Montagem de Fragmentos de DNA

Fábio Ribeiro Cerqueira

Dissertação de Mestrado

Montagem de Fragmentos de DNA

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fábio Ribeiro Cerqueira e aprovada pela Banca Examinadora.

Campinas, 25 de Janeiro de 2000.

João Meidanis (Orientador)

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

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA CENTRAL DA UNICAMP

C335m

Cerqueira, Fábio Ribeiro
Montagem de fragmentos de DNA / Fábio Ribeiro Cerqueira. -- Campinas, SP : [s.n.], 1999.

Orientador : João Meidanis.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Teoria dos grafos. 2. * Bioinformática. I. Meidanis, João. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Instituto de Computação
Universidade Estadual de Campinas

Montagem de Fragmentos de DNA

Fábio Ribeiro Cerqueira

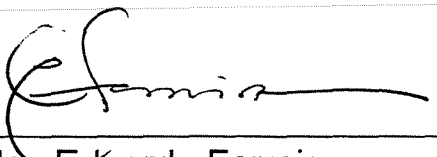
Dezembro de 1999

Banca Examinadora:

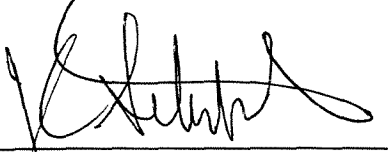
- João Meidanis (Orientador)
- Carlos Eduardo Ferreira
  IME - USP
- João Carlos Setubal
- Cid Carvalho de Souza (Suplente)

TERMO DE APROVAÇÃO

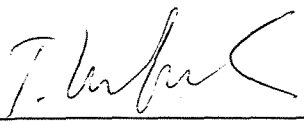
Tese defendida e aprovada em 21 de janeiro de 2000, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Carlos Eduardo Ferreira
IME - USP



Prof. Dr. João Carlos Setubal
IC-UNICAMP



Prof. Dr. João Meidanis
IC-UNICAMP

Dedicatória

Em memória de meu querido pai Odécio...

Agradecimentos

Eu gostaria de fazer um agradecimento muito especial à minha querida esposa Lucimar pela grande compreensão e apoio dado para que eu pudesse terminar este trabalho. Com seu amor, transpor os obstáculos foi sem dúvida muito mais fácil.

Como também não poderia deixar de ser, quero expressar minha imensa gratidão aos meus pais pelo grande incentivo que me foi dado durante todos esses anos. Obrigado também aos meus parentes próximos, com destaque para a vó Lucinda, que realmente torceram pela minha vitória.

Sempre prezei pelas grandes amizades e agradeço ainda aos meus verdadeiros amigos que deixaram suas marcas em meu coração. Obrigado amigos de Vitória, Viçosa, Rio e Campinas.

Finalmente, quero agradecer ao meu orientador João Meidanis, que além do grande auxílio, teve muita compreensão, principalmente em dois momentos do período da realização deste trabalho. O primeiro quando tive que me ausentar de Campinas por problemas pessoais e o segundo quando resolvi retornar e fui muito bem recebido.

Prefácio

As pesquisas na área de Bioinformática vêm crescendo e evoluindo cada vez mais a cada ano que passa. Dentre as várias linhas de estudo, o sequenciamento de genomas é um dos mais explorados hoje em dia. Vários organismos já tiveram seus genomas decifrados e em breve o genoma humano estará fazendo parte destas conquistas.

Sabemos que o tamanho de genomas, mesmo de seres microscópicos, são extremamente grandes e além disso, a natureza biológica do DNA faz com que a realização da montagem desses genomas acarrete problemas computacionais complexos que necessitem de forte estudo teórico e resultem em boas ferramentas para auxílio aos profissionais de biociências.

Neste trabalho, nos propomos a resolver o problema de montagem de fragmentos de DNA. Representamos os fragmentos e os relacionamentos entre estes por um grafo de sobreposição e objetivamos encontrar caminhos nestes grafos que reflitam boas soluções para a montagem. Apresentamos algumas formalizações para o problema e resultados teóricos que relacionam emparelhamento máximo a caminhos nos grafos. Descrevemos ainda as implementações de vários algoritmos polinomiais que em conjunto com outros programas formam a ferramenta `concam` que realiza a montagem segundo o modelo do mínimo contig. Os algoritmos contornam difíceis obstáculos que são comuns no processo de montagem como regiões repetidas e outros.

Conteúdo

Lista de Tabelas

Lista de Figuras

Montagem de Fragmentos de DNA

Fábio Ribeiro Cerqueira

25 de Janeiro de 2000

Capítulo 1

Introdução

Neste trabalho, apresentamos uma abordagem e sua implementação para solucionar o problema de montagem de fragmentos de DNA [25, 24, 18, 27, 19, 2], resultando na molécula de onde estes fragmentos se originaram. Optamos por uma solução baseada em algoritmos sobre grafos que representam os fragmentos e suas sobreposições. Mostramos técnicas para encontrar caminhos nestes grafos que nos indicarão como os fragmentos devem ser alinhados de modo que reproduzam a molécula original. Os grafos são direcionados e podem ser acíclicos ou cíclicos, tendo estes últimos maiores dificuldades de tratamento.

