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

Caetano Jimenez Carezzato

Projeto de Pesquisa — Mestrado

Vinculado ao Projeto Temático **CAGE** — **FAPESP** nº 99/07390-0

Orientador: **Professor Doutor Junior Barrera**

Co-orientador: **Sandro José de Souza**

Departamento de Ciência da Computação

Instituto de Matemática e Estatística

Universidade de São Paulo

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).

Para a comparação dos diversas métodos, modelaremos os dados por Gramáticas Estocásticas que serão estimadas a partir de dados reais. Essas gramáticas estocásticas serão utilizadas para a geração dos dados de testes. Dados conjuntos de seqüências de categorias diferentes, treinamos uma gramática estocástica para cada conjunto e geramos dados de testes através de realizações dessas gramáticas. Aplicaremos os algoritmos que queremos comparar nos dados gerados e verificaremos a precisão de cada um.

Sumário

1 Introdução e Justificativa

Com o crescente número de genomas seqüenciados sendo disponibilizados atualmente [31], inclusive o humano[47, 24], 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 [28] 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** [13, 52], montagem de fragmentos de **DNA** [1], mapeamento físico de **DNA** [2], árvores filogenéticas [53], reconhecimento de genes e partes de gens [39], busca de homologia [21], clustering [12, 16], predição da estrutura de proteínas [36] etc.

Busca de homologia e clustering têm aplicações muito importantes na área de Biologia Molecular. São utilizados principalmente para encontrar pedaços de gens e proteínas "parecidos" o que pode indicar que eles têm a mesma função. Tais pedaços novos devem ser investigados bioquimicamente, o que é um processo caro. Tal fato motiva a descoberta e comparação de algoritmos propostos para o problema, visto que um bom algoritmo para busca de homologia pode implicar numa grande economia de tempo e dinheiro.

Para realizar tal tarefa, diversos algoritmos para comparação e classificação de seqüências têm sido desenvolvidos ao longo dos últimos anos[17, 51, 7]. Pretendemos estudar diversos métodos diponíveis atualmente para comparação de seqüências em bancos de dados, desde os mais clássicos, como o *BLAST* [4, 3], *FAST* [27, 34, 35], **BLOCKS** [23] e **BLOSUM** [20, Seção 15.9], até os métodos mais recentes como os baseados em Gramáticas Estocásticas [38, 37], Árvores de Decisão [9] e métodos baseados em **COGs** [46].

O objetivo principal deste trabalho é comparar diversos métodos computacionais disponíveis atualmente para clustering e busca de homologia em seqüências de **DNA**, **RNA** e proteínas. Para tal, dentre os diversos modelos probabilísticos disponíveis para modelagem de seqüências [15], modelaremos os dados por Gramáticas Estocásticas [18] que serão estimadas a partir de dados reais utilizando-se diversas técnicas de reconhecimento de padrões [14, 48] e aprendizado computacional [50, 6]. Essas gramáticas estocásticas serão utilizadas para a geração dos dados de testes. Dados conjuntos de seqüências de categorias diferentes, treinamos uma gramática estocástica para cada conjunto e geramos dados de testes através de realizações dessas gramáticas. Aplicaremos os algoritmos que queremos comparar nos dados gerados e verificaremos a precisão de cada um.

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

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

de Matemática e Estatística da Universidade de São Paulo com o objetivo de estudar os mecanismos de expressão gênica.

Uma versão eletrônica deste documento com links para alguns dos documentos citados pode ser encontrada em:

http://www.vision.ime.usp.br/∼caetano/mestrado/projeto/proposta.pdf

2 Aspectos Teóricos

Ao longo desta seção vamos introduzir, de forma muito resumida, os conceitos básicos necessários para a formulação da proposta.

2.1 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 [20].

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, entre eles *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 [55].

2.1.1 Comparação de Seqüências

A área de Comparação e Casamento Aproximado de Seqüências é importantíssima na Biologia Molecular Computacional [20, 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*[52]. 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.

