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

Caetano Jimenez Carezzato

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU DE MESTRE
EM
CIÊNCIAS

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Junior Barrera

– Durante o desenvolvimento deste trabalho, –
– o autor recebeu apoio financeiro da FAPESP –

São Paulo
2004

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

Este exemplar corresponde à redação final
da dissertação de mestrado devidamente
corrigida e defendida por
Caetano Jimenez Carezzato
e aprovada pela comissão julgadora.

São Paulo, 10 de abril de 2004.

Banca examinadora:

- Prof. Dr. Junior Barrera (orientador) – IME-USP
- Prof. Dra. Ana Tereza Ribeiro de Vasconcelos – LNCC-RJ
- Prof. Dr. Georgios Joannis Pappas Júnior – UCB-DF

A todos aqueles a quem, de alguma forma, este texto venha a ser útil.

Agradecimentos

Inicialmente, agradeço a todos aqueles que apoiaram a realização deste trabalho, vocês são incríveis.

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

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

Ao professor Doutor **Roberto de Alencar Lotufo** pela sua colaboração, muito útil para a realização deste trabalho.

Um agradecimento especial ao professor Doutor **Junior Barrera**, meu orientador, pela paciência e confiança. Nunca nos esqueceremos das nossas seções de *brain storm*, sempre muito úteis para a realização deste trabalho e também das análises de questões não acadêmicas. É muito bom ser orientado por quem sabe orientar. Tenho certeza que encontrei um grande amigo.

Aos meus pais, que, apesar de não terem contribuído, em termos intelectuais, diretamente para a realização deste trabalho, estão sempre apoiando seus filhos.

Aos meus amigos **Márcio Carneiro**, **Livio Baldini Soares** e **Teofilo Emidio de Campos**, que sempre se dispuseram a perder seus preciosos tempos na tentativa de melhorar este trabalho.

Ao amigo **Alex Pires de Camargo**. Aprendemos muita coisa juntos, inclusive a importância da superação e aceitação mútua, apesar de todas as várzeas. Você é, sem sombra de dúvida, um grande amigo.

Ao grande amigo **Rodrigo Nonamor Pereira Mariano de Souza**, que me inspira em muitas coisas que faço.

Ao também grande amigo **Rodrigo Souza de Castro**, que sempre dá apoio moral a tudo que deve ser apoiado. Aprendi muito com você e quero continuar aprendendo.

Aos amigos **Roberto Hirata Junior** e **Roberto Marcondes Cesar Junior**, pela convivência ao longo destes anos. Aprendi muito com vocês, principalmente pela forma diferente de abordar as situações da vida. Aproveitando, ao amigo **David da Silva Pires** que me ensinou e ensinará muita coisa. Acredito ter encontrado excelentes amigos durante esta jornada.

Ao **pessoal do laboratório**, que teve de me agüentar por quase 5 anos, mas agora está livre de mim. Encontrei muitos amigos nesse ambiente, que jamais serão esquecidos.

Algumas pessoas importantes não são citadas aqui simplesmente porque esqueci, peço desculpas. Outras, não são citadas porque, apesar de serem importantes, já conversamos

tudo o que deveria ser conversado. Não há mais nada a ser dito. Muito obrigado por vocês existirem.

Um agradecimento bastante especial para os funcionários públicos em geral. Nestes quase 7 anos de universidade pública eu consegui compreender por que, apesar de serem somente 5% da população nacional, conseguem consumir mais de 25% do PIB nacional. Existem certas coisas que me surpreendem.

À **FAPESP**, que ao me conceder uma bolsa de mestrado (processo nº $01/03975-5$), permitiu que eu me dedicasse integralmente a este trabalho e compreendesse melhor o funcionalismo público.

Por fim, e portanto muito menos importante, um agradecimento especial a todos aqueles que fizeram e fazem questão de atrapalhar (intencionalmente ou não). Apesar de todo o esforço, vocês ainda precisam melhorar muito. Eu ia citá-los, mas prefiro parar por aqui. De qualquer forma, aprendi muita coisa útil com vocês. Sem isso, minha formação não chegaria aonde chegou. Muito obrigado por tudo!

Na Índia conta-se uma lenda de um menino que participava de um grupo religioso.
Tal grupo adorava determinada pessoa, acreditando que ela era Deus.
O menino, descrente, vai até ela, em particular, e fala:
– Eu não acredito que você é realmente Deus. Estou indo embora.
E, então, recebe a seguinte resposta:
– Tudo bem, você é livre para ir.
O menino sai e, já fora do templo, volta-se repentinamente para trás. Vê então uma cena muito curiosa. A pessoa está... chorando[a] e sorrindo para o menino.
O menino volta-se novamente e vai embora, partindo para uma nova jornada em busca de novos conhecimentos.

---

[a]Para compreender mais profundamente o significado desta lenda, é necessário entender que Deus, ou essa figura que representa Deus, ao contrário do "Deus Ocidental", nunca choraria de tristeza.

Resumo

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

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

Tais características são baseadas em técnicas de representação *Chaos Game* e de análise fractal multi-escala. Ao longo desta dissertação apresentamos os elementos necessários para a compreensão das diversas técnicas e características analisadas e um resumo dos principais resultados obtidos com a utilização de tais características para a busca de genes que raramente se expressam.

Abstract

In the past years, millions of **DNA** and proteins sequences have been deposited in public databases throughout the world. Currently, the main challenge in Computational Molecular Biology is to analyse and extract useful information of this great quantity of available data. This work's goal is to study several computational methods for homology searches and *clustering* (that is, supervised and non-supervised pattern recognition) in nucleotide sequences.

A vast bibliographical study of the main methods in the pattern recognition area has been performed; specially the ones most used in Computational Biology. A summary of the main methods in the area are presented, along with results of experiments performed with human genes utilizing a new form of features extraction from **DNA** sequences.

These features are based on *Chaos Game* representation techniques and on multi-scale fractal analysis. Throughout this dissertation, we present the necessary elements for the comprehension of the various techniques and features analyzed, and a summary of the principal results obtained with the utilization of the features for the search of genes rarely expressed.

Índice

Lista de Figuras

Capítulo 1

Introdução

Com o crescente número de genomas seqüenciados disponíveis atualmente [98], inclusive o humano [136, 71], um problema muito importante surge imediatamente na área de Biologia Molecular: como extrair informações desses enormes bancos de dados de seqüências.

A Biologia Molecular Computacional [89] 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 na área, entre eles: comparação de seqüências de **DNA** [36, 149], alinhamento de múltiplas seqüências [65, 139, 95, 96], montagem de fragmentos de **DNA** [1], mapeamento físico de **DNA** [2], árvores filogenéticas [151], reconhecimento de genes e partes de genes [117, 18], busca de homologia [59], *clustering* [35, 41], predição da estrutura de proteínas [113, 116], comparação de genomas [9] etc.

O objetivo principal deste trabalho é estudar os diversos métodos computacionais disponíveis para *clustering* e busca de homologia em seqüências de **DNA**, **RNA** e proteínas. Foi realizada vasta revisão bibliográfica dos principais métodos utilizados na área de reconhecimento de padrões, principalmente em Biologia Computacional. Nos restringimos à análise de técnicas que têm por objetivo a classificação de regiões cromossômicas, em especial na busca de genes humanos. Apresentamos um resumo das principais técnicas utilizadas na área e dos novos experimentos realizados com genes humanos.

Durante o projeto, foi realizada a investigação de novos métodos para extração de características de seqüências de **DNA**. Tais métodos consistem em observar uma seqüência dada por meio da freqüência de utilização de subpalavras de tamanho arbitrário.

Introduzimos e analisamos 6 novas características que podem ser extraídas a partir de técnicas de representação *Chaos Game* (**CGR**) e também de técnicas de dimensão fractal. Apresentamos um modelo para a busca de genes que utiliza tais características e analisamos a viabilidade da técnica.

Abordamos o problema de encontrar genes humanos raros (aqueles que raramente se expressam) através da utilização de técnicas de aprendizado computacional. Para tal, avaliamos o tamanho que as subpalavras devem ter, bem como comparamos diversos métodos que analisam as tabelas de freqüências obtidas.

Durante o trabalho, foi dada ênfase ao entendimento das técnicas de um ponto de vista mais global e conceitual, ou seja, foram estudadas diversas técnicas a fundo mas sem entrar em detalhes práticos a respeito de sua implementação.

1.1 Organização deste Documento

Há, entre os Capítulos 2 e 5, uma breve revisão bibliográfica dos principais tópicos da área estudados. Nos Capítulos 6 e 7 apresentamos alguns conceitos necessários para a compreensão das novas características propostas. Na Seção 6.5 e no Capítulo 8 destacamos as diversas contribuições que realizamos ao longo do trabalho.

No Capítulo 2 descrevemos brevemente alguns aspectos básicos a respeito de biologia molecular e biologia molecular computacional. Apresentamos o **DNA** e um modelo de gene atualmente aceito pelos biólogos.

No Capítulo 3 discutimos brevemente alguns aspectos básicos a respeito de reconhecimento de padrões supervisionado e não supervisionado. Apresentamos também algoritmos hierárquicos.

No Capítulo 4 exibimos algumas técnicas e abordagens necessárias para a compreensão das principais abordagens de reconhecimento de padrões. Analisamos algumas questões importantes a respeito do projeto de classificadores, dentre elas, técnicas para a redução do espaço de hipóteses e análise multi-escala e multi-resolução. Concluímos o capítulo com alguns modelos clássicos, como os modelos de Markov e os baseados em gramáticas.

No Capítulo 5 apresentamos técnicas clássicas para a comparação de seqüência e as principais técnicas utilizadas atualmente para a busca de genes.

No Capítulo 6 expomos de forma informal alguns aspectos sobre teoria do caos e sistemas dinâmicos. O objetivo principal é introduzir os elementos necessários para uma compreensão mais aprofundada da representação *Chaos Game* que é apresentada no final do capítulo.

No Capítulo 7 introduzimos alguns aspectos da dimensão fractal. Apresentamos as principais idéias e, no final, um método multi-escala para a análise de fractalidade que será aplicado em imagens de representação *Chaos Game*.

No Capítulo 8 propomos um novo modelo para a busca de genes que utiliza as características apresentadas nos capítulos anteriores. No final do capítulo mostramos alguns resultados de desempenho das características propostas usando experimentos de aprendizado não supervisionado e supervisionado que buscam validar o modelo e características introduzidos nos capítulos anteriores.

No Capítulo 9 concluímos a dissertação com as principais contribuições deste trabalho. Dentre elas, podemos destacar:

- Algoritmo eficiente para cálculo de dimensão fractal multi-escala
- Algoritmos para o cálculo de imagens **CGR** e de gramáticas estocásticas
- 6 novas formas de extração de características
- Proposta de modelo para busca de gene

Para finalizar, apresentamos apêndices com informações extras, úteis para a compreensão do trabalho.

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 no site do projeto:

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

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

Capítulo 2

Alguns Aspectos da Biologia Molecular

O objetivo deste capítulo é descrever brevemente alguns aspectos da Biologia Molecular necessários para o entendimento da parte biológica do trabalho. Iniciamos com uma breve descrição sobre o **DNA**, terminando na apresentação de um modelo de gene aceito atualmente.

2.1 O DNA

O **DNA** 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, 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**. Na Figura 2.1 temos uma representação esquemática de tal estrutura.

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 uma seqüência de letras. Além disso, muitos algoritmos já estão disponíveis para lidar com esse tipo de dado [58].

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 segmentos do genoma que são transcritos. Muitos genes são responsáveis pela codificação de *proteínas*. Nas Figuras 2.2 e 2.3 temos representações esquemáticas de cromossomos.

Proteínas são macromoléculas muito importantes para o organismo. Existem diversos tipos, 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 está presente. É a seqüência dessas moléculas que define cada tipo de proteína. Essa seqüência é codificada a partir da

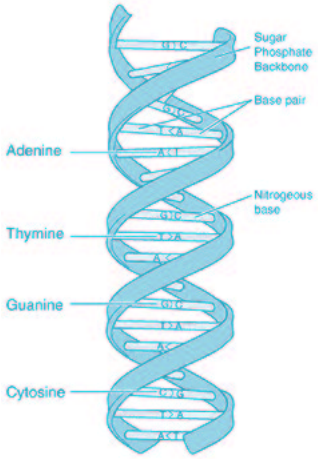


Figura 2.1: Representação esquemática do **DNA** extraída de [32]

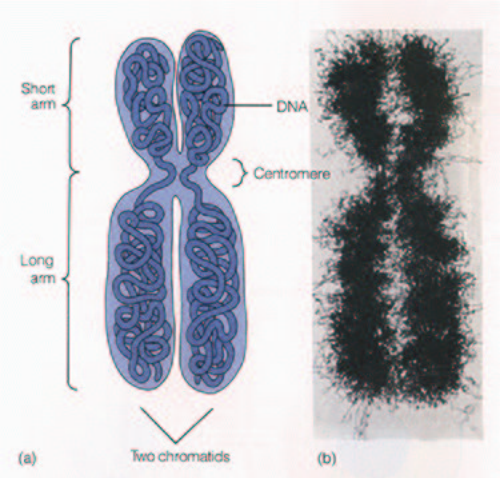


Figura 2.2: Representação esquemática de cromossomo extraída de [80]

seqüência do **DNA**, em que cada aminoácido é codificado por uma região de três *nucleotídeos*, denominada *códon*.

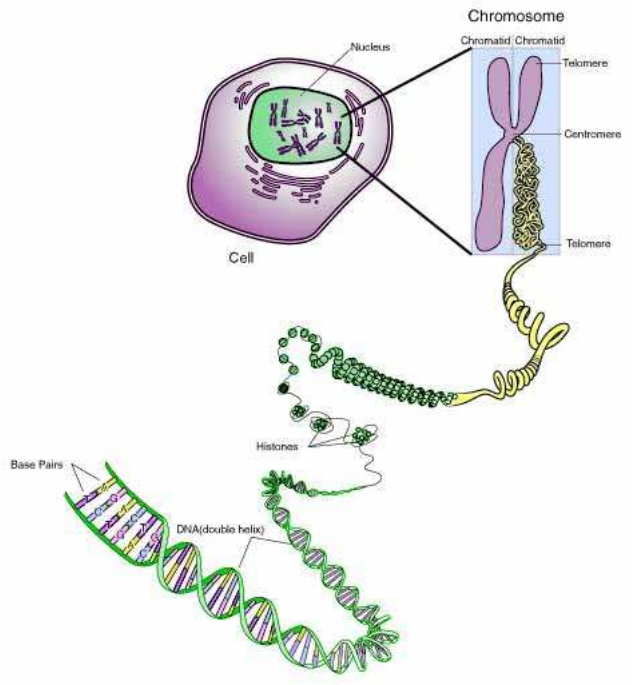


Figura 2.3: Representação esquemática da formação de cromossomos extraída de [88]

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 suas estruturas *secundária*, *terciária* e *quaternária*. Na Figura 2.4 temos uma representação esquemática tridimensional de uma proteína.

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 Zaha [156], Mount [97], Gibas *et al.* [52], Davis [29], Snustad *et al.* [127], Colton *et al.* [44], Voet *et al.* [144], Griffiths *et al.* [56] e Meidanis *et al.* [89].

2.2 Modelo de Gene

Um gene de organismos eucariontes é composto de diversas subestruturas. Ele inicia com uma região promotora, seguida pelo primeiro éxon que tem o códon de início. Esse primeiro éxon pode ser único ou seguido por uma série de íntrons e éxons externos. O gene termina com um éxon final que contém um códon de parada. Após o final de um gene há uma série de adeninas (poli-A). As bordas éxon/íntron e íntron/éxon (os *splice sites*) são sinalizadas por seqüências de duas bases (GT e AG). Uma borda éxon/íntron é denominada *donor site* e uma borda íntron/éxon é denominada *acceptor site*. O gene tem também uma região não

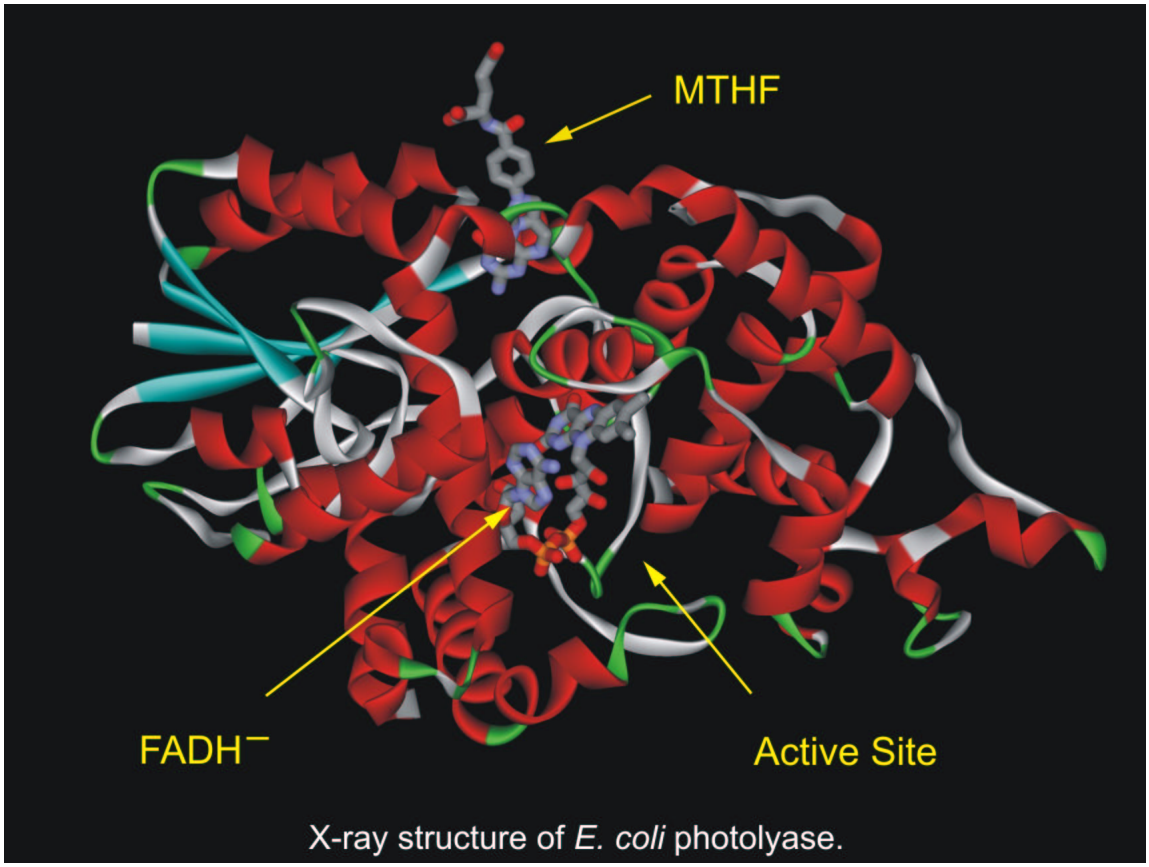


Figura 2.4: Representação esquemática tridimensional de proteína extraída de [159]

codificadora chamada 5′ UTR (*Untranslated Region*) e uma região 3′ UTR que podem estar em vários éxons.

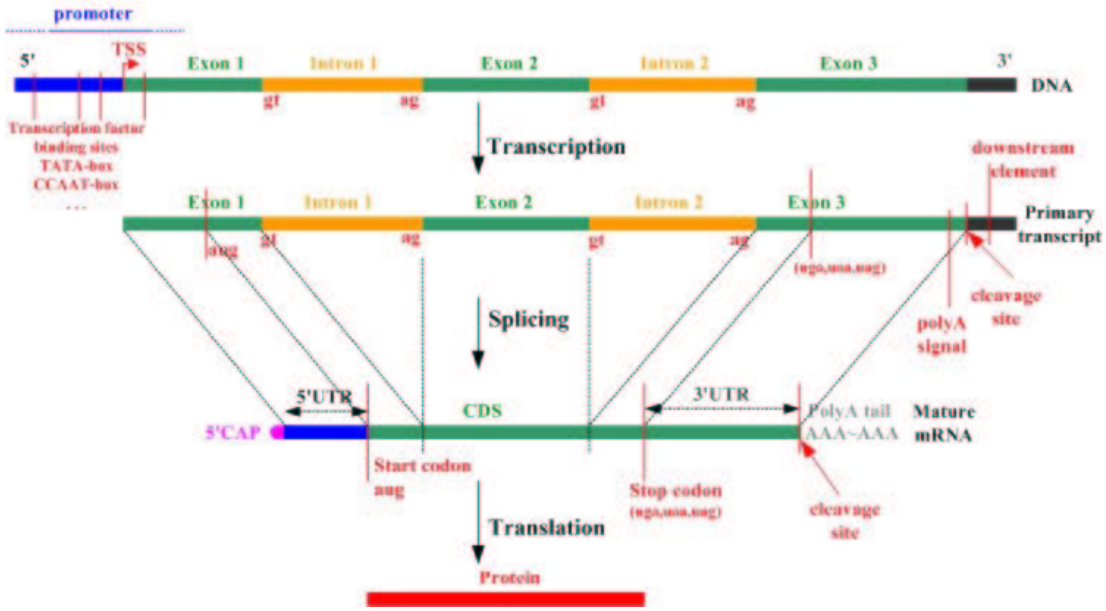


Figura 2.5: Modelo de gene extraído de [82]

Inicialmente todo o gene é transcrito para o **RNA**, excetuando a região promotora por uma enzima chamada **RNA** *Polimerase*. Logo em seguida, um processo denominado *splicing* elimina as partes correspondentes aos íntrons. Somente a parte codificante (CDS) dos éxons é, finalmente, traduzida em proteína.

Na Figura 2.5 temos um esquema de gene atualmente aceito.

O gene, além de ser uma região do genoma capaz de ser transcrita para produzir **RNA** funcional, precisa ser transcrito na hora e no lugar corretos para que seja funcional. Para organizar a transcrição, os genes têm uma região reguladora (promotora) que permite a interpretação de sinais oriundos de outras partes do genoma ou do ambiente.

As regiões do cromossomo em que não há genes são denominadas regiões intergênicas. Ainda não se conhece muito a respeito dessas regiões.

Atualmente, a capacidade de distinguir regiões intergênicas de regiões de genes é algo muito importante. É um problema amplamente estudado e difícil de abordar. Ao longo desta dissertação, vamos apresentar alguns aspectos necessários para a abordagem de alguns tópicos desse problema.

Capítulo 3

Reconhecimento de Padrões

A área de Reconhecimento de Padrões tem por objetivo encontrar ordens (padrões) em conjuntos de objetos que podem estar ou não classificados em categorias ou classes.

É uma área muito vasta. O objetivo deste capítulo é apresentar uma breve descrição a respeito de algumas poucas técnicas de Reconhecimento de Padrões necessárias para o entendimento dos tópicos estudados neste trabalho. Uma visão mais abrangente pode ser encontrada nem Theodoridis *et al.* [137], Russel *et al.* [115] e Duda *et al.* [37].

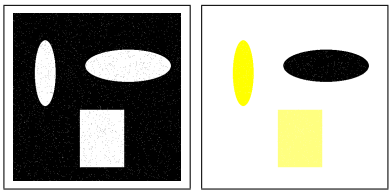


Figura 3.1: Exemplo de reconhecimento de padrões. Figura original e objetos reconhecidos

Por exemplo, suponha que se queiram reconhecer os diversos objetos na Figura 3.1, que está cheia de ruído. Um modo bem simplista de fazer isso é encontrar as componentes conexas da imagem e fazer um *thresholding* (limiarização) a partir do histograma das componentes conexas considerando apenas as "maiores" [53, 109]. O resultado obtido são três regiões distintas, como mostrado. Tal procedimento pode ser denominado *classificador*. As medidas utilizadas pelo classificador 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, o número de *pixels* de uma região conexa. O classificador simplesmente encontra o vetor que tem os maiores valores de acordo com algum critério que podemos estabelecer, como os valores que forem maiores 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 distinga homens de mulheres, a característica *cor do cabelo* não é muito interessante. Já a característica *presença de barba* pode ser mais útil (mas não suficiente). Um vetor de características bom poderia indicar o número de cromossomos X presentes numa célula da pele do indivíduo. Com esse vetor seria bem fácil a classificação.

Na prática, entretanto, nem todas as informações que queremos estão disponíveis. Pode ser que somente uma foto de cada indivíduo esteja disponível para a classificação, dessa forma, é necessário *extrair características* da foto para construir o vetor de características. Após essa primeira etapa, decide-se quais dentre as características geradas devem ser utilizadas. Em geral, um número de características maior do que o necessário acaba sendo gerado;

deve-se então escolher as características que participarão do vetor de características de forma que ele contenha as informações necessárias para o projeto de um bom classificador e que, ao mesmo tempo, não contenha muitas variáveis livres.

Tendo escolhido as características apropriadas, é necessário aplicar algoritmos que tentem encontrar ordem no conjunto de exemplos (amostra de exemplos).

Quando a classificação dos vetores de características não está disponível, ou seja, somente recebemos como conjunto de exemplos os objetos a serem avaliados, o objetivo é encontrar alguma ordem no conjunto fornecido. Uma abordagem muito utilizada é a baseada em distâncias ou similaridades, que buscam agrupar os vetores de características por algum critério. Esse procedimento é denominado *reconhecimento de padrões não supervisionado* e é muito usado para agrupar seqüências de **DNA**, por exemplo. A análise das ordens encontradas por diversos algoritmos pode revelar informação importante a respeito do problema abordado.

Quando a classificação dos vetores de características é conhecida, ou seja, sabe-se a que *classe* cada vetor de característica pertence (por exemplo, podemos ter vetores que correspondem a homens e outros que correspondem a mulheres), esse tipo de reconhecimento de padrões é denominado *reconhecimento de padrões supervisionado*; é como se tivéssemos um "professor" para nos "mostrar a verdade", mas o professor não é necessariamente perfeito. O objetivo também é encontrar ordem no conjunto de exemplos, mas de forma diferenciada, que permita uma descrição "genérica" de cada classe apresentada, possibilitando o desenvolvimento de um classificador. O objetivo principal é obter um classificador que "erre" o mínimo possível tendo em vista as limitações impostas pela extração de características e recursos computacionais.

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. **Busca de ordem no conjunto de exemplos**

   Tal busca pode ser realizada empiricamente por um operador humano ou automaticamente por aprendizado computacional.

5. **Avaliação dos resultados**

   Testes devem ser realizados, tanto com dados simulados como com reais. Testes são importantes para inferir se o processo está conseguindo abordar o problema de forma aceitável.

Nas próximas seções apresentamos um breve resumo de técnicas de *Aprendizado Computacional*, muito utilizadas na área de reconhecimento de padrões.

3.1 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* (agrupamento). O objetivo é encontrar alguma ordem no conjunto de exemplos fornecido. Uma abordagem muito utilizada é a baseada em distâncias ou similaridades, que buscam agrupar os vetores de características por algum critério.

Além das etapas básicas já citadas no início deste capítulo, o desenvolvimento de um processo de *clustering* requer:

1. **Escolha da medida de proximidade**

   Para medir o quão "similar" ou "diferente" dois vetores de características são.

2. **Escolha do critério de *clustering***

   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. Avalia-se se as ordens evidenciadas são relevantes e se permitem alguma conclusão a respeito dos dados avaliados.

Um processo de *clustering* depende muito de diversos fatores, dentre eles, os tipos de ordens presentes nos conjuntos avaliados e as formas de disposição das mesmas.

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 em Theodoridis *et al.* [137, Capítulo 11] e nos excelentes artigos de Jain *et al.* [70], Fasulo [45] e Kim *et al.* [73]. 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 [137, Capítulo 12].

- **Algoritmos hierárquicos**

  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 [137, Capítulo 13].

- **Algoritmos baseados em otimização de função de custo**

  Em geral têm um número fixo de agrupamentos e tentam maximizar uma dada função. Quando um máximo local é encontrado o algoritmo pára. Em geral, algoritmos probabilísticos e de Fuzzy são desta categoria [137, Capítulo 14].

- **Outros tipos**

  Existem diversos outros tipos, como algoritmos genéticos [91, 15], estocásticos e baseados em técnicas de transformações morfológicas [137, Capítulo 15].

Dependendo do problema sendo abordado, num primeiro momento, experimentos com *clustering* são necessários para auxiliar na avaliação de um determinado caminho a ser seguido. Tais experimentos permitem que se tenha uma idéia do desempenho que algoritmos supervisionados obteriam. Um exemplo de experimento que realizamos nesta linha é apresentado no Capítulo 8.

3.1.1 Algoritmos Hierárquicos

Os algoritmos hierárquicos são largamente utilizados pela sua simplicidade e velocidade, apesar de nem sempre gerarem resultados excelentes. Nesta seção introduzimos os principais aspectos necessários para a compreensão dos algoritmos hierárquicos utilizados nos experimentos do Capítulo 8.

Algoritmos hierárquicos constroem hierarquias que podem ser visualizadas por árvores correspondentes chamadas dendogramas.

Uma das grandes vantagens desse tipo de algoritmo é o fato dos resultados serem compostos por agrupamentos com estrutura, como é o caso de árvores filogenéticas, em que um grupo é formado por subgrupos.

Existem diversas métricas para *clustering*, algumas inclusive bastante sofisticadas, como mostrado por Wang *et al.* [146]. Dentre as principais métricas utilizadas, podemos citar:

- ***Simple*** – A distância entre dois agrupamentos de pontos é a menor distância entre os pontos de cada agrupamento, dois a dois.

- ***Complete*** – A distância entre dois agrupamentos de pontos é a maior distância entre os pontos de cada agrupamento, dois a dois.

- ***Average*** – A distância entre dois agrupamentos de pontos é a média das distâncias entre os pontos de cada agrupamento, dois a dois.

- ***Centroid*** – A distância entre dois agrupamentos de pontos é a distância dos centróides (aproximado se as distâncias não são euclidianas) dos agrupamentos.
- ***Ward*** – A distância entre dois agrupamentos de pontos é a soma dos quadrados das distâncias entre os pontos de cada agrupamento, dois a dois.

Existem dois tipos de algoritmos:

- **Aglomerativos** – Em cada passo geram um conjunto de agrupamentos menor, resultado da união de elementos do conjunto original.
- **Divisivos** – Em cada passo geram um conjunto de agrupamentos maior, resultado da divisão de elementos do conjunto original.

Independentemente do tipo do algoritmo, em cada passo é feita uma escolha gulosa, por exemplo, para o caso aglomerativo, uma vez que dois elementos são agrupados, eles sempre vão estar no mesmo agrupamento. É essa propriedade que permite que o algoritmo seja veloz, mas algoritmos gulosos nem sempre dão bons resultados, dependendo muito do problema. Ou seja, essa propriedade é a maior virtude, mas também a maior fraqueza dos algoritmos baseados na construção gulosa da hierarquia.

Um algoritmo hierárquico aglomerativo extremamente simples é o seguinte:

1. Inicia, criando um agrupamento para cada elemento da entrada.
2. Encontra o par de agrupamentos que minimiza (ou maximiza) a função dada como critério.
3. Junta os dois agrupamentos em um só.
4. Repete os dois passos anteriores até que se obtenha o número de agrupamentos desejado.

Como pode ser observado, esse tipo de algoritmo é extremamente simples, em geral muito rápido, mas também exige muita memória. A saída do algoritmo é uma árvore (dendograma) em que cada bifurcação representa uma união (ou separação para o tipo divisor) de agrupamentos.

Na Figura 3.2 temos um exemplo de dendograma aplicando o critério *Simple* e a distância euclidiana para os seguintes pontos no plano representados na Figura 3.3:

x_1	0	0
x_2	1	1
x_3	3	1
x_4	2	4
x_5	6	3
x_6	6	6
x_7	5	2
x_8	3	5
x_9	0	2
x_{10}	2	1

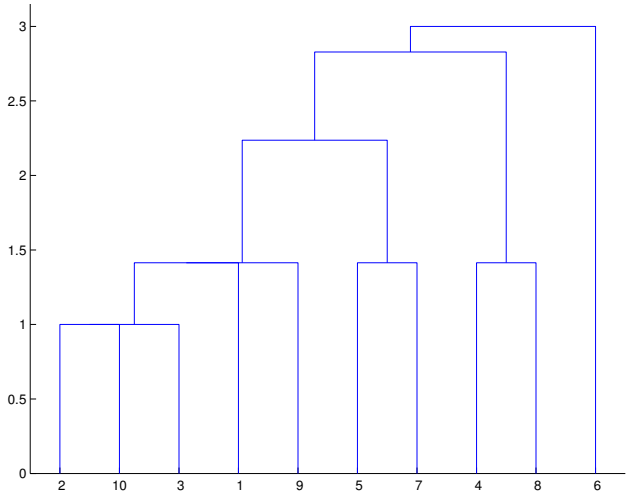


Figura 3.2: Exemplo de dendograma. No eixo x temos o número do ponto e no eixo y a distância de agrupamento a ponto

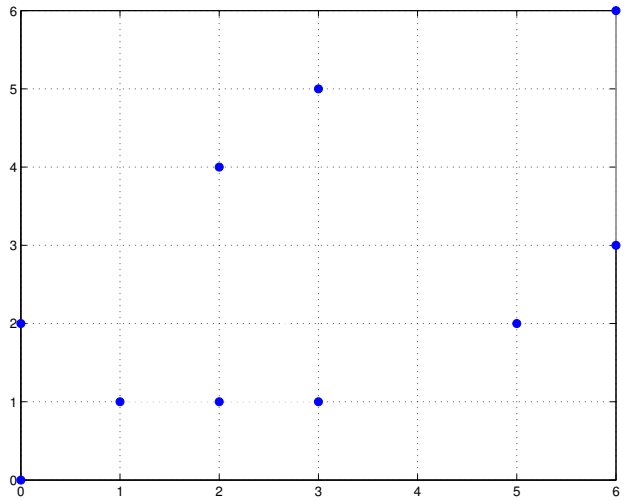


Figura 3.3: Exemplo de pontos no plano

Observe que cada bifurcação representa uma união de agrupamentos. Inicialmente, há 10 agrupamentos com um elemento cada.

Dado um dendograma, podemos representar os agrupamentos finais obtidos por cortes no mesmo. Por exemplo, para o dendograma da Figura 3.2, se realizamos um corte na altura 0.5 vamos obter os 10 agrupamentos originais. Se realizarmos um corte na altura 1.25 vamos obter 8 agrupamentos, na altura 1.75, 4 agrupamentos, na altura 2.5, 3 agrupamentos e na altura 3, 2 agrupamentos.

Dessa forma, é possível analisar os dados de forma eficiente para avaliar e validar os agrupamentos conforme a "forma" com a qual os agrupamentos estão sendo agrupados. Essa é uma característica muito útil de algoritmos hierárquicos.

Existem muitos tipos de algoritmos hierárquicos, mas vamos somente apresentar esse, suficiente para a compreensão dos experimentos realizados no Capítulo 8.

3.2 Aprendizado Computacional Supervisionado

Existem diversas abordagens para aprendizado supervisionado. Ao longo deste trabalho, vamos apresentar somente uma abordagem estatística genérica amplamente utilizada para o projeto automático de classificadores para o reconhecimento de padrões supervisionado. Uma visão mais aprofundada pode ser encontrada em Duda *et al.* [37], Kearns *et al.* [72] e Minka [93].

Informalmente, qualquer processo capaz de "aprender um conceito", a partir de exemplos que o ilustram, é denominado aprendizado[1]. 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 sorteando os alimentos e informando ao aluno "este é doce", "este não é doce". Dessa forma, espera-se que o aluno seja capaz de distinguir sabor doce dos demais sabores de alimentos.

Supondo que o problema de decisão pode ser resolvido de forma estatística, e, entendendo o vetor de característica como uma variável aleatória que segue determinada distribuição conhecida, dado um ponto de teste, um *classificador bayesiano* simplesmente determina qual dentre as classes de classificação tem a maior probabilidade de conter o ponto. Dessa forma, minimiza-se o erro, tendo em vista somente as características conhecidas.

Na Figura 3.4 temos um exemplo de projeto de classificador usando decisão bayesiana para o caso binário e somente uma característica. Em geral, o objetivo é encontrar uma superfície de separação que, de preferência, não seja muito complexa.

Uma das maiores limitações dessa abordagem é a necessidade de conhecer a distribuição de cada classe, em geral não disponível, o que impossibilita a aplicação direta da abordagem acima, conhecida como *abordagem bayesiana*.

As principais técnicas na área buscam contornar tal problema estimando a distribuição desconhecida e em seguida aplicando a abordagem bayesiana [143]. Tal estimação é uma tarefa em geral muito árdua, porque na maioria dos casos poucos exemplos de treinamento

[1] É importante distinguir aprendizado de compreensão e consciência. Apesar do computador ser capaz de "aprender" algo, ele não capaz de compreender o mesmo, muito menos ter consciência daquilo que está fazendo, como mostra Searle [121, Capítulo 2] ao expor sua famosa alegoria do "Quarto Chinês".

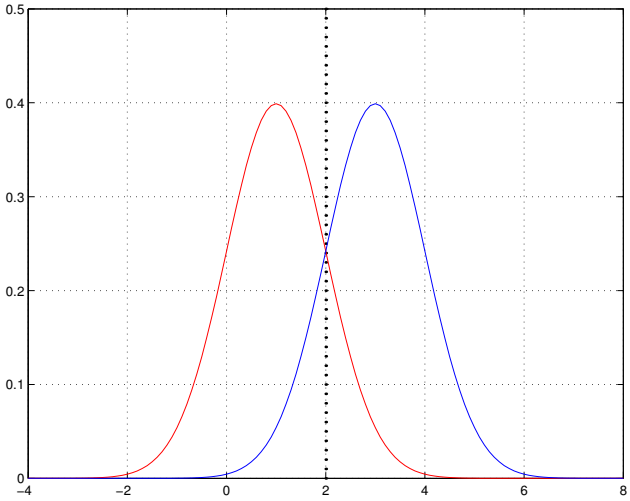


Figura 3.4: Exemplo de projeto de classificador usando decisão bayesiana. No gráfico, temos a densidade de probabilidades de duas classes. A decisão bayesiana consiste em escolher o melhor separador entre as classes que, neste caso, é simplesmente uma linha vertical em $x = 2$, dessa forma, classifica-se tudo à esquerda da reta como vermelho e tudo à direita da reta como azul

estão diponíveis e a distribuição dos exemplos não necessariamente corresponde à distribuição real do conceito que se quer aprender. No Capítulo 4 apresentaremos algumas abordagens clássicas muito utilizadas atualmente.

Bons algoritmos de treinamento conseguem produzir hipóteses cada vez melhores em relação ao conceito alvo conforme o número de exemplos vai aumentando. Ou seja, quanto mais informação sobre o conceito é fornecida, o classificador resultante melhora. Segundo Angluin [5], o número de exemplos em geral necessário para se obter um bom classificador é exponencial em relação ao tamanho do classificador ótimo.

No fundo, o que qualquer algoritmo de aprendizado faz é construir uma superfície de decisão, ou seja, uma partição do domínio em que cada elemento da partição corresponde a algum determinado rótulo. A fronteira entre tais elementos compõe a superfície de decisão. O domínio não precisa ser necessariamente finito.

Ao construir uma superfície de decisão, o que se está fazendo é a escolha de um ponto do *espaço de hipóteses*. Esse espaço é constituído de todas as hipóteses para se classificar o domínio, em que cada hipótese corresponde a um classificador.

O problema de encontrar um classificador dado um espaço de hipóteses pode ser encarado como um problema de otimização combinatória [103], cujo objetivo é encontrar um ponto do espaço de hipóteses que seja ótimo, ou seja, que "erre" o mínimo possível utilizando as características escolhidas.

Encontrar um ponto ótimo raramente é possível, visto que, em geral, os exemplos de treinamento não fornecem informações suficientes para cobrir todos os aspectos do conceito que se quer aprender. Em geral, o que é possível fazer é encontrar um ponto do espaço de hipóteses que seja ótimo em relação aos dados de treinamento. Observe que nem mesmo o "erro" é conhecido. Dado um ponto do espaço de hipóteses e um critério para cálculo de

erro, em geral não é possível determinar qual o seu erro, o que se faz na maioria dos casos é estimar o erro a partir dos exemplos de treinamento fornecidos.

Visando contornar as diversas limitações apresentadas, é necessário introduzir informações úteis que possam ser utilizadas pelos algoritmos de treinamento. O objetivo é sempre reduzir o espaço de hipóteses. Pode-se, por exemplo, introduzir hipóteses a respeito da distribuição das classes que compõem o conceito que se quer aprender. Uma outra forma de introduzir informação é modelando o espaço de hipóteses utilizando alguma linguagem útil para representar conhecimento e o classificador. Tais questões serão abordadas no Capítulo 4. Uma visão mais aprofundada desse assunto pode ser encontrada em Barrera *et al.* [8].

Uma variação interessante de aprendizado aqui apresentado é o aprendizado com um crítico, em que, diferentemente do aprendizado supervisionado, temos um crítico que diz se determinada classificação está correta ou não.

Esse tipo de aprendizado pode ser muito útil em sistemas adaptativos em que o classificador deve se adaptar aos dados que vão sendo classificados. Observe que a "forma" dos dados pode variar com o tempo ou ser sazonal. Da mesma forma, os dados de treinamento podem não ter sido representativos. Técnicas nessa linha são conhecidas como *Reinforcement Learning* (aprendizado por reforço) [133].

Capítulo 4

Projeto de Classificadores

O classificador final obtido pela abordagem estatística para o reconhecimento de padrões como apresentada na Seção 3.2 é baseada em dois elementos principais:

1. **Distribuição de cada classe**

2. **Critério de decisão**

Caso se conheça a distribuição de cada classe, o critério de decisão ótimo é o Bayesiano [37]. O classificador é simplesmente uma implementação eficiente de uma representação das distribuições e do critério de decisão. A implementação do critério de decisão é estritamente trivial.

O grande problema nessa área gira em torno do fato de que raramente se conhecem as distribuições das classes. De certa forma, esse é o único problema que tem de ser resolvido. As técnicas para projeto de classificadores buscam resolver tal questão.

A amostra de treinamento, que é dada como entrada para os algoritmos de treinamento, é um fator importante. Ela oferece uma idéia limitada de como é a distribuição de cada classe. A partir da amostra de treinamento, é possível estimar $p(y|x)$, ou seja, a probabilidade de um determinado exemplo ocorrer dado que se estão sorteando os exemplos de acordo com a distribução da classe x. O grande desafio é obter $p(x|y)$.