Neste capítulo expomos a motivação para este trabalho, a definição de alguns conceitos biológicos, a descrição do problema de montagem de sequências e seus principais complicadores, esclarecendo termos que serão largamente utilizados durante o decorrer do texto que se seguirá.

No segundo capítulo apresentamos toda a base conceitual que nos possibilitou colocar em prática o desenvolvimento de vários programas para as diversas etapas da montagem. Primeiramente mostramos trabalhos já realizados sobre o mesmo assunto, destacando alguns pontos de vista. Em seguida, discorremos sobre a montagem dos grafos ou seja, o que significa cada vértice e cada arco. Uma vez definido o grafo, demonstramos o principal resultado teórico deste trabalho que associa emparelhamento máximo a caminhos em grafos acíclicos. Posteriormente, mostramos que a solução utilizada para grafos acíclicos pode ser adaptada para os cíclicos de forma a possibilitar resultados significativos para casos que seriam de difícil tratamento.

No capítulo seguinte expomos nossa implementação, isto é, como os resultados teóricos foram postos em prática, destacando o tratamento de contratempos biológicos que ocorrem nos fragmentos e que dificultam a construção de bons grafos e a deter-

minação de caminhos nesses.

No quarto capítulo fazemos uma avaliação dos algoritmos produzidos e exibimos alguns testes que permitem a percepção de como os parâmetros criados em nossa implementação influem positiva ou negativamente na solução final dos conjuntos a serem montados, pois a escolha destes valores afeta diretamente a construção dos grafos e portanto a solução encontrada.

Finalmente, no último capítulo fazemos as conclusões finais e mostramos quais as futuras pesquisas que podem ser realizadas no sentido de melhorar cada vez mais a abordagem de entraves técnicos para a realização da montagem.

1.1 Motivação

Para discorremos um pouco sobre a relevância do assunto em questão é importante que se entenda o significado de *gene* e *genoma*. O primeiro é um trecho contínuo do *DNA* ou *ácido desoxirribonucleico*, contendo informações que possibilitam a síntese de proteínas, essenciais à manutenção da vida e que definem as características de cada ser vivo, sendo responsáveis ainda pelos fatores hereditários. O segundo é formado pelo conjunto de genes e *regiões intergênicas*. Certamente que essas definições são bastante simplificadas mas já passam um sentimento da grande importância deste assunto.

Logo que os trabalhos em direção ao sequenciamento de genomas se iniciaram em laboratórios, percebeu-se que não haveria a possibilidade de realizar esta tarefa sem ajuda computacional. Os tamanhos das moléculas são extremamente grandes, até mesmo de organismos microscópicos, de modo que o volume de informações envolvido impossibilita qualquer tentativa de tratamento manual.

Inicialmente os próprios biólogos desenvolviam programas que pudessem facilitar seus trabalhos, mas rapidamente perceberam a necessidade de técnicas aprimoradas que viabilizassem a solução do problema de montagem, levando-se em conta a complexidade computacional que tais problemas poderiam gerar. Da integração de biólogos com os profissionais da computação surgiu o que chamamos hoje de *bioinformática* que é a área da computação responsável pela formulação de algoritmos e suas implementações para o tratamento de problemas biológicos.

O número de projetos genoma tem aumentado muito nos últimos tempos em todo o mundo. O primeiro projeto deste tipo no Brasil surgiu em 1997 para sequenciar o genoma do microorganismo *Xylella fastidiosa* [22] causador de doenças na citricultura

e cafeicultura. Desde então, vários outros trabalhos neste sentido vêm surgindo e a tendência é de aumentar cada vez mais dada a importância destes estudos. No caso da *Xylella fastidiosa*, a leitura de seu genoma propiciará o surgimento de novas pesquisas para que sejam encontrados eficientes métodos de combate à doença provocada no cultivo da laranja e do café, evitando prejuízos que até agora são empecilhos para os produtores. O genoma humano [14] também vem sendo estudado desde 1990 e da mesma forma possibilitará a criação de pesquisas em busca do conhecimento do complexo funcionamento do organismo humano.

Já que o número de informações em um projeto genoma é tão grande, necessita-se desde banco de dados bem estruturados, até programas que automatizem o intenso processamento envolvido e é claro programas que realizem a montagem propriamente dita. Estes necessitam ser cada vez mais rápidos para viabilizarem projetos que são cada vez mais numerosos e maiores.

Nosso objetivo nesta tese é adquirir tecnologia de ponta em montagem de fragmentos, ou seja, saber implementar aqui no Brasil, algoritmos que possam competir com os melhores existentes.

1.2 Fundamentos Sobre DNA

1.2.1 Conceitos Básicos

A maioria das definições deste capítulo são baseadas nos livros de Setubal e Meidanis [25, 24].

