extração de Características

Durante o projeto, está sendo realizada a investigação de um novo método para se extrair características de seqüências de **DNA**. Tal método consiste de se observar uma seqüência dada através da freqüência de utilização de subpalavras de tamanho arbitrário.

Estamos abordando o problema de se encontrar genes humanos raros (ou seja, aqueles que raramente se expressam) através da utilização de técnicas de aprendizado computacional. Para tal, estamos avaliando o tamanho que tais subpalavras devem ter, bem como comparando diversos métodos que analisam as tabelas de freqüências obtidas.

Pelos nossos experimentos, acreditamos que palavras de tamanho 6 são de tamanho ideal, maximizando a quantidade de informação disponível que pode ser processada em tempo razoável, visto que a tabela de subpalavras tem tamanho exponencial em relação ao tamanho das mesmas.

Para a extração de características dessa tabela, estamos utilizando *Imagens CGR* e técnicas para a extração de dimensão fractal. Tais métodos são explicados nas próximas seções.

3.1 Imagens CGR

Na tentativa de obtermos alguma informação de assinatura gênica, estamos utilizando uma técnica de organização e visualização das tabelas de freqüências baseadas em "Chaos Game Representation" (**CGR**) [84].

CGR consiste em arranjar a tabela de freqüência numa matriz quadrada de forma recursiva. Ou seja, dada uma janela j de tamanho t, montaremos uma matriz m quadrada de tamanho $n \times n$ em que $n = 2^t$. Cada ponto desta matriz representa a freqüência de uma determinada subpalavra possível, ou seja, uma configuração possível de j.

Exemplos

No caso da janela ter tamanho 1, há somente 4 entradas na tabela de freqüências, o próprio alfabeto.

Tabela:

A	10
T	9
C	4
G	13

Matriz:

C	G
A	T

4	13
10	9

No caso da janela ter tamanho 2, há 16 entradas na tabela de freqüências:

AA	10
AT	9
AC	4
AG	13
TA	11
TT	15
TC	24
TG	17

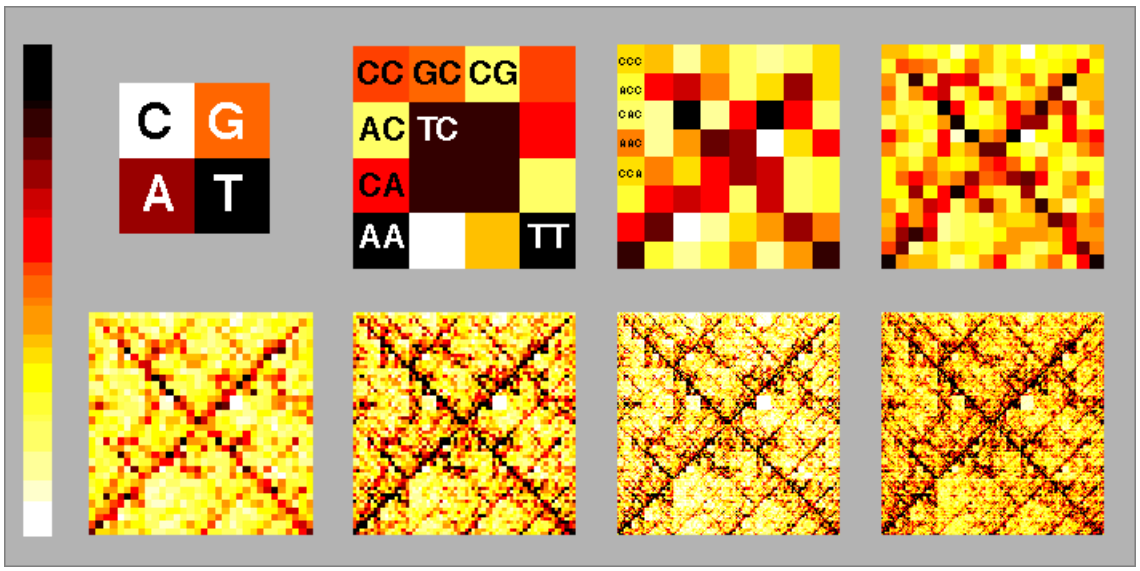
CA	35
CT	44
CC	0
CG	1
GA	22
GT	7
GC	90
GG	19

C	G
A	T

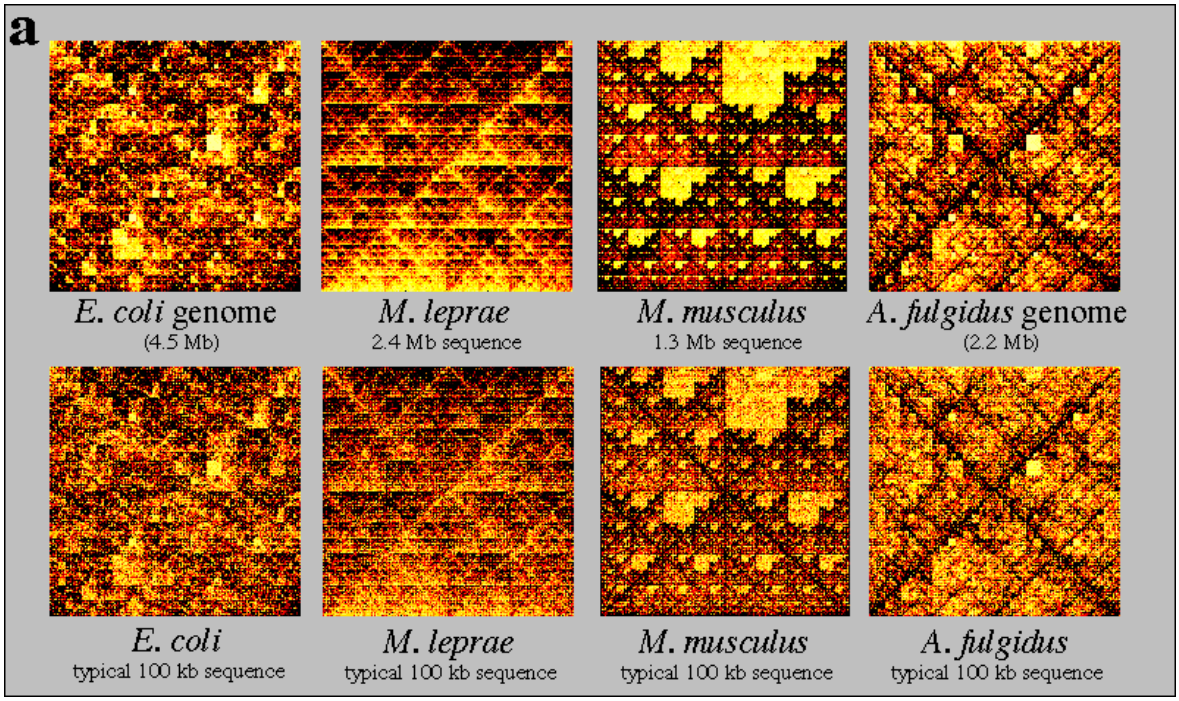
CC	GC	CG	GG
AC	TC	AG	TG
CA	GA	CT	GT
AA	TA	AT	TT

0	1	90	19
35	44	22	7
4	13	24	17
10	9	11	15

Ao invés de utilizarmos números, podemos utilizar tons para a visualização. No exemplo abaixo extraído de Deschavanne [26], temos janelas $t = 1 \ldots 8$ da bactéria A. *fulgidus*.