Uma das formas de se fazer isso é tentando determinar quão distante uma estimação de $p(x|y)$ está do valor real. Desta forma, fixado um critério de decisão, é possível avaliar o erro de um operador que utiliza a estimação em questão. Determinar o quão distante a estimação de $p(x|y)$ está do valor real é uma tarefa complicada, mas é possível estimar o erro do operador dadas as distribuições estimadas. Para tal, exitem alguns métodos, como utilizar parte do conjunto de treinamento para a estimação de $p(x|y)$ e outra parte para a estimação do erro.

Observando que a estimação de $p(x|y)$ pode ser representada como uma parametrização de um modelo utilizado para representar distribuições, então o objetivo do treinamento é encontrar uma parametrização que tenha um erro pequeno.

Dessa forma, cada ponto do espaço de hipóteses corresponde a uma parametrização possível do modelo. Pode-se então encarar o problema de estimação como problema de busca, que consiste em encontrar um ponto do espaço de hipóteses que tenha um erro pequeno de acordo com os critérios adotados. Dizemos que um determinado ponto é ótimo se

a ele é associado o menor erro possível em relação a todos os outros pontos do espaço de hipóteses.

Uma busca exaustiva não pode ser, em geral, implementada, visto que o espaço de hipóteses na maioria das vezes é infinito ou muito grande, o que torna tal método inviável.

Existem diversos métodos genéricos que permitem a busca em espaços de hipóteses grandes, como é o caso dos métodos de Monte Carlo [47] e técnicas de otimização combinatória [103], mas esses métodos muitas vezes não dão resultados suficientemente bons.

Projetar classificadores, em geral, não é tarefa simples, exigindo um bom operador humano e muito conhecimento a respeito do problema. Além disso, dependendo do número de variáveis livres, é necessário a utilização de técnicas automáticas.

Por melhores que sejam os algoritmos de estimação e as linguagens para a representação do classificador, em geral os resultados obtidos não são muito animadores. Tal fato acontece atualmente por três motivos principais:

1. **Conjunto de treinamento pequeno**

   Em geral estão disponíveis poucos exemplos para treinamento, como é o caso de amostras de pacientes com câncer. Técnicas para o treinamento com poucos dados não são eficientes a menos que se tenha muita informação útil a respeito do problema. Em geral, o que se quer é, a partir de um pequeno conjunto de exemplos, descobrir informações a respeito do problema, tarefa bem complicada.

2. **Dificuldade para compreender a "essência" do problema**

   Há atualmente uma tendência forte de simplesmente aplicar os algoritmos de treinamento simplesmente por aplicar. Em geral não se discute as capacidades, limitações ou mesmo razões históricas de um algoritmo. Além disso, olham-se tais algoritmos como uma caixa preta. A análise visual dos dados, por exemplo, ainda revela muita informação útil que nem sempre é aproveitada.

3. **Limitações computacionais**

   Os computadores atuais são extremamente rápidos e têm grande quantidade de memória comparados com os de vinte anos atrás, mas tal poder não confere aos algoritmos de treinamento a capacidade de escolher pontos no espaço de hipóteses de forma razoável devido à natureza não-linear da maioria dos problemas, o que dificulta muito a capacidade de generalização. A solução de muitos problemas está intimamente ligada com problemas inviáveis ou mesmo não computáveis.

Tendo em vista essas diversas limitações, é necessário então reduzir o espaço de busca, uma das formas de se fazer isso é introduzindo informações verdadeiras ou mesmo conjecturas durante a modelagem do problema. Existem diversas formas de se fazer isso. Entre elas:

- **Escolha de um critério de decisão**

  Importante observar que quando a distribuição das classes é estimada, o critério Bayesiano não é necessariamente ótimo, visto que alguma propriedade a respeito das distribuições pode ser conhecida, mas não necessariamente ter sido captada pelo algoritmo de estimação, desta forma, tal propriedade deve ser implementada no critério de decisão.

- **Escolha da representação das distribuições**

  A escolha da forma da representação é muito importante. A introdução de informação depende muito das linguagens e estruturas de dados escolhidas. Uma boa escolha pode facilitar em muito a introdução de informações conhecidas. Além disso, a linguagem determina o número de parâmetros que podem ser manipulados, determinando assim o tamanho do espaço de hipóteses.

- **Escolha do algoritmo de busca**

  O algoritmo de busca pode ser escolhido de acordo com alguma propriedade do espaço de hipóteses previamente conhecida, o que pode facilitar bastante a busca.

Ao longo deste capítulo, apresentamos uma breve análise de algumas técnicas utilizadas no projeto automático de classificadores que visam diminuir a complexidade da busca do classificador, pela introdução de conhecimento a respeito do problema abordado.

Importante observar que existem diversas abordagens em que não fica claro a dicotomia entre representação das distribuições e o critério de decisão, como é o caso de algumas abordagens em redes neurais [60], **SVM** (*Support Vector Machines*) [28], ou mesmo algumas abordagens baseadas em Casamento (*template matching*), como é o caso de se procurar genes usando o **BLAST** [4, 3]. De qualquer forma, mesmo nessas abordagens essa dicotomia existe, o que acontece, no fundo, é que essas abordagens simplesmente usam restrições do espaço de hipóteses.

Uma explicação mais detalhada de diversas abordagens pode ser encontrada em Jain *et al.* [69], Voit [145], Fishman [47], Grant *et al.* [54] e Durbin *et al.* [38].

Na Seção 4.1 apresentamos uma visão geral a respeito de alguns aspectos de modelagem que leva em conta a introdução de conhecimento. Na Seção 4.2 apresentamos algumas técnicas relacionadas principalmente com a escolha do algoritmo de busca. Nas Seções 4.3 e 4.4 apresentamos algumas formas de se representar distribuições. Na Seção 4.5 mostramos alguns aspectos históricos a respeito de tais representações.

4.1 Modelando o Projeto do Classificador a Partir de Informação *a priori*

A essência de qualquer método 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**

São utilizadas diversas abordagens, como as apresentadas ao longo deste capítulo, justamente porque cada uma oferece diferentes graus de facilidade para a generalização, inclusão e representação de informação dependendo do problema que está sendo abordado.

A escolha do método que será utilizado para abordar um determinado problema é passo importantíssimo e pré-requisito básico para que um bom classificador seja produzido. Tal escolha é fruto de uma modelagem que deve levar em consideração diversos aspectos a respeito do problema.

A escolha apropriada da representação do espaço de hipóteses é também muito importante. Cada tipo de representação define a forma do espaço de hipóteses. Em geral, quando se utiliza uma determinada representação, espera-se que seja suficientemente boa para representar o fenômeno estudado sem ser excessivamente custosa do ponto de vista computacional.

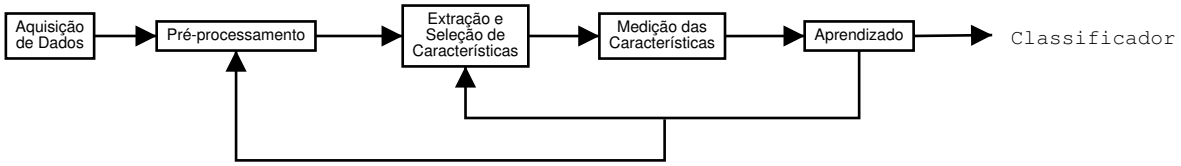


Figura 4.1: Representação esquemática de passos para o projeto de classificadores

Durante a modelagem do problema, é necessário levar em conta diversos aspectos a seu respeito. Dentre eles, a quantidade e qualidade dos exemplos e informações disponíveis sobre o problema. Na Figura 4.1 temos uma representação esquemática de passos para o projeto de classificadores. A modelagem do problema está fortemente ligada a cada um dos itens apresentados, que abrangem desde a escolha dos dados até a escolha do método de aprendizado utilizado.

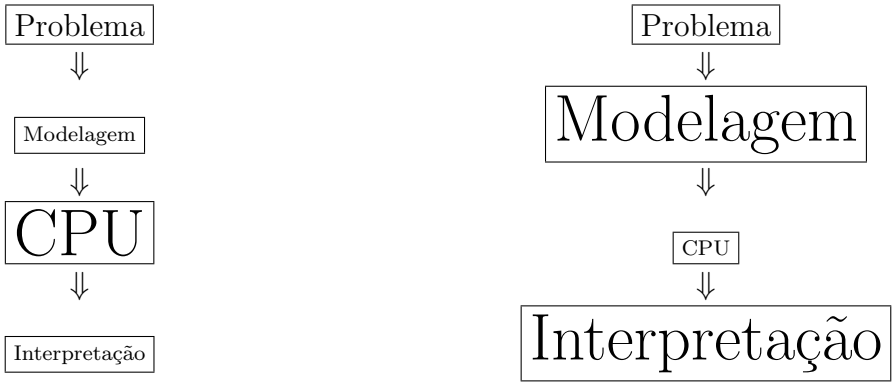


Figura 4.2: Duas formas para se abordar um problema

Em geral, o resultado final obtido está intimamente ligado à qualidade da modelagem. Na Figura 4.2 temos duas formas utilizadas para se modelar um problema. Em geral, quando é dedicado muito tempo para a realização de uma boa modelagem e análise dos resultados obtidos, obtêm-se bons resultados sem a necessidade de utilização de muito tempo de CPU, mas, em geral, o que se observa é a tentativa de se resolver o problema sem muita modelagem mas com muita CPU. Tal abordagem, em geral, leva a resultados muito ruins. Grande parte dos problemas enfrentados atualmente na área com certeza não serão resolvidos com muita CPU, mas sim com muita modelagem e idéias simples, mas inteligentes.

Existem diversos algoritmos que realizam a estimação de um classificador a partir de uma representação escolhida. Não vamos, nesta dissertação, descrever nenhum algoritmo de treinamento, vamos nos concentrar nas próximas seções em outros aspectos necessários para uma boa modelagem.

Uma das formas mais eficientes de tentar restringir o espaço de hipóteses é utilizando conhecimento a respeito do problema abordado. Para tal, é necessária uma representação adequada do que se quer introduzir.

Por exemplo, se sabemos que um grande número de éxons possui um determinado padrão conhecido e queremos introduzir essa informação, será mais fácil se estivermos utilizando algoritmos baseados em **HMM** [112] ou gramáticas [48] do que redes neurais [60]. Além disso, gramáticas e árvores [101] representam mais facilmente famílias de seqüências com tamanho variável do que uma rede neural, por exemplo.

Em contrapartida, se sabemos que uma determinada função que queremos aprender é crescente, esse tipo de informação é mais facilmente introduzido utilizando a abordagem baseada no **ISI** [140] ou redes neurais [60].

Importante observar que há limitações computacionais quanto ao tipo de conhecimento que pode ser introduzido em cada abordagem. No caso de gramáticas, não é possível representar, utilizando gramáticas regulares, repetições do tipo xy, em que $x = y$ podendo x ser de tamanho qualquer. Em contrapartida, tal expressão é facilmente representável utilizando gramáticas livres de contexto.

O conhecimento que é introduzido não é necessariamente verdadeiro ou completo, ou seja, é possível introduzir conjecturas ou mesmo conhecimentos que não explicam completamente algum fenômeno do problema abordado, mas, nesse caso, há um sério risco da restrição não ser "boa".

Suponha que uma conjectura errada seja usada para se realizar uma restrição no espaço de hipóteses. Por exemplo, genes humanos têm tamanho menor que 10.000 pares de bases. Isso reduz em muito o espaço de hipóteses, mas impede que genes maiores que 10.000 pares de bases sejam reconhecidos. Mesmo assim, isso pode ser bom ou ruim, dependendo do que se quer encontrar. Um exemplo de restrição com certeza ruim é supor que íntrons são menores que éxons, o que em geral não acontece.

Pode-se ganhar ou perder, dependendo do conhecimento que está sendo introduzido, cabendo ao operador humano conduzir a abordagem de restrição durante a modelagem do problema. Em geral, sem uma boa restrição é impossível resolver qualquer problema útil de reconhecimento de padrões. Devroye *et al.* [33, Teorema 7.2] mostra que não existe regra suficientemente boa para se treinar qualquer tipo de distribuição de forma eficiente com poucos exemplos, o que reforça a importância da redução do espaço de hipóteses.

4.2 Restrições do Espaço de Hipóteses

A grande questão envolvida na restrição do espaço de hipóteses é que a restrição pode impedir que o classificador ótimo seja encontrado, independentemente dos recursos disponíveis.

Uma restrição óbvia do espaço de hipóteses implícita quando se utiliza um computador para abordar algum problema é o fato de que somente variáveis discretas podem ser analisadas. Outra restrição é que o classificador deve ser representado de forma computável.

Do ponto de vista computacional, podemos definir um espaço de hipóteses genérico H como todos os algoritmos computáveis que podem ser utilizados para descrever um conceito como definido na Seção 3.2.

A grande pergunta é: Vale a pena? Quando se opta por utilizar um computador para projetar e representar um classificador, imediatamente uma grande restrição é imposta ao espaço de hipóteses. Pode ser que nem seja mais possível obter um classificador ótimo, mas tais perdas precisam ser compensadas pelas vantagens que um computador oferece, como é o caso da velocidade com que ele é capaz de fazer determinadas análises.

Quando procurar um classificador ótimo num espaço de hipóteses em que o mesmo faz parte é inviável, abrir mão desse ótimo pode ser interessante desde que uma restrição do espaço de hipóteses torne viável a busca de um classificador sub-ótimo (ou seja, ótimo da restrição mas sub-ótimo do espaço original) que não seja tão distante (do ponto de vista de erro) do classificador ótimo do espaço de hipóteses original.

Há restrições que são ótimas, ou seja, que restringem o espaço de busca, mas continuam contendo um classificador ótimo em relação ao espaço original, mas que não são viáveis do ponto de vista computacional. Podemos definir, por exemplo, uma restrição do espaço de busca como o espaço que contém classificadores cuja representação são programas que param. Tal restrição é, por definição, ótima, mas não é computável.

Encontrar uma "boa" restrição em geral não é tarefa simples, mas é imprescindível, afinal, são grandes as já conhecidas limitações de memória e velocidade dos nossos computadores atuais. É necessário restringir o espaço de hipóteses de forma a diminuir o espaço de busca.

Uma limitação bem simples é a escolha de alguma abordagem como as apresentadas ao longo deste capítulo. Quando se escolhe uma abordagem, além das propriedades já discutidas, está se fazendo uma restrição no espaço de hipóteses. Por exemplo, se optamos por modelar um conjunto de palavras por expressões regulares (usando gramáticas regulares), só é necessário avaliar expressões que são regulares. Podemos imediatamente descartar todos os outros tipos de expressões, o que diminui em muito o espaço de busca, mas isso pode ser bom ou ruim dependendo do problema abordado.

Caso o conceito ótimo h' do novo espaço restrito H' não esteja muito "distante" do ótimo h do espaço original H (ou seja, o erro aumenta "só um pouco"), então temos uma restrição "boa", desde que ela tenha custo computacional baixo. Todavia, a restrição pode ser tão "ruim" que não vale a pena utilizá-la. Como já dito, uma restrição é ideal quando o ótimo do espaço original se encontra no espaço restrito (ou seja, desta forma não se adiciona erro e é realizado um corte no espaço de busca).

Na Figura 4.3 temos uma visualização do efeito causado por restrições. O esquema da direita representa a situação ótima, enquanto o da esquerda representa uma situação típica em que o ótimo da restrição não é o ótimo do espaço original.

Para problemas reais a experiência tem mostrado que é muito difícil encontrar restrições ideais, mas em geral é possível achar restrições boas, ou seja, restrições que abrem mão da possibilidade de obter o classificador ótimo, mas que são computacionalmente baratas e que permitem a obtenção de classificadores em geral melhores, dadas as restrições do problema abordado.

Uma forma muito utilizada para a restrição do espaço de hipóteses é a seleção de características, em que um subconjunto de todas as características disponíveis é escolhido. Dessa forma, somente as características selecionadas são utilizadas no projeto do classificador. Ob-

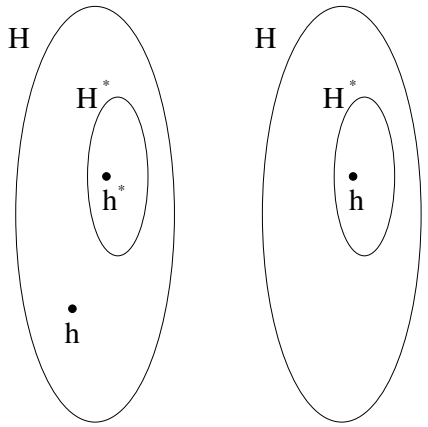


Figura 4.3: Esquemas do efeito causado por restrições. Na restrição ótima, o ótimo h do espaço H está contido na restrição H^*, enquanto que numa restrição típica, o ótimo h^* de H^* não é o ótimo de h

viamente há uma redução da informação disponível. As técnicas nessa linha buscam eliminar as características que oferecem informações muito redundantes em relação ao conjunto escolhido. Apesar da diminuição da quantidade de informação, há também uma redução da dimensionalidade do problema, o que pode transformar um problema antes intratável em outro bastante semelhante e viável. Técnicas de redução de dimensionalidade podem ser utilizadas para gerar um novo conjunto de novas características, a partir das características disponíveis, mas têm a desvantagem de não oferecerem uma forma fácil de interpretar as regras geradas pelo estimador do classificador. Técnicas baseadas em análise de componentes são um exemplo de redutores de dimensionalidade que apresentam as características acima, como é o caso do **PCA** e do **NLCA** [37, Seção 10.13] que são vastamente utilizados na área.

Uma forma também muito eficiente para a restrição do espaço de hipóteses são as técnicas multi-resolução e multi-escala. A idéia principal é manipular a quantidade de informação disponível durante a análise dos dados na tentativa de melhorar a precisão e capacidade de generalização sem a necessidade de mais exemplos.

4.2.1 Multi-resolução

Técnicas multi-resolução manipulam globalmente a quantidade de informação disponível. A idéia principal é criar regras para a redução de informação em diversos níveis e aplicá-las em toda a informação disponível. Utiliza-se então a melhor resolução (quantidade de informação) para cada subtarefa do processo de reconhecimento.

Por exemplo, na Figura 4.4 temos uma imagem de satélite. Caso se queira reconhecer as ruas, pode-se utilizar uma resolução menor, mas, caso se queira reconhecer baratas nas ruas, é necessário utilizar uma resolução muito maior. Uma abordagem multi-resolução, por exemplo, seria primeiro detectar onde estão as ruas utilizando uma resolução pequena. Com uma resolução um pouco maior, tentaria-se encontrar locais em que talvez existam baratas, e, finalmente, com uma resolução bem maior, explorariam-se somente os locais apontados

em busca de baratas.

É óbvio que se poderia fazer a busca utilizando a resolução máxima disponível, mas isso pode ser bem mais custoso computacionalmente e até mesmo inviável, porque quanto maior a resolução, mais detalhes e variações são verificadas e, portanto, mais exemplos de treinamento são necessários. Como em geral os exemplos para treinamento são poucos, então é interessante utilizar técnicas que minimizem a necessidade dos mesmos.

Figura 4.4: Imagem de satélite

No caso de imagens, é razoavelmente simples diminuir a quantidade de informação disponível, bastando cortar as imagens como exemplificado na Figura 4.5. Existem outras técnicas mais sofisticadas, como algumas baseadas em transformadas de imagens, mas a que mostramos já é bem eficiente.

Nem sempre é possível determinar uma boa função de redução de resolução, principalmente quando o que está sendo representado é mais abstrato. Uma visão multi-resolução de textos escritos, por exemplo, seria observar o texto com pouca informação (um índice), passa-se a observar as idéias gerais dos capítulos, as idéias gerais das seções, as idéias dos parágrafos e, por fim, o texto em si (resolução máxima) mas, dado um texto, não é possível escrever um algoritmo para a redução de informação como descrito.

4.2.2 Multi-escala

Diferentemente das técnicas multi-resolução, técnicas multi-escala manipulam localmente a quantidade de informação disponível. A idéia principal é criar regras para a redução

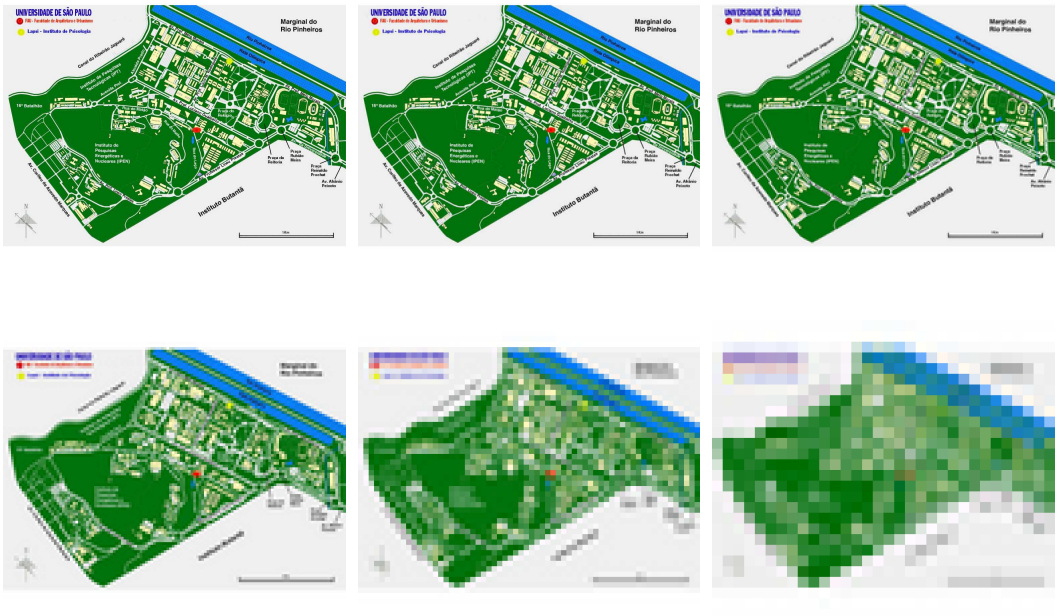


Figura 4.5: Mapa da USP em diversas resoluções

de informação em diversos níveis e aplicá-las somente na informação que o algoritmo está manipulando em dado momento. Dessa forma, pode-se utilizar a melhor resolução local para cada decisão local do algoritmo.

Diferentemente da multi-resolução, em que diferentes técnicas são aplicadas globalmente em cada resolução e depois há uma avaliação em conjunto dos resultados com eventual análise local, a multi-escala sempre tem disponível a resolução máxima, mas pode aplicar diferentes técnicas para cada quantidade de informação local que é consolidada localmente. Somente o resultado da consolidação é avaliado globalmente.

Para o caso de imagens, podemos definir a quantidade de informação como uma pequena janela (ou máscara) em que somente o que está "dentro" da janela é observado de cada vez (isso caracteriza o aspecto local). Analisa-se toda a imagem ao se transladar a janela para cada posição possível da mesma, realiza-se então uma consolidação global.

A definição de janelas muito grandes implica que há muitas variáveis livres, ou seja, há a necessidade de muitos exemplos. Quando os exemplos não estão disponíveis, o que acontece é um fenômeno conhecido como *overfitting*, em que o algoritmo simplesmente memoriza os exemplos de treinamento, a precisão é alta, mas a generalização é muito baixa. Como os exemplos de treinamento são pequenos então acontece um fenômeno descrito como curva em 'U'. Essa curva é a curva de erro. O erro começa grande para janelas muito pequenas, diminui com um pequeno aumento das janelas, mas quando as janelas aumentam muito, acontece o *overfitting* e o erro aumenta muito porque não há generalização. Dessa forma, o aspecto do gráfico é uma letra 'U'.

Com a multi-escala, esse tipo de efeito não acontece, porque se ajusta o tamanho da janela para cada pequena parte dos dados dos exemplos, dessa forma, utiliza-se janela grande

somente quando há dados disponíveis, senão, janelas menores são utilizadas. Ou seja, utiliza-se uma escala mais "apropriada" para cada caso localmente definido. A curva de erro, no final, é, do ponto de vista teórico, sempre decrescente.

Essa propriedade é muito importante, pois permite a criação de algoritmos que analisam a informação disponível.

Note que, para o exemplo do texto apresentado anteriormente, é mais fácil definir uma função que manipule a quantidade de informação. Poderíamos definir como a unidade básica como uma letra (ou seja, janela de tamanho 1), a segunda unidade como uma palavra (janela do tamanho da palavra), em seguida, uma frase, um parágrafo, uma seção, um capítulo. Como a quantidade de informação necessária é definida dinamicamente pelo algoritmo, não acontece o mesmo efeito que no exemplo apresentado para multi-resolução, em que é necessário definir como vai ser a redução de informação antes de se aplicar uma determinada técnica.

Técnicas híbridas são muito utilizadas também. A grande vantagem é que, se as funções de redução de informação forem bem construídas, há redução drástica do espaço de busca, bem como do número de exemplos necessários para treinamento. Note que é possível introduzir informação a respeito do problema ao se construir tais funções.

Tais funções podem ser visualizadas como partições do conjunto de todas as combinações possíveis de entrada. Criam-se, então, classes de equivalência. Se essas classes estão bem criadas, então as técnicas serão muito bem sucedidas, caso contrário, pode inclusive haver uma piora no resultado final.

Por exemplo, para o caso de imagens, se o objetivo é reconhecer números, quando diminuímos a resolução, imagens de números que antes eram diferentes por causa de pormenores mínimos (como até mesmo ruídos ou diferenças de alguns poucos pixels) podem se tornar iguais, ou seja, diz-se que tais imagens estão na mesma classe de equivalência. Tal informação pode facilitar em muito a classificação, principalmente a generalização, desde que a redução de informação não comprometa a precisão.

Observe que definir classes de equivalência para texto é até simples, como apresentamos, mas analisar o significado de determinada construção estar na mesma classe de equivalência é um problema tão complexo quanto definir um redutor de resolução para textos, o que acaba impedindo a consolidação global.

4.3 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* [17] e uma de suas principais extensões conhecida como **HMM** [110, 75]. O objetivo principal destes modelos é fornecer meios para modelar eventos que são fisicamente observáveis.

4.3.1 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. Dessa 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, temos 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 4.6 temos um exemplo de cadeia de Markov com algumas transições de estados.

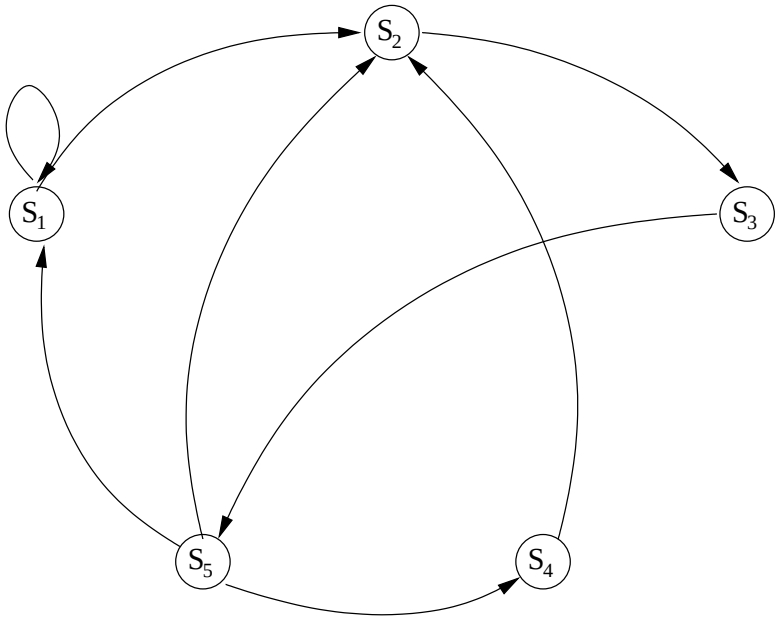


Figura 4.6: Exemplo de cadeia de Markov com 5 estados

Podemos chamar o processo estocástico (cujo desenvolvimento é governado por leis probabilísticas) 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. Uma observação importante, modelos de ordem d podem ser reduzidos a modelos de ordem 1 aumentando-se exponencialmente o número de estados.

Exemplo 4.1 Modelando o clima
Podemos utilizar um Modelo de Markov com 3 estados para modelar o clima de forma simplista. Considerando 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 e estimar uma matriz de transição 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 4.7.

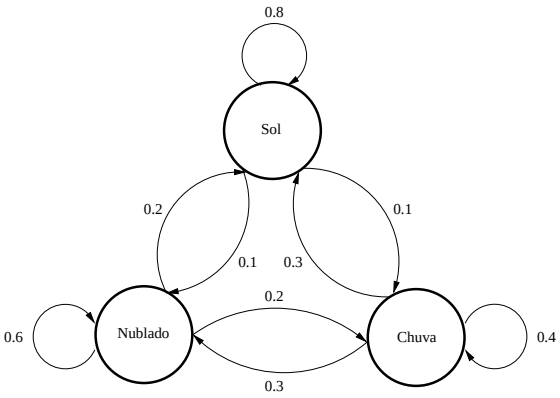


Figura 4.7: Exemplo de Cadeia de Markov para modelar o clima

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 (Sol) do que se mude do estado 3 para o estado 2 (Nublado) em 1 passo.

Exemplo 4.2 Modelando a distribuição de bases de regiões codificadoras
Cadeias de Markov são muito utilizadas na tentativa de se estimar a distribuição de bases de uma região codificadora.

A idéia básica é criar um conjunto de estados em que cada estado representa uma combinação de bases. Por exemplo, quando queremos avaliar a distribuição de triplas de pares de bases, criamos uma Cadeia de Markov com $4^3 = 64$ estados, um para cada tripla. Treinam-se então duas cadeias. Uma que utiliza somente exemplos de regiões codificadoras e outra que utiliza somente exemplos de regiões não codificadoras. No final, obtém-se a distribuição para cada caso, supondo que a próxima tripla só depende da atual.

Observe que a suposição acima não é verdadeira, ou seja, funciona como restrição do espaço de hipóteses que visa abrir mão de alguns casos especiais na tentativa de dar ênfase aos casos mais freqüentes (ou seja, supõe-se que, em geral, a próxima tripla só depende da anterior). Tal escolha pode favorecer ou piorar o desempenho da estimação, conforme o problema e dados disponíveis.

O treinamento das cadeias simplesmente constitui em estimar os valores de transição utilizando alguma técnica clássica de estimação. Após esses passos, podemos determinar

se uma seqüência desconhecia se "parece" mais com uma região codificadora ou com uma região não codificadora.

Esse tipo de informação obtida a partir da estimação de Cadeias de Markov é muito utilizado por *softwares* de predição gênica.

4.3.2 Modelos Escondidos de Markov (HMM)

HMM é uma extensão do modelo de Cadeias de Markov. As bases desse modelo foram publicadas por Baum e seus colegas [112] no final dos anos 1960 e início dos anos 1970.

Um Modelo Escondido de Markov – **HMM** (*hidden Markov model*) é um processo estocástico, dito oculto, imerso em outro processo estocástico, dito observável, em que 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.

No fundo, **HMM** é como Cadeias de Markov (parte observável) imersas em uma outra Cadeia de Markov principal (parte oculta), em que cada cadeia imersa corresponde a um estado da cadeia principal.

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 d, caso o processo de decisão do próximo estado só dependa dos últimos d 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 essa 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. Ou seja, é possível entender tal modelo como conjunto de distribuições, uma distribuição correspondente a cada estado da cadeia principal. Cada distribuição corresponde a algo fisicamente observável e o processo oculto é o processo de escolha da seqüência de distribuições.

Exemplo 4.3 Modelando o lançamento de moedas – cara × coroa
Suponha que, dado um conjunto de moedas, lança-se somente uma de cada vez e informa-se o resultado. O número de moedas não é conhecido e não é informada qual moeda foi lançada. As moedas escolhidas não são necessariamente idôneas.

Como não sabemos o número de moedas, podemos inicialmente supor que há só uma moeda sendo lançada e podemos facilmente modelar o problema utilizando uma cadeia de Markov com dois estados como na Figura 4.8, em que um estado corresponde a cara e outro a coroa. Basta então estimar as probabilidades de transição, que correspondem ao viés da moeda.

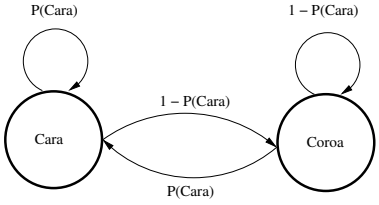
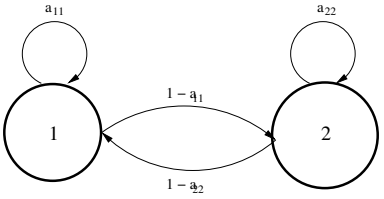


Figura 4.8: Modelando o problema da moeda utilizando cadeia de Markov

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. Dessa forma, podemos modelar o problema utilizando um **HMM** de ordem 1 com dois estados como exemplificado na Figura 4.9, 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.



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

Figura 4.9: Modelando o problema da moeda utilizando **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

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 lançada. Desta forma, podemos modelar o problema utilizando um **HMM** de ordem 1 com três estados como na Figura 4.10, de forma análoga ao caso anterior.

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

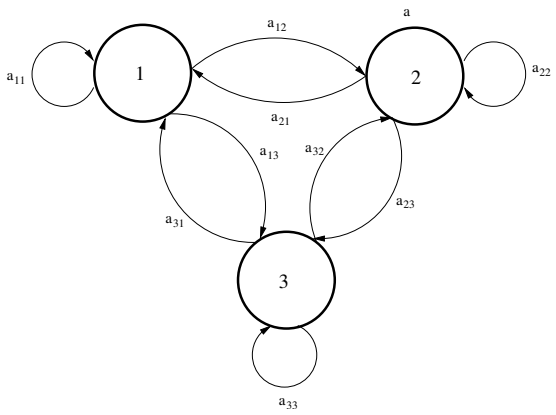


Figura 4.10: Modelando o problema da moeda utilizando **HMM** com 3 estados. Para cada estado temos uma probabilidade de "cara" ser emitida

seja feita de forma completamente aleatória, o primeiro modelo conseguirá resultados muitos bons independente do número de moedas.

Observe também que caso o número real de moedas seja menor que o número de moedas escolhido para modelar o problema, o resultado obtido também será bom, dependendo do caso pode-se até detectar tal fato analisando o resultado da estimação, o grande inconveniente é que mais exemplos são necessários para a estimação. Da mesma forma, o caso em que se supõe que existem menos moedas do que na realidade pode resultar numa estimação melhor do fenômeno, principalmente quando poucos exemplos de treinamento estão disponíveis.

Exemplo 4.4 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 estocástico associado à urna atual. Repete-se o processo de sorteio e escolha de uma nova urna.

Uma modelagem óbvia para o problema é a em que cada estado representa uma urna e cada cor de bola associada a essa urna é representada pela probabilidade da cor ser emitida no estado. A escolha da urna é ditada pela matriz de transição do **HMM**. O grande problema é que não se conhece o número de urnas.

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 o algoritmo de Viterbi, baseado em programação dinâmica [112], mas a complexidade do problema depende muito dos critérios utilizados.

Vamos dar ênfase na abordagem do último problema.

Resolvendo o último problema

Podemos dividir, didaticamente, 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 hipóteses utilizando informação biológica ou mesmo algumas técnicas como as baseadas em reticulados, que buscam construir uma arquitetura. Uma técnica muito utilizada na tentativa de encontrar o número de estados de um **HMM** é treinar diversos modelos variando o número de estados e verificando quais deles se adaptam melhor ao conjunto de treinamento. Tal tipo de técnica pode ajudar quando não se tem uma boa hipótese a respeito do número de variáveis livres que devem ser utilizadas.

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 modelos. Além disso, é necessário supor que o padrão que queremos modelar é computável pelo modelo.

Exemplo 4.5 Modelando *domains*, *motifs* e alinhamentos

A idéia básica para ambos os casos é modelar inserções, deleções e *matches* (casamentos ou emparelhamentos) utilizando estados para cada situação.

Na Figura 4.11 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 letra. Em estados de inserção temos valores proporcionais à probabilidade de continuar inserindo e nos estados de deleção temos o número do passo.

Na Figura 4.12 temos alguns modelos clássicos utilizados por algumas aplicações importantes em Biologia Computacional.

Um domínio muito conhecido é o $SH2$. Na Figura 4.13 temos um exemplo de arquitetura que modela tal domínio com técnicas parecidas com as do exemplo anterior, só que usando aminoácidos.

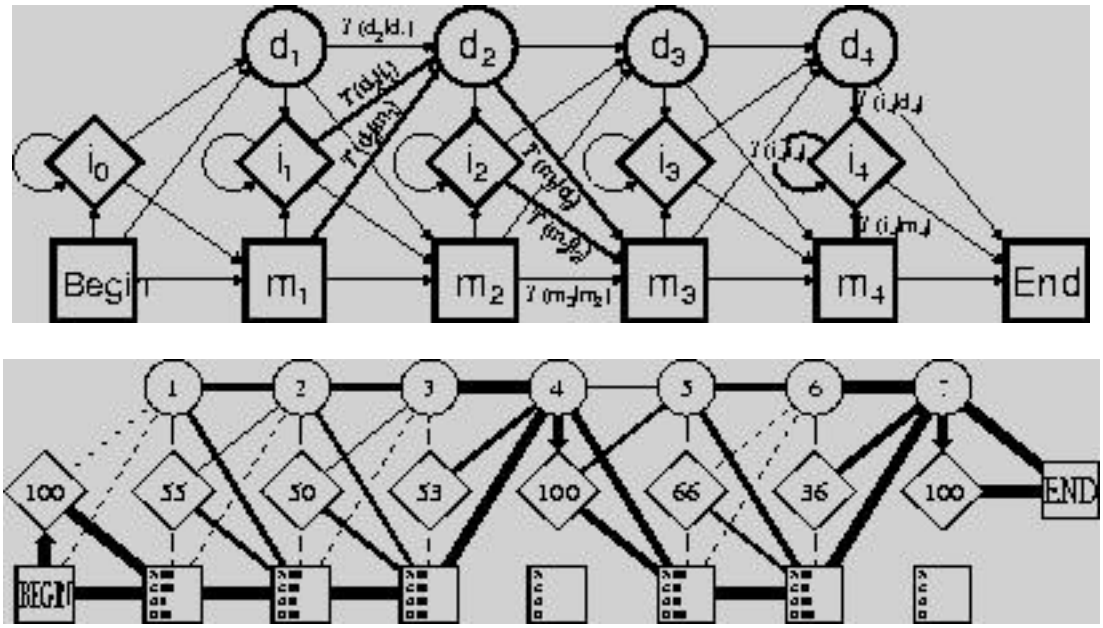


Figura 4.11: Exemplo de modelo extraído de [67] para modelar alinhamentos e exemplo de inferência do modelo

4.3.3 Modelos de Markov Escondidos Generalizados

GHMM é o Modelo de Markov Escondido Generalizado, ou seja, em cada estado, em vez 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 podem utilizar outros modelos para se estimar a distribuição de probabilidade de cada estado. Por exemplo, pode-se utilizar uma cadeia de Markov para representar a probabilidade de um determinado tipo de palavra em um estado e uma rede neural para representar outros tipos de palavras em outro estado, e assim por diante.

Obtém-se assim uma forma de variar a quantidade de informação necessária para se treinar cada estado, ou seja, utiliza-se mais informação onde é necessário mais informação e vice-versa. Tal propriedade é muito importante e é discutida mais genericamente na Seção 4.2.

