Coordenação do Curso de Ciência da Computação

Universidade Estadual de Mato Grosso do Sul

Montagem de Fragmentos de DNA

Gabriel Rodrigues Monteiro

André Chastel Lima (Orientador)

Novembro de 2014

Montagem de Fragmentos de DNA

Gabriel Rodrigues Monteiro

Este exemplar corresponde à relação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Gabriel Rodrigues Monteiro e aprovada pela banca examinadora, como parte dos requisitos para a obtenção do título de bacharel em Ciência da Computação.

Dourados, 14 de Novembro de 2014

André Chastel Lima (Orientador)

Resumo

São visivelmente crescentes as pesquisas e interesses na área da Biologia Computacional. Desde o início do Projeto Genoma, cientistas do mundo todo vêm trabalhando com sequenciamento do DNA de diferentes tipos de organismos e reconhecem que não há possibilidade de trabalhar com sequenciamento sem métodos computacionais.

Isso acontece, pois a grande maioria das moléculas de DNA são bastante extensas, algumas delas alcançando em torno de 10^{11} pares de base. Em contrapartida, os métodos laboratoriais de sequenciamento não são capazes de ler moléculas tão extensas.

Diante deste problema, surge um dos grandes papéis da Biologia Computacional que é o sequenciamento e montagem dos fragmentos de DNA. O processo de montagem de fragmentos de DNA consiste em amplificar, sequenciar e, posteriormente, remontar as subsequências de forma a obter uma sequência o mais próximo possível da original.

Estaremos, ao longo do texto, apresentando as principais técnicas utilizadas na solução deste problema, bem como a implementação de uma delas, as estruturas de dados mais adequadas a cada tipo de situação também fazem parte do conteúdo aqui apresentado.

Sumário

Lista de Figuras

Capítulo 1

Introdução

São consideradas, neste projeto, as técnicas de sequenciamento, alinhamento e montagem de fragmentos de DNA. Algumas técnicas e implementações voltadas à pesquisa na área da biologia molecular computacional serão apresentadas ao longo do texto, bem como, estudo da teoria dos grafos e implementações dos problemas abordados.

Antes de dar início ao estudo das técnicas que podem solucionar o problema da montagem de fragmentos, devemos ter um breve conhecimento da estrutura do DNA e os principais termos aqui utilizados. O capítulo 2 foi escrito com este objetivo, nele encontram-se definições e demais informações necessárias para compreensão do restante do texto. Já no capítulo 3, é apresentada uma pesquisa relacionada ao contexto principal do trabalho, descrição e embasamento teórico sobre o problema da montagem de fragmentos de DNA. No Capítulo 4, por sua vez, é abordado o problema de alinhamento de várias sequências bem como a apresentação de algumas possíveis soluções. O capítulo 5 traz uma síntese do que foi implementado e os resultados obtidos a partir da implementação. Por último, o capítulo 6 encerra o trabalho mostrando uma análise de resultados e possíveis trabalhos futuros para melhorar e incrementar o que já foi feito até aqui.

1.1 Objetivo

1.1.1 Objetivos Gerais

Diante da importância do estudo da montagem de fragmentos de DNA e das inúmeras situações em que esta técnica pode ser aplicada, consideramos como objetivo geral para este projeto, encontrar soluções teóricas que auxiliem no desenvolvimento de aplicações envolvendo as etapas do método de montagem de fragmentos de DNA.

1.1.2 Objetivos Específicos

Dentre as diversas técnicas apresentadas ao longo deste projeto, nosso principal objetivo é implementar o algoritmo de agrupamento de várias sequências de DNA, buscando encontrar a Menor Supercadeia Comum que mais se assemelhe à sua correspondente original. Podemos destacar nossos objetivos, como:

- Elaborar estratégias no casamento de cadeias de DNA;
- Adaptar algoritmos relacionados ao problema em questão;
- Encontrar uma solução para o problema do alinhamento de múltiplo de cadeias de DNA buscando atingir o resultado ótimo de comparação entre os fragmentos;

1.2 Justificativa

O mapeamento genético de qualquer organismo representa fontes infinitas de informações detalhadas sobre ele, biologicamente falando. É através desta técnica que podemos encontrar similaridade entre indivíduos da mesma espécie e, até mesmo, encontrar e prever doenças hereditárias, e demais inúmeras aplicabilidades deste recurso.

Sabemos que os métodos laboratoriais não são suficientes para conseguir realizar, na maioria dos casos, essas tarefas e que métodos computacionais são necessários nesta etapa. Por não conseguirem fazer leitura de longas cadeias de DNA de uma só vez, os métodos laboratoriais contam com a ajuda da bioinformática para gerar fragmentos acessíveis aos leitores e remonta-los, posteriormente, com o auxílio de algumas técnicas de montagem.

É neste problema que focaremos nossa atenção durante o desenvolvimento deste projeto, visando chegar a um resultado da maneira mais eficiente possível.

1.3 Metodologia

A metodologia utilizada é composta por três etapas. Em primeiro momento, é feito um estudo bibliográfico buscando uma compreensão de assuntos importantes para dar continuidade ao projeto. Pesquisas na área da Biologia, Genética, Teoria dos Grafos, sempre buscando soluções conhecidas para o problema em questão. A segunda etapa, consiste em mapear o problema da MSC em teoria dos grafos, seus conceitos e aplicações para resolver o problema da Menor Supercadeia Comum.

Por último, foram colocados em prática, os conceitos e soluções para o problema. Para isso, foi utilizada a programação nas linguagens C e C++ onde foram implementados os conceitos de alocação dinâmica de memória, listas encadeadas, ponteiros, manipulação de arquivos e matrizes.

Conhecimentos de lógica de programação também fizeram-se necessários durante todo o processo implementação do sistema para montagem de fragmentos.

Capítulo 2

Fundamentos do DNA

É importante que, antes de começarmos a discutir sobre a montagem de fragmentos, tenhamos um conhecimento prévio, básico, da estrutura do DNA bem como, informações necessárias para abordar o problema em questão.

O DNA ou ácido desoxirribonucleico é uma molécula orgânica de fundamental importância para os organismos, uma vez que é responsável pelo armazenamento dos seus respectivos genes (trechos de DNA que codificam proteínas) e pelos fatores hereditários que transferem esta carga genética através das gerações.
[SETUBAL; MEIDANIS, 1997]

Uma molécula de DNA consiste em duas longas cadeias de nucleotídeos (compostos que atuam como fonte de energia, coenzimas e reguladores fisiológicos). Existem quatro subunidades de nucleotídeos e as duas cadeias unem-se entre as suas bases nitrogenadas.

As bases nitrogenadas dos nucleotídeos são Adenina, Citosina, Guanina e Timina que, são representadas por suas respectivas iniciais: A, C, G e T.

Os genes carregam a informação biológica que deve ser precisamente copiada e transmitida quando uma célula se divide para formar duas células-filhas. O DNA codifica a informação na ordem, ou sequência, dos nucleotídeos ao longo de cada fita. Cada base – A, C, G ou T – pode ser considerada como uma letra em um alfabeto de quatro letras que é utilizado para codificar as mensagens biológicas na estrutura química do DNA. Os organismos diferem um do outro porque as suas respectivas moléculas de DNA têm diferentes mensagens biológicas. [ZAHA, 2000].

O *complemento* de cada base segue a determinação da tabela 2.1.

Base	A	C	T	G
Complemento	T	G	A	C

Tabela 2.1: Bases e seus complementos

Nas duas fitas que formam o DNA, uma é o complemento reverso da outra, ou seja, a cada base de uma temos a base complementar correspondente na outra, além disso, as duas fitas se encontram em orientações opostas. Portanto, podemos obter uma das fitas complementando a outra e invertendo a orientação desta. Na figura 2.1 um exemplo fictício mostra um trecho de DNA. [CERQUEIRA, 2000]

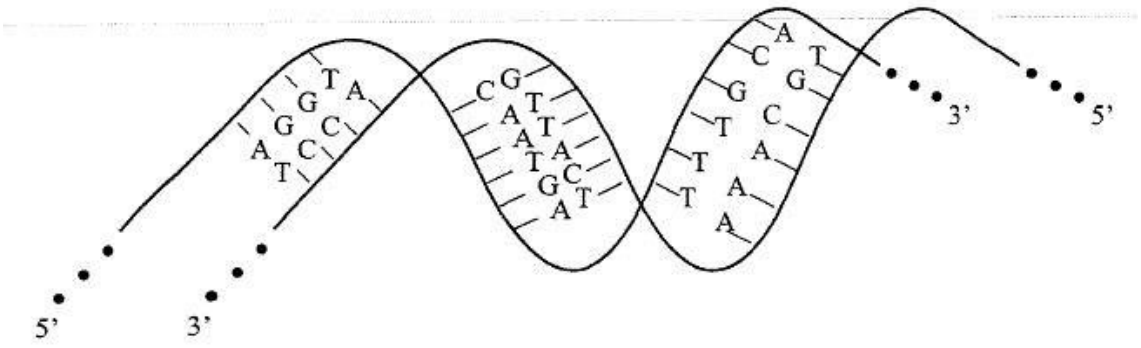


Figura 2.1: Fita dupla do DNA [CERQUEIRA, 2000]

O conjunto completo de informações no DNA de um organismo é chamado de genoma. [CERQUEIRA, 2000]

2.1 Sequenciamento

O sequenciamento de DNA é uma das principais áreas de estudo da Biologia Molecular Computacional que visa determinar a ordem das bases de uma sequência de DNA.