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.

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 [20, 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**.

2.2 Reconhecimento de Padrões

A área de reconhecimento de padrões [48] 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.

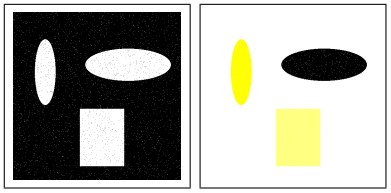

Figura 1: Exemplo de reconhecimento de padrões. Figura original e objetos reconhecidos

Por exemplo, suponha que eu 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 somente uma foto do indivíduo possa estar disponível para a classificação, dessa forma, teremos que *extrair carac-*

terísticas 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*. Nesse modelo, cada classificador ao invés de responder "sim" ou "não", indicam a probabilidade do objeto em questão pertencer ao *conceito* desejado. Dado um conjunto de clasificadores, a *decisão bayesiana* simplesmente é escolher o classificador que devolve a maior probabilidade.

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.

2.2.1 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

[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 [40] onde ele expõe sua famosa alegoria do "Quarto Chinês"

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 [49] na qual está baseado o modelo **PAC** básico de Valiant [50] 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 $L(s) = h$. 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 [5], 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 [22] generaliza o modelo **PAC** de forma a permitir exemplos contraditórios e dados em alfabetos que não são binários.

2.2.2 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 2.2, 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 [48]. 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 [48, 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 [48, 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 [48, Capítulo 14].
- **Outros tipos**. Existem diversos outros tipos como algoritmos para genética, algoritmos estocásticos e algoritmos baseados em técnicas de transformações morfológicas [48, Capítulo 15].

2.3 Linguagens Formais e Modelos Estocásticos

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

2.3.1 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 $L_1L_2 = \{st \in \Sigma^* \mid s \in L_1, t \in L_2\}$.

2.3.2 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 $s_i = t_iu_it'_i$, $s_{i+1} = t_iv_it'i$ 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 [26].

A classe de linguagens recursivamente enumeráveis é muito grande [32, 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 [11, 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 [10] 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 [45, 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 [42, 43] apresenta. Tudo isso motivou Searls a estudar uma classe de gramática intermediária chamada *string variable grammar* [41, 44, 42] que não é tão difícil de reconhecer quanto as linguagens sensíveis ao contexto mas é mais genérica que as livres de contexto.

2.3.3 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 2.3.2 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 2.3.2, decidimos representar os classificadores usando gramáticas estocásticas livres de contexto e regulares.

2.3.4 Treinamento de Gramáticas Estocásticas

Para treinar as gramáticas descritas na Seção 2.3.3, 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* [30] e o *Tree Grammar EM* [25], entre outros. Pretendemos estudar diversos métodos para treinamento e, eventualmente,

compará-los.

2.3.5 Geração Estocástica dos Dados de Testes

Existem diversos modelos probabilísticos para modelagem de seqüências [15]. Escolhemos utilizar Gramáticas Estocásticas para o processo estocástico de geração de seqüências de testes.

Dados conjuntos de exemplos identificados (ou seja, eu sei a que classes cada palavra de treinamento pertence), treinamos gramáticas estocásticas, uma para cada classe de palavras fornecidas, utilizando-se técnicas descritas na Seção 2.3.4. Utilizamos então um gerador estocástico de palavras, que recebe o conjunto de gramáticas treinadas e simplesmente gera palavras de exemplos estocasticamente de acordo com as gramáticas fornecidas.

2.4 Ferramentas Auxiliares

O conhecimento de diversas áreas da Teoria da Computação e Biologia é muito importante para a compreensão e implementação dos algoritmos necessários para o trabalho que está sendo proposto.