4.4 Linguagens Formais e Modelos Estocásticos

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

4.4.1 Conceitos Básicos

Seja Σ um conjunto, e seja Σ^* o conjunto de seqüências finitas de elementos de Σ. Se $\sigma_1, \sigma_2, \ldots, \sigma_n$ são elementos de Σ, para algum $n \geq 1$, a seqüência $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ é deno-

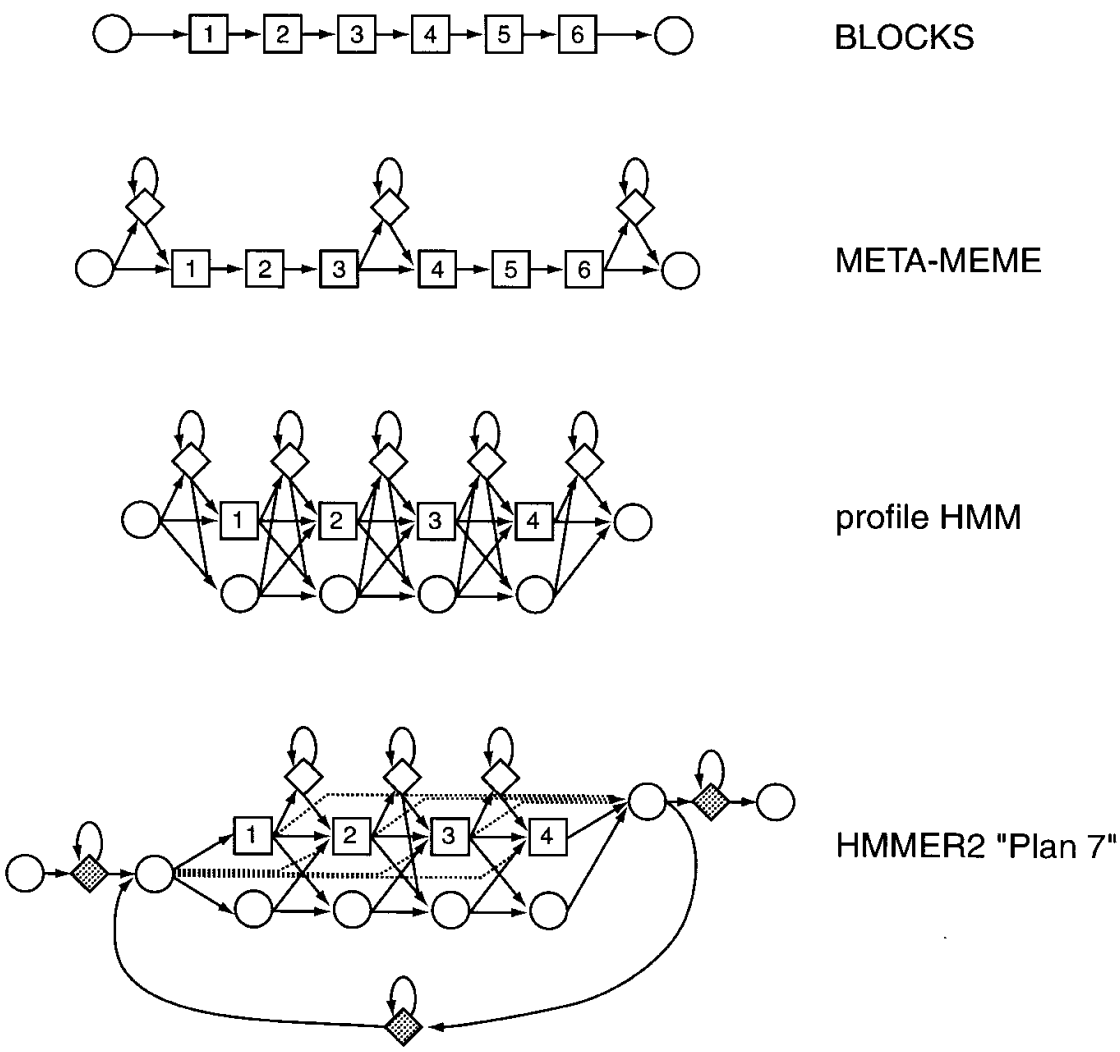


Figura 4.12: Exemplo de modelos extraídos de [39] utilizados em diversas aplicações em Biologia Computacional

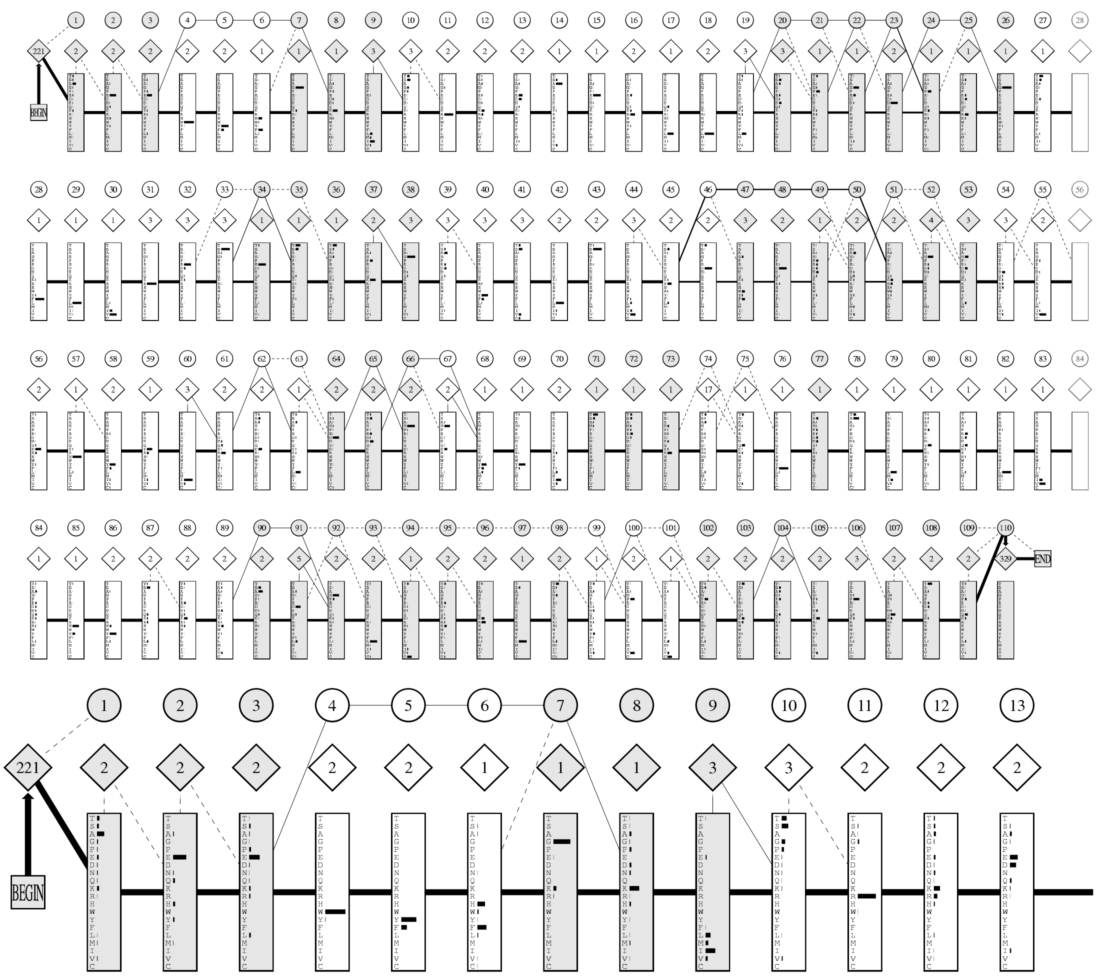


Figura 4.13: Exemplo de modelo para modelar o domínio $SH2$ extraído de [67]. A seção inicial está apresentada em baixo do modelo completo utilizando uma resolução maior. Os estados não sombreados representam elementos da estrutura secundária

minada $\sigma_1\sigma_2\ldots\sigma_n$, $n \geq 1$. A seqüência vazia () é denominada λ. Os elementos de Σ^* são denominados *palavras*, sendo λ a *palavra vazia*. Σ é denominado *alfabeto* e seus elementos, *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 64 palavras (todas as palavras de tamanho 3). L_2 é uma linguagem com um número infinito de palavras (todas as palavras que contêm exatamente duas letras 'A').

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

4.4.2 Gramáticas

Uma gramática G é uma quádrupla que 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 é denotada por (V, Σ, P, S). O alfabeto Σ é denominado *alfabeto terminal*, cujos símbolos são denominados *símbolos terminais*, enquanto o conjunto $N = V \setminus \Sigma$ é denominado *alfabeto não terminal*, cujos símbolos são denominados *símbolos não terminais*. Os elementos do conjunto P são denominados *produções*. Uma produção (s, s') de P é denotada por $s \longrightarrow s'$. s e s' são denominados *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 denominado *comprimento da derivação* d e é denotado por $|d|$.

Uma derivação $d' = (s_0, s_1)$ de comprimento 1 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 Lewis *et al.* [81].

A classe de linguagens recursivamente enumeráveis é muito grande [102, 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 é possível computar eficientemente linguagens que estão além da classe PTIME [24, Capítulo 36], ou seja, problemas decidíveis utilizando algum modelo clássico de computação como Máquinas de Turing determinísticas utilizando tempo no máximo polinomial.

Tendo em mente essa limitação, vamos introduzir a hierarquia de Chomsky [22] que restringe o conjunto de produções das gramáticas. O objetivo principal é obter 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, gasta-se espaço linear em relação ao tamanho da palavra [126, Capítulo 3]. O tempo necessário para decidir se s pertence à G pode vir a ser exponencial, o que torna inviável sua utilização com os equipamentos disponíveis atualmente.

Para as linguagens livres de contexto e regulares é possível decidir se uma dada palavra s pertencem às linguagens eficientemente. Este é o principal argumento para a utilização de linguagens do tipo 2 e 3 apesar de serem modelos bem simples em relação aos fenômenos que já se conhecem como os que Searls *et al.* [123, 124] apresenta. Tudo isso motivou Searls a estudar uma classe de gramática intermediária chamada *string variable grammar* [122, 125, 123] que não é tão difícil de reconhecer quanto as linguagens sensíveis ao contexto mas é mais genérica que as livres de contexto. Mesmo utilizando modelos restritos, o treinamento de gramáticas mais genéricas que as regulares é tarefa bem complicada.

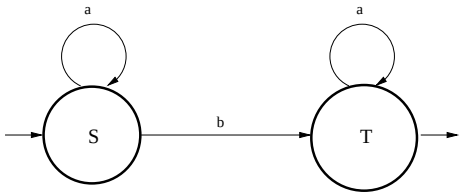


Figura 4.14: Exemplo de autômato para reconhecer a^*ba^*

Na Figura 4.14 temos um exemplo de autômato para a linguagem regular a^*ba^* e alfabeto $\Sigma = \{a, b\}$, ou seja, todas as palavras que contêm exatamente um único 'b'.

A gramática regular equivalente é $G = \{V, \Sigma, P, S\}$ com $V = \{S, T, a, b\}$, $S = \{S\}$ e

$$P = \begin{array}{rcl} S & \rightarrow & aS \\ S & \rightarrow & b \\ S & \rightarrow & bT \\ T & \rightarrow & aT \\ T & \rightarrow & a \end{array}$$

4.4.3 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 que diversas mutações podem ocorrer no código genético e que diversos erros podem ocorrer no processo de leitura das moléculas, como por exemplo erros de seqüenciamento e montagem. Tudo isso sugere que uma abordagem estocástica é adequada para o problema, tendo em vista as limitações computacionais atuais.

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

Uma gramática estocástica G é uma quíntupla que 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 probabilidades de produções que têm 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 4.4.2 foram apresentados 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 4.4.2, em geral se representam os classificadores usando gramáticas estocásticas livres de contexto e regulares.

4.4.4 Treinamento de Gramáticas Estocásticas

Para treinar as gramáticas descritas na Seção 4.4.3, são necessários dois passos principais: encontrar 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 mostra-se 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 (mas não suficiente) para o algoritmo gerar bons resultados. 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* [90] e o *Tree Grammar EM* [79], entre outros.

Observe que os métodos descritos nesta seção nada mais são do que restrição do espaço de hipóteses, como vai ser discutido na próxima seção. Existem métodos mais abrangentes, mas raramente são aplicados no contexto de gramáticas.

4.5 Aspectos históricos de HMM e Gramáticas

Observando as definições de **HMM** e gramáticas apresentadas anteriormente é possível perceber que **HMM** equivale computacionalmente ao modelo de gramática estocástica regular. A maioria dos artigos na área prefere se referir a **HMM** em vez de gramáticas regulares principalmente por questões de notação e bagagem estatística sólida do **HMM**.

Ambos os modelos foram desenvolvidos de forma independente, **HMM** tem berço na estatística com ênfase na modelagem de processos estocásticos e gramática regular tem origem na área de computação para o processamento de texto.

Uma das propriedades mais marcantes das gramáticas regulares é o fato de terem sempre uma expressão regular associada (e vice-versa), além disso, dado um autômato não-determinístico é possível determinar um autômato determinístico equivalente.

A generalização de gramáticas regulares para sua versão estocástica abre mão dessas qualidades importantíssimas, porque, no fundo, são um caso particular dos $K-\Sigma$-autômatos apresentados por Eilenberg [40].

Eilenberg mostra que tal tipo de autômato não tem mais uma expressão regular associada e que nem sempre é possível transformar um autômato não-determinístico em determinístico sem perda de informação.

Resumindo, a generalização de gramáticas apresentada, apesar de formalmente bem definida, acaba alienando os propósitos iniciais do significado de existir do modelo de gramática.

Por outro lado, **HMM** já nasceu para modelar justamente processos estocásticos, sendo bem mais fácil representar conhecimento para esse tipo de problema utilizando **HMM**. Não só isso, é mais fácil entender que algumas coisas não fazem sentido, como a maioria dos métodos para treinar gramáticas estocásticas cuja idéia é apresentada na Seção 4.4.4. É **impossível** estimar as produções, como em geral os autores de tais métodos gostariam de

fazer, porque elas correspondem à camada escondida de **HMM** e, portanto, não pode ser estimada diretamente. O máximo possível é reduzir o espaço de hipóteses, mas nem sempre a redução utilizada é "boa".

Na prática, entretanto, muitos artigos se referem à **HMM** mas utilizam máquinas de estados numa forma "dualizada" o que dificulta a compreensão para quem não está acostumado, porque conjuntos de produções equivalem a estados em **HMM** e conjuntos de estados de gramáticas equivalem a distribuições dos estados de **HMM**. Como os modelos são equivalentes, então os algoritmos de treinamento são exatamente igualmente equivalentes (ao contrário do que acreditam alguns defensores de gramáticas), mas é mais "fácil" para os humanos entenderem um **HMM** e, para uma máquina, mais fácil representar o conceito como máquina de estados, visto que é o que elas são[1].

Existem gramáticas mais poderosas, mas, em geral, a estimação é muito custosa porque o espaço de hipóteses associado é muito maior. Então, em geral, é necessária a realização da poda do espaço de busca utilizando conhecimento *a priori*. A menos que conhecimento *a priori* "forte" que exija gramáticas mais genéricas que as regulares seja conhecido, é fortemente recomendável a utilização de **HMM** em vez de gramáticas estocásticas mais poderosas para modelar o problema.

[1] Alguns teóricos poderiam dizer que há modelos mais poderosos do ponto de vista computacional, como as gramáticas livres de contexto, entre outras, mas é importante lembrar que a memória do computador é finita.

Capítulo 5

Buscando Genes por Técnicas de Reconhecimento de Padrões

Atualmente estão disponíveis muitas técnicas para a busca de genes. Nesta seção vamos apresentar um breve resumo das principais técnicas atualmente utilizadas. Tais metodologias podem ser usadas tanto no caso não supervisionado [13] como no caso supervisionado.

5.1 Casamento

A grande maioria das técnicas utilizadas atualmente para se encontrar genes são baseadas em casamento.

Em geral, define-se um conjunto de dados, como um conjunto de genes, e procura-se por fragmentos similares a esse conjunto utilizando técnicas de casamento aproximado, como, por exemplo, o **BLAST** [4, 3].

Uma técnica muito utilizada atualmente nessa linha é a comparação de genomas, em que se seqüencia o genoma de uma espécie e utiliza-se o conhecimento a respeito de genes de outros genomas para buscar eventuais novos genes. Uma versão mais sofisticada está em encontrar partes dos genomas similares, o que pode indicar uma eventual conservação.

Existem outras técnicas, como as que utilizam partes de seqüências expressas, como **EST** ou **SAGE**. A idéia é procurar pelas seqüências expressas no genoma porque com certeza as seqüências fazem parte de genes. Uma região em que muitos **EST**s ou **SAGE**s estão presentes pode indicar a existência de um gene. Diferentemente das técnicas anteriores, é recomendável nesse caso a utilização de técnicas de casamento exato ou mesmo com no máximo k erros, como as baseadas em árvores de sufixo, o que permite que grande quantidade de **EST**s e grandes regiões de um genoma sejam casadas em poucos minutos (com o **BLAST**, por exemplo, atualmente essa tarefa demora alguns dias ou mesmo meses dependendo do poder computacional disponível e da quantidade de dados a serem casados).

A área de Comparação e Casamento Aproximado de Seqüências é importantíssima na Biologia Molecular Computacional [58, Parte III]. O objetivo principal desta área é, além de permitir a busca de elementos similares, permitir uma boa modelagem para os processos de mutação e para a questão de presença de erros nos dados.

Uma outra forma de se buscar genes é através de expressões regulares. A partir de um banco de proteínas [153], escolhe-se um conjunto delas e se escrevem expressões regulares

que representam sítios ativos ou outras partes das proteínas. Utilizam-se então as expressões regulares geradas para fazer buscas de partes similares aos exemplos de treinamentos nos genomas seqüenciados.

Existem diversas formas de se comparar seqüências, *subseqüências* e *subpalavras*[149]. É importante observar a diferença entre subseqüência e subpalavra. Os caracteres numa subpalavra precisam ser um "pedaço" contínuo da seqüência original, enquanto que os caracteres em uma subseqüência não precisam. Por exemplo, "ata" é subpalavra e subseqüência de "ccatag", enquanto que "cag" é subseqüência, mas não é subpalavra de "ccatag".

Conceitos como similaridade e distância são critérios muito utilizados no aprendizado computacional, tanto não supervisionado como supervisionado, para avaliar seqüências genéticas. Tais conceitos serão apresentados nas próximas páginas.

5.1.1 Similaridade

Similaridade entre duas seqüências pode ser entendida como quanto "uma seqüência se parece com a outra". Para determinar a similaridade entre duas seqüências, utiliza-se algum *critério* para seu *alinhamento*.

O alinhamento entre duas seqüências é, intuitivamente, uma forma de dispor lado a lado os caracteres de duas seqüências dadas. Além do pareamento de caracteres, o alinhamento permite também o pareamento de caracteres e espaços inseridos em uma seqüência. O critério, em geral, determina pesos para erros, acertos e inserção de espaços nos pareamentos. 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. 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, pela 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

Dessa forma, podemos definir similaridade como 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 $p : \Sigma \times \Sigma \rightarrow \mathbb{R}$ em que Σ é o alfabeto utilizado nas palavras. Para penalizarmos a insersão de um caractere espaço $\sqcup \notin \Sigma$, construímos $p' : \{\Sigma \cup \{\sqcup\}\} \times \{\Sigma \cup \{\sqcup\}\} \rightarrow \mathbb{R}$ com as mesmas 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.

5.1.2 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* [89]. A idéia principal é transformar (editar) uma palavra na outra pelo 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. Atribuem-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 um caractere $a \in \Sigma$ por outro $b \in \Sigma, a \neq b$
2. Inserção ou deleção de um caractere arbitrário $a \in \Sigma$

Para cada tipo de operação acima, associamos um custo, dessa forma, podemos definir o custo de um conjunto de operações, denominado *cost*, como 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. Uma linha amplamente utilizada é a de funções baseadas em matriz **PAM**, **BLOCKS** [64, 63], entre outras.

5.1.3 Casamento aproximado

Diversas técnicas de casamento exato utilizadas para calcular similaridade e distância têm se mostrado inviáveis computacionalmente. Encontrar um alinhamento ótimo de múltiplas seqüências, por exemplo, é um problema difícil computacionalmente [58, 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. Dentre os mais utilizados, podemos citar os programas **BLAST** [4, 3] e **FASTA** [83, 104, 105].

5.2 Técnicas estatísticas

Diferentemente das técnicas da seção anterior, que são extremamente precisas, mas não têm uma grande capacidade de generalização, técnicas baseadas em aprendizado computacional buscam generalizar melhor o conjunto de exemplos.

Existe uma grande variedade de *softwares* que utilizam grande variedade de técnicas na tentativa de reconhecer genes. Ao todo, são 28 os principais *softwares* para predição de genes atualmente utilizados [164]. Além disso, há na literatura diversas abordagens genéricas ou extremamente específicas, como as apresentadas por Henderson *et al.* [61], Pizzi *et al.* [108, 107], Sandberg *et al.* [120], Campbell *et al.* [20], Tabaska *et al.* [134], Mironov *et al.* [94], Thanara *et al.* [135], Wang *et al.* [87, 147, 148], Knudsen [74] entre outros. Na tabela a seguir, temos uma lista com os principais *softwares* atualmente utilizados na área para a busca de genes.

Aat [66]	GeneSplicer [106]	EcoParse [76]	Fex [131]
Gap 3 [155]	GeneID [57]	GeneMark [14, 85]	GeneModeler [46]
GeneParser [128]	GeneParser2 [129]	GeneParser3 [129]	Genie [62]
GenLang [34]	GenScan [19, 77]	GenViewer [92]	Glimmer [119]
Grail [141]	Grail 2 [154]	Great [51]	Héxon / Fgeneh [130]
Morgan [118]	Mzef [158]	ORFgene [114]	Procrustes [50]
Sorfind [68]	Veil [76]	Xgrail [155]	Xpound [138]

Existem diversos artigos comparando técnicas e *softwares* para predição de genes, entre eles os de Burset *et al.* [86], Claverie [23], Mural [99] e Werner [150]. Ao longo deste trabalho não realizamos comparações de algoritmos. Foi dada ênfase ao entendimento de técnicas utilizadas em algoritmos que permitem uma modelagem ampla de diversos aspectos a respeito de busca de genes.

O **GenScan** é exemplo de *software* nessa linha muito utilizado atualmente. Na Figura 5.1 temos exemplos de três diferentes **GHMM**s que podem ser utilizados para a busca de genes. O último **GHMM** é o utilizado no **GenScan** e busca modelar de forma simples todas as informações sobre gene descritas na Seção 2.2. Para tal, são utilizados diversos artifícios em cada estado, como a mistura de cadeias de Markov com redes neurais para estimar a distribuição dos códons de éxons e íntrons, por exemplo.

A idéia básica é utilizar **GHMM** como uma camada multi-escala. Na tentativa de melhorar a capacidade de generalização, é necessário introduzir informação a respeito do problema, em geral, introduzem-se informações conhecidas a respeito dos genes, como as descritas na Seção 2.2.

Para tal, definem-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 grande maioria dos *softwares* de sucesso que utilizam técnicas estatísticas é como o **GenScan**. Em geral se destacam pela simplicidade. O treinamento é feito, em geral, somente com exemplos positivos (ou seja, regiões de genes) e utilizando técnicas básicas de estimação. O objetivo final é obter classificadores que devolvem algo como a verossimilhança de uma determinada região ser gene e também uma sugestão da estrutura do gene.

Uma grande limitação desse tipo de técnica é que, caso haja mais de um gene na seqüência fornecida para o algoritmo, os resultados podem ser péssimos.

Resumos interessantes a respeito das principais técnicas utilizadas por diversos *softwares* de predição de genes podem ser encontrados em Bishop *et al.* [12, Capítulo 11], Mount [97, Capítulo 8], Baldi *et al.* [7] e Baxevanis *et al.* [11].

5.3 Precisão × Generalização

Uma das questões mais clássicas da área de reconhecimento de padrões é a questão da precisão *versus* generalização. O objetivo é maximizar a precisão e a generalização, mas, em geral, quanto mais preciso é um algoritmo, menor sua capacidade de generalização e vice-versa.

Para o caso binário, precisão pode ser facilmente obtida "decorando-se" o conjunto de treinamento, mas em geral esse tipo de classificador não serve para muita coisa porque o número de falsos negativos tende a ser muito grande. Da mesma forma, generalização pode ser obtida fazendo com que o classificador simplesmente considere quase tudo como determinado objeto que se quer generalizar, mas, fazer isso é muito ruim porque o número de falsos positivos aumenta muito.

Um algoritmo com alta taxa de acerto, ou seja, alta precisão e generalização é, em geral, um grande desafio.

Na Figura 5.2 temos um esquema de como os algoritmos lidam com os conjuntos de exemplos. Do lado esquerdo, temos algoritmos que "decoram" o conjunto de exemplo e têm uma pequena capacidade de generalização devido à falta de conhecimento a respeito do problema. Do lado direito, temos os algoritmos que buscam generalizar o conjunto inteiro de treinamento, como é o caso do **GenScan**, a precisão é baixa também por causa da falta de conhecimento a respeito do problema.

No centro, há modelos intermediários, que buscam agrupar o conjunto de exemplos utilizando algumas regras. Pode ser uma simples regra matemática como a maioria dos algoritmos de *clustering* ou, de preferência, alguma regra com algum significado para o problema. Pode-se, por exemplo, tentar agrupar um conjunto de genes de acordo com seu papel (função) ou mesmo similaridade e treinar classificadores para cada agrupamento gerado. O objetivo é balancear precisão e generalização de forma a obter um resultado final melhor.

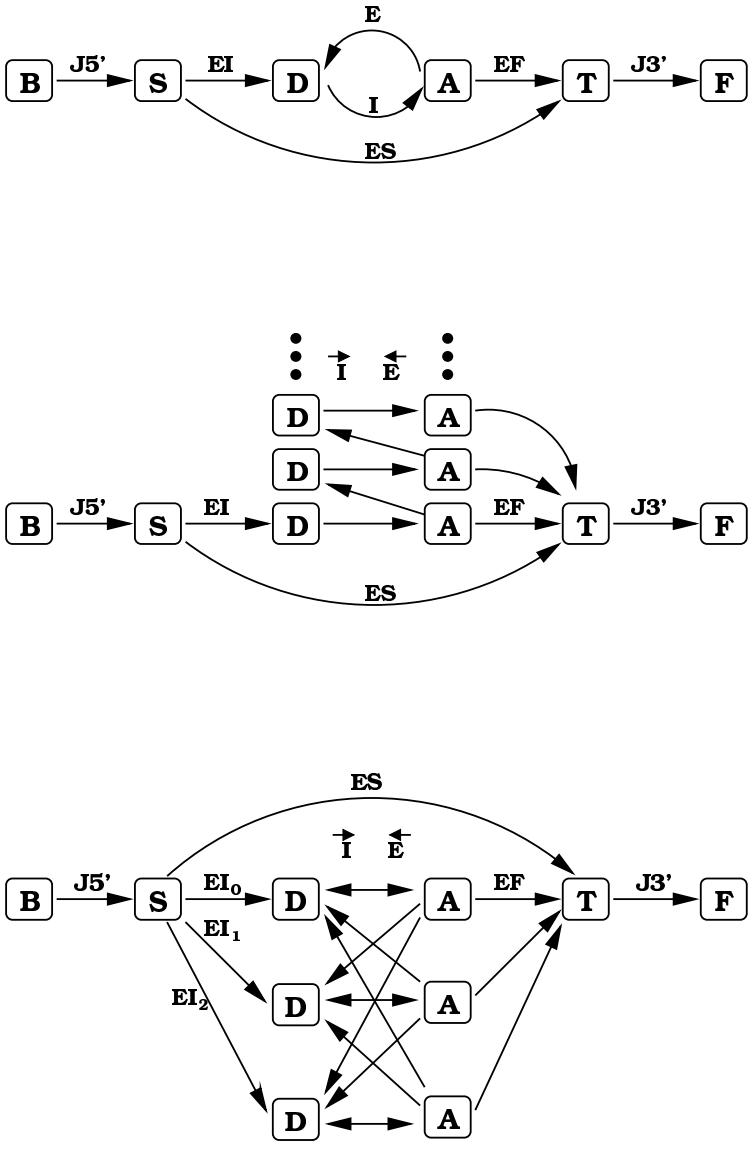


Figura 5.1: Exemplos de modelos baseados em **GHMM** utilizados na predição de genes extraídos de [77]

O caso de busca de genes tem algumas particularidades interessantes. Por exemplo, algoritmos com alta precisão e baixa capacidade de generalização são muito úteis para achar genes que se expressam "bastante", ou seja, o gene é transcrito pelas células com uma freqüência suficiente para ser detectado durante os experimentos com seqüenciamento.

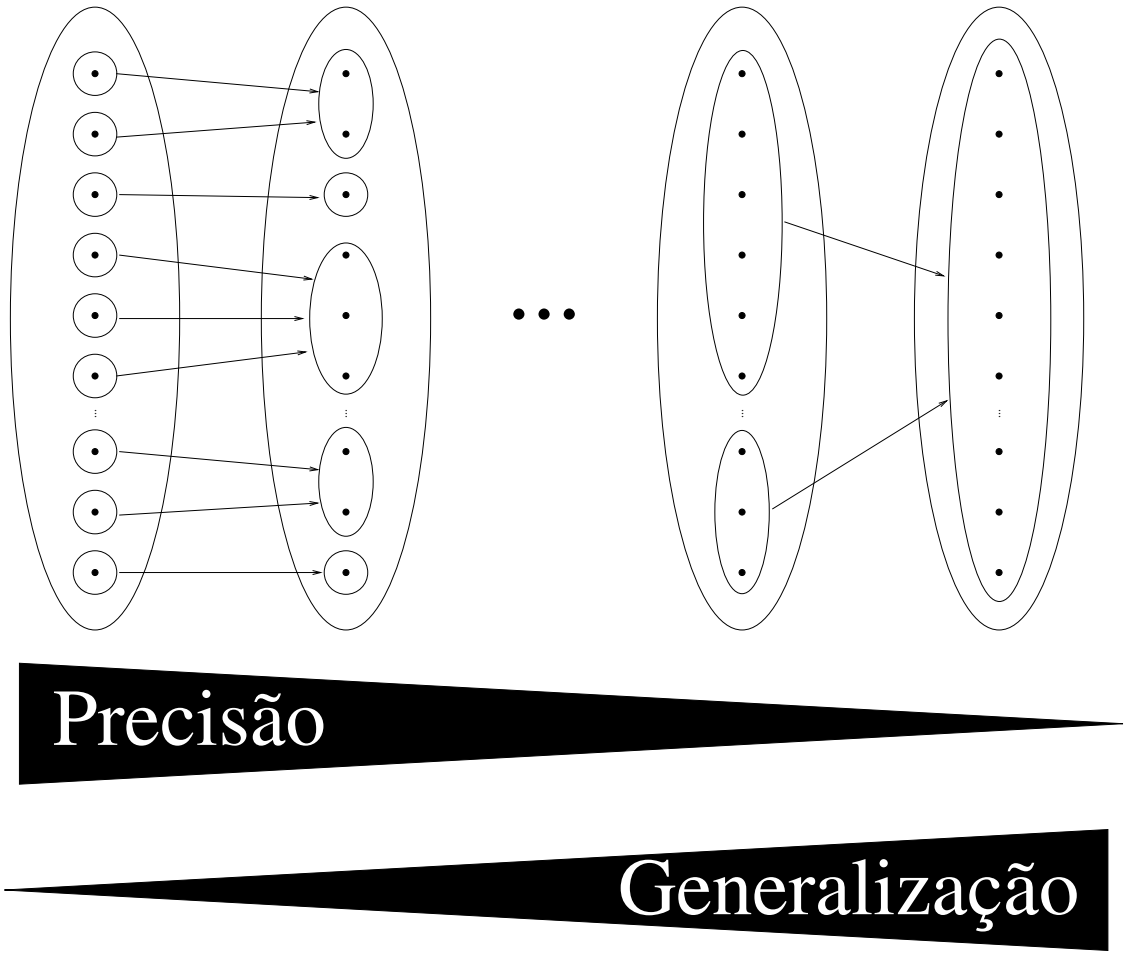


Figura 5.2: Precisão × Generalização

No entanto, para os genes "raros", aqueles que raramente são transcritos pelas células, são necessárias técnicas com maior capacidade de generalização, mesmo que a precisão não seja muito boa.

Observe que esse é um exemplo de introdução de conhecimento útil para a solução do problema. Utiliza-se menos esforço para os casos mais "fáceis" e reservam-se mais recursos para os casos mais complicados. É um exemplo raro de restrição no espaço de hipóteses que com certeza é ótima.

Nos Capítulos 6 e 7 apresentamos de forma informal alguns aspectos teóricos necessários para a compreensão de novas características propostas no Capítulo 8. Tais características visam permitir o desenvolvimento de novos algoritmos para a busca de genes raros.

Capítulo 6

Representação *Chaos Game*

Na tentativa de obtermos alguma informação de assinatura gênica, estamos utilizando uma técnica de organização e visualização das tabelas de freqüências de subpalavras de seqüências de **DNA** baseadas em "Chaos Game Representation" (**CGR**) [101, 31]. Nas próximas seções discutiremos um pouco a respeito de alguns aspectos teóricos e práticos dessa representação.

Iniciaremos com uma breve explicação informal de alguns aspectos da Teoria do Caos para logo em seguida explicarmos o *Chaos Game*. Tais conhecimentos são necessários para uma compreensão mais aprofundada de alguns aspectos da **CGR** que é apresentada logo em seguida.

6.1 Caos

No final do século XIX, a maioria dos cientistas acreditava que praticamente todos os fenômenos naturais poderiam ser modelados por equações lineares, como as equações do movimento exatas e as equações da termodinâmica. Na época, equações não-lineares eram consideradas muito complexas para serem resolvidas e, devido à natureza aparentemente caótica dos fenômenos físicos associados, em geral, evitava-se estudar sistemas não-lineares. Muitas equações não-lineares eram simplesmente "linearizadas", ignorando-se então pequenas oscilações e mudanças. "Assim como o mundo era um mecanismo de relojoaria para o século XVIII, ele foi um mundo linear para o século XIX e maior parte do século XX" [21].

6.1.1 Sistemas não-lineares

Diferentemente de equações lineares, em que pequenas mudanças produzem pequenos efeitos e grandes mudanças produzem grandes efeitos, em sistemas não-lineares pequenas mudanças podem ter efeitos enormes (e vice-versa). Equações não-lineares determinísticas simples podem produzir uma riqueza e variedade de comportamentos surpreendente.

Exemplos simples de sistemas não-lineares podem ser construídos utilizando uma técnica denominada iteração. Por exemplo, se $f(x) = 7x$, a iteração consiste em aplicar a fórmula várias vezes, ou seja, uma iteração de 1 passo equivale a $f(f(x)) = 7^2x$, uma iteração de 2 passos, $f(f(f(x))) = 7^3x$, e assim por diante.

Se visualizarmos a variável x como uma linha de números, a operação $x \rightarrow k \times x$, em que k é constante, mapeia cada número em outro número da linha.

Um exemplo clássico e muito simples de iteração não-linear e que é de alta complexidade é o mapeamento:

$$x \rightarrow k \times x(1-x), 0 \leq x \leq 1$$

Esse mapeamento é conhecido como "mapeamento logístico" e pode ser utilizado com diversas finalidades, dentre elas modelar o crescimento de populações.

É bem fácil calcular mapeamentos para alguns pontos, como o exemplo a seguir para $k = 3$, temos um mapeamento entre 0 e 0.75.

$$\begin{array}{lcll} 0 & \rightarrow & 3 \times 0(1-0) & = & 0 \\ 0.1 & \rightarrow & 3 \times 0.1(1-0.1) & = & 0.27 \\ 0.2 & \rightarrow & 3 \times 0.2(1-0.2) & = & 0.48 \\ 0.3 & \rightarrow & 3 \times 0.3(1-0.3) & = & 0.63 \\ 0.4 & \rightarrow & 3 \times 0.4(1-0.4) & = & 0.72 \\ 0.5 & \rightarrow & 3 \times 0.5(1-0.5) & = & 0.75 \\ 0.6 & \rightarrow & 3 \times 0.6(1-0.6) & = & 0.72 \\ 0.7 & \rightarrow & 3 \times 0.7(1-0.7) & = & 0.63 \\ 0.8 & \rightarrow & 3 \times 0.8(1-0.8) & = & 0.48 \\ 0.9 & \rightarrow & 3 \times 0.9(1-0.9) & = & 0.27 \\ 1 & \rightarrow & 3 \times 1(1-1) & = & 0 \end{array}$$

Observe que pontos entre 0 e 0.5 são mapeados para pontos entre 0 e 0.75 e pontos entre 0.5 e 1 são mapeados para pontos entre 0.75 e 0 de forma simétrica. Na figura 6.1 temos uma visualização desse efeito. Observe que o mapeamento dobra sobre si mesmo ao longo dos passos. A iteração deste mapeamento resulta em repetidas operações de estender e dobrar. Este tipo de iteração é também conhecido como "transformação do padeiro" por dar a idéia de se estar amassando e dobrando uma massa.

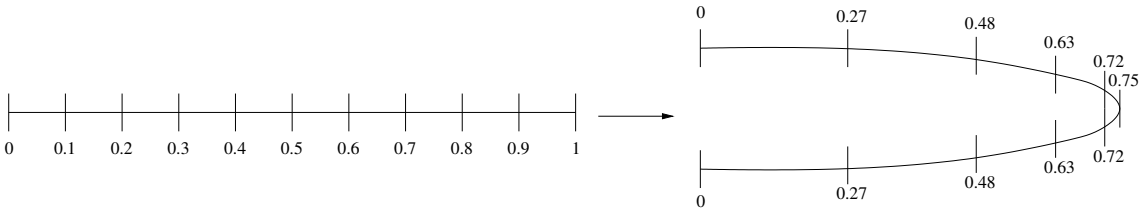


Figura 6.1: Transformação do padeiro

À medida que os passos se dão, fica impossível predizer a posição de um determinado ponto após muitos passos. Isso acontece devido a limitações computacionais. Em geral, a precisão dos computadores para ponto flutuante é muito pequena, mesmo para um sistema simples como o apresentado acima. Arredondamentos acabam acontecendo o que impossibilita o cálculo da posição de um ponto após muitas iterações. Entretanto, é possível utilizar bibliotecas de precisão infinita, mas os números ficariam tão grandes que não caberiam na memória. Mesmo que tivéssemos memória infinita como numa máquina de Turing, o tempo

que tais cálculos levariam seria proibitivo. Este é um belo exemplo da complexidade que sistemas não-lineares podem alcançar.

6.1.2 Sistemas Caóticos

Comportamentos complexos e aparentemente caóticos (como o que apresentamos na seção anterior) podem dar origem a estruturas ordenadas e padrões belos e sutis. O comportamento de um sistema caótico não é simplesmente aleatório, mas exibe um nível mais profundo de ordem padronizada.

Podemos definir um sistema caótico como um sistema em que a capacidade de previsão é pequena, como no caso do mapeamento logístico, também conhecido como transformação do padeiro. Tal transformação é um protótipo de processos não-lineares, altamente complexos e imprevisíveis, conhecidos tecnicamente como caos[1]. Por ser imprevisível, parece aleatório, mas não o é, visto que é um evento determinístico do ponto de vista de Física.

Um outro exemplo de processo caótico é o lançamento de uma moedinha. Podemos prever que ela vai cair, podemos até arriscar dizendo que ela vai cair com uma das faces voltadas para cima, mas é muito, mas muito difícil prever qual vai ser a face que ficará voltada para cima. Ou seja, é algo considerado imprevisível e, portanto, aleatório, tanto que exemplos com moedas são largamente utilizados em estatística. Para a teoria do Caos, no entanto, tal evento não é aleatório.

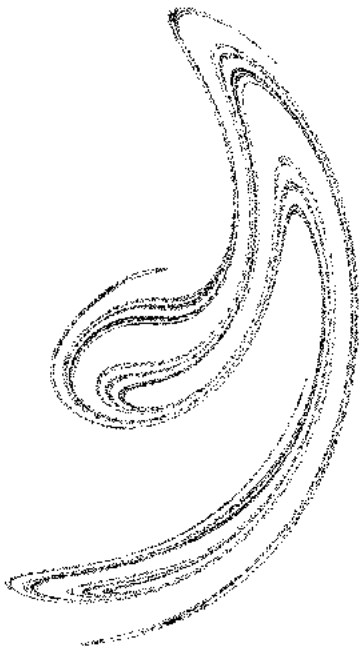
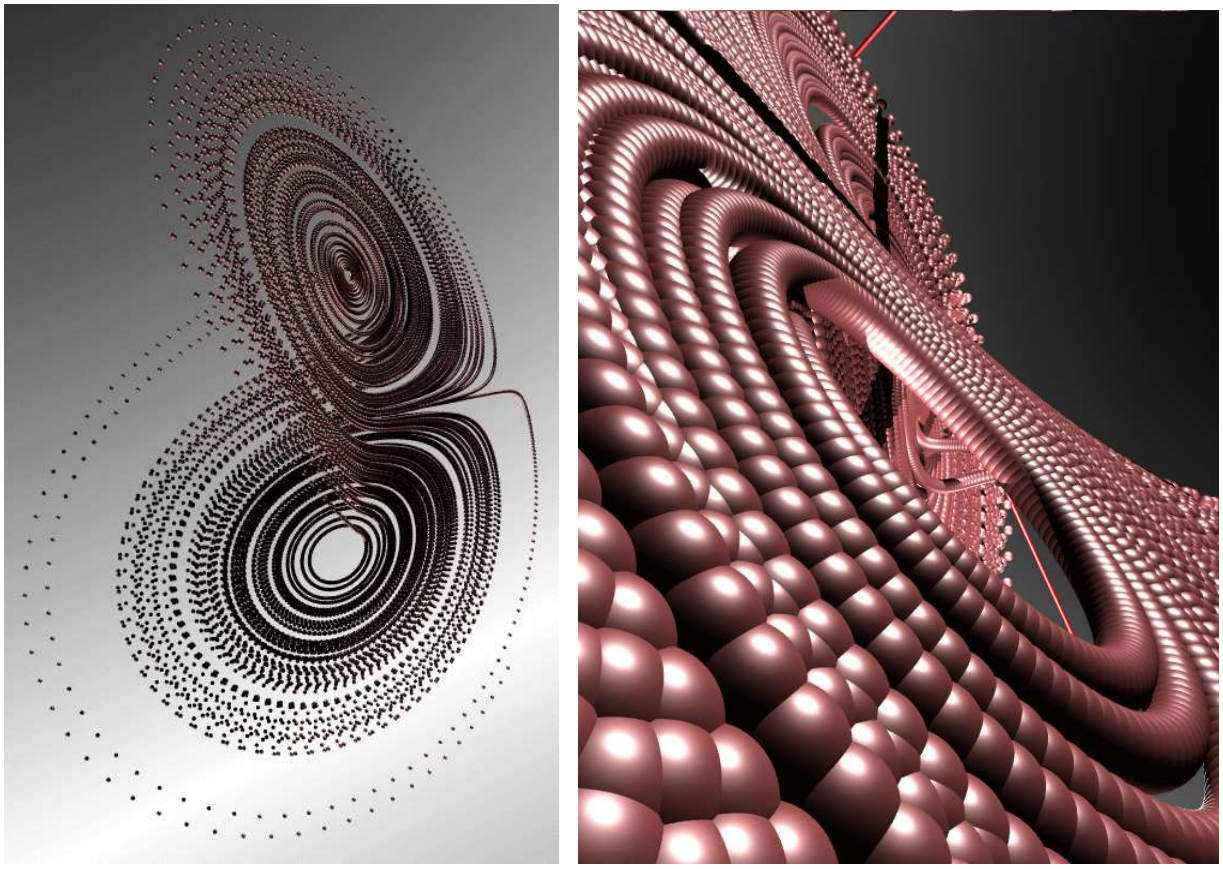


Figura 6.2: Atrator estranho de Ueda extraído de [142]

[1] A história sugere que uma definição mais irônica de caos pode ser construída simplesmente dizendo que algo é caótico se não o compreendemos.

A teoria de sistemas dinâmicos, que tornou possível trazer ordem ao caos, foi desenvolvida muito recentemente, mas seus fundamentos foram introduzidos no início do século passado por Jules Henri Poincaré, que re-introduziu a matemática visual. A matemática visual de Poincaré é bem diferente da geometria de Euclides. É uma matemática de padrões e relações conhecida como topologia. A topologia é uma "geometria" em que todos os comprimentos, ângulos e áreas podem ser distorcidos à vontade. Desse modo, um triângulo pode ser transformado, com continuidade, num retângulo ou mesmo num círculo. As figuras que podem ser transformadas umas nas outras por meio de dobramento, estiramento e torção são ditas "topologicamente equivalentes". Desta forma, intersecções de linhas precisam continuar sempre sendo intersecções. Uma rosquinha pode ser transformada topologicamente numa xícara de café (o buraco transforma-se numa asa), mas nunca numa panqueca. Podemos dizer que topologia é uma matemática de relações, de padrões imutáveis ou invariantes.



$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= a(y-x)\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(b-z)-y\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy-cz\end{aligned}$$

Figura 6.3: Atrator estranho de Lorenz extraído de [16]

6.1.3 Atratores

Poincaré utilizou concepções topológicas para analisar características qualitativas de complexos problemas dinâmicos. Dentre eles, temos o clássico problema dos três corpos em mecânica celeste, ou seja, o movimento relativo dos três corpos sob sua mútua atração gravitacional. Ninguém até então tinha sido capaz de resolver. Poincaré foi capaz de determinar uma forma geral das trajetórias para o problema ligeiramente simplificado e observou que a complexidade era assustadora:

"Quando se tenta representar a figura formada por essas duas curvas e sua infinidade de intersecções ... [descobre-se que] essas intersecções formam uma espécie de rede, de teia ou de malha infinitamente apertada; nenhuma das duas curvas pode jamais cruzar consigo mesma, mas deve dobrar de volta sobre si mesma de uma maneira bastante complexa a fim de cruzar infinitas vezes os elos da teia. Fica-se perplexo diante da complexidade dessa figura, que eu nem mesmo tento desenhar" [21].