A seguir temos dois quadros comparativos com o cálculo da imagem **CGR** para 4 espécies. Em cima, o cálculo utilizando-se o cromossomo inteiro, embaixo, o cálculo utilizando-se uma pequena região do cromossomo.



Note que há diferenças significativas no padrão apresentado por cada espécie.

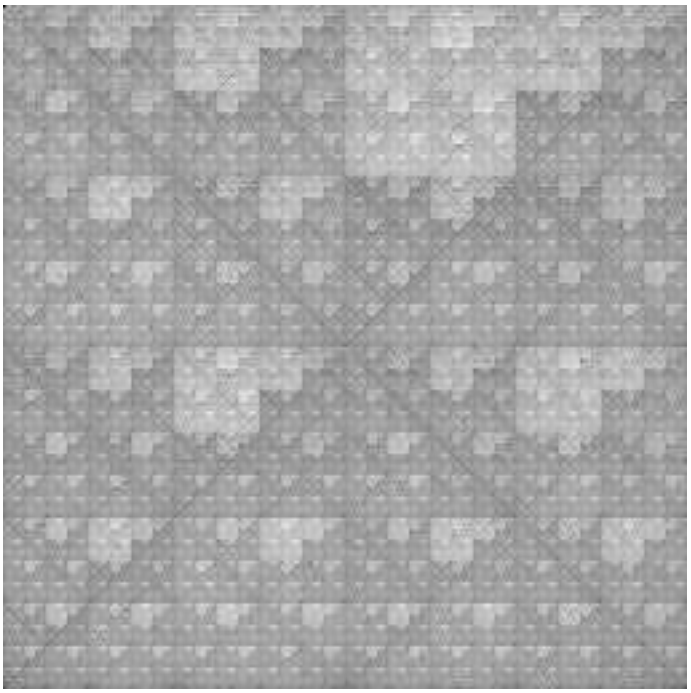
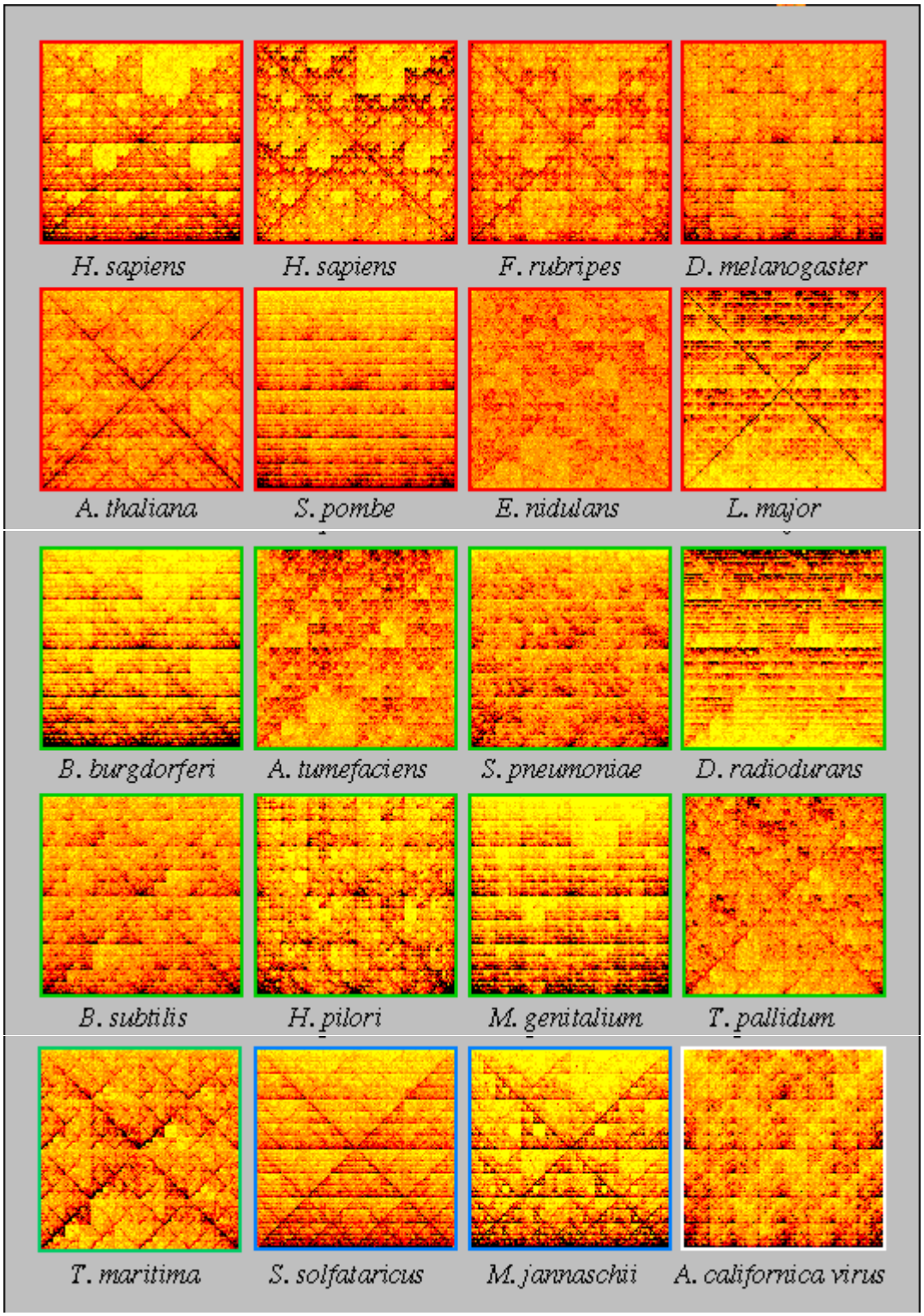


Figura 10: Imagem CGR do cromossomo 22 humano calculada para a janela de tramanho 8 e normalizada por log. A intensidade do preto é diretamente proporcional à freqüência de ocorrência de cada janela.

