Ferramentas para comparação genômica

Nalvo Franco de Almeida Junior

Tese de Doutorado

Ferramentas para comparação genômica

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Nalvo Franco de Almeida Junior e aprovada pela Banca Examinadora.

Campinas, 03 de maio de 2002.

Prof. Dr. João Carlos Setubal
IC–UNICAMP (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Instituto de Computação
Universidade Estadual de Campinas

Ferramentas para comparação genômica

Nalvo Franco de Almeida Junior[1]

03 de maio de 2002

Banca Examinadora:

- Prof. Dr. João Carlos Setubal
  IC–UNICAMP (Orientador)
- Profª. Drª. Ana Claudia Rasera da Silva
  IQ–USP
- Prof. Dr. Carlos Eduardo Ferreira
  IME–USP
- Prof. Dr. João Meidanis
  IC–UNICAMP
- Prof. Dr. Jorge Stolfi
  IC–UNICAMP

[1]**Apoio financeiro da CAPES, Fulbright e PRONEX/CNPq (664107/1997-4).**

Dedicatória

Dedico esta tese à memória de minha mãe.

Agradecimentos

- Aos meus pais, Nalvo e Olyntha, e à minha irmã Beth, pela dedicação, carinho, suporte e tudo mais. À minha esposa, Ligia, e aos meus filhos, Caio e Julia, sem os quais nada faria sentido.

- Ao meu orientador, Prof. João Carlos Setubal, pela sua amizade, pela paciência e pela intensa dedicação a este trabalho.

- Aos professores do IC que de alguma forma ajudaram durante meu doutoramento. Em especial, agradeço ao Prof. João Meidanis pelas sempre preciosas sugestões, e aos Professores Cid Carvalho de Souza e Claudia Maria Bauzer Medeiros, pelo incentivo.

- A Profª Ana Cláudia Rasera da Silva, do Instituto de Química da USP, pelos importantes comentários e sugestões.

- Ao Prof. Martin Tompa, do Department of Computer Science and Engineering, da University of Washington, pela prestimosa acolhida durante meu doutorado sanduíche.

- Aos colegas do DCT-UFMS, que direta ou indiretamente colaboraram para a realização deste trabalho. Em particular agradeço aos colegas Edson Norberto Cáceres, Leandro Sauer, Marcelo Henriques de Carvalho, Marcelo Ferreira Siqueira, Ronaldo Alves Ferreira e Sergio Roberto de Freitas, pelos comentários e sugestões muito úteis; e ao Denis Santos Silva, pela ajuda na implementação da visualização gráfica das Regiões Ortólogas.

- Aos colegas do IC: Guilherme Pimentel Telles, Guilherme Albuquerque Pinto, José Roberto Menezes Monteiro, Luis Mariano del Val Cura, Marcus Vinícius Alvim Andrade, Maria Emilia Machado T. Walter, Mário Massato Harada e Vagner Katsumi Okura, pelas importantes horas de convívio; e aos funcionários Daniel de Jesus Capeleto e Vera Lúcia de Oliveira Ragazzi, pela paciência.

Resumo

Com o crescente número de genomas seqüenciados e publicados, surge a necessidade de se analisar as seqüências geradas, com o objetivo de se entender melhor caracterizações funcionais dos organismos estudados, assim como aspectos evolutivos. Um projeto genoma de um organismo, em especial de um procarioto, consiste essencialmente de três grandes fases: o seqüenciamento, a anotação e a análise. A última etapa, por sua vez, consiste na tentativa de se obter uma visão global do genoma a partir da anotação e a partir de outras análises, como por exemplo a comparação com outros genomas. E é nesse contexto, comparação de genomas, que esta tese se insere. Nosso trabalho propõe metodologias para comparação detalhada de dois genomas, tanto no nível de DNA, quanto no de seus genes, assim como a implementação dessas metodologias. O principal objetivo é fornecer ao usuário um conjunto de ferramentas para caracterização funcional do organismo estudado, servindo também como ferramental auxiliar na anotação.

Abstract

With increasing availability of published genome sequences, we need to analyse them in order to understand functional and evolutionary issues of the organisms. A genome project, in particular for prokaryotes, consists of three main phases: sequencing, annotation and analysis. The last phase consists of getting an overview of the genome from the annotation and other analysis, like comparison to other genomes, for example. This thesis is about genome comparison. We propose methodologies for detailed comparison of two genomes, at the DNA and their genes levels. The main goal is to provide a set of tools for functional characterization of organisms, serving also as an auxiliar tool for annotation.

Sumário

Preâmbulo ii

Dedicatória vi

Agradecimentos vii

Resumo viii

Abstract ix

1 Introdução 1

1.1 Justificativa 2

1.2 Sumário de resultados 3

1.3 Organização do texto 4

2 Preliminares 5

2.1 Conceitos básicos e notação 5

2.2 Comparação de seqüências 8

2.2.1 Programação Dinâmica 8

2.2.2 Alinhamento local 11

2.2.3 Função afim para a penalização de buracos 12

2.3 Busca em bases de dados 14

3 Comparação de DNA 16

3.1 Árvores de sufixos . . . 17
3.2 Estruturas de repetição . . . 20
3.2.1 Pares maximais exatos . . . 20
3.2.2 Repetições maximais exatas . . . 23
3.2.3 Pares e repetições maximais aproximados . . . 24
3.3 BACON . . . 28
3.4 SBACON . . . 33
3.5 Resultados . . . 33
3.6 Discussão . . . 37

4 Programação dinâmica e comparação de proteínas 39
4.1 Programação dinâmica com curingas . . . 40
4.2 Aplicações e resultados . . . 43
4.2.1 Aplicação: alinhamentos locais . . . 43
4.2.2 Aplicação: identificação da estrutura correta de uma proteína . . . 47
4.3 Discussão . . . 49

5 Comparação de proteomas 51
5.1 Metodologia . . . 51
5.1.1 Genes ortólogos e específicos . . . 54
5.1.2 Regiões específicas (REs) . . . 55
5.1.3 Regiões ortólogas (ROs) . . . 55
5.1.4 Espinha dorsal dos proteomas . . . 57
5.1.5 Famílias de genes parálogos . . . 57
5.2 Implementação – o programa EGG . . . 59
5.2.1 Descrição das fases de EGG . . . 59
5.2.2 Famílias de genes parálogos . . . 65
5.2.3 Alguns detalhes sobre EGG . . . 66
5.3 Resultados . . . 67

5.4 Discussão 74
5.4.1 Aplicação: anotação de genoma 74
5.4.2 Outros trabalhos 77

6 Conclusões e perspectivas **80**

A Resultados adicionais das comparações de proteomas **86**

B Detalhes operacionais de EGG **94**
B.1 Descrição da ferramenta 94
B.2 Alguns exemplos dos arquivos de saída 97

Bibliografia **100**

Lista de Tabelas

1.1 Tópicos a respeito da comparação de genomas. 3

3.1 Maiores repetições maximais aproximadas encontradas na comparação de `Xac` e `Xcc`. 34

3.2 As 15 maiores repetições maximais exatas encontradas por BACON na comparação das cepas de *H. pylori*. 35

3.3 Repetições tandem exatas encontradas por BACON nas duas cepas de *H. pylori*. 36

3.4 Uma pequena amostra das DSs encontrados nas cepas de *H. pylori*. 36

4.1 Posições alcançadas pelo nosso algoritmo. 49

4.2 Posições obtidas pela programação dinâmica usual. 49

5.1 Genomas usados nas comparações, objetivando principalmente as comparações com `Xfa`, `Xac` e `Xcc`. 68

5.2 Genomas usados nas comparações de proteomas, objetivando comparações com *Agrobacterium tumefaciens*. 69

5.4 Números de runs e regiões ortólogas entre *Agrobacterium* com *Mesorhizobium*. 69

A.1 Números de matches e BBHs entre *Xylella* com outros genomas. 86

A.2 Número de matches e BBHs entre *Agrobacterium* com *Mesorhizobium*. 87

A.3 Número de matches e BBHs entre *Agrobacterium* com *Sinorhizobium*. 87

A.4 Números de runs e ROs entre *Xylella* com outros genomas. 88

A.5 Números de runs e ROs entre *Xanthomonas citri* e outros genomas. 89

A.6 Números de runs e regiões das comparações de *Agrobacterium* com *Sinorhizobium*. 89

Lista de Figuras

2.1 Exemplo de como as duas fitas de DNA pareiam. 6

3.1 Trecho de similaridade aproximada entre as *Xanthomonas*. 16

3.2 Árvore de Sufixos para a seqüência `xabxa$`. 18

3.3 Árvore de Sufixos para a seqüência `xabxa`. 19

3.4 Dois pares maximais exatos da seqüência $s =$ `abracadabradabra`. 21

3.5 Um par aproximado pode ser visto como a junção de dois pares exatos. . . . 24

3.6 Dois pares maximais exatos, (p, q, α) e (p', q', α') 24

3.7 Obtenção de repetições maximais aproximadas. 28

3.8 Algumas repetições maximais exatas encontradas na comparação das *Mycoplasmas* 30

3.9 Repetições tandem exatas encontradas por BACON. 31

3.10 Algumas DSs encontradas por BACON na comparação das *Xanthomonas*. . . 32

3.11 Algumas repetições maximais da comparação das *Xanthomonas*. 32

3.12 Maiores repetições maximais exatas encontradas na comparação de `Xac` e `Xcc`. 34

3.13 Maiores repetições maximais exatas encontradas em `Xac`. 35

4.1 Um alinhamento com 3 regiões curingas. 41

4.2 Dois alinhamentos das mesmas seqüências. 45

4.3 Dois alinhamentos de outra instância do problema, com os mesmos valores para os parâmetros. 46

5.1 Exemplo de uma RO. As arestas representam ortologia. 53

5.2 Exemplo de uma espinha dorsal de duas RGCs. 53

5.3 Exemplo de um run anti-paralelo consistente. A ordem dos genes no genoma de baixo aparecem invertida. 62
5.4 Exemplo de uma RO encontrada na comparação de *Xylella* e *Neisseria*. . . . 63
5.5 RO encontrada na comparação de *Xylella fastidiosa* e *Haemophilus influenzae*. 64
5.6 BBHs entre `Xac` e `Xcc`. 70
5.7 Trecho do arquivo que mostra o alinhamento dos proteomas de `Xac` e `Xcc`. . 70
5.8 BBHs entre o replicon `At1` de *Agrobacterium tumefaciens* e `Smc` de *Sinorhizobium meliloti*. 71
5.9 Exemplo de uma página www contendo quatro ROs entre *Xylella fastidiosa* e *E. coli*. 72
5.10 Algumas famílias de parálogos de `Xfa`. 73
5.11 RO obtida na comparação de *Xylella* e *Xanthomonas citri*. 75
5.12 RO obtida na comparação de *Xylella fastidiosa* e *Caulobacter crescentus*. . . 76
5.13 Visualização gráfica da mesma RO da figura 5.12. 76
5.14 Visualização gráfica de uma RO encontrada na comparação das *Xanthomonas*. 77
6.1 Árvore filogenética construída a partir das distâncias baseadas no número de BBHs. 82
6.2 Região específica de *Xylella fastidiosa* com relação a *Xylella fastidiosa* (Pierce's disease). 85
A.1 BBHs entre os genomas *Xylella fastidiosa* e *Haemophilus influenzae*. 88
A.2 RO encontrada entre os genomas `ci` e `ca`. 90
A.3 Região com alta conservação encontrada na comparação de `At2` e `Smc`. 91
A.4 Número médio de matches e BBHs, considerando as 15 comparações envolvendo `Xfas` e `Xac`. 92
A.5 Número médio de runs e RCOs, considerando as 15 comparações envolvendo `Xfas` e `Xac`. 92
B.1 Trecho do arquivo `cica.12`. 96
B.2 Trecho do arquivo `cica.k12`, resultante da comparação entre `Xac` e `Xcc`. . . . 97
B.3 Trecho do arquivo `cica.bes`, resultante da comparação entre as *Xanthomonas*. 99

Capítulo 1

Introdução

Os indiscutíveis avanços da Biotecnologia, aliados aos avanços da Biologia Molecular, em particular no desenvolvimento de técnicas de seqüenciamento de DNA, têm gerado uma imensa quantidade de dados, que geralmente consistem em seqüências de DNA e de proteínas. A existência e o rápido crescimento dessa grande massa de dados nos levam naturalmente à seguinte questão: como podemos transformar esses dados em relevantes informações biológicas, capazes de proporcionar a criação de novas drogas, vacinas e novos tratamentos de doenças genéticas? Enfim, como podemos entender, de uma forma ampla, todos os mecanismos biológicos que ditam a funcionalidade dos seres vivos?

É óbvio que o uso de computadores no trato desses conjuntos de dados é indispensável, tanto na maneira de interpretar as seqüências, quanto na forma de armazená-las. Menos óbvio é o uso da Ciência da Computação também na descoberta de técnicas para resolver problemas mais complexos de Biologia Molecular. Esses problemas envolvem, entre outras coisas, montagem de fragmentos, importante passo no seqüenciamento de genomas; construção de árvores filogenéticas, que são muito úteis na classificação de espécies; predição de genes e suas funções; e, principalmente para esta tese, a comparação de seqüências longas de DNA e de proteína, que consiste numa das principais ferramentas atuais da Biologia Molecular.

Todos esses problemas motivaram o surgimento de uma nova área de pesquisa, denominada **Bioinformática**, também chamada de **Biologia Molecular Computacional**, que vem crescendo a cada ano, principalmente devido aos inúmeros projetos genoma que surgem atualmente.

Um projeto genoma, em particular de um procarioto, consiste essencialmente de três grandes etapas: o seqüenciamento, a anotação e a análise. O seqüenciamento consiste na descoberta da exata seqüência de DNA de cada replicon do organismo; a anotação consiste na predição da posição de cada gene do genoma, incluindo a determinação de sua função, etapa aliás

que envolve principalmente a comparação com outros genes dos quais se sabe a função; a análise compreende a obtenção de uma visão mais geral do organismo, no sentido de se obter uma caracterização funcional, baseada na anotação e em outras informações, tais como a comparação com outros genomas. E é neste tópico, comparação de genomas, que esta tese se insere.

1.1 Justificativa

Com a realização de diversos projetos genoma, os laboratórios passaram a dispor de tecnologia para a geração, em larga escala, de muitos dados sobre genomas, genes, proteínas, etc. Para se ter uma idéia aproximada do volume de dados gerados, atualmente (março-2002) 81 genomas completos, 71 de procariotos e 10 de eucariotos, já foram publicados; 437 estão em andamento, sendo 264 de procariotos [12, 39].

Uma forma de auxiliar na descoberta de tais informações relevantes passa, certamente, pela determinação dos papéis que os mais diversos objetos envolvidos num genoma desempenham. Esses papéis estão muitas vezes relacionados às características estruturais de cada objeto. Isso acontece de forma bem clara no caso de proteínas, que têm suas funções determinadas diretamente pela sua forma e estruturação [43, 18]. Assim, é de se esperar que a comparação entre objetos, nas suas formas mais primárias, nos tragam pistas de relacionamentos entre eles e, por conseqüência, entre suas funcionalidades.

No caso de dois genomas é de se esperar, portanto, que a comparação entre seus objetos, especificamente seqüências de DNA e seus genes, seja útil na determinação de funcionalidades comuns. A idéia então é termos ferramentas que evidenciem aspectos funcionais comuns, além de proporcionarem uma melhor compreensão de como os genes se organizam nos diversos genomas.

Especificamente, a comparação de genomas tem como principais objetivos:

- detecção de similaridades e diferenças entre genomas completos, no nível de DNA;
- identificação de genes ou grupos de genes envolvidos em diversas funções;
- identificação de genes ou grupos de genes responsáveis por características fenotípicas peculiares a um genoma particular;
- identificação de genes homólogos (genes descendentes de um mesmo gene anscestral);
- anotação de genes de genomas não completos; e
- inferência de relações filogenéticas entre os organismos.

1.2 Sumário de resultados

O resultado principal descrito nesta tese é um conjunto de metodologias para a comparação de genomas, tanto no nível de DNA, quanto no nível de seus genes, e a implementação dessas metodologias.

Como resultado da implementação das metodologias, as ferramentas desenvolvidas e disponibilizadas são as seguintes:

- BACON – Bacterial Comparator, que compara dois genomas no nível de DNA;
- SBACON – Self BACON, que encontra repetições aproximadas na seqüência de DNA de um genoma; e
- EGG – Extended Genome-Genome comparison, que compara dois genomas no nível de seus genes.

A tabela 1.1 mostra tópicos relacionados aos principais objetivos de uma comparação de dois genomas e também da comparação de um genoma com ele mesmo, e as respectivas ferramentas desenvolvidas para tais fins, juntamente com o local na tese onde são descritas.

Tipo de comparação	**Comparação de dois genomas**	**Genoma comparado com ele mesmo**
DNA	repetições aproximadas BACON capítulo 3	repetições aproximadas SBACON capítulo 3
genes individuais que codificam proteína	ortólogos e específicos EGG capítulo 5	parálogos EGG capítulo 5
genes em contexto	regiões ortólogas e específicas EGG capítulo 5	*não se aplica*
alinhamento de genomas	espinha dorsal de proteomas EGG capítulo 5	*não se aplica*

Tabela 1.1: Tópicos a respeito da comparação de genomas. Para cada tópico, a tabela mostra o principal objetivo (primeira linha), a ferramenta que atende ao objetivo (segunda linha) e em qual parte da tese se encontra a sua descrição (terceira linha).

As metodologias e os programas propostos nesta tese já foram utilizados em alguns projetos genoma, como os das bactérias *Xylella fastidiosa* (Pierce's disease) [25], *Xanthomonas axonopodis* pv. *citri* e *Xanthomonas campestris* pv. *campestris* [16] e *Agrobacterium tumefaciens* [74].

Este trabalho nos possibilitou a publicação [59], além da participação, como co-autor, em outras três [16, 73, 74].

Adicionalmente propomos, no capítulo 4, uma utilização alternativa da programação dinâmica na comparação de duas proteínas, que leva em conta a presença de regiões curingas. Esta proposta resultou na publicação [4].

1.3 Organização do texto

O texto está organizado como segue. No capítulo 2 descrevemos alguns conceitos básicos, assim como algumas notações, necessários durante a leitura da tese. No capítulo 3 descrevemos uma metodologia baseada em repetições aproximadas e árvores de sufixos para comparar dois genomas no nível de DNA, além dos programas BACON e SBACON. No capítulo seguinte propomos uma forma alternativa de uso da programação dinâmica na comparação de proteínas. Em seguida, no capítulo 5, propomos uma metodologia para a comparação de proteomas, assim como o programa resultante, EGG. Finalmente, no capítulo 6, fazemos alguns comentários finais, além de apontarmos alguns possíveis tópicos para estudos futuros.

Capítulo 2

Preliminares

Neste capítulo fazemos uma descrição dos principais conceitos e notações utilizados no decorrer do texto, incluindo os conceitos de Biologia Molecular que consideramos mais importantes.

Além dos conceitos de Biologia Molecular vistos na seção 2.1, descrevemos conceitos importantes envolvendo seqüências de símbolos, que são os principais objetos tratados nesta tese. Especificamente, estamos interessados em comparar seqüências. Neste sentido, descrevemos neste capítulo um conceito muito importante na comparação de seqüências, o conceito de *alinhamento*, visto na seção 2.2, além de uma das principais ferramentas computacionais para tal fim, a *programação dinâmica*.

No final do capítulo, descrevemos o problema de busca em bases de dados, onde uma seqüência deve ser comparada com milhares de outras.

2.1 Conceitos básicos e notação

Este texto pressupõe que o leitor já tenha um certo conhecimento básico de Biologia Molecular. Por outro lado estaremos, durante o texto e na medida do possível, introduzindo os conceitos básicos necessários.

Vamos aqui introduzir alguns conceitos e notações, necessários neste ponto para que o leitor consiga ter uma visão geral dos problemas tratados na tese.

Para efeitos deste trabalho uma seqüência de **DNA** é uma seqüência de letras escrita no alfabeto formado por `A`,`C`,`G`,`T`. Cada uma dessas letras é denominada **nucleotídeo** ou **base**. Algumas vezes letras adicionais são admitidas, para representar combinações de nucleotídeos, quando não se tem certeza de qual letra deve ocupar uma posição. Esse alfabeto é descrito

em [46].

O DNA inteiro de um ser vivo é chamado de **genoma**. Costuma variar em tamanho de acordo com a espécie, desde milhões de letras, no caso de bactérias, até bilhões de letras, no caso de mamíferos. Além disso, costuma ser dividido em unidades, chamadas **cromossomos**. O DNA é formado por uma **fita dupla**. As duas fitas tem **orientações** opostas e são tais que um `A` sempre pareia com um `T` e um `C` sempre pareia com um `G`. As orientações são ditas serem da extremidade $5'$ para a extremidade $3'$. A figura 2.1 mostra um pequeno exemplo de como as fitas se pareiam. Uma fita é dita ser o **complemento-reverso** da outra. O complemento-reverso de um trecho de DNA g é denotado por g^{CR}. Medimos o tamanho de um trecho de DNA de fita dupla pelo seu número de **pares de base**, denotados por **bp** (de *base-pairs*).

```
5'      ...  A A T C G G C A T T C G G T C  ...      3'
             | | | | | | | | | | | | | | |
3'      ...  T T A G C C G T A A G C C A G  ...      5'
```

Figura 2.1: Exemplo de como as duas fitas de DNA pareiam.

O DNA de um organismo contém a informação necessária para a fabricação de *proteínas*. **Proteínas** são seqüências curtas de letras de um outro alfabeto, o alfabeto de 20 **amino-ácidos**. Cada três nucleotídeos codificam um aminoácido. Essa codificação é feita seguindo o que é chamado de **código genético**. Cada uma das $4^3 = 64$ possibilidades é denominada **codon**. Dos 64 codons possíveis, alguns codificam o mesmo aminoácido. Por isso temos apenas 20 aminoácidos. Existem ainda os **codons de parada** (ou **stop codons**), que determinam o final da tradução de uma seqüência de um alfabeto para o outro. Uma seqüência que inicia com um codon de início e não possui codon de parada é chamada de **open reading frame (ORF)**. Os trechos que são codificados em informação genética são chamados de **genes**. Alguns genes codificam proteínas, outros codificam RNA. O conjunto de genes de um genoma que codificam proteína é chamado de **proteoma**. Apesar de um gene poder codificar mais de uma proteína, e apesar de haver genes que não codificam proteína, vamos, quando for conveniente, usar os termos *gene* e *proteína* indistintamente. Além disso, quando tratamos de genomas, estamos apenas predizendo que um gene codifica tal ou tais proteínas, assim é comum usarmos o termo **proteína predita** de um determinado gene.

A seqüência de aminoácidos de uma proteína é conhecida como sua **estrutura primária**. No entanto, essas seqüências se enrolam tri-dimensionalmente formando diferentes formas até um quarto nível de estruturação. Descobrir a estrutura de uma proteína é muito importante, já que ela vai determinar a sua função.

Os organismos são divididos em dois grandes grupos: os *eucariotos* e os *procariotos*. Aqueles que têm núcleo são chamados de **eucariotos**. Os outros são os **procariotos**. Os eucariotos abrigam seu material genético no núcleo e em geral têm mais de um cromossomo. Uma grande diferença entre eucariotos e procariotos é que no primeiro grupo os genes não são em geral formados por seqüências contíguas de DNA. Um gene em eucarioto é dividido em pedaços chamados **exons**. Os pedaços intermediários (que não são traduzidos) são chamados de **íntrons**.

Dois genes são **homólogos** se são descendentes de um mesmo gene ancestral. Se forem de genomas distintos são chamados de **ortólogos**. Se forem de um mesmo genoma, são chamados de **parálogos**.

Muitas vezes estaremos interessados em medir a **similaridade** entre duas seqüências, sejam elas de DNA ou de proteína. O objetivo é o de inferir homologia através da similaridade.

O **tamanho** de uma seqüência (ou cadeia) s, denotado por $|s|$, é o número de símbolos que ela contém. O símbolo que ocupa a posição i é denotado por s_i. Assim, uma seqüência s contém os símbolos $s_1, \ldots, s_{|s|}$. Se $|s| = 0$, dizemos que s é **vazia**. Uma **subseqüência** de s é uma seqüência que pode ser obtida a partir de s retirando alguns de seus símbolos. Uma **subcadeia** de s é um trecho formado por símbolos contíguos de s. Assim, toda subcadeia é também uma subseqüência.

A **concatenação** de duas cadeias s e t, denotada por st é formada por s, seguida do acréscimo, no final de s, de todos os símbolos de t, exatamente na ordem em que eles aparecem em t. Um **prefixo** de s é qualquer subcadeia do tipo $s_1 s_2 \ldots s_j$, $j \leq |s|$. Se $j \leq 0$, então dizemos que o prefixo é vazio. Analogamente um **sufixo** de s é qualquer cadeia do tipo $s_j s_{j+1} \ldots s_{|s|}$, $j \geq 1$. Se $j > |s|$, então dizemos que o sufixo é vazio.

Um **grafo** é um conjunto de elementos chamados de **vértices** e um conjunto de elementos chamados de **arestas**. Um grafo $\mathcal{G}$ é denotado por $\mathcal{G} = (V, E)$, onde V é o conjunto de vértices e E o conjunto de arestas. Cada aresta é um par não-orientado de vértices. Assim, $(u, v) = (v, u)$. Um grafo é **simples**, quando (u, v) é tal que $u \neq v, \forall u, v \in V$ e tal que não há arestas múltiplas ligando o mesmo par de vértices. Dizemos que dois vértices u, v são **vizinhos** em $\mathcal{G}$, se $(u, v) \in E$.

Um grafo $\mathcal{G}$ é **bipartido** se V pode ser particionado em dois conjuntos X e Y tais que qualquer aresta (u, v) é tal que: ou $u \in X$ e $v \in Y$, ou $u \in Y$ e $v \in X$. Estaremos interessados em grafos bipartidos quando compararmos proteomas. Um conjunto $C \subseteq V$ é uma **clique** de $\mathcal{G}$ se para qualquer par $u, v \in C$, é verdade que $(u, v) \in E$. Uma clique C é **maximal** quando o acréscimo de qualquer vértice fora de C faz com que C deixe de ser clique. Um clique C é **máxima** se não existir clique no grafo contendo mais vértices que em C. Estaremos interessados em cliques quando procurarmos famílias de genes parálogos de

um proteoma.

Muitos dos conceitos e definições descritos aqui serão convenientemente repetidos ou reformulados durante o texto, para que sejam interpretados corretamente em cada ocasião.

Felizmente muitos são os textos introdutórios disponíveis para uma melhor compreensão de conceitos aqui utilizados. Setubal e Meidanis [60] e Gusfield [32], por exemplo, apresentam execelente material sobre os mais diversos problemas de Biologia Molecular Computacional, incluindo o tópico de comparação de seqüências. Em particular, os principais conceitos e convenções utilizados nesta tese estão de acordo com os três primeiros capítulos do livro de Setubal e Meidanis [60].

Para uma leitura sobre tópicos um pouco mais avançados de Biologia Computacional, sugerimos o livro de Pevzner [54] e o livro editado por Salzberg *et al* [56], além das notas escritas por Phil Green [30]. Os livros de Cummings e Klug [15], Lehninger [40] e Lewin [43] são excelentes para conceitos de Biologia Molecular.

2.2 Comparação de seqüências

Vamos agora apresentar alguns conceitos úteis para a compreensão do texto, referentes à comparação de seqüências. Especificamente, estamos interessados em ferramentas para comparar seqüências. A *programação dinâmica* é uma das mais utilizadas, e é referenciada com muita freqüência durante o texto. Por isso, vamos agora descrevê-la.

2.2.1 Programação Dinâmica

Descrevemos nesta seção alguns algoritmos baseados em programação dinâmica para a comparação de seqüências biológicas. Apesar desses algoritmos servirem também ao propósito de se comparar seqüências de quaisquer tipos, estamos aqui interessados, como já dissemos, na comparação de proteínas.

A técnica de programação dinâmica consiste em resolver uma instância de um problema usando soluções já obtidas para instâncias menores do mesmo problema. Alguns problemas de otimização possuem a característica de poderem ser tratados com essa abordagem. Na prática, a programação dinâmica resolve todos os sub-problemas somente uma vez e armazena as soluções numa tabela, de tal forma que seja possível a recuperação dessas soluções, sem que tenham de ser recomputadas. Uma excelente descrição dessa técnica, incluindo várias aplicações, pode ser vista no livro de Cormen, Leiserson e Rivest [14].

Especificamente, estamos interessados na utilização da programação dinâmica para a cons-

trução de alinhamentos entre duas seqüências. Um alinhamento basicamente consiste em acrescentar espaços nas duas seqüências, de tal forma que ambas fiquem do mesmo tamanho. O ideal é que o alinhamento evidencie o emparelhamento de trechos das seqüências que sejam similares, enquanto que os espaços apareçam bem mais em trechos não-similares. A seguir, definimos formalmente um alinhamento.

Definição 2.1 *Dadas as seqüências* $s = s_1 \dots s_m$ *e* $t = t_1 \dots t_n$*, com símbolos pertencentes ao alfabeto* Σ*, e com* $m, n \geq 0$*, um* **alinhamento** *de* s *e* t *é um mapeamento de* s *e* t *nas seqüências* s' *e* t'*, respectivamente, cujos símbolos pertencem ao alfabeto* $\Sigma' = \Sigma \cup \{-\}$*, onde o símbolo '*$-$*' é chamado de* **espaço***, tal que:*

1. $|s'| = |t'| = l$*;*
2. *a remoção dos espaços de* s' *e* t' *leva a* s *e* t*, respectivamente; e*
3. *não é permitida a condição* $s'_i = - = t'_i$*,* $1 \leq i \leq l$*.*

O **valor** *do alinhamento é dado por*

$$\sum_{i=1}^{l} \sigma(s'_i, t'_i). \tag{2.1}$$

onde $\sigma : \Sigma' \times \Sigma' \rightarrow \mathbf{R}$ *é uma função simétrica tal que* $\sigma(a, b)$ *denota o valor do emparelhamento entre o símbolo* a *com o símbolo* b*,* $\forall a, b \in \Sigma'$*. Apesar de não utilizado, o par* $(-, -)$ *deve ser tal que* $\sigma(-, -) = 0$*. A uma seqüência contígua de espaços em* s' *ou em* t' *denominamos* **buraco***. A cada coluna do tipo* (s'_i, t'_i)*, com* $s'_i \neq t'_i$*,* $s'_i, t'_j \neq -$*, damos o nome de* **substituição***. Se a coluna for do tipo* (s'_i, t'_i)*, com* $s'_i = t'_i$*,* $s'_i \neq -$*, damos o nome de* **casamento***. Uma coluna do tipo* $(s'_i, -)$ *recebe o nome de* **remoção** *e uma coluna do tipo* $(-, t'_i)$ *recebe o nome de* **inserção***.*

Uma observação pertinente diz respeito à escolha da função σ. Podemos estar interessados em alinhar duas seqüências no sentido de medir a *similaridade* entre elas, ou no sentido de medir a *distância de edição* entre elas. Na similaridade, estamos interessados em saber o quanto as duas seqüências são parecidas, enquanto que na distância estamos interessados em saber o número mínimo necessário de operações (inserções, remoções e substituições) para transformar uma seqüência na outra. Ou seja, a distância serve para sabermos quão diferentes são as seqüências. Muitas vezes estas medidas são intercambiáveis, no sentido de que uma pequena distância significa uma grande similaridade e vice-versa. No entanto, a similaridade é mais flexível: em alguns casos, exige-se que a medida de distância obedeça à desigualdade triangular. No nosso caso estaremos usando a noção de similaridade entre

duas seqüências. Uma descrição detalhada das duas abordagens pode ser vista no livro de Setubal e Meidanis [60].