A molécula de DNA é disposta em fita dupla em forma de hélice. As fitas têm orientações definidas que são ditas irem de 5′ para 3′ devido aos carbonos presentes em suas extremidades.

O DNA é formado por *monômeros* que são unidades semelhantes conhecidas neste caso como *nucleotídeos*. Cada um desses possui em sua formação uma molécula denominada de *base* que pode ser de quatro tipos, `Adenina`, `Citosina`, `Timina`, e `Guanina`, representadas pelas letras `A`, `C`, `T` e `G` respectivamente. Podemos nos referir ao tamanho de um DNA tanto pelo seu número de nucleotídeos quanto pelo número de bases, já que a relação é de um para um, mas geralmente utilizamos o número de bases. Como se trata de uma fita dupla é ainda mais usual nos referirmos ao número de pares de base.

O *complemento* de cada base segue a determinação da tabela 1.1.

Base	A	C	T	G
Complemento	T	G	A	C

Tabela 1.1: Bases e seus complementos.

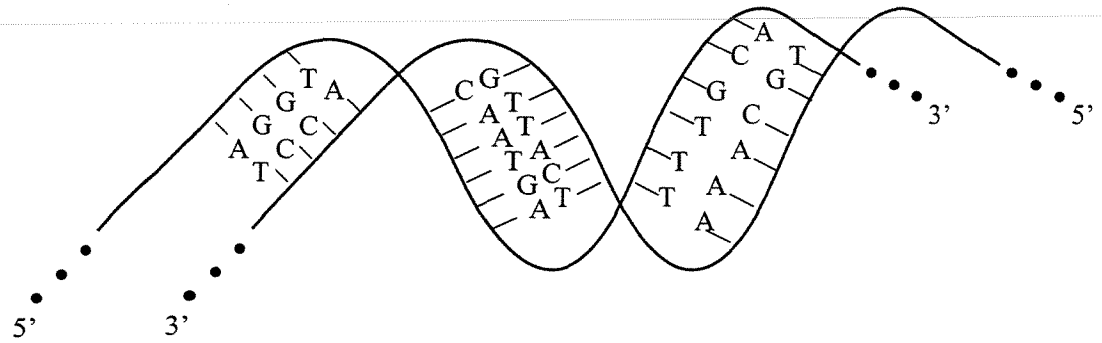


Figura 1.1: Fita dupla do DNA.

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 1.1 um exemplo fictício mostra um trecho de DNA.

1.2.2 Montagem

Vimos que 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 (bp - `base pairs` ou pares de base), ou seja, para ser possível sequenciar um DNA surge a necessidade de quebrá-lo em vários pedaços que denominaremos *fragmentos*. Então estes fragmentos de tamanho reduzido são sequenciados e remontados obtendo o que chamamos de *consenso*, que esperamos ser bem próximo à sequência da molécula que fora incialmente 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 o que chamamos de *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. Como a sequência de uma das fitas de DNA pode ser deduzida

```
Fragmentos:        Alinhamento Múltiplo:        Consenso:

GGTCAC               ---GGTCAC----              ACTGGTCACATTT
TCACATTT             -----TCACATTT
ACTGGT               ACTGGT-------
TGGTCA               --TGGTCA-----
```

Figura 1.2: Exemplo de fragmentação e montagem.

a partir da outra, só é necessário obter a sequência de uma das fitas na montagem.

Na figura 1.2 temos um exemplo para a fita de sequência ACTGGTCACATTT. 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.*

O exemplo da figura 1.2 é totalmente irreal, pois além do tamanho reduzido não ilustra os problemas comuns que ocorrem na prática. A seguir descreveremos cada um desses fatores que podem dificultar a montagem.

1.2.3 Complicadores da Montagem

São vários os fatores que dificultam a definição de um consenso para uma molécula de DNA. A própria natureza dessas macromoléculas contribui para isto. Mas além desse fato, os métodos laboratoriais de sequenciamento que conhecemos hoje, a geração das sequências e o tratamento das mesmas também podem acarretar dificuldades. Passamos agora a descrever cada um desses fatores.

Para que possamos ser capazes de remontar os fragmentos de um DNA obtendo um consenso bastante próximo da realidade, é necessário que estes fragmentos tenham sobreposição uns com os outros e além disso qualquer trecho do DNA em questão deve ter pelo menos um fragmento que o represente, ou seja, os fragmentos gerados devem cobrir toda a molécula. Temos então o primeiro problema a ser tratado que é a possível *falta de cobertura* do DNA que poder ser visualizado na figura 1.3. As regiões onduladas representam as que não têm cobertura.