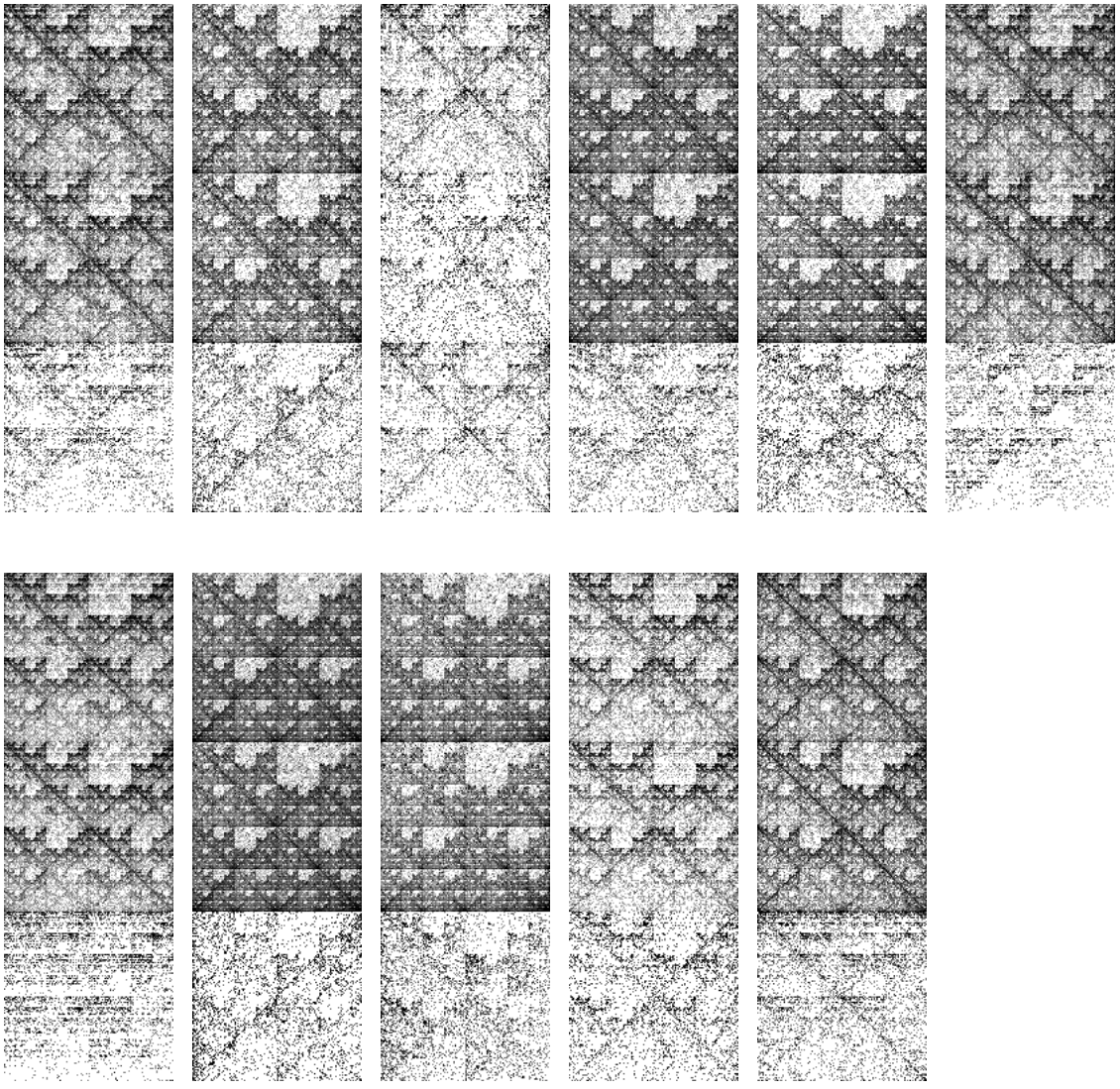
A seguir três quadros comparativos com as imagens **CGR** calculadas com o cromossomo de diversas espécies.



Deschavanne [26] observa que os padrões observados nas figuras são diferentes para cada espécies

e podem ser reproduzidos a partir de fragmentos pequenos do genoma de cada espécie. Além disso, é possível reconstruir ávores filogenéticas de forma bem razoável utilizando-se somente a imagem **CGR** do genoma de cada espécie.

Nosso objetivo é tentar extrair alguma informação útil a partir dessas tabelas de freqüência. A seguir, exemplos de imagens de genes do cromossomo 22 humano. Para cada imagem, em cima a imagem do gene inteiro, no meio só os íntrons e em baixo só os éxons:



Note que alguns genes são bem parecidos entre si e com a Figura 10 enquanto que outros são completamente diferentes. Além disso, há diferenças significativas entre as regiões de éxons e as regiões de íntrons.

3.2 Medida de Dimensão Fractal

Tendo em vista algumas propriedades visuais das imagens apresentadas, resolvemos avaliar imagens **CGR** utilizando técnicas de dimensão fractal.

O conceito de dimensão fractal fornece uma maneira de se medir a irregularidade dos objetos. Em geral é considerado que uma linha reta possui uma dimensão, uma superfície plana possui duas dimensões, um ponto possui dimensão zero e um sólido possui três dimensões.

Podemos definir a dimensão fractal como a capacidade que uma curva tem de preencher um determinado espaço. Ou seja, quanto menor a dimensão fractal de uma curva, maior a liberdade com a qual se consegue preencher o espaço de forma regular em todas as dimensões.

Estamos utilizando um método proposto por Luciano da Fontoura Costa [23] para estimar a dimensão fractal de um objeto através da utilização de técnicas multiescala.

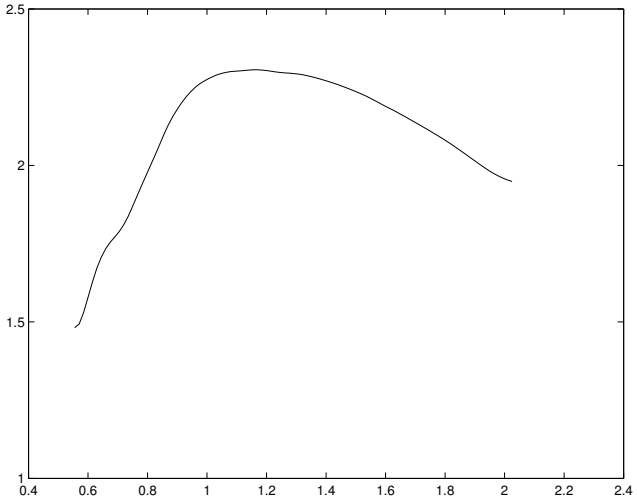


Figura 11: Curva de fratalidade da Figura 10

Tais técnicas constituem basicamente na determinação da liberdade com que se consegue fazer uma dilatação do objeto com um disco de raio r. Ao se variar o raio, obtemos uma curva que representa a área obtida para cada dilatação, ao calcularmos a derivada dessa curva, temos uma representação da liberdade com a qual se conseguiu crescer. Na Figura 11 temos um exemplo de tal curva calculada a partir da Figura 10. Observe que o comportamento fractal diminui em ambas as extremidades do gráfico, o que é natural, afinal as extremidades representam escalas muito grandes ou pequenas. No centro do gráfico podemos observar o comportamento fractal da imagem.

4 Resultados Preliminares

Ao longo do trabalho, realizamos algumas implementações, bem como experimentos no sentido de avaliar a possibilidade de se criar uma nova forma de se extrair características de um gene baseado em técnicas descritas na Seção 3.

Foram realizados diversas especificações, implementações e testes preliminares que serão descritos nas próximas seções.

Acreditamos que o ponto mais forte da nossa especificação deve ser a forma com que os dados são armazenados e as relações entre os mesmos. Ou seja, estamos especificando um ambiente que seja flexível o suficiente para que se possa abordar qualquer problema na área sem a necessidade de se reescrever o ambiente, bastando extendê-lo de forma simples. Além disso, o modelo deve ser maduro o suficiente para acompanhar as mudanças na área que são muito rápidas, tanto em relação aos dados disponíveis quanto em relação aos tipos de dados disponíveis, ou seja, o modelo deve ser de fácil atualização ou até mesmo com mecanismos de atualização automática dos dados quando for o caso.

Como motivação e forma de orientação, escolhemos o problema clássico de localização de genes devido à sofisticação que tal problema alcança na área. Na tentativa de resolver este problema, diversas técnicas de Aprendizado Computacional são utilizadas bem como técnicas híbridas, o que nos faz acreditar que ele seja um excelente problema a ser analisado. Aqui é importante observar que o objetivo principal do projeto não é achar genes ou entender como funciona um determinado algoritmo para tal tarefa, mas sim entender as técnicas e ferramentas principais que sustentam os algoritmos da área.

4.1 Especificação e implementações do Ambiente

Ao longo do trabalho, realizamos três especificações e implementações principais:

- Gerador estocástico de palavras.

  Durante sua especificação, levamos em conta os padrões de gramáticas estocásticas disponíveis, bem como as necessidades do nosso grupo, visto que tal gerador será usado em testes dos algoritmos que estão sendo desenvolvidos pelo nosso grupo e por este projeto. O pograma lê um conjunto de gramáticas estocásticas, alguns parâmetros de entrada, como o

número de seqüências de saída, tamanho máximo, formato da saída, entre outros e a saída do programa é simplesmente o conjunto de seqüências desejado.