Definição 2.2 *Um* **alinhamento ótimo** *de duas seqüências s e t é definido como um alinhamento que maximiza a soma 2.1, entre todos os possíveis alinhamentos de s e t.*

Vamos agora descrever sucintamente um algoritmo eficiente, baseado em programação dinâmica, para encontrar os alinhamentos ótimos de duas seqüências s e t, de tamanhos m e n, respectivamente. Para tanto, vamos utilizar uma matriz A, $m \times n$. Seja $A_{i,j}$ o valor de um alinhamento ótimo de $s_1 \ldots s_i$ e $t_1 \ldots t_j$. Assim, o valor de um alinhamento ótimo de s e t é dado por $A_{m,n}$. A idéia chave da programação dinâmica é resolver o problema mais geral computando todos os valores $A_{i,j}, 0 \leq i \leq m, 0 \leq j \leq n$, em ordem não-decrescente de i e j.

A base do algoritmo é o cálculo de

$$A_{0,0} = 0,$$

$$A_{i,0} = \sum_{k=1}^{i} \sigma(s_k, -) \quad \text{e}$$

$$A_{0,j} = \sum_{k=1}^{j} \sigma(-, t_k).$$

A fórmula geral de recorrência, para $1 \leq i \leq m$ e $1 \leq j \leq n$, é

$$A_{i,j} = \max \begin{cases} A_{i-1,j} + \sigma(s_i, -), \\ A_{i-1,j-1} + \sigma(s_i, t_j), \\ A_{i,j-1} + \sigma(-, t_j). \end{cases}$$

Ao final, $A_{m,n}$ é o valor de um alinhamento ótimo de s e t.

O algoritmo de programação dinâmica baseado nas fórmulas de recorrência acima, que calcula apenas o valor de um alinhamento ótimo das duas seqüências, requer tempo e espaço $O(mn)$. Para se obter os alinhamentos, basta armazenar, para cada par (i, j), quais as posições, entre $A_{i-1,j}$, $A_{i-1,j-1}$ e $A_{i,j-1}$, foram utilizadas. Isto também pode ser facilmente feito em tempo $O(m + n)$ [60].

A escolha de uma boa função σ certamente influencia nos resultados do algoritmo. Como nosso objetivo é o de evidenciar trechos similares, já que estamos trabalhando com similaridade, é de se esperar que um bom valor para uma coluna do alinhamento que representa uma substituição seja menor que o valor de uma coluna de casamento. Pela mesma razão, espera-se que o valor de uma inserção ou remoção custe mais que um casamento. Os valores 1, -1 e -2 são exemplos de bons valores para casamento, substituição e espaço, respectivamente.

Na prática, os sistemas de pontuação não são tão simples assim. No caso de proteínas, por exemplo, é importante sermos capazes de distinguir entre diversos pares de aminoácidos. Um emparelhamento de aminoácidos com características físico-químicas similares deve receber mais pontos do que um par de aminoácidos menos similares. Na prática utiliza-se, por exemplo, as chamadas *matrizes PAM* [17, 58] ou então *BLOSUM* [33] .

No que diz respeito aos buracos, devemos tratá-los como eventos unificados, no sentido de ser pouco provável que k espaços contíguos tenham ocorrido independentemente ou em diferentes instantes. Assim, precisamos de uma função que melhor penalize buracos. Isto vai ser visto na seção 2.2.3.

O tipo de alinhamento visto aqui é chamado de *alinhamento global*, já que queremos alinhar toda a seqüência s com toda a seqüência t. A seguir veremos um caso especial de alinhamento, chamado de *alinhamento local*, muito utilizado na comparação de proteínas.

2.2.2 Alinhamento local

Muitas vezes, ao se comparar proteínas de diferentes famílias, percebe-se a presença de unidades funcionais ou estruturais comuns (chamados de domínios), mesmo que no geral as seqüências tenham baixa similaridade. Mesmo em se tratando de proteínas da mesma família, é possível detectar a presença de tais unidades, separadas por trechos não-similares. Nesses casos, o alinhamento global pode perder a localização de tais similaridades locais.

A solução para esse problema pode ser o uso de outro tipo de alinhamento: o chamado **alinhamento local**, onde queremos encontrar um alinhamento entre uma subcadeia de s com uma subcadeia de t. Ou seja, queremos encontrar um alinhamento local de s e t com valor máximo, entre todos os possíveis alinhamentos locais de s e t.

O algoritmo nada mais é do que uma variação do algoritmo de alinhamento global. Desta vez, a interpretação do valor a ser armazenado na posição (i, j) da matriz A muda: $A_{i,j}$ armazena o valor do melhor alinhamento (de maior valor) entre um sufixo de $s_1 \dots s_i$ e um sufixo de $t_1 \dots t_j$.

A inicialização da matriz A é tal que os elementos da primeira coluna e os elementos da primeira linha são todos iguais a zero. Essa inicialização é feita dessa maneira porque sempre existe o alinhamento entre os sufixos vazios de $s_1 \dots s_i$ e $t_1 \dots t_j$. A recorrência para as posições posteriores é dada por:

$$A_{i,j} = \max \begin{cases} A_{i-1,j} + \sigma(s_i, -), \\ A_{i-1,j-1} + \sigma(s_i, t_j), \\ A_{i,j-1} + \sigma(-, t_j), \\ 0. \end{cases}$$

A exceção agora fica por conta da possibilidade adicional de um alinhamento vazio, de valor 0, que não ocorre no caso global. Note que agora todas as posições da matriz terão valores não-negativos.

Ao final, basta encontrarmos o maior elemento de A. Qualquer posição contendo esse valor pode ser usado como ponto de partida para se encontrar o alinhamento, como feito no caso global. Pode-se ainda obter todos os alinhamentos locais cujos valores estejam acima de um certo valor de corte. A complexidade desse algoritmo é exatamente a mesma do caso global, ou seja, $O(mn)$ de tempo e espaço.

Outras variações

Outras variações são possíveis. Podemos, por exemplo, estar interessados em não penalizar os buracos nas pontas de uma das seqüências. Isso acontece quando temos uma delas muito menor que a outra. Se não quisermos penalizar buracos inseridos nas pontas de s, temos o caso **global-local**, já que estamos esperando poder alinhar globalmente a seqüência s com um trecho de t. O inverso é chamado de **local-global**.

A inicialização obviamente vai depender de qual dos casos estamos tratando. Ao final, o resultado vai estar ou na última linha de A (caso global-local) ou na última coluna de A (caso local-global).

O algoritmo para o caso global é muitas vezes referido na literatura como algoritmo de Needleman-Wunsch [45], enquanto que o de alinhamento local é referido como algoritmo de Smith-Waterman [61].

2.2.3 Função afim para a penalização de buracos

Considerando que um alinhamento procura mostrar como duas seqüências se relacionam evolutivamente a partir de um ancestral comum, podemos entender um buraco como um evento mutacional único. Além disso, como dissemos, proteínas são muitas vezes formadas por domínios, onde alguns deles podem aparecer numa proteína e não na outra, é natural imaginarmos que a seqüência que não possui o domínio contribua com um buraco, ou com outro domínio, naquele trecho do alinhamento. Tais buracos não devem ser penalizados pura e simplesmente com a soma dos espaços que contêm.

Outra consideração que deve ser levada em conta está relacionada ao fato de que k espaços isolados não devem ser mais prováveis que um buraco contendo k espaços. Assim, precisamos de uma função que nos forneça o custo atribuído a um buraco de tamanho k, que seja fácil de calcular, além de ser tal que $w(k) \leq kw(1)$. Uma tal função w é chamada de **sub-aditiva**.

Uma função afim do tipo $w(k) = h+kg$, $k \geq 1$, com $w(0) = 0$ é sub-aditiva, quando $h, g > 0$.

A principal justificativa para usarmos uma função w como a descrita acima é a simplicidade, já que com apenas dois parâmetros, h para iniciar e g para continuar um buraco, somos capazes de construir uma função que penaliza menos um buraco de k espaços do que k espaços isolados. Além disso, podemos calcular $w(k+1)$ a partir de $w(k)$ em tempo constante, já que

$$w(k+1) = h + (k+1)g = h + kg + g = w(k) + g.$$

O algoritmo de programação dinâmica para encontrar um alinhamento ótimo de s e t usando a função w faz uso de três matrizes, ao invés de uma, como antes. Isto acontece pelo fato de precisarmos distinguir entre estarmos tratando de um primeiro espaço de um buraco e a continuação de um buraco. As matrizes A, B, e C têm, respectivamente, os seguintes significados:

A: s_i casa com t_j
B: um espaço em s casa com t_j
C: s_i casa com um espaço em t

Assim, $A_{i,j}$ é o valor máximo de um alinhamento entre $s_1 \ldots s_i$ e $t_1 \ldots t_j$, que termina com o emparelhamento de s_i e t_j; $B_{i,j}$ é o valor máximo de um alinhamento entre $s_1 \ldots s_i$ e $t_1 \ldots t_j$, que termina com o emparelhamento de um espaço em s e t_j; e $C_{i,j}$ é o valor máximo de um alinhamento entre $s_1 \ldots s_i$ e $t_1 \ldots t_j$, que termina com o emparelhamento de s_i com um espaço em t.

As recorrências, que agora são três, podem ser vistas abaixo, para $1 \leq i \leq m$ e $1 \leq j \leq n$.

$$A_{i,j} = \sigma(s_i, t_j) + \max \left\{ \begin{array}{l} A_{i-1,j-1}, \\ B_{i-1,j-1}, \\ C_{i-1,j-1} \end{array} \right\},$$

$$B_{i,j} = \max \left\{ \begin{array}{l} A_{i,j-1} - (h+g), \\ B_{i,j-1} - g, \\ C_{i,j-1} - (h+g) \end{array} \right\},$$

$$C_{i,j} = \max \left\{ \begin{array}{l} A_{i-1,j} - (h+g), \\ B_{i-1,j} - (h+g), \\ C_{i-1,j} - g \end{array} \right\}.$$

A inicialização das três matrizes depende se queremos ou não penalizar os buracos nas pontas das seqüências. Supondo que vamos penalizar todos os buracos, a inicialização deve ser feita da seguinte forma.

$$\begin{aligned} A_{0,0} &= 0, \\ A_{i,0} &= -\infty, 1 \leq i \leq m, \\ A_{0,j} &= -\infty, 1 \leq j \leq n; \end{aligned}$$

$$\begin{aligned} B_{i,0} &= -\infty, 0 \leq i \leq m, \\ B_{0,j} &= -(h + jg), 1 \leq j \leq n; \end{aligned}$$

$$\begin{aligned} C_{i,0} &= -(h + ig), 1 \leq i \leq m, \\ C_{0,j} &= -\infty, 0 \leq j \leq n. \end{aligned}$$

O resultado final do alinhamento é dado por $\max\{A_{m,n}, B_{m,n}, C_{m,n}\}$. A complexidade desse algoritmo é também $O(mn)$, já que estamos usando apenas 3 matrizes $m \times n$.

Obviamente o algoritmo que usa penalização de buracos com função afim pode ser usado para alinhamentos diversos (local, global,global-local, etc.). Basta para isso tomarmos o cuidado de inicializar as matrizes corretamente, além de fazer as devidas alterações nas recorrências.

2.3 Busca em bases de dados

Com o rápido crescimento do volume de seqüências gerado nos laboratórios, uma das mais utilizadas aplicações de algoritmos de comparação de seqüências na Bioinformática aparece na manipulação de bases de dados contendo essas seqüências biológicas. A razão óbvia para esse fato é que a comparação de duas seqüências, em especial de duas proteínas, pode levar a conclusões muito fortes acerca de suas estruturas e, por conseqüência, de suas funcionalidades.

A complexidade quadrática dos algoritmos baseados em programação dinâmica ainda os desabilita para o uso com grande quantidades de seqüências. Com isso, novas técnicas vem surgindo, em especial heurísticas que tem sido muito utilizadas na prática. Quando dizemos que *ainda os desabilita* é porque temos esperança de que um dia, com a diminuição do custo de processamento, assim como de espaço, tenhamos técnicas baseadas em programação dinâmica sendo utilizadas para esse fim. Um bom resumo sobre o problema de busca de seqüências biológicas em bases de dados pode ser visto no trabalho desenvolvido por Altschul e outros [5].

Lembramos aqui que o problema essencial é o de listar quais as proteínas de um conjunto (a base de seqüências) são similares àquela fornecida pelo usuário. A seqüência do usuário é chamada de **query**, enquanto que cada seqüência retornada da base é denominada **hit**.

Nós agora descrevemos brevemente um dos pacotes de programas mais populares de busca em bases de dados, chamado BLAST, de Basic Local Alignment Search Tool.

BLAST foi inicialmente proposto em 1990 por Altschul e outros [6], e desde então vem sendo aperfeiçoado [7]. Trata-se de uma heurística, e seu sucesso se deve principalmente: pela velocidade; pelo fato de devolver, como saída, uma lista de possíveis hits, acompanhados de alinhamentos; e pelo fato de cada hit vir acompanhado de uma estimativa de **significância estatística**, denominada **e-value**. Essencialmente, o e-value representa o número de hits ao acaso esperado. Assim, quanto menor o e-value, menor é a probabilidade de que um determinado hit tenha sido encontrado ao acaso.

Além do e-value, BLAST retorna, para cada hit encontrado, um alinhamento entre a seqüência query e a seqüência hit. Esse alinhamento também tem um valor, denominado **score**.

A estratégia usada por BLAST é basicamente a seguinte. Primeiro BLAST procura subcadeias curtas (tamanho 3 para proteínas) da seqüência query cujo alinhamento com subcadeias de mesmo tamanho das seqüências da base tenha valor maior ou igual a um valor de corte T, usando para isso alguma matriz de substituição de aminoácidos. As subcadeias da base e da seqüência query com essa propriedade são estendidas em ambos os lados, na tentativa de se obter segmentos com score maiores ou iguais a um outro valor de corte C. Esses segmentos são utilizados na construção do alinhamento.

BLAST então retorna todos os segmentos encontrados como acima entre a seqüência query e as seqüências da base. Obviamente todos os parâmetros utilizados atuam diretamente no resultado da busca, e podem levar a conclusões erradas se mal especificados. Isto faz com que a complexidade do BLAST seja díficil de calcular, já que todos os parâmetros acabam sendo fatores importantes na disputa entre sensibilidade e seletividade.

Apesar de originalmente BLAST usar como fonte principal de dados bases de seqüências pré-definidas, como o GenBank, no NCBI, ou o SWISS-PROT, no EMBL, é possível se instalar BLAST localmente, e ainda se criar bases de seqüências. Neste sentido, a comparação do proteoma de um genoma com o proteoma do outro genoma pode se tornar mais eficiente, já que pode-se construir uma base de seqüências para cada proteoma, como veremos no capítulo 5.

Capítulo 3

Comparação de DNA

Este capítulo trata da comparação de DNA. Entende-se por comparação de DNA qualquer comparação feita entre seqüências de nucleotídeos contendo símbolos IUB [46]. Em particular, estamos interessados em comparar seqüências genômicas de procariotos.

O grande problema na comparação de dois genomas no nível de DNA é que estamos lidando com seqüências muito grandes (da ordem de milhões de pares de bases), com possíveis ocorrências de grandes subsqüências com similaridades aproximadas. A figura 3.1, por exemplo, mostra um trecho de similaridade aproximada entre os genomas das bactérias *Xanthomonas axonopodis* pv. *citri* (`Xac`) e *Xanthomonas campestris* pv. *campestris* (`Xcc`), que têm, respectivamente, 5175554bp e 5076187bp. Nesse trecho `Xac` possui 52 genes e `Xcc` possui 50 genes, que formam 33 pares de genes ortólogos.

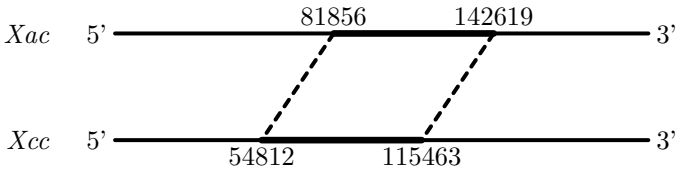

Figura 3.1: Trecho de similaridade aproximada entre os genomas de *Xanthomonas axonopodis* pv. *citri* e *Xanthomonas campestris* pv. *campestris*.

A presença de tais similaridades aproximadas sugere o uso da programação dinâmica na comparação. No entanto, seus altos custos de espaço e tempo inviabilizam sua utilização para tal fim. No caso de `Xac` e `Xcc`, por exemplo, precisaríamos de pelo menos três matrizes da ordem de 25×10^{12} células cada.

Uma alternativa para o tratamento de seqüências longas é uma estrutura de dados denomi-

nada *Árvore de Sufixos*, que é capaz de expor a estrutura interna de uma seqüência de uma maneira bem simples e direta, através da representação de seus sufixos. Em particular somos capazes de determinar muito rapidamente, a partir da árvore de sufixos de uma seqüência, as suas repetições exatas. A idéia é então, a partir das repetições encontradas na seqüência formada pela concatenação dos dois genomas, obtermos as regiões similares entre eles. O problema é que as similaridades que procuramos quase sempre não são exatas.

Nossa contribuição neste capítulo consiste numa metodologia que permite uma comparação aproximada dos genomas, a partir das repetições exatas fornecidas pela árvore de sufixos.

Na próxima seção veremos algumas noções gerais de árvores de sufixos, incluindo aspectos de implementação, que permitem seu uso na procura de repetições, que são úteis na busca de similaridades. Em seguida descrevemos as estratégias usadas na comparação de DNA, incluindo suas implementações, que resultaram no desenvolvimento de duas ferramentas, BACON e SBACON. Ao final deste capítulo mostramos os resultados obtidos e comparamos nossas ferramentas com outros trabalhos.

3.1 Árvores de sufixos

Uma árvore de sufixos armazena os sufixos de uma seqüência de símbolos. Foi inicialmente proposta para a solução de problemas de casamento de padrões, mas suas funcionalidades vão além disso, incluindo compressão de texto. Esta seção foi baseada no capítulo 6 do livro de Gusfield [32]. Uma descrição bastante detalhada, incluindo diversas aplicações de árvores de sufixos, pode ser encontrada nesse livro.

Definição 3.1 *Seja $s = s_1 s_2 \dots s_m$ uma seqüência de m símbolos de um alfabeto Σ, onde $s_m = \$$ é um símbolo especial que não ocorre em qualquer outra posição de s. A* **árvore de sufixos** *T_s para s é uma árvore enraizada, com m folhas, e no máximo $m-1$ nós internos, tal que:*

1. *as arestas de T_s são orientadas no sentido raiz→folhas e são rotuladas com subcadeias de s;*
2. *cada nó interno tem pelo menos dois filhos;*
3. *quaisquer duas arestas que saem de um mesmo nó estão rotuladas com subcadeias de prefixos diferentes;*
4. *cada folha está rotulada com um inteiro i, $1 \leq i \leq m$, que representa uma posição de s; e*

5. *a concatenação dos rótulos das arestas de um caminho da raiz até uma folha i corresponde ao sufixo de s que começa na posição i.*

A figura 3.2 mostra a árvore de sufixos para a seqüência $s =$ `xabxa$`. Note que a árvore tem $m = 6$ folhas, numeradas de 1 a 6, de tal modo que o caminho da raiz até a folha i representa o sufixo de s que começa em s_i, $i = 1, \ldots, 6$. Denotaremos esse sufixo como **sufixo i de s.**

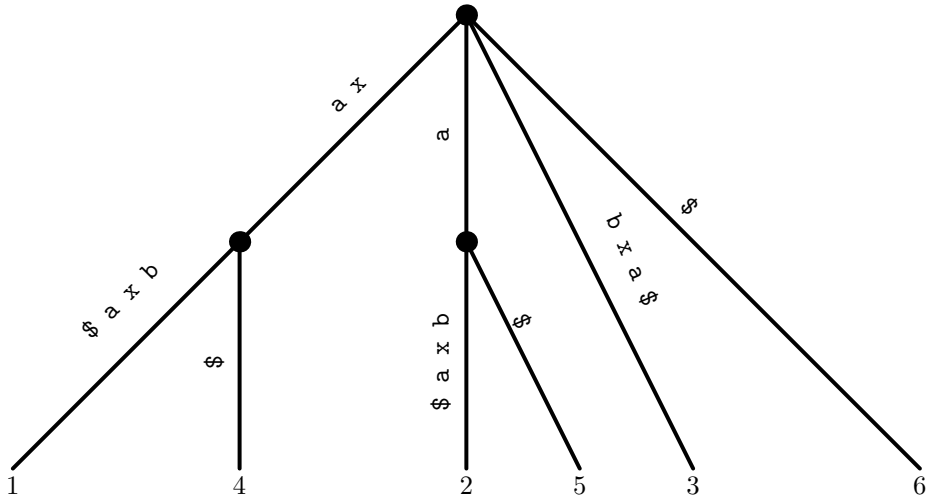

Figura 3.2: Árvore de Sufixos para a seqüência `xabxa$`

A presença do símbolo especial `$` se faz necessária pelo fato de ser impossível representar todos os sufixos de uma seqüência que tenha dois ou mais sufixos que compartilham um mesmo prefixo. A figura 3.3 mostra a árvore de sufixos para a seqüência $s =$ `xabxa`. Note que os sufixos 4 e 5 não estão explicitamente representados na árvore. Assim, para garantir que todos os sufixos de s sejam explicitamente representados, fazemos uso do símbolo especial `$`. Suporemos, a partir de agora, que s sempre possui o símbolo especial. Além disso, suporemos que o tamanho de Σ é constante, o que obviamente faz sentido para seqüências de DNA.

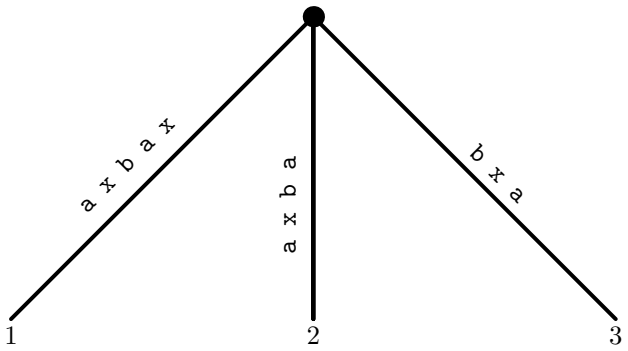

Figura 3.3: Árvore de Sufixos para a seqüência `xabxa` (sem o símbolo especial).

Weiner [72] propôs, em 1973, o primeiro algoritmo linear para a construção de árvores de sufixos. Alguns anos mais tarde McCreight [44] também propôs um algoritmo linear, mais econômico em termos de espaço. Recentemente, em 1995, Ukkonen [69] apresentou um algoritmo também linear, com as vantagens do algoritmo de McCreight, só que mais facilmente explicável.

A utilização da árvore de sufixos na procura de repetições reside num fato muito importante e óbvio: o caminho da raiz até um nó interno v da árvore representa uma repetição de s. Essa repetição acontece nas posições $i_1, \ldots i_k$, que são exatamente os rótulos das folhas da subárvore que tem v como raiz. Mais ainda, se uma cadeia α se repete em s, nas posições $i_1, \ldots i_k$, certamente teremos k sufixos, cujas folhas correspondentes são rotuladas com esses k valores, descendentes de um nó interno v comum. Ou seja, todas as repetições de s estarão representadas em sua árvore de sufixos.

Apesar de estarmos concentrados na obtenção de estruturas repetitivas numa única seqüência, nosso objetivo é o de comparar *dois* genomas, o que nos leva ao problema de encontrar essas estruturas repetitivas entre duas seqüências, digamos s e s'. No entanto, é possível obtermos tais estruturas usando apenas uma árvore de sufixos. Basta para isso construirmos a árvore de sufixos para a seqüência formada pela concatenação de s e s', tomando cuidados óbvios de acrescentar símbolos especiais no início, no final e também entre as duas seqüências.

Vamos, nas próximas seções, definir formalmente os tipos de estruturas de repetição que desejamos encontrar numa seqüência. Entre elas, estamos especialmente interessados em encontrar repetições aproximadas, que são repetições que podem conter erros de inserção, remoção e substituição.

3.2 Estruturas de repetição

Como já dissemos, estamos interessados na obtenção de repetições aproximadas que ocorrem numa seqüência. Mais especificamente, estamos interessados em obter as repetições maximais aproximadas. O interesse em repetições aproximadas é, como já dissemos, baseado no fato de que esse tipo de repetição acontece com muita freqüência em seqüências de DNA. Já as repetições maximais nos interessam porque são uma forma de não gerarmos dados redundantes, já que repetições maximais podem envolver repetições menores.

Para obtermos as repetições maximais aproximadas, vamos precisar de estruturas de repetição intermediárias. Essas estruturas, que serão detalhadas a partir de agora, são as seguintes:

- pares maximais exatos,
- repetições maximais exatas, e
- pares maximais aproximados.

Nossa principal contribuição aqui consiste na maneira como obtemos os pares e repetições maximais aproximados, a partir dos pares e repetições maximais exatos.

3.2.1 Pares maximais exatos

A grosso modo, um par maximal exato é um par de ocorrências exatas de uma subcadeia de s, tal que a inclusão de mais um símbolo à esquerda ou à direita destrói a igualdade das duas subcadeias. A seguir, definimos formalmente.

Definição 3.2 *Um* **par maximal exato** *numa seqüência s é uma tripla (p, q, α), onde α é uma subcadeia de s, com $|\alpha| < |s|$; p e q são números inteiros, com $p < q$; e tais que: $s_p \ldots s_{p+|\alpha|-1} = s_q \ldots s_{q+|\alpha|-1}$; $s_{p-1} \neq s_{q-1}$; e $s_{p+|\alpha|} \neq s_{q+|\alpha|}$. Nos casos particulares onde temos $p = 1$, $q = 1$, $p + |\alpha| - 1 = m$ ou $q + |\alpha| - 1 = m$, podemos supor a existência de símbolos especiais nos extremos de s, que não ocorrem em qualquer outra posição.*

A seqüência $s =$ `abracadabradabra`, por exemplo, tem como pares maximais exatos, entre outros, $(1, 8,$ `abra`$)$, $(1, 13,$ `abra`$)$, e $(6, 11,$ `adabra`$)$, enquanto que $(8, 13,$ `abra`$)$ não é. Note que um par maximal exato pode ser formado por repetições sobrepostas. A figura 3.4 mostra os pares $(1, 8,$ `abra`$)$ e $(1, 13,$ `abra`$)$.

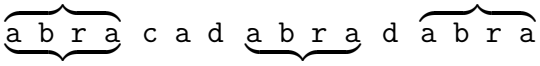

Figura 3.4: Dois pares maximais exatos da seqüência $s =$ `abracadabradabra`. O par $(1, 8,$ `abra`$)$ é mostrado na parte inferior de s, enquanto que $(1, 13,$ `abra`$)$ é mostrado na parte superior.

Vamos agora descrever um método para encontrar todos os pares maximais exatos de uma seqüência.

Como já vimos antes, pela própria construção da árvore de sufixos, o caminho da raiz até um nó interno v é rotulado por uma subcadeia α de s que ocorre exatamente nas posições $i_1, \ldots i_k$, dadas pelas folhas descendentes de v, onde $k \geq 2$, ou seja, α ocorre pelo menos duas vezes em s. Além disso, as arestas de v aos seus filhos são rotuladas por cadeias de prefixos distintos. Assim, podemos garantir que os símbolos à direita de α em todas essas k ocorrências são diferentes.

Se p e q são duas posições de ocorrência de α, que é a subcadeia que rotula o caminho da raiz até v, então p e q rotulam duas folhas da subárvore que tem v como raiz. Além disso, pelo parágrafo anterior, $s_{p+|\alpha|} \neq s_{q+|\alpha|}$. Assim, para que o par $(p, q, |\alpha|)$ seja maximal, é preciso apenas que s_{p-1} seja diferente de s_{q-1} (e isso obviamente pode não ocorrer). A idéia básica do algoritmo então consiste no fato de que se as folhas rotuladas com p e q estão na subárvore de v, e além disso $s_{p-1} \neq s_{q-1}$, então (p, q, α) é maximal.

Precisamos então manter, para cada nó interno v de T_s, informação sobre quais folhas estão na subárvore de v e, para cada tal folha rotulada com i_f, qual o símbolo s_{i_f-1}.

Usamos a seguinte estratégia para gerar e manter essas informações eficientemente. Primeiramente, na própria construção de T_s, mantemos em cada folha rotulada com i_f, o símbolo s_{i_f-1}. Para cada nó interno v, devemos construir e manter $|\Sigma|$ listas ligadas, uma para cada símbolo de Σ (incluindo os símbolos especiais do início e final de s). A lista ligada, indexada pelo símbolo x em v, denotada por L_v^x, contém todas as posições de s onde α ocorre, tendo o símbolo x a esquerda.

Tudo que precisamos saber para construirmos L_v^x são as listas $L_{v'}^x$, de todos os filhos v' de v. L_v^x nada mais é do que a união de todas as listas $L_{v'}^x$. Assim, o algoritmo usa a estratégia de busca pós-ordem, ou seja, só visita um nó de T_s após ter visitado todos os seus filhos. Quando se trata de um filho v' que é folha, não precisamos de $L_{v'}^x$, pois v' contém a informação de quem é o símbolo à esquerda do único sufixo representado por ele, o que seria equivalente a termos em v' uma lista indexada por esse símbolo contendo o sufixo dessa folha e todas as outras listas vazias.

A parte mais interessante do algoritmo está no fato de que, antes da construção de L_v^x, podemos obter todos os pares maximais exatos (p, q, α), onde α é a subcadeia de s que rotula v. Basta para isso, para cada v' filho de v e cada símbolo x, fazermos o produto cartesiano de $L_{v'}^x$ com todas as outras listas $L_{v''}^y$, onde v'' também é filho de v, $v' \neq v''$ e $y \neq x$. Cada par do tipo (p, q) desse produto cartesiano determina duas posições (p e q) de s onde ocorre α com símbolos diferentes à esquerda, ou seja, um par maximal. Após a determinação de todos os pares maximais referentes ao nó interno v, podemos construir L_v^x, para que seja posteriormente usado pelo pai de v.

De uma forma geral, o algoritmo consiste em percorrer T_s em pós-ordem e, a cada visita num nó interno v, executam-se os seguintes passos:

1. descrição dos pares maximais referentes a v, através dos produtos cartesianos, como descrito acima; e
2. união das listas $L_{v'}^x$, para cada filho v' de v e para cada símbolo x do alfabeto Σ.

A complexidade do algoritmo deve ser descrita não apenas em termos de m, tamanho de s, mas também do número de pares maximais de s, denotado por κ. Lembre-se que sempre estamos considerando o tamanho do alfabeto constante.