Segundo [BURKS, 1994], as informações codificadas no DNA podem servir, entre outras aplicações, de plataforma para aumentar nossa habilidade em: caracterizar e entender doenças e infecções humanas; projetar e desenvolver medicina terapêutica e preventiva; desenvolver fontes nutricionais melhoradas e empregar ferramentas microbiais em ambiente.

[SETUBAL; MEIDANIS, 1994] também afirmam que, se a sequência de um gene for conhecida, "pode-se, através do código genético, obter a sequência de aminoácidos da proteína correspondente, pode-se ainda fazer análises de

preferência de códons, verificar se há pontos de corte para enzimas de restrição conhecidas, comparar o mesmo gene de diferentes indivíduos para localizar diferenças (que podem indicar doenças hierárquicas) etc.".

O grande desafio dos métodos de sequenciamento empregados atualmente é de que consigam ser úteis em projetos que lidam com sequenciamento em larga escala, ou seja, sequenciamento de longas moléculas de DNA, com até mesmo milhões de pares de bases.

Nas últimas décadas, diversos métodos de sequenciamento de DNA foram propostos, os quais podem ser categorizados em métodos diretos e métodos indiretos de sequenciamento [DRMANAC *et al.*, 2001]. Dentre estes métodos, podemos destacar o Sequenciamento por *Shotgun*, Método de Sanger, Degradação Química, *Primer Walking*, *Nested Deletion*.

2.1.1 – Método *Shotgun*

O Método *Shotgun* consiste em sequenciar, randomicamente, o genoma completo para, posteriormente, ser montado em sequência contígua, o chamado *contig*. Esta técnica de sequenciamento é de extrema importância para aplicação desde genomas mais simples até os mais complexos, como o genoma humano, por exemplo.

Podemos classificar, hoje, o Método *Shotgun* como o método mais utilizado no sequenciamento de longas moléculas de DNA. Segundo [SANGER, *et al.*, 1982], ele tem as características positivas de ser *paralelizável*, *econômico* e *automatizável*.

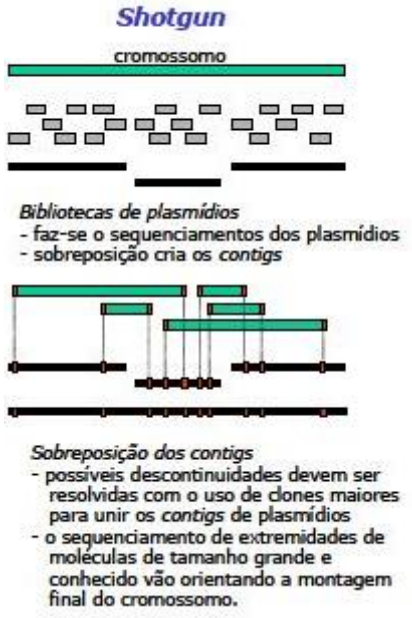


Figura 2.2: Sequenciamento por *Shotgun* [SANTOS].

O *Shotgun* é estruturado a partir das seguintes etapas:

- Fragmentação aleatória de diversas cópias do DNA – esta etapa consiste no processo de replicação ou clonagem da molécula de DNA, resultando em várias cópias quebradas em posições aleatórias de suas sequências. É importante que haja a replicação, devido à necessidade de pontos de sobreposição entre os fragmentos resultantes de cópias diferentes, ajudando na reconstrução da sequência;
- Seleção dos fragmentos – Dos fragmentos obtidos, nem todos podem ser reaproveitados. Fragmentos muito grandes ou, muito pequenos, são descartados;
- Clonagem dos Fragmentos selecionados – os fragmentos selecionados são em seguida clonados para que cada fragmento não seja único no experimento;
- Sequenciamento dos fragmentos – todos os fragmentos clonados são sequenciados por um processo de sequenciamento;
- Montagem de fragmentos – esta etapa consiste em reunir os fragmentos sequenciados de modo que se consiga deduzir a sequência original.

2.1.2 – Método de Sanger

Também conhecido como Método de Terminação de Cadeia de Sanger, possui uma estrutura um pouco mais simplificada que a do *Shotgun*, restringindo-se à apenas três etapas básicas.

Inicialmente, os fragmentos do DNA são gerados em diversos tamanhos, de modo que o nucleotídeo terminal de cada um deles possa ser facilmente identificado. A etapa seguinte trata da organização de todos os fragmentos por tamanho. E, por último, é feita a leitura adequada dos nucleotídeos terminais dos fragmentos fornecendo a sequência de nucleotídeos da sequência alvo.

Segundo [BAPTISTA, 2003], o Método de Sanger envolve, na prática, quatro reações bioquímicas, as quais têm por componentes cópias da molécula de DNA a ser sequenciada: iniciadores (*primers*), deoxinucleotídeos (dATP, dCTP, dGTP, dTTP), dideoxinucleotídeos (ddATP, ddCTP, ddGTP, ddTTP) e enzima DNA polimerase. Os iniciadores são pequenas sequências de nucleotídeos complementares ao trecho inicial da molécula de DNA. O deoxinucleotídeo e o dideoxinucleotídeo são componentes químicos que liberam os nucleotídeos que irão compor os fragmentos.

Esta técnica, apesar de ter sido criada há mais de 30 anos e, passado por evoluções que incluem processos automatizados, continua sendo bastante utilizada pelo fato de ser tecnicamente simples.

Em contrapartida, a grande desvantagem dessa técnica, segundo [SETUBAL; MEIDANIS, 1997], é que, mesmo com os avanços tecnológicos atuais, ela não pode ser empregada diretamente no sequenciamento de longas moléculas de DNA. A razão é que o tamanho máximo de sequenciamento é limitado pela qualidade da resolução do gel empregado na eletroforese, de modo que, na prática, o tamanho da maior sequência que se consegue determinar é de aproximadamente 700 bases.

2.1.3 – Método de Degradação Química

Trabalhando de maneira análoga ao Método de Sanger, o Método de Degradação Química baseia-se em decompor a sequência de DNA em quatro

conjuntos de fragmentos de acordo com as bases terminais que possuem. O que difere esses métodos é o tipo das reações empregadas.

2.1.4 – *Primer Walking*

O Método *Primer Walking* pertence à categoria dos métodos indiretos de sequenciamento de DNA e, pode ser classificado como um método dirigido, ou seja, diferente do *Shotgun*, neste método os fragmentos são obtidos e sequenciados de acordo com a ordem que aparecem na sequência de DNA investigada. [BAPTISTA, 2003] define o funcionamento do *Primer Walking* baseando-se no uso de um iniciador adequado para o sequenciamento das *j* primeiras bases da sequência, onde *j* é o tamanho usualmente permitido pelos métodos diretos de sequenciamento. A primeira subsequência obtida, além de ser parte da resposta procurada, o seu trecho final serve para a confecção de um novo iniciador, o qual começará o sequenciamento das próximas *j* bases. Esse processo é repetido até que a sequência de DNA seja completamente conhecida. As abordagens dirigidas possuem duas grandes desvantagens. A primeira é que elas são essencialmente sequenciais, isto é, são não-*paralelizáveis*, e por conseguinte, lentas. A segunda está relacionada ao fato de que, se o processo for interrompido em algum lugar da sequência, o sequenciamento do trecho além daquele ponto ficará prejudicado.

2.1.5 – *Nested Deletion*

Assim como o *Primer Walking*, o *Nested Deletion* é um método indireto e dirigido de sequenciamento. Trabalha baseando-se em duas etapas que se repetem sucessivamente até que a sequência o DNA tenha sido determinada por completo. Na primeira etapa, o trecho inicial da sequência é determinado pela aplicação de um método direto de sequenciamento. Já na segunda, uma enzima é aplicada para remover o que foi sequenciado e, assim, expor um novo trecho inicial da sequência restante para que o processo recomece.

2.2 Montagem

Em problemas reais lidamos com moléculas grandes, no entanto os métodos laboratoriais de sequenciamento são viáveis somente para pequenos trechos do DNA que tenham tamanhos variando em torno de 700bp (*base pairs* ou pares de

base), ou seja, para ser possível sequenciar um DNA surge a necessidade de quebra-lo em vários pedaços que denominamos fragmentos. Estes fragmentos de tamanho reduzido são sequenciados e remontados, obtendo o que chamamos de consenso, que esperamos ser bem próximo à sequência de molécula que foi inicialmente fragmentada. Na prática, numa análise bastante simplificada, o genoma é dividido em pedaços menores, mas ainda grandes e esses são fragmentados em pedaços menores ainda que são sequenciados e montados obtendo um *contig* para cada pedaço grande surgido da primeira quebra. Para cada *contig* é gerada uma sequência consenso. Montamos então os consensos, resultando em um *contig* maior que culminará no consenso final. [CERQUEIRA, 2000]

Na figura 2.3 temos um exemplo para a fita de sequência ACTGGTCACATTT.

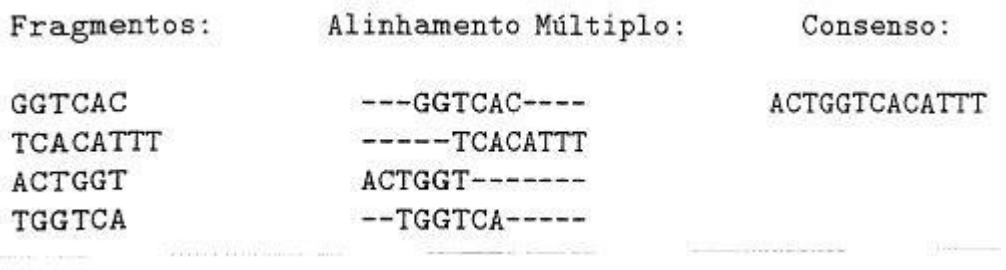


Figura 2.3: Exemplo de fragmentação e montagem [CERQUEIRA, 2000]

Um *contig* é representado pelo alinhamento múltiplo entre os fragmentos envolvidos e a partir deste alinhamento a sequência consenso poder ser obtida. Podemos observar que a caracterização de um alinhamento entre esses fragmentos é a inserção em suas sequências do que definiremos como espaços representados pelo caractere "-", de forma que as sequências fiquem com o mesmo tamanho e as partes que forem similares coincidam. Uma cadeia de espaços consecutivos será denominada de buraco. [CERQUEIRA, 2000]

Embora os tamanhos das subsequências da figura 2.3 não sejam realistas, elas representam o caso considerado ideal, onde os fragmentos cobrem inteiramente a molécula original, sobrepõem-se uns aos outros e estão livres de erros que podem ocorrer no processo de fragmentação e sequenciamento.