"As técnicas matemáticas que permitiram aos pesquisadores, nas últimas décadas, descobrir padrões ordenados em sistemas caóticos baseiam-se na abordagem topológica de Poincaré e estão estreitamente ligadas com o desenvolvimento de computadores. Com a ajuda dos computadores atuais de alta velocidade, os cientistas podem resolver equações não-lineares por meio de técnicas que antes não estavam disponíveis. Esses poderosos computadores podem facilmente traçar as trajetórias complexas que Poincaré nem mesmo tentou desenhar" [21].

Métodos numéricos permitiram que equações não-lineares associadas a fenômenos caóticos pudessem ser visualizadas. Dessa forma, descobriu-se ordem sob a aleatoriedade aparente. Para que tais padrões sejam revelados, as variáveis de um sistema complexo são exibidas num espaço matemático abstrato denominado *espaço de fase*, em que se associa cada uma das variáveis do sistema a um eixo de coordenadas diferente. Marcam-se então diversos pontos simulando-se as equações e obtêm-se então figuras no espaço de fase.

Um conceito muito importante é o de *atrator*, em que um conjunto de pontos fixos no espaço atrai trajetórias. Diversos sistemas não-lineares foram analisados e descobriu-se que há poucos tipos de atratores. Há três tipos básicos:

1. **Punctiformes**

   Correspondentes a sistemas que atingem um equilíbrio estável, como é o caso de um sistema que modela um pêndulo oscilando com atrito.

2. **Periódicos**

   Correspondentes a sistemas que atingem um equilíbrio dinâmico, como é o caso de um sistema que modela um pêndulo oscilando sem efeito de atrito.

3. **Estranhos**

   Correspondentes a sistemas caóticos, como a transformação do padeiro.

Na Figura 6.2 temos um exemplo de atrator estranho de Ueda (Mapeamento Logístico) que é uma trajetória num espaço de fase bidimensional que gera padrões que quase se repetem, mas não totalmente, característica típica de todos os sistemas caóticos. Note que

podemos visualizar a figura como um corte de um pedaço de massa de farinha que foi esticado e amassado repetidamente.

Um fato notável a respeito de atratores estranhos é que eles tendem a ter dimensionalidade muito baixa, mesmo num espaço de fase com elevado número de dimensões. Desse modo, vemos que o comportamento caótico é muito diferente do aleatório, errático. Com a ajuda de atratores estranhos pode-se fazer uma distinção entre a mera aleatoriedade, ou ruído, e o caos.

O comportamento caótico é determinístico e padronizado, e os atratores estranhos nos permitem transformar os dados aparentemente aleatórios em formas visíveis distintas.

Na Figura 6.3 temos exemplos do famoso e amplamente estudado atrator estranho de Lorenz, que é tridimensional e foi introduzido em 1963, marcando o início da Teoria do Caos.

6.2 *Chaos Game*

O *Chaos Game* [101] é um algoritmo, geralmente controlado por uma série de números randômicos, que permite a produção de imagens no espaço de fase por meio da utilização de atratores.

Para jogar o *Chaos Game*, é necessário seguir os seguintes passos:

1. Escolha n pontos em um plano
2. Pegue um dado em que cada face represente cada ponto escolhido
3. Escolha um ponto P qualquer no plano
4. Jogue o dado

   Mova então o ponto P em direção ao ponto sorteado. Sua nova posição será o meio do segmento de reta que separa os dois pontos.
5. Realize o passo 4 várias vezes
6. Jogue o dado

   Agora, para cada sorteio, ao invés de simplesmente mover o ponto P como no passo 4, marque os pontos que são visitados, ou seja, para cada novo sorteio, gere um novo ponto.
7. Realize o passo 6 quantas vezes desejar

Os passos 4 e 5 servem simplesmente para que o ponto P seja posicionado dentro do fecho convexo dos n pontos inicialmente escolhidos.

Os passos 6 e 7 são a essência do jogo, em que a figura é formada.

Observe que tal algoritmo define um atrator estranho como os descritos na Seção 6.1.

Na Figura 6.4 temos um exemplo de trajetória para três pontos e sorteio de vermelho, verde, azul, azul.

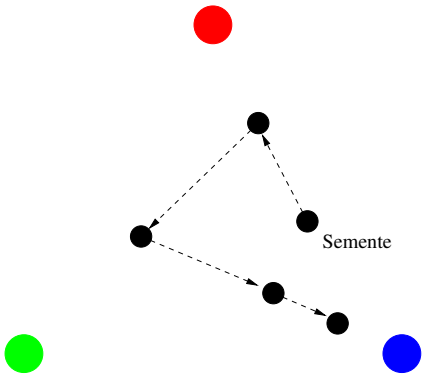


Figura 6.4: Exemplo de trajetória *Chaos Game*

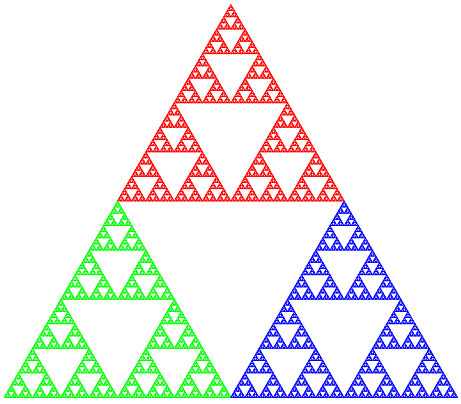


Figura 6.5: Exemplo de figura gerada pelo atrator *Chaos Game* com 3 pontos eqüidistantes

Na Figura 6.5 temos um exemplo de trajetória para três pontos e distribuição uniforme para cada ponto. O resultado é o famoso triângulo equilátero de Sierpinski.

Ao manipularmos as posições dos pontos do atrator, sempre obtemos figuras com componentes auto-similares. Em casos em que a distribuição do sorteio não é uniforme, observa-se que áreas na figura aparecem "apagadas".

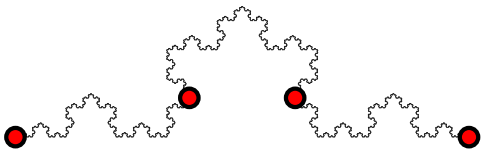


Figura 6.6: Atrator para a curva de Koch

Ao mudarmos características do atrator, como número de pontos e a forma de transformação da trajetória (inclusive com transformações afins, como rotação), podemos obter uma série de figuras bem diferentes. Na Figura 6.6 temos a curva de Koch com os 4 pontos necessário para o atrator. Quando sorteiam-se os dois pontos de cima, realiza-se também rotações de 60^o e -60^o.

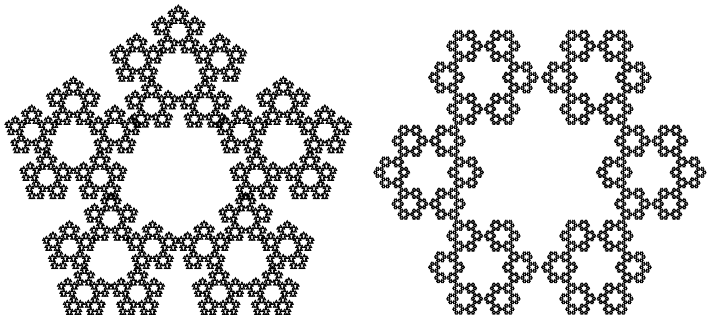


Figura 6.7: Pentágono e Hexágono de Sierpinski

Ao escolhermos 5 e 6 pontos respectivamente como atratores e variando um pouco a regra para o cálculo da trajetória, podemos produzir figuras como os pentágono e hexágono de Sierpinski apresentados na Figura 6.7. Note que as bordas dessas figuras são iguais à curva de Koch.

Outras formas muito interessantes podem ser obtidas também acertando o atrator, como é o caso da árvore de Barnsley representado na Figura 6.8 e das samambaias de Barnsley.

Nas Figuras 6.9 e 6.10 temos exemplos de como uma imagem é construída e de samambaias de Barnsley.

6.3 Construção de Imagens CGR

Oliver *et al.* [101] propôs o uso de seqüências de **DNA** no lugar das séries de números randômicos. Chamamos uma imagem gerada pela seqüência de **DNA** de Representação *Chaos Game* (**CGR**) da seqüência de **DNA**.

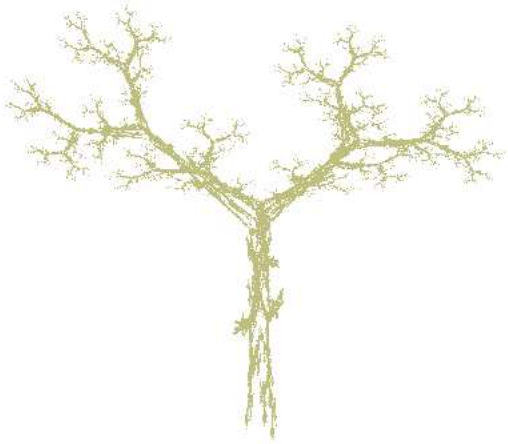


Figura 6.8: Árvore de Barnsley extraída de [111]

Figura 6.9: Construção de uma samambaia de Barnsley extraída de [100]

Figura 6.10: Samambaias de Barnsley extraídas de [100]

Como o **DNA** tem quatro tipos de bases possíveis, associamos cada uma delas a um ponto do atrator. A regra para a atualização da trajetória é a mesma do triângulo de Sierpinski.

Fazemos ainda uma restrição, sem perda de generalidade, que consiste na introdução do conceito de resolução da figura final, ou seja, diferentemente das figuras que geramos até agora, que têm infinitos pontos, assumimos que as nossas figuras são imagens digitais com $n \times n, n = 2^t$ pontos, em que t é a resolução que queremos utilizar. Dessa forma, a trajetória dos pontos é restrita aos pontos da imagem, sendo arredondada a cada passo da iteração. Criamos então uma matriz em que cada entrada corresponde ao número de vezes em que o ponto deslocado caiu sobre o ponto correspondente à entrada, após cada sorteio.

Tal regra, faz com que cada subpalavra de tamanho t possível corresponda a exatamente um ponto na figura final. Na Figura 6.11 temos um exemplo de trajetória para $t = 2$ e seqüência de **DNA** $ACCTATTG$ percorrida da direita para a esquerda.

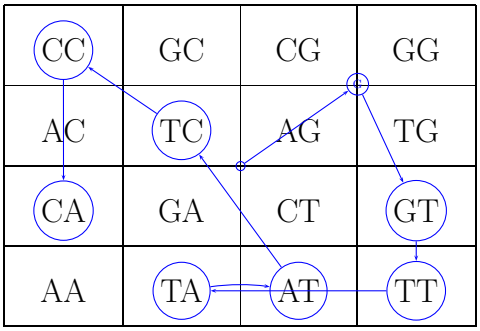


Figura 6.11: Exemplo de trajetória **CGR** para $t = 2$ e seqüência de **DNA** $ACCTATTG$

6.3.1 CGR Recursivo

Podemos olhar **CGR** de forma mais iterativa e recursiva, lembrando que cada ponto da imagem corresponde à uma subpalavra. Dessa forma, podemos dizer que **CGR** consiste em arranjar a tabela de freqüência de subpalavras 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 dessa matriz representa a freqüência de uma determinada subpalavra possível, ou seja, uma configuração possível de j.

Para o caso em que a janela tem tamanho 1, há somente 4 entradas na tabela de freqüências, o próprio alfabeto. Nós arranjamos a tabela de exemplo a seguir

Tabela:

A	10
T	9
C	4
G	13

na seguinte matriz 2×2:

C	G
A	T

$\Longrightarrow$

4	13
10	9

Para o caso em que a janela tem 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

Note que para o exemplo apresentado anteriormente para a seqüência $ACCTATTG$, temos a seguinte tabela de freqüências:

AA	0
AT	1
AC	1
AG	0
TA	1
TT	1
TC	0
TG	1

CA	0
CT	1
CC	1
CG	0
GA	0
GT	0
GC	0
GG	0

que corresponde exatamente à Figura 6.11 gerada pelo *Chaos Game*, visto que esta regra recursiva é equivalente à regra do *Chaos Game*. Note que nos exemplos com matriz de subpalavras elas estão escritas da direita para a esquerda, a matriz é apresentada assim por razões históricas. Note que caso percorrêssemos a palavra no *Chaos Game* da esquerda para a direita, as palavras na matriz poderiam ser lidas normalmente da esquerda para a direita.

6.4 Exemplos

Em vez de números, podemos utilizar tons para a visualização. Na Figura 6.12, temos imagens **CGR** usando janelas $t = 1 \ldots 8$ para o genoma da bactéria *A. fulgidus*. A intensidade do nível de preto de cada ponto é diretamente proporcional à freqüência da palavra correspondente. As últimas três imagens estão normalizadas por log.

Diferentemente das figuras obtidas quando se escolhe 3, 5, 6 ou mais pontos correspondentes ao vértice de uma forma regular como apresentado na seção anterior, quando escolhemos os 4 pontos do vértice de um quadrado como atrator, obtém-se um quadrado preenchido como figura final no espaço de fase. Quando se utiliza um dado honesto, o resultado é um quadrado razoavelmente homogêneo (caso se sorteie muitos pontos) ou um quadrado preenchido com ruído, parecido com o de televisão (caso se sorteie poucos pontos). Os padrões que observamos nas figuras desta seção surgem porque, quando a distribuição do sorteio não é uniforme, áreas apagadas aparecem nas imagens como é o caso dos pequenos quadrados

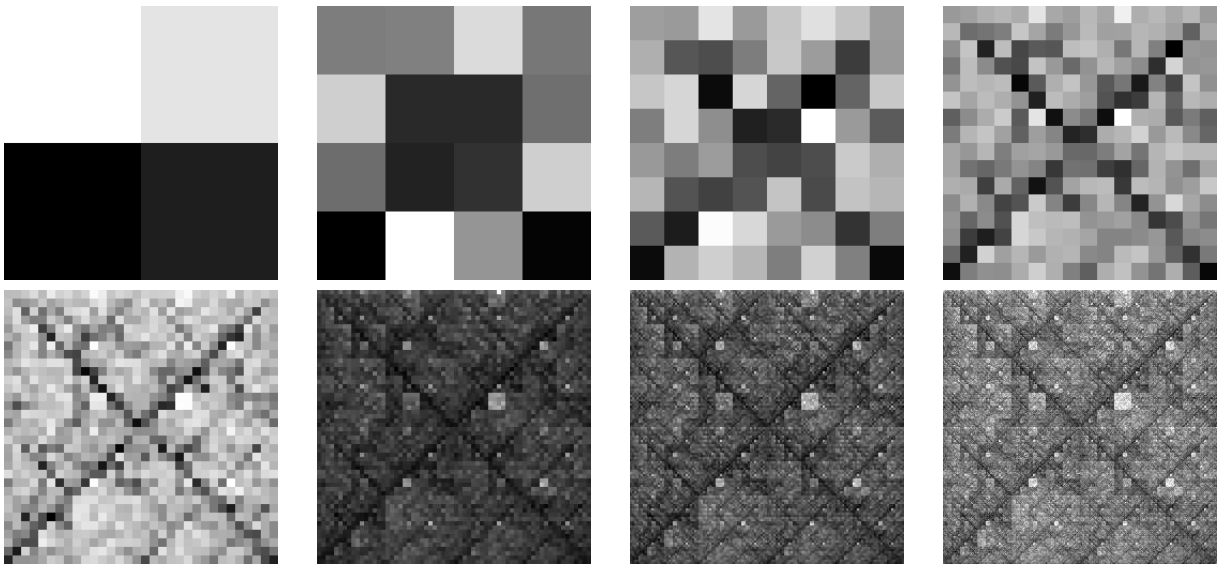


Figura 6.12: Imagens **CGR** para janelas $t = 1 \ldots 8$ da bactéria *A. fulgidus*

claros presentes na Figura 6.12. Dessa forma, imagens **CGR** são ferramenta muito interessante para visualizar a distribuição de subpalavras numa seqüência de **DNA**. Note que o atrator definido desta forma exibe "ordem" em algo aparentemente randômico.

Na Figura 6.13 temos imagens **CGR** calculadas com o genoma de diversas espécies. Note que há diferenças significativas no padrão apresentado por cada espécie.

Deschavanne *et al.* [30] observa que os padrões observados nas figuras são diferentes para cada espécie e podem ser reproduzidos a partir de fragmentos pequenos do genoma de cada uma. Além disso, é possível reconstruir árvores filogenéticas de forma bem razoável utilizando somente a imagem **CGR** do genoma de cada espécie.

Nosso objetivo é tentar extrair alguma informação útil a partir dessas tabelas de freqüência. Na Figura 6.15, exemplos ilustrativos de imagens normalizadas por log 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 6.14 enquanto outros são completamente diferentes. Além disso, há diferenças significativas entre as regiões de éxons e as regiões de íntrons.

6.5 Análise Multi-resolução e Multi-escala de imagens CGR

Durante a realização deste trabalho, foi desenvolvido um algoritmo para cálculo de imagens **CGR** que tem algumas extensões em relação à definição original de **CGR**. Tais extensões consistem basicamente da possibilidade de definição de janelas, inclusive janelas "esburacadas" que permitem a realização de análise multi-resolução e multi-escala como definido na Seção 4.2.

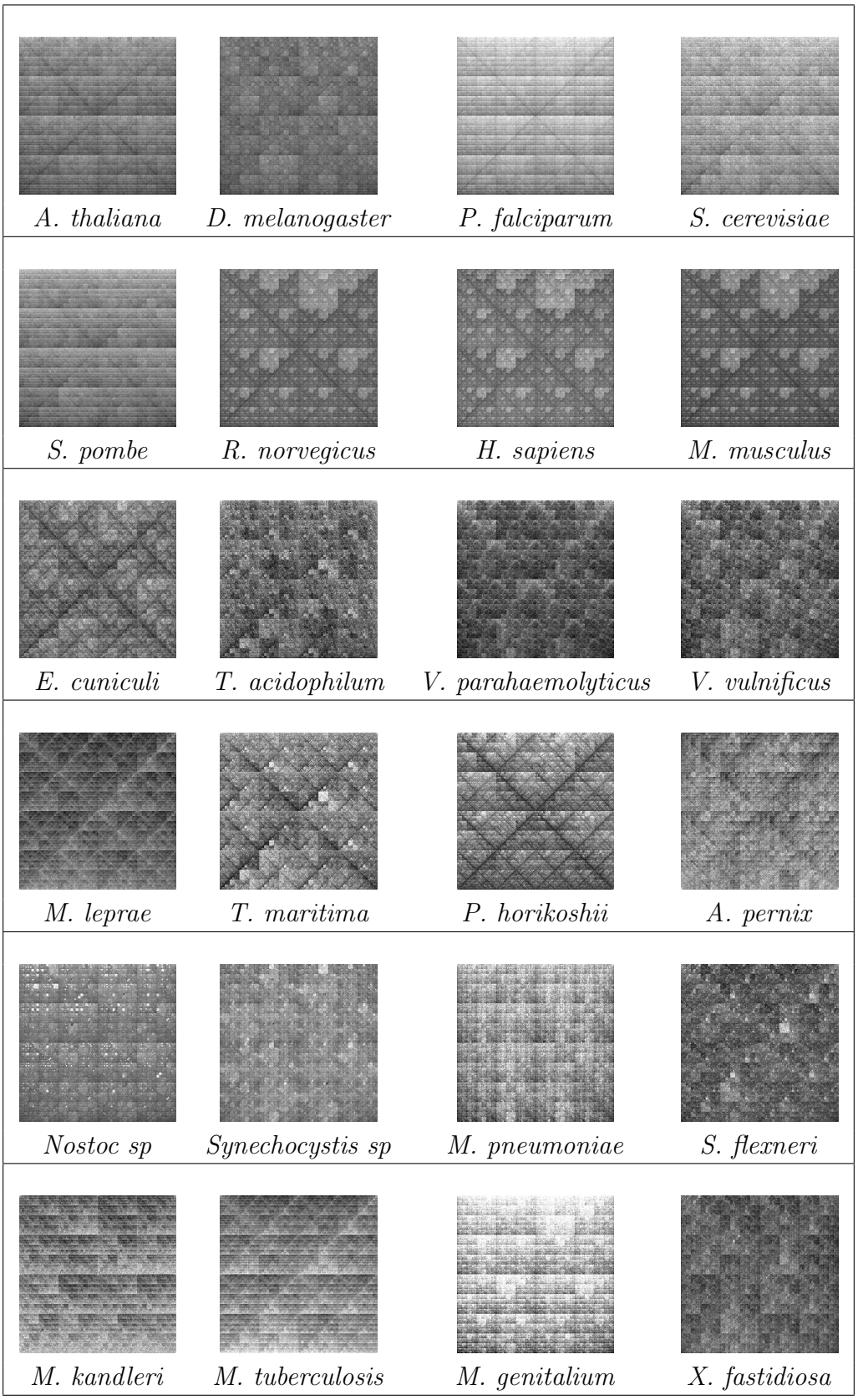


Figura 6.13: Imagens **CGR** do genoma de diversas espécies usando janela de tamanho 8

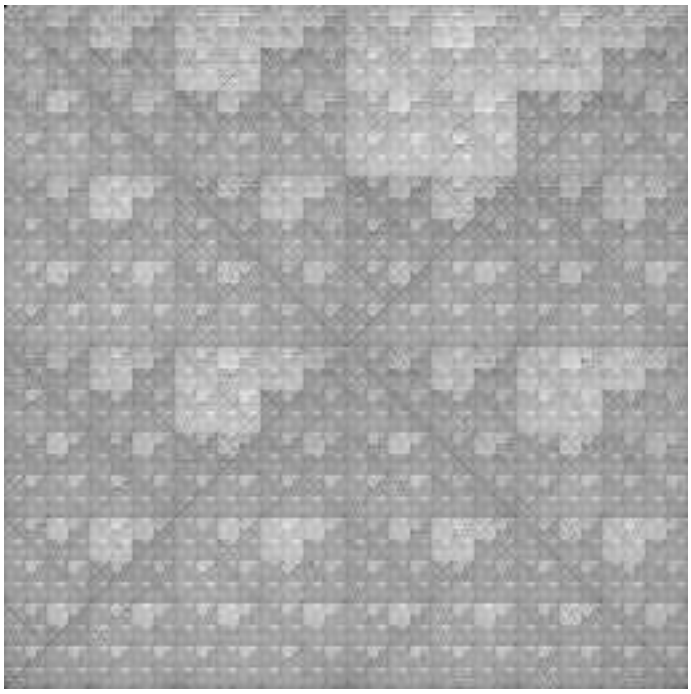


Figura 6.14: Imagem **CGR** normalizada por log do cromossomo 22 humano para janela de tramanho 8

6.5.1 Variando a resolução da imagem CGR

Primeiramente, avaliamos o comportamento das imagens **CGR** variando-se o tamanho da janela.

Nas Figuras 6.16 e 6.17 podem-se visualizar exemplos do efeito causado pelo aumento de resolução. Observe que quando normalizadas linearmente as últimas 5 imagens são pouco visualizáveis, enquanto no caso da normalização por log ainda é possível ver alguma coisa quando $j = 12$.

Tal fato acontece porque o número de subpalavras é praticamente constante independentemente da resolução utilizada enquanto o número de pontos na imagem aumenta exponencialmente, dessa forma, a distribuição de pontos vai ficando "esparsa", mesmo quando usamos o cromossomo 22 inteiro que gera mais do que 40 milhões de subpalavras.

6.5.2 Análise Multi-resolução e Multi-escala

Tendo em vista o efeito que que é gerado quando se aumenta a resolução das imagens **CGR**, aplicamos métodos multi-resolução e multi-escala.

Para a multi-resolução, definimos regras de corte da seqüência de **DNA** na tentativa de simular um "afastamento", ou seja, o efeito observado quando diminuímos a resolução de um mapa. No início, somos capazes de enxergar ruas, ao diminuirmos a resolução, só enxergamos bairros, cidades, países e assim por diante. No caso de **DNA**, utilizamos uma regra de corte do tipo: "a cada 2 bases, joga-se a terceira fora", ou seja ATCTTA ficaria

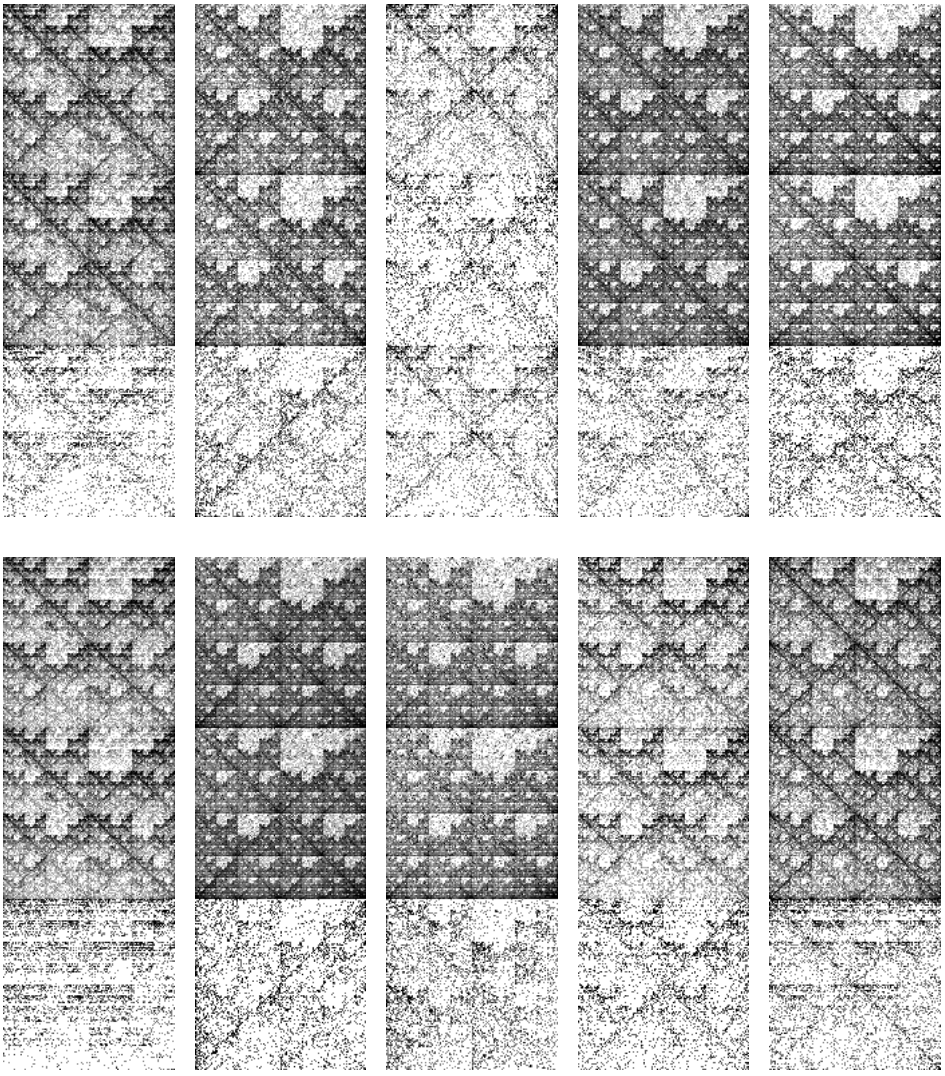


Figura 6.15: Imagens **CGR** normalizadas por log de genes do cromossomo 22 humano para janela de tramanho 8

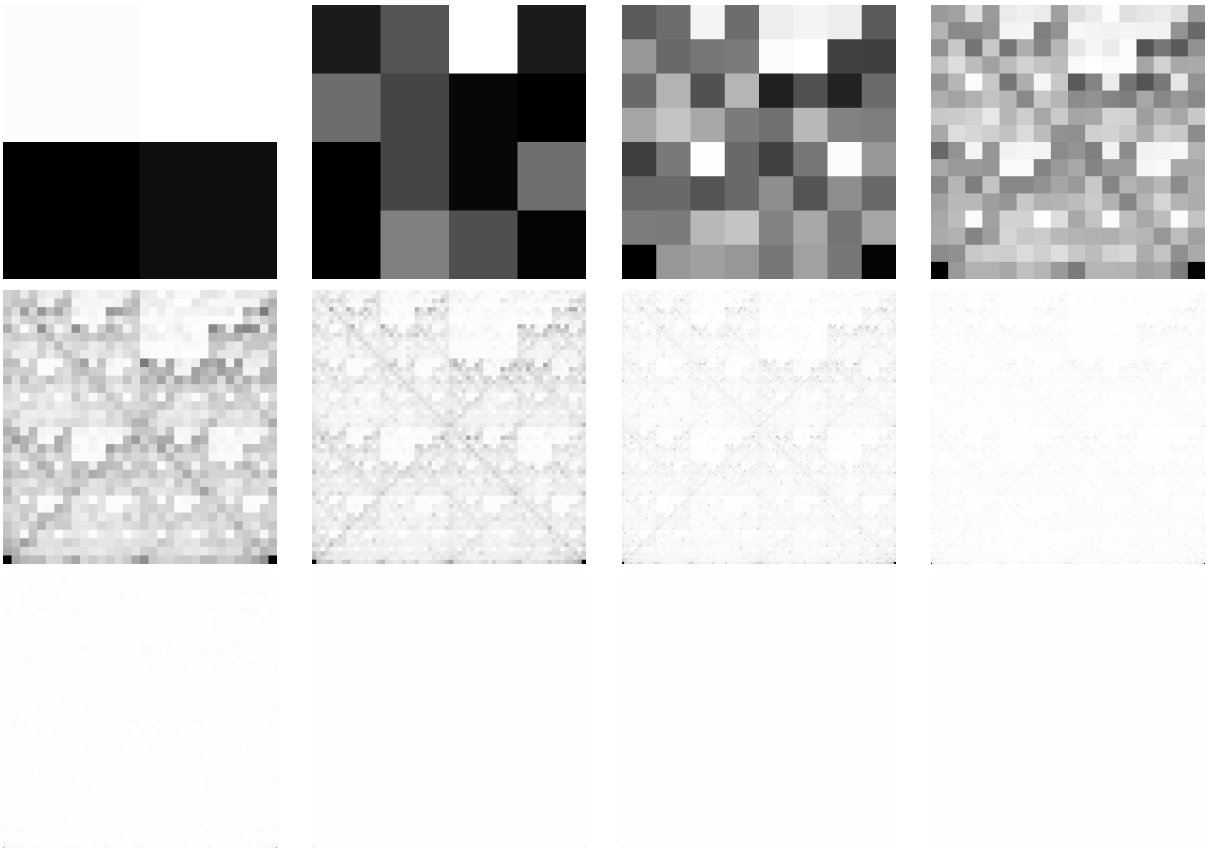


Figura 6.16: Imagens **CGR** normalizadas linearmentes do cromossomo 22, $j = 1 \ldots 12$

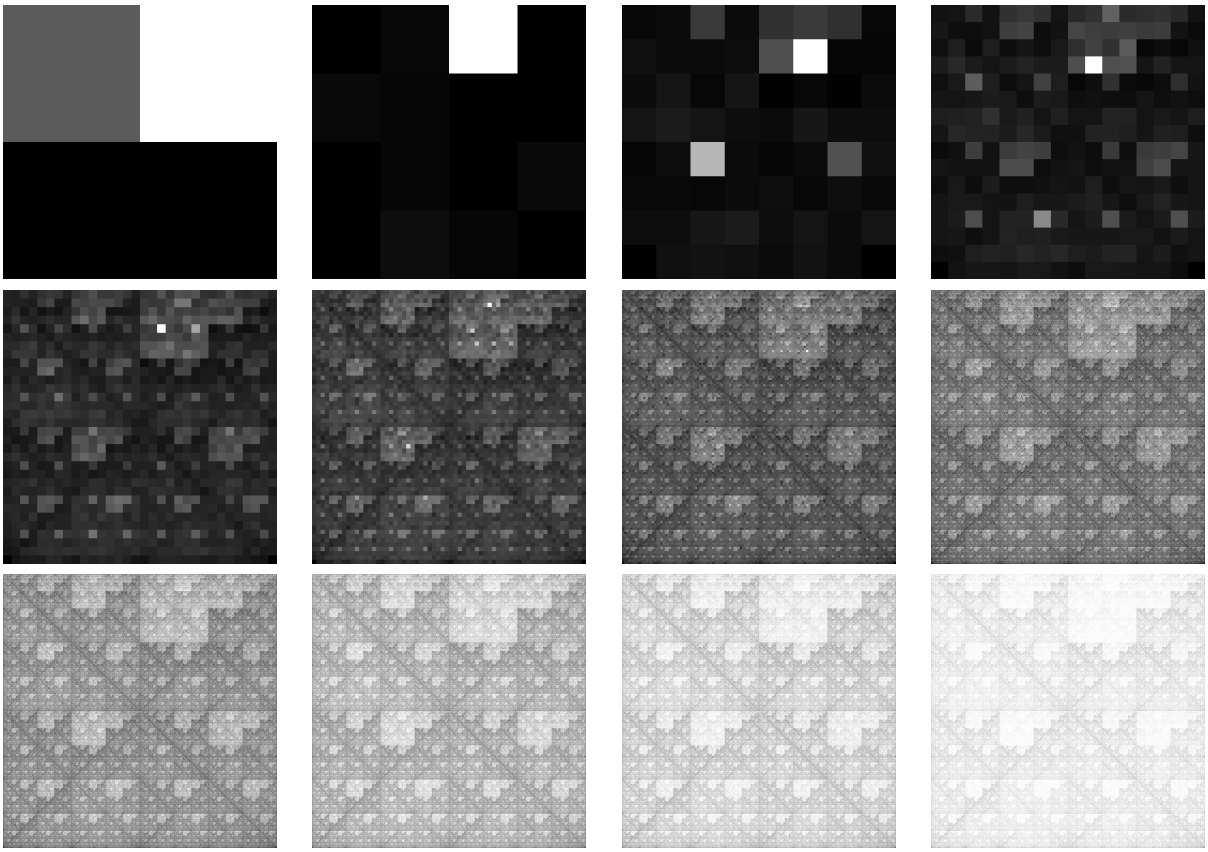


Figura 6.17: Imagens **CGR** normalizadas por log do cromossomo 22, $j = 1 \ldots 12$

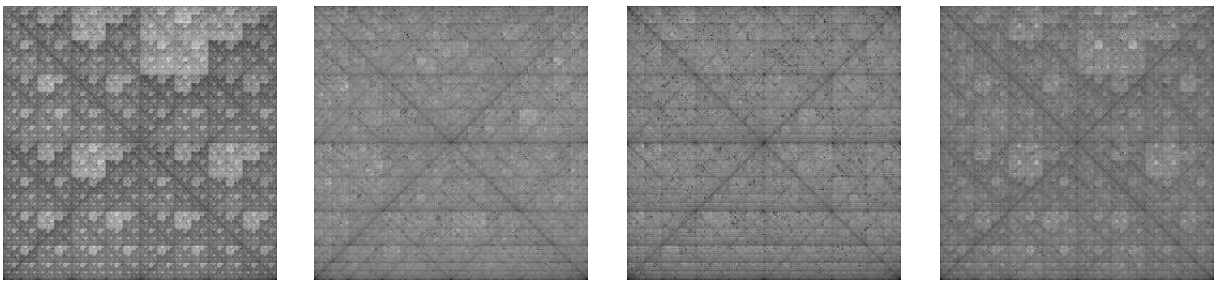


Figura 6.18: Imagens **CGR** multi-resolução normalizadas por log do cromossomo 22 humano de acordo com as regras apresentadas

ATT, que é representada por XX_ em que 'X' representa preenchimento e '_' exclusão ou buraco.

Na Figura 6.18 temos exemplos de imagens obtidas através da utilização de tais técnicas para o cromossomo 22. Note que a degeneração acontece rapidamente.

Para a multi-escala, definindo janelas com buracos, como se faria em processamentos de imagens, criando classes de equivalência. Utilizamos janelas simples. Por exemplo, uma janela X_X significa que A**T**G, A**A**G, A**C**G e A**G**G estão na mesma classe de equivalência, ou seja, correspondem todos a um mesmo ponto na imagem **CGR**. Dessa forma, é possível levar em consideração mais pontos ao mesmo tempo (escala maior), mas analisamos menos informação (resolução da entrada, seqüência de **DNA** no caso, menor). No fundo é uma tentativa de tornar a análise "mais global" sem precisar de mais exemplos.

Tal método em geral permite que se utilizem resoluções maiores sem sofrer tanto com o efeito apresentado, o que pode significar um ganho na capacidade de representação significativo.

Na Figura 6.19 apresentamos exemplos de imagens obtidas através da utilização de tais técnicas para o cromossomo 22. Note que a degeneração acontece rapidamente dependendo da janela, mas nos dois últimos exemplos é possível analisar janelas de tamanho 12 e 16 respectivamente sem que a imagem se torne esparsa, o que é muito interessante. Note que o padrão praticamente se mantém para janelas que desconsideram alguns vizinhos.

Como dito na Seção 4.2, tal tipo de análise multi-resolução e multi-escala só é boa se a regra para a criação de classes de equivalência for "boa". Aplicamos tais regras (inclusive simultaneamente) em diversos cromossomos e o que observamos é uma degeneração das imagens **CGR**, o que sugere que análises multi-resolução e multi-escala não podem ser realizadas desta forma. Tal degeneração acontece porque subpalavras formadas por somente duas letras (representadas pelas linhas retas que observamos nas figuras se desconsiderarmos prefixos das mesmas) são ampliadas, enquanto as outras estruturas se perdem. Isso sugere que as subpalavras do genoma humano que contêm somente dois tipos de letras são bem mais numerosas e maiores que as janelas utilizadas, por exemplo: $ATAAATTAATAA$ se tornaria $AAATAA$ se utilizarmos a janela de corte multi-resolução $X\_$ e se tornaria $TATATA$ se utilizarmos a janela de multi-escala $\_X\_X\_X\_X\_X\_X$, por exemplo. Ou seja, após a transformação, janelas com somente dois tipos de letras continuam com no máximo dois

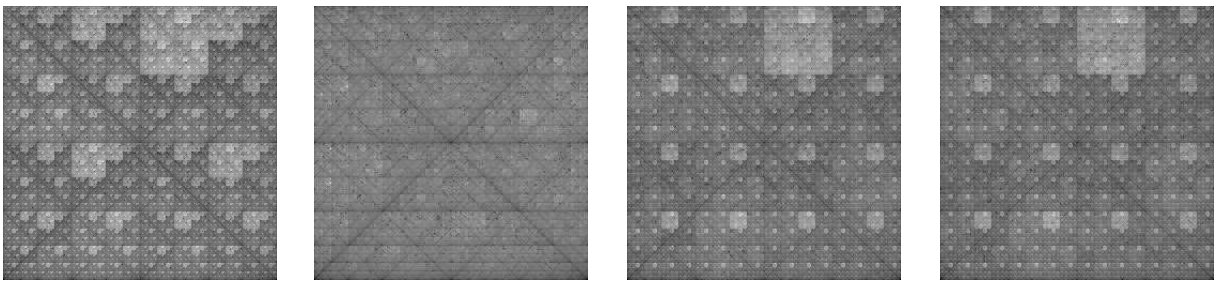


Figura 6.19: Imagens **CGR** multi-escala normalizadas por log do cromossomo 22 humano utilizando as janelas de acordo com as regras apresentadas

tipos de letras.

Apesar da mesma propriedade valer para janelas com mais letras, o que se observa é uma rápida degeneração, o que sugere uma baixa relação entre bases próximas.

Chegamos à conclusão que tanto multi-resolução quanto multi-escala, como se faz em imagens (em geral com a função zoom), não se aplicam muito bem em seqüências de **DNA** diretamente. Tal fato se deve principalmente porque não é claro, dada 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 tridimensional "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. Aumentando ainda mais o zoom, chega-se ao código de **DNA** em si.

O problema é que para calcular essa função de detalhamento, ter-se-ia de 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 descrita 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 suficientes para deformar a estrutura básica 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.

Uma visão multi-escala interessante baseada em **HMM** é a proposta por David Kulp *et al.* [77]. 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.

Capítulo 7

Análise Fractal Multi-escala

Tendo em vista a natureza caótica que imagens **CGR** possuem, vamos utilizar técnicas de Geometria Fractal para analisá-las.

A Geometria Fractal se desenvolveu independentemente da Teoria do Caos, mas fornece uma forma convincente de descrever a estrutura em "escala fina" dos atratores caóticos.

Tal geometria foi introduzida pelo francês Benoît Mandelbrot e permite a realização de uma análise qualitativa de alguns aspectos dos atratores. Tal análise nos permite prever qualidades dos atratores, mas não quantidades, como a localização exata de um ponto numa trajetória caótica num determinado instante.

Ao longo deste capítulo apresentaremos uma breve e informal introdução a alguns aspectos da Geometria Fractal, bem como uma técnica multi-escala para análise de fractais introduzida por Costa *et al.* [27, 25]. Tal técnica será usada para analisar seqüências de **DNA**.

7.1 Fractais

Mandelbrot criou a geometria fractal – "uma linguagem para falar de nuvens" – para descrever e analisar a complexidade das formas irregulares do mundo natural que nos cerca.

A propriedade mais característica das formas fractais é que seus padrões característicos são repetidamente encontrados em escala descendente, de modo que suas partes, em qualquer escala, são, na forma, semelhantes ao todo. Essa propriedade é conhecida como *auto-similaridade* e pode ser observada em muitas formas da natureza, em diversas escalas, como é o caso do exemplo clássico da couve-flor apresentado por Mandelbrot, em que um pedaço se parece muito com o todo. Pode-se dividir novamente o pedaço que continuará sendo semelhante ao todo.

Uma descoberta surpreendente foi que atratores estranhos exibem estrutura fractal, ou seja, quando partes da estrutura de um atrator estranho são ampliadas, elas revelam subestruturas em muitas camadas nas quais os mesmos padrões são repetidos diversas vezes. Dessa forma, pode-se definir atratores estranhos como trajetórias no espaço de fase que exibem geometria fractal.

Como já visto no Capítulo 6, é impossível calcular, utilizando a tecnologia atual, os valores das variáveis de um sistema caótico, mas a observação acima permite que se crie uma forma de predizer as características qualitativas do comportamento do sistema.

7.1.1 Modelando a Natureza

Uma das abordagens para modelar formas fractais que ocorrem na natureza é a utilização de figuras geométricas que exibem auto-similaridade precisa.

Tais figuras podem ser construídas utilizando iterações de operações geométricas [55], como é o processo de iterações apresentado pela transformação do padeiro, que é a característica matemática subjacente aos atratores estranhos e que liga a Teoria do Caos à geometria fractal.