A construção de T_s pode ser feita em tempo $O(m)$, assim como o percurso em pós-ordem, visto que T_s tem no máximo $m - 1$ nós internos. Os produtos cartesianos são tais que no total obtemos não mais que κ pares maximais. A união das listas, conforme o passo 2 acima, pode ser feita em tempo constante, se mantivermos um apontador para o final de cada lista. Isto fará com que as listas fiquem ligadas após a união, sem que tenhamos feito cópia delas. No entanto, é fácil notarmos que, após a descrição dos pares maximais exatos de v, não mais precisamos das listas dos filhos.

Portanto, o algoritmo gasta tempo total de $O(m + \kappa)$. Vale notarmos que, caso estejamos interessados em encontrar os pares maximais com um certo tamanho mínimo m', basta percorrermos T_s até os nós cuja cadeia α seja tal que $|\alpha| \geq m'$. Esse tipo de informação pode ser facilmente armazenado em cada nó interno.

Antes de passarmos às próximas seções, precisamos fazer algumas considerações. A primeira é sobre a implementação da árvore de sufixos. Optamos por implementar o algoritmo linear de Ukkonen [69], conforme descrito em Gusfield [32], devido à sua simplicidade. Alguns cuidados com relação à economia de espaço foram tomados, mas todos eles no nível de implementação. A quantidade de espaço mais significativa é a utilizada pelos nós internos da árvore, principalmente por causa das $|\Sigma|$ listas ligadas que são mantidas nesses nós, mesmo considerando o alfabeto de tamanho constante. As folhas, por outro lado, armazenam pouca informação. O problema é que a árvore tem exatamente m folhas e pode ter no máximo

$m-1$ nós internos. Assim, dependendo do número de repetições da seqüência, podemos ter muitos nós internos. E é o que acontece quando tratamos de seqüências de DNA.

As economias de espaço e de tempo são as principais vantagens da utilização de árvores de sufixos. Obviamente a implementação das listas ligadas dos nós internos aumenta consideravelmente a quantidade de espaço utilizado. Por outro lado, tais listas fazem com que consigamos eficientemente obter os pares maximais, principal objeto do nosso algoritmo para encontrar as repetições aproximadas.

A principal desvantagem na utilização de árvores de sufixos está no fato de que, a partir delas, somos capazes de encontrar apenas repetições exatas. Nossa tentativa, então, é a de usar repetições exatas extraídas da árvore na determinação de repetições aproximadas.

3.2.2 Repetições maximais exatas

Vamos agora descrever o que são as repetições maximais exatas e como obtê-las, a partir dos pares maximais exatos.

Definição 3.3 *Uma* **repetição maximal exata** *de uma seqüência s é uma subcadeia α de s que ocorre em algum par maximal exato.*

Novamente recorremos à árvore de sufixos de s para obtermos as repetições maximais de s, da seguinte forma: uma vez que, na execução do algoritmo de pares exatos maximais, estamos num nó interno v tal que existem pelo menos duas listas L_v^x e L_v^y não-vazias, com $x \neq y$, podemos afirmar que pelo menos dois sufixos p e q, representados por folhas da subárvore de v, são tais que $x = s_{p-1} \neq s_{q-1} = y$ e que, portanto, a seqüência α que rotula o caminho da raiz até v participa do par maximal (p, q, α). Ou seja, α é uma repetição maximal exata. Mais ainda, o conjunto de posições dadas pela união de todas as listas ligadas L_v^x, para todo x em Σ, forma o conjunto de posições onde ocorre a repetição.

A listagem das repetições maximais exatas relacionadas a um nó interno pode assim ser obtida diretamente durante a visita no algoritmo de obtenção dos pares maximais exatos, com o custo de $O(|\Sigma|)$ por nó. Ou seja, em tempo $O(m)$ podemos obter todas as repetições maximais de uma seqüência.

Como uma repetição maximal exata pode ocorrer em vários pares maximais, é de se supor que o número de repetições maximais exatas seja menor que o número de pares maximais exatos de uma seqüência. A despeito desse fato, e do fato de podermos considerar o conjunto de repetições maximais exatas como sendo tão representativo quanto o conjunto de pares maximais exatos, por uma questão estratégica, que ficará clara mais tarde, não vamos utilizar

as repetições maximais exatas na obtenção das repetições maximais aproximadas, mas sim os pares maximais aproximados, que serão descritos a seguir.

3.2.3 Pares e repetições maximais aproximados

Usando o algoritmo da seção 3.2.1, somos capazes de obter todos os pares maximais exatos (p, q, α) de uma seqüência s, onde $|\alpha| \geq k$. Basta para isso, como já dissemos anteriormente, executarmos as visitas até nós internos v tais que a seqüência que rotula o caminho da raiz até v tenha tamanho não menor que k.

Nós agora vamos propor um método para obtermos os pares maximais aproximados a partir dos pares maximais exatos. O método basicamente consiste na junção de dois pares maximais cujas respectivas ocorrências estejam próximas entre si, e representa nossa principal contribuição no uso de árvores de sufixos na procura de repetições aproximadas.

A justificativa dessa abordagem se dá pelo fato de podermos enxergar um par aproximado como sendo um par exato ou aproximado com a inclusão de algumas diferenças. A figura 3.5 ilustra essa situação.

$$\cdots \quad \underline{\texttt{TA}}\texttt{CG}\underline{\texttt{TAG}} \quad \cdots \quad \underline{\texttt{TA}}\texttt{GTC}\underline{\texttt{TAG}} \quad \cdots$$

Figura 3.5: Um par aproximado pode ser visto como a junção de dois pares exatos (sublinhados).

As definições de junção e par maximal aproximado são dadas agora.

Sejam (p, q, α) e (p', q', α') pares maximais exatos, $l_\alpha = |\alpha|$ e $l_{\alpha'} = |\alpha'|$. Seja ainda $d_{pp'}$ o número de símbolos entre o final da ocorrência de α em p e o início da ocorrência de α' em p', ou seja, a distância entre a primeira ocorrência de α e a primeira ocorrência de α'. Analogamente, seja $d_{qq'}$ o número de símbolos entre o final da ocorrência de α em q e o início da ocorrência de α' em q'. Veja a figura 3.6.

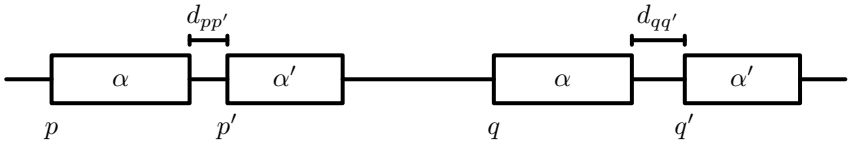

Figura 3.6: Dois pares maximais exatos, (p, q, α) e (p', q', α')

Definição 3.4 *Sejam os pares maximais exatos* (p, q, α) *e* (p', q', α') *como acima. Sejam ainda* t_d *e* t_l *dois números reais tais que* $0 \leq t_d, t_l \leq 1$. *Dizemos que* (p, q, α) *e* (p', q', α') *são* **pares maximais exatos próximos** *se pelo menos uma das condições abaixo é satisfeita.*

1. $\max(d_{pp'}, d_{qq'}) \leq t_d \cdot \min(l_\alpha, l_{\alpha'})$;
2. $\min(l_\alpha, l_{\alpha'}) \leq t_l \cdot \max(l_\alpha, l_{\alpha'})$ *e*
   $\max(d_{pp'}, d_{qq'}) \leq t_d \cdot \max(l_\alpha, l_{\alpha'})$.

O parâmetro t_d serve para regularmos a relação entre o tamanho dos espaços entre os pares exatos e o tamanho de suas ocorrências, enquanto que t_l serve para regularmos a relação entre os tamanhos da menor e da maior ocorrência dos pares maximais exatos.

Intuitivamente, a primeira condição acima checa se os tamanhos dos espaços entre os pares não são muito grandes, comparados aos tamanhos das próprias cadeias envolvendo os pares maximais exatos. Caso a primeira condição não seja verdadeira, então a segunda condição diz que uma das subcadeias deve ser significativamente menor que a outra e que, além disso, os espaços não devem ser muito grandes, comparados às cadeias, mas desta vez usamos a maior delas como referência.

Essencialmente o objetivo aqui é o de não permitir que o espaço entre as ocorrências dos pares não seja significativamente maior do que uma fração do tamanho das componentes dos pares. Assim, estamos tentando evitar a criação de repetições aproximadas que não tenham significado biológico.

Definição 3.5 *Sejam dois pares maximais exatos próximos* (p, q, α) *e* (p', q', α'). *A operação de* **junção-simples** *consiste na eliminação dos dois pares maximais exatos e na criação de um novo par, denominado* **par maximal aproximado**, (p, q, β_1, β_2), *onde* $\beta_1(\beta_2)$ *é a concatenação da ocorrência de* α *em* $p(q)$, *do espaço entre a ocorrência de* α *em* $p(q)$ *e a ocorrência de* α' *em* $p'(q')$, *e da ocorrência de* α' *em* $p'(q')$. *As cadeias* β_1 *e* β_2 *são chamadas de* **repetições maximais aproximadas**.

Note que, apesar da maximalidade ter se mantido entre β_1 e β_2, já que os símbolos à esquerda(direita) de β_1 e β_2 são diferentes, β_1 e β_2 são certamente diferentes. Se ambas fossem iguais, formariam elas próprias uma par maximal exato. Note também a mudança de notação: um par maximal aproximado deve carregar informação sobre as duas subcadeias β_1 e β_2 de s, que podem inclusive ter tamanhos diferentes. Especificamente, $|\beta_1| = l_\alpha + d_{pp'} + l_{\alpha'}$ e $|\beta_2| = l_\alpha + d_{qq'} + l_{\alpha'}$.

Na prática, estamos permitindo que um par maximal, que antes era exato, agora contenha erros de inserção, remoção e substituição.

Nós agora vamos generalizar a definição de pares maximais próximos, no sentido de permitir a junção de quaisquer pares maximais próximos. Note que a definição de par maximal aproximado é tal que um par exato pode ser visto como um par aproximado (p, q, β_1, β_2), com $\beta_1 = \beta_2$. Assim, quando nos referirmos daqui em diante a um par maximal (p, q, β_1, β_2), significa que este pode ser exato ou aproximado.

Definição 3.6 *Sejam os pares maximais* (p, q, β_p, β_q) *e* $(p', q', \beta_{p'}, \beta_{q'})$. *Seja* $d_{pp'}$ *o número de símbolos entre o final da ocorrência de* β_p *e o início da ocorrência de* $\beta_{p'}$. *Analogamente, seja* $d_{qq'}$ *o número de símbolos entre o final da ocorrência de* β_q *e o início da ocorrência de* β'_q. *Dizemos que* (p, q, β_p, β_q) *e* $(p', q', \beta_{p'}, \beta_{q'})$ *são* **pares maximais próximos** *se pelo menos uma das condições* 1 *e* 2 *abaixo é satisfeita.*

1. $\max(d_{pp'}, d_{qq'}) \leq t_d \cdot \min(l_{\beta_p}, l_{\beta_{p'}})$; *e*
   $\max(d_{pp'}, d_{qq'}) \leq t_d \cdot \min(l_{\beta_q}, l_{\beta_{q'}})$.

2. $\min(l_{\beta_p}, l_{\beta_{p'}}) \leq t_l \cdot \max(l_{\beta_p}, l_{\beta_{p'}})$; *e*
   $\min(l_{\beta_q}, l_{\beta_{q'}}) \leq t_l \cdot \max(l_{\beta_q}, l_{\beta_{q'}})$; *e*
   $\max(d_{pp'}, d_{qq'}) \leq t_d \cdot \max(l_{\beta_p}, l_{\beta_{p'}})$; *e*
   $\max(d_{pp'}, d_{qq'}) \leq t_d \cdot \max(l_{\beta_q}, l_{\beta_{q'}})$.

Ainda visando o objetivo de possibilitar a junção de quaisquer pares maximais, vamos agora generalizar a definição de junção-simples.

Definição 3.7 *Sejam dois pares maximais próximos* (p, q, β_p, β_q) *e* $(p', q', \beta_{p'}, \beta_{q'})$. *A operação de* **junção** *consiste na criação de um novo par* (p, q, β_1, β_2), *onde* $\beta_1(\beta_2)$ *é a concatenação da ocorrência de* $\beta_p(\beta_q)$, *do espaço entre* β_p *e* $\beta_{p'}(\beta_q$ *e* $\beta_{q'})$ *e de* $\beta_{p'}(\beta_{q'})$.

Considere então a lista $\mathcal{L}_e$ dos pares maximais exatos, obtida através do método da seção 3.2.1. O algoritmo abaixo descreve como obter a lista $\mathcal{L}_a$ dos pares maximais aproximados e exatos de s.

```
ordene L_e pela primeira ocorrência da cadeia α
L_a ← L_e
repita
  para cada P_i de L_a faça
    para cada P_j de L_a à direita de P_i faça
        se P_i e P_j são próximos então
```

$P \leftarrow$ junção(P_i, P_j)
$\mathcal{L}_a \leftarrow \mathcal{L}_a \cup \{P\}$
até que não haja mais junções

O algoritmo acima percorre a lista $\mathcal{L}_a$ quantas vezes forem necesárias, até garantir que não haja mais junções a fazer. Além disso, permite a junção de quaisquer tipos de pares (exato e exato, exato e aproximado, aproximado e exato, aproximado e aproximado).

Daqui em diante vamos nos referir aos pares obtidos através do algoritmo acima como pares maximais aproximados.

Repetições maximais aproximadas

Vamos agora descrever um método para a obtenção, a partir dos pares maximais aproximados, das repetições maximais aproximadas.

Note que uma repetição maximal aproximada pode ocorrer em mais de um par maximal aproximado. Quando isso acontece, precisamos manter informações sobre os pares maximais exatos que foram juntados, formando os pares maximais aproximados. Isto é feito da seguinte forma. Quando criamos um par maximal exato, atribuímos a ele um código, que identifica de forma única a repetição maximal exata a qual ele pertence. Assim, quando uma junção é feita, o código do novo par maximal aproximado é dado pela lista ordenada dos códigos de seus componentes. Dessa forma, pares maximais aproximados que possuem mesmo código estão envolvendo repetições aproximadas de mesma subcadeia.

A figura 3.7 mostra um exemplo de como o código é atribuído. Suponha que A_1, A_2 e A_3 sejam repetições exatas da cadeia α, com código c_A, e que B_1, B_2 e B_3 sejam repetições exatas da cadeia β, com código c_B. Suponha ainda os pares maximais exatos (abusando um pouco da notação) (A_1, A_2), (A_1, A_3), (B_1, B_2) e (B_1, B_3). Os códigos atribuídos aos pares maximais exatos foram os seguintes: c_A a (A_1, A_2) e (A_1, A_3); e c_B a (B_1, B_2) e (B_1, B_3). Quando juntamos (A_1, A_2) e (B_1, B_2), formamos um par maximal aproximado com código (c_A, c_B), que consiste na lista ordenada dos códigos dos pares juntados. Da mesma forma, quando juntamos (A_1, A_3) e (B_1, B_3), temos um par maximal aproximado com código (c_A, c_B). Como os dois pares maximais aproximados têm o mesmo código, ambos irão pertencer à mesma repetição maximal aproximada. Basta então tomarmos o cuidado de separarmos pares maximais aproximados que possuem a mesma lista ordenada de códigos.

Figura 3.7: Obtenção de repetições maximais aproximadas.

Complexidade

A obtenção de um par maximal aproximado tem exatamente o custo de uma operação de junção, que é constante, visto que apenas engloba um número constante de testes. O custo do algoritmo de obtenção dos pares maximais aproximados, portanto, é de $O(\kappa)$, onde κ é o número de pares maximais exatos maiores ou iguais a k da seqüência s, já que podemos no máximo fazer $\kappa - 1$ junções.

Uma vez que mantemos para cada par maximal aproximado o seu código, que consiste na lista ordenada dos códigos das repetições nas quais seus componentes estão, é barato obtermos as repetições maximais aproximadas: basta ordenarmos e percorrermos a lista de pares maximais aproximados e criarmos, a cada novo código encontrado, as posicões de repetição aproximada. Esta operação custa no máximo $O(\kappa \log \kappa)$.

A seguir, apresentamos as ferramentas de comparação de genomas, no nível de DNA, baseadas na fundamentação descrita nesta seção.

3.3 BACON

Apresentamos agora uma de nossas ferramentas de comparação de dois genomas. O programa, que compara no nível de DNA, se chama BACON, de *Bacterial comparator*, e tem como base a utilização de árvores de sufixos, assim como a metodologia envolvendo pares e repetições maximais aproximados, proposta na seção anterior.

A entrada para o BACON são as seqüências de DNA dos dois genomas, no formato FASTA. O usuário tem várias opções de parâmetros. São eles: o tamanho mínimo k de uma repetição exata (valor padrão = 100); os valores de t_d e t_l, de acordo com a definição 3.4, com valores padrão iguais a 0.5; e a opção de ordenar as repetições por coordenada ou por tamanho (valor padrão: por coordenada).

Como saída, BACON reporta os seguintes resultados:

- repetições maximais exatas e aproximadas que ocorrem nos dois genomas;
- repetições tandem exatas;
- diferenças singulares; e
- blocos de inserção e remoção.

Ao mesmo tempo em que descrevemos o programa, mostramos como cada uma das funcionalidades acima pode ser obtida.

Sejam g e h as seqüências dos genomas G e H, respectivamente. BACON inicialmente constrói a árvore de sufixos para a seqüência $\$_0 g \$_1 h \$_2 h^{CR} \$_3$, onde os símbolos especiais $\$_0$, $\$_1$ e $\$_2$ não pertencem ao alfabeto de DNA. A seqüência é construída dessa forma para garantirmos que repetições que começam no sufixo de uma das cadeias e que terminam no prefixo da outra não sejam considerados. Além disso, precisamos considerar o reverso-complemento de h, já que algumas repetições de g podem ocorrer também lá. Assim, obviamente é preferível escolhermos o menor dos genomas para fazer o papel de h.

A concatenação acima reflete o fato de estarmos interessados principalmente nas repetições que envolvam pares de ocorrências em g e h ou em g e h^{CR}, já que estamos comparando os genomas G e H. As repetições envolvendo apenas ocorrências em g e g^{CR} são tratadas pelo programa SBACON, que compara um genoma com ele mesmo.

Há outros tipos de repetições que não consideramos, por entendermos que não são interessantes do ponto de vista biológico, como por exemplo uma das repetições consideradas por Kurtz e Schleiermacher em [37, 38], onde tem-se o par na mesma fita, porém com ocorrências invertidas.

A quantidade de espaço utilizado na construção da árvore é um tópico que merece alguns comentários. Como dissemos anteriormente, as folhas da árvore tendem a ocupar bem menos espaço do que os nós internos. No nosso caso específico, a implementação testada numa máquina Compaq 64-bit (alpha) DS-20 foi tal que cada nó interno ocupa cerca de 290 bytes, enquanto que uma folha ocupa 32 bytes. É possível economizar ainda mais. No entanto, nossa preocupação maior foi a de ganhar velocidade e não espaço. Esses valores, especialmente para nós internos, valem apenas para a construção da árvore. A partir do momento em que os pares maximais exatos são construídos, mais espaço é necessário para cada nó interno, para abrigar as listas ligadas resultantes das informações passadas pelos filhos.

Repetições maximais exatas

Uma vez que a árvore foi construída, BACON lista as repetições maximais exatas, de acordo com o que foi descrito na seção 3.2.2. A figura 3.8 mostra um trecho do arquivo de saída de BACON para os genomas de *Mycoplasma genitalium G37* e *Mycoplasma pneumoniae M129*, com respectivamente 580074bp e 816394bp, com algumas repetições maximais exatas. Antes porém, BACON mostra algumas informações referentes aso genomas comparados, como nome do genoma, tamanho, quais os parâmetros usados, etc. BACON também identifica as repetições maximais exatas que ocorrem em apenas um genoma.

```
>Exact repeats
Size Qty Start    End(ge) Start     End(ge) ...
 69   2 170013 170081(1 )  87488  87420(2R)
 71   2 170090 170160(1 )  87411  87341(2R)
 63   2 170202 170264(1 )  87299  87237(2R)
158   2 170273 170430(1 )  87228  87071(2R)
155   2 170501 170655(1 )  87000  86846(2R)
187   2 170659 170845(1 )  86842  86656(2R)
```

Figura 3.8: Algumas repetições maximais exatas encontradas na comparação de *Mycoplasma genitalium* e *Mycoplasma pneumoniae*. As colunas mostram, respectivamente, o tamanho da repetição, a quantidade de vezes que a repetição acontece. Em seguida, para cada repetição, as coordenadas são mostradas. Entre parênteses aparece o genoma no qual aquela ocorrência da repetição acontece (`R` representa o complemento reverso do genoma). Os valores usados para os parâmetros foram $k = 50$, $t_d = 0.5$ e $t_l = 0.5$.

Repetições tandem exatas

Dizemos que uma cadeia de símbolos w, não-vazia, é **múltipla** se, e somente se, existe outra cadeia z tal que $w = z^P$, onde z^P é a concatenação de $P \geq 2$ ocorrências da cadeia z.

Uma **repetição tandem exata** é uma cadeia do tipo $w^n w_{(m)}$, onde:

- w é uma cadeia não-vazia, não-múltipla, de tamanho u;
- w^n é a concatenação de $n \geq 2$ ocorrências de w; e
- $w_{(m)}$ é o prefixo de tamanho m de w, com $0 \leq m < u$.

Se (i, j, α) é um par maximal exato da seqüência s tal que $j \leq i + l_\alpha$, então $w = s_i \dots s_{j-1}$, de tamanho $u = j - i$, ocorre consecutivamente $1 + \frac{l_\alpha}{j-i}$ vezes, a partir da posição i de s. Se

$j = i + l_\alpha$, então temos uma repetição tandem começando em i, com $n = 2$ e $m = 0$. Se $j < i + l_\alpha$, então temos uma repetição tandem começando em i, com $n = \lfloor \frac{l_\alpha}{j-i} \rfloor$ e $m = j - i$.

Assim, repetições tandem com $u \geq k$ são encontradas como um subproduto de BACON, onde k é o limite mínimo para repetições maximais exatas.

A figura 3.9 mostra as repetições tandem encontradas na comparação das *Mycoplasmas*. Mais importante que encontrar repetições tandem exatas é encontrar repetições tandem aproximadas. No entanto, BACON não foi desenvolvido para esse propósito. Na verdade, as repetições tandem encontradas são subprodutos de BACON pelo fato de serem repetições maximais exatas com características especiais. Uma ferramenta específica para encontrar repetições tandem aproximadas pode ser vista no trabalho feito por Benson [11].

```
>Exact tandem repeats
Block size  #of_repetitions First pos.(ge)
        60          2.0333    390793 (1 )
        42           2.381     27215 (2 )
        42          2.3333     40623 (2 )
        12          5.8333     93812 (2 )
        72          2.5556    469188 (2 )
        12          16.333    611996 (2 )
        12          15.833    681320 (2 )
```

Figura 3.9: Repetições tandem exatas encontradas por BACON na comparação das *Mycoplasmas*, com $k = 50$, $t_d = t_l = 0.5$. Da esquerda para a direita, as colunas mostram o tamanho do bloco de repetição, o número n de repetições do bloco, a posição da primeira ocorrência do bloco e finalmente o genoma onde ocorreu.

Diferenças singulares

Sejam w e z duas seqüências de DNA. Sejam ainda aXb e aYb subcadeias de w e z, respectivamente, com a e b cadeias e X e Y bases ou espaços. Uma **diferença singular (DS)** entre w e z ocorre quando $X \neq Y$.

Note que uma DS pode representar uma substituição, uma remoção ou uma inserção. Para encontrarmos as DSs nós simplesmente verificamos se os pares a serem juntados, (p, q, α) e (p', q', α'), com buracos de tamanho $d_{pp'}$ e $d_{qq'}$, respectivamente, são tais que $d_{pp'} \leq 1$ e $d_{qq'} \leq 1$. Se ambos são iguais a 1, então temos uma substituição; se um deles é zero, então temos uma inserção/remoção.

Novamente, BACON não tem como objetivo principal a determinação das DSs entre os dois

genomas. No entanto, as DSs encontradas são um sub-produto de BACON e somente são vistas quando os pares maximais exatos que as envolvem são de tamanho maior ou igual a k. Uma pequena parte das DSs entre as *Xanthomonas* é mostrada na figura 3.10.

```
in [174527..174624](1 ) vs. [157526..157623](2 ): 174583 vs. 157582
in [187762..206800](1 ) vs. [175566..198894](2 ): 196204 vs. 187520, 202762 vs. 195210, 204901 vs. 197249
in [541361..541525](1 ) vs. [534230..534393](2 ): 541420 vs. 534289
in [592199..592261](1 ) vs. [589158..589220](2 ): 592230 vs. 589189
in [164031..165380](1 ) vs. [465817..467166](2 ): null  vs. 466179, 165300  vs. null
```

Figura 3.10: Algumas DSs encontradas por BACON na comparação das *Xanthomonas*, com $k = 30$ e $t_d = t_l = 0.5$. Cada linha representa um par maximal e suas respectivas DSs. A primeira linha, por exemplo, diz que houve uma substituição entre as posições 174583 de `Xac` e 157582 de `Xcc`. A última linha diz que houve uma inserção na posição 466179 de `Xac` e uma remoção na posição 165300 de `Xcc`.

Repetições maximais aproximadas

As repetições maximais aproximadas são encontradas por BACON seguindo exatamente a estratégia descrita na seção 3.2.3, ou seja, a partir dos pares maximais exatos. Assim, algumas repetições maximais passaram de exatas para aproximadas, enquanto que outras continuaram exatas por não terem seus pares juntados. BACON então lista, como parte de sua saída, todas as repetições maximais encontradas, sejam elas exatas ou aproximadas.

A figura 3.11 mostra um trecho dessa parte da saída, com algumas repetições encontradas por BACON na comparação das *Xanthomonas*. A maneira de mostrar essas repetições é parecida com aquela usada quando BACON mostra as repetições maximais exatas, com exceção de que agora as exatas e aproximadas são identificadas pela letra no início da linha, e o número de vezes em que o ocorre a repetição é mostrado.

```
A       2        345    44974(1 )      345    45447(2 )
E       2        492      526(1 )      492      526(2 )
E       2        621      682(1 )      621      682(2 )
E       2        795      859(1 )      795      859(2 )
E       2        879      913(1 )      879      913(2 )
E       3     141694   141728(1 )   143759   143793(1 )   112055   112089(2 )
```

Figura 3.11: Algumas repetições maximais da comparação das *Xanthomonas*.

Blocos de inserção e remoção

A última parte da saída de BACON mostra os chamados blocos de inserção e remoção. Esses blocos com tamanho não inferior a k, também chamados de **blocos únicos**, consistem em regiões que não aparecem em qualquer repetição.

A motivação para encontrarmos os blocos únicos está no fato de que essas regiões podem conter potenciais genes que não acontecem no outro genoma.

3.4 SBACON

SBACON é uma outra ferramenta desenvolvida por nós, que segue exatamente a mesma linha de BACON, só que dessa vez o objetivo é comparar um genoma contra ele mesmo, na tentativa de encontrar as repetições maximais do genoma.

Nesse caso, a seqüência de entrada para a árvore de sufixos é $\$_0 g \$_1 g^{CR} \$_2$, e todas as propriedades relacionadas às estruturas de repetição vistas anteriormente valem também aqui.

As funcionalidades de SBACON são praticamente as mesmas de BACON, com exceção de que SBACON não reporta as DSs, nem os blocos de inserção e remoção.

3.5 Resultados

Nesta seção mostramos alguns dos resultados obtidos por BACON sobre alguns genomas disponíveis. Os testes foram feitos numa máquina Compaq DS-20 com 4 Gigabytes de memória RAM.

A comparação entre `Xac` e `Xcc` levou cerca de 2 minutos na máquina citada acima. Os valores dos parâmetros foram: $k = 30$, $t_d = t_l = 0.5$. A figura 3.12 mostra as maiores repetições maximais exatas encontradas. Apenas para ilustrar, a maior delas, de tamanho 1051, ocorre quatro vezes nos dois genomas e contém ocorrências de rRNA.

```
Size Qty   Start     End(ge)   Start     End(ge) ...
1051   4 4580326 4581376(1 )  5069568 5070618(1 )  4561566 4562616(2 )  4949434 4950484(2 )
 621   4 4577421 4578041(1 )  5066663 5067283(1 )  4558659 4559279(2 )  4946527 4947147(2 )
 611   4 4576767 4577377(1 )  5066009 5066619(1 )  4558005 4558615(2 )  4945873 4946483(2 )
 564   4 4578740 4579303(1 )  5067982 5068545(1 )  4559980 4560543(2 )  4947848 4948411(2 )
 362   4 4578043 4578404(1 )  5067285 5067646(1 )  4559281 4559642(2 )  4947149 4947510(2 )
 312   2  933992  934303(1 )   879519  879830(2 )
 282   4 4580043 4580324(1 )  5069285 5069566(1 )  4561283 4561564(2 )  4949151 4949432(2 )
 259   2 1120750 1121008(1 )  1041290 1041548(2 )
 249   2 1138309 1138557(1 )  1058771 1059019(2 )
 234   4 4581378 4581611(1 )  5070620 5070853(1 )  4562618 4562851(2 )  4950486 4950719(2 )
```

Figura 3.12: Maiores repetições maximais exatas encontradas na comparação de `Xac` e `Xcc`.

Informações sobre as maiores repetições maximais aproximadas entre `Xac` e `Xcc` podem ser vistas na tabela 3.1. A tabela traz ainda o número de pares de genes ortólogos [1] que ocorrem em cada uma das regiões de similaridade aproximada.

Início em `Xac`	Final em `Xac`	Início em `Xcc`	Final em `Xcc`	DSs	Genes em `Xac`	Genes em `Xcc`	Pares de ortólogos
2784710	3369074	2288355	3430012	31	474	922	349
881111	1117195	830078	1055066	35	267	200	165
1634078	1771443	1595638	1736670	27	111	112	86
3921990	4001647	3766054	3860842	7	63	74	56
3922284	3994371	3766348	3843885	5	58	62	51
81971	142619	54927	115463	3	51	50	31

Tabela 3.1: Maiores repetições maximais aproximadas encontradas na comparação de `Xac` e `Xcc`.

A figura 3.13 mostra as três maiores repetições maximais exatas encontradas em `Xac`. A terceira repetição tem três ocorrências. Cada uma delas contém uma proteína predita, todas anotadas como "ISxac1 transposase".

[1] Os pares de ortólogos foram determinados segundo a metodologia descrita no capítulo 5.

```
Size Qty   Begin End      Begin     End
5598   2 4576389 4581986  5065631 5071228
1871   2 3101674 3103544  3356994 3355124 C
1371   3  109237 110607    179298 180668     5116841 5118211
```

Figura 3.13: Maiores repetições maximais exatas encontradas em `Xac`. O símbolo `C` representa a fita complemento-reversa do genoma.