Esses casos, onde a sequência está livre de erros e imperfeições, são muito pouco prováveis de acontecer, pois existem alguns complicadores do processo de montagem de fragmentos apresentados a seguir.

Os principais fatores complicadores na montagem são:

1- Erros no *reads* introduzidos pelo sequenciador ou mesmo na duplicação do DNA, como fragmentos quiméricos ou contaminação com DNA dos vetores usados na duplicação do DNA estudado;

2- Regiões repetidas ou repetições são regiões que aparecem duas ou mais vezes na molécula original, podendo causar ambiguidades nas soluções;

3- Falta de cobertura acontece quando os *reads* não são suficientes para montar o consenso único do DNA, deixando buracos (*gaps*) na montagem por falta de *reads* para cobrir aquela região, formando *contigs* que não se conectam, como apresentado na Figura 2.4.

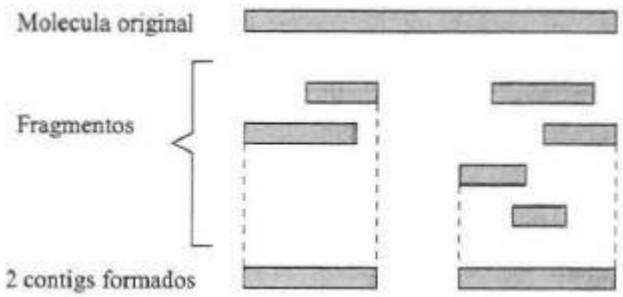


Figura 2.4: Problema da falta de cobertura. [LIN, 2001]

Capítulo 3

Agrupamento de Sequências

O agrupamento de sequências é uma das técnicas básicas da bioinformática que visa reconstruir uma sequência original de DNA a partir de seus fragmentos. Esta técnica é utilizada para reunir em uma só molécula virtual, as sequências obtidas das moléculas reais, construindo consensos cada vez maiores, que podem chegar a milhões de pares de bases, como é o caso da montagem de cromossomos eucarióticos.

Os algoritmos de agrupamento são frequentemente executados em duas etapas principais, onde a primeira consiste na separação das sequências em grupos, baseado na similaridade entre elas ser maior do que um limiar pré-definido, já a segunda, encarrega-se pela montagem do consenso, baseado na sobreposição das sequências do mesmo grupo e análise dos valores de qualidade para construção do consenso [HUANG, 1999]. Alguns algoritmos, entretanto, realizam apenas uma ou outra função, como é o caso do BLASTclust, do pacote BLAST [ALTSCHUL, 1997], que apenas mostra quais são as sequências do mesmo grupo, não realizando nenhum procedimento de montagem dos consensos. No caso do algoritmo PHRAP o escore mínimo para o agrupamento de sequências é igual a 30. Já o CAP3 leva em consideração valores de qualidade também no momento de realizar o agrupamento das sequências e, portanto, o escore do alinhamento é ponderado por estes valores [HUANG, 1999].

3.1 - Softwares para montagem de fragmentos

Apesar da existência de diversos programas que podem solucionar o problema descrito anteriormente, os algoritmos mais utilizados pelos pesquisadores são o Phrap [GREEN, 2008] e o CAP3 [HUANG; MADAN, 1999]. Os dois algoritmos levam em consideração os valores de qualidade produzidos pelos algoritmos de nomeação

de bases de forma a tentar produzir uma versão mais consistente das sequências consenso.

O Phrap é utilizado para montagem de fragmentos de DNA e utiliza a ideia do *shotgun* produzindo informações de alta qualidade a partir da leitura de uma sequência original de DNA.

O CAP3, segundo [HUANG; MADAN, 1999], utiliza sobreposição de bases para montagem de sequências e algoritmos para correção de erros e regiões de baixa qualidade. Gratuito para uso acadêmico, o CAP3, possui 3 fases principais. A primeira consiste na identificação e remoção de pontas de baixa qualidade, cálculo das sobreposições e eliminação das falsas sobreposições. Na segunda etapa, os *contigs* são criados através da união das sequências em ordem decrescente de pontuação das sobreposições. A terceira e última etapa, consiste em construir o alinhamento das sequências e determinar as suas qualidades.

Comparando o CAP3 com o Phrap, [TELES; DA SILVA, 2001] realizaram clusterização da cana de açúcar podendo afirmar que o CAP3 forma menor quantidade de *contigs* e menor índice de erros.

Além dessas duas grandes ferramentas da bioinformática existem também, cada um com suas particularidades, alguns outros softwares utilizados na montagem de fragmentos de DNA, como:

- SEQAID [PETOLA *et al.*, 1984];
- AMASS [KIM *et al.*, 1999];
- Celera Assembler [HUSON *et. al.,* 2001];

Embora possamos contar com essa diversidade de diferentes implementações de algoritmos de montagem de fragmentos, todos seguem a mesma ideia, que consiste em três passos básicos: detecção de sobreposição, layout dos fragmentos e decisão da sequência consenso [GUSFIELD, 1997] [MYERS, 1994], onde o primeiro e

o último passo correspondem a problemas de cadeias bem definidos [GUSFIELD, 1997].

3.2 Detecção de sobreposição

O texto seguinte foi extraído e adaptado da pesquisa de Melissa Lemos [LEMOS, 2003].

Sendo o primeiro passo básico da implementação dos algoritmos de montagem de fragmentos, a detecção de sobreposição consiste em descobrir, para cada par de cadeias (fragmentos), quando o sufixo da primeira casa-se com o prefixo da segunda. Se as sequências não possuírem erros, este problema se resumiria a encontrar, para cada par de cadeias S1 e S2, o maior sufixo de S1 que casasse com o prefixo de S2. Mas, na realidade, as sequências possuem erros e por isso é necessário o uso de casamentos aproximados. Com isso, a maioria das implementações definem o casamento entre sufixo-prefixo quando for resolvido o seguinte problema para cada par de cadeias S1 e S2: Encontrar o sufixo de S1 e o prefixo de S2 cuja similaridade é a maior entre todos os pares de sufixos e prefixos de S1 e S2.

Neste caso, a similaridade é baseada em um critério de pontuação onde o casamento exato de caracteres representa um valor positivo, e desigualdades e buracos representam valores negativos. Este problema pode ser resolvido por programação dinâmica e possui complexidade quadrática.

Para aumentar a velocidade deste passo, existem heurísticas com o objetivo de encontrar os melhores casamentos entre sufixos e prefixos, ou seja, encontrar os pares cujos casamentos possuam similaridade alta e, consequentemente, sejam candidatos de pares que se sobrepõem (ou seja, que são vizinhos). Estes candidatos são usados no próximo passo do algoritmo de montagem de fragmentos.

Além disso, é possível aumentar a velocidade deste passo se forem identificados os pares de cadeias cuja melhor similaridade sufixo-prefixo não seja claramente suficiente para serem considerados candidatos atrativos. Nestes casos, a programação dinâmica não precisaria ser utilizada.

Para identificar quais pares são ou não atrativos existem vários métodos. Por exemplo, para que duas cadeias fossem consideradas atrativas, elas deveriam ter uma subcadeia comum significativamente longa, sendo que o comprimento é considerado significativo de acordo com algum argumento probabilístico. O ponto principal aqui é descobrir e excluir muitos pares de cadeias que não aparentam ser vizinhos na sequência original.

Uma heurística parecida com a do BLAST poderia encontrar pares de cadeias vizinhas ao encontrar regiões de alta similaridade entre pares de cadeias. Novamente, a ideia seria de que duas sequências seriam consideradas vizinhas, quando elas possuíssem regiões com comprimento bom de similaridade local alta.

3.3 Layout de Fragmentos

Existem diversos tipos de obtenção de layouts, alguns obtidos através de algoritmos bastante complicados, mas a maioria segue uma variação do método guloso [GUSFIELD, 1997]. Primeiro, o par de cadeias com sobreposição sufixo-prefixo com maior pontuação é escolhido e intercalado. Depois, o próximo par com pontuação maior é escolhido e intercalado. Depois, o próximo par com pontuação maior é escolhido e intercalado, resultando em um *contig* com três fragmentos ou em dois *contigs* com quatro fragmentos (figura 3.1).