Além das áreas já citadas como Reconhecimento de Padrões e Linguagens Formais, podemos citar Processamento de Imagens [19], Teoria dos Grafos [8], Álgebra Booleana [29], Otimização Combinatória [33]. Tais áreas tem profunda correlação com os objetivos do trabalho e devem ser estudadas também.

3 Objetivos

O objetivo principal deste projeto é comparar as diversas técnicas de reconhecimento de padrões supervisionado e não supervisionado aplicado à seqüências de **DNA** e proteínas.

Para alcançar o objetivo supracitado, podemos destacar os seguintes tópicos:

- Estudo de técnicas de Biologia Computacional
  Será feita uma pesquisa dos diversos algoritmos de clustering e comparação de seqüências disponíveis atualmente, inclusive a leitura de [20]. Queremos comparar as diversas abordagens e observar suas vantagens e limitações.
- Estudo de técnicas de reconhecimento de padrões e clustering
  Será realizada uma leitura dirigida de [48] visando compreender melhor diversos aspectos relacionados ao tema. Reconhecimento de padrões na área de Processamento de Imagens já foi tema dos últimos 3 anos de iniciação científica do aluno.
- Estudo de linguagens formais e gramáticas
  Estudaremos diversos aspectos do treinamento de gramáticas estocásticas e algoritmos disponíveis.
- Avaliação de diversos algoritmos disponíveis
  O objetivo do projeto é apresentar uma avaliação de diversos algoritmos que estão disponíveis atualmente na área de reconhecimento de padrões aplicado à Biologia Molecular principalmente para *data mining*, como **BLAST**, **FASTA**, **PFAM**, **BLOCKS**, **BLOSUM**, **COG** entre muitos outros. Os dados devem ser apresentados de forma clara e consistente.

4 Análise dos Resultados

Os resultados obtidos serão avaliados principalmente em relação a *taxa de erro* verificada com dados simulados por gramáticas estocásticas e eventualmente reais. Na Figura 2 temos uma representação esquemática do ambiente de testes simplificado.

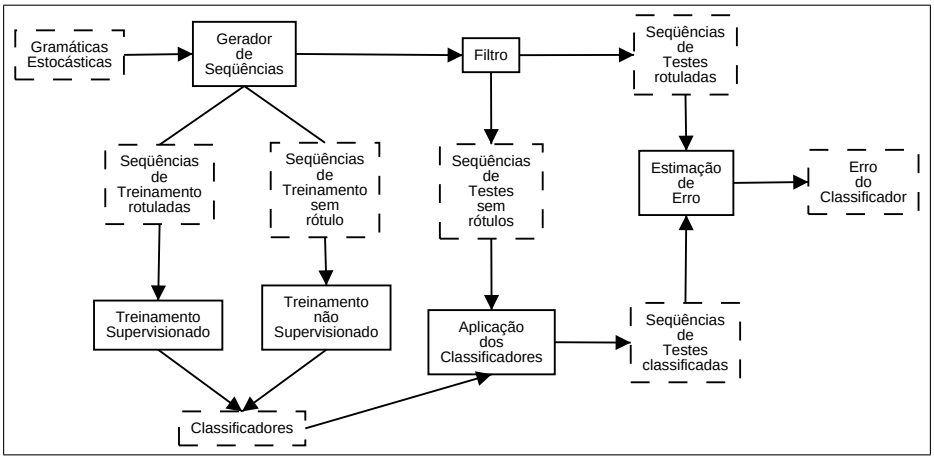

Figura 2: Diagrama simplificado do ambiente de testes

Podemos definir o *erro de um classificador* como a proporção entre classificações erradas e o número de classificações realizadas. Dessa forma, podemos gerar dados simulados de seqüências de **DNA** através do treinamento de Gramáticas Estocásticas. Cada Gramática Estocástica corresponde a uma classe de seqüências representadas de forma finita.