Comparamos também duas cepas de *H. pylori*, com 1667867bp e 1643831bp e rotuladas como h_1 e h_2, respectivamente. A tabela 3.2 mostra as 15 maiores repetições maximais exatas detectadas por BACON. A tabela 3.3 mostra as repetições tandem, enquanto que a tabela 3.4 mostra algumas das DSs encontradas. Os parâmetros usados neste caso foram $k = 100, t_d = t_l = 0.5$. O tempo total de execução foi de aproximadamente 45 segundos.

tamanho	# de occorrências	localização(genoma)	
4051	2	1057087–1061137(2)	1426925–1430975(2)
2114	2	1377709–1379822(2)	233913–231800(2R)
2114	2	231800–233913(2)	1379822–1377709(2R)
1890	2	1060309–1062198(1)	1156709–1158598(1)
1693	2	1188333–1190025(2)	1463351–1465043(2)
1621	2	1060579–1062199(1)	1613256–1614876(1)
1620	2	1156979–1158598(1)	1613256–1614875(1)
1620	2	454237–455856(1)	1050959–1052578(1)
1108	2	499022–500129(1)	987082–988189(1)
960	2	727448–728407(2)	963735–964694(2)
929	2	426501–427429(1)	1069592–1070520(1)
890	2	912794–913683(2)	1296919–1297808(2)
886	2	1063912–1064797(2)	1436343–1437228(2)
879	3	44699–447869 (1)	1428628–1427750(2R)
			1058790–1057912(2R)
808	3	1511644–1512451(1)	1188539–1189346(2)
			1463557–1464364(2)

Tabela 3.2: As 15 maiores repetições maximais exatas encontradas por BACON na comparação das cepas de *H. pylori*. As repetições são reportadas mesmo quando ocorrem no mesmo genoma. Aqui apenas as duas últimas são em genomas diferentes.

u	n	início (genoma)
7	25.857	1 (1)
138	2.4203	130053 (1)
390	2.0692	559014 (1)
21	10.143	696789 (1)
57	3.2105	1263291 (1)
15	8.5333	5153 (2)
228	2.2588	520719 (2)
21	7.1905	659300 (2)
15	8.7333	787255 (2)
12	11.833	1061291 (2)
12	13.5	1431133 (2)
612	2.0065	1562960 (2)

Tabela 3.3: Repetições tandem exatas encontradas por BACON nas duas cepas de *H. pylori*. u é o tamanho do bloco de repetição; n é o número de ocorrências do bloco.

Localização(genome)		DSs	
264989–265214(1)	261873–262098(2)	265113	261997
314321–314591(1)	311730–312000(2)	314434	311843
330960–333076(1)	946684–948800(1)	331203	946927
331523–332725(1)	1317500–1318702(1)	null	1317645
377508–377813(1)	1124932–1124627(2R)	377703	1124737

Tabela 3.4: Uma pequena amostra das DSs encontrados nas cepas de *H. pylori*. Note que na quarta linha vemos uma iserção de h_2 com relação a h_1.

Na comparação das *Mycoplasmas*, a maior repetição maximal exata obtida por BACON foi de 281 bases. A maior repetição maximal aproximada foi de 4281 bases. Os valores para os parâmetros foram $k = 50$, $t_d = t_l = 0.5$. O programa, neste caso, demorou cerca de 20 segundos.

Executamos o programa `cross_match` [29], devido a P.Green, que implementa eficientemente o algoritmo de Smith-Waterman [61] na comparação das *Mycoplasmas*, usando o parâmetro `min_score=100`, e obtivemos 42.7% de *M. genitalium* casados com 30.4% de *M. pneumonias*. Esses números são bem mais expressivos do que os alcançados por BACON (cerca de 1% de cada genoma). Isso acontece porque muitas substituições ocorrem no nível de DNA, em

espaços muito curtos, o que certamente faz com que apareçam muitas repetições maximais exatas de tamanho pequeno (menor que k), não capturadas pelo nosso programa.

O uso de `cross_match` na comparação de genomas grandes, por outro lado, é inviável. Sua execução na comparação das duas *Xylellas*, por exemplo, demora cerca de 12 horas. Supondo a existência de memória suficiente, considerando o tamanho das *Xylellas* (aproximadamente 2.5×10^6bp e 2.7×10^6bp, respectivamente) e levando em conta que `cross_match` tem complexidade quadrática, podemos estimar o tempo de mais de 48 horas na comparação das *Xanthomonas*.

3.6 Discussão

No trabalho desenvolvido por Delcher e outros [19], o programa denominado MUMMER foi proposto como ferramenta de comparação de dois genomas. MUMMER constrói um alinhamento dos dois genomas, e também (assim como BACON) é baseado em árvore de sufixos. O programa usa o conceito de *maximal unique match* (MUM) (daí o seu nome).

Um MUM é uma seqüência que ocorre exatamente uma vez em cada genoma, de tamanho maior ou igual a um parâmetro fornecido pelo usuário, semelhante ao k do BACON. Um MUM pode ser facilmente encontrado olhando num nó interno da árvore de sufixos (construída para os dois genomas) tal que esse nó interno tem apenas dois fihos que são folhas e que representam posições em genomas diferentes.

MUMMER junta MUMs próximos (menos que 5000 bases de distância), independente do tamanho dos MUMs, desde que pertençam a uma subseqüência mais longa de MUMs. Neste sentido, MUMMER é bem mais restritivo do que BACON, já que não admite repetições múltiplas. Quando MUMMER junta dois MUMs, aplica o algoritmo de Smith-Waterman [61] para alinhar os buracos entre eles.

Além das diferenças cruciais entre BACON e MUMMER citadas acima, BACON tem a vantagem de reportar repetições múltiplas (que ocorrem mais que duas vezes), ao invés de reportar apenas os MUMs. Isso é possível porque BACON faz uso de vértices internos mais gerais na árvore. Além disso, BACON tem alguns subprodutos que MUMMER não tem, como por exemplo blocos únicos, repetições tandem e diferenças singulares.

O objetivo de MUMMER é apenas o de alinhar os dois genomas. Neste caso, as ocorrências múltiplas podem atrapalhar um alinhamento. O uso dos MUMs, assim, faz bastante sentido.

Com relação ao SBACON, podemos compará-lo com o programa denominado REPUTER, devido a Kurtz e outros [37, 38]. Ambos são similares em funcionalidade. O programa `Miropeats` [52], desenvolvido por J.Parsons, também mostra regiões de repetição, só que

graficamente, no formato `ps`.

Outros métodos para a comparação de seqüências genômicas podem ser vistos no trabalho de Karlin [36], onde se procura fazer uma análise puramente estatística da presença de polinucleotídeos diversos, ou ainda nos trabalhos de Leung e outros [42], e Vincens e outros [70], onde se usa idéia similar àquela usada no programa BLAST, descrito na seção 2.3, ou seja, onde se busca palavras pequenas de tamanho fixo, e depois tenta-se estendê-las, para a obtenção de regiões com similaridade. Na mesma linha do BLAST, temos o trabalho recente de Webb Miller e outros [75], denominado MEGABLAST, que serve para comparar dois grandes conjuntos de seqüências um contra o outro. Tais seqüências são esperadas serem bastante similares, provavelmente contendo erros de seqüenciamento.

Capítulo 4

Programação dinâmica e comparação de proteínas

Este capítulo trata da utilização da programação dinâmica na comparação de proteínas. Estamos aqui nos referindo à comparação de seqüências cujos símbolos pertencem ao alfabeto de aminoácidos.

A motivação principal para o desenvolvimento de algoritmos e técnicas de comparação de proteínas é a detecção de homologia entre genes. A elucidação de tais homologias consiste num dos principais ramos da Biologia Molecular Computacional, já que é um caminho rápido e seguro na determinação da função de genes descobertos.

O alto grau de similaridade entre proteínas é um forte indício de que são homólogas. No entanto, quando se trata de se descobrir homologia entre proteínas com seqüências com pouca similaridade, o problema se torna mais complicado. Essas homologias menos aparentes, que formam o que é chamada de *twilight zone* (Doolittle [20]) costumam ser diluídas em ruídos capazes de confundir determinadas técnicas de comparação [20, 71].

A contribuição deste capítulo, publicada em [4], consiste num algoritmo de programação dinâmica, adaptado dos já conhecidos, e que utiliza o conceito de *regiões curingas*. A definição formal de região curinga é vista mais adiante, mas intuitivamente podemos dizer que regiões curingas são regiões nas quais nós não estamos preocupados com a qualidade do alinhamento, por serem regiões supostamente menos conservadas. A idéia então é a de pontuarmos tais regiões de maneira mais relaxada, em detrimento de pontuarmos mais rigidamente regiões mais conservadas, que tendem a abrigar estruturas secundárias e onde acontecem menos substituições, inserções e remoções. Esse algoritmo foi implementado e testado, mas os resultados ainda não são satisfatórios, indicando que refinamentos são necessários.

O capítulo tem a seguinte organização: na seção 4.1 apresentamos nosso algoritmo de programação dinâmica com curingas. Na seção seguinte apresentamos algumas aplicações e os resultados obtidos. Ao final fazemos alguns comentários relevantes.

4.1 Programação dinâmica com curingas

Nesta seção apresentamos um algoritmo, inicialmente proposto por nós em [4], para alinhar duas seqüências, baseado no método de programação dinâmica, usando função afim para penalizar buracos, com a possibilidade adicional da existência de uma segunda função σ' de pontuação das colunas. Para tanto, vamos agora fazer algumas formalizações necessárias.

Seja um alinhamento conforme a definição 2.1, com o acréscimo de uma função alternativa σ' para a pontuação de colunas. Seja $\mathcal{S}$ o conjunto de colunas do alinhamento pontuadas com a função σ e $\mathcal{S}'$ o conjunto de colunas pontuadas com a função σ'. Assim, o valor do alinhamento passa a ser dado por

$$\sum_{i \in \mathcal{S}} \sigma(s'_i, t'_i) + \sum_{i \in \mathcal{S}'} \sigma'(s'_i, t'_i).$$

Uma coluna i pertencente ao conjunto $\mathcal{S}'$ é chamada de **coluna curinga**. Uma **região curinga** é um conjunto contíguo de colunas curingas.

A idéia deste tipo de alinhamento, como dissemos, é podermos pontuar de forma menos rígida as regiões que apresentam pouca conservação e, conseqüentemente, apresentam menos substituições, inserções e remoções. Essas regiões são as regiões curingas.

Motivação

Um importante passo na determinação do relacionamento entre duas proteínas é, como vimos, o alinhamento de suas seqüências de aminoácidos. Quando as proteínas apresentam alta similaridade os alinhamentos costumam ser bons, diferenciando de forma clara as regiões conservadas das não-conservadas. No entanto, o problema fica bem mais difícil quando elas são homólogas, mas a similaridade é baixa. Para esses casos, os alinhamentos baseados somente na similaridade das seqüências costumam ter baixos valores, além de serem diferentes daqueles obtidos com o uso de informação estrutural, ou seja, informação sobre estruturas secundárias.

Os alinhamentos baseados em informações estruturais mostram que regiões conservadas contêm relativamente poucas diferenças, enquanto que as não-conservadas apresentam muitas diferenças. Um bom algoritmo deveria então usar este conhecimento e aplicar o conjunto de

penalidades usual somente àquelas regiões supostamente conservadas; regiões não-conservadas, uma vez detectadas, deveriam ser excluídas do alinhamento. É isto que nosso algoritmo faz, atribuindo penalidades diferenciadas a regiões menos conservadas.

O algoritmo

O algoritmo é baseado em programação dinâmica e usa funções afins de penalização de buracos que podem operar em dois modos. Num modo aplica as penalizações usuais para iniciar um buraco (h) e estender um buraco (g). Assim, um buraco com k espaços custa $h + kg$. No outro modo o algoritmo usa: um parâmetro de inicialização de região curinga (h'); um parâmetro de extensão de região curinga com casamentos e substituições (g'); e um parâmetro de extensão de região curinga com buraco ($\bar{g}$). Dessa forma, uma região curinga com k_1 casamentos/substituições e k_2 espaços tem valor total $h' + k_1 g' + k_2 \bar{g}$. A troca entre um modo e outro é feita automaticamente pelas recorrências da programação dinâmica, como veremos.

A entrada do algoritmo consiste: nas seqüências das proteínas s e t, $|s| = m$ e $|t| = n$; numa matriz de comparação M, que fornece o valor do emparelhamento de qualquer par de aminoácidos; e valores para os parâmetros de penalização h, g, h', g', e $\bar{g}$. A saída é um alinhamento de valor máximo. A figura 4.1 mostra um exemplo de saída do algoritmo.

```
GA-----PVPVDENDEGLQALQFATAEYNRASNKYSSRVVQ-ISANRQLVSGIKY-----IL
| *******|   || |     **** | |        *******||    |  |
GEWEIIDIGPFTQNDGGFADEENKTGQYGRLTFVVIRPCMQKIYENRREIKGYEYQLYAKLF
```

Figura 4.1: Um alinhamento com 3 regiões curingas. Cada coluna numa região curinga é denotada pelo símbolo `*`, e o símbolo `|` denota um casamento. Note que pode haver casamentos dentro de uma região curinga.

A computação do valor do alinhamento, bem como sua construção, é feita da mesma forma que é feita pela programação dinâmica padrão. Nós usamos seis matrizes A, B, C, D, E, e F. A posição (i, j) de cada matriz contém o valor de um alinhamento máximo do prefixo de s de tamanho i com o prefixo de t de tamanho j, de tal forma que:

- A: s_i casa com t_j
- B: um espaço em s casa com t_j
- C: s_i casa com um espaço em t
- D: s_i casa com t_j numa região curinga
- E: um espaço em s casa com t_j numa região curinga
- F: s_i casa com um espaço em t numa região curinga

Considerando que $0 \leq i \leq m$, $0 \leq j \leq n$, e que $M(s_i, t_j)$ é o valor do emparelhamento de s_i com t_j, as fórmulas de recorrência da programação dinâmica são dadas como segue:

$$A_{i,j} = M(s_i, t_j) + \max \left\{ \begin{array}{ll} A_{i-1,j-1}, & B_{i-1,j-1}, \\ C_{i-1,j-1}, & D_{i-1,j-1}, \\ E_{i-1,j-1}, & F_{i-1,j-1} \end{array} \right\}, \quad i,j > 0$$

$$B_{i,j} = \max \left\{ \begin{array}{ll} A_{i,j-1} - (h+g), & B_{i,j-1} - g, \\ C_{i,j-1} - (h+g), & D_{i,j-1} - (h+g), \\ E_{i,j-1} - (h+g), & F_{i,j-1} - (h+g) \end{array} \right\}, \quad \begin{array}{c} j > 0 \\ (i,j) \neq (0,1) \end{array}$$

$$C_{i,j} = \max \left\{ \begin{array}{ll} A_{i-1,j} - (h+g), & B_{i-1,j} - (h+g), \\ C_{i-1,j} - g, & D_{i-1,j} - (h+g), \\ E_{i-1,j} - (h+g), & F_{i-1,j} - (h+g) \end{array} \right\}, \quad i,j > 0$$

$$D_{i,j} = \max \left\{ \begin{array}{ll} A_{i-1,j-1} - (h'+g'), & B_{i-1,j-1} - (h'+g'), \\ C_{i-1,j-1} - (h'+g'), & D_{i-1,j-1} - g', \\ E_{i-1,j-1} - g', & F_{i-1,j-1} - g' \end{array} \right\}, \begin{array}{c} i,j > 0 \\ (i,j) \neq (1,1) \end{array}$$

$$E_{i,j} = \max \left\{ \begin{array}{ll} A_{i,j-1} - (h'+\bar{g}), & B_{i,j-1} - (h'+\bar{g}), \\ C_{i,j-1} - (h'+\bar{g}), & D_{i,j-1} - \bar{g}, \\ E_{i,j-1} - \bar{g}, & F_{i,j-1} - \bar{g} \end{array} \right\}, \quad \begin{array}{c} j > 0 \\ (i,j) \neq (0,1) \end{array}$$

$$F_{i,j} = \max \left\{ \begin{array}{ll} A_{i-1,j} - (h'+\bar{g}), & B_{i-1,j} - (h'+\bar{g}), \\ C_{i-1,j} - (h'+\bar{g}), & D_{i-1,j} - \bar{g}, \\ E_{i-1,j} - \bar{g}, & F_{i-1,j} - \bar{g} \end{array} \right\}, \quad i,j > 0.$$

Essas recorrências são extensões diretas da programação dinâmica usual. Como um exemplo, nós explicamos as recorrências para a matriz F. Neste caso a última coluna do alinhamento tem s_i e um espaço numa região curinga. As posições de F consideradas são aquelas que contêm o melhor alinhamento de $s_1 \ldots s_{i-1}$ and $t_1 \ldots t_j$. Precisamos verificar se o espaço é o início de um buraco numa região curinga ou apenas uma extensão. Se for uma extensão de uma região curinga, então devemos penalizar com $\bar{g}$. Caso seja o início de um buraco numa região curinga, devemos penalizar com $h' + \bar{g}$.

As bases das recorrências dependem de como queremos penalizar buracos iniciais e finais. Abaixo apresentamos a inicialização para o caso local-global, onde os símbolos de s não são

penalizados quando emparelhados com espaços nas pontas de t.

$$
\begin{aligned}
&A_{0,0} = B_{0,0} = C_{0,0} = D_{0,0} = E_{0,0} = F_{0,0} = 0,\\
&A_{i,0} = B_{i,0} = D_{i,0} = E_{i,0} = F_{i,0} = -\infty \text{ e } C_{i,0} = 0, \quad \text{for } i = 1, \ldots, m,\\
&A_{0,j} = C_{0,j} = D_{0,j} = F_{0,j} = -\infty, \quad \text{para } j = 1, \ldots, n,\\
&B_{0,1} = -(h + g),\\
&E_{0,1} = -(h' + \bar{g}),\\
&D_{1,1} = -(h' + g').
\end{aligned}
$$

Depois de computarmos todas as posições de todas as matrizes, o valor final do alinhamento depende do tipo de alinhamento escolhido (global, local, ou global-local). No caso local-global o valor desejado é dado pelo máximo entre os valores contidos nas últimas colunas de todas as seis matrizes.

A complexidade do algoritmo é $O(mn)$, pois o cálculo do valor final do alinhamento é feito pelo preenchimento das seis matrizes, cada uma com $(m+1)(n+1)$ posições, e cada posição requer tempo constante para o seu preenchimento. A determinação de um alinhamento ótimo se dá pela varredura das seis matrizes, em tempo $O(m + n)$. O espaço usado é também $O(mn)$.

4.2 Aplicações e resultados

Nesta seção apresentamos algumas aplicações do algoritmo de programação dinâmica com curingas, e mostramos alguns resultados obtidos.

4.2.1 Aplicação: alinhamentos locais

Nós apresentamos agora uma aplicação bem simples do algoritmo: a obtenção de alinhamentos locais de duas seqüências s e t, as quais sabidamente contêm um número de subseqüências similares, separadas por subseqüências não-similares. O resultado esperado de um bom algoritmo para este problema é um alinhamento no qual as subseqüências similares sejam de fato alinhadas (formando bons alinhamentos locais). Este é um problema que pode ser usado para compararmos nosso algoritmo com o algoritmo de programação dinâmica usual, para sabermos se as regiões curingas de fato ajudam na obtenção de melhores alinhamentos.

Nós executamos esta comparação usando pares de seqüências geradas aleatoriamente por computador, que têm as seguintes propriedades:

1. o número de subseqüências similares (entre 1 e 5; mesmo número entre s e t);

2. os tamanhos mínimo e máximo dessas seqüências;
3. o percentual de mutações (inserções, remoções e substituições) nelas; e
4. os tamanhos mínimo e máximo das subcadeias entre as subseqüências similares (o número total dessas subcadeias é o número das similares mais um).

Para compararmos com nosso algoritmo, implementamos também o algoritmo de programação dinâmica usual, usando função afim para a penalização de buracos.

A figura 4.2 mostra o resultado para uma instância particular do problema. Há duas subseqüências similares a serem alinhadas, e cada par difere em 2 aminoácidos (10% de mutação). O primeiro alinhamento mostrado é resultante da aplicação do nosso algoritmo, usando a matriz dada por Gonnet e outros [28], enquanto que o segundo mostra o alinhamento das mesmas subseqüências dado pelo algoritmo usual de programação dinâmica.

```
               1111111111111 1111111                  2222222
HRYDNKH-GDRI---RHDSKNPMVEEYN-AWQLCAN-----MHPEKTDDVGDILAIKYEGE
***************||| ||||||||| |||||||*****************||| |||
--GHWMYGSMKRGLARHDHKNPMVEEYNMAWQLCANWKDQIEPTGVAFGIFNFHAIKAEGE
               11111111111111111111                  2222222

2222222 222222
PYMHSRI-VTKECIEAWMEKACIDYQISRCS
||||||| |||||| ||    |*********
PYMHSRIFVTKECI-AWDVREC-----KMLL
22222222222222
```

```
                   1111111111111 1111111                   2
HRYDNKHGDRI---------RHDSKNPMVEEYN-AWQLCANMHPEKTDDVG------DILA
       |            ||| ||||||||| |||||||         |         |
-------GHWMYGSMKRGLARHDHKNPMVEEYNMAWQLCANWK-DQIEPTGVAFGIFNFHA
                   11111111111111111111                   2

2222222222222222222
IKYEGEPYMHSRIVTKECIEAWMEKACIDYQISRCS---
|| ||||||||||           | ||      |
IKAEGEPYMHSRI--------FVTKECIAWDVRECKMLL
222222222222        2222222
```

Figura 4.2: Dois alinhamentos das mesmas seqüências. O primeiro mostra o resultado obtido pela programação dinâmica com curingas, enquanto que o segundo mostra a saída da programação dinâmica usual com $h = 3.3$ e $g = 0.26$ (de acordo com Fischer e outros em [24], estas são as melhores escolhas para os parâmetros). As subseqüências rotuladas com números são as que devem ser alinhadas.

Nosso algoritmo obteve um bom alinhamento para ambas as subseqüências, enquanto que a programação dinâmica usual não conseguiu alinhar bem a segunda. Isto pode ser medido pelo número de espaços inseridos nas subseqüências que deveriam ser alinhadas. No entanto, a figura 4.3 mostra outra instância do problema na qual obtivemos resultado oposto.

```
              1111111111111111111              2222222222222
KTPHRMSQWQNTKAPILCEAIFALDYSKRISYQS---IDQFTRYGNVVIWNFYASLLFMGD
**************|||*****|||||||||||**************|||||| ||||||
GVCPTYQEDEVEAYPILMCEAIALDYSKRISYQSWLMVNFAPLTMFQNIWNFYAQLLFMGD
              1111111111111111111              2222222222222
2222222
YTNKPQMMFFCSVCVNGHVSQ
|*******************
Y---NTNKPQMPEEGPSHYVN
2   2222222
```

```
                 1111111111111111111                22222222
KTPHRMSQWQNTKA---PILCEAIFALDYSKRISYQS-----IDQFTRYGNVVIWNFYASL
          |       |||     ||||||||||||         |   |  |||||| |
---GVCPTYQEDEVEAYPILMCEAIALDYSKRISYQSWLMVNFAPLTMFQN--IWNFYAQL
                 1111111111111111111                22222222

222222 222222
LFMGDY-TNKPQMMFFCSVCVNGHVSQ
|||||| ||||||          |
LFMGDYNTNKPQM----PEEGPSHYVN
2222222222222
```

Figura 4.3: Dois alinhamentos de outra instância do problema, com os mesmos valores para os parâmetros.

Nós executamos o mesmo experimento para várias instâncias do problema, considerando diferentes conjuntos de valores para os parâmetros. Uma análise dos resultados é vista mais adiante, na seção 4.2.2.

4.2.2 Aplicação: identificação da estrutura correta de uma proteína

Nesta seção nós descrevemos o que acreditamos ser a principal aplicação do algoritmo de regiões curingas, o problema de identificar qual a estrutura correta de uma proteína. Neste problema a entrada é um conjunto S de seqüências de proteínas com estrutura tri-dimensional não conhecida e uma base de dados T de seqüências de proteínas cuja estrutura é conhecida. A saída desejada aqui é, para cada seqüência s em S, uma lista de pares (s, t), onde $t \in T$, em ordem não-crescente de similaridade estrutural entre s e todas as seqüências de T. Os elementos de T são chamados de *folds*, enquanto que os elementos de S são chamados de *probes*. Os trabalhos de Abagyan e Batalov [1] e de Richards [55] servem para se ter uma boa visão do problema.

Para o problema ser interessante, a similaridade entre um probe e seu fold correto deve ser baixa. Algoritmos para esse problema que usam apenas comparação das seqüências devem então ser aptos a distinguir entre dois tipos de baixa similaridade: aquelas que resultam do alinhamento correto entre duas regiões conservadas e aquelas que resultam de regiões não conservadas. Nosso objetivo é exatamente alcançar essa distinção.

Benchmark

Um benchmark bastante conhecido para este problema é dado por Fischer e outros [24], que nós descrevemos brevemente. Neste benchmark, $|S| = 68$ e $|T| = 301$. As soluções fornecidas estão na forma de um conjunto P de pares do tipo (probe,fold). O conjunto P é obtido por comparações estruturais. Os conjuntos S e T foram cuidadosamente construídos seguindo vários critérios, tais como o de que todos os pares em P sejam tais que as seqüências do par tenham aproximadamente 30% de identidade ou menos.

Um algoritmo aplicado a este benchmark deve ter como saída, para cada probe, uma lista de todos os folds em ordem decrescente de similaridade estrutural, de acordo com o algoritmo aplicado. O desempenho do algoritmo é medido pelo valor de R, dado por

$$R = \frac{\sum_{i=1}^{|S|} \frac{1}{r_i}}{|S|},$$

onde r_i denota a posição na lista onde o fold correto do probe i foi colocado pelo algoritmo. Assim, um algoritmo perfeito teria como resultado $R = 1$. Note que $0 \leq R \leq 1$. Os autores do benchmark testaram vários algoritmos segundo esta abordagem.

Dados experimentais

Para testarmos nosso algoritmo com o benchmark descrito, vários aspectos tiveram que ser especificados. Primeiramente optamos por usar a versão local-global para os alinhamentos, visto que, em geral, os folds são mais curtos que os probes, e nós não queremos penalizar os espaços nas pontas dos probes.

Outro aspecto importante é a maneira na qual a lista de folds é computada. Como os valores dos alinhamentos usando algoritmo local ou global em geral dependem do tamanho das seqüências, métodos usando esses algoritmos requerem algum tipo de normalização (dividindo o valor do alinhamento pelo logarítmo do tamanho do fold, supostamente menor, por exemplo). Por outro lado, os resultados de [24] mostraram que no caso local-global os melhores resultados foram obtidos sem normalização. Assim, decidimos não usar qualquer normalização em nossos experimentos.

A matriz de comparação de aminoácidos utilizada foi a descrita por Gonnet e outros em [28], mas com cada elemento dividido por 4. Esta divisão foi feita em [24], e nós fizemos o mesmo para podermos comparar os resultados.

Valores para os parâmetros de penalização de buracos e regiões curingas foram obtidos pelo método de força-bruta. A busca foi feita num intervalo de possíveis valores para cada parâmetro, e testada para cada possível combinação. Os melhores valores obtidos foram: $h = 2.4$, $g = 0.15$, $h' = 2.6$, $g' = 0.18$ e $\bar{g} = 0.2$.

Resultados

Ao aplicarmos nosso algoritmo ao benchmark obtivemos $R = 0.68$, com 60.3% de hits corretos (probe com fold correto).

O benchmark tem um índice de dificuldade para cada um dos 68 probes, baseados nos testes com vários algoritmos. Este índice é a média da posição do probe determinada pelos algoritmos. Isto é, um probe com índice de dificuldade 1.0 é considerado muito fácil, porque todos os algoritmos conseguiram determinar seu correto fold. O índice mais difícil é igual a 20.0. Entre os 68 probes, 12 são os considerados muito fáceis, com índice 1.0, enquanto que 7 têm índice 20.0. Nosso algoritmo obteve o fold correto para todos esses 12, além de outros 29 probes com índice de dificuldade maior do que um.

A tabela 4.1 resume nossos resultados, dando as posições e os respectivos números e percentuais de probes. A tabela 4.2 mostra os resultados obtidos com a programação dinâmica usual. O valor de R, neste caso, foi de 0.66, o que nos confere um ganho de aproximadamente 3%.

Posições	# probes	%
1	41	60.3
2–5	11	16.2
>5	16	23.5

Tabela 4.1: Posições alcançadas pelo nosso algoritmo. A primeira coluna mostra uma classificação de posições nas quais o fold correto foi colocado na lista de folds. Por exemplo, a segunda linha diz que 11 probes tiveram seus folds corretos colocados entre as posições 2 e 5.

Posições	# probes	%
1	40	58.8
2–5	10	14.7
>5	18	26.5

Tabela 4.2: Posições obtidas pela programação dinâmica usual.

Esses resultados mostram que nosso algoritmo teve um desempenho pouco melhor do que o algoritmo usual de programação dinâmica, o que não nos surpreende, vistos os resultados da seção 4.2.1. Experimentos com muitos outros valores para os parâmetros foram feitos, levando a resultados similares, o que reforça a necessidade de refinamentos ao algoritmo.

4.3 Discussão

Dadas duas proteínas homólogas, o problema de determinar o alinhamento estrutural mais bem ajustado (onde as estruturas se alinham corretamente) entre elas é diferente do problema de encontrar a melhor estrutura de uma proteína [23, 24, 31, 53]. Lesk e outros [41] propuseram panalizações variáveis juntamente com o uso de estrutura secundária para resolver este problema. Eles usaram 3 penalidades distintas: uma para posições dentro de estrutura, uma fora de estrutura, e uma nas pontas de estrutura. Com isso, foram capazes de obter bons alinhamentos entre uma nova proteína, com nenhum conhecimento sobre a sua estrutura, e outra com estrutura secundária conhecida.

Nosso algoritmo é facilmente adaptável a este tipo de situação, mas de uma forma mais geral. Basta aplicarmos pesos a posições específicas de estruturas secundárias. Tal abordagem, se aplicada com sucesso, poderia ser utilizada como uma pista para a predição de estruturas secundárias, como hélices α e fitas β, dada a informação estrutural de alguma proteína relacionada a ela.

Outras aplicações não exploradas são: a de usar a mesma idéia de regiões curingas em alinhamentos múltiplos; e a de adaptar nosso algoritmo para resolver o problema de casamento aproximado de padrões, onde o padrão pode conter um número variável de símbolos curingas, como visto no trabalho de Akutsu [2].

Capítulo 5

Comparação de proteomas

Comparações de genomas no nível de DNA, tais como feitas por BACON, apenas mostram regiões que se repetem nos genomas, sem levarem em conta aspectos funcionais das regiões similares detectadas. Além disso, tais comparações são muito úteis apenas quando os genomas são evolutivamente próximos.

Precisamos então de uma estratégia de comparação de dois genomas que nos permita identificar regiões comuns em termos dos genes que elas contêm. Mais ainda, estamos interessados numa ferramenta que ajude a explicar como a reordenação e o reagrupamento dos genes influenciam nas diferenças entre as funcionalidades dos genomas (Sankoff e Nadeau [57]). Técnicas para comparação de proteomas são, portanto, necessárias para esse fim.

Nossa contribuição neste capítulo consiste numa metodologia para a comparação de dois proteomas, além de uma implementação da metodologia, o programa EGG.

O capítulo está organizado da seguinte forma. Na seção 5.1 descrevemos a metodologia proposta. Na seção 5.2 descrevemos o programa EGG, que implementa a metodologia vista aqui. Ao final, nas seções 5.3 e 5.4, mostramos alguns resultados e fazemos alguns comentários a respeito de outros trabalhos.

5.1 Metodologia

Vamos agora descrever o que consideramos uma de nossas principais contribuições, uma metodologia para comparação de dois proteomas.