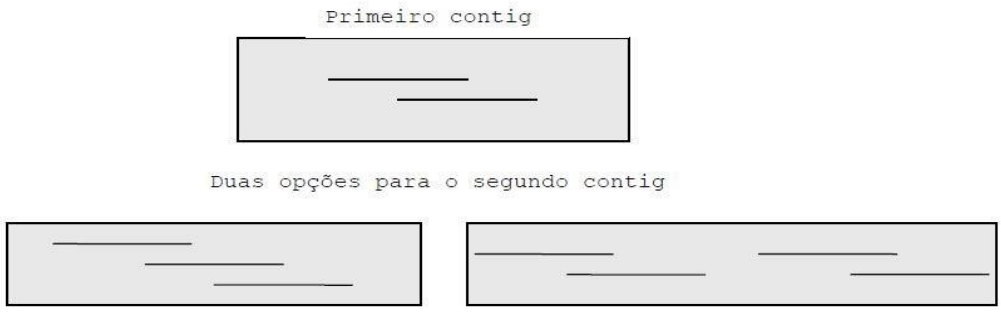


Figura 3.1 – Layout de Fragmentos [LEMOS, 2003]

Como a sobreposição de sufixo e prefixo é determinada por um critério de similaridade, são permitidos buracos nos casamentos de fragmentos. Consequentemente, como vários fragmentos são adicionados em um *contig*, buracos adicionais podem ser inseridos na cadeia que está sendo adicionada ao *contig* para

que ela fique consistente com os buracos da cadeia intercaladas previamente. Este é exatamente o mesmo conceito de alinhamento múltiplo e, de fato, o que está sendo construindo aqui é uma estrutura de alinhamento múltiplo.

Note que este método pode não calcular a melhor sobreposição sufixos-prefixos entre os fragmentos de um determinado *contig* e entre os *contigs*. Provavelmente exista um melhor alinhamento múltiplo, mas geralmente seria computacionalmente caro e por isso na prática não é calculado. Em vez disso, todas as decisões de layout dos fragmentos são baseadas em pontuações dos pares sufixos-prefixos calculados no passo 1, e as inconsistências no layout são resolvidas no passo 3. Além disso, como o passo 2 não envolve programação dinâmica, ele é calculado rapidamente.

3.4 Decisão da sequência consenso

Cada *contig* criado no passo 2 é utilizado aqui para criar um único fragmento (sequência consenso), ou seja, para definir as bases deste fragmento [GUSFIELD, 1997].

Existem diferentes formas de calcular a sequência consenso, mas todas seguem um método similar. Cada fragmento que compõem um *contig* possui um caractere em uma posição particular. Se os caracteres de uma determinada posição em todos os fragmentos forem iguais, então é definido que aquele caractere é a base daquela posição no fragmento. Mas, se isto não ocorrer, deve ser escolhido um caractere ou deve ser indicado que há uma divergência muito grande para sua determinação.

Dentre as diversas formas de determinar a sequência consenso, algumas são aqui ilustradas.

A maneira mais simples seria fazer um relatório com a frequência de cada caractere em cada coluna e deixar o usuário decidir como usar tal informação. Alternativamente, uma sequência consenso pode ser calculada obtendo o caractere que mais se repete em cada coluna. É possível também determinar as janelas no *contig* que possuem muitas divergências e utilizar um método de alinhamento múltiplo nelas para calcular os seus caracteres. Fazendo isso, vários caracteres podem ser mudados ou deslocados.

3.5 Grafos

Podemos, basicamente, resumir um grafo como uma forma de modelar um problema. Este modelo pode ser usado na representação dos problemas aqui abordados, pois suas propriedades são bem conhecidas e podem auxiliar na solução. A seguinte pesquisa de [BRAGA, 2000] apresenta as definições básicas com relação aos grafos.

Considerando um grafo G composto por um conjunto de vértices (ou nós) ligados entre si por linhas denominadas arestas, os conjuntos de vértices e arestas de um grafo G são dados, respectivamente, por V(G) e E(G).

Cada vértice pode ter muitas arestas incidindo sobre ele. O número de arestas que incidem em um vértice determina o grau deste vértice. Dado um grafo G, denotaremos por *d(u)* o grau de um vértice $u \in$ V(G). Denotaremos também por Δ(G) o grau máximo em G. Assim, temos $\Delta(G) = mmm_{u \in V(G)} d(u)$.

Um grafo é dito orientado quando suas arestas desde um vértice de origem até um vértice de destino. Caso o contrário, o grafo é dito não orientado.

Por outro lado, um grafo é ponderado quando a cada aresta é associado um peso.

Como já foi visto, grafos são representados geralmente pela letra G. Utilizaremos então letras minúsculas, como *s*, t, u, v, *w* e *x* para representar vértices. Uma aresta, por sua vez, pode ser representada pelos dois vértices unidos por ela. Se uma aresta liga um vértice *u* a um vértice *v* em um grafo não-orientado, a sua representação é *{u, v}*. No caso de grafos orientados, a ordem dos vértices deve ser levada em consideração nesta representação, sendo que primeiro deve aparecer o vértice de origem e depois o vértice de destino da aresta. Portanto, em um grafo orientado, a aresta (*u, v*) seria distinta da aresta (*v, u*).

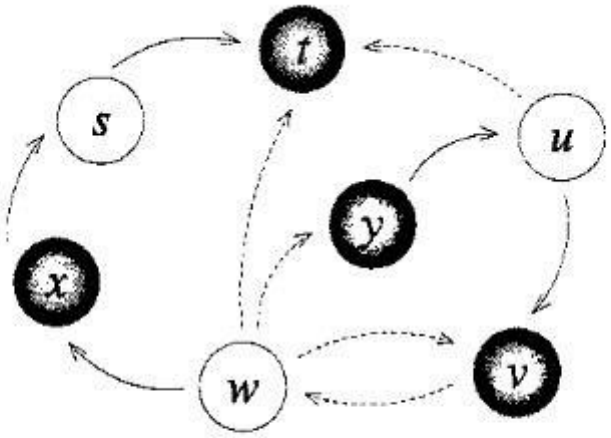
Definição 3.5.1 *Dois grafos G e G' são ditos isomorfos denotados por* $G \cong G'$ *se existe uma bijeção f:* $V(G) \rightarrow V(G')$, *tal que* $E(G') = \{(f(u),f(v)) \mid (u,v) \in E(G)\}$. *Além disso, a bijeção f é chamada de isomorfismo entre G e G'.*

Um caminho em um grafo G é definido por uma sequência alternada de vértices e arestas, sendo que tanto a sua origem quanto o seu destino devem ser vértices e, além disso, o caminho deve passar por cada vértice ou aresta do grafo no máximo uma vez. Dizemos que um elemento de um grafo (isto é, uma aresta ou um vértice) é coberto por um caminho quando este elemento faz parte da sequência que compõe o caminho. Uma vez que cada aresta une apenas dois vértices distintos, a sequência de vértices é suficiente para representar o caminho. Por uma questão de conveniência, definiremos um caminho corno um conjunto ordenado de vértices $U = (u_1, u_2, \dots, u_{|U|})$, de modo que, para todo $I \leq i < /U/$, a aresta $(u_i, u_{i+1}) \in E(G)$.

Dado que um grafo G e um caminho $U = (u_1, u_2, \dots, u_{|U|})$ em G_1 se $(u_{|U|}, u_1) \in E(G)$, dizemos que U é também um ciclo em G.

Dizemos que um grafo G é conexo quando, para quaisquer $u, v \in V(G)$, existe um caminho que vai de u para v. Dizemos ainda que G é grafo completo quando, também para quaisquer vértices $u, v \in V(G)$, temos $(u, v) \in E(G)$, se o grafo for orientado, e $\{u, v\} \in E(G)$, se o grafo for não orientado.

Um conjunto independente em um grafo G é um subconjunto V' de $V(G)$ no qual, para todo par de vértices $u, v \in V'$, as arestas (u,v) e (v,u) não pertencem a $E(G)$. Dizemos que um grafo é bipartido (Figura 3.2) quando todos os seus vértices podem ser agrupados em dois conjuntos independentes A e B tal que $A \cup B = V(G)$ e $A \cap B = \emptyset$.

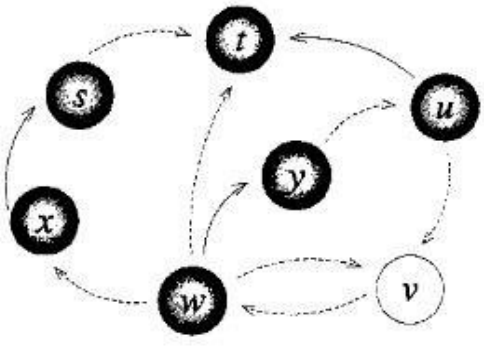


Como os vértices do grafo acima podem ser divididos nos conjuntos independentes $\{s, u, w\}$ e $\{t, v, x, y\}$, este é um grafo bipartido.

Além disso, os conjuntos (w, x, s, t) e (y, u, v) são caminhos disjuntos no mesmo grafo.

Figura 3.2: Caminhos em um grafo bipartido. [BRAGA, 2000]

Um emparelhamento (Figura 3.3), por sua vez, é um subconjunto E' ⊂ E(G) no qual, para todo par de arestas (u, v), (z, w) ∈ E', temos sempre u ≠ z e u ≠ w, como também temos v ≠ z e v ≠ w, ou seja, não existem duas arestas em E' incidindo sobre o mesmo vértice. Um emparelhamento E' é dito maximal se, para toda aresta (x, y) em E(G) \ E', pelo menos um entre os vértices x e y é coberto por alguma aresta em E'.



O conjunto $\{(u, t), (x, s), (w, y)\}$ é um emparelhamento de arestas maximal.

Figura 3.3: Emparelhamento de arestas em um grafo. [BRAGA, 2000]

Além disso, uma coloração de arestas em G é um conjunto de emparelhamentos disjuntos $C = \{M_1, M_2, \dots, M_{|c|}\}$, tal que $M_1 \cup M_2 \cup M_3 \dots \cup M_{|c|} = \mathrm{E(G)}$. Sabe-se, pelo Teorema de Vizing, que uma coloração de arestas mínima C para um grafo G é tal que $|\mathrm{C}| = \Delta(\mathrm{G})$ ou $|\mathrm{C}| = \Delta(\mathrm{G}) + 1$.