- Ambiente de trabalho.

  Tendo em vista as necessidades já citadas, optamos por construí-lo baseado no sistema de arquivos do Linux e a utilização de *Makefiles* e *scripts* para sua atualização. Tal forma de armazenamento facilita o desenvolvimento rápido de filtros que permitem a utilização de tudo quanto é tipo de programa disponível atualmente, mesmo que rodem em sistemas operacionais distintos. Além disso os dados estão rapidamente disponíveis, visto que estão no HD do micro de testes.

  Dividimos este ambiente em quatro módulos principais:

  Dados de entrada

  Este módulo é o que consideramos o mais importante. Como base deste módulo temos os dados brutos. Tais dados são filtrados de forma a extrair somente as informações que nos interessam. Não só isso, muitas vezes filtramos dados de diversas fontes de forma que eles sejam unificados. Com esses dados unificados e de formato simples em mãos, fica facilitada a confecção de scripts que convertam tais dados para o formato de entrada de algoritmos específicos. Não só isso, fica facilitada a seleção dos dados que se quer utilizar.

  Este módulo é desenvolvido todo baseado em *Makefiles* e filtros de forma que qualquer alteração nos dados brutos (como por exemplo, a alteração dos arquivos do genoma humano no **NCBI**) possa facilmente ser incorparada, bastando fazer download dos novos dados e rodar o utilitário *make*, ações que podem até mesmo serem colocadas num script automático. Resumindo este módulo foi projetado para ser de fácil atualização, modificação e extensão.

  Dados de teste

  Neste módulo guardamos uma série de filtros necessários para a utilização dos diversos programas. A função de tais filtros é gerar a partir dos dados de entrada filtrados, conjuntos de dados de testes prontos para serem rodados pelos programas a serem utilizados.

  Outro objetivo deste módulo é também manter um histórico dos experimentos realizados.

Programas

Neste módulo mantemos todos os programas utilizados no projeto, tanto os desenvolvidos por nós como os disponibilizados por terceiros. Serve simplesmente como um índice das opções disponíveis.

Relatórios

Neste último módulo mantemos relatórios sobre tudo o que estamos fazendo, desde os experimentos realizados até os principais links acessados por nós.

- Testes.

  Realizamos uma especificação de como nossos testes devem ser, ou seja, como vamos testar os algoritmos e técnicas que estamos avaliando. Esta especificação ainda não foi concluída.

Como estamos abordando o problema de predição de genes, estamos utilizando dados do **NCBI** [137], **Ensembl** [133], Genome Project Working Draft [135] e dados fornecidos pelo nosso co-orientador para gerar dois tipos de dados principais:

1. Um mapa completo do genoma humano (respeitando as limitações atuais quanto à completude do genoma humano) em que a posição de genes reais e preditos bem como outros elementos são assinalados.

2. Um diretório para cada gene conhecido contendo diversas informações sobre o gene, como a posição dos seus éxons, tabelas de freqüências etc.

3. Mapas extras dos cromossomos e contigs contendo informações como áreas mascaradas, de freqüência alterada, entre outros ítens.

Tais dados serão muito úteis para os nossos experimentos, principalmente porque estão num formato muito simples apesar de ocuparem bastante espaço em relação às outras representações utilizadas.

Também implementamos parcialmente os filtros para a geração dos dados de testes.

Uma informação curiosa a respeito desta implemantação é que foi uma das partes mais difíceis do projeto até agora, não pela dificuldade para se escrever os filtros e *Makefiles*, mas sim pela

dificuldade de encontrar os dados que precisamos. Por exemplo, a localização dos genes num cromossomo, é uma informação muito importante para nós, mas tal informação é difícil de se obter através dos *sites* do **NCBI**, entre outros. Só conseguimos extrair tal informação ao pegarmos os arquivos disponíveis no ftp dos *sites* e filtrarmos os mesmos. Outra informação difícil de se extrair é se um gene foi predito por algum programa ou se o gene está lá porque ele alinha razoavelmente bem com alguma proteína ou mRNA conhecidos. Outro problema é em relação à nomenclatura dos genes, em cada site que você vai muitas vezes o mesmo gene aparece com nomes distintos. São questões simples mas que nos tomaram diversos meses, muito mais do que o esperado.

Infelizmente a área de Biologia Computacional ainda tem muito a evoluir no sentido de armazenamento de informações, mas acreditamos que conseguimos contornar esses problemas através dos nossos filtros após passar um bom tempo estudando o problema. Além disso, o **Ensembl** acabou de lançar uma nova versão de seu banco de dados que, através de uma excelente interface escrita na linguagem **perl**, facilita a obtenção de diversas informações que precisamos para o nosso projeto.

Implementamos também o algoritmo descrito em [26]. Fomos capazes de reproduzir completamente o artigo e realmente os resultados obtidos foram muito interessantes. Criamos um programa que gera tanto as imagens descritas, como as tabelas brutas necessárias para a realização de alguns experimentos.

Também foi realizado uma implementação incrementada de árvores de sufixo que economiza memória proposta por Kurtz [65] . Acreditamos que tal implementação eventualmente seja útil para alguns experimentos que poderão ser realizados, como os que eventualmente usem o cálculo de tabelas de freqüência de palavras grandes.

Outra implementação importante foi a implementação de uma versão multi-escala e multi-resolução do **CGR**.

4.2 Resultados

Ao longo do trabalho realizamos dois conjuntos de experimentos principais que servirão para decidir os rumos dos próximos experimentos. Ambos os experimentos foram baseadas em técnicas **CGR**, só que com metodologias distintas. No primeiro conjunto de experimentos exploramos técnicas multi-resoluções. Já no segundo conjunto, foi utilizada uma técnica baseada em dimensão fractal que tem a propriedade de explorar a organização de imagens. Tal informação poderia ser

utilizada como parte de algoritmos baseados nas técnicas descritas na Seção 2.

Experimentos baseados em CGR multi-resolução

Após implementarmos o **CGR** multi-resolução e multi-escala, realizamos diversos experimentos, variando inclusive diversos tipos de janela. A idéia de multi-resolução e multi-escala, é tentar variar o nível de detalhamento com que se observa um objeto. No caso de imagens, podemos definir informalmente a função de nível de detalhamento simplesmente como sendo um "zoom". Ou seja, se queremos pouco detalhe, dando ênfase para as estruturas principais, podemos olhar com um zoom pequeno. No caso de um mapa mundi, seria equivalente a olhar somente as águas e continentes. Caso se queira mais detalhe, ajusta-se a função zoom, ou seja, é possível aplicar um zoom maior num pequeno pedaço do continente para que se possa estudar melhor as propriedades de um determinado objeto que não podem ser determinadas avaliando-se somente o zoom menor. A idéia de multi-resolução é tentar reduzir custos computacionais, buscando primeiro no zoom menor e só depois no zoom maior. Funcionaria como um filtro, que podaria buscas desnecessárias que precisariam ser realizadas num mapa somente com zoom maior.