Apesar de alguns conceitos vistos aqui já terem sido dados na seção 2.1, vamos refazê-los, para que sejam melhor interpretados no decorrer das próximas seções.

- Um **gene**, para efeitos deste capítulo (objetivando simplificar a exposição), é uma seqüência traduzida para aminoácidos de uma ORF predita como pertencendo a um gene.
- A **fita** de um gene é a fita de DNA a qual ele pertence; é representada por um dos sinais `+` ou `-`.
- As **coordenadas de um gene** são as posições, na fita `+`, do seu início e fim. Assim, um gene da fita `-` tem como coordenada de início a posição correspondente, na fita `+`, à sua posição final; e tem como posição final a posicão correspondente, na fita `+`, à sua posição inicial.
- O **tamanho** de um gene g, denotado por $|g|$, é o seu número de aminoácidos menos um (o stop codon).
- O **proteoma** é o conjunto de genes de um genoma. Para facilitar a exposição, suporemos que um **genoma** é o mesmo que um replicon.
- A **ordem dos genes** de um proteoma é dada pela ordem não-decrescente das coordenadas de início dos genes.
- Dois genes g_i e g_j de um mesmo genoma G são **parálogos** se são descendentes de um mesmo gene ancestral (através de um evento de duplicação).
- Uma **região de genes consecutivos (RGC)** é um conjunto de genes consecutivos num proteoma, de acordo com suas coordenadas de início, independente da fita. Note que o próprio proteoma é uma RGC.

Para os conceitos descritos a seguir, considere a comparação entre os proteomas dos genomas G e H, e os genes g, g' de G e h, h' de H.

- Dois genes g e h são **ortólogos** se são descendentes de um mesmo gene ancestral. Dizemos, neste caso, que (g, h) é um **par de ortólogos**.
- Um gene g é **específico** em relação a H se não existir gene h em H tal que g e h são ortólogos.
- Uma **região específica (RE)** de um genoma G em relação a H é uma região de G particularmente rica em genes específicos. Tais regiões podem ocorrer por transferência lateral ou perda. A definição do que é particularmente rica em genes específicos será vista mais adiante.

- Uma **região ortóloga (RO)** é um par (α, β) tal que:
  - α é uma RGC em G;
  - β é uma RGC em H;
  - α e β são descendentes de uma mesma região ancestral; e
  - α e β contêm aproximadamente o mesmo número de genes.
- Dois pares de ortólogos (g, h) e (g', h') formam um **cruzamento** quando a ordem de g e g' em G e a ordem de h e h' em H são invertidas.
- A **espinha dorsal de duas RGCs** α de G e β de H é uma seqüência de pares de ortólogos do tipo (g, h), tal que:
  - cada gene de α tem no máximo um gene ortólogo a ele em β, e vice-versa; e
  - não existe cruzamentos entre os pares da seqüência.

A figura 5.1 mostra um exemplo abstrato de RO. Note que pode haver genes da RO que participam de mais de um par de ortólogos na própria RO, além de cruzamentos. A figura 5.2 mostra a espinha dorsal de duas RGCs.

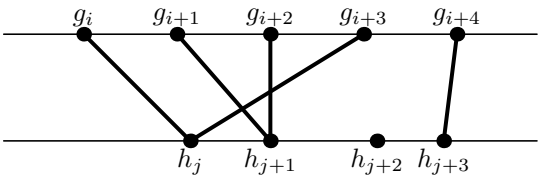

Figura 5.1: Exemplo de uma RO. As arestas representam ortologia.

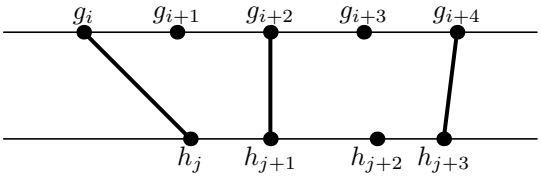

Figura 5.2: Exemplo de uma espinha dorsal de duas RGCs.

Antes de descrevermos nossa metodologia, vamos listar quais os objetivos queremos alcançar através da comparação de dois proteomas:

1. encontrar genes específicos de um proteoma em relação ao outro;
2. encontrar regiões específicas de um proteoma em relação ao outro;
3. encontrar pares de genes ortólogos;
4. encontrar regiões ortólogas;
5. determinar a espinha dorsal dos proteomas; e
6. determinar as famílias de genes parálogos de um proteoma.

Nossa metodologia consiste de passos e/ou critérios que devem ser seguidos para cada um dos objetivos citados acima, descritos nas seções subseqüentes.

5.1.1 Genes ortólogos e específicos

Para encontrarmos os pares de genes ortólogos, vamos supor duas abordagens, relacionadas a duas perguntas que queremos responder.

A primeira abordagem corresponde à seguinte pergunta: existe em H algum gene h ortólogo ao gene g de G? Neste caso queremos determinar uma ortologia "unidirecional". Para este fim, propomos o seguinte critério:

> O gene h é **ortólogo** ao gene g se, e somente se,
>
> - a medida de significância estatística $s(g, h)$ da similaridade entre g e h seja menor ou igual a um limite fixo S; e
> - dado o alinhamento de g e h, pelo menos $P\%$ de $|g|$ e $P\%$ de $|h|$ sejam cobertos pelo alinhamento.

Dizemos que g e h são **ortólogos** se, e somente se, g é ortólogo a h e vice-versa.

A medida de significância citada acima será posteriormente implementada através do e-value dado pelo BLAST (seção 2.3). Assim, quanto mais significante é a pontuação dada pelo alinhamento de g e h, menor é o e-value. Por isso queremos que a medida seja menor ou igual a S.

A noção de cobertura pelo alinhamento dada acima pode ser formalizada como segue. Sejam g_I e g_J respectivamente o primeiro e o último símbolos de g que apareceram no alinhamento[1]. Dizemos que $P\%$ de $|g|$ são **cobertos** pelo alinhamento se, e somente se,

$$J - I + 1 \geq \frac{P}{100} \cdot |g|.$$

[1] Como o alinhamento pode ser local, g_I e g_J podem ser tais que $g_I > 1$ e $g_J < |g|$.

A outra abordagem objetiva o alinhamento dos proteomas. Neste caso suporemos que farão parte do alinhamento somente genes ortólogos, de acordo com o critério acima e, além disso, devem ser tais que h é o gene de H que tem mais similaridade com g, dentre todos os genes de H ortólogos a g, e vice-versa. O objetivo deste critério é o de minimizar a interferência de genes parálogos. Neste caso dizemos que g e h são **fortemente ortólogos**.

Para encontrar genes específicos, nossa metodologia sugere o seguinte critério:

> O gene g de G é **específico** com relação ao genoma H se, e somente se, a medida de significância $s(g, h)$ da similaridade entre eles é maior que um limite fixo S', para todo gene h de H, tal que $S' > S$.

Um algoritmo que compara dois genes e que fornece a similaridade e a significância estatística entre eles é suficiente para a implementação dessa parte da metodologia. Vamos ver como isso pode ser feito na seção 5.2.

Vale lembrar que os critérios aqui propostos são "aproximações operacionais" de conceitos biológicos. São aproximações porque estamos usando similaridade para inferir ancestralidade comum. São operacionais porque estamos propondo critérios que nos permitem escrever programas que automatizam a detecção dessa ortologia aproximada.

5.1.2 Regiões específicas (REs)

A metodologia que propomos para encontrar uma RE é baseada no problema computacional conhecido como *subcadeia de máxima soma*. Esse problema tem como entrada uma seqüência L de inteiros e como saída a subcadeia de L cuja soma dos elementos é máxima, entre todas as subcadeias de L.

A estratégia consiste na atribuição de valores para cada um dos genes do proteoma, digamos ε para um gene que não é específico e ϵ para genes específicos, tais que $\epsilon > \varepsilon$. A seqüência de entrada para o problema da subcadeia de máxima soma é a então a seqüência dos valores atribuídos aos genes. Dessa forma o algoritmo deve encontrar todas as subcadeias cuja soma dos valores é maior ou igual a um valor limite.

5.1.3 Regiões ortólogas (ROs)

Para que possamos descrever o método que propomos para encontrar as regiões ortólogas, precisamos do conceito de *run*. Seja α uma RGC de G formada pelos genes $g_i, \ldots g_k$ e β uma RGC de H formada pelos genes $h_j, \ldots h_l$, tais que $k - i + 1 = l - j + 1$, $k > i$, e $l > j$. Dizemos que α e β formam um **run** se uma das seguintes seqüências de pares de genes ortólogos acontece:

- $(g_i, h_j), (g_{i+1}, h_{j+1}), \ldots, (g_k, h_l)$; ou
- $(g_i, h_l), (g_{i+1}, h_{l-1}), \ldots, (g_k, h_j)$.

Quando a primeira opção ocorre, dizemos que o run é **paralelo**. Quando a outra acontece, o run é **anti-paralelo**. Um run é dito ser **consistente** se, no caso de ser paralelo, todos os pares são tais que os dois genes participantes de cada par têm a mesma fita; e no caso de ser anti-paralelo, os dois genes de cada par têm fitas opostas. As duas desigualdades ($k > i$, e $l > j$) nos dizem que um run tem que ter pelo menos 2 pares de ortólogos.

A estratégia para a determinação dos runs é baseada na varredura dos pares de ortólogos contíguos. Como veremos na seção 5.2, isto pode ser feito com o uso de uma matriz binária, que indica a ortologia para cada par de genes dos proteomas.

Nosso critério para a determinação de uma região ortóloga consiste basicamente na junção de runs próximos. Mais formalmente:

Uma **região ortóloga** R é:

- ou um run isolado com pelo menos M pares de ortólogos;
- ou a união de runs (de quaisquer tamanhos) com um total de pelo menos M pares de ortólogos, e cuja distância entre os genes extremos dos runs (aqueles mais próximos do outro run) não seja maior que um certo valor fixo K, em número de genes.

A justificativa para este tipo de estratégia está no fato de que a existência desses pequenos "buracos", formados por genes não casados na região, reflitam a existência de inserções, remoções ou substituições, passíveis de uma região ortóloga.

A justificativa para o número mínimo M de pares se baseia no fato de que runs isolados com apenas dois pares de ortólogos podem ser fruto do acaso, segundo Tamames e outros [64, 65]. Para confirmar esse fato, executamos alguns testes com proteomas verdadeiros, porém com um deles tendo seus genes rearranjados aleatóriamente. Runs isolados com apenas dois pares de genes ortólogos continuaram aparecendo. No entanto, runs com 3 ou mais pares (aparentes na comparação dos proteomas originais) deixaram de aparecer. A simulação sugere que com $M = 3$ a probabilidade é grande que um run com pelo menos 3 pares de ortólogos não é fruto do acaso.

Na descrição da implementação da metodologia, na seção 5.2, a noção de distância vista no critério descrito acima será melhor formalizada.

A estratégia para a obtenção das ROs também é simples. Basta percorrermos todos os runs, da esquerda para a direita (segundo a ordem dadas pelos genes de um dos proteomas), e juntarmos cada vez que são próximos, de acordo com o segundo critério da definição de RO acima. Durante a descrição da implementação veremos com isso é feito.

5.1.4 Espinha dorsal dos proteomas

A idéia aqui é obtermos um alinhamento global dos proteomas. Para tanto, vamos fazer uso da definição de "ortologia forte" vista anteriormente. Ou seja, vamos exigir que um par de genes (g, h) seja candidato a se alinhar somente se g e h forem fortemente ortólogos. Estamos tentando minimizar a interferência de parálogos, que obviamente pode embaralhar o alinhamento.

O objetivo mais específico é o de obter o maior alinhamento possível (com maior número de pares de genes fortemente ortólogos) sem que haja cruzamentos. É natural imaginarmos que, quanto mais próximos filogeneticamente forem os genomas, maior a espinha dorsal.

Note que a espinha dorsal de duas RGCs não é necessariamente única. Assim, nossa tarefa é a de determinar a espinha dorsal que mais consiga evidenciar o quanto os proteomas são próximos, considerando um alinhamento.

A estratégia que propomos para a construção da espinha dorsal é baseada no conhecido problema computacional chamado de *subseqüência comum mais longa* [14]. Neste caso, cada símbolo corresponde ao número seqüencial do gene no proteoma e dois símbolos das cadeias são iguais se, e somente se, os respectivos genes são fortemente ortólogos.

Vale notarmos aqui a diferença entre a abordagem que usamos para o alinhamento global dos proteomas e aquela que usamos para os alinhamentos locais, ou seja, para a obtenção dos runs e das regiões ortólogas. No primeiro caso apelamos para o conceito de ortologia forte, no sentido de obtermos relações biunívocas que minimizam a ação dos parálogos. Já no segundo caso não poderíamos ser tão exigentes, principalmente se considerarmos o fato de um mesmo gene poder participar de mais de um par de ortólogos ou até mesmo de mais de uma região ortóloga, por causa das duplicações internas que podem ocorrer num genoma.

5.1.5 Famílias de genes parálogos

O objetivo neste caso é encontrar famílias de genes parálogos. Ou seja, famílias de genes que supostamente evoluíram a partir de um mesmo ancestral, por duplicação.

A idéia aqui é usarmos critérios semelhantes aos utilizados na definição de ortologia, quais sejam, significância estatística da similaridade e cobertura de alinhamento.

Sejam $\mathcal{G}$ e $\mathcal{G}'$ dois grafos cujos vértices são os genes do proteoma do genoma G. Sejam v e w dois vértices de $V(\mathcal{G})$ representando, respectivamente, os genes g_v e g_w. Seja ainda os valores limítrofes S_{par} e P_{par}, equivalentes a S e P, respectivamente. As arestas de $\mathcal{G}$ e $\mathcal{G}'$ são definidas da seguinte forma:

- $(v, w) \in E(\mathcal{G})$ se, e somente se, g_v e g_w são parálogos, considerando os valores limítrofes S_{par} e P_{par}, como no critério de ortologia definido anteriormente;
- $(v, w) \in E(\mathcal{G}')$ se, e somente se, g_v e g_w são parálogos, considerando os valores limítrofes S'_{par} e P'_{par}, como no critério de ortologia definido anteriormente, tais que $S'_{par} \geq S_{par}$ e $P'_{par} \leq P_{par}$.

A estratégia para a determinação de uma família de parálogos tem os seguintes 3 passos:

1. Encontre todas as cliques máximas de $\mathcal{G}$;
2. Para cada clique máxima C de $\mathcal{G}$, faça
   $C \leftarrow C \cup \{v \in V(\mathcal{G}) \setminus C \mid (v, w) \in E(\mathcal{G}'),$ para algum $w \in C\}$.

Os genes corespondentes aos vértices acrescentados às famílias, segundo o item 2 acima, são chamados de **agregados**.

Agora, seja $sm(v, C)$ o valor da significância de similaridade média entre o gene g_v e todos os genes cujos vértices correspondentes pertencem a $C - \{v\}$. Sejam ainda $C_1, \ldots, C_t$ os conjuntos criados no passo 2 que contêm v, e $C_{max} \in \{C_1, \ldots, C_t\}$ tal que

$$C_{max} = \max_{i=1\ldots t} \quad sm(v, C_i).$$

3. Faça $C \leftarrow C - \{v\}, \forall C \neq C_{max}$.

Qualquer conjunto C criado seguindo os três passos acima, tal que $|C| \geq 2$, forma uma **família de genes parálogos** do proteoma do genoma G.

Intuitivamente, estamos propondo a seguinte regra para a determinação de uma família de parálogos. Devem pertencer a uma família de parálogos os genes que são parálogos entre si, segundo algum critério que envolva significância de similaridade e cobertura de alinhamento. Além desses, um gene de fora pode vir a participar da família desde que seja parálogo a pelo menos um gene da família, só que agora segundo um critério menos exigente de significância estatística de similaridade e de cobertura.

O passo 3 da estratégia proposta acima diz respeito ao critério de desempate, caso um gene venha a pertencer a mais de uma família. Este critério basicamente diz que o gene deve pertencer a apenas uma família, que deve ser aquela com a qual o gene mais se identifica. Isso é medido pela média das significâncias entre o gene e todos os outros, de cada família.

Novamente estamos tentando aproximações operacionais de conceitos biológicos. Especificamente, neste caso, o conceito é o de parálogo.

5.2 Implementação – o programa EGG

Nesta seção vamos descrever uma proposta de implementação para a metodologia vista na seção anterior. A implementação resultou no programa chamado EGG, de Extended Genome-Genome comparison. EGG foi inicialmente proposto por nós em [3], e depois reformulado em [59]. O que é descrito aqui é uma segunda reformulação. O objetivo geral de EGG, portanto, é o de comparar dois proteomas.

Em linhas gerais, EGG constrói um grafo bipartido onde os vértices são os genes de cada proteoma e as arestas representam ortologia entre os genes. Após a construção desse grafo bipartido, EGG encontra estruturas organizacionais envolvendo os genes, objetivando atender aos objetivos listados na seção 5.1.

5.2.1 Descrição das fases de EGG

EGG tem três grandes fases:

- **Fase 1.** comparações dos genes;
- **Fase 2.** determinação das arestas do grafo; e
- **Fase 3.** determinação de estruturas organizacionais.

As três fases são agora descritas.

Fase 1 – comparações dos genes

Na fase 1 cada gene g_i de G é comparado com todos os genes de H e, em seguida, cada gene h_j de H é comparado com todos os genes de G. O objetivo aqui é obtermos as ortologias unidirecionais, que citamos na metodologia, para podermos determinar os pares de ortólogos.

Para essa fase, optamos por usar como ferramenta de comparação o programa BLAST, devido a Altschul e colegas [6, 7], que fornece para cada gene g_i uma lista de genes considerados similares a ele, dentre todos os genes de H. Cada gene h_j retornado por BLAST é denominado **hit**. Maiores detalhes sobre essa ferramenta podem ser vistos na seção 2.3.

Um aspecto muito importante a ser citado aqui é que a similaridade detectada pelo BLAST, como vimos, é medida através de uma estimativa de significância estatística do hit, chamada de **e-value**, que essencialmente representa o número esperado de hits encontrados ao acaso. Ou seja, quanto menor o e-value, menor é a probabilidade de um determinado hit ter sido

encontado ao acaso. Assim, quanto mais significante estatisticamente é o hit, menor é o e-value.

Para implementar a medida de significância estatística de similaridade $s(g, h)$, proposta na metodologia apresentada anteriormente, vamos usar exatamente o e-value dado pelo BLAST. Assim, o valor limite S de significância estatística de similaridade deve ser visto como um valor máximo que não deve ser ultrapassado, enquanto que S' deve ser visto como um valor mínimo que não deve ser alcançado.

Outro aspecto importante diz respeito ao fato de podermos ter e-values diferentes para quando h_j é hit de g_i e para quando g_i é hit de h_j. Esse fato é raro, mas pode acontecer, já que podemos ter proteomas de tamanhos muito diferentes e além disso proteomas contendo muitas seqüências repetidas.

Com isso, podemos dizer que já conseguimos atingir o objetivo 1, descrito na metodologia, qual seja, o de determinar os genes específicos de um proteoma em relação ao outro. Especificamente, EGG considera um gene g_i específico em relação a H quando g_i não obteve hits com e-values menores ou iguais a $S' = 10^{-3}$.

A complexidade da fase 1 depende diretamente da complexidade da ferramenta utilizada nas comparações, no caso o BLAST. Uma das vantagens do uso do BLAST é exatamente a velocidade, como visto na seção 2.3.

A principal estrutura de dados usada para o armazenamento dos hits é muito simples. EGG usa, para cada gene g_i (o mesmo vale para h_j), uma lista ligada contendo todos os hits encontrados por g_i, na ordem não-decrescente de e-value. Assim, dado o gene g_i, seu melhor hit é facilmente recuperável em tempo constante.

Fase 2 – determinação das arestas do grafo

BLAST também retorna, para cada hit h_j encontrado para g_i, um alinhamento entre g_i e h_j e uma medida de similaridade, baseada na qualidade do alinhamento, denominada **score**.

Para alcançar o objetivo 3 da lista de objetivos descrita na metodologia, ou seja, para determinar os pares de ortólogos, EGG cria o que denominamos um *match*. Um **match** entre g_i e h_j acontece quando g_i encontrou h_j como hit, tendo como limites os valores: $S = 10^{-5}$, $P = 60\%$, e vice-versa. Esses valores são usados na literatura [63, 64], mas podem ser alterados pelo usuário de EGG. Essencialmente, um match é uma aresta do grafo bipartido. Usaremos, daqui em diante, o termo "match", no lugar de "par de genes ortólogos".

Vale aqui comentarmos sobre os valores $S = 10^{-5}$, $S' = 10^{-3}$, e $P = 60\%$. Estamos considerando a região compreendida entre 10^{-5} e 10^{-3} como uma "zona de dúvida". Um e-

value menor ou igual a 10^{-5} indica homologia com alta probabilidade. Assim, dois genes são considerados ortólogos se forem homólogos com alta probabilidade e, além disso, passarem pelo teste da cobertura de alinhamento. No outro extremo, um gene que não consegue hits no outro genoma com e-value menor ou igual que $S' = 10^{-3}$ é considerado específico.

Em resumo, EGG considera que

> g e h são ortólogos se, e somente se, $s(g, h) \leq 10^{-5}$ e pelo menos 60% de g e 60% de h são cobertos pelo alinhamento de g e h.

Além disso, EGG considera que

> g é específico em relação a H se, e somente se, $s(g, h) > 10^{-3}$.

Sejam m e n o número de genes de G e H, respectivamente. EGG armazena os matches numa matriz binária $A_{m \times n}$, tal que $A_{i,j} = 1$ se, e somente se, g_i e h_j formam um match.

Para implementar o conceito de par de genes fortemente ortólogos, EGG usa o que chamamos de *BBH*, de *Bidirectional Best Hit*. Um par (g_i, h_j) de genes ortólogos forma um **BBH**, se h_j é o melhor hit encontrado por g_i, ou seja, aquele com menor e-value, e vice-versa. Daqui para frente, usaremos o termo "BBH", ao invés de "par de genes fortemente ortólogos".

Considerando, como já foi dito, que os hits de um gene g_i são armazenados numa lista ligada, de tal forma que o primeiro nó da lista armazena seu melhor hit, EGG é capaz de obter, em tempo constante, o melhor hit de g_i. Assim, em tempo linear EGG é capaz de obter todos os BBHs de dois proteomas.

Fase 3 – determinação de estruturas organizacionais

As estruturas organizacionais as quais nos referimos são os runs, as ROs e a espinha dorsal de dois proteomas. Vamos agora descrever como EGG implementa cada uma dessas estruturas.

Runs

Para a implementação de uma região ortóloga, como vimos anteriormente, EGG precisa determinar previamente os runs. Dada a definição de match acima, podemos dizer que um run é uma seqüência de pelo menos dois matches.

A estratégia que EGG usa para determinar os runs é muito simples e eficiente, uma vez que temos os matches armazenados na matriz A. Os runs são simplesmente diagonais de A onde

as posições estão preenchidas com 1. Estamos interessados em diagonais com pelo menos dois uns consecutivos.

O programa gera automaticamente um código para cada run encontrado. O código traz seqüencialmente as seguintes informações: os quatro primeiros símbolos identificam o par de genomas comparados; em seguida temos o ano, o mês e o dia (dois dígitos para cada um); um número seqüencial do run, naquela comparação genômica; e finalmente a descrição se ele é paralelo ou anti-paralelo e se é consistente ou inconsistente. A figura 5.3 mostra um exemplo de run.

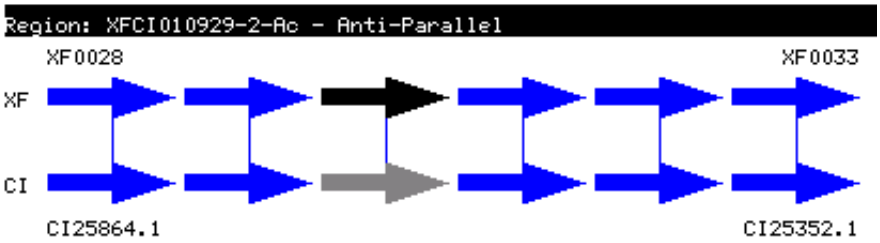

Figura 5.3: Exemplo de um run anti-paralelo consistente. A ordem dos genes no genoma de baixo aparecem invertida.

Regiões ortólogas

Dissemos, na descrição da metodologia, que uma RO, ou é um run isolado com pelo menos $M = 3$ matches ou a união de runs que distam no máximo um valor fixo K. Na implementação, vamos refinar um pouco mais essa idéia, para que tenhamos a noção exata de proximidade a que nos referimos.

Sejam

$$(g_i, h_j), (g_{i+1}, h_{j+1}), \ldots, (g_k, h_l) \quad \text{e} \quad (g_p, h_q), (g_{p+1}, h_{q+1}), \ldots, (g_r, h_s)$$

dois runs. O número de genes entre os runs no genoma G é dado por $p - k - 1$, e em H dado por $q - l - 1$, e são chamados de **intervalos**. Nossa metodologia para detectar as ROs é procurar juntar runs próximos, formando uma só região que evidencie um bloco de genes com certo grau de ortologia, desde que esses intervalos não sejam muito grandes, onde os tamanhos desses intervalos não são necessariamente iguais.

Certamente, se não tomarmos o devido cuidado, esse critério pode ser bastante arbitrário. Então decidimos adotar a seguinte estratégia: sejam

$$I_{\min} = \min\{p - k - 1, q - l - 1\} \quad \text{e} \quad I_{\max} = \max\{p - k - 1, q - l - 1\},$$

e sejam `max_small_gap` e `max_large_gap` dois parâmetros fornecidos pelo usuário que, respectivamente, determinam o tamanho máximo do menor e do maior intervalo entre os runs, em número de genes. Nós juntamos os dois runs formando uma RO se

$$I_{\min} \leq \texttt{max\_small\_gap} \quad \text{e} \quad I_{\max} \leq \texttt{max\_large\_gap}.$$

Neste caso dizemos que os runs são **próximos** e essa operação recebe o nome de **junção** de runs. Os valores padrão para esses parâmetros são: 5 para `max_large_gap` e 2 para `max_small_gap`.

O algoritmo para a determinação de todas as ROs entre os dois proteomas funciona de forma incremental, no sentido de que uma região resultante da junção de dois runs próximos pode ainda ser juntada com o próximo run à direita. Assim, uma RO pode conter vários pares de intervalos e vários runs. Essas junções são feitas da esquerda para a direita, de acordo com as coordenadas do genoma G. O algoritmo não faz qualquer restrição quanto à consistência ou quanto ao paralelismo dos runs envolvidos na junção. Ou seja, se dois runs são próximos, de acordo com a definição acima, então nós os juntamos, formando uma RO. A figura 5.4 mostra uma RO resultante da junção de três runs.

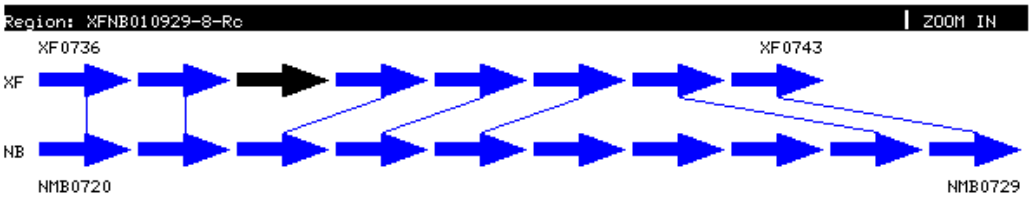

Figura 5.4: Exemplo de uma RO encontrada na comparação de *Xylella fastidiosa* e *Neisseria meningitidis serogroup B strain MC58*, resultante da junção de 3 runs (respectivamente com 2, 3 e 2 matches).

Essa ausência de maiores exigências com relação ao tipo dos runs a serem juntados pode acarretar em regiões que na verdade não possuem homologia nenhuma e tenham se formado apenas pela proximidade dos runs. No entanto, estamos tentando determinar pistas para ROs que devem, posteriormente, passar por um crivo onde se deve levar em conta aspectos relacionados às funcionalidades dos genes envolvidos, ou até mesmo aspectos mais complexos, como por exemplo aqueles relacionados a vias metabólicas.

Como parte da saída de EGG, temos um arquivo texto, onde as ROs encontradas são mostradas. Na figura 5.5 temos uma RO resultante da comparação entre *Xylella fastidiosa* e *Haemophilus influenzae*. O código da região é o mesmo daquele do run mais à esquerda que a gerou, com exceção do sufixo, que passa a ter o símbolo `R`, de região, seguido de `c` (consistente) ou `i` (inconsistente). Os símbolos que têm a ver com paralelismo deixam de fazer sentido, já que runs com paralelismo diferentes podem ser juntados.

```
>XFHI020115-17-Rc
4 matches
4kb in xf - 4kb in hi
=====================================================================
Gene (xf)      gi        size  product
=====================================================================
+XF2243      9107397    602aa  GTP binding protein
+XF2244      9107398    266aa  signal peptidase I
+XF2245      9107399    137aa  hypothetical protein
+XF2246      9107400    212aa  ribonuclease III
+XF2247      9107401    298aa  GTP binding protein
=====================================================================
Gene (hi)      gi        size  product
=====================================================================
-HI0013      1572957    302aa  GTP-binding protein (era)
-HI0014      1572958    227aa  ribonuclease III (rnc)
-HI0015      1572959    349aa  signal peptidase I (lepB)
-HI0016      1572960    598aa  GTP-binding membrane protein (lepA)
=======
matches
=======
=============================================================================================
Gene         start size  e-value [     best hit    ] product
=============================================================================================
+XF2247    2137103 298     9e-83 [best             ] GTP binding protein
-HI0013      13423 302     1e-82 [best             ] GTP-binding protein (era)
---------------------------------------
+XF2246    2136468 212     7e-52 [best             ] ribonuclease III
-HI0014      14328 227     1e-51 [best             ] ribonuclease III (rnc)
---------------------------------------
+XF2244    2135226 266     5e-34 [best             ] signal peptidase I
-HI0015      15013 349     7e-36 [best             ] signal peptidase I (lepB)
---------------------------------------
+XF2243    2133317 602         0 [best             ] GTP binding protein
-HI0016      16071 598         0 [best             ] GTP-binding membrane protein (lepA)
---------------------------------------
=============================================================================================
```

Figura 5.5: RO encontrada na comparação de *Xylella fastidiosa* e *Haemophilus influenzae*, contendo 4 matches. Informações mostradas da região são as seguintes: código; número de matches; número aproximado de bases em cada genoma na região; quais genes participam, e para cada um deles, o identificador no genoma, o identificador `gi`, o tamanho e o produto; finalmente os matches daquela região.

Um detalhe importante deve ser citado aqui. Imagine a situação onde temos um run próximo a um match isolado. Se este run e este match estiverem isolados no decorrer dos proteomas, eles não vão ser juntados. Entretanto, essa junção pode ser interessante, já que pode resultar numa região de três ou mais matches. Decidimos então adotar a seguinte estratégia para resolver esse tipo de caso. Basta fazer com que um match isolado seja considerado um run, se esse match for um BBH. Assim, ele passa a ser juntado normalmente a uma RO ou a um run, desde que obedeça às regras de distância. Intuitivamente estamos dizendo que um match isolado não deve contribuir para a formação de uma RO, a menos que a ortologia do par que forma esse match seja forte.

Espinha dorsal dos proteomas

A implementação da espinha dorsal dos proteomas é feita seguindo exatamente o critério descrito na seção 5.1, objetivando a construção de um alinhamento entre os proteomas.