3.5.1 Grafos de sobreposição

Definição 3.5.1.1 *Dado um conjunto de cadeias C escrito sobre um alfabeto* $\sum$*, o grafo de k-sobreposição de C, denotado por* $G_k(C)$*, é o grafo G orientado tal que V(G) = C e E(G) =* $\{(u, v) \geq k\}$.

Um vértice v de $G_k(\mathrm{C})$ é denominado terminal quando $v \in T(C)$. Portanto, a propriedade de um vértice ser ou não terminal não depende de k.

Capítulo 4

Agrupamento de várias sequências

O princípio inicial de montagem utilizado neste trabalho foi o de agrupamento de sequências por sobreposição. As técnicas de sobreposição utilizadas consistem primeiramente em identificar, para cada par de fragmentos, a existência de casamentos em suas extremidades.

Para identificar o caractere que ocupa a posição i de uma sequência s utilizamos *s[i]*, onde $0 \leq i < |s|$. Definimos intervalo de uma sequência s, como sendo o conjunto de índices consecutivos *[0 .. j]* tal que $0 \leq i \leq j + 1 \leq |s| + 1$. Para um intervalo *[i .. j]* qualquer de *s*, *s[i .. j]* representa a subcadeia *s[i] s[i + 1]* ... *s[j]* de *s*, onde $i \leq j$. Caso $i = j + 1$ então *s[i .. j]* representará a sequência vazia.

Um prefixo de *s* é qualquer subcadeia de *s*, da forma *s[0..j],* onde $0 \leq j \leq |s|$. Para $j = 0$, a sequência vazia representada por *s[1..0]* também é um prefixo de *s*. Do mesmo modo uma subcadeia de *s* escrita como *s[i .. |s|],* tal que $0 \leq i \leq |s| + 1$ representa um sufixo em s e a sequência vazia *s[|s| + 1..|s|]* também é um sufixo de *s*. Podemos nos referir ao prefixo e sufixo de l caracteres numa sequência *s*, onde $0 \leq l \leq |s|$, utilizando a notação prefixo *(s, l)* e sufixo *(s, l)* respectivamente.

Desconsideramos os erros e imperfeições das sequências citados anteriormente, nosso problema consiste em encontrar, para cada par de cadeias s_1 e s_2, o maior sufixo de s_1 que se case com o prefixo de s_2 ou, o maior sufixo de s_2 que se case com o prefixo de s_1. A similaridade é baseada em um critério de pontuação dado pela quantidade de caracteres envolvidos no casamento destas cadeias.

Algumas importantes técnicas ajudam na solução deste problema como, por exemplo, o método da Comparação Dois a Dois, a utilização de Grafos de Sobreposição para modelagem e o problema da *Menor Supercadeia Comum (MSC).*

4.1 Comparação Dois a Dois

O objetivo desta técnica se baseia em determinar o grau de similaridade que um fragmento s_i de L tem com as demais sequências do mesmo conjunto. O grau de similaridade, ou sobreposição, deste fragmento com os demais é obtido através do número de caracteres envolvidos no casamento prefixo-sufixo de cada comparação. É importante salientar que para cada sequência s∈ L devemos executar duas vezes o procedimento de comparação com todos os demais fragmentos, pois a orientação do casamento é desconhecida e buscamos, também, pelo maior índice de similaridade entre as sequências.

Para exemplo, consideramos dois fragmentos de L, s_1 e s_2. Para encontrar o valor de similaridade entre os fragmentos, devemos procurar pela existência de casamento entre o sufixo de s_1e o prefixo de s_2, bem como, entre o sufixo de s_2 e o prefixo de s_1 para então atribuir ao índice de similaridade o maior valor obtido entre as duas comparações. O procedimento **COMPARA(**s_1, s_2**)** pode resolver este problema:

```
COMPARA (s1, s2)
1.   m ← max(s1, s2);
2.   para i de 0 até m faça
3.        k ← i;
4.        j ← 0;
5.        enquanto ((s1[k] = s2[j]) e (j < m)) faça
6.             k ← k+1;
7.             j ← j+1;
8.        fim
9.        se((k=fim(s1))ou(j=fim(s2))) entao
10.            retorne j;
11.  fim
```

Complexidade: $O(n^2)$

O procedimento **COMPARA(**s_1, s_2**)** recebe como entrada as duas sequências de *L* e, para cada posição de s_1, procura pela existência de casamento com o prefixo s_2. Caso encontre similaridade em determinada posição de s_1, o índice de similaridade é incrementado até que se chegue ao final de s_1 ou de s_2 retornando, então, o valor de similaridade *j*. Caso encontre o final de uma das cadeias sem incrementação do

valor de similaridade o procedimento retorna 0, pois não houve casamento entre estes fragmentos nesta orientação.

O maior valor de similaridade entre s_1 e s_2 é obtido após executarmos o procedimento duas vezes, uma para **COMPARA(**s_1**,** s_2**)** e outra para **COMPARA(**s_2**,** s_1**)** passando a considerar o alinhamento que retornar o maior índice de similaridade.

Por vezes, neste trabalho, foi preciso obter não apenas o valor de qualidade, ou similaridade, entre as sequências, mas também a posição onde inicia o casamento dos fragmentos. Neste caso, utilizamos a mesma estrutura do procedimento **COMPARA()** retornando, a posição do casamento armazenado na variável *i*.

4.2 Menor Supercadeia Comum (MSC)

O problema da Menor Supercadeia Comum ou (*Shortest Common Superstring (SCS)*), se adequa à definição do nosso problema pois, assim como em nosso estudo, o MSC não prevê erros e mutações de nenhuma espécie na leitura do fragmento de DNA a ser sequenciado nem tampouco a sua natureza e, devido a isso, é um modelo com pouca ou nenhuma aplicabilidade na prática. Por outro lado, é muito importante como recurso teórico para compreensão das questões envolvidas no problema de montagem.

No problema do MSC, dado um conjunto L de sequências, procuramos uma sequência S menor possível tal que para qualquer $s \in L$, onde L é supercadeia de s, ou seja, procuramos a menor cadeia possível de modo que todas as cadeias originais sejam subcadeias da cadeia resultante.

Definição 4.2.1: *Dado um conjunto* $L = \{s_1, s_2, \ldots, s_n\}$ *de cadeias, encontrar a menor cadeia S tal que cada cadeia* $s_i \in L$ *seja uma subcadeia de S.*

Tomaremos como exemplo o conjunto de sequências L = {TGTAC, ACAAC, AACGG, ACGGT, GTGAA}. A sequência S= TGTACAACGGTGAA é a menor supercadeia comum de L. A Figura 4.1 ilustra o funcionamento deste método.

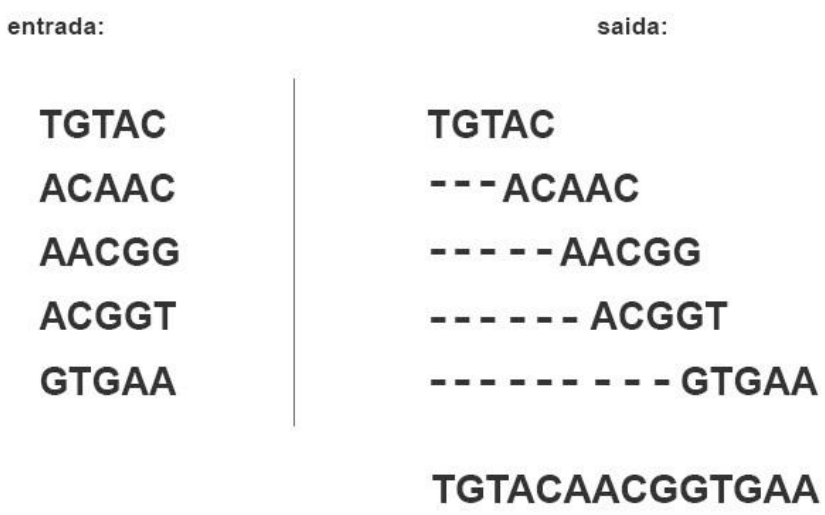


Figura 4.1: Entrada e saída para o MSC.

Este problema é NP-completo [GALLANT *et all.*, 1980]. Turner [Turner, 1989] apresentou vários algoritmos, incluindo o método guloso. O primeiro algoritmo de aproximação constante relacionado ao tamanho da menor *superstring* foi apresentado por [BLUM *et al.*, 1994], que desenvolveram um algoritmo 3-aproximado. Este trabalho se baseou na relação do MSC com os problemas do caixeiro viajante e o problema de cobertura cíclica (*cycle cover problem*) [TURNER, 1989].

Determinar a menor supercadeia comum para um conjunto L de sequências é uma questão da área da Teoria da Computação bastante conhecida que resultou numa abordagem matematicamente elegante para ela, a qual consiste em mudar o contexto do problema para o domínio dos grafos. Tal mudança reduz o problema à tarefa de encontrar um Caminho Hamiltoniano em um grafo de sobreposição, construído adequadamente com as sequências do conjunto L. Basicamente, Caminho Hamiltoniano pode ser definido em um grafo como um caminho onde ocorrem todos os vértices do grafo exatamente uma vez. Analogamente, um ciclo hamiltoniano é um ciclo que contém todos os vértices do grafo exatamente uma vez, com exceção dos vértices inicial e final que têm de coincidir.