Chegamos à conclusão que tanto multi-escala quanto multi-resolução como se faz em imagens (em geral com a função zoom) não se aplica muito bem em seqüências de DNA diretamente. Tal fato se deve principalmente porque não é claro para uma seqüência, o que é "mais detalhe" e o que é "menos detalhe". Provavelmente, no caso de seqüências, esse conceito só faz sentido num ambiente tridimensional, ou seja, no zoom menor, teríamos um cromossomo em 3D "visto de longe", conforme vamos aumentando o zoom, ou seja, o nível de detalhe, surgem as primeiras dobras e estruturas tridimensionais, aumentando-se ainda mais o nível de detalhe, surgem novas estruturas tridimensionais que vão sendo responsáveis por guardar informações, por exemplo. Aumentando ainda mais o zoom, chega-se ao código de DNA em si.

O problema é que para calcular essa função de detalhamento, teria-se que conhecer as estruturas tridimensionais das seqüências. Só que essa informação está longe de estar disponível, o que inviabiliza uma abordagem multi-resolução do ponto de vista mais purista como descrito aqui.

O que tentamos fazer, foi uma abordagem simplista, simplesmente cortando pedaços de DNA, como um zoom cortaria uma imagem (ou algo também parecido com os **HMM** que utilizam histórico de tamanho variável), mas não foram obtidos resultados animadores. Um fato no entanto

curioso, é que apesar dos cortes serem suficiente para deformar a estrutura da imagem, tal fato não acontece, o que talvez esteja sugerindo que a estrutura baseada em freqüência de bases é uma assinatura razoavelmente confiável da espécie.

Tendo em vista um resultado não satisfatório neste experimentos, optamos por buscar técnicas que utilizam elementos mais concretos.

Uma visão multi-resolução interessante baseada em **HMM** é a proposta por David Kulp [64]. Ele procura dividir os cromossomo em diversas regiões e tenta localizar essas regiões usando três níveis de detalhamento. Talvez essa abordagem seja a mais razoável dadas as limitações computacionais existentes atualmente.

Experimentos baseados em CGR e dimensão fractal

Realizamos então uma série de 25 experimentos de *clustering* na tentativa de avaliar se a medida fractal como descrita na Seção 3.2 é eficiente para se procurar genes. Aparentemente os primeiros resultados são animadores.

Para efeito comparativo, comparamos 5 tipos de características:

- Imagem CGR da seqüência
- Imagem CGR da seqüência normalizada usando logaritmo
- Curva de dimensão fractal da imagem CGR da seqüência
- Curva de dimensão fractal da imagem CGR da seqüência normalizada usando logaritmo
- Conteúdo GC puro (equivalente à calcular a imagem CGR com janela de tamanho 1)

Foram extraídas as características acima de 526 genes do cromossomo 22 (seqüência completa) e de 655 regiões contínuas do mesmo cromossomo em que se acredita com forte confiança que não há genes ou pedaços de genes nas mesmas.

A idéia do experimento foi tentar separar tais regiões utilizando-se somente uma única característica por vez, dentre as descrita acima.

Para cada uma das características, foi calculada uma matriz de distâncias e foram utilizados os 5 algoritmo hierárquicos disponíveis no Matlab (totalizando 25 experimentos) para se tentar

realizar a separação. Importante observar que para calcular a matriz de distância, foram utilizados os seguinte métodos:

Para o conteúdo GC e as imagens CGR normalizadas linearmente e por log simplesmente realiza-se-se a somatória do módulo da subtração ponto a ponto. Esse é o valor da distância. Para as curvas baseadas em dimensão fractal, calcula-se a soma da integral das curvas e subtraí-se duas vezes o valor da intersecção entre as duas integrais. Não foram usadas nenhuma informação adcional que é disponibilizada pela técnica de dimensão fractal.

Cada um dentre os 25 experimentos consistiu no cálculo de 1180 *clusters* usando o mesmo método com número de agrupamentos variando de 2 a n.

Foi calculado então o "erro" de cada cluster da seguinte forma: "Se para cada agrupamento do cluster for necessário associar um único rótulo, qual o erro que será obtido no final?". Atribui-se então o rótulo "gene" à um agrupamento se no mesmo há mais fragmentos que são genes ou rotula-se como "não-gene" caso contrário. O erro é simplesmente o número de elementos de cada agrupamento cujo rótulo é diferente do rótulo do agrupamento, dividindo-se o total pelo número de pontos que no caso é 1181. Obtém-se assim a taxa de erro.

Foram calculados também o número de falsos positivos (FP - Atribuição de rótulo de gene para seqüências que não são gene) e o número de falsos negativos (FN - Atribuição de rótulo de não gene para seqüência que são genes).

Na Figura 12 temos um gráfico somente com as taxas de acerto calculadas para as cinco medidas apresentadas anteriormente utilizando-se o método de *clustering* Complete. O eixo das abcissas representa a porcentagem do número de agrupamento de cada *cluster* em relação ao número total de fragmentos. O eixo das ordenadas representa a taxa de acerto.

Interessante observar que independentemente do método de *cluster* utilizado, as curvas obtidas são bem semelhantes às apresentadas na Figura 12. Em todos os experimentos, a curva azul claro (correspondente ao método baseado em dimensão fractal com normalização pela função log) dá resultados bem interessantes e ligeiramente superiores às outras curvas, principalmente em relação à azul escuro, que serve de controle.

Importante observar que só faz sentido comparar as curvas nos primeiros $10 - 20\%$, visto que para clusters com número de agrupamentos maior do que 20% do número total de pontos ocorre o problema de overffiting.

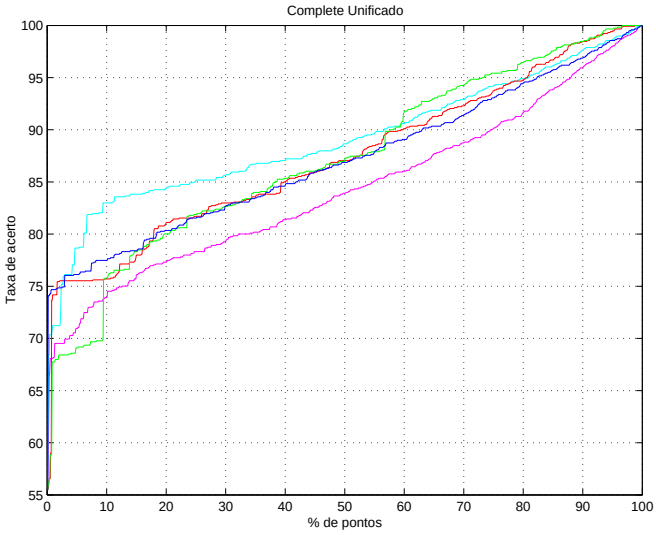


Figura 12: Resumo dos resultados obtidos com o método Complete. Imagem CGR em vermelho, imagem CGR normalizada por log em verde, dimensão fractal em roxo, dimensão fractal normalizada por log em azul claro, conteúdo GC em azul escuro.

A medida baseada em dimensão fractal normalizada por log parece ter dado resultados bem interessantes, enquanto que a mesma medida com normalização linear foi a pior de todas.

Os resultados parecem ser interessantes, porque não utilizam nenhuma informação biológica adcional e, apesar da separação obtida não ser muito boa, ela é significativamente superior à técnica baseada em conteúdo GC que são muito utilizadas por algoritmos baseadas em **HMM**, o que sugere que talvez tal medida possa ser utilizada em substituição ou complemento ao conteúdo GC nesses algoritmos.

De qualquer forma, esses ainda são resultados preliminares. É necessário ainda uma série de experimentos para validarem essa medida. É importante observar que o experimento é realizado de forma simples, e equivale a treinar e aplicar o classificador nos próprios dados de treinamento, o que em geral não tem muito significado.