EGG implementa um algoritmo de programação dinâmica, exatamente como proposto no livro de Cormen e colegas [14]. As duas seqüências s e t que usamos como entrada, neste caso, foram tais que $s_i = i$, $1 \leq i \leq m$, representando os genes de G e $t_j = p(j), 1 \leq j \leq n$, representando os genes de H, onde $p(j) = i$ caso (g_i, h_j) seja BBH, ou $p(j) = 0$ caso contrário. O objetivo é o de encontrar uma subseqüência comum às duas seqüências que seja mais longa. O custo do algoritmo é $O(mn)$.

Vale a pena lembrar aqui que o conceito de alinhamento de proteomas proposto está fortemente relacionado à noção de BBH, enquanto que no caso de estruturas menores (alinhamentos locais), como runs e ROs, a noção de match é a mais apropriada. A intenção, como já dissemos, é a de minimizar a interferência dos genes parálogos, que atrapalhariam a formação de um alinhamento global.

A figura 5.7 (seção 5.3) mostra um trecho da espinha dorsal dos preoteomas das duas *Xanthomonas*. Essa espinha dorsal foi utilizada na construção de um mapa, apresentado em [16], que mostra graficamente como os dois proteomas estão alinhados. Vale ressaltar aqui que nos casos em que os genomas são muito próximos (duas *Xanthomonas*, por exemplo), a comparação nos dois níveis, DNA e genes, vão essencialmente produzir resultados equivalentes. A explicação para isso vem do fato de que se a espinha dorsal de dois proteomas é facilmente detectável, então é porque eles são suficientemente próximos para que seus genes compartilhem trechos pequenos exatos, detectáveis também por programas que comparam no nível de DNA, como o BACON, por exemplo.

5.2.2 Famílias de genes parálogos

A implementação da metodologia para encontrar famílias de genes parálogos descrita na seção 5.1.5 tem como base as fases 1 e 2 descritas na seção 5.2.1, já que o objetivo aqui é o de comparar um proteoma com ele mesmo.

Na descrição da metodologia, dissemos que estamos interessados nas cliques máximas do grafo cujos vérices são os genes do proteoma. No entanto optamos, por simplicidade, pela implementação de um algoritmo que encontra as cliques *maximais* do grafo, e não as máximas.

Por outro lado, se garantirmos que *todas* as cliques maximais são encontradas, as máximas estarão entre elas. Basta para isso garantirmos que as cliques maximais sejam encontradas de tal maneira que a determinação de uma não interfira na determinação de outra. Nossa

implementação foi feita exatamente no sentido de se obter essa garantia.

O algoritmo implementado para encontrar as cliques maximais funciona da seguinte forma. Primeiro são criadas as arestas (matches) dos grafos $\mathcal{G}$ e $\mathcal{G}'$. Depois, cada aresta é considerada uma clique de $\mathcal{G}$ e vértices são adicionados um-a-um nas cliques até que não conseguimos mais aumentá-las.

O único cuidado que deve ser tomado pelo usuário diz respeito aos parâmetros S_{par}, P_{par}, S'_{par} e P'_{par}, especificamente no acréscimo dos vértices agregados às famílias já existentes. Se a idéia é relaxar um pouco o critério de ortologia, para que os agregados possam de fato entrar para uma determinada família, S'_{par} e P'_{par} devem ser escolhidos cuidadosamente. Especificamente, eles devem ser tais que $S'_{par} \geq S_{par}$ e $P'_{par} \leq P_{par}$.

No caso específico do programa EGG, supusemos os seguintes valores padrão para S_{par}, P_{par}, S'_{par} e P'_{par}, respectivamente, 10^{-5}, 60%, 10^{-5} e 33%.

No caso dos genes que venham a pertencer a mais de uma família, EGG utiliza, ao invés do e-value, outra medida como critério de desempate. A medida é dada pelo score, retornado pelo BLAST. Assim, escolhe-se aquela família com a qual o gene tem maior score médio, considerando todos os outros genes da família, dentre todas as famílias que contêm o referido gene.

A justificativa para esse critério de desempate está no fato de que, se um gene deve pertencer a apenas uma família de parálogos, então que seja aquela com a qual ele mais se identifica. O score médio entre ele e todos os outros elementos da família é uma forma de determinar a família com a qual ele mais se identifica.

A saída de EGG para esta versão é uma lista das famílias encontradas, com respectivos números que as identificam. Para cada família, EGG mostra o score médio de todos os matches e o desvio-padrão. Para cada gene da família, o programa mostra informações como coordenadas de DNA, tamanho, produto, categoria (se disponível) e ainda quais as famílias que continham o gene e que deixaram de contê-lo por causa do critério de desempate. Na seção 5.3 mostramos alguns exemplo de famílias.

5.2.3 Alguns detalhes sobre EGG

Circularidade

Considerando que a grande maioria dos replicons de procariotos são circulares, mas descritos nos arquivos do NCBI como lineares, EGG precisa dar um tratamento especial a essa situação, para não deixar de identificar runs que começam antes e terminam depois do corte feito no genoma.

Para tanto, usamos a seguinte estratégia: aumentamos em k linhas e k colunas a matriz A, que armazena os matches, de tal forma que tenhamos as linhas $m+1, m+2, \ldots m+k$, e colunas $n+1, n+2, \ldots n+k$, e tal que $A_{m+i,j} = A_{i,j}$, para $1 \leq i \leq k$, e $A_{i,n+j} = A_{i,j}$, para $1 \leq j \leq k$. Com isso, foi possível detectar todos os runs que passavam do corte do replicon circular. Para todos os pares de genomas que comparamos, $k = 10$ foi suficiente para resolver este problema.

Custos de tempo e espaço

O programa EGG é dividido em dois módulos. No primeiro a fase 1 é executada, enquanto que as fases 2 e 3 são executadas no segundo módulo. Isso é feito dessa forma por causa da diferença entre os tempos de execução dos dois módulos.

A fase 1 consiste essencialmente em executar BLAST de todos os genes de G contra a base de genes formada por H, e vice-versa. O tempo gasto por EGG para a fase 1 na comparação das duas *Xanthomonas*, por exemplo, é aproximadamente de duas horas e meia. Entretanto, as fases 2 e 3 juntas não ultrapassam 15 segundos, numa máquina do tipo PC (linux) com 1GB de memória RAM.

Considerando que a fase 1 apenas determina as ortologias unidirecionais, independente dos parâmetros, é interessante para o usuário estar apto a executar rapidamente as outras duas fases, já que pode fazê-lo várias vezes, testando os diversos parâmetros, quais sejam, S, P, S', P', M, `max_large_gap` e `max_small_gap`.

Detalhes adicionais

Alguns detalhes adicionais de EGG, como formatos de arquivos de entrada e saída, opções de utilização, etc, estão descritos no apêndice B.

5.3 Resultados

Vamos agora mostrar algumas amostras de resultados que obtivemos com a utilização do programa EGG, nas comparações que fizemos. Os dois grupos de genomas que escolhemos para as comparações de proteomas são mostrados respectivamente nas tabelas 5.1 e 5.2. Estas tabelas contêm siglas usadas daqui em diante.

Essas escolhas surgiram em função da nossa participação nos projetos genoma das bactérias *Xanthomonas axonopodis* pv. *citri* e *Xanthomonas campestris* pv. *campestris* [16], e *Agrobacterium tumefaciens* [73, 74].

sigla	genoma	bases	genes
Xfa	*Xylella fastidiosa*	2679306	2766
Xac	*Xanthomonas axonopodis* pv. *citri*	5175554	4313
Xcc	*Xanthomonas campestris* pv. *campestris*	5076187	4182
Hin	*Haemophilus influenzae Rd*	1830137	1709
Pae	*Pseudomonas aeruginosa PA01*	6264403	5565
Eco	*Escherichia coli K-12 MG1655*	4639221	4289
Nma	*Neisseria meningitidis serogroup A str Z2491*	2184406	2065
Nmb	*Neisseria meningitidis serogroup B str MC58*	2272351	2025
Ccr	*Caulobacter crescentus*	4016947	3737

Tabela 5.1: Genomas usados nas comparações, objetivando principalmente as comparações com Xfa, Xac e Xcc.

sigla	genoma	bases	genes
At1	*Agrobacterium tumefaciens C58 - replicon 1.1*	2841485	2789
At2	*Agrobacterium tumefaciens C58 - replicon 2.1*	2075560	1882
At3	*Agrobacterium tumefaciens C58 - replicon 3.1*	542779	550
At4	*Agrobacterium tumefaciens C58 - replicon 4.1*	214233	198
Mla	*Mesorhizobium loti - plasmid pMLa*	351911	320
Mlb	*Mesorhizobium loti - plasmid pMLb*	208315	209
Mmc	*Mesorhizobium loti*	7036074	6752
Sma	*Sinorhizobium meliloti plasmid pSymA*	1354226	1294
Smb	*Sinorhizobium meliloti plasmid pSymB*	1683333	1571
Smc	*Sinorhizobium meliloti*	3654135	3341

Tabela 5.2: Genomas usados nas comparações de proteomas, objetivando comparações com *Agrobacterium tumefaciens*. O replicon 2.1 de *Agrobacterium tumefaciens* é linear. Os demais são circulares.

A tabela 5.3 mostra o número de matches e BBH's encontrados na comparção de Xac com outros genomas, enquanto que a tabela 5.4 mostra números sobre runs e ROs na comparação dos replicons de *Agrobacterium tumefaciens* e *Mesorhizobium loti*.

A figura 5.6 mostra os BBH's encontrados entre Xac e Xcc, enquanto que a figura 5.7 mostra um trecho da espinha dorsal dos dois proteomas. Uma análise desses resultados nos permite perceber um alto grau de conservação na ordem dos genes dos dois genomas. O gráfico dos BBHs também mostra três grandes inversões entre os proteomas. Uma delas ocorreu no que supostamente é o término da replicação (meio do gráfico), enquanto que as outras ocorreram quase que simetricamente, com translocações com relação à origem de replicação.

genoma1 × genoma2	# total de matches	# total de BBHs
Xfa × Xac	6890	1526
Xfa × Xcc	6553	1523
Xfa × Hin	2649	809
Xfa × Pae	9213	1253
Xfa × Eco	6104	1109
Xfa × Nma	2469	905
Xfa × Nmb	2531	895
Xfa × Ccr	5583	1028
Xac × Xcc	22246	3441
Xac × Hin	4775	952
Xac × Pae	28320	2056
Xac × Eco	15797	1594
Xac × Nma	4306	1054
Xac × Nmb	4375	1043
Xac × Ccr	17321	1641

Tabela 5.3: Número de matches e BBHs entre *Xanthomonas citri* e outros genomas.

genoma1 × genoma2	# total de runs	# de runs com 4+ matches e 3+ BBHs	# total de ROs
Mla × At1	140	3	42
Mla × At2	309	5	90
Mla × At3	72	3	24
Mla × At4	36	5	11
Mlb × At1	20	0	8
Mlb × At2	19	1	6
Mlb × At3	9	1	2
Mlb × At4	6	3	1
Mlc × At1	2296	183	539
Mlc × At2	3716	77	1072
Mlc × At3	846	77	222
Mlc × At4	202	4	54

Tabela 5.4: Números de runs e regiões ortólogas das comparações de *Agrobacterium* com *Mesorhizobium*.

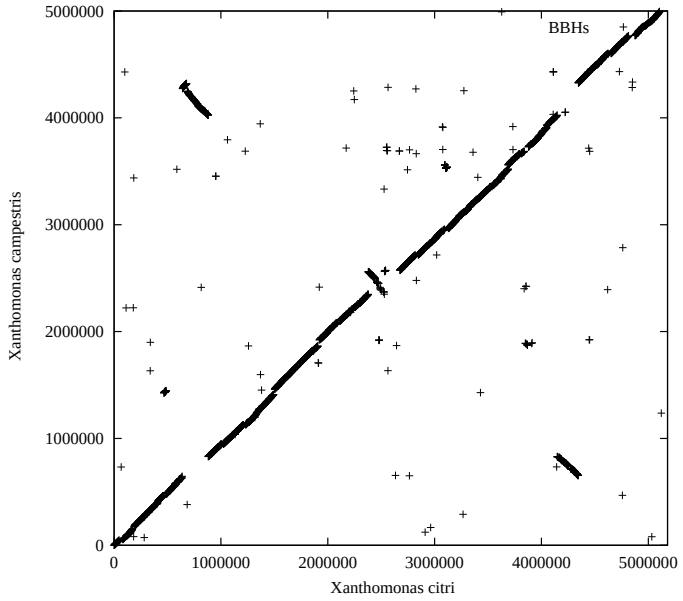

Figura 5.6: BBHs entre `Xac` e `Xcc`.

```
=============================================================================================================================
                        PRODUCT  START..END        GENE(STRAND)      (STRAND)GENE      START..END     PRODUCT
=============================================================================================================================

   conserved hypothetical protein   94577..95551    XAC0077 (-)   <<<>>>   (-) XCC0053    67292..68254   conserved hypothetical protein
       ATP-dependent RNA helicase   96061..98466    XAC0078 (-)   <<<>>>   (-) XCC0054    68779..71184   ATP-dependent RNA helicase
                                                          -                (-) XCC0055    71497..72642   conserved hypothetical protein
                    hemolysin III   98737..99390    XAC0079 (+)   <<<>>>   (+) XCC0056    72924..73583   hemolysin III
   conserved hypothetical protein   99461..99715    XAC0080 (+) #              -
   conserved hypothetical protein   99706..100002   XAC0081 (+) #              -
       short chain oxidoreductase  100188..101084   XAC0082 (-)                -
        short chain dehydrogenase  101147..101881   XAC0083 (-) *              -
scriptional regulator lysR family  102007..102915   XAC0084 (+) *              -
   conserved hypothetical protein  103087..104085   XAC0085 (-) #              -
             hypothetical protein  104350..104862   XAC0086 (+) #              -
             hypothetical protein  105104..105421   XAC0087 (+) #              -
                                                          -              # (-) XCC0057    73656..73970   hypothetical protein
                                                          -              # (-) XCC0058    73970..74344   conserved hypothetical protein
                                                          -              # (-) XCC0059    74673..75380   hypothetical protein
          NAD(P)H oxidoreductase  105664..106137   XAC0088 (-)   <<<>>>   (-) XCC0060    75555..76070   NAD(P)H oxidoreductase
```

Figura 5.7: Trecho do arquivo que mostra o alinhamento dos proteomas de `Xac` e `Xcc`. Cada símbolo `<<<>>>` mostra um BBH pertencente à espinha dorsal. O símbolo `#` indica que o gene é específico; o símbolo `*` indica que o gene teve um hit, mas não fez BBH; quando um gene aparece sem casamento e sem qualquer símbolo, temos uma translocação, ou seja, ele teve um BBH, mas esse BBH não conseguiu ficar aparente, porque está cruzando com outros, e esses outros foram mostrados pela programação dinâmica.

A espinha dorsal de `Xac` e `Xcc` tem um total de 2929 pares de genes fortemente ortólogos. Além desses pares, outros 512 pares foram translocados, totalizando 3441 BBHs. `Xac` tem 872 genes específicos em relação a `Xcc`, enquanto que `Xcc` tem 741 genes específicos em

relação a `Xac`. A espinha dorsal mostrada aqui foi utilizada, como dissemos, na confecção de uma mapa, mostrado em [16], que mostra como os proteomas se alinham.

A figura 5.8 mostra os BBHs entre os replicons `At1` de *Agrobacterium tumefaciens* e `Smc` de *Sinorhizobium meliloti*. Essa figura deu origem à figura 3 de [74]. O gráfico mostra uma perda de conservação de ordem nas regiões que envolvem os intervalos [2500000 . . . 2800000] de `At1` e [2700000 . . . 3300000] de `Smc`. A mesma região em `Smc` apresenta duas regiões de conservação quando esse replicon é comparado ao replicon circular `At2` de *Agrobacterium*, uma com 46 genes (44 kb) e outra com 65 genes (89 kb). A primeira delas pode ser vista no apêndice A.

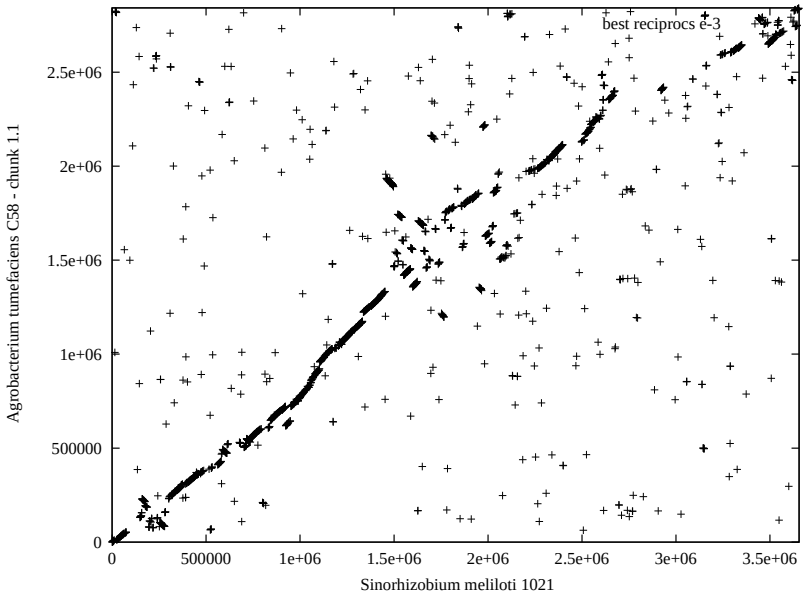

Figura 5.8: BBHs entre o replicon `At1` de *Agrobacterium tumefaciens* e o replicon `Smc` de *Sinorhizobium meliloti*.

Para que os participantes dos projetos das *Xanthomonas* e do *Agrobacterium* pudessem ter acesso a informações referentes às comparações dos proteomas, disponibilizamos via web algumas páginas com arquivos contendo informações sobre runs, ROs, gráficos, etc. Em particular, disponibilizamos uma interface que possibilita ao usuário uma visualização gráfica de como se comportam todas as regiões ortólogas encontradas. A figura 5.9 mostra uma dessas páginas, que são geradas automaticamente, para a comparação dos proteomas de `Xfa` e `Eco`.

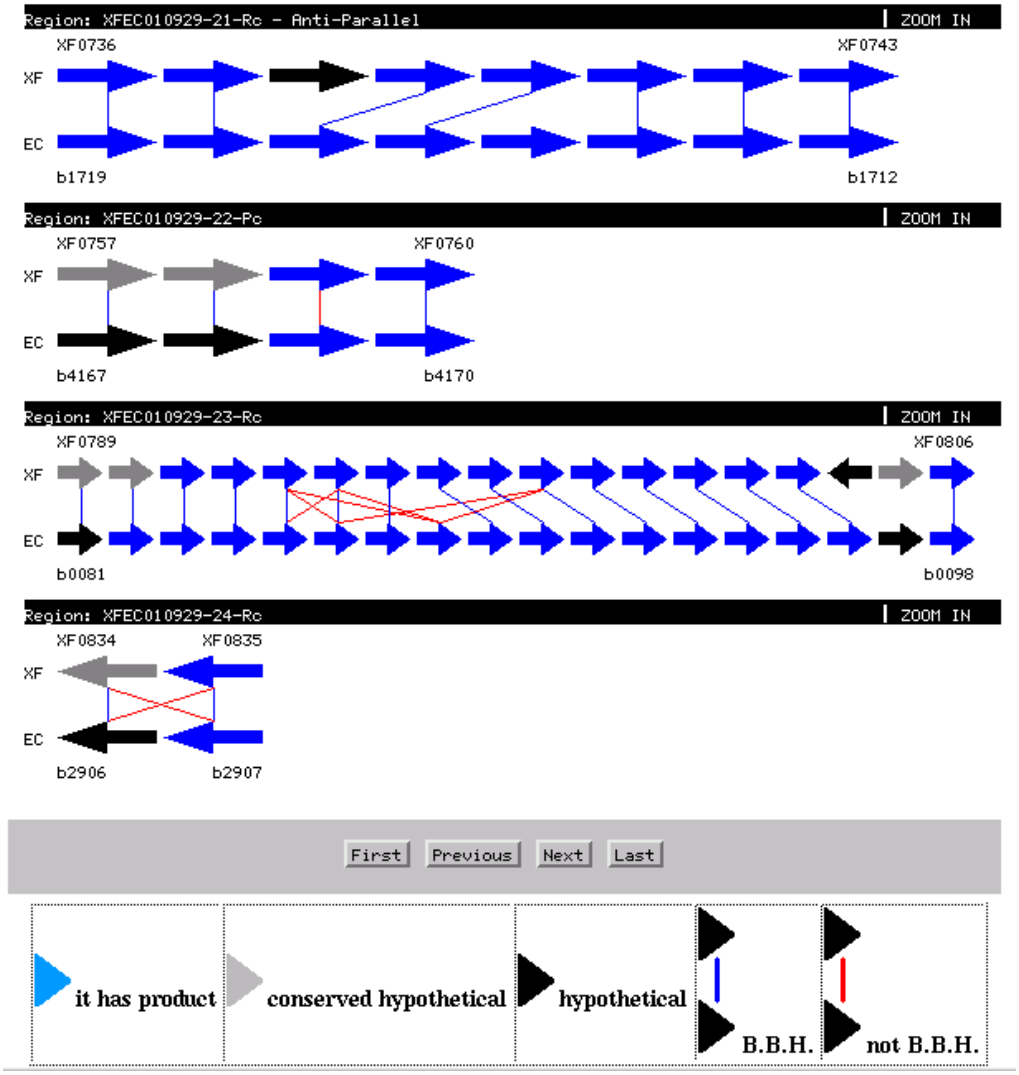

Figura 5.9: Exemplo de uma página www contendo quatro ROs entre *Xylella fastidiosa* e *E. coli*. Cada seta representa um gene (a direção da seta mostra a fita). Apesar de não estarmos vendo as cores aqui, a página é colorida. Genes azuis têm função definida; pretos são hipotéticos; e os cinzas são hipotéticos conservados. Os matches também são coloridos: os azuis representam BBHs e os vermelhos representam os não-BBHs. O código da RO (mostrado na barra horizontal acima da região) contém uma ligação para o arquivo textual da região.

A figura 5.10 mostra algumas famílias de genes parálogos de `Xfa`. Parte significativa das famílias apresentam concordância entre as funções dos genes a elas pertencentes. Além disso, alguns genes agregados têm anotação distinta dos demais na família.

No apêndice A mostramos resultados adicionais nas comparações entre os diversos genomas com os quais trabalhamos.

```
>32 (7 orfs)  score_medio=98.9524 desvio-padrao=4.86826
    -XF0145  147702..148457   251aa oxidoreductase ## 575 ##
    -XF0173  182075..182830   251aa 3-oxoacyl-[ACP] reductase
[*] -XF0319  334769..335509   246aa acetoacetyl-CoA reductase
    +XF0671  642421..643164   247aa 3-oxoacyl-[ACP] reductase ## 226 ##
    +XF0913  879990..880775   261aa tropinone reductase ## 575 226 ##
    -XF1726 1645626..1646393  255aa 2,5-dichloro-2,5-cyclohexadiene-1,4-diol dehydrogenase ## 575 226 ##
    -XF1744 1663245..1663988  247aa oxidoreductase ## 575 ##

>39 (3 orfs)  score_medio=135.667 desvio-padrao=11.225
    +XF0151  154635..156038   467aa phosphomannomutase
    -XF0260  272001..273503   500aa phosphoglucomutase/phosphomannomutase
    +XF1468 1413388..1414734  448aa phosphomannomutase

>109 (7 orfs)  score_medio=99.3333 desvio-padrao=5.37587
    +XF0322  338428..339123   231aa two-component system, regulatory protein ## 152 ##
    +XF0389  400823..401506   227aa two-component system, regulatory protein ## 152 ##
[*] -XF0450  457972..458370   132aa two-component system, regulatory protein ## 152 ##
[*] -XF1955 1863756..1864193  145aa pilus protein
    -XF2336 2215260..2215985  241aa two-component system, regulatory protein
    +XF2534 2407544..2408221  225aa two-component system, regulatory protein ## 549 ##
    -XF2593 2472975..2473775  266aa two-component system, regulatory protein ## 549 152 ##

>113 (4 orfs)  score_medio=68.5 desvio-padrao=6.69328
    +XF0323  339107..340489   460aa two-component system, sensor protein ## 688 ##
[*] +XF0390  401522..402970   482aa two-component system, sensor protein ## 688 ##
[*] -XF1849 1763610..1764665  351aa two-component system, sensor protein
    +XF2535 2408430..2409551  373aa two-component system, sensor protein

>114 (2 orfs)  score_medio=210 desvio-padrao=0
    +XF0325  341646..341960   104aa transposase OrfA
    +XF0535  528651..528965   104aa transposase OrfA

>115 (2 orfs)  score_medio=644 desvio-padrao=0
    +XF0326  341991..342965   324aa transposase OrfB
    +XF0536  528966..530138   390aa transposase OrfB

>116 (3 orfs)  score_medio=148.333 desvio-padrao=11.4673
    +XF0339  347467..350250   927aa conserved hypothetical protein ## 616 ##
    -XF0550  537646..540771  1041aa conserved hypothetical protein
    +XF2237 2126364..2129240  958aa conserved hypothetical protein ## 616 ##

>155 (3 orfs)  score_medio=78.6667 desvio-padrao=8.63134
    +XF0465  471787..473184   465aa GTP-binding protein
    +XF2247 2137103..2137999  298aa GTP binding protein
[*] -XF2778 2672723..2674078  451aa thiophene and furan oxidation protein
```

Figura 5.10: Algumas famílias de parálogos de `Xfa`. Cada família tem um identificador, que é mostrado após o símbolo `>`. Também são mostrados o número de genes da família; o score (BLAST) médio e o desvio padrão dos scores dos pares. Para cada gene da família, além das informações sobre o gene, EGG também mostra quais as famílias que também continham o gene e deixaram de contê-lo por causa do critério de desempate (delimitadas com `##`). Os genes marcados com `[*]` são os agregados. Note que o gene `-XF1955` é agregado e possui anotação distinta dos demais na família. O mesmo acontece com o gene `-XF2778`.

5.4 Discussão

Vamos agora mostrar uma aplicação adicional da nossa metodologia, assim como comentar trabalhos desenvolvidos por outros autores.

5.4.1 Aplicação: anotação de genoma

As principais aplicações da comparação de proteomas foram listadas no início deste capítulo. Entretanto, entendemos que esse tipo de comparação pode ser usado numa importante fase de um projeto genoma: a anotação.

A anotação consiste na predição da localização dos genes e suas funções. A predição de função é feita com base na comparação do gene com outros dos quais se sabe a função. No entanto, quando a similaridade entre os genes não é tão aparente, programas que determinam pares de ortólogos com base na similaridade podem perder alguns pares. Dessa forma, a posição relativa de um gene numa região ortóloga pode ajudar na determinação de homologias não detectadas por similaridade.

Além disso, a disposição dos genes numa região pode auxiliar na anotação do gene, no sentido de corrigir falhas na atribuição de função. A figura 5.11, por exemplo, mostra uma RO entre os proteomas de *Xylella fastidiosa* e *Xanthomonas axonopodis* pv. *citri*, onde o penúltimo gene de *Xylella*, `+XF1109` foi anotado como hipotético, formando um BBH com o gene `+CI2349.1` de *Xanthomonas citri*. Essa anotação (que classifica a proteína como hipotética) mereceria uma investigação.

Outro tipo de situação que pode ser resolvida com a determinação das regiões ortólogas é a descoberta de *genes falsos*. Denominamos **genes falsos** aqueles que foram erroneamente preditos.

A figura 5.12 mostra uma RO encontrada na comparação entre *Xylella fastidiosa* e *Caulobacter crescentus*. A disposição dos genes da região é mostrada na figura 5.13. Note que o segundo gene de *Xylella*, `-XF0951`, além de ter sido anotado como hipotético e ser pequeno, com apenas 79 aminoácidos, tem fita inconsistente com os demais genes da região e, principalmente, é o único gene da região que não tem matches na região. `-XF0951`, portanto, é um candidato a gene falso. Outra possibilidade para explicar sua aparição é que pode ser um gene recente, inserido em *Xylella*.

Dois outros exemplos equivalentes são os dos genes `XAC1943`, com 195 aminoácidos e `XAC1956`, com apenas 34 aminoácidos, encontrados numa mesma RO, na comparação entre as *Xanthomonas*, como pode ser visto na figura 5.14. No primeiro caso o número de aminoácidos não é tão pequeno, mas o gene se encontra bem no centro de uma grande região (28 BBHs); a fita

não está consistente com as dos outros genes; e ele foi anotado como hipotético. No segundo caso ainda temos como agravante o tamanho do gene, com apenas 34 aminoácidos.