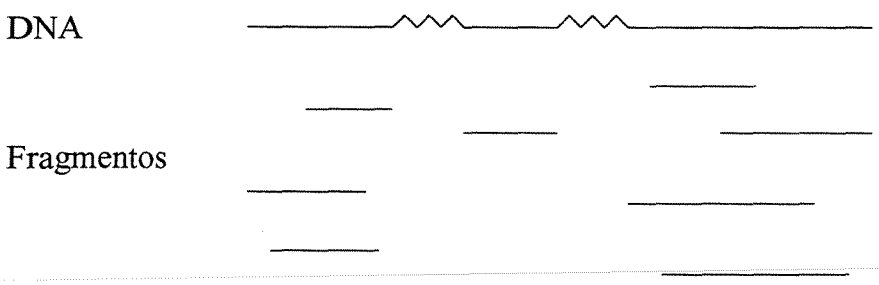


Figura 1.3: Regiões sem cobertura no DNA.

Para que se corra pouco risco de não haver cobertura total, surge a necessidade de se produzir vários *clones* do DNA para então fragmentá-los, ou seja, a redundância de informações é bastante grande. Para a produção dessas cópias é utilizado o DNA de algumas bactérias ou vírus denominados de *vetores* ou *hospedeiros*. O trecho da molécula a ser sequenciada é introduzido ao DNA desses microscópicos seres vivos, que ao se multiplicarem sintetizam cópias exatas do material inserido. Ocorre então o segundo complicador que é a possível *contaminação pelo DNA do vetor*. Para que isso não atrapalhe é necessário saber de antemão qual é a sequência deste DNA utilizado para a clonagem de forma que com o auxílio de programas que fazem comparação de cadeias de caracteres, os trechos pertencentes ao vetor, presentes nos fragmentos a serem montados, sejam identificados e devidamente tratados.

Os biólogos podem contar hoje em dia com poderosas máquinas que automatizam boa parte do processo de sequenciamento e que geram, para cada fragmento, um arquivo binário denominado de *cromatograma*. Estes arquivos possuem um grande número de informações e são lidos por um programa específico de modo a interpretá-los e produzir para cada um, novos arquivos em formato texto contendo a sequência de bases de cada fragmento. Para definir as sequências, este programa realiza sobre os cromatogramas o que chamamos de *base call*. A cada base call uma nova base é identificada. Este processo está sujeito a erros, principalmente em regiões na extremidade dos fragmentos devido ao método de sequenciamento que hoje é empregado. Então podem ocorrer substituições, inserções ou deleções de bases, constituindo os *erros de base call*.

Ultimamente pouco se ouve falar de sequenciamento sem associar a cada base, valores de *qualidade* [6, 8, 7, 23, 20] por critérios probabilísticos. Isto facilita o tratamento dos erros de base call, pois quando uma base possui qualidade baixa, significa que tem alta probabilidade de estar incorreta. Mais adiante indicaremos o programa que utilizamos para interpretação dos cromatogramas. Os arquivos textos gerados por este programa, além da sequência de bases, também contêm as qualidades as-

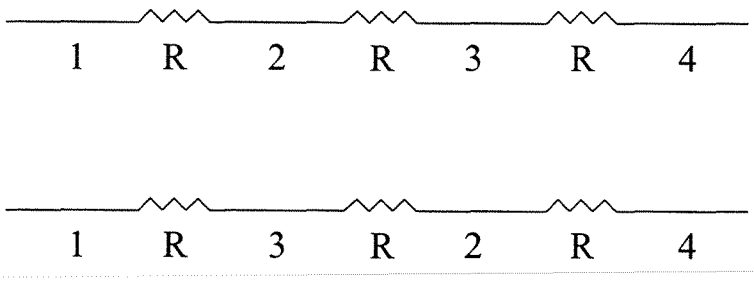


Figura 1.4: Regiões repetidas no DNA.

sociadas às bases. Em capítulos posteriores veremos como o aspecto da qualidade auxiliou este trabalho.

No processo de fragmentação, um outro fator indesejado é o aparecimento dos *fragmentos quiméricos*. Estes são formados por duas ou mais partes provindas de regiões não contínuas na molécula original. Os pontos nesses fragmentos que separam essas partes são definidos como *pontos de separação*, pois separam o que se conhece por *regiões reais*, assim chamadas pelo fato de, individualmente, pertencerem à trechos contínuos na molécula. Similarmente, denominamos de *regiões quiméricas* as partes do fragmento que contêm um ou mais pontos de separação e que portanto não existem de forma contínua no genoma estudado. O aparecimento de fragmentos com esta característica pode dificultar muito nosso trabalho, pois ocasiona ambiguidades, já que pode acarretar a união de pedaços que não deveriam ser juntados.

Uma outra característica do processo de montagem é que quando se realiza a quebra das moléculas grandes em fragmentos, sabemos que estes podem vir de qualquer uma das fitas do DNA. Devemos prever então este fato da *orientação desconhecida*, fazendo testes de sobreposição não só com os fragmentos na orientação em que são fornecidos mas também com seus complementos reversos.

Existe ainda uma peculiaridade do DNA que talvez seja o maior complicador para a realização da tarefa de montar fragmentos. Trata-se de moléculas que possuem *regiões repetidas* ou somente *repetições*, ao longo de sua formação. Isto pode ocasionar consensos ambíguos para o problema. O exemplo da figura 1.4 mostra duas possíveis soluções para um DNA qualquer, contendo três repetições, em que há uma permutação entre as regiões 2 e 3. As regiões onduladas e nomeadas por R representam as repetições.