5 Próximas Etapas

Nesta seção pretendemos apresentar uma breve proposta para os próximos experimentos baseados nos experimentos anteriores. Pretendemos dividir os experimentos em três categorias:

1. Como visto na Seção 4.2, foram realizados alguns experimentos preliminares com técnicas baseadas em dimensão fractal. Pretendemos fazer uma série de experimentos com o objetivo de tentar validar a técnica. Para tal, pretendemos realizar mais alguns experimentos com outros dados e também utilizar alguns métodos como *bootstrap*. Dependendo dos resultados obtidos, poderemos optar por realizar o mesmo tipo de experimento com outros cromossomos humanos.

   No caso da técnica se mostrar positiva, poderemos abordar também alguns problemas técnicos, como a normalização das imagens **CGR**.

   Caso a técnica seja validada, ela poderá ser utilizada como substituição ou complemento à técnicas baseadas em conteúdo GC que são largamente utilizadas como parte de softwares para predição de genes.

2. A abordagem acima necessita da definição de uma forma de construir um classificador. Atualmente estamos desenvolvendo algumas abordagens na tentativa de resolver tal questão. A nossa idéia mais básica é simplesmente utilizar como classificador o *cluster* obtido. Para a classificação de um novo ponto, poderia-se atribuir o rótulo do agrupamento menos distante ao ponto em questão, por exeplo.

3. Poderemos realizar também experimentos com adaptações da abordagem acima para outros subproblemas, como discriminação das regiões intergênicas, ou mesmo determinação de fronteiras íntron/éxon, éxon/íntron.

6 Estrutura da Dissertação

Pretendemos apresentar uma dissertação estruturada em duas partes principal. A primeira contendo uma revisão bibliográfica da área com os principais modelos e técnicas utilizados.

Pretendemos apresentar uma introdução em reconhecimento de padrões supervisionado e não supervisionado seguido dos principais modelos como cadeis de Markov, **HMM**, Gramáticas Estocásticas, Redes Neurais entre outros.

Logo em seguida haverá uma seção comparando os diversos modelos.

Na segunda parte, pretendemos apresentar os diversos resultados dos experimentos que virão a ser realizados. Explicaremos em detalhe a metodologia aplicada e cada um dos resultados obtidos.

Referências

[1] M. D. Adams, C. Fields, and J. C. Venter, editors. *Automated DNA Sequencing and Analysis*. Academic Press, 1994.

[2] S. S. Adi. Ferramentas de Auxilio ao Sequenciamento de DNA por Montagem de Fragmentos: um estudo comparativo. Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, 2000. http://www.ime.usp.br/∼said/prjdiss.ps [8/Jan/2001].

[3] F. Alizadeh, K. Karp, L. Newberg, and D. Weisser. Physical mapping of chromosome: A combinatorial problem in molecular biology. In *the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, ACM Press, pages 371–381, 1993. http://citeseer.nj.nec.com/alizadeh93physical.html [21/Feb/2001].

[4] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. In *Nucleic Acids Research*, volume 25, pages 3389–3402, 1997. http://nar.oupjournals.org/cgi/content/full/25/17/3389 [31/Jan/2001].

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[6] D. Angluin. Computational Learning Theory: Survey and Selected Bibliography. In *Proceedings of the twenty-fourth annual ACM Symposium on Theory of Computing*, pages 351–369, 1992.

[7] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Research*, 28(1):45–48, 2000.

[8] P. Baldi and S. Brunak. *Bioinformatics: The Machine Learning Approach (Adaptive Computation and Machine Learning)*. MIT press, 1998.

[9] S. Batzoglou, L. Pachter, J. Mesirov, B. Berger, and E. Lander. Human and mouse gene structure: Comparative analysis and application to exon prediction. *Genome Research*, 10:950–958, 2000.

[10] A. D. Baxevanis. The molecular biology database collection: 2002 update. *Nucleic Acids Research*, 30(2):1–12, 2002.

[11] A. D. Baxevanis and B. F. Ouellette, editors. *Bioinformatics: A Pratical Guide to the Analysis of Genes and Proteins*. Wiley-Interscience, 1998.

[12] G. Bejerano and G. Yona. Modeling protein families using probabilistic suffix trees. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 15–24, Lyon, France, 1999. ACM Press.

[13] M. J. Bishop and C. J. Rawlings, editors. *DNA and protein sequence analysis: a pratical approach*. The Pratical Approach Series. Oxford University Press, 1st edition, 1997.

[14] E. Bolten, A. Schliep, S. Schneckener, D. Schomburg, and R. Schrader. Clustering protein sequences - structure prediction by transitive homology, 2000. ftp://ftp.zpr.uni-koeln.de/pub/paper/pc/zpr2000-383.zip [16/Fev/2001].

[15] J. Bondy and U. Murty. *Graph Theory with Applications*. MacMillan, London, 1976.

[16] M. Bot and W. Langdon. Application of genetic programming to induction of linear classication trees. In *European Conference on Genetic Programming EuroGP2000*, pages 247–258, Apri 2000. http://www.cs.vu.nl/∼mbot/mijnpapers/euroGP2000/paper.ps [6/Fev/2001].

[17] P. Bremaud, editor. *Markov Chains*. Springer Verlag, 1999.

[18] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

[19] A. Campbell, J. Mrazek, and S. Karlin. Genome signature comparisons among prokaryote, plasmid, and mitochondrial DNA. In *Proceedings of the National Academy of Sciences*, volume 96, pages 184–9189, 1999.

[20] T. E. Campos. Técnicas de seleção de características com aplicações em reconhecimento de faces. Master's thesis, Instituto de Matemática e Estatística – Universidade de São Paulo, SP - Brasil, maio 2001.

[21] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

[22] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2nd edition, 1998.

[23] L. F. Costa and A. G. C. Blanchi. A outra da dimensão fractal. *Ciência Hoje*, 31(183):40–47, 2002.

[24] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[25] R. E. Davis. Introduction to bioinformatics and genomics, 2002. http://www.library.csi.cuny.edu/∼davis/Bioinfo_326/ [29/Jun/2002].

[26] P. J. Deschavanne, A. Giron, J. Vilain, G. Fagot, and B. Fertil. Genomic signature: Characterization and classification of species assessed by chaos game representation of sequences. *Mol. Biol. Evol.*, 16(10):1391–1399, 1999.

[27] L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic Theory of Pattern Recognition*. Springer, 1996.

[28] R. Diestel. *Graph Theory*. Springer, 2nd edition, 2000.

[29] E. R. Dougherty, J. Barrera, M. Brun, S. Kim, R. M. Cesar, Y. Chen, M. Bittner, and J. M. Trent. Inference from clustering with application to gene-expression microarrays. *Journal of Computational Biology*, 9(1):105–126, 2002.

[30] D.Sankoff and J. Kruskal. *An overview of sequence comparison, Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Massachusetts: Addison-Wesley, 1983.

[31] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley and Sons, 2000.

[32] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of protein and nucleic acids*. Cambridge University Press, 2000.

[33] S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14(9), 1998.

[34] A. J. Enright and C. A. Ouzounis. Generage: a robust algorithm for sequence clustering and domain detection. *Bioinformatics*, 16(5):451–457, 2000. http://bioinformatics.oupjournals.org/cgi/reprint/16/5/451 [6/Fev/2001].

[35] A. B. et al. Plasmodb: the plasmodium genome resource. an integrated database providing tools for accessing, analyzing and mapping expression and sequence data (both finished and unfinished). *Nucleic Acids Research*, 30(1):87–90, 2002.

[36] D. A. B. et al. Genbank. *Nucleic Acids Research*, 30(2):17–20, 2002.