Obviamente essas análises devem seguir critérios biológicos mais rigorosos. Mas vale notar que a visualização gráfica da RO pode ajudar nessas análises.

```
>XFCI011204-72-Pc
6 matches
8kb in xf - 8kb in ci
=====================================================================
Gene (xf)      gi        size  product
=====================================================================
+XF1105      9106060    237aa  dihydrodipicolinate reductase
+XF1106      9106061    399aa  carbamoyl-phosphate synthase small chain
+XF1107      9106062   1080aa  carbamoyl-phosphate synthase large chain
+XF1108      9106063    154aa  transcriptional elongation factor
+XF1109      9106064    293aa  hypothetical protein
+XF1110      9106065    581aa  single-stranded DNA exonuclease
=====================================================================
Gene (ci)      gi        size  [      category      ] product
=====================================================================
+CI2343.1    234301     237aa  [  II.A.2            ] dihydrodipicolinate reductase
+CI2345.1    234501     390aa  [  II.B.2 II.A.1     ] carbamoyl-phosphate synthase small chain
+CI2347.1    234701    1089aa  [  II.B.2 II.A.1     ] carbamoyl-phosphate synthase large chain
+CI2348.1    234801     157aa  [  III.B.5 VII.E     ] transcriptional elongation factor
+CI2349.1    234901     305aa  [  VII.H             ] regulatory protein
+CI2350.1    235001     577aa  [  III.A.3           ] single-stranded-DNA-specific exonuclease
=======
matches
=======
=============================================================================================
Gene         start size  e-value [best hit] [category     ] product
=============================================================================================
+XF1110    1063270 581         0 [best    ] [             ] single-stranded DNA exonuclease
+CI2350.1  2162288 577         0 [best    ] [III.A.3      ] single-stranded-DNA-specific exonuclease
---------------------------------------
+XF1109    1062323 293    1e-112 [best    ] [             ] hypothetical protein
+CI2349.1  2161371 305    1e-112 [best    ] [VII.H        ] regulatory protein
---------------------------------------
+XF1108    1061812 154     4e-71 [best    ] [             ] transcriptional elongation factor
+CI2348.1  2160878 157     2e-71 [best    ] [III.B.5 VII.E] transcriptional elongation factor
---------------------------------------
+XF1107    10585611080         0 [best    ] [             ] carbamoyl-phosphate synthase large chain
+CI2347.1  21576091089         0 [best    ] [II.B.2 II.A.1] carbamoyl-phosphate synthase large chain
---------------------------------------
+XF1106    1057161 399    1e-174 [best    ] [             ] carbamoyl-phosphate synthase small chain
+CI2345.1  2156012 390    1e-174 [best    ] [II.B.2 II.A.1] carbamoyl-phosphate synthase small chain
---------------------------------------
+XF1105    1056325 237     2e-94 [best    ] [             ] dihydrodipicolinate reductase
+CI2343.1  2155056 237     8e-95 [best    ] [  II.A.2     ] dihydrodipicolinate reductase
---------------------------------------
```

Figura 5.11: RO obtida na comparação de e *Xylella fastidiosa* e *Xanthomonas axonopodis* pv. *citri*. O gene `+XF1109`, de *Xylella*, anotado como hipotético, deve ter sua função investigada.

```
>XFCC010905-14-Rc
4 matches
3kb in xf - 3kb in cc
================================================================
Gene(xf) gi        size  product
================================================================
+XF0950  9105878   364aa riboflavin-specific deaminase
-XF0951  9105879    79aa hypothetical protein
+XF0952  9105880   200aa riboflavin synthase alpha chain
+XF0953  9105881   377aa GTP cyclohydrolase II/3,4-dihydroxy-2-butanone 4-phosphate synthase
+XF0954  9105882   154aa 6,7-dimethyl-8-ribityllumazine synthase
================================================================
Gene(cc) gi       size  product
================================================================
+CC0885  13422148 331aa riboflavin biosynthesis protein RibD
+CC0886  13422149 205aa riboflavin synthase, alpha subunit
+CC0887  13422150 400aa 3,4-dihydroxy-2-butanone 4-phosphate synthase/GTP cyclohydrolase II
+CC0888  13422151 153aa riboflavin synthase, beta subunit
=======
matches
=======
================================================================================
Gene    start size e-value[best hit ] product
================================================================================
+XF0954 915898 154  2e-24 [best     ] 6,7-dimethyl-8-ribityllumazine synthase
+CC0888 982033 153  1e-24 [best     ] riboflavin synthase, beta subunit
--------------------------------
+XF0953 914735 377  1e-69 [best     ] GTP cyclohydrolase II/3,4-dihydroxy-2-butanone 4-phosphate synthase
+CC0887 980815 400  8e-70 [best     ] 3,4-dihydroxy-2-butanone 4-phosphate synthase/GTP cyclohydrolase II
--------------------------------
+XF0952 914136 200  3e-37 [best     ] riboflavin synthase alpha chain
+CC0886 980222 205  2e-37 [best     ] riboflavin synthase, alpha subunit
--------------------------------
+XF0950 912455 364  3e-36 [best     ] riboflavin-specific deaminase
+CC0885 979216 331  1e-36 [best     ] riboflavin biosynthesis protein RibD
--------------------------------
```

Figura 5.12: RO obtida na comparação de *Xylella fastidiosa* e *Caulobacter crescentus* onde, pela disposição e fita dos genes; pelos matches vistos; e pelo seu tamanho e função, `-XF0951` tem chance de ser falso.

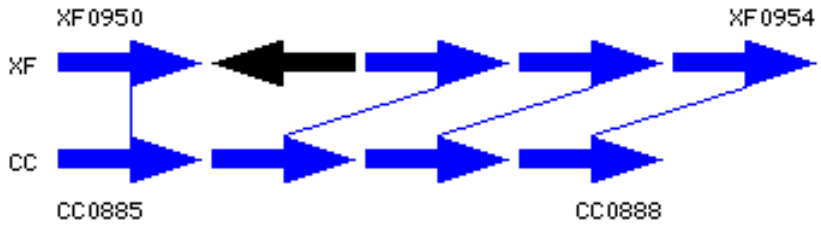

Figura 5.13: Visualização gráfica da mesma RO da figura 5.12, extraída da saída gráfica de EGG.

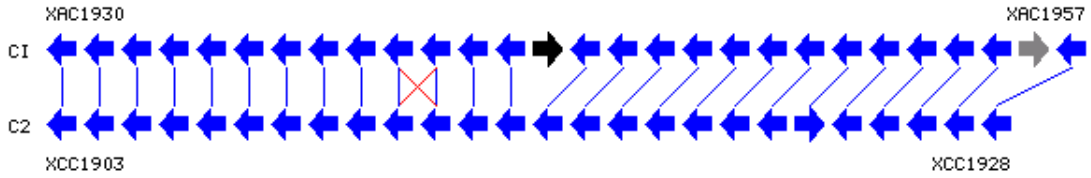

Figura 5.14: Visualização gráfica de uma RO encontrada na comparação das *Xanthomonas*. Os candidatos a genes falsos são aqueles de `Xac` que não possuem matches na região.

5.4.2 Outros trabalhos

Nós agora vamos fazer um breve relato de trabalhos que também fazem uso da presença de regiões ortólogas, ou estruturas parecidas, na determinação de função de genes e caracterização de aspectos ligados à funcionalidade dos genomas.

Tatusov e colegas [66, 67, 68] desenvolveram o que chamaram de `COG`, *Cluster of Orthologous Groups*. A idéia consiste na criação de grupos de proteínas que supostamente evoluíram a partir de um anscestral comum. Cada grupo é identificado a partir da comparação, no estilo todas-contra-todas, de proteínas de genomas completos. Além disso, é formado por pelo menos três proteínas de genomas distantes e cada par do grupo é um BBH.

Overbeek e outros [49, 50, 51] objetivam a reconstrução de vias metabólicas, a partir de estruturas organizacionais encontradas entre os genomas. De maneira similar à nossa definição de RO, eles usam a noção PCBBH, de *pair of close bidirectional best hit*, onde dois genes próximos (menos que 300 bases entre um e outro) de uma mesma fita formam BBHs com dois genes, também próximos, no outro genoma. Uma estrutura não tão exigente também é definida, que é o PCH, *pair of close homologs*, que corresponde a pares de genes respectivamente próximos em cada genoma, mas que formam apenas matches entre si.

Basicamente eles usam essas estruturas e valores a elas atribuídos para a construção de grupos de genes de vários genomas, a partir de um gene sabidamente pertencente a uma via metabólica. Ao final, o grupo obtido serve como proposta de conjunto de genes que pertencem a uma via metabólica.

Nosso trabalho é bem menos abrangente, no sentido de não envolver aspectos relacionados às vias metabólicas, além de envolver apenas dois genomas no conceito de RO. Por outro lado nosso objetivo é diferente, já que estamos interessados em propor regiões que possam indicar a existência de relacionamento e, por conseqüência, de funcionalidade comum entre dois genomas. Por isso não podemos ser tão exigentes na construção da RO.

Fujibuchi, Ogata e outros [26] propõem uma forma de encontrar regiões ortólogas (eles cha-

mam tais regiões de *clusters*), também baseada na comparação entre os genes de dois genomas, mas usando para isso uma implementação do algoritmo de Smith-Waterman, ao invés do BLAST, como fazemos na comparação entre os genes.

Os clusters encontrados em várias comparações são então comparados através de um algoritmo de comparação de grafos, baseado na sobreposição de vértices, proposto por Ogata e outros em [48]. Esse algoritmo, aliás, foi inicialmente proposto com o objetivo de se sobrepor grafos de genes com grafos de etapas de vias metabólicas, visando a identificação de vias metabólicas que usam genes de uma mesma região.

Esses dois trabalhos são parte importante de KEGG, *Kyoto Encyclopedia of Genes and Genomes*, uma base de dados envolvendo genes, genomas e vias metabólicas, desenvolvido por Kanehisa e Goto em [35].

Ogata e outros [47] seqüenciaram o genoma de *Rickettsia conorii*, e ao mesmo tempo compararam com o genoma já seqüenciado de *Rickettsia prowazekii*. Nessa comparação eles sugerem dois aspectos que são interessantes na comparação de dois genomas e que não foram abordados pela nossa metodologia. O primeiro deles tem a ver com o que eles chamam de *resquícios de genes*, que são trechos sem ORFs num genoma que tem alta homologia com um gene no outro genoma. Ambos (o trecho num genoma e o gene no outro genoma) estão interpostos por dois matches. O outro aspecto tem a ver com o que eles chamam de *gene splitting*, que ocorre quando uma região codante codifica duas ou mais proteínas, e essa região tem alta similaridade com uma única proteína no outro genoma.

Suyama e Bork [63] fazem um pequeno estudo de como a conservação de ordem dos genes se preserva em 21 pares de genomas filogeneticamente próximos (distância *16S rRNA* menor que 0.13). Nesses casos, eles concluíram que há uma forte tendência no aumento de conservação, à medida em que a distância filogenética diminui, o que não nos surpreeende. Eles também usam, para efeito de definição de ortólogos, BBHs com e-value de 10^{-4} e cobertura mínima de alinhamento de 80%. Mesmo tipo de análise foi feita por Bansal e outros [8, 9], só que envolvendo um grupo menor de genomas.

Huynen e colegas [34] propõem três aspectos relacionados aos genes de um genoma, como ferramentas auxiliares na identificação da funcionalidade de genes. Esses aspectos devem, obviamente, complementar aquele que envolve a comparação direta de proteínas. São eles: fusão de genes; conservação de ordem; e co-ocorrência de genes em outros genomas. Os autores fazem uma análise quantitativa dos genes da bactéria *Mycoplasma genitalium*, levando em conta os três aspectos considerados. Para efeitos de critério de homologia, eles consideram como ortólogos genes pareados como BBHs com alta similaridade, usando Smith-Waterman.

Snel e outros [62] propõem uma plataforma, denominada `STRING`, *Search Tool for Recurring Instances of Neighbouring Genes*, que objetiva mostrar graficamente como são as regiões

onde um determinado gene ocorre repetidamente em vários genomas. Trata-se de uma implementação das idéias apresentadas por Huynen e colegas [34]. A idéia algorítmica é a de usar o próprio gene de entrada como uma semente para a busca. Caso esse gene não esteja em nenhuma região da base, um gene ortólogo a ele (usando similaridade a partir do algoritmo de Smith-Waterman) é usado como semente. A cada iteração, os genes encontrados como ortólogos a ele são usados como sementes, e assim sucessivamente.

Capítulo 6

Conclusões e perspectivas

Este trabalho teve como principal resultado um conjunto de metodologias para a comparação de genomas, tanto no nível de DNA, quanto no nível de seus genes, e a implementação dessas metodologias.

A metodologia proposta para a comparação de genomas no nível de DNA utiliza, como principal ferramenta, uma árvore de sufixos para as seqüências genômicas. A principal contribuição está no uso desse tipo de árvore, essencialmente preparada para lidar com repetições exatas, na busca por repetições aproximadas envolvendo os genomas.

Na nível de proteínas, propomos uma metodologia para a comparação dos proteomas de dois genomas. A principal contribuição neste caso é a determinação de estruturas organizacionais envolvendo os genes dos genomas que apresentam conservação de ordem e funcionalidade.

Além disso, propomos uma metodologia para determinar a espinha dorsal de dois proteomas, na forma de um alinhamento, assim como uma metodologia para encontrar as famílias de genes parálogos de um genoma.

Propomos também um algoritmo adaptado da programação dinâmica usual para comparação de proteínas, que faz uso das chamadas regiões curingas.

Como resultado, as metodologias propostas, assim como os programas, foram utilizados nos projetos genoma das bactérias *Xylella fastidiosa* [25], *Xanthomonas axonopodis* pv. *citri* e *Xanthomonas campestris* pv. *campestris* [16] e *Agrobacterium tumefaciens* [73, 74]. Além disso, nosso trabalho resultou nas publicações [4, 59] e co-autoria em [16, 73, 74].

As ferramentas computacionais desenvolvidas a partir das metodologias foram:

- BACON: programa que compara dois genomas, no nível de DNA e que devolve repetições aproximadas que ocorrem em dois genomas, repetições tandem exatas, diferenças sin-

gulares e blocos de inserção e remoção;

- SBACON, que devolve as repetições aproximadas num genoma;
- EGG, que compara dois genomas no nível de seus genes, e retorna os pares de genes ortólogos, além de regiões ortólogas.

EGG pode ser usado também na comparação de um proteoma com ele mesmo, gerando assim famílias de genes parálogos.

Os programas estão disponíveis, incluindo manuais e códigos-fonte, no endereço

`http://www.dct.ufms.br/~nalvo/baconegg/`.

Além disso, alguns resultados de várias comparações podem ser vistos no mesmo website.

Trabalhos futuros

Entendemos que muita coisa ainda pode ser feita para melhorar as metodologias e ferramentas aqui propostas e o detalhamento dos seus resultados. Listamos a seguir algumas dessas possibilidades.

Interface gráfica para o BACON

Os arquivos de saída de BACON e SBACON são todos textuais e muitas vezes de difícil compreensão. Uma interface gráfica para os resultados obtidos, principalmente aqueles envolvendo os grandes blocos de repetição, deve ser muito útil.

Uma possibilidade seria a de mostrar as repetições da mesma maneira feita nos gráficos gerados por EGG. Mas isto é muito pouco. A idéia aqui é a de se obter uma maneira mais elegante e prática de mostrar as repetições em larga escala, com perspectivas para escalas menores, com *zoom's*, etc.

Integração entre BACON e EGG

Apesar da grande diferença existente entre os resultados fornecidos por BACON e por EGG, pode-se imaginar algum tipo de confronto entre essas informações. Por exemplo, podemos imaginar que as três grandes inversões encontradas entre as duas *Xanthomonas* sejam detectadas também por BACON, através da aparição de pequenos blocos de repetição de DNA agrupados.

Medidas de distância e filogenia

Alguns dos números retornados por EGG podem ser vistos como possíveis medidas de distância entre os genomas comparados, como por exemplo, número de matches e BBHs, ROs, etc. Uma forma de testar essas medidas é usando-as em algoritmos conhecidos para filogenia baseados em distância.

Fizemos alguns testes, considerando algumas medidas fornecidas por EGG. Para citar como exemplo, usamos como medida de distância uma função baseada no número de BBHs dividido pelo número de genes do menor genoma. Usamos o pacote `PHYLIP` [21, 22] de inferência de filogenia, para a construção da árvore. A figura 6.1 mostra a árvore resultante.

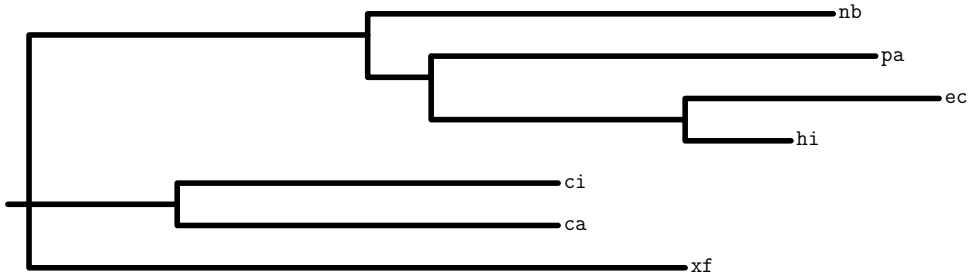

Figura 6.1: Árvore filogenética construída a partir das distâncias baseadas no número de BBHs e no número máximo possível de BBHs entre os genomas.

Construímos também uma árvore a partir de seqüências de *16S rRNA* dos mesmos organismos, armazenadas no Genbank [10], usando o mesmo pacote, com mesmos parâmetros e obtivemos uma árvore com a mesma topologia, o que nos dá indícios de que o número de BBHs pode ser uma boa medida de distância.

Entretanto, se por um lado EGG nos fornece um conjunto de potenciais medidas de distância, a partir de matches, BBHs, runs e ROs, por outro lado sabemos que o conjunto de informações obtidas com as comparações de proteomas é muito mais rico do que simplesmente um conjunto de medidas de distância. Assim, deveríamos ser capazes de criarmos um método para inferir filogenia que envolvesse todo esse conjunto de informações, incluindo informações referentes ao rearranjos, aplicadas a algoritmos baseados em características, como feito no trabalho de Gallut e colegas [27].

Alinhamento múltiplo de genomas

O trabalho feito por Fujibuchi e outros [26] propõe uma tentativa de alinhamento múltiplo de genomas, baseada no grafo obtido pela sobreposição dos grafos referentes à cada comparação de genomas dois-a-dois. Ou seja, eles propõem um alinhamento múltiplo dos genes dos genomas, levando-se em conta suas respectivas posições.

Em se tratando de genomas próximos, talvez sejamos capazes de determinar alinhamentos múltiplos que levem em conta não somente os genes ortólogos, mas também as ocorrências múltiplas das estruturas que estudamos neste trabalho, como BBHs, runs e ROs. O trabalhos de Bork e colegas [13] e Snel e outros [62] propõem algumas abordagens para esse problema.

Banco de dados de genomas, genes e ROs

Uma pergunta interessante é: seja C o conjunto de genes do genoma G que participam de uma dada RO: em quais outras comparações envolvendo G o conjunto C aparece? Em outras palavras, em quais outros genomas aquela RO aparece? Para responder a essa pergunta devemos reunir todas as informações de todas as comparações num mesmo ambiente. Uma saída seria a construção de um banco de dados de genes, genomas, matches, runs e ROs.

Resquícios de genes

Como vimos anteriormente, de acordo com o trabalho proposto por Ogata e outros [47], um resquício de gene é uma região não codante interposta por dois genes g_i e g_{i+1} de G tais que g_i e g_{i+1} fazem BBHs com h_j e h_{j+2} de H. Ou seja, supostamente haveria um gene g'_i entre g_i e g_{i+1}, pareando com h_{j+1}.

Uma forma de tentar captar esse tipo de situação seria comparando, via BLAST, e considerando todas as seis possibilidades de tradução, a seqüência não-codante com o gene h_{j+1}. Caso essa comparação fique acima de um certo valor de corte, então poderíamos inferir a existência de um tal gene.

Fusão/fissão

Uma tentativa de identificar os fenômenos de fusão/fissão de genes pode ser feita da seguinte forma. Quando dois genes consecutivos g_i e g_{i+1} do genoma G são tais que (g_i, h_j) e (g_{i+1}, h_j) são matches, podemos supor que houve uma fusão de g_i e g_{i+1} no gene h_j, ao mesmo tempo que houve uma fissão de h_j nos genes g_i e g_{i+1}.

Após EGG ter encontrado os matches, é muito fácil fazer uma lista contendo as fusões/fissões, de acordo com o descrito acima. Porém, esse método é muito impreciso. Uma possibilidade de aprimoramento seria uma análise mais precisa dos alinhamentos retornados pelo BLAST, para tentar captar detalhes que informam, por exemplo, o fato de uma proteína predita g_i ter duas partes, ${g'}_i$ e ${g''}_i$, e ${g'}_i$ ter se alinhado bem com h_j e e ${g''}_i$ ter se alinhado bem com h_{j+1}.

Neste caso, algumas condições poderiam ser verificadas na tentativa de validar a suspeita de fusão/fissão:

- a soma dos tamanhos de h_j e h_{j+1} é aproximadamente igual ao tamanho de g_i;
- g'_i e h_j são muito similares;
- g''_i e h_{j+1} são muito similares;
- h_j e h_{j+1} não têm qualquer similaridade;
- as regiões de similaridade de g'_i e h_j, e de g''_i e h_{j+1} devem ter pouca sobreposição.

Vias metabólicas

Um estudo envolvendo as ROs e as vias metabólicas parece promissor. É bem provável que os genes que participam de uma mesma via metabólica em mais de um genoma devam se organizar de forma semelhante nos genomas envolvidos.

A idéia então seria a de fazer um estudo para sabermos se os genes que fazem parte de uma via necessariamente estão dispostos da mesma forma nos diversos genomas que produzem aquela via. Além disso, pode-se verificar se algumas ROs encontradas podem, por esse motivo, serem consideradas boas pistas para possívies vias.

Implementação das regiões específicas

Parte da nossa metodologia de comparação de proteomas inclui a determinação de regiões específicas, conforme descrito na secão 5.1. EGG ainda não contém a implementação da estratégia proposta.

A figura 6.2 mostra uma região específica de *Xylella fastidiosa* em relação ao proteoma de *Xylella fastidiosa* (Pierce's disease). A existência dessa região, que foi encontrada a partir da espinha dorsal dos 2 proteomas, mostra que vale a pena a implementação dessa metodologia.

```
# -XF1728  1647681..1649024   transport protein
  -XF1729  1649021..1649896   phenylacetaldehyde dehydrogenase
  -XF1730  1650033..1650923   transcriptional regulator (LysR family
# +XF1731  1650909..1651088   hypothetical protein
# +XF1732  1651143..1652249   NAD(P)H-dependent 2-cyclohexen-1-one reductase
  +XF1733  1652388..1652966   tryptophan repressor binding protein
  +XF1734  1652981..1654027   NADP-alcohol dehydrogenase
# +XF1735  1654062..1655135   conserved hypothetical protein
# +XF1736  1655213..1655608   hypothetical protein
# -XF1737  1655625..1656353   conserved hypothetical protein
# -XF1738  1656364..1657689   hypothetical protein
  -XF1739  1657690..1658847   outer membrane protein
# -XF1740  1658850..1660067   glucose dehydrogenase B
  -XF1741  1660130..1661014   daunorubicin C-13 ketoreductase
  -XF1742  1661083..1662153   conserved hypothetical protein
  -XF1743  1662185..1663240   esterase
  -XF1744  1663245..1663988   oxidoreductase
# -XF1745  1664020..1665036   conserved hypothetical protein
  -XF1746  1665101..1666150   alcohol dehydrogenase
# -XF1747  1666171..1666812   conserved hypothetical protein
# -XF1748  1666809..1667522   5-amino-6-(5-phosphoribosylamino)uracil reductase
# -XF1749  1667542..1668771   transcriptional regulator
# -XF1750  1668885..1670021   conserved hypothetical protein
# -XF1751  1670047..1670550   hypothetical protein
  +XF1752  1670660..1671634   transcriptional regulator (LysR family
# -XF1753  1671676..1673526   hypothetical protein
# +XF1754  1673842..1674468   conserved hypothetical protein
# +XF1755  1674481..1675110   conserved hypothetical protein
# +XF1756  1675195..1675413   hypothetical protein
# -XF1757  1675818..1676078   hypothetical protein
# -XF1758  1676095..1676502   hypothetical protein
# -XF1759  1676603..1676926   conserved plasmid protein
# -XF1760  1677021..1677683   hypothetical protein
# -XF1761  1677741..1678868   hypothetical protein
# -XF1762  1678907..1679425   conserved hypothetical protein
# -XF1763  1679422..1680264   phage-related protein
# -XF1764  1680412..1680765   hypothetical protein
# -XF1765  1681958..1683376   drug:proton antiporter
# -XF1766  1683408..1684190   hypothetical protein
# +XF1767  1684178..1685248   hypothetical protein
  +XF1768  1685230..1686171   transcriptional regulator (LysR family
# +XF1769  1686247..1687188   hypothetical protein
# +XF1770  1687397..1688287   hypothetical protein
# -XF1771  1688250..1688642   hypothetical protein
# -XF1772  1688664..1689074   hypothetical protein
# -XF1773  1689210..1689968   hypothetical protein
  -XF1774  1690011..1694029   DNA methyltransferase
# +XF1775  1691893..1693602   reverse transcriptase
  -XF1776  1694565..1696424   DNA topoisomerase III
# +XF1777  1696611..1696781   hypothetical protein
# -XF1778  1696852..1697304   single-stranded DNA binding protein
# -XF1779  1697378..1697911   hypothetical protein
# -XF1780  1697908..1698657   hypothetical protein
# -XF1781  1698842..1700098   hypothetical protein
# -XF1782  1700095..1700655   conserved hypothetical protein
# -XF1783  1700671..1701579   hypothetical protein
# -XF1784  1701566..1702327   hypothetical protein
```

Figura 6.2: Região específica de *Xylella fastidiosa* com relação a *Xylella fastidiosa* (Pierce's disease). Cada gene marcado com # é um gene específico.

Apêndice A

Resultados adicionais das comparações de proteomas

Neste apêndice mostramos alguns resultados adicionais obtidos com a aplicação de EGG nos genomas que utilizamos.

As tabelas A.1, A.2 e A.3 trazem números de matches e runs obtidos em diversas comparações.

genoma1 × genoma2	**# total de matches**	**# total de BBHs**
Xfa × Xac	6889	1526
Xfa × Xcc	6509	1506
Xfa × Hin	2649	809
Xfa × Pae	9213	1235
Xfa × Eco	6104	1109
Xfa × Nma	2469	905
Xfa × Nmb	2531	895
Xfa × Ccr	5583	1028

Tabela A.1: Números de matches e BBHs das comparações de *Xylella* com outros genomas.

genoma1 × genoma2	# total de matches	# total de BBHs
Mla × At1	1527	96
Mla × At2	2549	96
Mla × At3	611	62
Mla × At4	232	51
Mlb × At1	332	58
Mlb × At2	310	47
Mlb × At3	141	48
Mlb × At4	58	29
Mlc × At1	28607	2016
Mlc × At2	38541	1039
Mlc × At3	9967	314
Mlc × At4	2374	87

Tabela A.2: Número de matches e BBHs das comparações de *Agrobacterium* com *Mesorhizobium*.

genoma1 × genoma2	# total de matches	# total de BBHs
Sma × At1	6622	382
Sma × At2	10255	398
Sma × At3	2867	210
Sma × At4	683	67
Smb × At1	9731	488
Smb × At2	15588	553
Smb × At3	3862	198
Smb × At4	878	47
Smc × At1	14839	1892
Smc × At2	18269	766
Smc × At3	4644	222
Smc × At4	1095	70

Tabela A.3: Número de matches e BBHs das comparações de *Agrobacterium* com *Sinorhizobium*.

A figura A.1 mostra os BBHs entre os genomas de *Xylella fastidiosa* e *Haemophilus influen-*

zae. O gráfico serve para mostrar a falta de conservação de ordem entre os dois organismos.

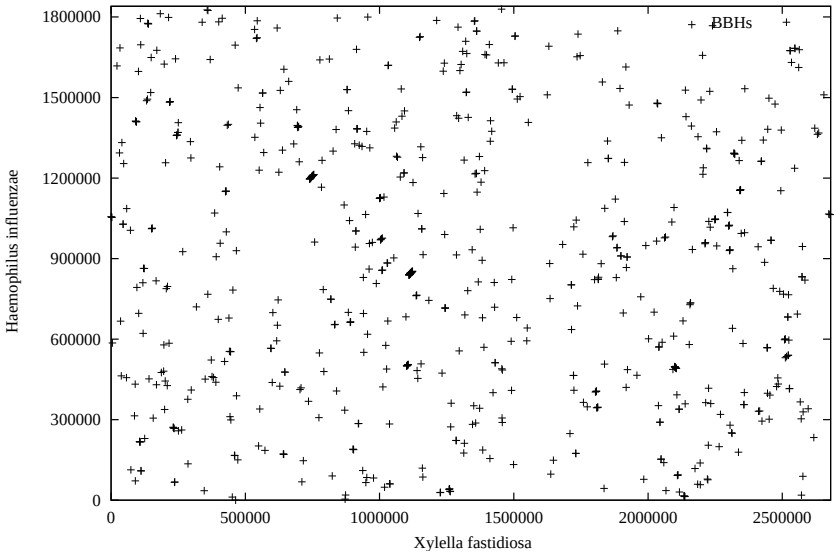

Figura A.1: BBHs entre os genomas *Xylella fastidiosa* e *Haemophilus influenzae*.

As tabelas A.4, A.5, e A.6 trazem números sobre runs e ROs.

g1 × g2	**# total de runs**	**# de runs com 4+ matches e 3+ BBHs**	**# total # de # ROs**	**total de kb's nas ROs**	
				g1	**g2**
`Xfa` × `Xac`	612	174	169	1152	1364
`Xfa` × `Xcc`	574	169	170	1140	1149
`Xfa` × `Hin`	167	37	29	121	115
`Xfa` × `Pae`	636	81	96	373	384
`Xfa` × `Eco`	408	57	75	284	284
`Xfa` × `Nma`	155	34	30	113	122
`Xfa` × `Nmb`	161	34	31	147	160
`Xfa` × `Ccr`	287	37	46	82	110

Tabela A.4: Números de runs e ROs das comparações de *Xylella* com outros genomas.

g1 × g2	# total de runs	# de runs com 4+ matches e 3+ BBHs	# total # de # ROs	total de kb's nas ROs g1	g2
Xac × Xcc	1838	329	213	3888	4197
Xac × Hin	249	40	49	178	141
Xac × Pae	1898	128	274	758	753
Xac × Eco	1099	69	180	485	428
Xac × Nma	218	42	38	124	149
Xac × Nmb	228	41	40	150	168
Xac × Ccr	907	48	133	244	240

Tabela A.5: Números de runs e ROs entre *Xanthomonas citri* e outros genomas.

genoma1 × genoma2	# total de runs	# de runs com 4+ matches e 3+ BBHs	# total de ROs
Sma × At1	528	8	138
Sma × Aa2	10255	20	362
Sma × Aa3	250	12	84
Sma × Aa4	60	6	21
Smb × At1	845	21	215
Smb × At2	1694	40	560
Smb × At3	384	8	121
Smb × At4	80	0	25
Smc × At1	1179	217	294
Smc × At2	1582	62	474
Smc × At3	352	5	98
Smc × At4	85	3	28

Tabela A.6: Números de runs e regiões das comparações de *Agrobacterium* com *Sinorhizobium*.

A figura A.2 mostra uma grande RO encontrada na comparação das *xanthomonas*.

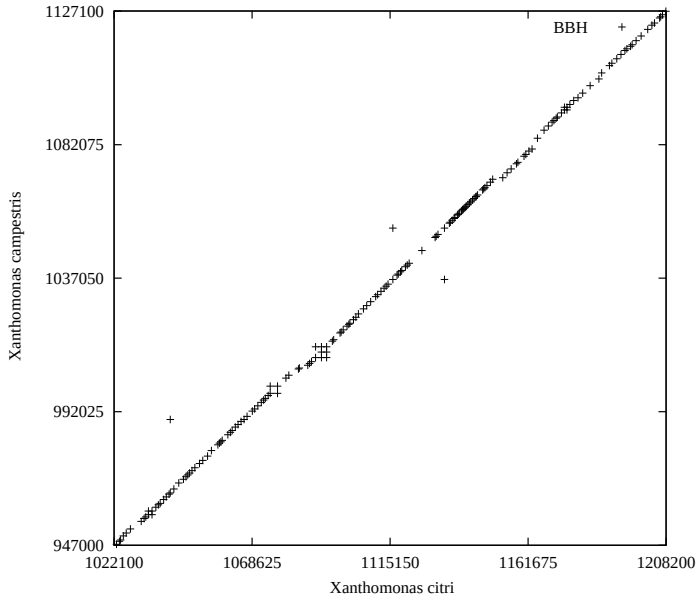

Figura A.2: RO encontrada entre os genomas `ci` e `ca`.