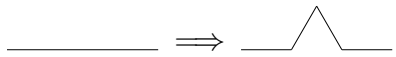


Figura 7.1: Transformação básica da curva de Koch

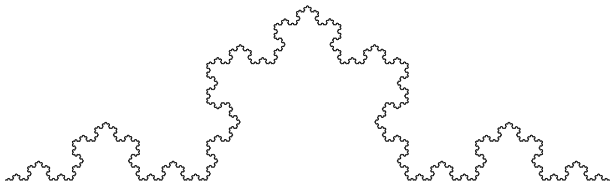


Figura 7.2: Curva de Koch após 6 passos

Um outro exemplo simples de formas fractais geradas por iteração é a clássica curva de Koch, também conhecida como curva de floco de neve. Para construir tal figura, basta dividir uma linha em três partes iguais e substituir a seção central por dois lados de um triângulo eqüilátero, como mostrado na Figura 7.1. A repetição dessa operação muitas e muitas vezes (para todas as linhas da figura em cada passo) leva à curva de Koch representada na Figura 7.2.

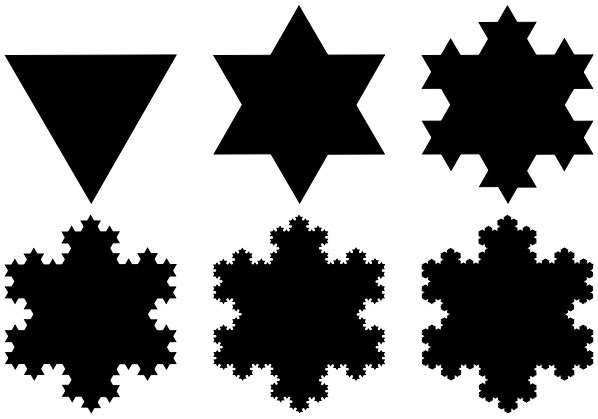


Figura 7.3: Construção iterativa do floco de neve de Koch

Diversas outras figuras podem ser construídas utilizando regras simples. Na Figura 7.3

temos um exemplo de construção do floco de neve de Koch. As outras figuras apresentadas no capítulo anterior também têm regras simples de construção.

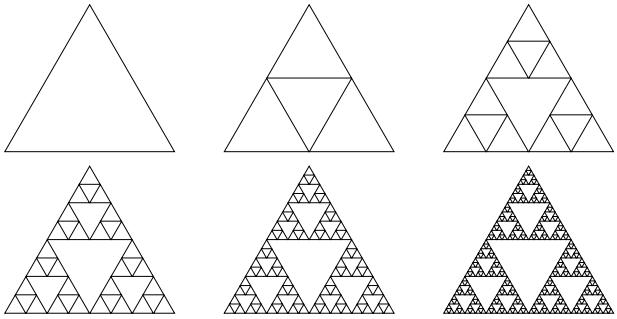


Figura 7.4: Construção iterativa do triângulo eqüilátero de Sierpinski

Na Figura 7.4 temos a transformação necessária para a construção do triângulo eqüilátero de Sierpinski mostrado na Figura 6.5.

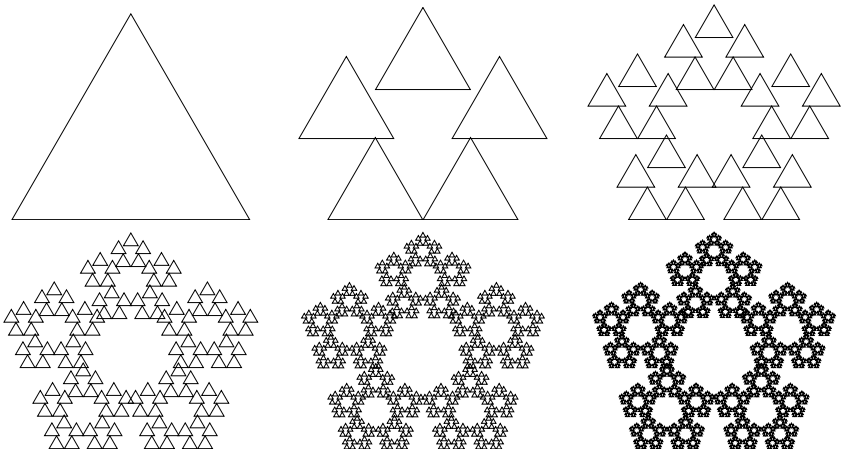


Figura 7.5: Construção iterativa do pentágono eqüilátero de Sierpinski

Na Figura 7.5 temos a transformação necessária para a construção do pentágono eqüilátero de Sierpinski mostrado na Figura 6.7.

Na Figura 7.6 temos a transformação necessária para a construção do hexágono eqüilátero de Sierpinski mostrado na Figura 6.7.

Na Figura 7.7 temos a transformação necessária para a construção do galho de Barnsley e na Figura 7.8 temos a transformação necessária para a construção da árvore de Barnsley mostrada na Figura 6.8.

Na Figura 7.9 temos a transformação necessária para a construção de um cristal.

Na Figura 7.10 temos a regra necessária para a construção de uma samambaia de Barnsley mostradas na Figura 6.10.

Na Figura 7.11 temos a regra necessária para a construção de uma samambaia de Garcia. Essa regra muito simples produz resultados incríveis.

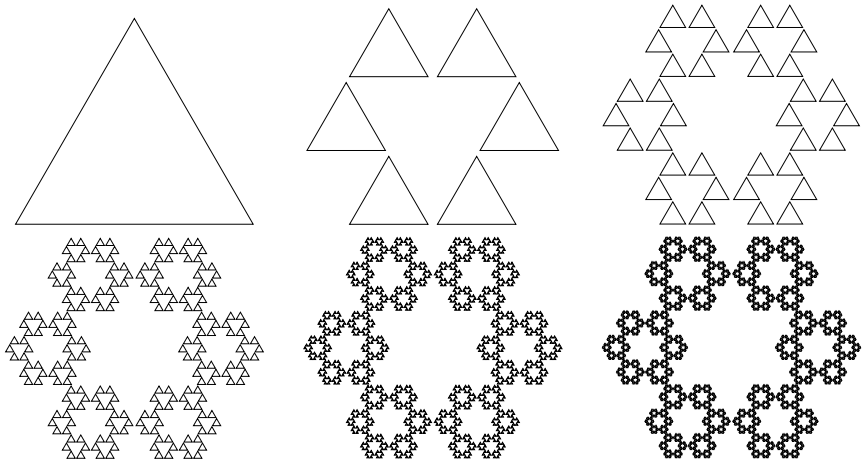


Figura 7.6: Construção iterativa do hexágono eqüilátero de Sierpinski

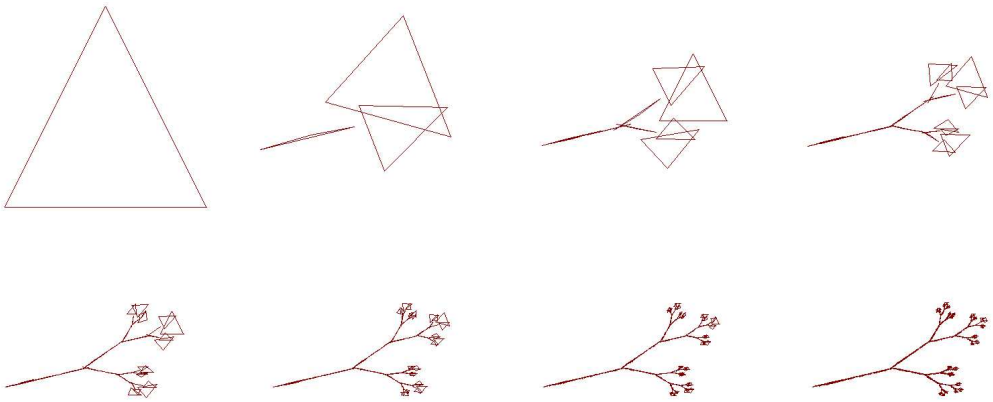


Figura 7.7: Construção iterativa de um galho de Barnsley

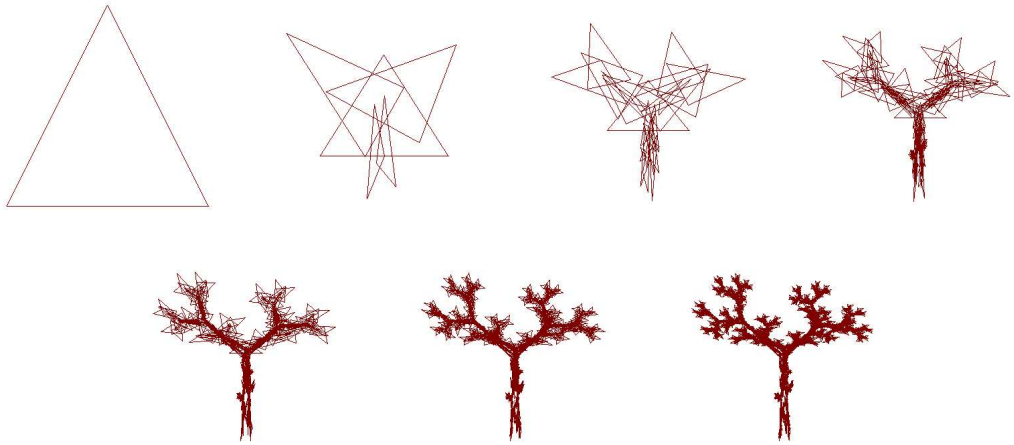


Figura 7.8: Construção iterativa de uma árvore de Barnsley

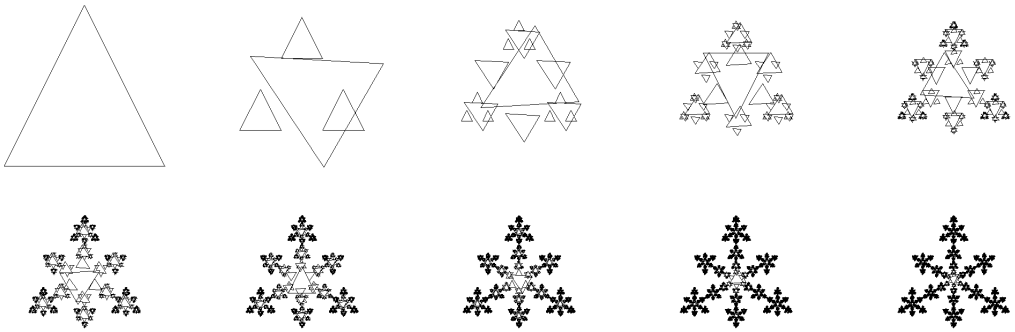


Figura 7.9: Construção iterativa de um cristal

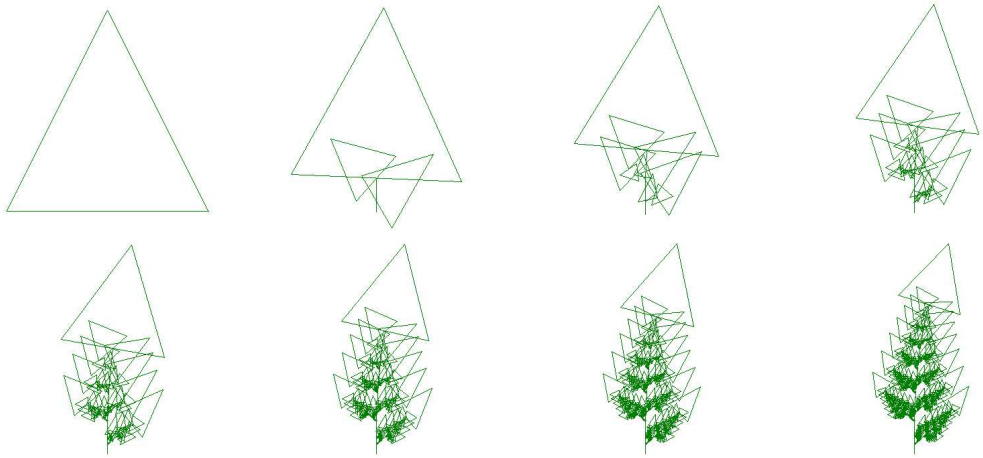


Figura 7.10: Construção iterativa de uma samambaia de Barnsley

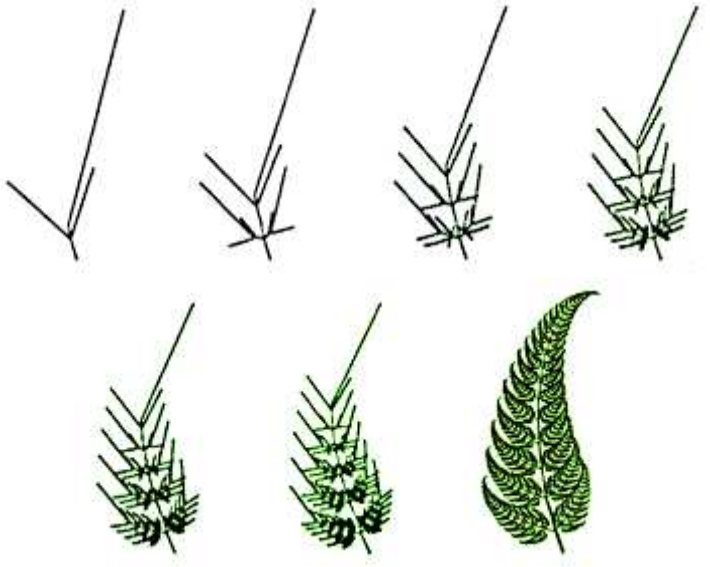


Figura 7.11: Construção iterativa de uma samambaia de Garcia extraído de [49]

É importante observar que as regras aqui apresentadas não são únicas, existe mais de uma regra iterativa capaz de produzir a mesma figura.

7.1.2 Conjunto de Mandelbrot

Muitas formas fractais podem ser matematicamente geradas por meio de procedimentos iterativos no plano complexo. Um exemplo muito conhecido são os conjuntos de Julia, cuja base é simplesmente o mapeamento

$$z \rightarrow z^2 + c$$

em que z e c assumem valores complexos. Na Figura 7.12 temos exemplos de conjuntos de Julia. O conjunto de Julia é o conjunto de todos os valores de z que permanecem finitos sob a iteração.

Os conjuntos de Julia apresentam uma variedade muito grande de formas, como nuvens, arbustos, fogos de artifício, coelhos, caudas de cavalo-marinho, entre muitos outros. Ao se ampliar as figuras, observa-se padrões dentro de padrões, semelhantes, mas nunca idênticos.

Mandelbrot criou uma forma de catalogar tais conjuntos, em que um ponto no plano complexo corresponde à um valor da constante c. O ponto é marcado caso o conjunto de Julia correspondente seja uma peça isolada e conexa. Na Figura 7.13 temos o famoso conjunto de Mandelbrot, que é único e considerado o objeto matemático mais complexo já inventado, apesar da simplicidade das suas regras de construção. É considerado um "superfractal", porque além de apresentar auto-similaridade por repetir os mesmos padrões, inclusive pequenas réplicas de todo o conjunto, contém elementos provenientes de um número infinito de conjuntos de Julia.

A geometria fractal e a Teoria do Caos forçaram os cientistas a reexaminar a própria concepção de complexidade, porque na matemática clássica, fórmulas simples levam a formas

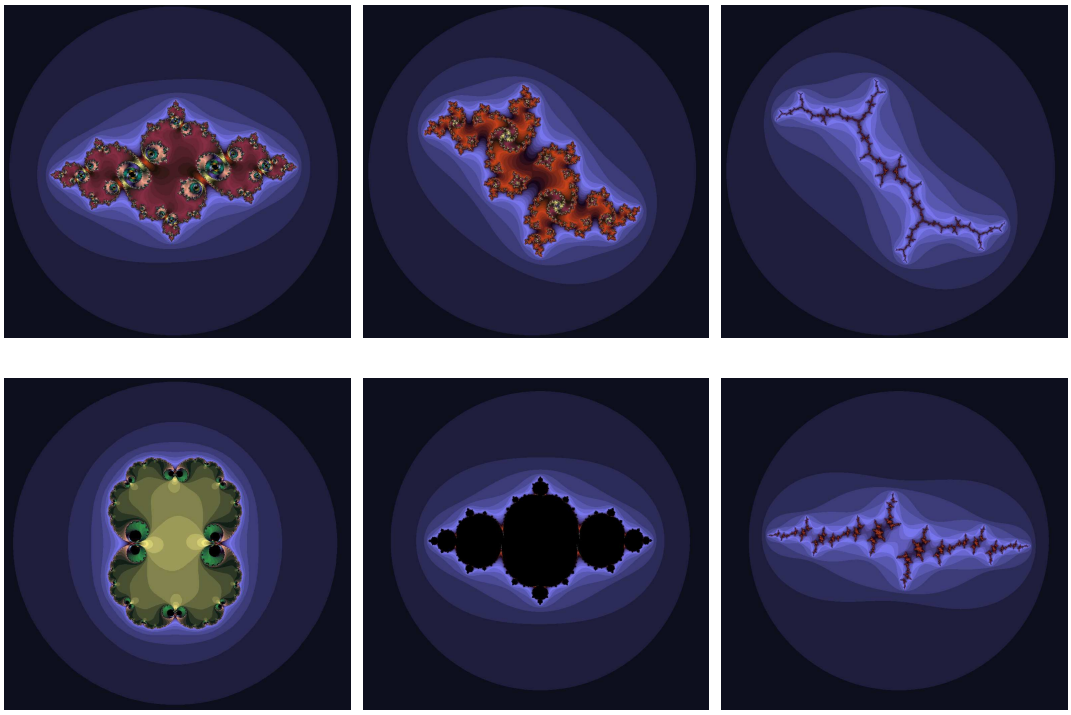


Figura 7.12: Diversos tipos de conjunto de Julia

simples e fórmulas complexas a formas complexas. Isso não acontece com sistemas não-lineares. Mandelbrot vê isso como um desenvolvimento científico novo e muito instigante:

"Trata-se de uma conclusão muito otimista porque, no final das contas, o significado inicial do estudo do caos foi a tentativa de descobrir regras simples no universo ao nosso redor. ... O esforço foi sempre o de procurar explicações simples para realidades complicadas. Mas a discrepância entre simplicidade e complexidade nunca foi, em nenhum outro lugar, comparável àquela que encontramos neste contexto"

7.2 Dimensão Fractal

Uma das características mais interessantes de formas fractais é a impossibilidade de se calcular o comprimento ou a área de uma forma fractal, mas é possível definir o grau de "denteamento" de uma maneira qualitativa. Mandelbrot apresentou a seguinte questão: "Qual é o comprimento do litoral da Inglaterra?", ele mostrou que, como o comprimento medido pode ser indefinidamente estendido ao considerarmos escalas cada vez menores, não há uma resposta bem definida para essa pergunta.

No entanto, é possível definir um número entre 1 e 2 que caracteriza o "denteamento" do litoral. No caso da costa britânica esse número é aproximadamente 1.58 e para a costa norueguesa, bem mais acidentada, aproximadamente 1.70. Mandelbrot denominou esse número de *Dimensão Fractal*, visto que ele tem algumas propriedades de dimensão.

Podemos entender intuitivamente essa idéia observando que uma linha denteada num plano ocupa mais espaço que uma linha reta (que tem dimensão 1), mas menos espaço que

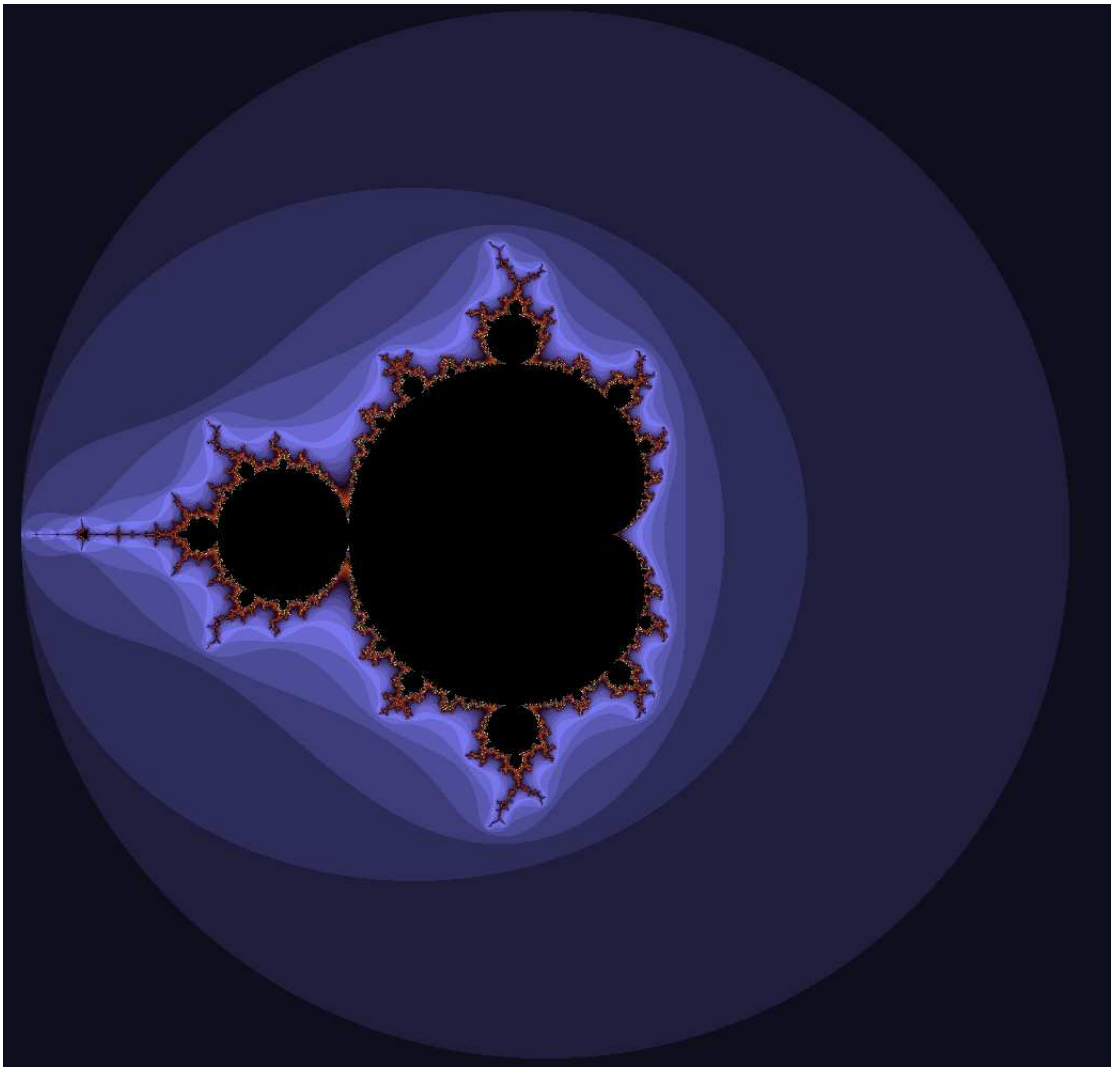


Figura 7.13: Conjunto de Mandelbrot

o plano (que tem dimensão 2). Quanto mais denteada for a linha, maior será sua dimensão. Dessa forma, podemos associar dimensão fractal com quanto espaço uma forma fractal ocupa. O conceito de dimensão fractal fornece uma maneira de se medir a irregularidade dos objetos.

7.2.1 Auto-similaridade

Existem outras formas de compreender o conceito de dimensão fractal, uma delas é observando a auto-similaridade das figuras. Obviamente, uma linha tem dimensão 1, um plano dimensão 2 e um cubo dimensão 3, afirmamos isso porque elas crescem em 1 sentido, 2 sentidos e 3 sentidos, respectivamente, todos linearmente independentes.

Mas o que tais afirmações significam? Observe que não existem sentidos "meio" linearmente independentes. Uma forma alternativa de abordar a questão da dimensão é observar que todos esses objetos são auto-similares. Podemos dividir, por exemplo, um segmento de reta em 4 pedaços auto-similares, todos com o mesmo comprimento e cada um pode ser ampliado pelo fator 4 que se obtém a figura inicial. Em geral podemos dividir o segmento de reta em n partes auto-similares bastando aplicar uma ampliação de fator n num dos pedaços para obtermos o segmento de reta original.

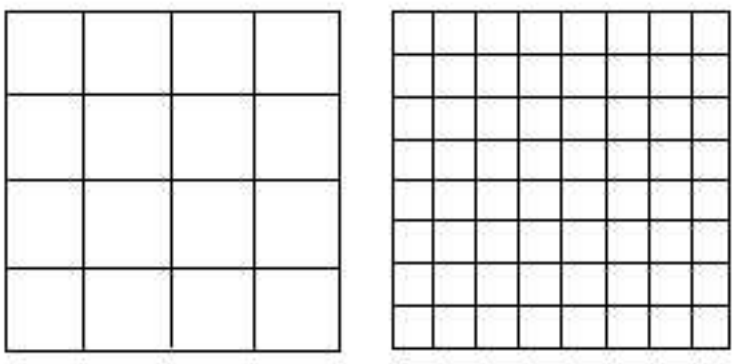


Figura 7.14: Quadrados divididos em 16 e 64 partes auto-similares

No caso de um quadrado, podemos dividi-lo, por exemplo, em 16 partes auto-similares como mostrado na Figura 7.14, mas basta ampliarmos qualquer pedaço pelo fator 4 que obtemos o quadrado original. Em geral podemos dividir um quadrado em n^2 partes auto-similares bastando aplicar uma ampliação pelo fator n num dos pedaços para obtermos o quadrado original.

No caso de um cubo, podemos dividi-lo, por exemplo, em 64 partes auto-similares, mas basta ampliarmos cada pedaço pelo fator 4 que obtemos o cubo original. Em geral podemos dividir um cubo em n^3 partes auto-similares bastando aplicar uma ampliação pelo fator n num dos pedaços para obtermos o cubo original.

Dimensão fractal é uma medida de quão "complicada" uma figura auto-similar é. De forma grosseira, mede "quantos pontos" um determinado conjunto ocupa. Podemos dizer que um plano é "mais largo" que uma linha enquanto outras curvas ocupam algo entre

esses dois conjuntos. A dimensão fractal provê um modo quantitativo de caracterizar a complexidade de curvas levando em conta a auto-similaridade.

Importante observar que um plano não tem mais pontos do que uma reta, visto que esses conjuntos são incontáveis, mas dimensão fractal captura a noção de "quão grande um conjunto é" de forma bem eficiente. Dimensão fractal serve muito bem para expressar quanto uma determinada curva se estende pelo espaço (que é a motivação original para a qual tal medida foi desenvolvida), o que significa que, curvas fractais mais complexas vão ter dimensão fractal maior.

Dessa forma, podemos construir uma forma simples para medir dimensão fractal:

$$F = \text{Dimensão Fractal} = \frac{\log(\text{número de peças auto similares na figura})}{\log(\text{fator de ampliação})}$$

O fator de ampliação é o fator de escala que tem que ser aplicado a qualquer peça auto-similar para que ela se torne igual à figura original.

Para uma linha reta, $F = \frac{\log(n)}{\log(n)} = 1$, para um plano, $F = \frac{\log(n^2)}{\log(n)} = 2$ e para um cubo, $F = \frac{\log(n^3)}{\log(n)} = 3$.

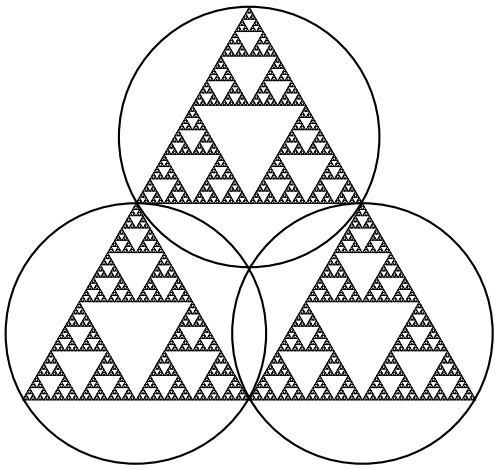


Figura 7.15: Triângulo de Sierpinski com três partes auto-similares assinaladas

Na Figura 7.15 temos três partes auto-similares do triângulo de Sierpinski assinaladas. Para obtermos a figura original, basta aplicarmos o fator de ampliação 2 em qualquer uma das partes auto-similares, desta forma, $F = \frac{\log(3)}{\log(2)} \approx 1.58$.

Para a curva de Koch cujo padrão básico está mostrado na Figura 7.1, temos dimensão fractal $F = \frac{\log(4)}{\log(3)} \approx 1.26$.

7.2.2 Contagem de Bolas

Um método muito conhecido para determinar o valor da dimensão fractal de uma figura é conhecido como contagem de bolas e é baseado nas observações sobre auto-similaridade da Seção 7.2.1.

A idéia é bem simples e consiste em contar, para cada valor r variando numa determinada faixa, qual o número mínimo de bolas de raio r (ou, em algumas variações do algoritmo, quadrados de lado r) que é necessário para cobrir a figura toda. Para valores r_1 e r_2 de r bem escolhidos, podemos dizer que

$$F = \text{Dimensão Fractal} = \frac{\log\left(\frac{\text{número de bolas para } r_1}{\text{número de bolas para } r_2}\right)}{\log\left(\frac{r_2}{r_1}\right)}$$

Na Figura 7.15, para $r = 1$ precisamos de somente uma bola, para $r = \frac{1}{2}$ precisamos de três bolas, para $r = \frac{1}{4}$ precisamos de nove bolas.

Desta forma, podemos dizer que a dimensão fractal do triângulo eqüilátero de Sierpinski é $F = \frac{\frac{1}{3}}{\frac{1}{2}} = \frac{\frac{3}{9}}{\frac{1}{4}} \approx 1.58$.

Para regras de transformação simples a escolha dos diversos valores de r é razoavelmente simples, mas para figuras mais complexas em geral a precisão do método é bem baixa.

7.3 Dimensão Fractal Multi-escala

Os métodos descritos na Seção 7.2.1 ou na Seção 7.2.2 são muito eficientes quando um operador humano é capaz de determinar os parâmetros necessários para o cálculo da dimensão. Quando tal tarefa é atribuída a um computador, ou seja, escreve-se um programa em que dada uma imagem deve-se determinar a dimensão fractal da figura, algoritmos como os descritos anteriormente não são eficientes e precisos o suficiente.

Diversos fatores são responsáveis por tal fato, dentre eles, podemos citar:

- **A resolução finita das imagens**

  Um fractal é uma curva infinita, ao se escolher arbitrariamente uma resolução para se representar a curva, automaticamente informação a respeito da curva é descartada, o que faz com que a "fractalidade" só se expresse em algumas escalas da figura, e não em todas as escalas como acontece quando temos a curva original.

- **Auto-similaridade imperfeita**

  As figuras na natureza não são sempre perfeitamente auto-similares, como as figuras que construímos ao longo deste capítulo. Além disso, arredondamentos, ou mesmo regras complexas podem produzir pequenas variações que podem ser desconsideradas no cálculo de dimensão fractal em uma figura finita.

- **Custo computacional**

  Os algoritmos baseados em contagem de bolas ou outros métodos disponíveis na literatura, em geral, são muito caros do ponto de vista computacional. Minimizar o número de bolas necessário, por exemplo, é um problema de otimização combinatória simples, mas de difícil solução.

Um método proposto por Costa *et al.* [26] para estimar a dimensão fractal de um objeto pela utilização de técnicas multi-escala visa contornar tais problemas.

Diferentemente dos métodos descritos anteriormente, que exploram diretamente a propriedade de auto-similaridade pro meio de uma busca por partes similares, ou da contagem de quanto espaço uma forma ocupa, esse novo método procura determinar a dimensão fractal transformando a imagem.

Esse método utiliza uma técnica conhecida como dilatação morfológica por discos de diferentes raios, de forma que tais operações simulam um "enchimento" da figura original como se faz ao soprar uma bexiga vazia.

Usando tal técnica, analisa-se a dificuldade de se expandir a curva em questão. Um ponto, por exemplo, pode ser expandido em todas as direções possíveis, sem nenhuma restrição. Isso não acontece com uma reta, que já preenche uma direção e, portanto, limita as direções em que o preenchimento pode avançar. Já um plano preenche ainda mais o espaço e, portanto, a liberdade de se preencher o espaço é ainda menor. Observe que quanto maior a liberdade que uma curva permite o seu enchimento, menor é a sua dimensão fractal e vice-versa.

A transformação morfológica é razoavelmente barata computacionalmente comparando-a com outros métodos atualmente disponíveis. Além disso, o enchimento é capaz de lidar bem com imperfeições da figura, como problemas de arredondamento ou auto-similaridade imperfeita. O problema principal, que é a escala em que a fractalidade se expressa, pode ser abordado utilizando uma abordagem multi-escala simples, que consiste em avaliar a variação da dificuldade de encher a curva em relação à variação do raio. Essa extensão é a essência dessa nova técnica que permite calcular a dimensão fractal de figuras com precisão bem mais alta, além de permitir uma análise mais profunda de alguns aspectos da fractalidade da figura.

7.3.1 Implementação do Método

Tal método é baseado 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.

O algoritmo em si é bem simples:

1. Lê a entrada e os parâmetros
2. Lê a imagem e a normaliza
3. Monta o cubo (espaço em que a expansão é realizada)
4. Calcula a transformada da distância
5. Calcula o histograma da transformada
6. Calcula o log das áreas e dos volumes
7. Interpola e deriva os resultados
8. Filtra e apresenta os resultados finais

Ao longo deste trabalho desenvolvemos uma versão eficiente deste algoritmo que consiste na utilização de uma transformada eficiente da distância que foi desenvolvida por Lotufo *et al.* [84].

7.3.2 Exemplos

Na Figura 7.16 temos um exemplo de tal curva calculada a partir da Figura 6.14. 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. No eixo x, temos o log do raio e no eixo y a dimensão fractal.

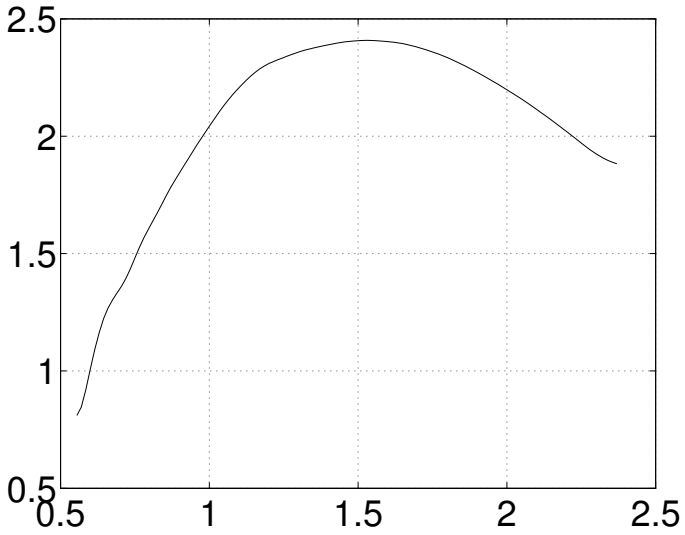


Figura 7.16: Curva de fratalidade da Figura 6.14

Nas Figuras 7.17 e 7.18 apresentamos um quadro com figuras **CGR** calculadas para diferentes genomas e respectivos gráficos de dimensão fractal tridimensional. Apesar de visualmente algumas curvas serem bem semelhantes, principalmente por causa do tamanho em que estão apresentadas, a curva fractal é eficiente para classificar o genoma de espécies.

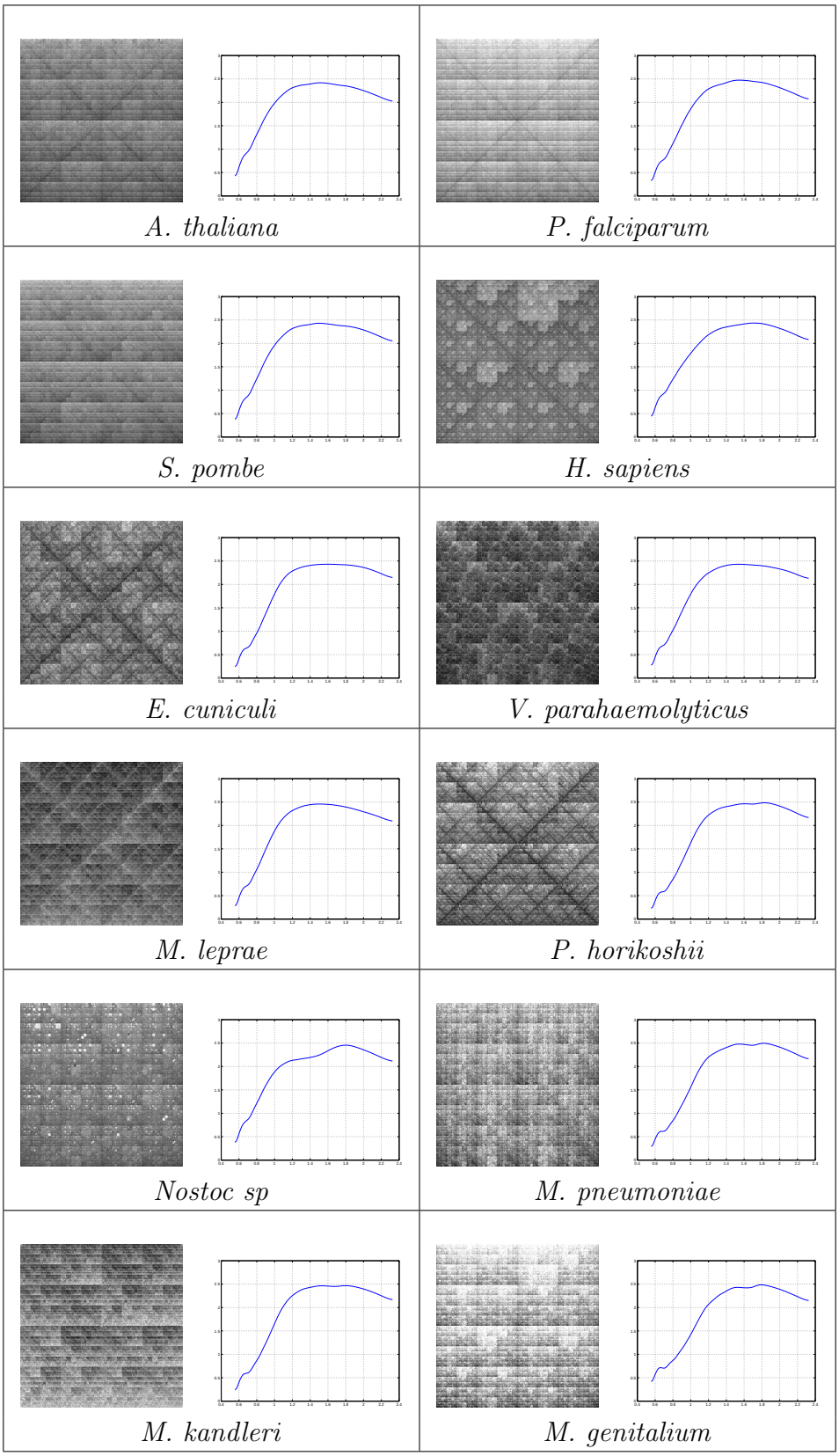


Figura 7.17: Imagens **CGR** normalizadas por log do genoma de diversas espécies usando janela de tamanho 8 e respectivas dimensões fractal tridimensional

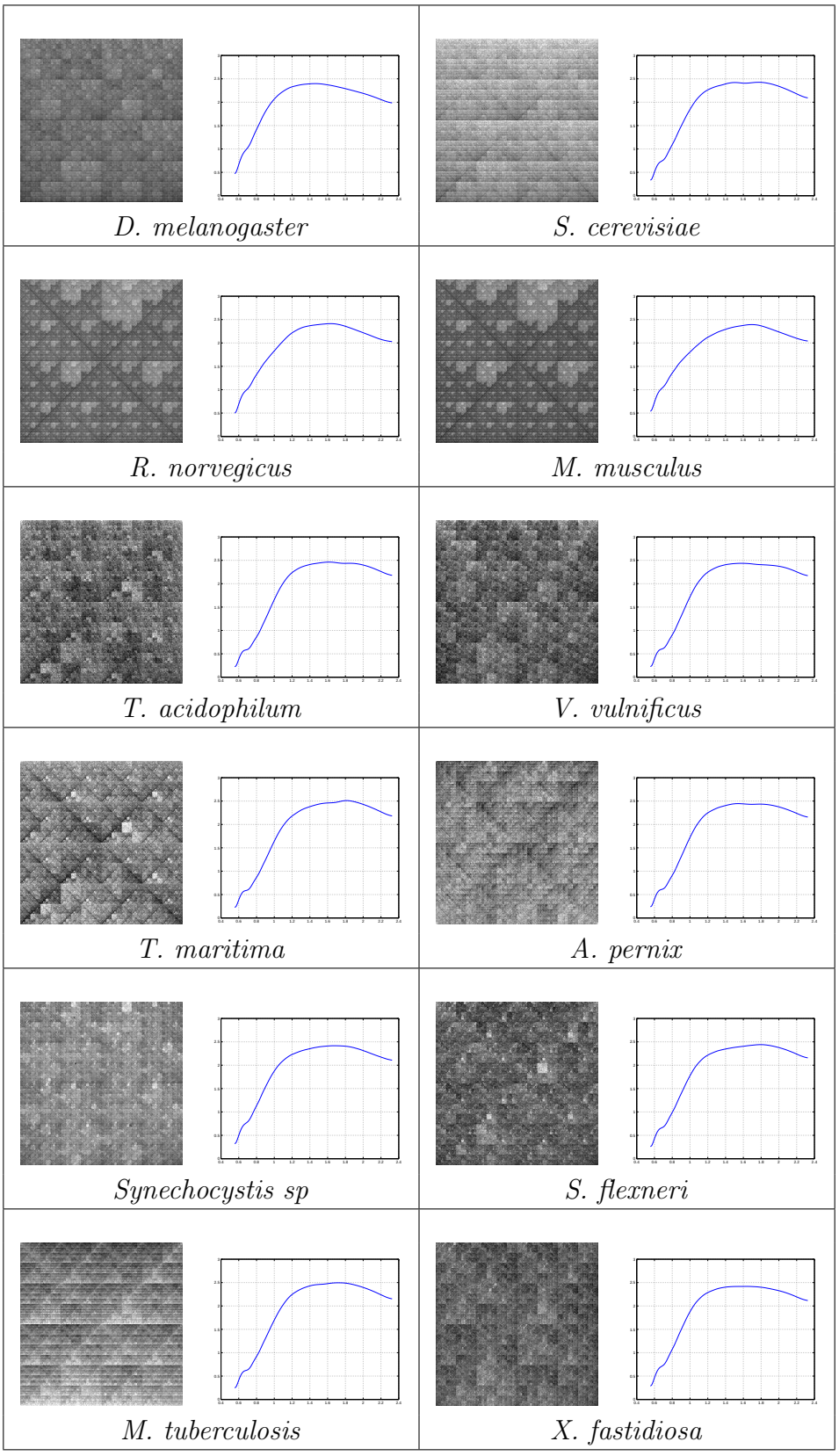


Figura 7.18: Imagens **CGR** normalizadas por log do genoma de diversas espécies usando janela de tamanho 8 e respectivas dimensões fractal tri-dimensional

Capítulo 8

Reconhecimento de Genes

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

Ao longo deste 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 no Capítulo 7.