[37] D. L. W. et al. Database resources of the national center for biothecnology information: 2002 update. *Nucleic Acids Research*, 30(2):13–16, 2002.

[38] M. Farach-Colton, F. S. Roberts, M. Vingron, and M. Waterman, editors. *Mathematical Support for Molecular Biology*, volume 47 of *DIMACS series in discrete mathematics and theorical computer science*. America Mathematical Society, 1999.

[39] D. Fasulo. An analysis of recent work on clustering algorithms, April 1999. http://www.cs.washington-.edu/homes/dfasulo/clustering.ps [26/Jan/2001].

[40] I. Felger, V. M. Marshal, J. C. Reeder, J. A. Hunt, C. S. Mgone, and H.-P. Beck. Sequence diversity and molecular evolution of the merosoite surface antigen 2 of plasmodium falciparum. *Journal of Molecular Evolution*, 45:154–160, 1997.

[41] R. S. Feris. Rastreamento eficiente de faces em um espaço wavelet. Master's thesis, Intituto de Matemática e Estatística – Universidade de São Paulo, SP - Brasil, maio 2001.

[42] G. S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer Series in Operations Research. Springer Verlag, 1996.

[43] C. Frontali. Genome plasticity in plasmodium. *Genetica*, pages 91–100, 1994.

[44] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1982.

[45] C. Gibas and P. Jambeck. *Desenvolvendo bioinformática: ferramentas de software para aplicações em biologia*. Campus – O'Reilly, 2001. Tradução de: Developing bioinformatics computer skills.

[46] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[47] G. R. Grant and W. J. Ewens. *Statistical Methods in Bioinformatics: An Introduction*. Springer Verlag, 2001.

[48] D. Gusfield. *Algorithms on strings, trees, and sequences. Computer Science and Computational Biology*. Cambridge University Press, 1999.

[49] B. K. Hall, editor. *Homology: The Hierarchical Basis of Comparative Biology*. Academic Press, 1994.

[50] M. H. Hassoun. *Fundamentals of ARTIFICIAL NEURAL NETWORKS*. MIT Press, 1995.

[51] D. Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. *Information and Computatuion*, 100:78–150, 1992.

[52] J. Henderson, S. Salzberg, and K. Fasman. Finding genes in DNA with a Hidden Markov Model. *Journal of Computational Biology*, 4(2):121–141, 1997. http://citeseer.nj.nec.com/179996.html [21/Feb/2001].

[53] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci USA*, 89:10915–10919, 1992.

[54] D. Higgins and P. Sharpe. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. In *Gene*, volume 73, pages 237–244, 1988.

[55] R. Hughey and A. Krogh. Hidden markov models for sequence analysis: extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996.

[56] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[57] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3), September 1999.

[58] B. R. Jasny and D. Kennedy. The human genome. *Science*, 291(5507), February 2001. http://www.sciencemag.org/genome2001/1153.html [16/Fev/2001].

[59] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1997.

[60] J. Kim, L. Ohno-Machado, and I. Kohane. Unsupervised learning from complex data: The matrix incision tree algorithm. In *Pacific Symposium on Biocomputing*, volume 6, pages 30–41, 2001. http://www.smi.stanford.edu/projects/helix/psb01/kim.pdf [20/Jan/2001].

[61] S. Knudsen. Promoter2.0: for the recognition of polii promoter sequences. *Bioinformatics*, 15(5):356–361, 1999.

[62] T. Koski and T. Koskinen. *Hidden Markov Models for Bioinformatics*. Kluwer Academic Publishers, 2001.

[63] A. Krogh, I. Mian, and D. Haussler. A Hidden Markov Model that finds genes in E. coli DNA. *Nucleic Acids Research*, 22:4768–4778, 1994.

[64] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A Generalized Hidden Markov Model for the recognition of human genes in DNA. In *Fourth International Conference on Intelligent Systems for Molecular Biology*, 1996.

[65] S. Kurtz. Reducing the space requirement of suffix trees. *Software – Practice and Experience*, 29(13):1149–1171, 1999.

[66] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[67] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 2nd edition, 1998.

[68] W. Li. A bibliography on computational gene recognition. http://linkage.rockefeller.edu/wli/gene/[29/Ago/2002].

[69] D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, pages 1435–1441, 1985.

[70] A. V. Lukashin and M. Borodovsky. Genemark.hmm: new solutions for gene finding. *Nucleic Acids Research*, 26(4):1107–1115, 1998.

[71] Q. Ma and J. T. L. Wang. Application of bayesian neural networks to biological data mining: A case study in dna sequence classification. In *Twelfth International Conference on Software Engineering and Knowledge Engineering*, pages 23–30, 2000.

[72] J. Meidanis and J. C. Setubal. *Introduction to Computational Molecular Biology*. PWS Publishing Co., 1997.

[73] E. Mendelson. *Álgebra Booleana e Circuitos de Chaveamento*. Coleção Schaum. McGraw-Hill, 1977.

[74] Y. S. Michael. The application of stochastic context-free grammars to folding, aligning and modeling homologous rna sequences, 1993. ftp://ftp.cse.ucsc.edu/pub/tr/ucsc-crl-94-14.ps.Z [21/Feb/2001].

[75] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3rd rev. and extended edition, 1999.

[76] T. Minka. A statistical learning/pattern recognition glossary. http://www-white.media.mit.edu/∼tpminka/statlearn/glossary/ [20/May/2002].

[77] A. A. Mironov, J. W. Fickett, and M. S. Gelfand. Frequent alternative splicing of human genes. *Genome Research*, 9:1288–1293, 1999.

[78] B. Morgenstern. Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999.

[79] B. Morgenstern, A. Dress, and T. Werner. Multiple dna and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA*, 93:12098–12103, 1996.

[80] D. W. Mount. *Bioinformatics: Sequence and Genome Analysis*. CSHL Press, 1st edition, 2001.

[81] M. Moura. O novo produto brasileiro. *Pesquisa FAPESP*, 55:8–15, julho 2000. http://www.fapesp.br/-capa551.htm [4/Set/2000].

[82] A. M. L. Oliveira. Laboratório de geração de classificadores de seqüências. Master's thesis, Instituto de Matemática e Estatística – Universidade de São Paulo, SP - Brasil, julho 2002.

[83] T. Oliveira and A. Brunstein. HIV sequence analysis and bioinformatics course, November 2001. http://www.vision.ime.usp.br/∼tulio/ [29/Jun/2002].

[84] J. L. Oliver, P. Bernaola-Galván, J. Guerrero-García, and R. Román-Roldán. Entropic profile of DNA sequences trought chaos-game-derived images. *Journal of Theorical Biology*, 160:457–470, 1993.

[85] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994. Reprinted August, 1995.

[86] C. H. Papadimitrou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[87] W. Pearson and D. Lipman. Improved tools for biological sequence comparison. In *Proceedings of National Academy of Sciences of the USA*, volume 85, pages 2444–2448, 1988.

[88] W. R. Pearson. Rapid and sensitive sequence comparisions with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990.

[89] M. Pertea, X. Lin, and S. Salzberg. Genesplicer: a new computational method for splice site prediction. *Nucleic Acids Research*, 29(5):1185–1190, 2001.

[90] E. Pizza, S. Liuni, and C. Frontali. Detection of latent sequence periodicities. *Nucleic Acids Research*, 18(13):3745–3752, 1990.

[91] E. Pizzi and C. Frontali. Low-complexity regions in plasmodium falciparum proteins. *Genome Research*, 11:218–229, 2001.