A figura A.3 mostra parte de uma das regiões com alto grau de conservação de ordem entre o replicon `At2` de *Agrobacterium tumefaciens* e o replicon `Smc` de *S. meliloti*.

```
================================================================================
Gene_id    start   size e-value [    best hit    ] product
================================================================
+SMc00682  2893494 387   1e-159 [best            ] PUTATIVE HIPPURATE HYDROLASE PROTEIN
+Atu3635   687381  386   1e-159 [best            ] amidohydrolase
-------------------------------------
-SMc00683  2892160 360   1e-129 [best            ] PUTATIVE PENICILLIN-BINDING PROTEIN
-Atu3634   686102  388   1e-129 [best            ] penicillin-binding protein dacf precursor
-------------------------------------
-SMc00685  2891001  46    4e-15 [best            ] HYPOTHETICAL PROTEIN
-Atu3633   684790  410    1e-13 [-SMc00684/1e-157] conserved hypothetical protein
-------------------------------------
-SMc00684  2891001 381   1e-157 [best            ] CONSERVED HYPOTHETICAL PROTEIN
-Atu3633   684790  410   1e-157 [best            ] conserved hypothetical protein
-------------------------------------
+SMc00688  2889988  79    6e-23 [best            ] CONSERVED HYPOTHETICAL PROTEIN
+Atu3632   684327   77    1e-22 [best            ] conserved hypothetical protein
-------------------------------------
+SMc00689  2888546 480   1e-129 [best            ] HYPOTHETICAL LIPOPROTEIN
+Atu3631   683191  378   1e-129 [best            ] conserved hypothetical protein
-------------------------------------
+SMc00690  2887348 317   1e-153 [best            ] PROBABLE ACETYL-COENZYME A CARBOXYLASE CARBOXYL TRANSFERASE SUBUNIT A
+Atu3630   681785  316   1e-153 [best            ] acetyl-CoA carboxylase carboxyl transferase, alpha subunit
-------------------------------------
+SMc00691  2886349 311   1e-121 [best            ] PROBABLE INTEGRASE/RECOMBINASE DNA RECOMBINATION PROTEIN
+Atu3629   680726  330   1e-121 [best            ] site-specific recombinase
-------------------------------------
+SMc00692  2886203  49    4e-14 [best            ] HYPOTHETICAL SIGNAL PEPTIDE PROTEIN
+Atu3628   680556   56    7e-14 [best            ] conserved hypothetical protein
-------------------------------------
-SMc00695  2884678 192    1e-64 [best            ] PUTATIVE SHIKIMATE KINASE I PROTEIN
-Atu3626   679790  162    1e-64 [best            ] shikimate kinase
-------------------------------------
-SMc00696  2883540 377   1e-173 [best            ] PUTATIVE 3-DEHYDROQUINATE SYNTHASE TRANSMEMBRANE PROTEIN
-Atu3625   678651  362   1e-173 [best            ] 3-dehydroquinate synthase
-------------------------------------
+SMc00698  2881681  93    1e-34 [best            ] PUTATIVE TRANSCRIPTION REGULATOR PROTEIN
+Atu3624   678347   92    2e-34 [best            ] stress induced morphogen
-------------------------------------
-SMc00699  2880906 182    2e-82 [best            ] CONSERVED HYPOTHETICAL PROTEIN
-Atu3623   677600  210    5e-82 [best            ] molecular chaperone, DnaJ family
-------------------------------------
-SMc00700  2879835 331   1e-179 [best            ] PROBABLE COBALAMIN BIOSYNTHESIS PROTEIN
-Atu3622   676542  337   1e-178 [best            ] cobalt insertion protein
-------------------------------------
-SMc00701  2877890 631         0 [best            ] PROBABLE COBALAMIN BIOSYNTHESIS PROTEIN
-Atu3621   674526  638         0 [best            ] cobyrinic acid synthase
-------------------------------------
-SMc00702  2876586 330   1e-104 [best            ] HYPOTHETICAL SIGNAL PEPTIDE PROTEIN
-Atu3620   673511  340   1e-104 [best            ] conserved hypothetical protein
-------------------------------------
+SMc00703  2875865 212    3e-84 [best            ] HYPOTHETICAL TRANSMEMBRANE PROTEIN
+Atu3618   672419  211    5e-84 [best            ] conserved hypothetical protein
-------------------------------------
+SMc00704  2875439  96    6e-46 [best            ] PROBABLE 50S RIBOSOMAL PROTEIN L28
+Atu3617   671963   95    1e-45 [best            ] 50S ribosomal protein L28
-------------------------------------
-SMc00705  2874231 267    6e-74 [best            ] CONSERVED HYPOTHETICAL SIGNAL PEPTIDE PROTEIN
-Atu3616   670940  247    9e-74 [best            ] conserved hypothetical protein
-------------------------------------
+SMc00707  2872763 146    1e-53 [best            ] CONSERVED HYPOTHETICAL PROTEIN
+Atu3615   670388  145    2e-53 [best            ] conserved hypothetical protein
-------------------------------------
+SMc00708  2871991 256   1e-100 [best            ] PUTATIVE HYDROXYACYLGLUTATHIONE HYDROLASE (GLYOXALASE II) (GLX II) PR
+Atu3614   669618  255    2e-99 [best            ] glyoxalase II
-------------------------------------
-SMc00709  2871125 254   1e-118 [best            ] CONSERVED HYPOTHETICAL PROTEIN
-Atu3613   668740  252   1e-118 [best            ] conserved hypothetical protein
-------------------------------------
-SMc00710  2869894 368   1e-160 [best            ] PUTATIVE HISTIDINOL-PHOSPHATE AMINOTRANSFERASE PROTEIN
-Atu3612   667549  367   1e-159 [best            ] histidinol-phosphate aminotransferase
-------------------------------------
-SMc00711  2868971 307   1e-142 [best            ] PUTATIVE CYCLOHEXADIENYL DEHYDROGENASE AND ADH PREPHENATE DEHYDROGENA
-Atu3611   666620  308   1e-142 [best            ] prephenate dehydrogenase
-------------------------------------
-SMc00713  2867067 184    3e-61 [best            ] PUTATIVE CATION TRANSPORT PROTEIN
-Atu3610   664846  218    6e-61 [best            ] transporter
-------------------------------------
+SMc00714  2866275 260   1e-121 [best            ] PUTATIVE 1-ACYL-SN-GLYCEROL-3-PHOSPHATE ACYLTRANSFERASE (PLSC) PROTEI
+Atu3609   664070  265   1e-120 [best            ] 1-acyl-sn-glycerol-3-phosphate acyltransferase
-------------------------------------
```

Figura A.3: Região com alta conservação encontrada na comparação de `At2` e `Smc`.

O gráfico da figura A.4 mostra os números médios de matches e BBHs considerando as 15 comparações envolvendo *Xylella fastidiosa* e *Xanthomonas axonopodis pv. citri*, levando em conta as coberturas mínimas de 0%, 40%, 60% e 80%. O gráfico da figura A.5 mostra o equivalente para o número médio de runs e RCOs.

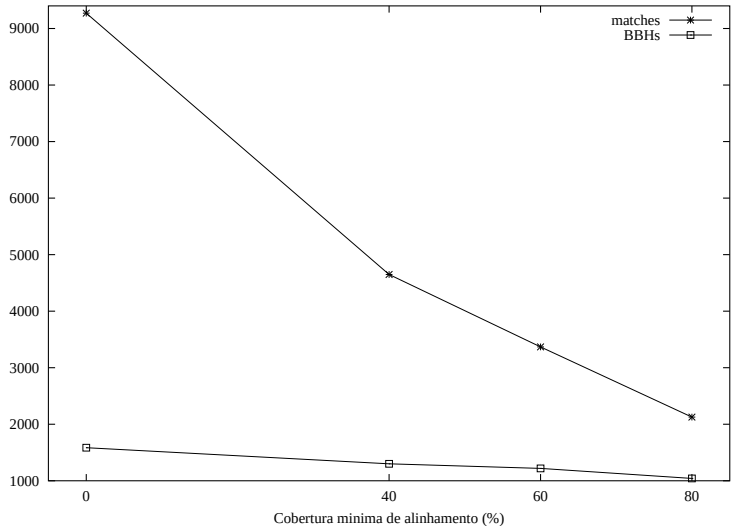

Figura A.4: Número médio de matches e BBHs, considerando as 15 comparações envolvendo `Xfas` e `Xac`, e alguns valores para a cobertura mínima de alinhamento.

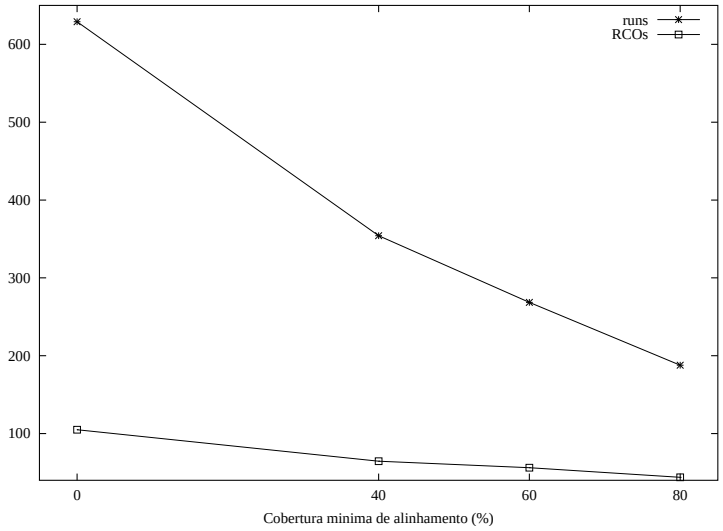

Figura A.5: Número médio de runs e RCOs, considerando as 15 comparações envolvendo `Xfas` e `Xac`, e alguns valores para a cobertura mínima de alinhamento.

Fizemos vários testes para descobrir como a exigência de cobertura mínima de alinhamento influencia na quantidade de matches e BBHs, e também na quantidade de runs e ROs. É óbvio que o aumento de exigência na cobertura diminui o número de ortólogos e das estruturas organizacionais. Porém, detectamos através dos gráficos que essa influência é quase que a mesma, tanto para matches e BBHs, quanto para as estruturas, o que nos diz que podemos calibrar os parâmetros de cobertura mínima olhando apenas para um dos casos.

Apêndice B

Detalhes operacionais de EGG

A implementação da metodologia proposta no capítulo 5 resultou no programa EGG. Alguns detalhes sobre o programa são descritos a seguir.

EGG pode ser obtido no website

`http://www.dct.ufms.br/~nalvo/baconegg/`,

onde também podem ser obtidos manuais para a completo uso de todos os módulos e vários exemplos de comparações de proteomas.

B.1 Descrição da ferramenta

Todos os programas envolvidos no pacote de EGG foram escritos em C++ usando o compilador GCC (GNU software) em ambiente Linux.

A entrada para EGG consiste nos conjuntos de genes do proteoma G e do proteoma H. Mais especificamente, EGG supõe a existência, para cada proteoma, do arquivo de extensão `ptt`, que traz informações sobre cada um dos seus genes, como coordenadas de DNA, fita, nome, produto, identificador no Genbank (`gi`), etc.; e do arquivo de extensão `faa`, que traz a seqüência, no formato FASTA, de todos os genes do proteoma. Ambos são disponíveis para genomas publicados e estão de acordo com a terminologia adotada pelo NCBI.

Como vimos na seção 5.2, EGG compara cada gene de um proteoma com todos os genes do outro proteoma usando BLAST (seção 2.3). Assim, precisamos, antes de executar EGG, construir uma base de seqüências para cada um dos proteomas. O pacote de BLAST possui um programa específico para esse fim, chamado `formatdb`. Após a construção da base de seqüências de gene de cada genoma, EGG pode ser executado.

Como dissemos na seção 5.2, por razões de desempenho, EGG é dividido em dois módulos. O primeiro é responsável pela comparação dos genes, todos-contra-todos, com o objetivo de encontrar, para cada gene de cada um dos dois genomas, os seus hits. O segundo módulo determina os matches, BBHs, runs e regiões ortólogas.

O primeiro módulo de EGG chamado de `egg_hits`, precisa dos seguintes arquivos de entrada, supondo que os nomes dos arquivos dos proteomas sejam, respectivamente, `genoma1` e `genoma2`, e supondo que o nome do arquivo de saída fornecido pelo usuário seja `g1g2`.

- `genoma1.ptt`;
- `genoma1.faa`;
- `genoma2.ptt`; e
- `genoma2.faa`.

Os arquivos de saída de `egg_hits` são:

- `g1g2.1` – mesmas informações de `genoma1.ptt`, num formato mais facilmente parseável;
- `g1g2.2` – mesmas informações de `genoma2.ptt`, num formato mais facilmente parseável;
- `g1g2.12` – para cada gene do `genoma1`, os hits encontrados e, para cada hit, o e-value, o score, e as porcentagens de cobertura do alinhamento; e
- `g1g2.21` – para cada gene do `genoma2`, os hits encontrados e, para cada hit, o e-value, o score, e as porcentagens de cobertura do alinhamento.

A figura B.1 mostra um trecho do arquivo `cica.12`, resultante da comparação de `Xac` e `Xcc`. A idéia é que esses arquivos sejam facilmente parseáveis, de tal modo que possam ser usados por quaisquer outros programas que usem esse tipo de informação.

```
# 6 542301 2
>  6 542311 399 1e-112 100 100
>  956 340011 73 2e-14 60 31
# 7 542001 1
>  7 542011 509 1e-145 100 100
# 8 541901 7
>  8 541911 154 6e-39 41 41
>  1570 1090511 60 2e-10 35 31
>  2052 91111 53 2e-08 39 62
>  2566 394011 52 7e-08 35 34
>  3157 156111 48 6e-07 36 18
>  3912 285711 39 0.0003 24 16
>  3730 2027011 39 0.0005 19 14
```

Figura B.1: Trecho do arquivo `cica.12`. Cada gene do `genoma1` é marcado com `#`, seguido de seu número seqüencial no genoma (de acordo com o arquivo `ptt`) e a quantidade de hits. Para cada um dos hits, marcados com o símbolo `>`, o número seqüencial (no `genoma2`) é mostrado, além do `gi`, score, e-value, e as porcentagens de cobertura do alinhamento da query e do hit.

O segundo módulo de EGG, denominado `egg_regs` tem como entrada os arquivos de saída de `egg_hits`, e como saída os seguintes arquivos, com as respectivas funcionalidades.

- `g1g2.k12` – para cada gene de `genoma1`, se teve ou não hits, (caso tenha hits) qual o e-value do melhor hit e em quais ROs aparece;
- `g1g2.k21` – para cada gene de `genoma2`, se teve ou não hits, (caso tenha hits) qual o e-value do melhor hit e em quais ROs aparece;
- `g1g2.exc` – genes exclusivos de cada proteoma;
- `g1g2.bes` – BBHs;
- `g1g2.rns` – runs;
- `g1g2.reg` – ROs;
- `g1g2.pdf` – gráfico mostrando os BBHs; e
- `g1g2.esp` – espinha dorsal de `genoma1` e `genoma2`.

Dois arquivos adicionais são também gerados por `egg_reg`: `g1g2.ros` e `g1g2.html`. O primeiro deles é binário e contém informações de todas as ROs encontradas. Juntos, eles servem para mostrar graficamente as regiões ortólogas, como na figura 5.4, por exemplo. A partir desses arquivos, páginas `html` são geradas automaticamente mostando as regiões. Para tanto, EGG também traz um programa chamado `readruns.cc`.

Famílias de parálogos

O programa para encontrar as famílias de genes parálogos tem dois módulos, como acima. O primeiro deles se chama `self_egg_hits` e tem como entrada os arquivos `genoma1.faa` e `genoma1.ptt`. Como saída gera dois arquivos: `g1g1.1` e `g1g1.12`.

O segundo módulo é denominado `egg_par` e gera apenas um arquivo como saída, denominado `g1g1.par`.

Na seção seguinte mostramos alguns exemplos de arquivos gerados pelo EGG.

B.2 Alguns exemplos dos arquivos de saída

A figura B.2 mostra um trecho do arquivo de saída com extensão `k12`, resultado da comparação entre os genomas das bactérias *Xanthomonas axonopodis pv. citri* e *Xanthomonas campestris pv. campestris*.

```
Gene     start product                                e-value [regions]
=====================================================================
CI5423.1  8552 conserved hypothetical protein     1e-112 [isolated]
CI5420.1  9636 conserved hypothetical protein     1e-145 [isolated]
CI5419.1 10983 TonB protein                       ===> NO HITS
CI5417.1 11740 biopolymer transport ExbB protein  1e-117 [1 ]
CI5414.1 12548 biopolymer transport ExbD1 protein 4e-75  [1 ]
CI5413.1 12974 biopolymer transport ExbD2 protein 8e-63  [1 ]
CI5411.1 13652 pyridoxal phosphate biosynthetic   1e-102 [1 ]
CI5410.1 14427 conserved hypothetical protein     2e-15  [1 ]
```

Figura B.2: Trecho do arquivo `cica.k12`, resultante da comparação entre `Xac` e `Xcc`. As colunas são, respectivamente, o número do gene (no genoma); a coordenada de início; o nome do produto; o e-value do melhor hit, caso o gene tenha algum; e as regiões homólogas das quais o gene faz parte. Esta útlima informação serve para que o usuário possa ir diretamente à RO e verificar como se comporta a vizinhança do gene.

A figura B.3 mostra um trecho do arquivo com extensão `bes`, resultante da comparação entre as *Xanthomonas*.

```
============================================================
Gene      gi     start..end  size  [category ] product
============================================================
+CI5435.1 543501    42..1367 441aa [III.A.1  ] chromosomal rep
+CA5435.1 543511    42..1367 441aa [III.A.1  ] chromosomal rep
------------------------------------------------------------
+CI5434.1 543401  1647..2744 365aa [III.A.1  ] DNA polymerase
+CA5434.1 543411  1646..2743 365aa [III.A.1  ] DNA polymerase
------------------------------------------------------------
```

Figura B.3: Trecho do arquivo `cica.bes`, resultante da comparação entre as *Xanthomonas*. As colunas mostram, da esquerda para a direita, a identificação do gene (no genoma); o identificador no Genbank; as coordenadas de DNA; o tamanho do gene; a categoria, caso esteja disponível; e o nome do produto;

A figura B.4 mostra como um run é mostrado textualmente, através do arquivo de saída com extensão `rns`.

```
>CICA020115-594-Pc
# of matches: 5
4kb in ci - 4kb in ca
=============================================================================================
Gene       start size value [best hit       ] [category] product
=============================================================================================
-CI7570.1  279076 157 1e-87 [best           ] [III.B.4 ] tRNA/rRNA methyltransferase
-CA7570.1  265558 157 1e-87 [best           ] [III.B.4 ] tRNA/rRNA methyltransferase
-------------------------------------
-CI15141.1 279579 152 1e-43 [best           ] [VIII.C  ] Xanthomonas conserved hypothetical
-CA15141.1 266089 151 5e-34 [best           ] [VIII.C  ] Xanthomonas conserved hypothetical
-------------------------------------
+CI7566.1  280436 101 1e-42 [best           ] [VIII.A  ] conserved hypothetical protein
+CA7566.1  266937 103 1e-44 [best           ] [VIII.A  ] conserved hypothetical protein
-------------------------------------
+CI7564.1  280846 337     0 [best           ] [II.E    ] 3-oxoacyl-[ACP] synthase III
+CA7564.1  267353 337     0 [best           ] [II.E    ] 3-oxoacyl-[ACP] synthase III
-------------------------------------
+CI7563.1  281974 388 1e-21 [-CA7563.1/4e-69] [VIII.A  ] conserved hypothetical protein
+CA20029.1 268442 386 8e-29 [best           ] [VIII.A  ] conserved hypothetical protein
=============================================================================================
```

Figura B.4: Trecho do arquivo `cica.rns`, resultante da comparação entre `Xac` e `Xcc`. Inicialmente o código do run a que os referimos no texto é mostrado; na linha seguinte aparece o número de matches do run e em seguida o tamanho aproximado (em número de bases) do run em cada genoma. Cada par mostrado é um match pertencente ao run. As colunas mostram, da esquerda para a direita, a fita e o nome do gene; a coordenada de início; o tamanho; o e-value do melhor hit; a palavra "best" na linha de cima do par diz que o gene de cima teve o gene de baixo como melhor hit, e o mesmo acontece para a palavra "best" na linha de baixo – caso contrário aparece (como no último match do run) qual o melhor hit do gene, com qual e-value; em seguida a categoria, caso disponível; e finalmente o produto.

Referências Bibliográficas

[1] R.A. Abagyan and S. Batalov. Do aligned sequences share the same fold? *J. Mol. Biology*, pages 355–368, 1997.

[2] T. Akutsu. Approximate string matching with variable length don't care characters. *IEICE Trans. Inf. & Syst.*, E79-D(9):1353–1354, September 1996.

[3] N.F. Almeida and J.C. Setubal. A set of tools for detailed syntatic pairwise comparison of whole bacterial genomes. manuscript, 2000.

[4] N.F. Almeida, J.C. Setubal, and M. Tompa. On the use of don't care regions for protein sequence alignment. Technical Report 99-07, Institute of Computing, University of Campinas, Brazil, 1999.

[5] S.F. Altschul, M.S. Boguski, W. Gish, and J.C. Wootton. Issues in searching molecular sequence databases. *Nature Genetics*, 6:119–129, 1994.

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

[7] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acid Research*, 25:3389–3402, 1997.

[8] A.K. Bansal. An automated comparative analysis of 17 complete microbial genomes. *Bioinformatics*, 15(11):900–908, 1999.

[9] A.K. Bansal, P. Bork, and P.J. Stuckey. Automated pai-wise comparisons of microbial genomes. *Math. Modeling and Sci. Computing*, 9:1–23, 1998.

[10] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, B. Rapp, and D. Wheeler. Genbank. *Nucleic Acids Res.*, 28(1):15–18, 2000.

[11] G. Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, 27(2):573–580, 1999.

[12] A. Bernal, U. Ear, and N. Kyrpides. Genomes online database (GOLD): a monitor of genome projects world-wide. *Nucleic Acids Research*, 29:126–127, 2001.

[13] P. Bork, B. Snel, G. Lehmann, M. Suyama, T. Dandekar, W. Lathe III, and M. Huynen. Comparative genome analysis: exploiting the context of genes to infer evolution and predict function. In *Comparative genomics*, pages 281–294. Kluwer Academic Publishers, 2000.

[14] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, Cambridge,MA/New York, 1990.

[15] K. Cummings and W. Klug. *Concepts of Genetics*. Prentice Hall, New Jersey, USA, 5^{th} edition, 1997.

[16] A.C. Rasera da Silva, J.C. Setubal, N.F. Almeida *et al.* Comparison of the genomes of two *Xanthomonas* pathogens with differing host specificities. *Nature*, 417(6887):459–463, 2002.

[17] M. Dayhoff, R. Schwartz, and B. Orcutt. A model of evolutionary change in proteins. In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 345–352. Natl. Biomed. Res. Found., 1978.

[18] M.V. de Souza, F.A. Torres, C.A. Ricart, W. Fontes, and M.A. Silva. *Gestão da vida?: genoma e pós-genoma*. Bluhm - UnB, 2001.

[19] A.L. Delcher, S. Kasif, R.D. Fleischmann, O. White J. Peterson, and S.L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[20] R.F. Doolittle. *Of URFs and ORFs: a primer on how to analyze derived amino acid sequences*. University Science Books, Mill Valley California, 1986.

[21] J. Felseinstein. Phylip – phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

[22] J. Felseinstein. Phylip (phylogeny inference package) version 3.5c. Distributed by the author, 1993. Department of Genetics, University of Washington, Seattle.

[23] D. Fischer and D. Eisenberg. Protein fold recognition using sequence-derived predictions. *Protein Science*, 5:947–955, 1996.

[24] D. Fischer, A. Elofsson, D. Rice, and D. Eisenberg. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In *Proc. Pacific Symposium on Biocomputing*, pages 300–318, January 1996.

[25] Organization for Nucleotide sequencing and analysis. The genome of *Xylella fastidiosa*, strain temecula 1, causal agent of Pierce's disease of grapevine. Paper in preparation.

[26] W. Fujibuchi, H. Ogata, H. Matsuda, and M. Kanehisa. Automatic detection of conserved gene clusters in multiple genomes by graph comparison and P-quasi grouping. *Nucleic Acids Research*, 28:4029–4036, 2000.

[27] C. Gallut, V. Barriel, and R. Vignes. Gene order and phylogenetic information. In *Comparative genomics*. Kluwe Academic Publishers, 2000.

[28] G. Gonnet, M. Cohen, and S. Benner. Exhaustive matching of the entire potein sequence database. *Science*, 256:1443–1445, June 1992.

[29] P. Green. Documentation for the `cross_match` program. Available upon request at http://www.phrap.org.

[30] P. Green. Genome sequence analysis. Lecture Notes - University of Washington, October 1996. (web site: http://www.genome.washington.edu/MBT599C/).

[31] M. Gribskov, A.D. McLachlan, and D. Eisenberg. Profile analysis: detection of distantly related proteins. In *PNAS*, volume 84, pages 4355–4358, 1987.

[32] D. Gusfield. *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology*. Cambridge University Press, 1997.

[33] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. In *Proc. Natl. Acad. Sci. USA*, volume 89, pages 10915–10919, 1992.

[34] M. Huynen, B. Snel, W. Lathe III, and P. Bork. Predicting protein function by genomic context: quantitative evaluation and qualitative inferences. *Genome Research*, 10:1204–1210, 2000.

[35] M. Kanehisa and S. Goto. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res.*, 28:27–30, 2000.

[36] S. Karlin. Assessing inhomogeneities in bacterial long genomic sequences. In *RECOMB'97*, pages 164–171, Santa Fe, New Mexico USA, 1997.

[37] S. Kurtz, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich. Computation and visualization of degenerate repeats in complete genoemes. In *8th international conference on intelligence systems for molecular biology – ISMB*, UC San Diego, California, August 2000.

[38] S. Kurtz and C. Schleiermacher. REPuter: Fast computation of maximal repeats in complete genomes. *Bioinformatics*, 15(5):426–427, 1999.

[39] N. Kyrpides. Genomes online database (GOLD): a monitor of complete and ongoing genome projects world wide. *Bioinformatics*, 15:773–774, 1999.

[40] A. Lehninger. *Bioquímica*, volume 4. Edgard Blücher, 1977.

[41] A. Lesk, M. Levitt, and C. Chothia. Alignment of the amino acid sequences of distantly related proteins using variable gap penalties. *Protein Engineering*, 1(1):77–78, 1986.

[42] M. Leung, B.E. Blaisdell, C. Burge, and S. Karlin. An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *J. Mol. Biology*, 221:1367–1378, 1991.

[43] B. Lewin. *Genes V.* Oxford University Press, Oxford, 1994.

[44] E.M. McCreight. A space-economical suffix tree construction algorithm. *J.ACM*, 23:262–272, 1976.

[45] S. Needleman and C. Wunsch. A general method applicable to the search for similarity in the amino acid sequence of two proteins. *Journal of Moleular Biology*, 48:443–453, 1970.

[46] Nomenclature Comittee of the International Union of Biochemistry NC-IUB. Nomenclature for incompletely specified bases in nucleic acid sequences– recomendations 1984. *Eur. J. Biochem.*, 150:1–5, 1985.

[47] H. Ogata, S. Audic, P. Renesto-Audiffren, P. Fournier, V. Barbe, D. Samson, V. Roux, P. Cossart, J. Weissenbach, J. Claverie, and D. Raoult. Mechanisms of evolution in Rickettsia conorii and R. prowazekii. *Science*, 293:2093–2098, September 2001. Reports.

[48] H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Research*, 28:4021–4028, 2000.

[49] R. Overbeek, M. Fonstein, M. D'Souza, G. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *PNAS*, 96:2896–2901, 1999.

[50] R. Overbeek, M. Fonstein, M. D'Souza, G.D. Pusch, and N. Maltsev. Use of contiguity on the chromosome to predict functional coupling. *In Silico Biology*, 1, 1998.

[51] R. Overbeek *et al.* http://wit.mcs.anl.gov/wit2/. web site.

[52] J.D. Parsons. Miropeats: graphical dna sequence comparisons. *Bioinformatics*, 11(6):615–619, 1995.

[53] W.R. Pearson. Comparison of methods for searching protein sequence databases. *Protein Science*, 4:1145–1160, 1995.

[54] P.A. Pevzner. *Computational Molecular Biology*. MIT Press, 2000.

[55] F.M. Richards. The protein fold problem. *Scientific american*, 1991.

[56] S.L. Salzberg, D.B. Searls, and S. Kasif, editors. *Computational Methods in Molecular Biology*, volume 32 of *New comprehensive biochemistry*. Elsevier, Netherlands, 1998.

[57] D. Sankoff and J.H. Nadeau, editors. *Comparative genomics*. Kluwer Academic Publishers, 2000.

[58] R. Schwartz and M. Dayhoff. Matrices for detecting distant relationships. In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 353–358. Natl. Biomed. Res. Found., 1978.

[59] J.C. Setubal and N.F. Almeida. Detection of related genes in procaryotes using syntenic regions. In *DIMACS Workshop on Whole Genome Comparison*. DIMACS Center, Rutgers University, February 2001.

[60] J.C. Setubal and J. Meidanis. *Introduction to computational molecular biology*. PWS Publishing Co., 1997.

[61] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Moleular Biology*, 147:195–197, 1981.

[62] B. Snel, G. Lehmann, P. Bork, and M.A. Huynen. STRING: a web-server to retrieve and display the repeatedly occuring neighbourhood of a gene. *Nucleic Acids Research*, 28:3442–3444, 2000.

[63] M. Suyama and P. Bork. Evolution of prokaryotic gene order: genome rearrangements in closely related species. *Trends in Genetics*, 17(1):10–13, January 2001.

[64] J. Tamames. Evolution of gene order conservation in prokaryotes. *Genome Biology*, 2(6), 2001.

[65] J. Tamames, G. Casari, C. Ouzounis, and A. Valencia. Conserved clusters of functionally related genes in two bacterial genomes. *Journal of Molecular Evolution*, 44:66–73, 1997.

[66] R.L. Tatusov, M.Y. Galperin, D.A. Natale, and E.V. Koonin. The COG database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Research*, 28(1):33–36, 2000.

[67] R.L. Tatusov, E.V. Koonin, and D.J. Lipman. A genomic perspective on protein families. *Science*, 278(5338):631–637, 1997.

[68] R.L. Tatusov, D.A. Natale, I.V. Garkavtsev, T.A. Tatusova, U.T. Shankavaram, B.S. Rao, B. Kiryutim, M.Y. Galperin, N.D. Fedorova, and E.V. Koonin. The COG database: new developments in phylogenetic classification of proteins from complete genome. *Nucleic Acids Res*, 29(1):22–28, 2001.

[69] E. Ukkonen. On-line costruction of suffix-trees. *Algorithmica*, 14:249–260, 1995.

[70] P. Vincens, L. Buffat, C. André, J. Chevrolat, J. Boisvieux, and S. Hazout. A strategy for finding regions of similarity in complete genome sequences. *Bioinformatics*, 14(8):715–725, 1998.

[71] G. Vogt, P. Etzold, and P. Argos. An assessment of amino acid exchange matrices in aligning protein sequences: the twilight zone revisited. *Journal of Molecular Biology*, 249:816–831, 1995.

[72] P. Weiner. Linear pattern matching algorithms. In *Proc. of the 14th IEEE Symp. on Switching and Automata Theory*, pages 1–11, 1973.

[73] D.W. Wood, J.C. Setubal, N.F. Almeida *et al.* Sequencing and analysis of the *Agrobacterium tumefaciens* genome. In *10th Int'l congress on Molecular plant-microbe interactions*, 2001. Madison, WI (poster).

[74] D.W. Wood, J.C. Setubal, N.F. Almeida *et al.* The genome of *Agrobacterium tumefaciens*: insights into the evolution and evolution of a natural genetic enginner. *Science*, 294:2317–2323, December 2001.

[75] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning dna sequences. *J Comput Biol*, 7(1-2):203–214, Feb-Apr 2000.