Foram realizados diversas especificações, implementações e testes preliminares que serão descritos nas próximas seções e no Apêndice A.

Ao longo deste trabalho pesquisamos novos modelos e medidas para a busca de genes "raros", ou seja, genes que raramente são transcritos pelas células. Desenvolvemos uma abordagem para a busca de genes, bem como um ambiente de testes. Utilizamos tal ambiente para testar novas características que extraímos a partir de técnicas **CGR** e dimensão fractal. Os experimentos e resultados serão descritos nas próximas seções.

8.1 Modelo para busca de genes

Tendo em vista as limitações dos algoritmos de reconhecimento de padrões discutidas na Seção 5.3, pesquisamos novos tipos de características.

As características que geramos são:

- Imagem **CGR** da seqüência
- Imagem **CGR**, normalizada usando logaritmo, da seqüência
- Curva de dimensão fractal da imagem **CGR** da seqüência
- Curva de dimensão fractal da imagem **CGR**, normalizada usando logaritmo, da seqüência

Essas características utilizam os elementos apresentados nos Capítulos 6 e 7. A idéia básica é considerar essas 4 novas características em diferentes resoluções. Tais características podem atuar como um filtro, auxiliando técnicas baseadas em **GHMM** ou mesmo como uma regra de *clustering* para gerar agrupamentos como mostrado na Figura 5.2 (página 51).

O objetivo não é substituir nenhum dos métodos apresentados no Capítulo 5, mas agir em conjunto com eles, visto que as regiões filtradas podem servir como entrada para os algoritmos de predição de genes já existentes.

Para o caso de *clustering*, avaliamos se as novas características são eficientes para a geração de partições do espaço de treinamento de forma que permita um treinamento diferenciado para cada região da partição gerada. O classificador final simplesmente realiza a consolidação dos resultados obtidos com os classificadores de cada região da partição. Dependendo do caso, isso pode permitir maior generalização, mas sem grande perda na precisão, principalmente no caso em que poucos exemplos de treinamento estão disponíveis.

Já o segundo tipo de utilização das características é mais interessante: projeta-se um filtro que varre o genoma em busca de regiões candidatas a genes e, nessas regiões, aplica-se algoritmos para a busca.

Como os algoritmos para a busca de genes são muito caros computacionalmente, um filtro que eventualmente consiga apontar com alta capacidade de cobertura tais regiões é muito útil.

A idéia básica é escolher arbitrariamente uma janela, ou conjunto de janelas. Desliza-se então a janela pelo genoma andando p bases em cada passo. Para cada passo, calcula-se a característica desejada e compara-se com os dados de treinamento. Pode-se, então, usar alguma métrica, como a baseada em k-vizinhos, por exemplo, em que se leva em conta os k elementos mais próximos do ponto gerado. Toma-se uma decisão, como, por exemplo, se a região "se parece" mais com regiões de gene ou com regiões de não gene.

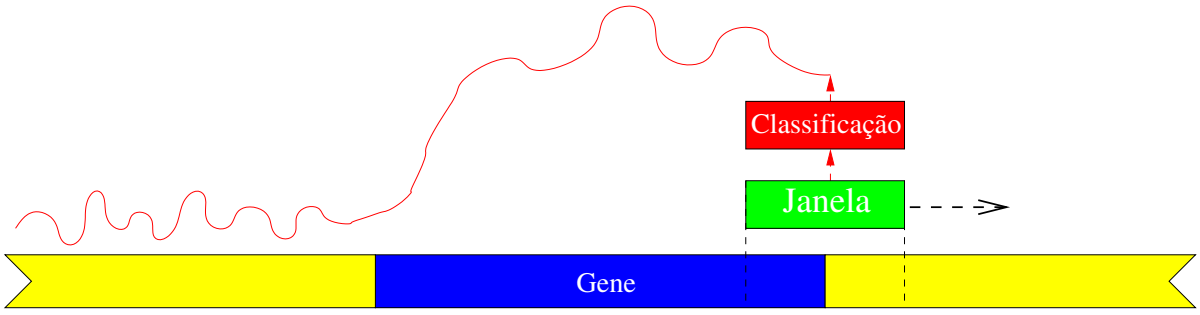


Figura 8.1: Exemplo de técnica para a busca de gene

Como em cada passo gera-se um número, temos uma curva para o genoma como exemplificado na Figura 8.1, que pode ser analisada com diversas técnicas, dentre elas, as mais simples, como a definição de um limiar global ou local.

Dessa forma, analisando a curva pode ser que seja possível apontar regiões de gene de forma computacionalmente menos custosa.

8.1.1 Curva de Domínio Protéico

Um caso particular das formas de geração de curvas discutidas é a curva de domínio protéico. Tal curva consiste em escolher arbitrariamente um conjunto de proteínas e deslizar ao longo do genoma uma janela determinando quão "parecida" a janela é em relação ao conjunto de proteínas.

Compara-se então a freqüência de subpalavras dos dois conjuntos. Ou seja, calcula-se a freqüência de cada subpalavra em cada conjunto e compara-se a tabela de freqüências gerando uma pontuação.

Uma forma simples de gerar essa pontuação é calculando a somatória da multiplicação de cada ponto correspondente à mesma subpalavra em cada tabela.

Inicialmente analisamos as curvas geradas por esse método, mas ao longo do trabalho preferimos dar ênfase à análise de curvas geradas pelo método que usa imagens **CGR**, visto que a tabela de freqüências de proteínas é, exatamente, uma imagem **CGR**, ou seja, a curva de domínio protéico é um caso particular que associa imagens **CGR** utilizando a função de pontuação particular apresentada acima.

8.2 Experimentos com Aprendizado Não Supervisionado

Inicialmente realizamos uma série de 25 experimentos de *clustering* na tentativa de avaliar se imagens **CGR** e a medida fractal são eficientes para a busca de genes. 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)

8.2.1 Metodologia

Foram extraídas as características acima de 526 genes do cromossomo 22 (seqüência do gene completa) versão Ensembl [161] 7.29 e de 655 regiões contínuas do mesmo cromossomo em que se acredita fortemente não haver genes ou pedaços de genes.

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

Para cada uma das características, foi calculada uma matriz de distâncias e foram utilizados as 5 métricas disponíveis no **Matlab** [160] (totalizando 25 experimentos) para se tentar realizar a separação. Importante observar que, para calcular as matrizes de distâncias, foram utilizados as seguintes métricas:

Para o conteúdo **GC** e as imagens **CGR** normalizadas linearmente e por log simplesmente realiza-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 subtrai-se duas vezes o valor da intersecção entre as duas integrais. Não foi usada nenhuma

dentre as informações adicionais que são disponibilizadas pela técnica de dimensão fractal multi-escala.

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. Em todas as imagens **CGR** utilizamos janela de tamanho 6.

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" a um agrupamento se nele 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 falso positivos (FP – Atribuição de rótulo de gene para seqüências que não são gene) e o número de falso negativos (FN – Atribuição de rótulo de não gene para seqüência que são genes).

8.2.2 Resultados

Na Figura 8.2 temos um gráfico somente com as taxas de acerto calculadas para as cinco medidas apresentadas anteriormente utilizando os métodos de *clustering Complete*, *Average*, *Centroid* e *Ward* respectivamente. As cores são como segue:

- Magenta – Dimensão Fractal Multi-escala da imagem **CGR**
- Ciano – Dimensão Fractal Multi-escala da imagem **CGR** normalizada por log
- Vermelho – Imagem **CGR**
- Verde – Imagem **CGR** normalizada por log
- Azul – Conteúdo **GC**

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.

Observe que os resultados são consistentes independentemente do método utilizado.

Aplicamos também um método simples de *bootstrap* que consistiu em selecionar 100 conjuntos de testes de tamanho ligeiramente menor que o original. Cada conjunto consiste de 350 genes e 350 regiões de não-genes selecionadas randomicamente.

Na Figura 8.3 os resultados para o método de *clustering Complete*. Cada gráfico representa uma característica diferente.

Para os primeiros 5 gráficos, em preto e amarelo temos a média de falsos positivos e falsos negativos com barras de erros, respectivamente. O último gráfico mostra a média de acerto de cada característica.

Na Figura 8.4 temos o gráfico que mostra a diferença na taxa de acerto de cada método em relação ao conteúdo **GC** com barras de erro.

Interessante observar que independentemente do método de *cluster* utilizado, as curvas obtidas são bem semelhantes às apresentadas na Figura 8.3. Em todos os experimentos, a

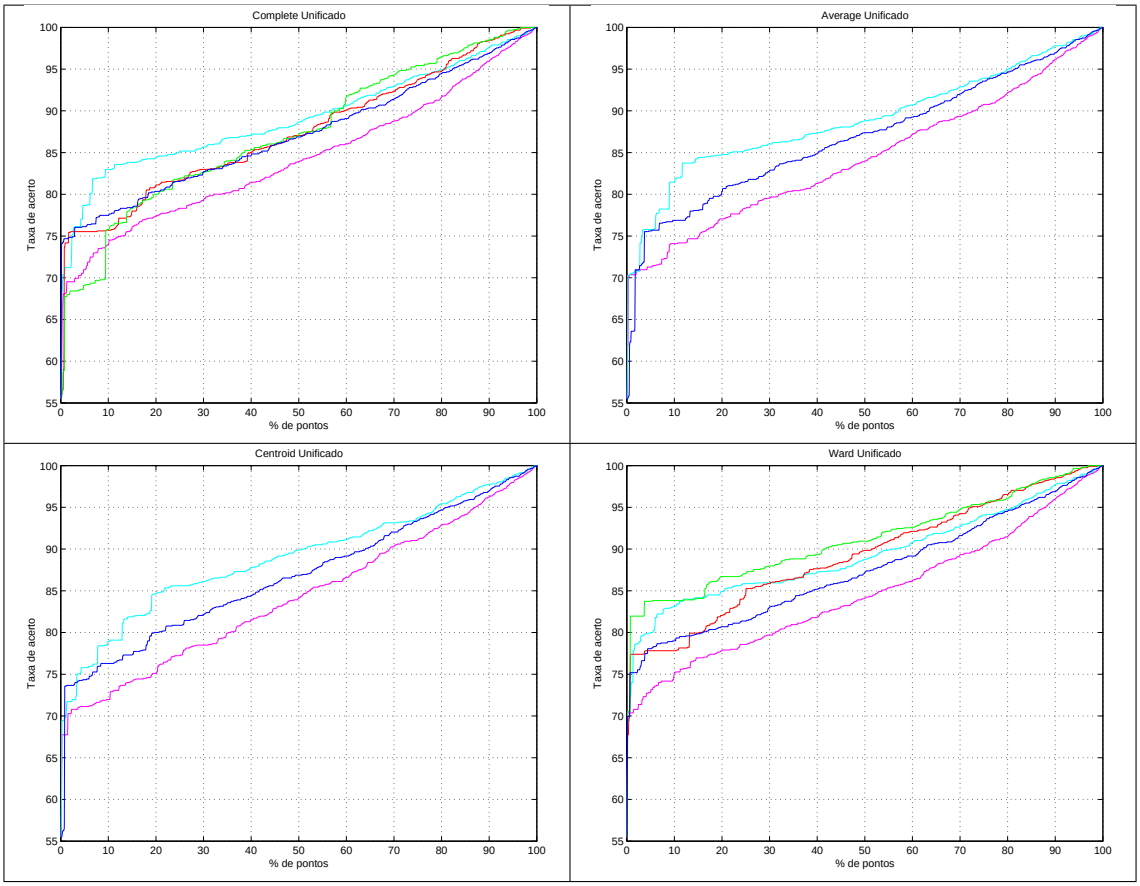


Figura 8.2: Resumo dos resultados obtidos com 4 métodos de *clustering*

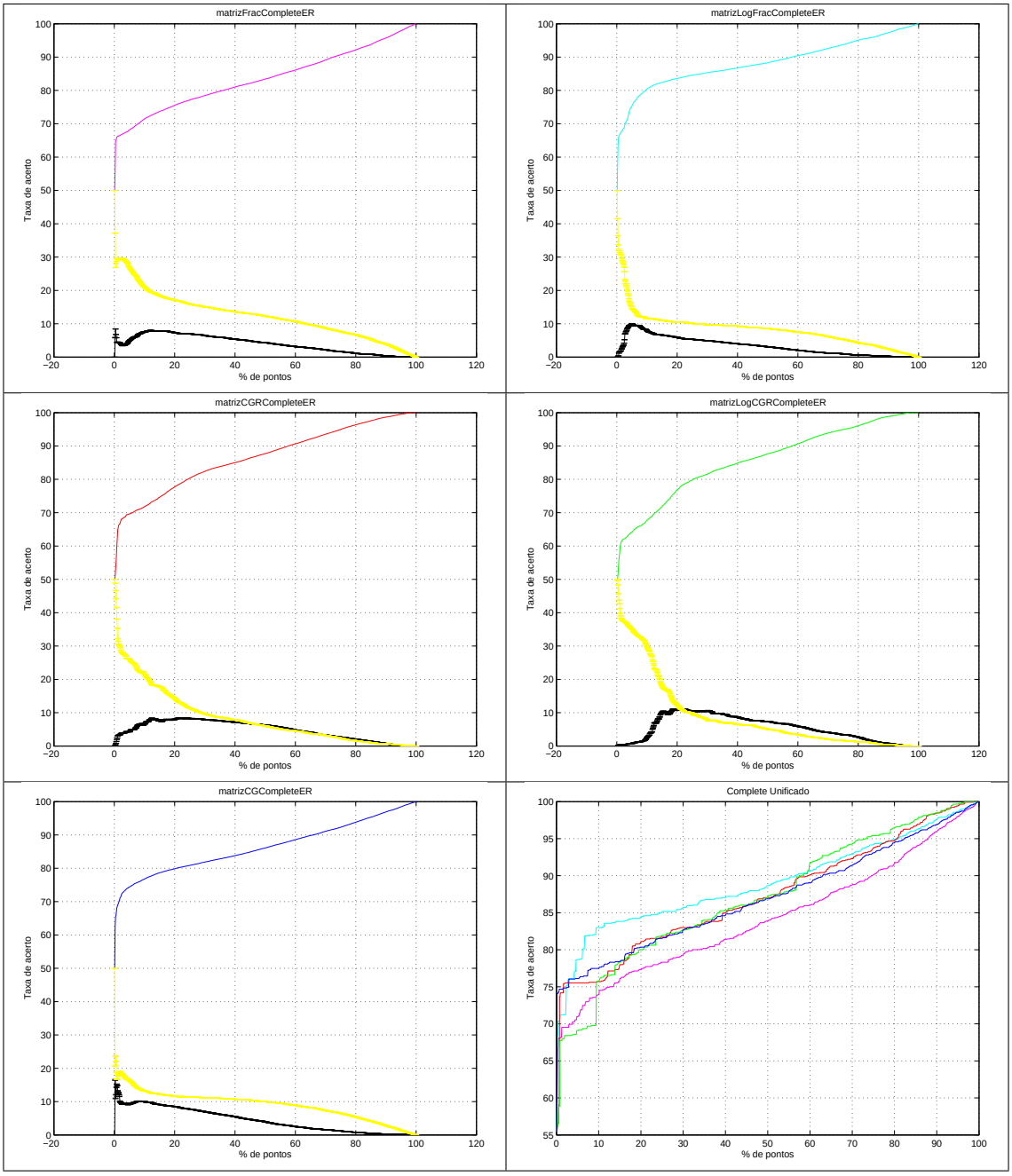


Figura 8.3: Resumo dos resultados obtidos com o método *Complete*. As curvas em preto e amarelo correspondem à média de falsos negativos e falsos positivos com barras de erros, respectivamente

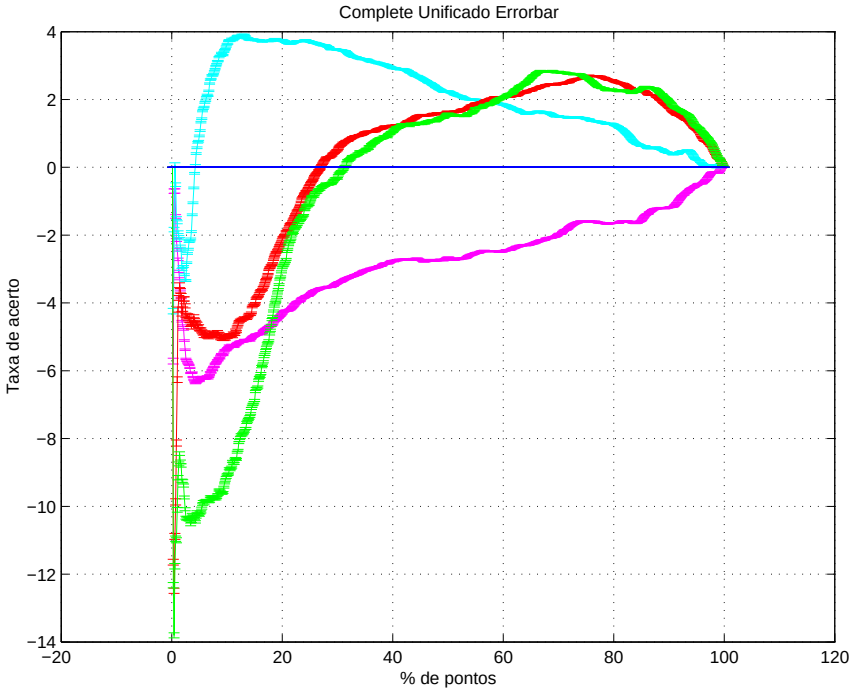


Figura 8.4: Diferença da taxa de erro com barras de erro em relação ao conteúdo **GC**

curva azul claro (correspondente ao método baseado em dimensão fractal com normalização pela função log) dá resultados 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*.

Nas Figuras 8.5, 8.6, 8.7, 8.8, e 8.9 temos os resultados do *bootstrap* para a mesma metodologia, utilizando porém a versão 13.31 do Ensembl e variando o tamanho da janela e a profundidade das imagens. Só estão mostrados os primeiros 20% dos gráficos, parte que interessa. Os testes com resolução 7 deram os melhores resultados. Observe que há mais duas curvas nos gráficos que correspondem a formas distintas de preencher a curva fractal pelo processo de dilatação. Utiliza-se um disco planar em vez de uma bola nas dilatações. As curvas em amarelo representam a dimensão fractal multi-escala planar da imagem **CGR** normalizada por log e as em preto representam a dimensão fractal multi-escala planar da imagem **CGR** normalizada linearmente.

8.2.3 Análise de Agrupamentos

Durante o processo de *bootstrap* descrito na Seção 8.2.2 observamos que alguns genes sempre tendem a ficar no mesmo agrupamento quando o algoritmo de *clustering* é forçado a gerar um número de agrupamentos maior do que 1% do número de pontos.

Avaliamos essa propriedade com o objetivo de verificar se há alguma relação *funcional* entre os genes que permanecem sempre no mesmo agrupamento. Tal propriedade poderia

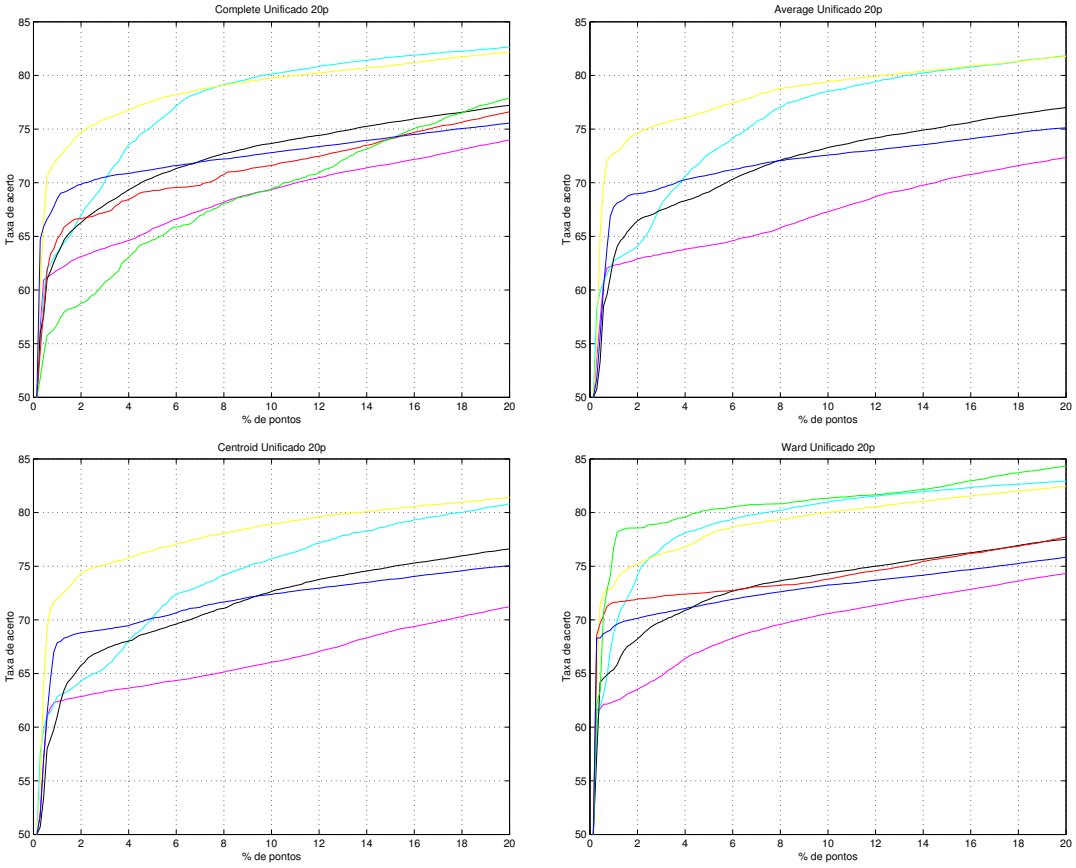


Figura 8.5: Resumo dos resultados dos métodos de *clustering*, primeiros 20%, janela tamanho 6 e profundidade 64

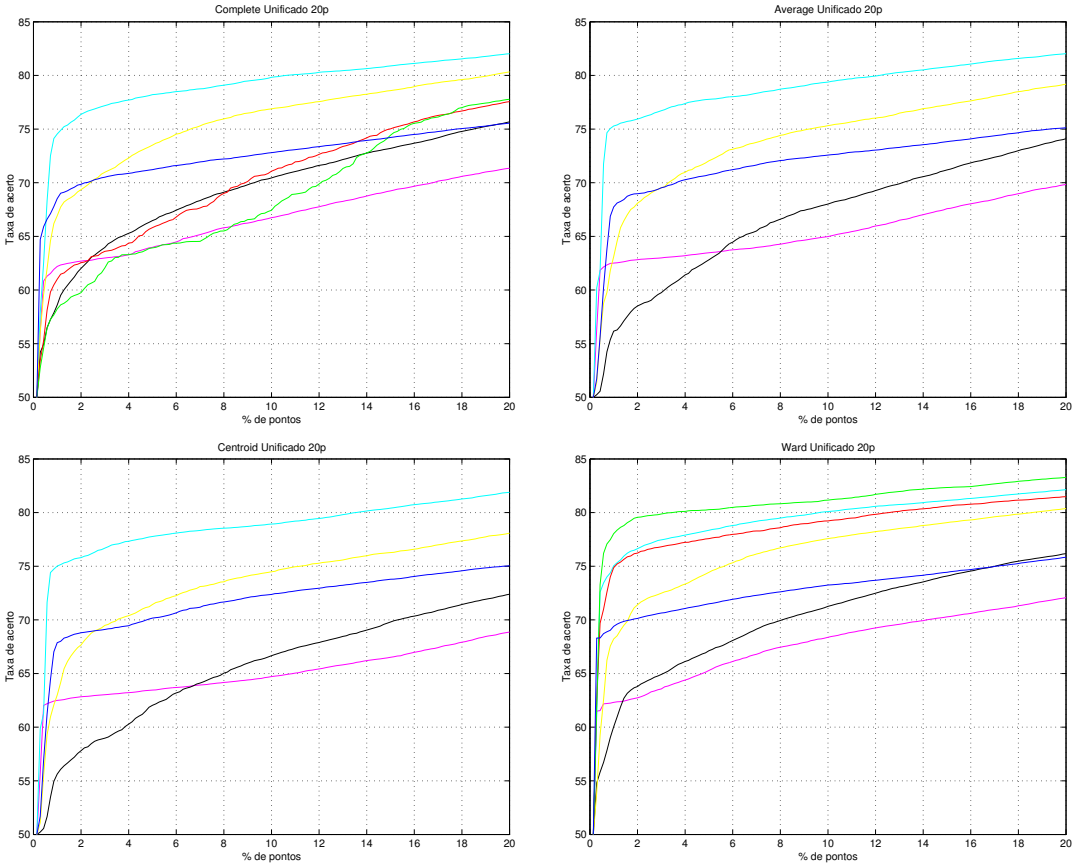


Figura 8.6: Resumo dos resultados dos métodos de *clustering*, primeiros 20%, janela tamanho 7 e profundidade 64

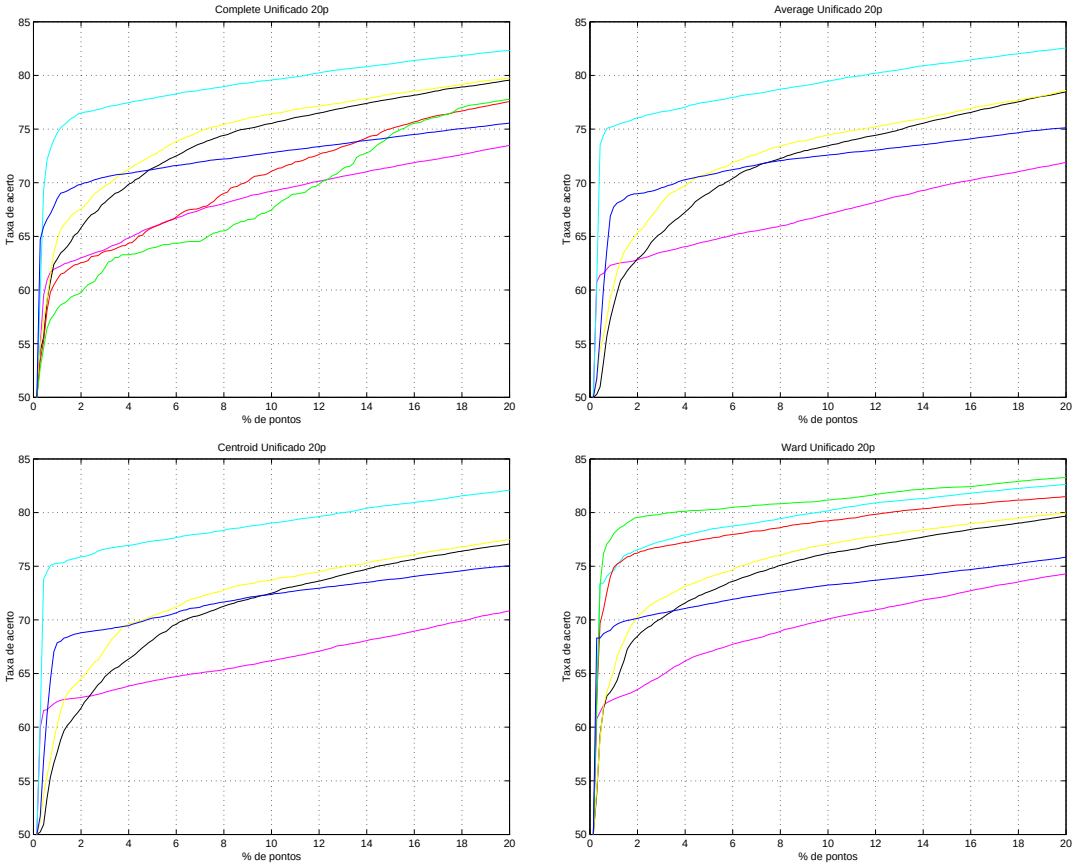


Figura 8.7: Resumo dos resultados dos métodos de *clustering*, primeiros 20%, janela tamanho 7 e profundidade 128

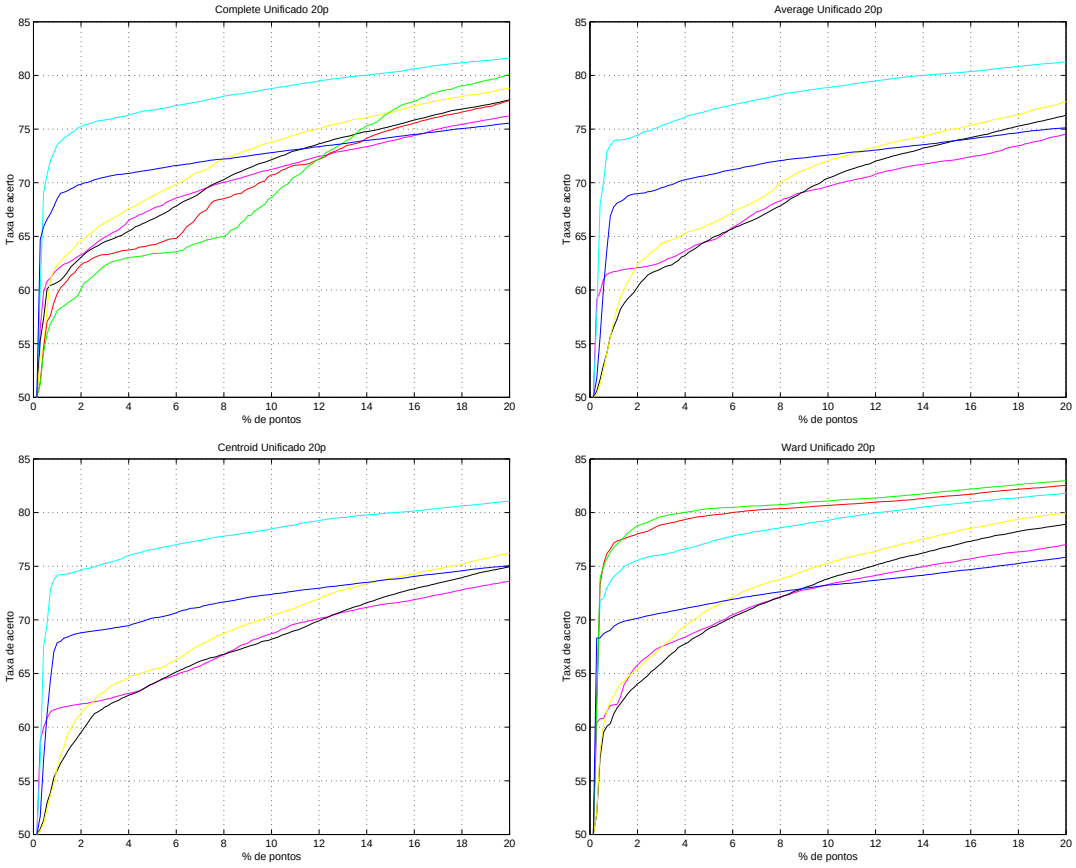


Figura 8.8: Resumo dos resultados dos métodos de *clustering*, primeiros 20%, janela tamanho 8 e profundidade 64

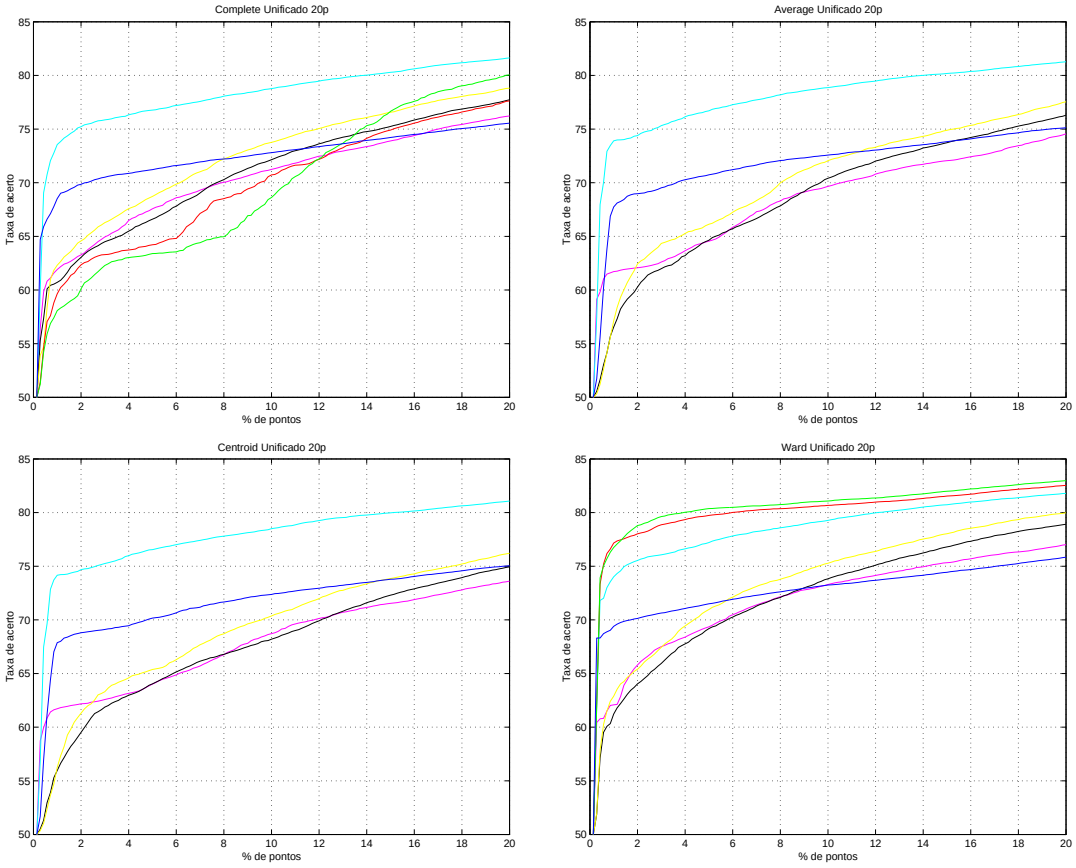


Figura 8.9: Resumo dos resultados dos métodos de *clustering*, primeiros 20%, janela tamanho 8 e profundidade 128

ser útil para revelar relações *funcionais* desconhecidas, ou se desenvolver algoritmos mais flexíveis como os baseados na discussão da Seção 5.3.

Não encontramos nenhuma relação *funcional* entre os genes que permanecem no mesmo agrupamento.

8.2.4 Análise dos Resultados

Apesar dos resultados apresentados na Seção 8.2.3 não serem muito animadores (visto que os índices de separação obtidos são menores que o esperado para um bom classificador), os resultados apresentados na Seção 8.2.2 são bem interessantes, visto que não utilizam nenhuma informação biológica adicional. Apesar da separação não ser excelente, os resultados desses experimentos não servem para comparar as medidas, simplesmente servem para verificar se elas são consistentes.

Observe que nesses experimentos a medida baseada em dimensão fractal normalizada por log gerou resultados interessantes, enquanto a mesma medida com normalização linear foi a pior de todas. O fato da técnica baseada em dimensão fractal ter dado resultados significativamente superiores à técnica baseada em conteúdo **GC**, que são muito utilizadas por algoritmos baseados em **HMM**, sugere que a medida eventualmente possa ser utilizada em substituição ou complemento ao conteúdo **GC** nesses algoritmos.

De qualquer forma, esses ainda são resultados preliminares. É necessária ainda uma série de experimentos para validarem a utilidade dessas características. 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.

Na próxima seção, apresentamos os resultados de uma série de experimentos supervisionados que buscam inferir de forma mais precisa a capacidade das características aqui avaliadas.

8.3 Experimentos com Aprendizado Supervisionado

Tendo em vista que os resultados apresentados na Seção 8.2 não são conclusivos, ou seja, significam simplesmente que as características em questão são consistentes no sentido que elas mantêm informações necessárias para a classificação, resolvemos determinar o verdadeiro potencial de tais medidas.

Para tal, é necessário realizar experimentos supervisionados. Foram realizados experimentos que completam os da Seção 8.2, dentre eles, testes de taxa de acerto, cobertura e precisão de diversos métodos.

O algoritmo básico de classificação é bem simples. O classificador é composto da característica escolhida dos dados positivos (genes) e negativos (região de não-genes) e a classificação constitui em determinar os k vizinhos mais próximos de cada elemento do classificador. Se a maioria dos k vizinhos mais próximos é gene, classifica-se a região como gene e vice-versa. Também desenvolvemos uma versão do classificador que leva em conta não só o número de vizinhos, mas também as distâncias dos vizinhos, o que corresponde a um k-vizinho balanceado.

Adicionamos mais duas medidas, que correspondem à uma forma diferente de "encher" a curva, em vez de uma bola como descrito na Seção 7.3, utilizamos um disco planar, dessa forma, temos as seguintes cores:

- Magenta – Dimensão Fractal Multi-escala da imagem **CGR**
- Ciano – Dimensão Fractal Multi-escala da imagem **CGR** normalizada por log
- Preto – Dimensão Fractal Multi-escala planar da imagem **CGR**
- Amarelo – Dimensão Fractal Multi-escala planar da imagem **CGR** normalizada por log
- Vermelho – Imagem **CGR**
- Verde – Imagem **CGR** normalizada por log
- Azul – Conteúdo **GC**

8.3.1 Reconhecimento de Genes do Cromossomo 22

Realizamos um *bootstrap* (com os mesmos conjuntos do *bootstrap* da Seção 8.2 para a versão 13.31 do Ensembl) em que utilizamos o conjunto sorteado como conjunto de treinamento (aproximadamente 75% do total de pontos) e os não sorteados como conjunto de teste. Esse teste permite observar o comportamento do espaço de hipóteses.

Na Figura 8.10 temos um exemplo de resultado do *bootstrap* para janela de tamanho 7 e profundidade 64 e k ímpar variando de 1 a 21. Observe que os melhores resultados foram obtidos com as medidas de dimensão fractal, que chegam a apresentar 80% de taxa de acerto, com 90% de precisão e 90% de cobertura aproximadamente, o que é muito bom. Observe que a cobertura das imagens **CGR** chega a quase 100% (quinto e sexto gráfico em amarelo).

Na Figura 8.11 temos um comparativo da diferença em relação ao conteúdo **GC** com barras de erro.

Os resultados para outras resoluções são similares, um resumo é apresentado na Figura 8.12.

Utilizando também não só o número de vizinhos mais próximos, mas também a distância dos vizinhos (o que equivale a um k-vizinhos balanceado), os resultados são praticamente idênticos.

8.3.2 Teste de Cobertura

Os resultados da seção anterior são muito bons, visto que estão muito próximos de uma situação real de reconhecimento de padrões, visto que foram utilizados poucos exemplos de treinamento e muitos de teste. O *bootstrap* como feito é extremamente significativo e sugere que as novas medidas propostas têm um grande potencial de classificação.