1.3 Sumário

Neste capítulo vimos a importância de se realizar investigações na área de sequenciamento e ainda descrevemos conceitos básicos de biologia molecular no que se refere ao DNA. Mostramos os contratempos que podem ocorrer na montagem, sendo que alguns deles podem ser facilmente resolvidos, inclusive alheios à computação envolvida. Outros fatores, no entanto, são completamente dependente do tratamento computacional e representam dificuldades que justificam o esforço despendido neste trabalho. Estaremos mostrando toda a teoria e a prática ao longo deste texto que tenta contornar os possíveis complicadores.

Capítulo 2

Modelagem em Grafos

2.1 Introdução

Apresentamos a seguir importantes resultados teóricos usados em nosso trabalho que guiaram o processo de implementação. Primeiramente, descrevemos uma formalização para o significado de grafos de sobreposição [25]. Em seguida relacionamos estes grafos com o trabalho de Ferreira, Souza e Wakabayashi [9], que também implementa algoritmos para montagem de fragmentos e cuja a formulação envolve encontrar caminhos disjuntos em grafos de sobreposição. A seguir, mostramos como emparelhamentos máximos em determinados grafos originados de grafos de sobreposição acíclicos podem significar caminhos nestes últimos. Para encerrar veremos como o mesmo resultado para os grafos acíclicos poderão também ser empregados para os casos cíclicos.

É importante salientar que as definições e notações utilizadas acerca de grafos em todo o texto será baseada no livro de Bondy e Murty [1].

2.2 Grafos de Sobreposição

De um modo geral, um *grafo de sobreposição* para um determinado conjunto de fragmentos é a representação das sobreposições que estes possuem entre si. Nas seções à frente estaremos mostrando um significado mais preciso para isto. Estes grafos são orientados e podem ser acíclicos ou cíclicos.

2.2.1 Definições Para Sequência

Para o enfoque computacional da montagem, são de interesse apenas as sequências de bases dos fragmentos. Então passemos a definir melhor alguns termos, seguindo o livro de Setubal e Meidanis [25].

Uma *sequência* é uma sucessão ordenada de *caracteres* ou *símbolos* provindos de um conjunto finito chamado de *alfabeto*. Pelo primeiro capítulo podemos concluir que o alfabeto em que se baseiam as sequências com as quais trabalharemos consiste no conjunto {A,C,T,G}.

O *tamanho* de uma sequência s é equivalente ao número de caracteres que a mesma possui e será denotado por $|s|$. Há uma sequência de tamanho zero chamada de *sequência vazia.*

Chamaremos de *subsequência* de s, qualquer sequência obtida pela remoção de alguns caracteres de s. Se esta remoção for tal que a subsequência resultante seja formada por caracteres consecutivos de s então teremos também uma *subcadeia* de s. A sequência vazia é subsequência ou subcadeia de qualquer sequência. Se t é uma subcadeia de s então dizemos que s é uma *supercadeia* de t ou ainda que s *cobre* t.

Para duas sequências s e t quaisquer, dizemos que o resultado da *concatenação* entre elas é a sequência denotada por st obtida pela inclusão dos caracteres de t ao final de s na ordem que estes aparecem em t.

Para identificar o caractere que ocupa a posição i da sequência utilizamos $s[i]$, onde $1 \leq i \leq |s|$. Note que o primeiro caractere tem índice 1. Definimos *intervalo* de uma sequência s, como sendo o conjunto de índices consecutivos $[i..j]$ tal que $1 \leq i \leq j+1 \leq |s|+1$. 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 $s[1..j]$, 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 $1 \leq i \leq |s|+1$ representa um *sufixo* em s e a sequência vazia $s[|s|+1..|s|]$ também é um sufixo de s. Algumas vezes nos referimos ao prefixo e sufixo de ℓ caracteres numa sequência s, onde $0 \leq \ell \leq |s|$, utilizando a notação prefixo(s, ℓ) e sufixo(s, ℓ) respectivamente.

2.2.2 Vértices

Por todo o texto, identificaremos cada fragmento com sua sequência. Seja o conjunto $\mathcal{F}$ de fragmentos a serem montados. Para cada elemento deste conjunto, há dois vértices no grafo de sobreposição $G(\mathcal{F})$, um sendo a própria sequência do fragmento na orientação em que é fornecido e outro representando o complemento reverso desta sequência. O número de vértices do grafo então será igual a $2|\mathcal{F}|$. Se $s \in \mathcal{F}$ é um vértice do grafo, $\overline{s}$ será o vértice que representará o complemento reverso de s. De maneira mais formal, podemos escrever o conjunto de vértices de $G(\mathcal{F})$ como $V = \{s, \overline{s} \mid s \in \mathcal{F}\}$. Note que $\overline{\overline{s}} = s$.