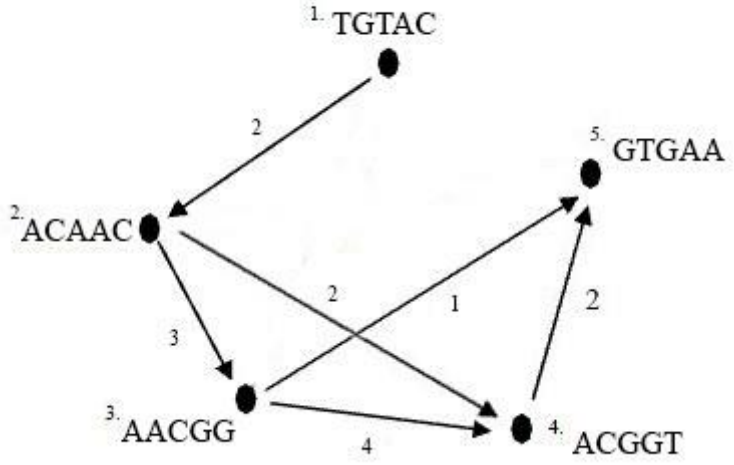


Figura 4.2 Grafo de sobreposição para as sequências de *L*.

No grafo de sobreposição, cada vértice representa uma sequência de *L* e, cada aresta, a sobreposição existente entre as duas sequências que liga. As arestas são orientadas e ponderadas. A orientação será no sentido da sequência $s_1$para a sequência s_2, se a sobreposição ocorrer entre as bases terminais de s_1 e as bases iniciais de s_2, será no sentido de s_2 para s_1 caso o contrário. O peso das arestas corresponde justamente ao tamanho das sobreposições a que se referem. A figura 4.2 representa o grafo de sobreposições para os fragmentos apresentados na figura 4.1.

Cada caminho corresponde a uma supercadeia dos vértices pertencentes a ele. Na realidade, tem-se que cada caminho gera um layout com esses vértices, a partir do qual a supercadeia pode ser reduzida. Por exemplo, para o caminho com a sequência de vértices 1, 3, 5 do grafo da Figura 4.2, obtemos o layout e a supercadeia ilustrados na figura 4.3.

```
TGTAC-----
---ACAAC--
-----AACGG
----------
TGTACAACGG
```

Figura 4.3: Supercadeia correspondente ao caminho 1, 3, 5 do grafo da Figura 4.2.

4.3 Alinhamento de várias sequências

O Alinhamento de várias sequências consiste em comparar mais de dois fragmentos de um conjunto L de sequências e resultar a menor supercadeia comum entre elas. Existem várias formas de calcular a similaridade entre as sequências no alinhamento múltiplo, uma delas seria alinhar todos os fragmentos de L em uma matriz alocada dinamicamente e encontrar, para cada coluna, um caractere representando o *i-ésimo* elemento da menor supercadeia comum para este conjunto de elementos.

Para que consigamos montar a matriz precisamos, antes, obter os alinhamentos dois a dois, apresentados na seção 4.1, entre todos os fragmentos de L e guardar as informações básicas destes alinhamentos em uma matriz de adjacência, que contém os dados para estruturar do grafo hamiltoniano, apresentado no capítulo anterior.

Com essas informações, podemos dar início à construção da nossa matriz de múltiplo alinhamento.

A matriz a ser criada para alinhar várias sequências é alocada dinamicamente, onde suas dimensões estão diretamente relacionadas ao número de sequências a alinhar e o comprimento da menor supercadeia comum.

Tomando o grafo da Figura 4.2 como exemplo, teríamos uma matriz de alinhamento múltiplo como mostra a Figura 4.4.

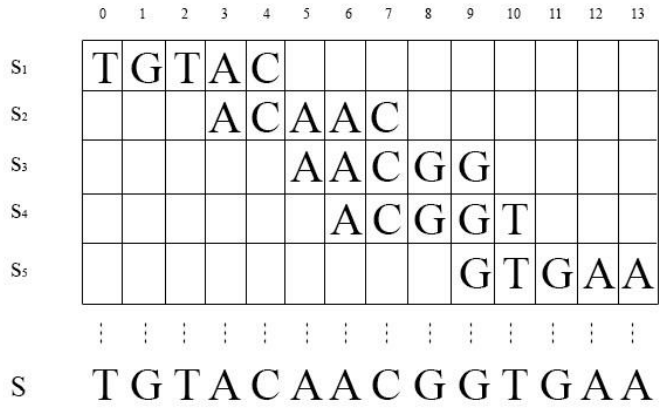


Figura 4.4: Exemplo de matriz para alinhamento múltiplo a partir do grafo hamiltoniano da Figura 4.2.

Capítulo 5

Implementações

Nesta seção apresentamos o funcionamento dos algoritmos utilizados no sequenciamento, alinhamento e agrupamento das cadeias de DNA. Durante toda a implementação dos algoritmos foi utilizado, para testes, um fragmento do genoma completo da bactéria Carboxydothermus Hydrogenoformans Z-2901, disponibilizado no site da NCBI. Este fragmento possui 210 bases e é armazenado em um arquivo de extensão .fasta, como apresentado na figura 5.1.

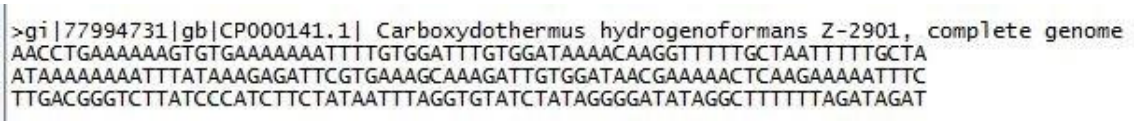


Figura 5.1 Fragmento de DNA utilizado para testes.

Ao longo da implementação este fragmento precisou ser sequenciado, alinhado por comparações dois a dois, por alinhamento de múltiplas cadeias e reagrupado. Nas seções seguintes apresentamos mais detalhadamente cada uma dessas etapas.

5.1 Sequenciamento

Nesta etapa, foi utilizado o método do shotgun apresentado na seção 2.1.1. Sua principal função é ler a sequência original *S* e gerar, randomicamente, *n* subcadeias de *S*.

O algoritmo de sequenciamento utilizando a técnica do *shotgun* é apresentado no pseudo-codigo *QUEBRA_SEQ(S, n)*. Utilizaremos *T(S)* para denotar o tamanho da sequência e *L* para as o grupo de sequências geradas onde, $L = \{l_1, l_2, ..., l_n\}$.

```
QUEBRA_SEQ(S, n)
Entrada: Sequência S, número n de fragmentos a gerar

1. para i de 0 até n-1 faça
2.      r ← rand()%T(S);
3.      t ← rand()%T(S);
4.      se (r > t) então
5.           para j de t até r faça
```

```
6.                $l_i$[j] = S[j];
7.      senão
8.           para j de r até t faça
9.                $l_i$[j] = S[j];
10.retorne L;
```

Complexidade: $O(n^2)$.

O procedimento *QUEBRA_SEQ(S, n)* recebe como entrada uma sequência original e o número de fragmentos que deve gerar, calcula dois números aleatórios que representam o início e fim da nova subsequência e a preenche com as respectivas bases do intervalo randômico da sequência original. A saída do procedimento é um conjunto de *n* fragmentos de *S*.

Em nossas aplicações, o conjunto de saída deste procedimento é gravado em um arquivo .fasta, como mostra a Figura 5.2.

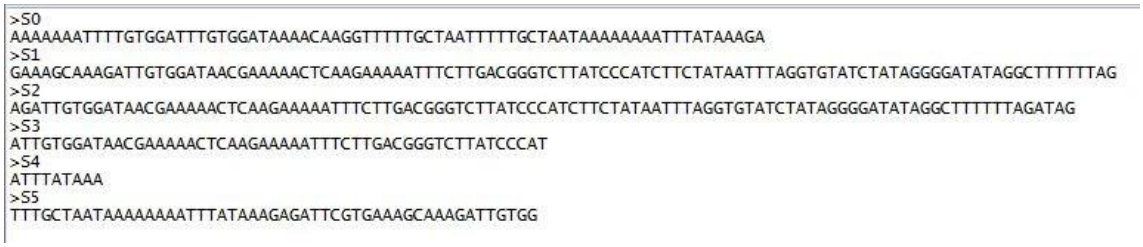


Figura 5.2: Saída do procedimento QUEBRA_SEQ(S, 6).

5.2 Alinhamento Dois a Dois

Nesta etapa fazemos a leitura dos *n* fragmentos gerados pelo procedimento *QUEBRA_SEQ(S, n)* e fazemos a comparação dois a dois para cada um destes fragmentos.

O procedimento que realiza esta tarefa é apresentado na seção 4.1 e tem como entrada duas sequências distintas pertencentes ao conjunto *L* de fragmentos. Retorna a posição do casamento, caso haja, ou retorna 0 em caso contrário.

A figura 5.3 apresenta um exemplo de saída para este procedimento.

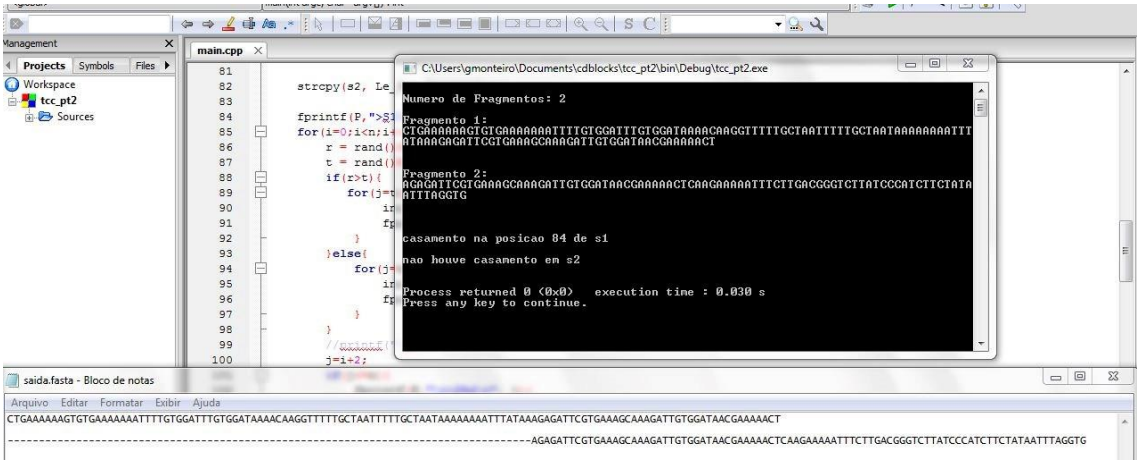


Figura 5.3: Exemplo de saída para o procedimento de alinhamento entre duas sequências.