Para o caso supervisionado, simplesmente geramos um conjunto de seqüências de treinamento rotuladas e as fornecemos para os algoritmos de treinamento supervisionado. Para calcular o erro dos classificadores gerados, geramos um conjunto t de seqüências de testes rotuladas e fornecemos t para um filtro que gera outro conjunto t' que contém as mesmas seqüências de t só que sem rótulo.

Fornecemos t' aos classificadores que geram um conjunto c que contém as mesmas seqüências que t' só que rotuladas pelos classificadores. O erro então é facilmente calculado comparando-se os rótulos de t com os de c.

O caso não supervisionado é análogo, só que as seqüências de treinamento fornecidas não são rotuladas.

Dessa forma, podemos aplicar essa abordagem a diversos algoritmos, medidas de distâncias e critérios de clustering, apresentando um comparativo entre os mesmos. Além disso, uma avaliação interessante é gerar diversas classes de gramáticas e testar os algoritmos misturando classes cada vez mais parecidas e observar como o erro evolui.

5 Materiais e Métodos

Estão disponível nos nossos laboratórios diversas ferramentas necessárias para a realização do nosso projeto. Na Figura 3 temos um esquema da estrutura principal. Dentre as ferramentas disponíveis, podemos destacar:

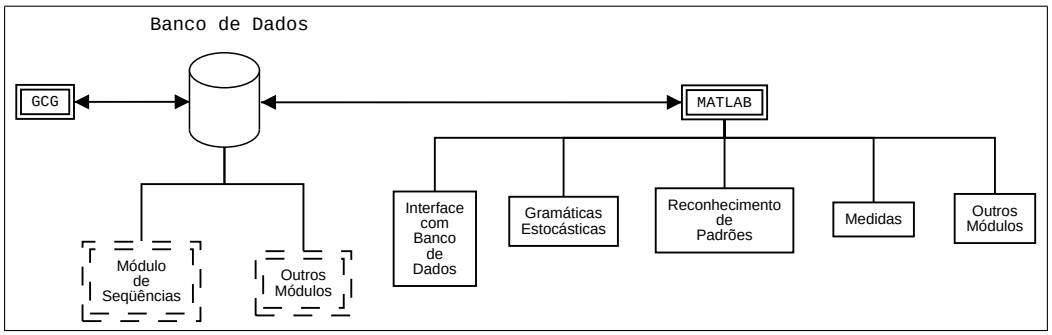

Figura 3: Ambiente de Trabalho Simplificado

- Um banco de dados com cerca de 9 milhões de seqüências (Gene bank)
- O **software GCG** [54] para análise de seqüências, muito popular em laboratórios pelo mundo todo.
- O **software Matlab**, ambiente multiplataforma e para desenvolvimento rápido.
- Rede **Linux** com diversos computadores capazes de processar os dados necessários
- Um **Beowulf** com 16 nós. Cada nó tem um processador 1.2 Ghz e 768 Megabytes de RAM.

O **Matlab** permite a criação de um ambiente padrão integrado de forma que todos os programas que o grupo está desenvolvendo sejam facilmente acessíveis através da mesma ferramenta. O desenvolvimento dos módulos propostos serão preferencialmente realizados na linguagem **C**, de forma a garantir uma maior performance e facilidade de depuração. Todo o desenvolvimento será realizado no ambiente **Linux** e eventualmente se necessário **Unix** de forma a garantir uma maior estabilidade e escalonibilidade, podendo vir a ser desenvolvidas versões paralilizadas de alguns algoritmos.

Como mostrado na Figura 3, utilizaremos diversos outros módulos como "Interface com Banco de Dados", "Gramáticas Estocásticas", "Reconhecimento de Padrões" e "Medidas". Os dois primeiros estão em desenvolvimento sob responsabilidade de outros membros do grupo. Já os dois últimos estão implementados no **Matlab**, **GCG** e outros pacotes, podendo ser complementados a medida que realizarmos os diversos testes propostos no nosso trabalho.