Para confirmar tal conjectura, realizamos uma série de experimentos de cobertura que consiste em considerar como dados de treinamento os elementos do cromossomo 22 e os genes dos outros cromossomos como dados de testes. Esta é uma situação real de classificação,

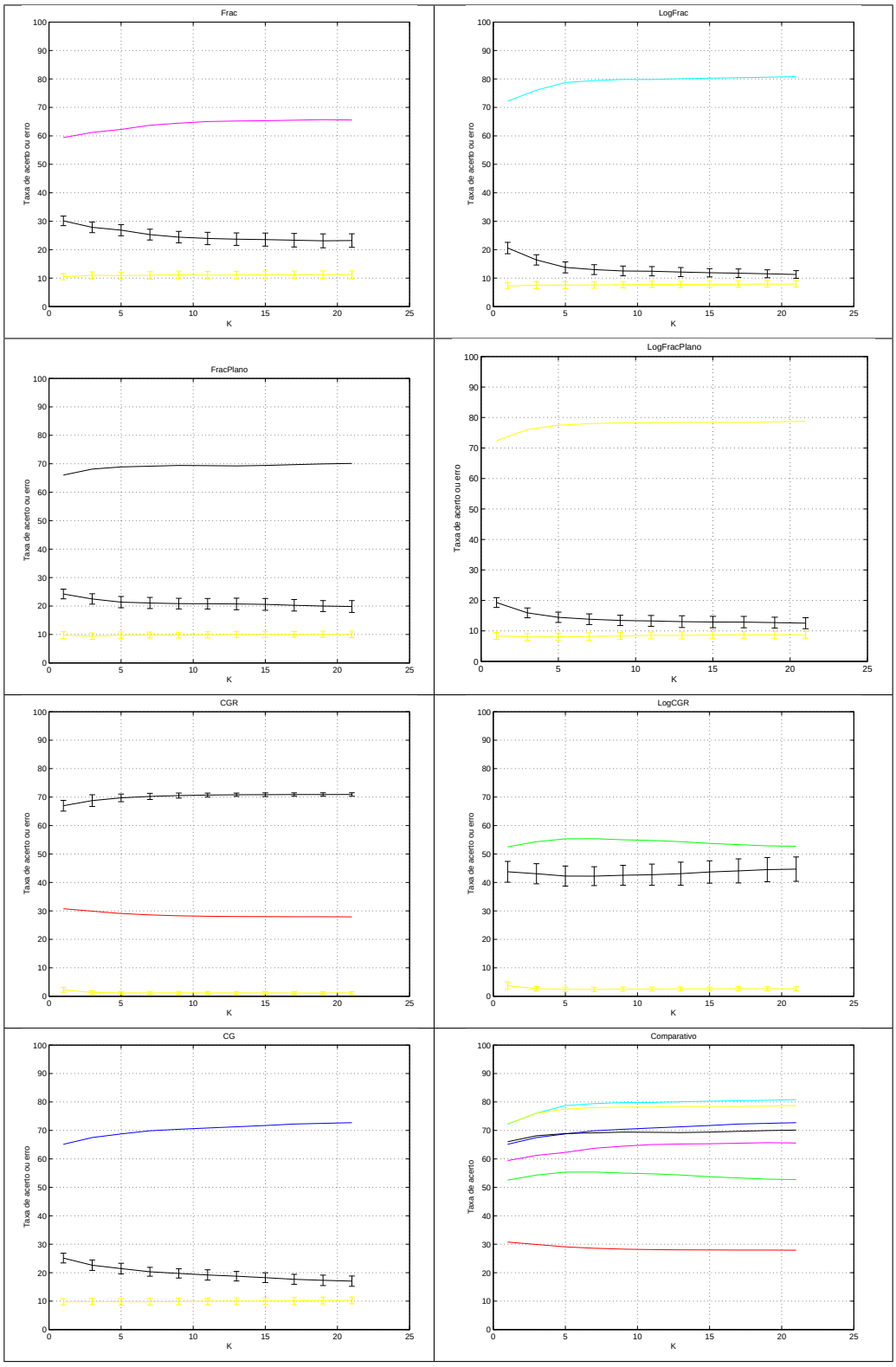


Figura 8.10: Resumo dos resultados do *bootstrap* obtidos utilizando a classificação pelos k-vizinhos, janela de tamanho 7 e profundidade 64. As curvas em preto e amarelo correspondem à média de falsos negativos e falsos positivos com barras de erros, respectivamente

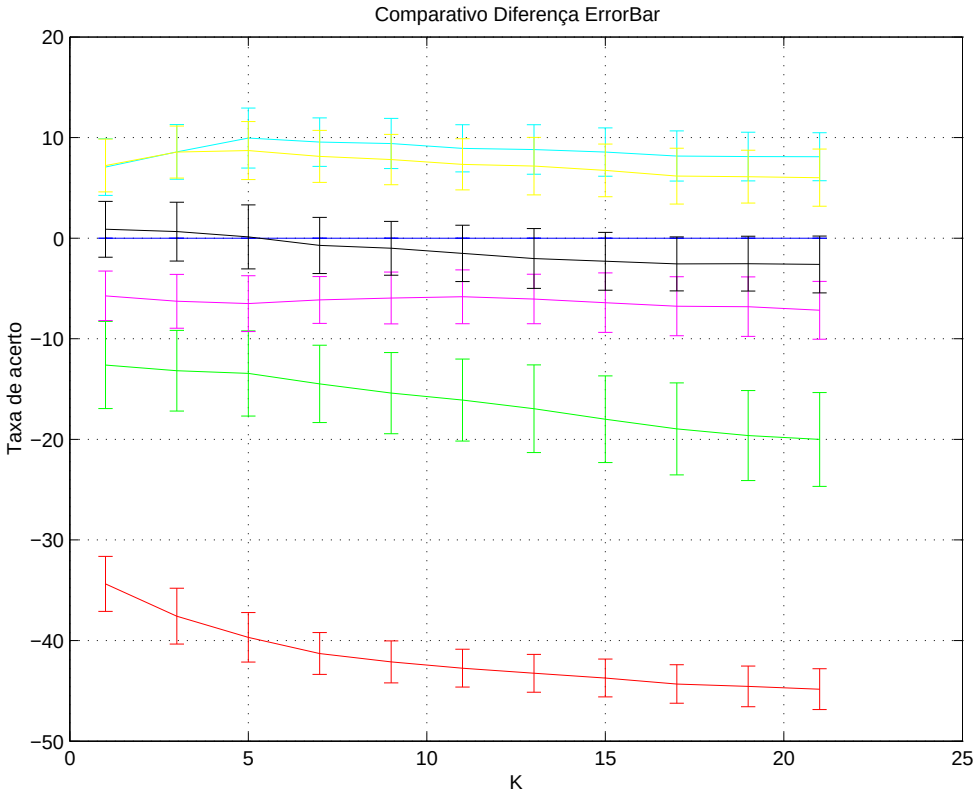


Figura 8.11: Diferença dos resultados do *bootstrap* utilizando k-vizinhos, janela de tamanho 7 e profundidade 64 com barras de erros

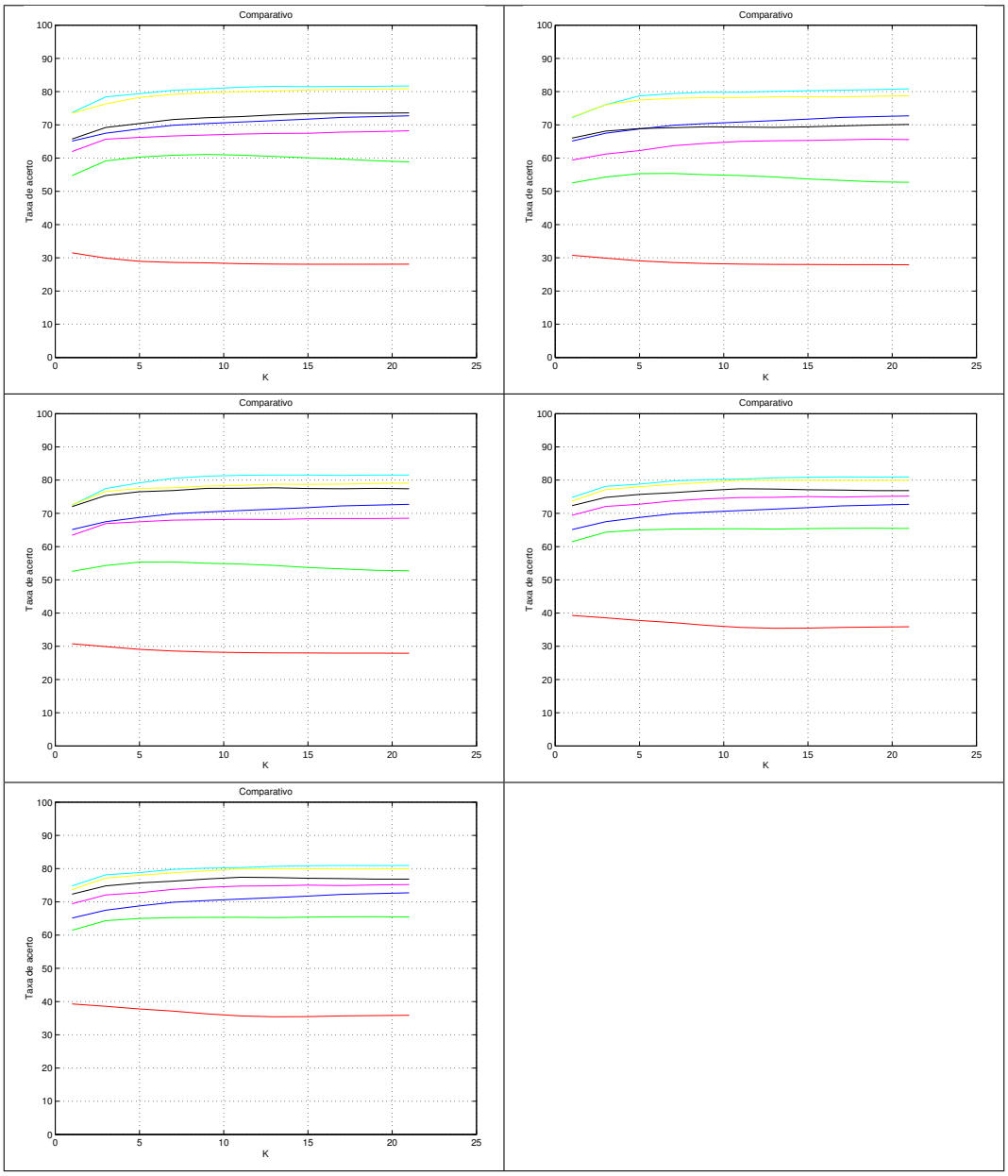


Figura 8.12: Resumo dos resultados do *bootstrap* obtidos utilizando a classificação pelos k-vizinhos, diferentes janelas e resoluções ($6-64$, $7-64$, $7-128$, $8-64$ e $8-128$)

em que são utilizados poucos pontos de treinamento (cerca de 576 genes do cromossomo 22) e muitos pontos de testes (os 23.067 genes restantes do genoma humano, Ensembl versão 15.33).

Realizamos esse teste, que consistiu simplesmente em determinar a cobertura de cada medida utilizando o mesmo método já descrito. Inicialmente o teste foi feito com os genes do cromossomo 21 e os resultados podem ser observados na Figura 8.13. Note que a taxa de cobertura tende a zero quando k aumenta, isso acontece porque o conjunto de treinamento tem mais pontos negativos do que positivos.

Na Figura 8.14 temos o mesmo gráfico consolidado mas somente para k ímpar entre 1 e 21 e para diferentes resoluções. Observe que a cobertura dos métodos de dimensão fractal chegam a 75%, em alguns testes chegou a 80%, o que é bem interessante e indica boa capacidade de generalização. Observe que métodos baseados em **CGR** puro têm taxa de cobertura maior do que 90%, o que também é consistente com os resultados obtidos na seção anterior.

Note que o nosso controle, o conteúdo **GC**, simplesmente não consegue classificar bem os genes, o que também é consistente com o que se têm observado na área.

Apesar de ser impressionante a taxa de cobertura de 90% que os métodos baseados em **CGR** obtiveram, é importante lembrar que a precisão não é avaliada aqui. Os testes descritos na Seção 8.3.3 sugerem que a precisão de tais métodos é muito baixa.

Também realizamos o mesmo teste com os outros genes do genoma humano. Consideramos os elementos do cromossomo 22 como exemplos de treinamento e os genes de todos os outros cromossomos exceto o 22 como exemplos de testes. Os resultados obtidos são semelhantes aos apresentados para o experimento que leva em conta somente os genes do cromossomo 21.

Na Figura 8.15 temos os resultados do experimento para resolução 7 e profundidade 128 e na Figura 8.16 nós temos o mesmo gráfico consolidado mas somente para k ímpar entre 1 e 21 e para diferentes resoluções.

8.3.3 Teste de Precisão

Uma das grandes dificuldades na área de busca de genes é a ausência de exemplos negativos de treinamento, o que dificulta muito a determinação da precisão dos métodos. Uma idéia alternativa é considerar a região cromossômica ao redor de um gene.

Para testar a precisão das características, utilizamos dois métodos principais:

- Utilizando como exemplo negativo de teste as regiões laterais aos genes

  A idéia básica é considerar as regiões imediatamente laterais aos genes como exemplos negativo de testes que devem ser classificados como região de não gene.

  Podem-se escolher as regiões de diversas formas. Realizamos a escolha de dentro para fora, que avaliam as regiões de tamanho entre 2.000 e 30.000 pares de bases, dispostas como mostrado na Figura 8.17.

  O objetivo deste método é testar a capacidade de determinar as bordas dos genes durante uma classificação.

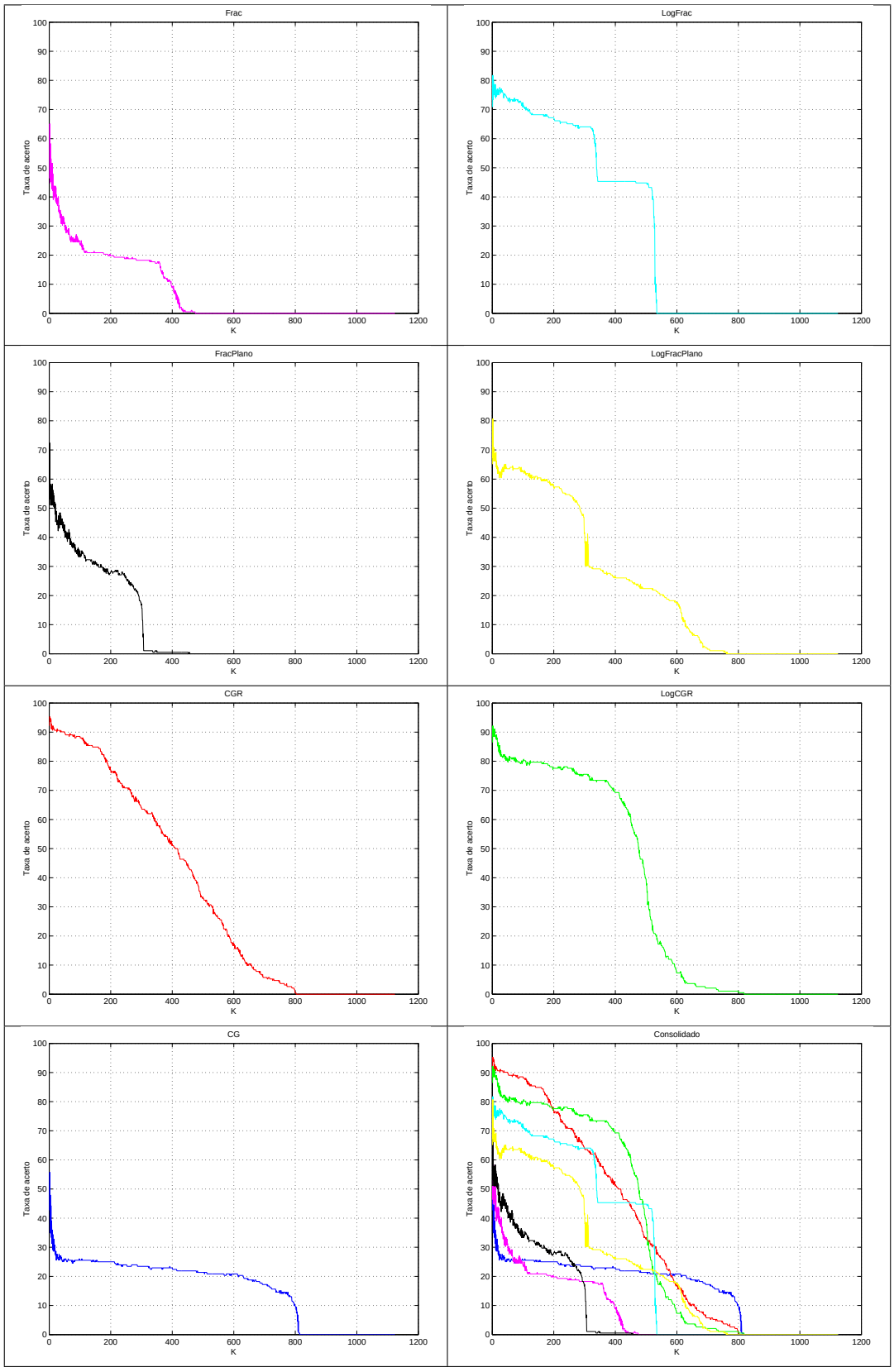


Figura 8.13: Resumo dos resultados obtidos para cobertura do cromossomo 21 utilizando a classificação pelos k-vizinhos, janela de tamanho 7 e profundidade 64

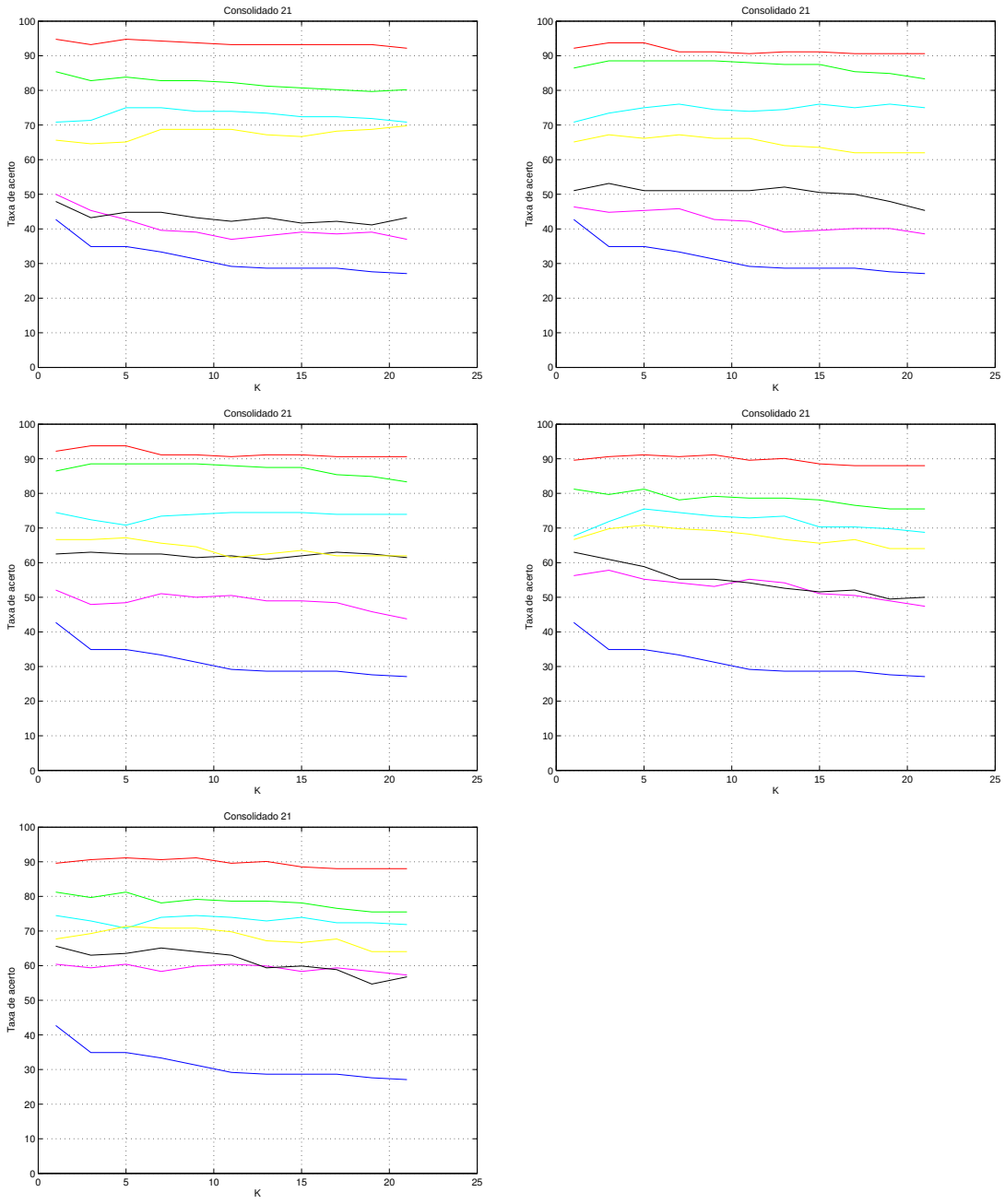


Figura 8.14: Resumo dos resultados obtidos para cobertura do cromossomo 21 utilizando a classificação pelos k-vizinhos, diferentes janelas e resoluções ($6-64$, $7-64$, $7-128$, $8-64$ e $8-128$)

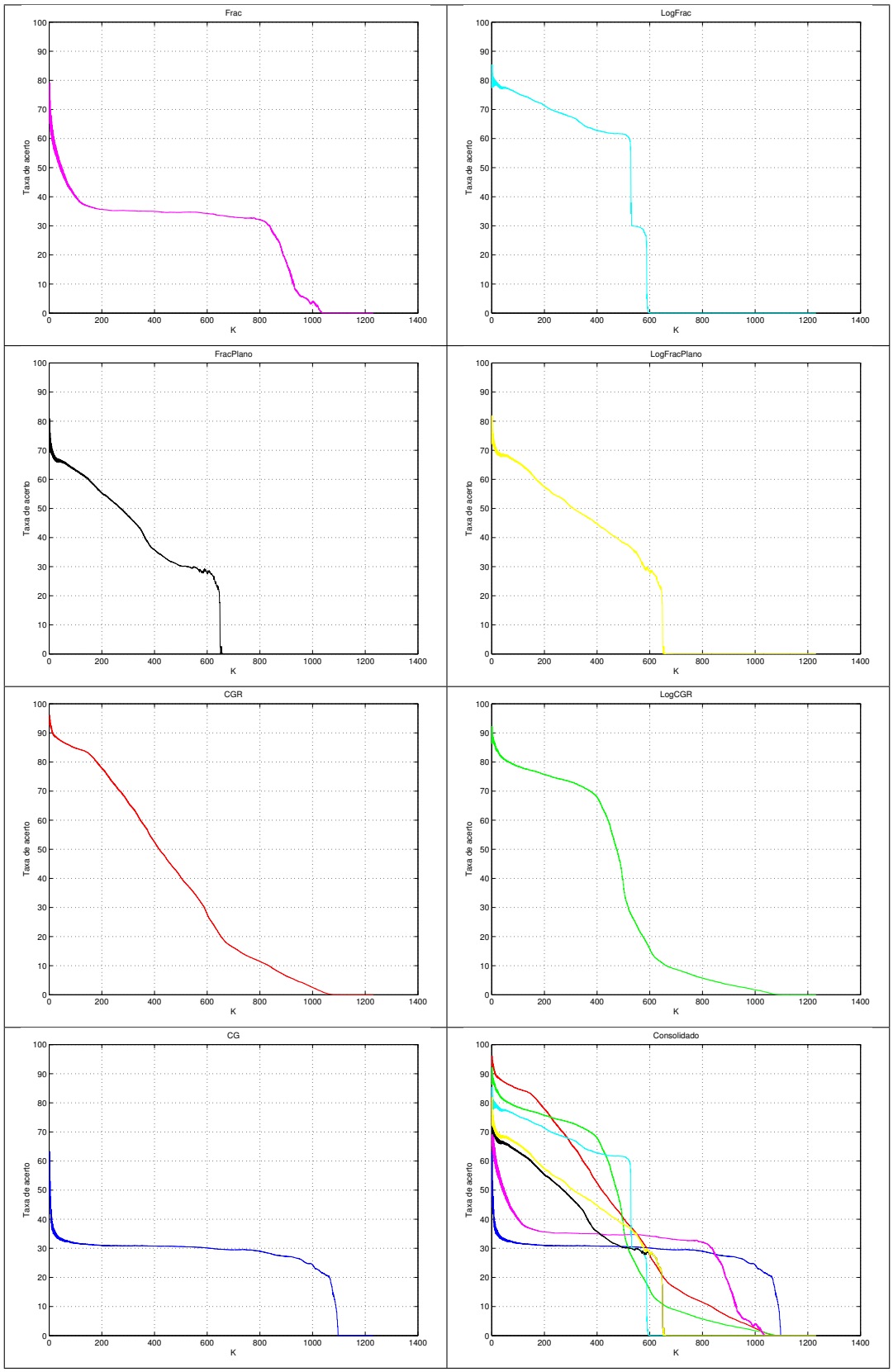


Figura 8.15: Resumo dos resultados obtidos para cobertura do genoma humano, utilizando a classificação pelos k-vizinhos, janela de tamanho 7 e profundidade 128

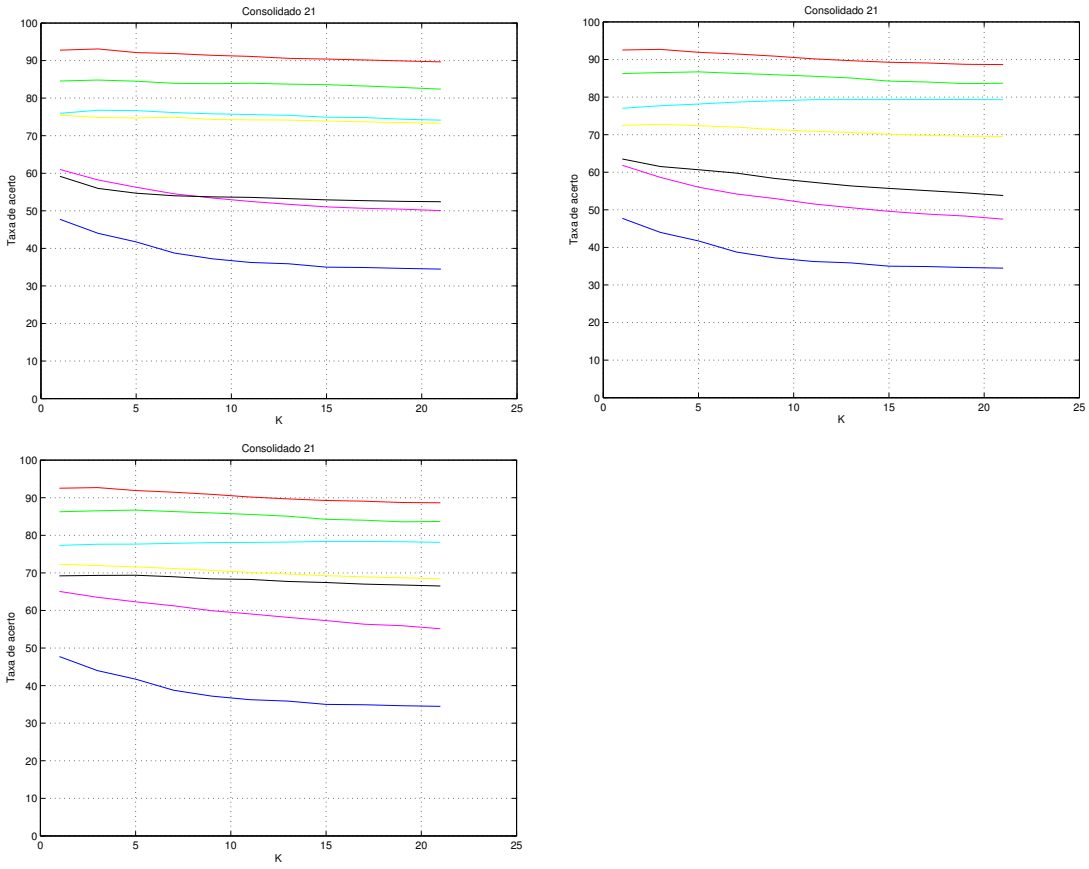


Figura 8.16: Resumo dos resultados obtidos para cobertura do genoma humano utilizando a classificação pelos k-vizinhos, diferentes janelas e resoluções ($6-64$, $7-64$ e $7-128$)

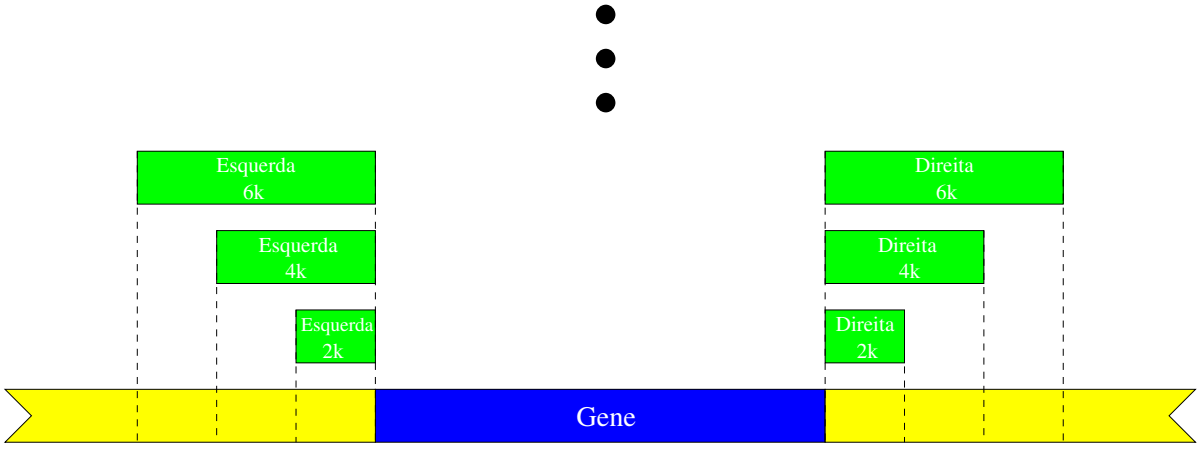


Figura 8.17: Três formas de escolher regiões próximas a um gene

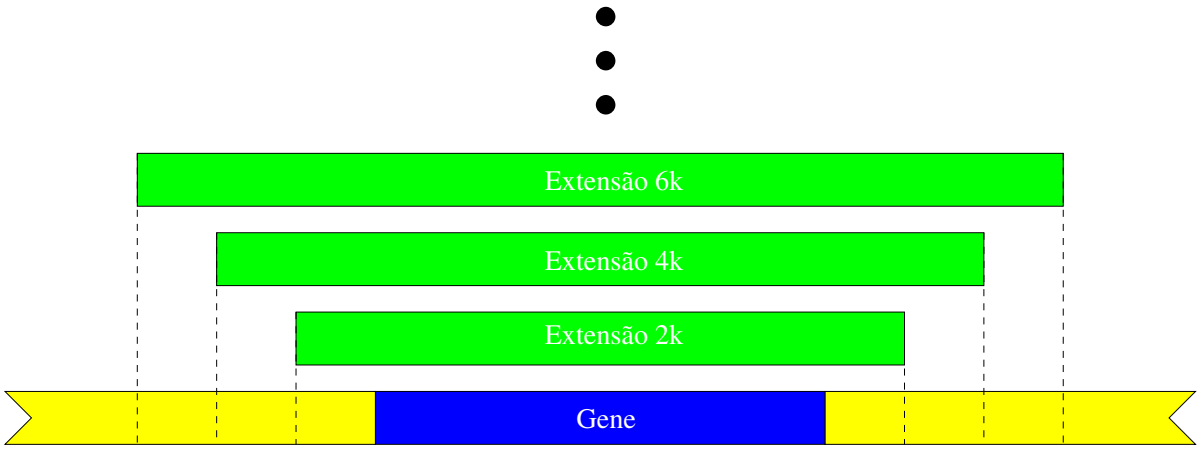


Figura 8.18: Exemplos de regiões estendidas de genes

- Utilizando como exemplos de teste regiões do gene estendida

  Nesse caso, os exemplos são formados pela concatenação de pedaços imediatamente laterais aos genes variando de 2.000 a 30.000 pares de bases como mostrado na Figura 8.18.

  O objetivo deste método é testar a precisão das características como um todo.

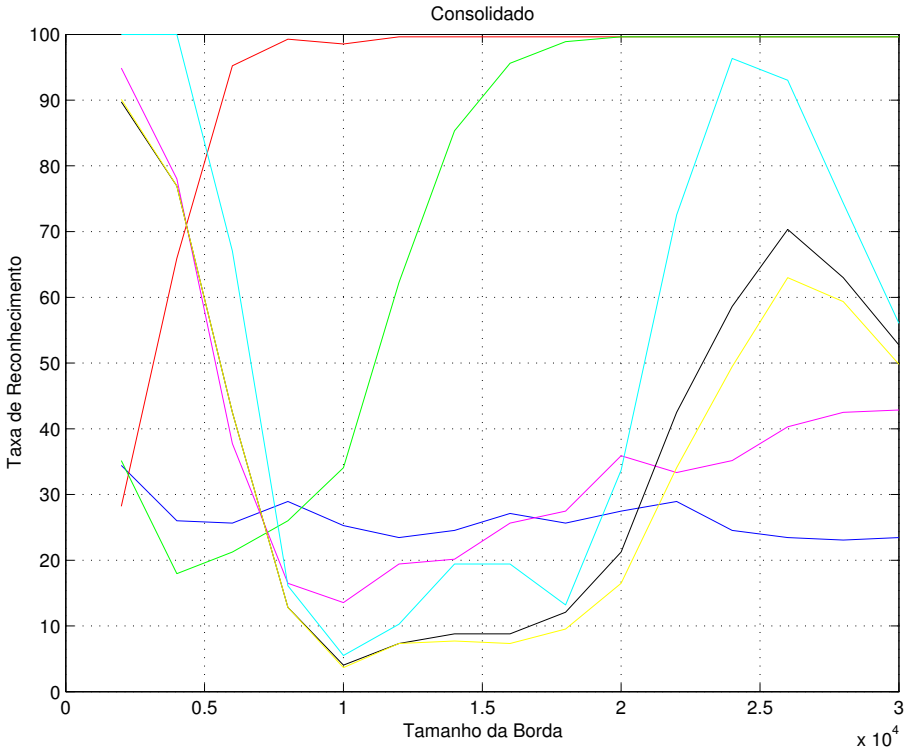
A taxa de acerto deve diminuir conforme a região considerada aumenta de tamanho, se isso acontece, é porque o método pode ter uma alta precisão e capacidade de determinar bordas, se não acontece, provavelmente a precisão do método é baixa. Na Figura 8.19 temos os resultados do experimento para o método de bordas e na Figura 8.20 os resultados do experimento para o método da concatenação. Os exemplos positivos são os do cromossomo 22 e os genes de testes são extraídos do cromossomo 21.

8.3.4 Análise dos Resultados

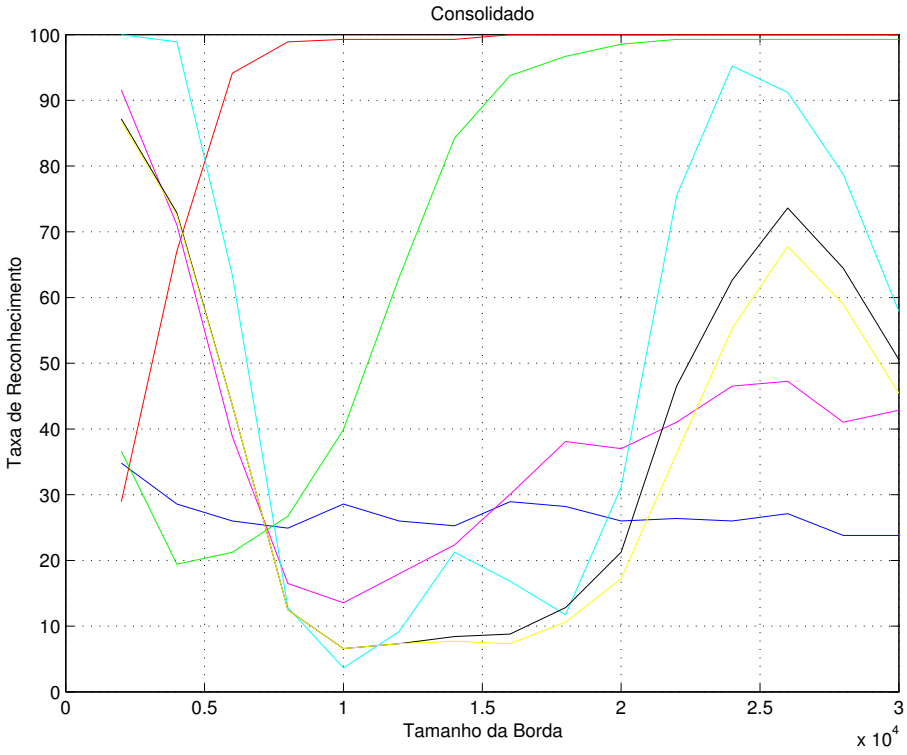
Como pode ser observado nos diversos gráficos presentes nas seções anteriores, a cobertura para as medidas baseadas simplesmente em **CGR**, tanto normalizada por log ou linearmente, é muito boa, mas a precisão é péssima, ou seja, aparentemente não vale a pena utilizar tais medidas para gerar uma curva de similaridade como proposto na Seção 8.1.

Em contrapartida, as medidas baseadas em **CGR** e que utilizam a técnica de análise de fractalidade descrita na Seção 7.3 mostram ser muito superiores para a busca de genes. Apesar da cobertura estar em torno de 80%, a precisão parece ser muito melhor, mas não o suficiente para que valha a pena gerar uma curva de similaridade.

Os resultados mostrados na Figura 8.19 e 8.20 são bastante interessantes. Na faixa entre 8.000 e 20.000 (ou 4.000 e 10.000 para o caso da extensão) pares de base, a medida de dimensão fractal multi-escala da imagem **CGR** normalizada por log mostra o seu melhor potencial em reconhecer corretamente regiões de não-gene. Lembrando que numa extensão de tamanho x se estende o gene x bases para a direita e x bases para a esquerda, as faixas



Bordas esquerdas



Bordas Direitas

Figura 8.19: Resultado dos experimentos de precisão para o método de bordas, utilizando k-vizinhos, janela 7 e profundidade 128

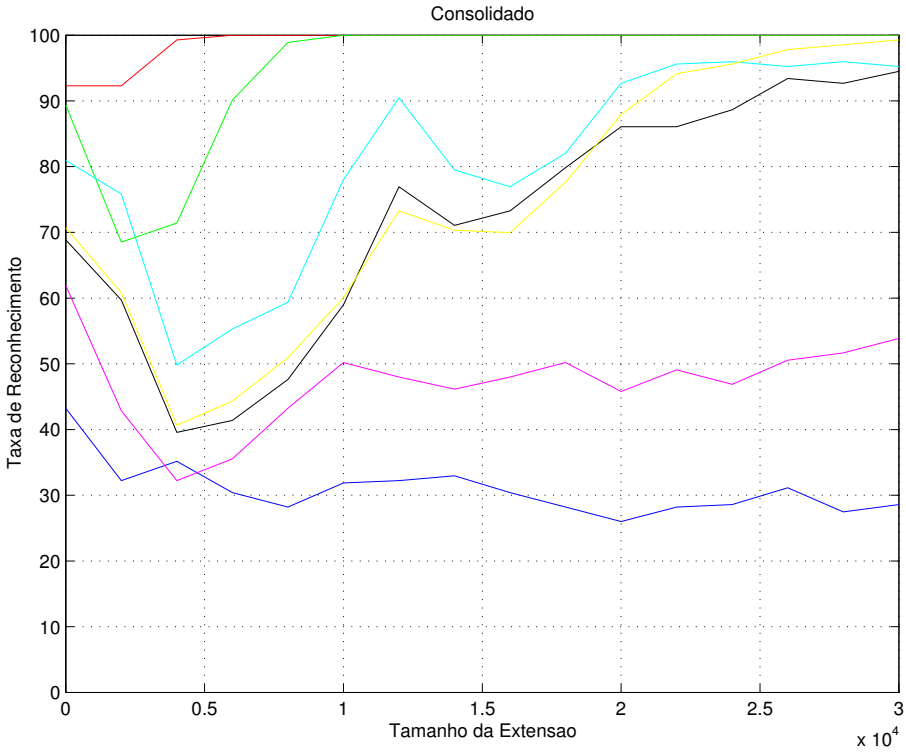


Figura 8.20: Resultado dos experimentos de precisão para o método de concatenação, utilizando k-vizinhos, janela 7 e profundidade 128

(do ponto de vista do tamanho do fragmento) em que a taxa de reconhecimento cai são equivalentes para todos os três gráficos. Observe na Figura 8.19 que, para as partes adjacentes a genes de tamanho final em torno de 10.000 – 20.000 pares de base, somente 10 – 20% dessas regiões são classificadas como genes, enquanto a Figura 8.20 mostra que regiões de extensão de gene de mesmo tamanho (região entre 4.000 e 10.000 pares de bases) são reconhecidas em $40-60\%$ dos casos. Isso sugere que quando há um gene numa determinada região, mesmo que estendida levemente, o resultado da classificação tende a indicar que há um gene na região, enquanto se não há um gene, a classificação tende a classificar também a região corretamente.

De qualquer forma, não se verifica uma configuração em que há alta cobertura e alta precisão. A precisão só aumenta quando a cobertura diminui, o que sugere que, apesar das medidas baseadas em técnicas fractais serem melhores que as outras, não são suficientemente boas para a busca de genes, mas mais testes são necessários para que se verifique tal conjectura.

Capítulo 9

Conclusão

Ao longo deste trabalho foram propostas novas formas de extração de características de **DNA** com o objetivo principal de utilizá-las na busca de genes. Avaliamos diversos aspectos dessas características, como aspectos teóricos, o tamanho da janela (resolução), a profundidade e o tamanho da região que deve compor a janela.

Foi proposto também um modelo para a utilização de tais características e uma forma de projetar classificadores baseados em k-vizinhos.

A análise apresentada na Seção 6.5 revelou que tanto multi-resolução quanto multi-escala, 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, dada uma seqüência, o que é "mais detalhe" e o que é "menos detalhe".

Após uma extensa série de experimentos, conclui-se que as características baseadas em medidas fractais são significativamente melhores que as outras avaliadas, mas, mesmo assim, aparentemente não são suficientemente boas para detecção de genes.

Importante observar que técnicas que utilizam cadeias de Markov ou mesmo Redes Neurais para a representação da distribuição de bases num éxon são, de certa forma, equivalentes à análise das imagens **CGR** e que tais técnicas são razoavelmente boas. Infelizmente as técnicas de análise fractal não podem ser aplicadas nesse caso porque as regiões de éxons são, em geral, muito pequenas.

O trabalho de extração e avaliação de características que realizamos é, também, uma forma de restrição do espaço de hipóteses, como apresentado no Capítulo 4. Tal análise revela a importância de métodos que levam em conta informação a respeito do problema como é o caso dos métodos apresentados no Capítulo 5.

Tentar descobrir genes utilizando somente uma característica que não explora aspectos locais e informação biológica é meta ambiciosa e talvez até impossível. Apesar dos resultados obtidos não serem muito animadores, sugerem que tais características podem ser utilizadas para abordar problemas com características mais globais de forma mais eficiente que o conteúdo **GC**, como é o caso de comparações entre genomas para a busca de regiões preservadas ou transmitidas horizontalmente.

De qualquer forma, somente com a realização de alguns itens propostos na Seção 9.2 será possível verificar se as novas medidas propostas são boas ou não para a busca de genes.

Uma das maiores limitações que enfrentamos ao longo deste trabalho é a computacional. Apesar do nosso parque instalado ter uma capacidade bem razoável de processamento (como

mostrado na Seção 9.4), acreditamos que seria necessário multiplicar essa capacidade por 10 para viabilizar a realização de experimentos que eventualmente permitam uma conclusão mais forte a respeito das características propostas. Importante observar que tal necessidade sugere que o problema deva ser reavaliado, como argumentado na Seção 4.1.

9.1 Principais Contribuições

Dentre as contribuições deste trabalho, podemos destacar:

- Algoritmo e implementação eficientes para cálculo de dimensão fractal multi-escala

  Desenvolvemos um algoritmo eficiente para o cálculo de dimensão fractal multi-escala baseado na transformada tridimensional da distância.

- Algoritmos e implementação eficientes para o cálculo de imagens **CGR** e de gramáticas estocásticas

  Além do algoritmo clássico para o cálculo de imagens **CGR**, também foi desenvolvida uma versão eficiente para o cálculo de árvores de sufixo, úteis para o cálculo de imagens **CGR** de resolução muito grande.

- 6 novas características

  Desenvolvemos 6 novas características que podem ser extraídas a partir de seqüências de **DNA**

- Modelo para busca de gene

  Propomos um modelo para a busca de gene que eventualmente pode ser útil na análise dos genomas atuais.

Apresentamos também um pôster no *First International Conference on Bioinformatics and Computational Biology* – **ICoBiCoBi**. O resumo e o pôster estão disponíveis no Apêndice B.

9.2 Trabalhos Futuros

Ao longo deste trabalho, diversas variações e idéias para continuações surgiram. Dentre as diversas opções para trabalhos futuros, podemos destacar:

- Experimentos com redução de dimensionalidade

  Para o caso da curva de fractalidade multi-escala, levamos em conta a curva inteira para o cálculo da distância entre duas curvas. Pode ser que somente algumas poucas partes das curvas sejam necessárias para uma boa classificação. Tais partes poderiam ser reveladas utilizando técnicas de redução de dimensionalidade.