[92] W. K. Pratt. *Digital Image Processing*. Wiley-Interscience, 1991.

[93] B. Prum. Markov models and hidden markov models in genome analysis. 6th Brazilian School of Probability, Praia das Tininhas - Ubatuba, São Paulo, August 5–10 2002.

[94] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77(2), pages 257–284, 1989.

[95] S. Rich, M. Ferreira, and F. Ayala. The origin of antigenic diversity in plasmodium falciparum. *Parasitology Today*, 16(9):390–396, 2000.

[96] S. M. Rich, R. R. Hudson, and F. J. Ayala. Plasmodium falciparum antigenic diversity: Evidence of clonal population structure. *Proc. Natl. Acad. Sci. USA*, 94:13040–13045, 1997.

[97] F. Richards. The protein folding problem. *Scientific American*, 264(1):54–63, January 1991.

[98] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjolander, R. C. Underwood, and D. Haussler. Recents Methods for RNA Modeling Using Stochastic Context-Free Grammars. In *Combinatorial Pattern Matching, 5th Annual Symposium*, pages 289–306, 1994. ftp://ftp.cse.ucsc.edu/pub/rna/cpm94.ps.Z [21/Feb/2001].

[99] Y. Sakakibara, M. Brown, R. Underwood, I. S. Mian, and D. Haussler. Stochastic context-free grammars for modeling RNA. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, pages 284–283, Honolulu, 1994. IEEE Computer Society Press.

[100] M. K. Sakharkar, T. W. Tan, and S. J. de Souza. Generation of a database containing discordant intron positions in eukaryotic genes (midb). *Bioinformatics*, 17(8):671–675, 2001.

[101] S. Salzberg, A. Delcher, K. Fasman, and J. Henderson. A decision tree system for finding genes in DNA. *Journal of Computational Biology*, 5(4), 1998. http://www.tigr.org/∼salzberg/morgan.ps.gz [21/Feb/2001].

[102] R. Sandberg, G. Winberg, C.-I. Bränden, A. Kaske, I. Ernberg, and JoakimCöster. Capturing whole-genome characteristics in short sequences using a naïve bayesian classifier. *Genome Research*, 11:1404–1409, 2001.

[103] J. R. Searle. *Minds, Brains and Science*. Harvard University, March 1986.

[104] D. Searls. String variable grammar: a logic grammar formalism for the biological language of DNA. *The Journal of Logic Programming*, 12:1–30, 1993. http://citeseer.nj.nec.com/searls93string.html [21/Feb/2001].

[105] D. Searls. Linguistic approaches to biological sequences. *CABIOS*, 13(4):333–344, 1997.

[106] D. Searls. Formal language theory and biological macromolecules. *Series in Discrete Mathematics and Theoretical Computer Science*, 47:117–140, 1999. http://citeseer.nj.nec.com/searls99formal.html [21/Feb/2001].

[107] D. Searls and S. Dong. A syntactic pattern recognition system for DNA sequences. In *Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, pages 89–101, 1993. http://citeseer.nj.nec.com/searls93syntactic.html [21/Feb/2001].

[108] I. Simon. *Linguagem Formais e Autômatos*. Segunda Escola de Computação, 1981. Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP.

[109] P. Smyth. Clustering sequences with hidden markov models. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 648. The MIT Press, 1997.

[110] D. P. Snustad, M. J. Simmons, and J. B. Jenkins. *Principles of Genetics*. John Wiley and Sons, 1997.

[111] G. Stoesser. The EMBL nucleotide sequence database. *Nucleic Acids Research*, 30(2):21–26, 2002.

[112] J. E. Tabaska, R. V. Davuluri, and M. Q. Zhang. Identifying the 3'-terminal exon in human dna. *Bioinformatics*, 17(7):602–607, 2001.

[113] T. A. Thanaraj. Positional characterisation of false positives from computational prediction of human splice sites. *Nucleic Acids Research*, 28(3):744–754, 2000.

[114] The Genome International Sequencing Consortium. Initial sequencing and analysis of the human genome, February 2001. http://www.nature.com/cgi-taf/DynaPage.taf?file=/nature/journal/v409/-n6822/full/409860a0_fs.html [16/Fev/2001].

[115] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 1999.

[116] J. Thompson, D. Higgins, and T. Gibson. improving the sensitivity of progressive multiple sequence alignment through sequence weighting. *Nucleic Acids Research*, 22:4673–4680, 1994.

[117] N. S. Tomita. Programação Automática de Máquinas Morfológicas Binárias baseada em Aprendizado PAC. Master's thesis, Instituto de Matemática e Estatística - Universidade de São Paulo, SP - Brasil, março 1996.

[118] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[119] D. Voet and J. G. Voet. *Biochemistry*. John Wiley and Sons, 2nd edition, 1995.

[120] E. O. Voit. *Computational Analysis of Biochemical Systems: a pratical guide for biochemists and molecular biologists*. Cambridge University Press, 2000.

[121] J. Wang, X. Wang, K. Lin, D. Shasha, B. Shapiro, and K. Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 307–311, San Diego CA, August 1999. ACM. http://www.msci.memphis.edu/∼linki/_mypaper/kdd99.ps.gz [26/Jan/2001].

[122] J. T. L. Wang, Q. Ma, D. Shasha, and C. H. Wu. Application of neural networks to biological data mining: A case study in protein sequence classification. In *The Sixth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 305–309, 2000.

[123] J. T. L. Wang, Q. Ma, D. Shasha, and C. H. Wu. New techniques for extracting features from protein sequences. *IBM Systems Journal, Special Issue on Deep Computing for Life Sciences*, 40(2), 2001.

[124] M. S. Waterman. *Mathematical Methods for DNA sequences*. Boca Raton, FL: CRC Press, 1989.

[125] J. J. Wiens, editor. *Phylogenic Analysis of Morphological Data*. Smithsonian Series in Comparative Evolutionary Biology. Smithsonian Institution Press, 2000.

[126] Wisconsin package version 10.0. Genetics Computer Group (GCG). Madison, Wisc.

[127] C. H. Wu. The protein information resource: an integrated public resource of functional annotation of proteins. *Nucleic Acids Research*, 30(2):35–37, 2002.

[128] A. Zaha. *Biologia Molecular Básica*. Mercado Aberto, 1996.

[129] Catálogo de disciplinas de pós-graduação em ciência da computação. http://sistemas.usp.br/fenixweb/fexDisLista?codare=45134&codcpg=45 [28/Jun/2002].

[130] Matlab 6. http://www.mathworks.com/products/matlab/ [28/Jun/2002].

[131] Bielefeld university bioinformatics server. http://bibiserv.techfak.uni-bielefeld.de/ [29/Jun/2002].

[132] Bioinformatics – oxford journals online. http://bioinformatics.oupjournals.org/ [29/Jun/2002].

[133] Ensembl genome browser. http://www.ensembl.org/ [29/Jun/2002].

[134] geneid web server. http://www1.imim.es/geneid.html [29/Jun/2002].

[135] Human genome project working draft. http://www.genome.ucsc.edu/ [29/Jun/2002].

[136] I latin american course on bioinformatics for tropical disease research. http://icb.ime.usp.br/tdr/ [29/Jun/2002].

[137] National center for biotechnology information. http://www.ncbi.nlm.nih.gov/ [29/Jun/2002].

[138] Researchindex. http://citeseer.nj.nec.com/cs [29/Jun/2002].

[139] VSNS biocomputing division multiple alignment resource page. http://www.techfak.uni-bielefeld.de/bcd/Curric/MulAli/welcome.html [29/Jun/2002].