Para alcançarmos os objetivos descritos na Seção 3, as seguintes tarefas principais devem ser realizadas:

- Especificação e implementação de um gerador estocástico de palavras
  Peça fundamental para os testes dos algoritmos a serem desenvolvidos. A entrada para esse módulo é um conjunto de Gramática Estocástica e o número de palavras desejado e a saída é um conjunto de palavras que podem estar rotuladas ou não.
  Será especificado e implementado também uma biblioteca para realizar a comunicação entre cada módulo do projeto que utilize gramáticas e seqüências.

– Especificação de um ambiente de treinamento para reconhecimento de padrões
  Procuraremos projetar um ambiente para reconhecimento de padrões a ser implementado no **Linux** utilizando-se principalmente a linguagem **C** para a implementação dos algoritmos e o ambiente **Matlab** para a integração com os demais módulos produzidos pelo grupo.
  Inicialmente especificaremos um ambiente para treinamento supervisionado. Após testes desse ambiente, especificaremos os módulos necessários para clustering.
– Especificação de um ambiente para testes
  O ambiente de testes é a peça mais importante deste projeto. Ele simplesmente aplica os classificadores obtidos em dados simulados ou reais para poder fazer a comparação entre os diversos tipos de algoritmos, medidas e abordagens de treinamento disponíveis. Ele deve ser capaz de gerar diversas informações automaticamente, com gráficos e tabelas comparativas.
– Implementação dos ambientes especificados
  A implementação deve ser feita de forma clara e bem documentada de forma a facilitar a leitura do código. Além disso, o código deve ser facilmente extensível, multiplataforma e escalonável, visto que poderemos ter diversos ambientes para rodar os experimentos como máquinas paralelas de grande porte.
– Especificação e implementação do Aprendizado de Gramáticas Estocásticas
  Diversos métodos estão sendo desenvolvidos no nosso laboratório para o treinamento de gramáticas estocásticas regulares e livres de contexto. Este item está sendo realizado por outros membros do grupo.

Na Figura 4 temos uma representação simples do ambiente de treinamento a ser desenvolvido. Se os dados são rotulados, o treinamento é supervisionado, caso contrário, o treinamento é não supervisionado.

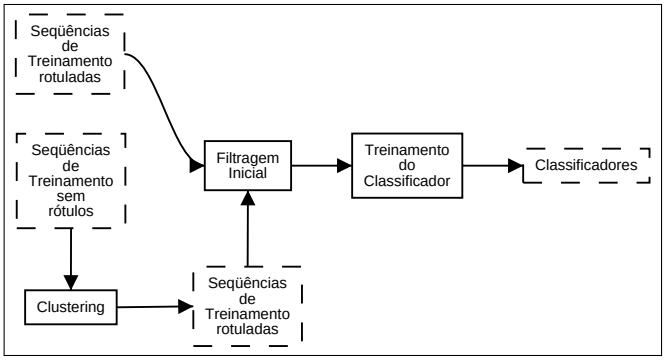

Figura 4: Representação esquemática de treinamento computacional supervisionado e não supervisionado

A filtragem inicial apresentada na Figura 4 será implementada utilizando-se abordagens semelhantes às utilizadas na área de Processamento de Imagens. Essa filtragem servirá para eliminar seqüências "muito diferentes" do conjunto, seqüências que dificultem o treinamento, ou seqüências que diminuem a precisão dos classificadores por causa do método de treinamento utilizado. O treinamento supervisionado de classificadores já está implementado para imagens e modificado para lidar com **DNA**. Atualmente o nosso grupo está implementado uma versão modificada desse treinamento que utilizará gramáticas estocásticas para representar os classificadores, que, de certa forma, "descrevem" um conjunto de seqüências.