Sejam s e t sequências quaisquer de V. Se s é subcadeia de t, chamamos o vértice s de *vértice de Steiner* [9]. Como o complemento reverso de s é da mesma forma uma subcadeia do complemento reverso de t, então $\overline{s}$ também será um vértice de Steiner. Para exemplificar, sejam $s =$ `CCAGC` e $t =$ `GCCAGCTTA`, os complementos reversos de cada sequência são $\overline{s} =$ `GCTGG` e $\overline{t} =$ `TAAGCTGGC` respectivamente. Vemos então que o vértice s bem como o $\overline{s}$ no grafo são vértices de Steiner. Os vértices que não são de Steiner são denominados de *terminais* [9].

2.2.3 Arcos

Sejam s e t vértices do grafo de sobreposição $G = G(\mathcal{F})$. Existirá o arco $(s,t) \in A(G)$ se e somente se, sufixo$(s,\ell) =$ prefixo(t,ℓ), onde $1 \leq \ell \leq min(|s|,|t|) - 1$ e além disso s e t não forem complementos reversos um do outro e $s \neq t$.

Pela tabela 1.1 é fácil verificar a propriedade de que $(s,t) \in A(G)$ se e somente se $(\overline{t},\overline{s}) \in A(G)$. Exemplificando para as sequências $s =$ `TCCACGTTA` e $t =$ `GTTACCCTAGGA`, vemos que um sufixo de s é igual a um prefixo de t. Seus complementos reversos são respectivamente $\overline{s} =$ `TAACGTGGA` e $\overline{t} =$ `TCCTAGGGTAAC` e percebemos também que um sufixo de $\overline{t}$, que corresponde a um prefixo complementado de t, é igual a um prefixo de $\overline{s}$ que corresponde a um sufixo complementado de s.

No capítulo onde mostraremos os testes realizados, veremos que um valor mínimo para ℓ, não muito baixo, deve ser fixado, de modo que possamos obter arcos realmente interessantes para fragmentos cujos tamanhos variam na prática, aproximadamente, entre 200 a 1000 bases.

Na figura 2.1, vemos o grafo de sobreposição para as sequências `GGTAGTCACTAGCCTA`, `TAGCCTAATTTCGGCGGAAT`, `GCGCACTGTTATTCC` e `CACTAGCC`, rotuladas como 1, 2, 3 e 4 respectivamente. Os arcos são definidos para $\ell \geq 5$. Os vértices 4 e $\overline{4}$ são de Steiner, sendo exibidos como $4(s)$ e $\overline{4}(s)$.

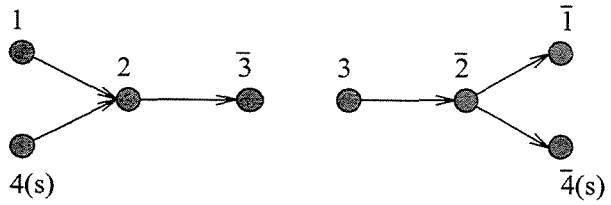


Figura 2.1: Exemplo de grafo de sobreposição.

2.2.4 Contigs

Uma das primeiras tentativas de formalizar montagem de fragmentos foi através do problema da *Menor Supercadeia Comum* ou *MSC* [26]. Neste problema, para uma dada coleção $\mathcal{F}$ de sequências, procuramos uma sequência S menor possível, tal que para qualquer $f \in \mathcal{F}$, S seja supercadeia de f. Como exemplo, seja $\mathcal{F} =$ {TGTAC, ACAAC, AACGG}. A sequência $S =$ TGTACAACGG é a menor supercadeia comum de $\mathcal{F}$.

No modelo MSC, não se prevê erros de nenhum tipo e tampouco a natureza do DNA como no caso de regiões repetidas. É esperado que S seja uma supercadeia perfeita da coleção de sequências, portanto apesar do estudo deste modelo ter grande importância, sua utilização não é possível para problemas reais. Algumas extensões deste trabalho como os modelos *Reconstrução* [21, 15, 16] e *Multicontig* [25] foram realizados prevendo erros, orientações e no caso do segundo até regiões repetidas, mas, como no MSC, são formalizações NP-difíceis, não sendo portanto apropriados para grandes instâncias do problema de montagem de fragmentos.

No trabalho de Ferreira, Souza e Wakabayashi [9], como neste trabalho, os grafos de sobreposição são estudados em conexão com o problema de montagem de fragmentos. Refraseamos aqui suas definições e problemas principais relacionados com esta dissertação. Note que usamos os termos contig e consenso de maneira um pouco diferenciada.

Dadas duas sequências s e t tais que sufixo(s, ℓ) = prefixo(t, ℓ) e t tem tamanho n, denotaremos por $s|_\ell t$ a sequência obtida pela concatenação de s com os últimos $n - \ell$ caracteres de t. A notação $s_1|_{\ell_1}s_2|_{\ell_2}...|_{\ell_{m-1}}s_m$ é utilizada para denotar a sequência definida como se segue. Primeiramente, obtemos a sequência $s'_1 = s_1|_{\ell_1}s_2$, então para $2 \leq i \leq m-1$ definimos $s'_i = s'_{i-1}|_{\ell_i}s_{i+1}$.