- Extração de novas características

  É possível extrair algumas características das curvas de fractalidade multi-escala, dentre elas, a extensão da curva que está acima de um determinado limiar, ou o comportamento da curva ao redor do ponto máximo. Tais características podem ser muito úteis para a classificação.

- Cálculo da curva de classificação para todo o genoma

  Para determinar a precisão da técnica e testar na bancada os eventuais resultados, é necessário calcular a curva de classificação proposta na Seção 8.1 para todo o genoma.

- Comparação com regiões aleatórias do genoma

  Uma outra forma de verificar a precisão das técnicas avalidas é tomando como exemplo negativo de teste regiões do genoma escolhidas de forma aleatória. Como se acredita que o espaço ocupado pelos genes em relação ao tamanho do genoma é pequeno, então os conjuntos gerados dessa forma podem ser bons candidatos a exemplos negativos.

- Fatiamento das regiões do genoma

  Uma questão muito importante para a realização do modelo para a busca de genes descrito na Seção 8.1 é a forma com a qual se deve fatiar o genoma para análise local e como os dados da análise local devem ser consolidados para a análise global.

  Teriam que ser investigadas diversas alternativas de fatiamento, o tamanho ideal da janela, ou até mesmo uma maneira multi-escala de analisar o genoma.

- Granulometria

  Outras medidas para a análise de imagens poderiam ser utilizadas em vez da análise fractal multi-escala que estamos utilizando. Entre elas, podemos destacar a granulometria, que é uma medida muito usada na área de processamento de imagens.

- Combinação de características

  Algumas características poderiam ser utilizadas em conjunto, e não separadamente, como é feito neste trabalho. Pode ser que algumas combinações dêem resultados muito melhores que as características sozinhas.

Apesar de simples, as sugestões aqui apresentadas não foram realizadas por questões de tempo. Acreditamos que tais sugestões poderiam ser implementadas em cerca de 1 mês, mas estimamos que os resultados dos experimentos só estariam disponíveis em pelo menos 1 ou 2 anos, porque são extremamente custosos do ponto de vista computacional.

Uma outra aplicação interessante para as medidas descritas neste trabalho é a comparação de genomas. As medidas de fractalidade avaliadas aparentemente são muito boas para avaliar textura, funcionando como redutor de dimensionalidade, o que poderia facilitar a busca de regiões preservadas ou mesmo de transferências horizontais entre espécies.

9.3 Considerações Finais

Técnicas que permitem a leitura de genomas e daquilo que se expressa em uma célula abriram um horizonte completamente novo para a humanidade. Apesar de ainda muito primitivas, tais técnicas permitem que se compreendam eventos antes nem imagináveis.

Acredito que, nas próximas décadas, a análise de expressão gênica será o grande problema da área. Uma pequena melhora tanto nos métodos computacionais como biológicos levarão a um nível de compreensão nunca antes visto, mas tal pequena evolução deve demorar a acontecer, porque depende demais da solução de problemas muito complexos. Acredito que somente a partir de algumas décadas vamos ver resultados concretos sobre este tema.

O problema de busca de genes é muito interessante, tanto que foi escolhido como tema para os nossos experimentos. Acredito que só será definitivamente resolvido quando for possível manipular o **DNA** em três dimensões. Apesar do problema ganhar mais uma dimensão, deve ocorrer fenômeno parecido com o associado a atratores estranhos, que são razoavelmente simples apesar de modelarem problemas de dimensionalidade alta. Métodos para tal ainda não são sequer imaginados.

O que me deixa muito decepcionado com a área em si é a utilização de muitos métodos sem a mínima compreensão do que fazem ou para que servem. A importância da modelagem é muitas vezes esquecida, o que impede que muitas descobertas sejam feitas, como é mostrado na Figura 4.2, na página 24. Como já foi dito, grande parte dos problemas enfrentados atualmente na área com certeza não serão resolvidos com muita CPU, mas sim, com muita modelagem e idéias simples, mas inteligentes.

Ainda há muitas coisas a serem descobertas na área sem a necessidade de geração de novos dados. As ferramentas de análise atuais não estão preparadas para lidar com a dinâmica dos dados atualmente gerados. Acredito que tal problema também vai ter solução encaminhada nas próximas décadas, mas vai ser bem mais difícil de abordar do que expressão gênica, por exemplo.

A área de Teoria do Caos revelou que padrões simples podem resultar em formas complexas, e que estudos quantitativos não são adequados para abordar tais problemas. Acredito que somente idéias qualitativas vão gerar nova revolução na área, e o primeiro passo é criar ferramentas para uma visualização eficiente, por humanos, da quantidade fenomenal de dados gerados, dessa forma, será possível utilizar o cérebro em vez de CPU de forma mais útil e eficiente.

Diferentemente do que muitos acreditam, eu prefiro acreditar que o **DNA** não é a "última fronteira", mas sim um minúsculo passo em direção ao auto-conhecimento. É hipocrisia acreditar que a única forma de transmitir informação e evoluir está no **DNA**.

9.4 Aspectos Técnicos

Esta dissertação foi escrita utilizando somente ferramentas *livres*, dentre elas, o sistema operacional *Debian GNU/Linux 3.0*, *Emacs 21.3+1*, LaTeX *2.0.2*, *Gs 7.07*, *Xfig 3.2.4.d-rel-9*, *Dia 0.91*, *ImageMagick 5.5.7.9* entre muitos outros.

Os arquivos fonte desta dissertação totalizam aproximadamente $317k$ em 10.280 linhas e as imagens no formato *pdf* ocupam em torno de $5.7M$. O arquivo *pdf* final gerado tem 164 páginas e ocupa aproximadamente $6.32M$.

Os experimentos foram realizados em micros 32 bits *AMD Athlon* com freqüências entre $1.6Ghz$ e $2.1Ghz$. A quantidade de memória de cada micro variava entre $512M$ e $2G$. Foram utilizados 5 micros duais e um micro *single* nessa configuração.

São ao todo 6.099 linhas de código na linguagem **C** e 7.871 linhas de código na linguagem **Matlab**, sem contar scripts em **Perl** e **Bash**, totalizando aproximadamente $605K$ de código fonte.

Apêndice A

Especificação e Implementações do Ambiente de Testes

Acreditamos que o ponto mais forte da nossa especificação é a forma com que os dados são armazenados e as relações entre os mesmos. Ou seja, especificamos 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 estendê-lo de forma simples. Além disso, o modelo deve ser maduro 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.

Escolhemos o problema clássico de localização de genes devido à sofisticação que tal problema alcança na área. Na tentativa de resolvê-lo, 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.

Ao longo deste projeto realizamos a implementação de diversos algoritmos, dentre eles, implementamos o algoritmo descrito por Deschavanne *et al.* [30] para o cálculo de imagens **CGR**. 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 implementamos uma versão multi-resolução e multi-escala do **CGR** como descrito na Seção 6.5.2.

Também foi realizada uma implementação incrementada de árvores de sufixo que economiza memória proposta por Kurtz [78]. Essa implementação é útil para o cálculo de tabelas de freqüência de palavras grandes sem gastar muita memória, mas acabamos não utilizando tal possibilidade.

Ao longo do trabalho, além dos programas para cálculo de imagens **CGR** e cálculo eficiente de dimensão fractal multi-escala, realizamos três especificações e implementações principais para realização dos experimentos que serão descritas a seguir.

A.1 Gerador estocástico de palavras

É um programa simples, em que dados uma especificação de gramática e alguns parâmetros ele gera palavras que pertencem à gramática. A gramática pode ser de qualquer tipo.

Durante sua especificação, levamos em conta os padrões de representação de gramáticas estocásticas disponíveis, bem como as necessidades do nosso grupo, visto que tal gerador é utilizado em testes dos algoritmos que estão sendo desenvolvidos pelo nosso grupo e por este projeto. O programa 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.

A.2 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, entre eles os programas do **GCG** [152] e **bioperl** [157], mesmo que rodem em sistemas operacionais distintos. Além disso, os dados estão rapidamente disponíveis, visto que estão no HD do micro de testes.

Esse ambiente foi dividido em quatro módulos principais.

A.2.1 Módulo 1 – Dados de entrada

É o módulo mais importante. Como base desse 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.

Esse 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 incorporada, bastando fazer download dos novos dados e rodar o utilitário *make*, ações que podem até mesmo ser colocadas num script automático. Resumindo esse módulo foi projetado para ser de fácil atualização, modificação e extensão.

Na prática, no entanto, os formatos dos arquivos de seqüência de **DNA** em alguns bancos [10, 43, 42, 132, 153, 6] mudam, o que dificulta um pouco.

A.2.2 Módulo 2 – Dados de teste

Nesse 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.

A.2.3 Módulo 3 – Programas

Nesse 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.

A.2.4 Módulo 4 – Relatórios

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

A.3 Dados de Testes

Como estamos abordando o problema de predição de genes, utilizamos dados do **NCBI** [163], **Ensembl** [161], Genome Project Working Draft [162] 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 sua completude) 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 itens.

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

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

Uma informação curiosa a respeito dessa implementação é que foi uma das partes mais difíceis do projeto, 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 obter-se pelos *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 é visitado, 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 por meio 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, por meio de uma excelente interface escrita na linguagem **perl**, facilita a obtenção de diversas informações que precisamos para o projeto.

Apêndice B

Participação no ICOBICOBI

Participamos do "1st International Conference on Bioinformatics and Computational Biology", em Riberão Preto, em que subtemos um abstract e apresentamos um pôster, ambos podem ser visualizados nas próximas páginas.

Os arquivos originais tanto do pôster como do resumo podem ser acessados na página do projeto.

Classification of genomic regions by chaos game representation images and fractal dimension

Caetano Jimenez Carezzato
DCC-IME-USP, University of São Paulo – Brazil – caetano@vision.ime.usp.br
Junior Barrera
DCC-IME-USP, University of São Paulo – Brazil – jb@ime.usp.br
Sandro José de Souza
ILPC – Brazil – sandro@compbio.ludwig.org.br
Luciano da Fontoura Costa
IFSC-USP, University of São Paulo – Brazil – luciano@if.sc.usp.br

Abstract

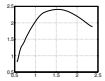
This work describes a new approach to classify genomic regions by applying the multiscale fractal dimension [1] over images generated by chaos game representation (CGR) of sequences. Since the introduction of chaos game representation of sequences [4], it has been found evidences that it is possible to obtain discriminative measures from images produced by this methodology and to feed such features into classifiers in order to identify the origin of gene fragments. Also, it has been showed [2] that it is possible to reconstruct filogenetic trees just using this representation.

In this project, we propose a new feature extractor of sequences that can help the classification of genomic regions. The feature extraction consists of determining the CGR of a sequence and estimating the fractal dimension of the generated image. Such measurements are organized into a feature vector.

For certain genomic regions, as the GC-content (mean percentage of guanine and cytosine) changes, the CGR also changes. We are going to verify if CGR contains more meaningful information than GC-content that can be easily and fastly exploited for genomic regions classification. Our preliminary results based on cluster techniques show that methods based on this approach should be better than GC-content based ones. We compared, for each approach, the ability to distinguish genic regions of the human chromossome 22. Currently, we are finishing the validation of the tests and developing a way to classify genic regions.


DNA sequence


CGR image


fractal dimension

References

[1] Costa, L. da F., Manoel, E.T.M., Faucereau F., Chelly J., van Pelt J., and Ramakers G. A shape analysis framework for neuromorphometry. Network, 13, 283-310, 2002.

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

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

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

Classification of Genomic Regions by Chaos Game Representation Images and Fractal Dimension

Caetano Jimenez Carezzato[1], Junior Barrera[1], Sandro José de Souza[2] and Luciano da Fontoura Costa[3]

[1]DCC-IME-USP, University of São Paulo – Brazil – {caetano, jb}@vision.ime.usp.br

[2]ILPC – Brazil – sandro@compbio.ludwig.org.br, [3]IFSC-USP, University of São Paulo – Brazil – luciano@if.sc.usp.br

Abstract

This work describes a new approach to classify genomic regions by applying the multiscale fractal dimension [2] over images generated by chaos game representation (CGR) of sequences. Since the introduction of chaos game representation of sequences [8], evidences have been found that it is possible to obtain discriminative measures from images produced by this methodology and to feed such features into classifiers in order to identify the origin of gene fragments. Also, it has been showed [3] that it is possible to reconstruct filogenetic trees just using this representation.

In this project, we propose a new feature extractor of sequences that can help the classification of genomic regions. The feature extraction consists of determining the CGR of a sequence and estimating the fractal dimension of the generated image. Such measurements are organized into a feature vector.

For certain genomic regions, as the GC-content (mean percentage of guanine and cytosine) changes, the CGR also changes. We are going to verify if CGR contains more meaningful information than GC-content that can be easily and fastly exploited for genomic regions classification. Our preliminary results based on cluster techniques show that methods based on this approach should be better than GC-content based ones. We compared, for each approach, the ability to distinguish genic regions of the human chromosome 22. Currently, we are finishing the validation of the tests and developing a way to classify genic regions.

CGR Images

The chaos game is an algorithm, generally controlled by a series of random numbers, which allows one to produce images (attractors) of fractal structures. The use of DNA sequences, rather than random numbers, has been proposed to control the chaos game [8]. We call an image generated by the DNA sequence of Chaos Game Representation (**CGR**) of the DNA sequence.

CGR consists of disposing a table of frequencies in one square matrix using a recursive rule.

Given one window w of size s, we build the square matrix m of size $n \times n$, $n = 2^s$. Each point of the matrix m represents the frequency of one possible configuration of the window w.

In the case where $s = 1$, there are only 4 rows in the frequency table, the alphabet itself. We dispose the following example table

A	10
T	9
C	4
G	13

in the following 2×2 matrix:

C	G
A	T

$\Longrightarrow$

4	13
10	9

In the case where $s = 2$, there are 16 rows in the frequency table as shown in the following example:

AA	10	CA	35
AT	9	CT	44
AC	4	CC	0
AG	13	CG	1
TA	11	GA	22
TT	15	GT	7
TC	24	GC	90
TG	17	GG	19

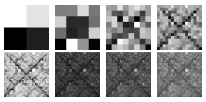
C	G
A	T

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

$\Downarrow$

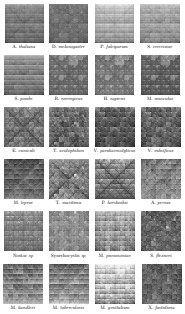
0	1	90	19
35	44	22	7
4	13	24	17
10	9	11	15

We can use gray levels, instead of using numbers, for visualization. In the following example, we have the CGR images for $s = 1 \ldots 8$ of the bacteria A. *fulgidus*. The dark intensity is direct proportional to the frequency. The last three images are log normalized.

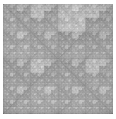


In the next set of imagens we can see the CGR image log normalized, $s = 8$ for some organisms.

Note that there are significative pattern differences between each organism.

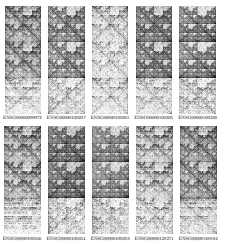


Deschavanne [3] showed that these patterns can be used to build filogenetic trees. Is is also possible to reconstruct the CGR image using just small pieces of the genome. This property can be used to construct a classifier that can decide from which organism is a given piece of DNA [9].



The image above shows the log normalized CGR image of the chromosome 22 where $s = 8$. In the following set of images, we show some CGR images of genes of the chromosome 22. For each image, there are three square sub-images inside. The top one is the CGR of the gene DNA sequence, the medium one is the CGR of the introns sequences and the bottom one is the CGR of the exons sequences.

Note that some genes look like each other and with the CGR image of the chromosome 22 above. Also note that some gene CGR images are very different from each other, specially the exon region images.



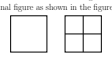
We are trying to explore local variations of the CGR image in the chromosome. Our main goal is to extract some useful information from CGR images that can be explored locally. We are trying to do this using fractal dimension techniques.

Fractal Dimension

To explain the concept of fractal dimension, it is necessary to understand what we mean by dimension. Obviously, a line has dimension 1, a plane dimension 2, and a cube dimension 3.

So why is a line one-dimensional and the plane two-dimensional? Note that both of these objects are self-similar. We may break a line segment into 4 self-similar intervals, each with the same length, and each of which can be magnified by a factor of 4 to yield the original segment. We can also break a line segment into 7 self-similar pieces, each with magnification factor 7, or 20 self-similar pieces with magnification factor 20. In general, we can break a line segment into n self-similar pieces, each with magnification factor n.

A square is different. We can decompose a square into 4 self-similar sub-squares, and the magnification factor here is 2. Alternatively, we can break the square into 9 self-similar pieces with magnification factor 3, or 25 self-similar pieces with magnification factor 5. Clearly, the square may be broken into n^2 self-similar copies of itself, each of which must be magnified by a factor of n to yield the original figure as shown in the figure bellow.



Finally, we can decompose a cube into n^3 self-similar pieces, each of which has magnification factor n.

Fractal dimension is a measure of how "complicated" a self-similar figure is. In a rough sense, it measures "how many points" lie in a given set. A plane is "larger" than a line, while other curves sit somewhere in between these two sets. The fractal dimension provides a quantitative characterization of the complexity of curves as induced by self-similarity.

On the other hand, all three of these sets have the same number of points in the sense that each set is uncountable. Somehow, though, fractal dimension captures the notion of "how large a set is" quite nicely. Therefore, the fractal dimension provides a nice indication of how much the curve extends itself through space. As a consequence, more intricate curves will cover the surrounding space more effectively, leading to higher fractal dimensions.

Now we see an alternative way to specify the dimension of a self-similar object: the dimension is simply the exponent of the number of self-similar pieces with magnification factor n into which the figure may be broken. So we can write

$$F = \text{fractal dimension} = \frac{\log(\text{number of self-similar pieces})}{\log(\text{magnification factor})}$$

So, for a straight line, $F = \frac{\log(n)}{\log(n)} = 1$, for a plane, $F = \frac{\log(n^2)}{\log(n)} = 2$ and for a cube, $F = \frac{\log(n^3)}{\log(n)} = 3$.

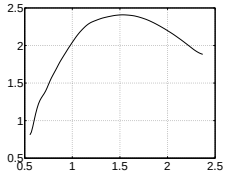
For the classical Koch Triadic curve, which basic pattern is bellow, the fractal dimension $F = \frac{\log(4)}{\log(3)} \approx 1.26$.

"Despite the good potential of this measure to characterize complexity in a more objective fashion, its extension to real objects is complicated by the fact that the latter are not perfectly self-similar. In fact, only a few orders of similarity are usually found for natural objects, such as the three or four orders found in fern leaves. Indeed, the fractality of such objects, especially when represented in digital images, is limited at both microscopic and macroscopic scales. First, for scales smaller than the image resolution, the fractal dimension tends to zero, the dimension of the image pixels. On the other hand, for scales larger th an the object, the respective dimension tends to zero, as the object tends to behave as a point for large distances. Therefore, real objects will present higher fractal values only along limited intervals of spatial scale. This problem can be suitably addressed by using the multiscale extension of the fractal dimension recently described in [1], which involves the numerical estimation of the first derivative of a log log cumulative function, more specifically the graph of the logarithm of the dilated area in terms of the logarithm of the spatial scale (i.e. radius of the dilating discs).

This extension involves obtaining not a scalar value of fractal dimension as usually done, but expressing a fractal function in terms of the spatial scale that properly reflects the behavior of the object when observed at different magnifications. Therefore, the multiscale fractal dimension represents a less degenerate geometric characterization, in the sense of preserving more information about the geometry of the original object." [2]. We are using this approach to extract information from DNA sequences.

More information on fractal dimension and chaos game can be found in Devaney [4] and Costa [2].

The following image is an example of multiscale fractal dimension graph obtained from the CGR image of the human chromosome 22. The x-axis is the resolution (log of the radius of the ball size in pixels) and the y-axis is the fractal dimension.



Preliminary Results

We did a series of 25 clustering experiments to evaluate the potential of the fractal dimension based approach to searching genes. We compared 5 kinds of characteristics:

1. GC-content
2. CGR image of the sequence
3. CGR image log normalized of the sequence
4. Multiscale Fractal Dimension of CGR image
5. Multiscale Fractal Dimension of CGR image log normalized

We extracted the above characteristics from 526 genes of the chromosome 22 (complete sequence) and 655 continuous regions of the same chromosome where there is a big confidence that there is no genes or genes pieces in them.

The basic idea was to try to separate these 1181 DNA sequences just using one of the above characteristics each time. For each characteristic we calculated a matrix of distance D using, respectively, the following rules where x and y are the two points being considered:

1. GC-Content - Sum of the module of the differences of x and y
2. CGR image - $D = \sum_{i,j} |x_{ij} - y_{ij}|$
3. Multiscale Fractal Dimension - $D = \sum_{i} |x_i - y_i|$

For each distance matrix D, we applied 5 hierarchical clustering algorithms available in Matlab 6 trying to separate the whole set.

For each one of the 25 experiments we calculated 1180 clustering of the same method with the number of clusters from 2 to n.

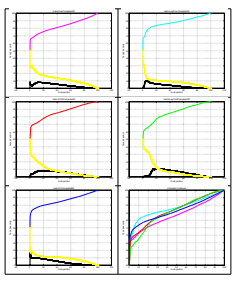
The "error rate" of each experiment is the minimum error possible that we do by labelling each cluster with either "genes" or "not genes". So, for each clustering, the total error will be the sum, for each cluster, of the minimum number between genes and not genes elements in the cluster.

We applied a simple bootstrap method that consists of selecting randomly 350 regions of genes and 350 regions of not genes. We build 100 data sets using this method and applied the clustering procedure for each data set.

In the following set of graphs we can see the results for the "Complete" clustering method. The x-axis represents the number of clusters (in percentage of the total set size) and the y-axis the error rate. Each graph represents a different characteristic.

For the first 5 graphs, in black we have the mean of false positive rates (FP - labelling as gene regions that are not genes) with error bars and in yellow the mean of false negative rates (FN - labelling as not gene regions that are genes) with error bars. In the third color we have the mean of hit rates (100 - mean(FP) - mean(FN)). The last graph shows the mean of hit rates for each characteristic.

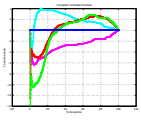
- Magenta - Multiscale Fractal Dimension graph of CGR image
- Cyan - Multiscale Fractal Dimension graph of CGR image log normalized
- Red - CGR image
- Green - CGR image log normalized
- Blue - CG-Content



It is important to observe that for the others clustering methods the results are almost similar. In all experiments the cyan curve (Multiscale Fractal Dimension graph of CGR image log normalized) is better than the others curves. Note that it only make sense to compare the curves in the first 20% part (the beginning of the graph) because values higher than that characterizes overfitting.

Theses preliminary results are interesting because they don't use any additional biological information. In spite of the separation for all methods is not excellent, these results suggests that measures based on fractal dimension of CGR images of sequences should be used in gene searching systems. Note that this is a preliminary result that must be validated through more experiments. In the way they were done, these experiments are equivalent to training a supervised classifier and applying the classifier in the training set.

The following graph shows the mean hit rate for each characteristic being considered, with error bars.



Further Work in Progress

Currently we are doing a series of experiments to validate the application of the multiscale fractal dimension technique. We pretend to test the ability of classification of this measure using randomly selected regions of the human chromosome 22 and to extend these experiments to others chromosomes.

It is important to note that while the Multiscale Fractal Dimension graph of CGR image log normalized has produced nice results, the Multiscale Fractal Dimension graph of CGR image linearly normalized was the worst technique evaluated. This suggests that normalization plays an important role in the process, so we pretend to make experiments to find a nice normalization that should be used.

The biggest problem blocking the realization of these experiments was the algorithm to calculate the fractal dimension. It was very slow. Now, we have just finished a new nice and very faster algorithm so we plan to have final results in some weeks.

This approach needs the definition of a way to build a classifier given a training set. We also pretend to study this problem.

We also may adapt this approach to other problems in Computational Biology such as the determination of intron/exon and exon/intron frontiers.

Acknowledgment

This work is supported by grant 01/03975-5 to C.J.C. from FAPESP of the Brazilian Government. This work is linked to the thematic project **CAGE** http://www.vision.ime.usp.br/~cage/ FAPESP 99/07390-0.

References

[1] Costa, L. da F., Campos, A. G., and Manoel, E. T. M. An integrated approach to shape analysis: results and perspectives. In *Int. Conf. on Quality Control by Artificial Vision* (2001), pp. 23–34.

[2] Costa, L. da F., Manoel, E. T. M., Faucereau, F., Chelly, J., van Pelt, J., and Ramakers, G. A shape analysis framework for neuromorphometry. *Network 13* (2002), 283–310.

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

[4] Devaney, R. L. *Chaos in the Classroom*. Boston University, 1995. http://math.bu.edu/DYSYS/chaos-game/chaos-game.html.

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

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

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

[8] Oliver, J. L., Bernaola-Galván, P., Guerrero-Garcia, J., and Roman-Roldan, R. Entropic profile of DNA sequences through chaos-game-derived images. *Journal of Theoretical Biology 160* (1993), 457–470.

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

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

Referências Bibliográficas

[1] S. S. Adi. Ferramentas de Auxílio 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]. 1

[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.Fev.2001]. 1

[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]. 23, 45, 48

[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. 23, 45, 48

[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. 18

[6] 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. 122

[7] P. Baldi and S. Brunak. *Bioinformatics: The Machine Learning Approach (Adaptive Computation and Machine Learning)*. MIT press, 1998. 49

[8] J. Barrera, E. R. Dougherty, and N. S. Tomita. Automatic programming of binary morphological machines by design of statistically optimal operators in the context of computational learning theory. *Journal of Electronic Imaging*, 1997. 19

[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. 1

[10] A. D. Baxevanis. The molecular biology database collection: 2002 update. *Nucleic Acids Research*, 30(2):1–12, 2002. 122

[11] A. D. Baxevanis and B. F. Ouellette, editors. *Bioinformatics: A Pratical Guide to the Analysis of Genes and Proteins*. Wiley-Interscience, 1998. 49

[12] 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. 49

[13] 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]. 45

[14] M. Borodovsky and J. McIninch. GENMARK: parallel gene recognition for both DNA strands. *Comput. Chem.*, 17(2):123–133, 1993. 48

[15] 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, Abril 2000. http://www.cs.vu.nl/∼mbot/mijnpapers/euroGP2000/paper.ps. 14

[16] P. Bourke. The lorenz attractor in 3D. astronomy.swin.edu.au/∼pbourke/fractals/lorenz/. 56

[17] P. Bremaud, editor. *Markov Chains*. Springer Verlag, 1999. 30

[18] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997. 1

[19] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, 268(1):78–94, 1997. 48

[20] 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. 48

[21] F. Capra. *A Teia da Vida*. Cultrix, 7 edition, 1996. 53, 57

[22] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959. 41

[23] J. Claverie. Computational methods for the identification of genes in vertebrate genome sequences. *Hum. Mol. Genet.*, 6(10):1735–1744, 1997. 48

[24] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2nd edition, 1998. 41

[25] L. da F. Costa, A. G. Campos, and E. T. M. Manoel. An integrated approach to shape analysis: results and perspectives. In *Int. Conf. on Quality Control by Artificial Vision*, pages 23–34, 2001. 73

[26] L. da F. Costa, E. T. M. Manoel, F. Faucereau, J. Chelly, J. van Pelt, and G. Ramakers. A shape analysis framework for neuromorphometry. *Network*, 13:283–310, 2002. 84

[27] L. F. Costa and A. G. C. Blanchi. A outra da dimensão fractal. *Ciência Hoje*, 31(183):40–47, 2002. 73

[28] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000. 23

[29] R. E. Davis. Introduction to bioinformatics and genomics, 2002. http://www.library.csi.cuny.edu/∼davis/Bioinfo_326/ [29.Jun.2002]. 7

[30] 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. 64, 121

[31] R. L. Devaney. *Chaos in the Classroom*. Boston University, 1995. http://math.bu.edu/DYSYS/chaos-game/chaos-game.html. 53

[32] T. Devitt. Genetic moves on. http://whyfiles.org/075genome/. 6

[33] L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic Theory of Pattern Recognition*. Springer, 1996. 25

[34] S. Dong and D. Searls. Gene structure prediction by linguistic methods. *Genomics*, 23(3):540–551, 1994. 48

[35] 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. 1

[36] 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. 1

[37] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley and Sons, 2nd edition, 2000. 11, 17, 21, 27

[38] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of protein and nucleic acids*. Cambridge University Press, 2000. 23

[39] S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14(9), 1998. 38

[40] S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, 1974. 43

[41] 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]. 1

[42] D. A. B. et al. Genbank. *Nucleic Acids Research*, 30(2):17–20, 2002. 122

[43] D. L. W. et al. Database resources of the national center for biothecnology information: 2002 update. *Nucleic Acids Research*, 30(2):13–16, 2002. 122

[44] 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. 7

[45] D. Fasulo. An analysis of recent work on clustering algorithms, Abril 1999. http://www.cs.washington.edu/homes/dfasulo/clustering.ps [26.Jan.2001]. 13

[46] C. Fields and C. Soderlund. GM: a practical tool for automating DNA sequence analysis. *Comput. Appl. Biosci.*, 6(3):263–270, 1990. 48

[47] G. S. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer Series in Operations Research. Springer Verlag, 1996. 22, 23

[48] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1982. 25

[49] L. Garcia. Garcia ferns. http://www.macroinnovation.com/fern.htm. 78

[50] M. Gelfand, A. Mironov, and P. Pevzner. Gene recognition via spliced sequence alignment. *Proc. Natl. Acad. Sci. USA*, 93(17):9061–9066, 1996. 48

[51] M. Gelfand and M. Roytberg. Prediction of the exon-intron structure by a dynamic programming approach. *Biosystems*, 30:173–182, 1993. 48

[52] C. Gibas and P. Jambeck. *Desenvolvendo bioinformática: ferramentas de software para aplicações em biologia*. Campus – O'Reilly, 2001. Tradução de: Developing bioinformatics computer skills. 7

[53] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, 1992. 11

[54] G. R. Grant and W. J. Ewens. *Statistical Methods in Bioinformatics: An Introduction*. Springer Verlag, 2001. 23

[55] E. Green. Iterated function systems. www.cs.wisc.edu/∼ergreen/honors_thesis/IFS.html. 74

[56] A. J. F. Griffiths, W. M. Gelbart, J. H. Miller, and R. C. Lewontin. *Modern Genetic Analysis*. W. H. Freeman and Company, 1999. www.ncbi.nlm.nih.gov/books/bv.fcgi?call=bv.View..ShowTOC&rid=mga.TOC. 7

[57] R. Guigo, S. Knudsen, N. Drake, and T. Smith. Prediction of gene structure. *J. Mol. Biol.*, 226(1):141–157, 1992. 48

[58] D. Gusfield. *Algorithms on strings, trees, and sequences. Computer Science and Computational Biology*. Cambridge University Press, 1999. 5, 45, 48

[59] B. K. Hall, editor. *Homology: The Hierarchical Basis of Comparative Biology*. Academic Press, 1994. 1

[60] M. H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, 1995. 23, 25

[61] 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.Fev.2001]. 48

[62] J. Henderson, S. Salzberg, and K. Fasman. Finding genes in DNA with a hidden markov model. *J. Comput. Biol.*, 4(2):127–142, 1997. 48

[63] 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. 47

[64] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci USA*, 89:10915–10919, 1992. 47

[65] D. Higgins and P. Sharpe. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. In *Gene*, volume 73, pages 237–244, 1988. 1

[66] X. Huang, M. D. Adams, H. Zhou, and A. Kerlavage. A tool for analyzing and annotating genomic sequences. *Genomics*, 46(1):37–45, 1997. 48

[67] R. Hughey and A. Krogh. Hidden markov models for sequence analysis: extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996. 37, 39

[68] G. Hutchinson and M. Hayden. The prediction of exons through an analysis of spliceable open reading frames. *Nucleic Acids Res.*, 20(13):3453–3462, 1992. 48

[69] 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. 23

[70] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3), Setembro 1999. 13

[71] B. R. Jasny and D. Kennedy. The human genome. *Science*, 291(5507), Fevereiro 2001. http://www.sciencemag.org/genome2001/1153.html [16.Fev.2001]. 1

[72] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1997. 17

[73] 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]. 13

[74] S. Knudsen. Promoter2.0: for the recognition of polii promoter sequences. *Bioinformatics*, 15(5):356–361, 1999. 48

[75] T. Koski and T. Koskinen. *Hidden Markov Models for Bioinformatics*. Kluwer Academic Publishers, 2001. 30

[76] A. Krogh, I. Mian, and D. Haussler. A hidden markov model that finds genes in E. coli DNA. *Nucleic Acids Res*, 22(22):4768–4778, 1994. 48

[77] 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. 48, 50, 71

[78] S. Kurtz. Reducing the space requirement of suffix trees. *Software – Practice and Experience*, 29(13):1149–1171, 1999. 121

[79] 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. 43

[80] G. Lendvai. Genetic moves on. micro.utexas.edu/courses/levin/bio304/genetics/genetics.html. 6

[81] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 2nd edition, 1998. 40

[82] W. Li. A bibliography on computational gene recognition. http://www.nslij-genetics.org/gene/[13.Ago.2003]. 9

[83] D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, pages 1435–1441, 1985. 48

[84] R. A. Lotufo and F. A. Zampirolli. Fast multidimensional parallel euclidean distance transform based on mathematical morphology. In *XIV Brazilian Symposium on Computer Graphics and Image Processing*, IEEE Computer Society, pages 100 – 105, 2001. 85

[85] A. V. Lukashin and M. Borodovsky. Genemark.hmm: new solutions for gene finding. *Nucleic Acids Research*, 26(4):1107–1115, 1998. 48

[86] B. M. and R. Guigo. Evaluation of gene structure prediction programs. *Genomics*, 34(3):353–367, 1996. 48

[87] 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. 48

[88] A. H. McGowan. The genome – an introduction. http://www.geneinformation.org/lemonick-background.htm. 7

[89] J. Meidanis and J. C. Setubal. *Introduction to Computational Molecular Biology*. PWS Publishing Co., 1997. 1, 7, 47

[90] Y. S. Michael. The application of stochastic context-free grammars to folding, aligning and modeling homologous rna sequences, 1993. ftp.cse.ucsc.edu/pub/tr/ucsc-crl-94-14.ps.Z. 43

[91] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3rd rev. and extended edition, 1999. 14

[92] L. Milanesi, N. Kolchanov, I. Rogozin, I. Ischenko, A. Kel, Y. Orlov, M. Ponomarenko, and P. Vezzoni. Genviewer: a computing tool for protein-coding regions prediction in nucleotide sequences. *The Second International Conference on Bioinformatics, Supercomputing and Complex Genome Analysis*, pages 573–587, 1993. 48

[93] T. Minka. A statistical learning/pattern recognition glossary. http://www-white.media.mit.edu/∼tpminka/statlearn/glossary/ [20.Mai.2002]. 17

[94] A. A. Mironov, J. W. Fickett, and M. S. Gelfand. Frequent alternative splicing of human genes. *Genome Research*, 9:1288–1293, 1999. 48

[95] B. Morgenstern. Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999. 1

[96] 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. 1

[97] D. W. Mount. *Bioinformatics: Sequence and Genome Analysis*. CSHL Press, 2001. 7, 49

[98] M. Moura. O novo produto brasileiro. *Pesquisa FAPESP*, 55:8–15, julho 2000. http://www.fapesp.br/capa551.htm [4.Set.2000]. 1

[99] R. Mural. Current status of computational gene finding: a perspective. *Methods Enzymol.*, 303:77–83, 1999. 48

[100] D. Nicholls and C. Nicholls. Fractal ferns. home.aone.net.au/byzantium/ferns/fractal.html. 61

[101] 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. 25, 53, 58, 60

[102] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994. Reprinted August, 1995. 41

[103] C. H. Papadimitrou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982. 18, 22

[104] 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. 48

[105] W. R. Pearson. Rapid and sensitive sequence comparisions with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990. 48

[106] M. Pertea, X. Lin, and S. Salzberg. Genesplicer: a new computational method for splice site prediction. *Nucleic Acids Research*, 29(5):1185–1190, 2001. 48

[107] E. Pizzi and C. Frontali. Low-complexity regions in plasmodium falciparum proteins. *Genome Research*, 11:218–229, 2001. 48

[108] E. Pizzi, S. Liuni, and C. Frontali. Detection of latent sequence periodicities. *Nucleic Acids Research*, 18(13):3745–3752, 1990. 48

[109] W. K. Pratt. *Digital Image Processing*. Wiley-Interscience, 1991. 11

[110] 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. 30

[111] K. C. Publishing. Barnsley's fern ant the fractal garden. http://www.heartofmath.com/activities/IFSChaosGameApplet.htm. 61

[112] 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. 25, 33, 36

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

[114] I. Rogozin, L. Milanesi, and N. Kolchanov. Gene structure prediction using information on homologous protein sequence. *Comput. Appl. Biosci.*, 12(3):161–170, 1996. 48

[115] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002. 11

[116] 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. 1

[117] 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.Fev.2001]. 1

[118] S. Salzberg, A. Delcher, K. Fasman, and J. Henderson. A decision tree system for finding genes in DNA. *J. Comput. Biol.*, 5(4):667–680, 1998. 48

[119] S. Salzberg, A. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated markov models. *Nucleic Acids Res.*, 26(2):544–548, 1998. 48

[120] 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. 48

[121] J. R. Searle. *Minds, Brains and Science*. Harvard University, March 1986. 17

[122] 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.Fev.2001]. 41

[123] D. Searls. Linguistic approaches to biological sequences. *CABIOS*, 13(4):333–344, 1997. 41

[124] 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.Fev.2001]. 41

[125] 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.Fev.2001]. 41

[126] I. Simon. *Linguagem Formais e Autômatos*. Segunda Escola de Computação, 1981. Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP. 41

[127] D. P. Snustad, M. J. Simmons, and J. B. Jenkins. *Principles of Genetics*. John Wiley and Sons, 1997. 7

[128] E. Snyder and G. Stormo. Identification of coding regions in genomic DNA sequences: an application of dynamic programming and neural networks. *Nucleic Acids Res.*, 21(3):607–613, 1993. 48

[129] E. Snyder and G. Stormo. Identification of protein coding regions in genomic DNA. *J. Mol. Biol.*, 248(1):1–18, 1995. 48

[130] V. Solovyev, A. Salamov, and C. Lawrence. Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Nucleic Acids Res.*, 22(24):5156–5163, 1994. 48

[131] V. Solovyev, A. Salamov, and C. Lawrence. The prediction of human exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *ISMB*, 2:354–362, 1994. 48

[132] G. Stoesser. The EMBL nucleotide sequence database. *Nucleic Acids Research*, 30(2):21–26, 2002. 122

[133] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. MIT Press, 1998. 19

[134] J. E. Tabaska, R. V. Davuluri, and M. Q. Zhang. Identifying the 3'-terminal exon in human dna. *Bioinformatics*, 17(7):602–607, 2001. 48

[135] T. A. Thanaraj. Positional characterisation of false positives from computational prediction of human splice sites. *Nucleic Acids Research*, 28(3):744–754, 2000. 48

[136] The Genome International Sequencing Consortium. Initial sequencing and analysis of the human genome, Fevereiro 2001. nature.com/cgi-taf/DynaPage.taf?file=/nature/journal/v409/n6822/full/409860a0_fs.html [16.Fev.2001]. 1

[137] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 1999. 11, 13, 14

[138] A. Thomas and M. Skolnick. A probabilistic model for detecting coding regions in DNA sequences. *IMA J. Math. Appl. Med. Biol.*, 11(3):149–160, 1994. 48

[139] J. Thompson, D. Higgins, and T. Gibson. Improving the sensitivity of progressive multiple sequence alignment through sequence weighting. *Nucleic Acids Res.*, 22:4673–4680, 1994. 1

[140] 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. 25

[141] E. Uberbacher, J. Einstein, X. Guan, and R. Mural. Gene recognition and assembly in the GRAIL system: progress and challenges. *The Second International Conference on Bioinformatics, Supercomputing and Complex Genome Analysis*, pages 465–476, 1993. 48

[142] Y. Ueda. Japanese attractor. http://www2.masashi.ne.jp/ohta/ueda/ja.html. 55

[143] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. 17

[144] D. Voet and J. G. Voet. *Biochemistry*. John Wiley and Sons, 2nd edition, 1995. 7

[145] E. O. Voit. *Computational Analysis of Biochemical Systems: a pratical guide for biochemists and molecular biologists*. Cambridge University Press, 2000. 23

[146] 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. 14

[147] 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. 48

[148] 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. 48

[149] M. S. Waterman. *Mathematical Methods for DNA sequences*. CRC Press, 1989. 1, 46

[150] T. Werner. Models for prediction and recognition of eukaryotic promoters. *Mamm. Genome*, 10(2):168–175, 1999. 48

[151] J. J. Wiens, editor. *Phylogenic Analysis of Morphological Data*. Smithsonian Series in Comparative Evolutionary Biology. Smithsonian Institution Press, 2000. 1

[152] Wisconsin package version 10.0. Genetics Computer Group (GCG). Madison, Wisc. 122

[153] C. H. Wu. The protein information resource: an integrated public resource of functional annotation of proteins. *Nucleic Acids Research*, 30(2):35–37, 2002. 45, 122

[154] Y. Xu, J. Einstein, R. Mural, M. Shah, and E. Uberbacher. An improved system for exon recognition and gene modeling in human DNA sequences. *ISMB*, 2:376–384, 1994. 48

[155] Y. Xu, R. Mural, and E. Uberbacher. Constructing gene models from accurately predicted exons: an application of dynamic programming. *Comput. Appl. Biosci.*, 10(6):613–623, 1994. 48

[156] A. Zaha. *Biologia Molecular Básica*. Mercado Aberto, 1996. 7

[157] Bioperl, 2003. http://www.bioperl.org/. 122

[158] M. Zhang. Identification of protein coding regions in the human genome based on quadratic discriminant analysis. *Proc. Natl. Acad. Sci. USA*, 94(2):565–568, 1997. 48

[159] D. Zhong. The zhong group. http://www.physics.ohio-state.edu/∼dongping/. 8

[160] Matlab 6. http://www.mathworks.com/products/matlab/ [28.Jun.2002]. 91

[161] Ensembl genome browser. http://www.ensembl.org/ [29.Jun.2002]. 91, 123

[162] Human genome project working draft. http://www.genome.ucsc.edu/ [29.Jun.2002]. 123

[163] National center for biotechnology information. http://www.ncbi.nlm.nih.gov/. 123

[164] Database mining in the human genome initiative. biodatabases.com, 2003. http://www.biodatabases.com/whitepaper02.html. 48

Índice Remissivo