Inicialmente simplesmente vamos fazer uma comparação dos diversos algoritmos que realizam o que foi descrito na Seção 2.1.1 e na Seção 2.2 disponíveis no mercado, principalmente os do **GCG** e **Matlab** . Caso julguemos conveniente, implementaremos outros algoritmos. É necessário bastante estudo nestes tópicos para a escolha do que vamos comparar, pois existem muitos algoritmos e diversas medidas disponíveis atualmente.

6 Plano de Trabalho e Cronograma

Resumidamente, os primeiros dois semestres serão mais dedicados a leitura, aulas, especificação do sistema e um pouco de implementação. O terceiro semestre para testes e avaliações e o último é mais reservado para a redação de papers, dissertação e defesa, bem como uma geração mais detalhada de experimentos. A seguir, apresentamos uma visão mais detalhada das principais tarefas que necessárias para o projeto.

Atividades			
1	Disciplinas do programa de pós-graduação	2	Leitura supervisionada de [48]
3	Estudo de Biologia Computacional	4	Estudo de linguagens formais e gramáticas
5	Estudo do **Matlab** e **GCG** e suas ferramentas	6	Especificação do gerador estocástico de palavras
7	Especificação do Ambiente de Treinamento	8	Especificação dos Testes
9	Implementação do gerador estocástico de palavras	10	Implementação dos Ambientes Especificados
11	Implementação dos Testes	12	Primeira avaliação dos algoritmos
13	Segunda avaliação dos algoritmos	14	Redação de relatórios semestrais para a FAPESP
15	Redação da proposta da dissertação	16	Seminários
17	Avaliação dos resultados	18	Redação da dissertação

Cronograma

2001/2002

	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez	Jan	Fev	Mar
1	•	•	•	•		•	•	•	•			
2					•	•	•	•	•			
3	•	•					•	•				
4	•	•	•	•								
5		•	•	•	•		•		•		•	
6	•											
7					•	•	•					
8								•	•	•		
9		•										
10								•	•	•	•	•
14						•						•

2002/2003

	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez	Jan	Fev	Mar
3	•	•					•	•				
5	•		•		•		•		•			
10	•	•			•	•		•	•			
11	•	•	•		•	•		•	•			
12			•	•	•							
13							•	•	•			
14						•						
15	•											
16	•	•	•	•	•							
17									•	•	•	
18										•	•	•

Referências

[1] 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].

[2] 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].

[3] 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].

[4] 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.

[5] 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.

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

[7] 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].

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

[9] 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].

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

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 21st edition, 1998.

[12] 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. *Submetido*, 2001.

[13] 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.

[14] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973.

[15] S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.

[16] 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].

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

[18] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Inc., Englewood Cliffs,NJ, 1982.

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

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

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

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

[23] J. G. Henikoff and S. Henikoff. BLOCKS database and its applications. In R. F. Doolittle, editor, *Methods in Enzymology*, volume 266, pages 402–427. Academic Press, 1996.

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

[25] 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.

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

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

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

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

[30] 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].

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

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

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

[34] 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.

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

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

[37] 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].

[38] Y. Sakakibara, M. Brown, I. Mian, R. Underwood, and D. Haussler. Stochastic context-free grammars for modeling RNA. In *Hawaii Intl. Conf. on System Sciences*. IEEE Computer Society Press, 1993. http://citeseer.nj.nec.com/sakakibara93stochastic.html [30/Jan/2001].

[39] 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].

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

[41] 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].

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

[43] 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].

[44] 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].

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

[46] R. L. Tatusov, D. A. Natale, I. V. Garkavtsev, T. A. Tatusova, U. T. Shankavaram, B. S. Rao, B. Kiryutin, M. Y. Galperin, N. D. Fedorova, and E. V. Koonin. The COG database: new developments in phylogenetic classification of proteins from complete genomes. *Nucleic Acids Res*, 29(1):22–28, Jan 2001. http://nar.oupjournals.org/cgi/reprint/29/1/22.pdf [11/Abr/2001].

[47] 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].

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

[49] 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.

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

[51] 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].

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

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

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

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