Seja o conjunto de fragmentos $\mathcal{F}$. Seja a m-upla $P = (s_1, s_2, ..., s_m)$ de sequências em $\mathcal{F}$, que tenham a propriedade de que para qualquer par consecutivo das sequências em P, denotado por s_j e s_{j+1}, existe um inteiro $\ell_i \geq k$ tal que sufixo(s_j, ℓ) =

prefixo(s_{j+1}, ℓ) e s_j e s_{j+1} não são subcadeias uma da outra. Então P representa um caminho no grafo de sobreposição $G(\mathcal{F})$ e a sequência $S(P) = s_1|_{\ell_1} s_2|_{\ell_2} \ldots |_{\ell_{m-1}} s_m$ é chamada de um *k-consenso* em $\mathcal{F}$. O alinhamento múltiplo das sequências do caminho P, que deu origem ao consenso, representará um *k-contig* em $\mathcal{F}$.

Para cada inteiro positivo k, definimos o *problema do mínimo k-contig* ou MkCP (do inglês Minimum k-Contig Problem) da seguinte forma. Dado um conjunto de sequências $\mathcal{F}$, encontrar uma coleção C de k-contigs com a propriedade de que cada sequência de $\mathcal{F}$, que não seja coberta por nenhuma outra sequência deste mesmo conjunto, seja subcadeia de um k-consenso originado de algum k-contig de C e de modo que a cardinalidade de C seja mínima possível. Podemos definir ainda, para o caso de grafos de sobreposição, como o menor conjunto de caminhos de vértices disjuntos em $G(\mathcal{F})$ que cubram os vértices terminais. Os vértices de Steiner já estarão automaticamente cobertos pelos terminais. O MkCP é NP-completo como pode ser visto no trabalho de Ferreira e colegas [9].

O MkCP não faz menção ao complemento reverso, mas, como visto anteriormente, o grafo de sobreposição de nosso interesse possui não só as sequências na orientação que aparecem em $\mathcal{F}$ mas também os complementos reversos dessas sequências, de modo que possamos contornar a questão da orientação desconhecida. Uma extensão do MkCP, chamada MkCP^r, é então definida para solucionarmos para tais grafos. Neste caso, o objetivo é encontrar o menor conjunto de caminhos de vértices disjuntos em $G(\mathcal{F})$ que cubram os vértices terminais em uma de suas orientações. Veremos como isto será resolvido nas próximas seções.

Como mencionado na seção 2.2.3, um valor mínimo de sobreposição será fixado na prática, por isso omitiremos o valor k do restante do texto. Então chamaremos MkCP^r, k-contig e k-consenso, simplesmente de MCP^r, contig e consenso respectivamente.

2.3 Caminhos em Grafos Acíclicos e Emparelhamento Máximo

Nesta seção faremos uma relação entre caminhos e emparelhamentos. No capítulo sobre fluxo máximo do livro de Cormen, Leiserson e Rivest [4] pode ser encontrado um exercício que faz a relação de caminhos em grafos acíclicos com fluxo máximo e sabe-se que um caso especial deste último pode ser modelado de forma a corresponder ao emparelhamento máximo. Neste trabalho provamos e estudamos mais a fundo este tema.

Um *caminho* em um grafo orientado G é um subgrafo C de G tal que $|V(C)| = 1$ ou então:

1) existe um único $r \in V(C)$ tal que $d_C^-(r) = 0$ e $d_C^+(r) = 1$

2) existe um único $s \in V(C)$ tal que $d_C^-(s) = 1$ e $d_C^+(s) = 0$

3) para todo $t \in V(C)$ com $t \neq r$ e $t \neq s$, temos $d_C^-(t) = d_C^+(t) = 1$

Seja $G = (V(G), A(G), \psi_G)$ um grafo orientado. Definimos o *dobro* de G como sendo o grafo bipartido $G' = (V(G'), E(G'), \psi_{G'})$, com partições $X(G')$ e $Y(G')$ e as bijeções $\theta : V(G) \to X(G')$, $\phi : V(G) \to Y(G')$, $\beta : A(G) \to E(G')$, tais que, $\theta(v) = v_x$, $\phi(v) = v_y$, com $v \in V(G)$, $v_x \in X(G')$, $v_y \in Y(G')$ e $\psi_G(a) = (u, v)$ se e somente se $\psi_{G'}(\beta(a)) = \theta(u)\phi(v)$, onde $u, v \in V(G)$ e $a \in A(G)$.

Teorema 2.1 *Seja G um grafo orientado acíclico, G' o seu dobro, M um emparelhamento em G' e H o subgrafo de G definido por $V(H) = V(G)$ e $A(H) = \beta^{-1}(M)$. Então o conjunto de componentes conexas $C(H)$ será um conjunto de caminhos em G, disjuntos nos vértices, que cobrem $V(G)$. Além disso,*

$$|M| + |C(H)| = |V(G)|.$$

Prova: Provaremos o resultado por partes.