5.3 Agrupamento de várias sequências

O agrupamento de várias sequências é o foco principal a ser atingido neste trabalho. Para alcançar este objetivo passamos por algumas etapas já citadas anteriormente e algumas demais que são apresentadas nesta seção.

Para chegar ao algoritmo de agrupamento de várias sequências precisamos obter o alinhamento dois a dois entre todas as sequências, montar o grafo hamiltoniano bem como, sua matriz de adjacência.

5.3.1 Construção do Grafo e Matriz de Adjacência

O algoritmo de alinhamento entre duas sequências apresentado na seção 4.1 pode também retornar um arquivo contendo informações úteis para o agrupamento de várias sequências. Na figura 5.4 temos um exemplo de saída para o algoritmo de comparação dois a dois, contendo as informações necessárias.

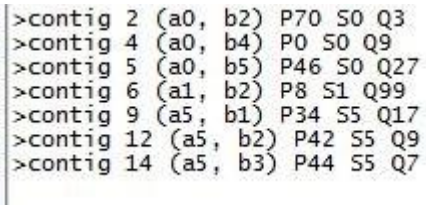


Figura 5.4: Arquivo .fasta resultante da manipulação da execução do algoritmo de comparação de sequências dois a dois.

Este arquivo de saída é resultante das sequências apresentadas no arquivo da figura 5.2 e contém as informações que usaremos para construir o grafo de sobreposição e a matriz de adjacência. As informações trazidas por este procedimento são

a = primeira sequência;
b = segunda sequência;
P = posição do casamento entre elas;
S = orientação;
Q = qualidade.

Note que alguns *contigs* foram descartados deste conjunto. Isto aconteceu, pois há algumas ocasiões em que o alinhamento não é útil para as nossas aplicações e poderia tomar tempo durante a leitura. Desta maneira, são considerados apenas os *contigs* que podem ser usados pelo algoritmo INSERE_ARESTA, apresentado na seção seguinte.

Os *contigs* são descartados quando:

- o alinhamento entre duas sequências s_1 e s_2 resulta em s_1 ser, por completo, subcadeia de s_2 ou s_2 de s_1;

- o valor da qualidade do casamento entre as sequências é menor que um valor de qualidade mínimo aceitável definido.

Com acesso à essas informações, o próximo passo para a construção do nosso Grafo G é descobrir o número de vértices que ele deve conter, que é dado pelo número de sequências a serem alinhadas, contidas em nosso conjunto L composto

por n fragmentos. Podemos, então, criar nosso grafo G alocando, dinamicamente, n vértices.

A estrutura do grafo G é representada pelo registro "grafo".

```
registro grafo{
  V: inteiro;
  A: inteiro;
  **adj: inteiro;
};
registro grafo G;
```

Onde, V é o número de vértices, A é o número de arestas e $adj =$ é um ponteiro para a matriz de adjacência do grafo.

Ao criar o grafo alocamos, também, a matriz de adjacência com V linhas e A colunas. A matriz é preenchida com valor negativo -1, indicando que ainda não existe nenhuma aresta alocada.

Com as estruturas de grafo e matriz criadas, podemos começar a manipulá-las inserindo as informações obtidas anteriormente.

É importante lembrar que, cada vértice do grafo G é dado por um fragmento e cada aresta que liga estes vértices, pelo casamento entre eles. A orientação das arestas é dada pela ordem do casamento entre os fragmentos envolvidos onde, a orientação é de s_2 para s_1 se o casamento envolver o sufixo de s_2 com o prefixo s_1 e de s_1 para s_2 se o casamento envolver o sufixo de s_1 com o prefixo s_2.

Para criar uma nova aresta, utilizamos o procedimento insereAresta(G, v, w, i).

```
insereAresta(G, v, w, i)
Entrada: grafo G, vértice de entrada, vértice de saída,
qualidade do casamento entre os vértices.
    1. se (G→adj[v][w] = -1) então
    2.     G→adj[v][w] ← i;
    3.     G→A++;
```

Inserindo todas as arestas fornecidas pelo arquivo de saída do procedimento de comparação dois a dois, obtemos a matriz de adjacência como ilustrado na Figura 5.5.

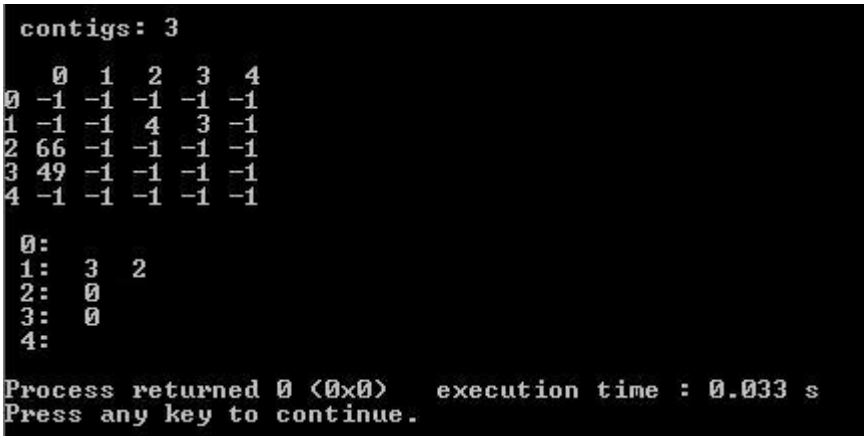


Figura 5.5: Exemplo de uma matriz de adjacência.

A partir da matriz de adjacência da figura 5.5, obtemos o grafo apresentado na Figura 5.6.

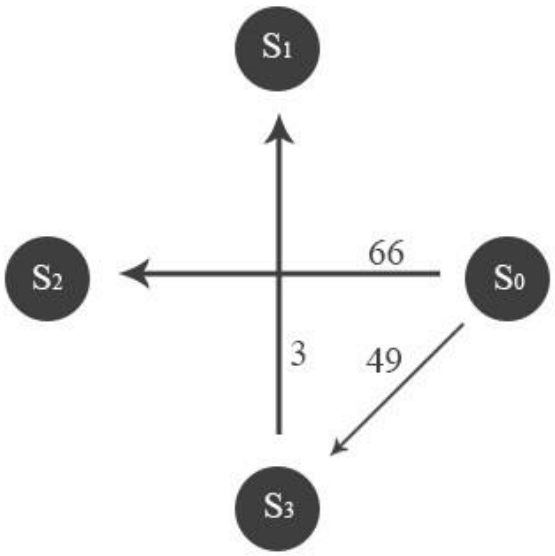


Figura 5.6: Representação do grafo referente à matriz de adjacência da Figura 5.5.

Embora, nesta ilustração, o grafo não aparenta ser um grafo hamiltoniano ele é. Existem arestas ocultas resultantes de fragmentos que não influenciam no resultado final do nosso alinhamento. Um exemplo seria a aresta dos arcos s_1 e s_2 que não é exibida devido ao fato de s_1 estar totalmente contida dentro de s_2.

5.3.2 Agrupamento

Antes de começar a descrever os algoritmos precisamos, para melhor compreensão, definir alguns pontos importantes.

Consideraremos, ao longo desta seção, a variável p como um registro que contém as informações básicas de cada vértice do nosso grafo, a estrutura de p é definida como apresentado no registro "vertice".

```
registro vertice{
  posicao: inteiro;
  qual: inteiro;
  id: inteiro;
  comp: inteiro;
};
registro vertice p;
```

O agrupamento das sequências é obtido executando o procedimento JUNCTION(L,r), que possui como entrada um conjunto L composto de r fragmentos. E retorna uma sequência S que representa a menor supercadeia comum entre os fragmentos de L.

```
JUNCTION(L, r)
 Entrada: s_1,…, s_r            {Alinhamentos de entrada}

 1.      M ← cria_matriz(r);
 2.      p ← INI_SEQ(G); {primeiro fragmento a inserir na matriz}
 3.      C ← s_{p→id}; {trabalharemos com a sequência retornada em p}
 4.      aux[]: vertice;
 5.      para i de 0 até (n-1) faça {onde n é o tamanho de C}
 6.            M[1][i] ← S[i];
 7.            se (p->posicao_matriz < 0) então
 8.                  p->posicao_matriz ← i;
 9.      para i de 1 até r-1 faça
 10.                 p ← PROX_SEQ(G); {próximo fragmento a inserir}
 11.           S ← s_{p→id}; {sequência retornada em p}
 12.           k ← 0;
 13.           enquanto((aux[k]->id)≠(p->comp)) faça
 14.                 k ← k+1;
 15.           p->posição_matriz←((aux[k]->posição_matriz) + (p->posição))
 16.           para j de n->posição_matriz até (n-1) faça
 17.                 M[i][j] ← S[i];
 18.     para i de 0 até j-1 faça
 19.           para k de 0 até r-1 faça
 20.                 se (M[k][i] ≠ ' ') então
 21.                       se (S[i] = ' ') então
 22.                             S[i] ← M[k][i];
 23.                       senão
```

```
24.                                  se (S[i] = M[k][i]) então
25.                                        S[i] ← M[k][i];
25. retorne S;
```

Complexidade: $O(n^2)$.

Este procedimento preenche a matriz de alinhamentos considerando os caminhos do grafo montado na seção anterior. Sua função principal é analisar os dados fornecidos pelos procedimentos já executados e encontrar a melhor posição para inserir as r sequências na matriz.

Em seu último laço, o procedimento *JUNCTION* analisa para cada coluna se todas as bases nela presentes são iguais. Se sim, a base é inserida em sua posição adequada da sequência final. Caso houver bases diferentes na mesma coluna fica, naquela posição da sequência final, um espaço em branco.

O procedimento *INI_SEQ* procura pela primeira sequência a ser adicionada na matriz de alinhamento. O principal critério para escolha deste fragmento consiste em encontrar a sequência que não possui nenhum casamento envolvendo seu prefixo, apenas o sufixo. Esta sequência será a extremidade esquerda, o sufixo, da cadeia final.

```
INI_SEQ(G)
Entrada: grafo de sobreposições do conjunto L
1. p->qual ← -1;
2. para i de 0 até G->V faça
3.     para j de 0 até G->V faça
4.          se (G->adv[i][j]≥0)então
5.               aux ← G->adv[i][j];
6.               p->id ← i;
7.     se (p ≥ 0) então
8.          se (aux>p->qual) então
9.               p->qual ← aux;
10.              p->id ← i;
11.    senao
12.         p->qual ← aux;
13.         p->id ← i;
14.retorne p;
```

Complexidade: $O(n^2)$.

O procedimento *PROX_SEQ* procura pela próxima sequência a ser adicionada na matriz de alinhamento, seus critérios de escolha baseiam-se em encontrar sequências cujo prefixo tem casamento com algum sufixo já inserido na matriz. Após

encontrar a sequência adequada, o procedimento exclui a aresta da matriz de adjacência para que o mesmo não volte a ser comparado.

```
PROX_SEQ(G)
Entrada: grafo de sobreposições do conjunto L
1. p->qual ← -1;
2. para i de 0 até G->V faça
3.     para j de 0 até G->V faça
4.         se (G->adv[i][j]≥0)então
5.             para k de 0 até G->V faça
6.                 aux ← G->adv[j][k]
7.             se (aux≥0) então
8.                 p->id ← i;
9.                 p->qual ← G->adv[i][j];
10.                p->sub ← j;
11.retorne p;
```

Complexidade: $O(n^3)$.

Como resultado da execução destes procedimentos chegamos, enfim, ao alinhamento final de todas as sequências envolvidas. Um arquivo .fasta contendo as sequências envolvidas é gerado mostrando o alinhamento final obtido, como mostra a Figura 5.7.

```
>S0
TTTGCTAATAAAAAAAATTTATAAAGAGATTCGTGAAAGCAAAGATTGTGGATAACGAAAAACTCAAGA
>S3
--------------------ATAAAGAGATTCGTGAAAGCAAAGATTGTGGATAACGAAAAACTCAAGAA
>S2
---GCTAATAAAAAAAATTTATAAAGAGATTCGTGAAAGCAAAGATTGTGGATAACGAAAAACTCAAGAAAAATTTCTTGACGGGTCTTATCCCATCTTCTATAATTTAGGTGTATCTATAGGGGATATAGGCT
>S1
------------------------------------------------------------------------GAAAAATTTCTTGACGGGTCTTAT

TTTGCTAATAAAAAAAATTTATAAAGAGATTCGTGAAAGCAAAGATTGTGGATAACGAAAAACTCAAGAAAAATTTCTTGACGGGTCTTATCCCATCTTCTATAATTTAGGTGTATCTATAGGGGATATAGGCT
```

Figura 5.7: Resultado do alinhamento de várias sequências.

Capítulo 6

Conclusão

Este projeto teve como abordagem principal enfatizar a importância do estudo da biologia molecular computacional, apresentar diferentes técnicas de montagem de fragmentos de DNA e implementar o problema do MSC utilizando programação dinâmica e a estrutura de grafos de sobreposição.

Durante seu desenvolvimento apresentamos embasamentos teóricos que propõem soluções viáveis ao problema montagem de vários fragmentos de DNA. Implementamos o problema do MSC utilizando as técnicas e estruturas previstas e alcançamos o resultado do alinhamento e agrupamento múltiplo de sequências.

Os resultados obtidos neste trabalho são satisfatórios, as sequências foram aleatoriamente quebradas, seus fragmentos foram comparados, alinhados e agrupados como esperado. Contudo, os resultados podem ser melhorados. Ao trabalhar com um grande número de *contigs* o sistema apresenta inconsistências.

Sabemos que ainda há muito que estudar, pesquisar e solucionar com relação à biologia molecular computacional. Este projeto pode inclusive ser aperfeiçoado passando a considerar os erros e mutações, que ocorrem na leitura dos fragmentos, em trabalhos futuros para, então, conseguir utilizar na prática os resultados obtidos.

Referências

[ALTSCHUL, 1997] Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ. Gapped BLAST and PSIBLAST: a new generation of protein database search programs. Nucleic Acids Res. 1997 Sep 1;25(17):3389-402. Review.

[BAPTISTA, 2003] BAPTISTA, Ennio dos Santos "Uma abordagem alternativa para sequenciamento por hibridização", Recife, [PE], 2003.

[BLUM et al., 1994] Blum, A., Jiang, T., Li, M., Tromp, J. and Yannakakis, M. Linear approximation of shortest superstrings. Journal of the ACM, 41(4):630-647.

[BRAGA, 2000], BRAGA, Marilia D. V. "Grafos de sequências de DNA", Campinas, 2000.

[BURKS, 1994], BURKS, C. DNA Sequence Assembly. IEEE Engineering in Medicine and Biology Magazine, v. 13, n. 5, p. 771-773, 1994.

[CERQUEIRA, 2000] Cerqueira, Fábio Ribeiro. "Montagem de fragmentos de DNA".

[DIESTEL, 1997] R. Diestel. Graph Theory. Springer-Verlag New York, lnc, 1997.

[DRMANAC et. al., 2001] Drmanac, R., Drmanac, S (2001) "Sequencing by hybridization arrays", In: RAMPAL, J. B. DNA Arrays: Methods and Protocols. Totowa: Humana, 2001. 170v. (Methods in Molecular Biology).

[GALLANT et al., 1980] Gallant, J., Maier, D. and Storer, J. On finding minimal length superstrings. Journal of Computer and System Science, 20:50-58.

[GREEN, 2008] P. Green. Phrap Homepage: phred, phrap, consed, swat, cross match and RepeatMasker Documentation, July 2008. http://www.phrap.org.

[GREEN, 1998] Green P. Documentation for PHRAP and cross-match. http://www.phrap.org/phrap.docs/phrap.html. 1998.

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

[HUANG, 1999] Huang X, Madan A. CAP3: A DNA sequence assembly program. Genome Res. 1999 Sep;9(9):868-77.

[HUANG; MADAN, 1999] X. Huang and A. Madan. CAP3: a DNA sequence assembly program. Genome Research, 9:868{877, 1999.

[HUSON *et. al.,* 2001] Huson, D.H., Reinert, K., Myers, E.W.. The greedy path-merging algorithm for sequence assembly. Proceedings of the Fifth Annual International Conference on Research in Computational Biology, 157-163, 2001 (RECOMB2001). ACM, 2001.

[KIM *et. al.,* 1999] Kim, S., Segre, A.M.. AMASS: A Structured Pattern Matching Approach to Shotgun Sequence Assembly. Journal of Computational Biology, vol6 (2),163-186,1999. Disponível em http://citeseer.nj.nec.com/85142.html.

[LEMOS, 2003], LEMOS, Melissa "Um estudo dos Algoritmos de Montagem de fragmentos de DNA", Rio de Janeiro, 02/2003.

[LIN, 2001] Lin, Tzy Li "Montagem de fragmentos de DNA pelo método "Ordered Shotgun Sequencing" (OSS)", Campinas, [SP], 2001.

[MYERS, 1994] Myers, E.W.. Advances in sequence assembly in Automated DNA Sequencing and Analysis Techniques (C.Ventner, ed.), Academic Press Limited (London, 1994), 231-238.

[PETOLA *et al*., 1984] Peltola, H., Soderlund, H., Ukkonen, E.. SEQAID: A DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, 12, 307-321, 1984.

[SANGER *et al*., 1982], SANGER, F. et al. Nucleotide sequence of bacteriophage λ DNA.

Journal of Molecular Biology, v. 162, p. 729-773, 1982.

[SANTOS], Fabrício R., ORTEGA, José M. "Bioinformática Aplicada à Genômica".

[SETUBAL; MEIDANIS, 1994] J. C. Setubal and J. Meidanis. Uma Introdução à Biologia Computacional. CIP, 1994.

[SETUBAL; MEIDANIS, 1997] J. C. Setubal and J. Meidanis, Introduction Computational Molecular Biology. PWS Publishing Company, 1997.

[TELES; DA SILVA, 2001] G. P. Telles and F. R. da Silva. Trimming and clustering sugarcane ESTs. Genetics and Molecular Biology, 24(1-4):17{23, December 2001.

[TURNER, 1989] Turner, J. Approximation algorithms for the shortest common superstring problem. Information and Computation, 83:1-20.

[ZAHA, 2000] ZARRA, Arnaldo; Biologia Molecular Básica. 2.ed. Porto Alegre: Mercado Aberto, 2000. 336p.