A primeira parte consiste em provar que $C(H)$ é um conjunto de caminhos. Seja $P = (V(P), A(P)) \in C(H)$ uma componente conexa arbitrária. Desejamos provar que P é um caminho em G.

Primeiramente, se H é um subgrafo de G, então P, por ser uma componente conexa de H, também é um subgrafo de G. Segundo, se $|V(P)| = 1$, então, pela definição, P é um caminho em G. Por último, se $|V(P)| > 1$, devemos mostrar que:

1) existe um único $r \in V(P)$ tal que $d_P^-(r) = 0$ e $d_P^+(r) = 1$. Pela definição de H, $(u, v) \in A(H)$ se e somente se $\theta(u)\phi(v) \in M$, onde $u, v \in V(H)$, mas como $\theta(u)\phi(v)$ é aresta de um emparelhamento, então $d_{G'}(\theta(u)) = d_{G'}(\phi(v)) = 1$ e isto implica em $d_H^-(u) = d_H^+(v) = 1$, ou seja, os graus de entrada e saída dos vértices de H têm valor no máximo 1. Se G é acíclico, então H como subgrafo de G, também é acíclico, portanto, P possui pelo menos um vértice r, tal que $d_P^-(r) = 0$ e $d_P^+(r) = 1$. Suponha que haja um conjunto $F(P)$ de vértices como r, onde $|F(P)| > 1$. Já que P é conexo, existe pelo menos um vértice q de P, não pertencente a $F(P)$, que é alcançável a partir de pelo menos dois vértices de $F(P)$, tal que $d_P^-(q) > 1$. Isto é absurdo, pois vimos que os graus são no máximo 1, então $|F(P)| = 1$.

2) existe um único $s \in V(P)$ tal que $d_P^-(s) = 1$ e $d_P^+(s) = 0$. Da mesma forma que no ítem anterior, H ser acíclico implica em existir em P pelo menos um vértice s, tal que, $d_P^- = 1$ e $d_P^+ = 0$. Suponha o conjunto $T(P)$ de vértices como s, onde $|T(P)| > 1$. Já que P é conexo, existe pelo menos um vértice q de P, não pertencente a $T(P)$, a partir de onde, pelo menos dois vértices de T são alcançáveis, tal que $d_p^+ > 1$. Isto é absurdo, pois os graus são no máximo 1, sendo $|T(P)| = 1$.

3) para todo $t \in V(P)$ com $t \neq r$, $t \neq s$, temos $d_p^-(t) = d_p^+(t) = 1$. Vimos que os graus de entrada e saída de vértices de P são no máximo 1. Vimos nos ítens anteriores que r e s são únicos. Sabemos também que $|V(P) > 1|$ e que P é conexo, o que elimina a possibilidade $d_p^-(t) = d_p^+(t) = 0$. Então só restam os vértices cujo os graus de entrada e saída têm valor 1.

A segunda parte consiste em mostrar que os caminhos de $C(H)$ são disjuntos nos vértices. Mas cada vértice em um grafo, só pode participar de uma única componente conexa deste grafo.

A terceira parte consiste em mostrar que cobrem $V(G)$. Isto segue da definição de H onde pusemos $V(H) = V(G)$.

Finalmente, devemos demonstrar a equação. Podemos escrever $|V(H)| = |V(G)| = |V(C_1)| + |V(C_2)| + ... + |V(C_n)|$, onde $\{C_1, C_2, ..., C_n\}$ são todas as n componentes conexas de $C(H)$. Podemos ainda, devido à bijeção β, escrever $|A(H)| = |M| = (|V(C_1)| - 1) + (|V(C_2)| - 1) + ... + (|V(C_n)| - 1) = |V(C_1)| + |V(C_2)| + ... + |V(C_n)| - n$. Se somarmos $|M|$ a $|C(H)|$, teremos $|V(C_1)| + |V(C_2)| + ... + |V(C_n)| - n + n = |V(C_1)| + |V(C_2)| + ... + |V(C_n)| = |V(G)|$. □

Teorema 2.2 *Seja G um grafo orientado acíclico, G' o seu dobro, C um conjunto de caminhos em G, disjuntos nos vértices, que cobrem $V(G)$ e $M = \beta(A(C))$. Então M é um emparelhamento em G'. Além disso,*

$$|M| + |C| = |V(G)|.$$

Prova: Se os caminhos de C cobrem $V(G)$, então $V(G) = V(C)$ e como $M = \beta(A(C))$, então $M \subseteq E(G')$. Temos que provar que qualquer duas arestas de M não são adjacentes em G'. Suponha que as arestas $a, b \in M$ sejam adjacentes em G'. Isto poderia ocorrer pelas partições $X(G')$ ou $Y(G')$. No primeiro caso, considerando $a = u_x v_y$ e $b = u_x t_y$, teríamos $d_G^+(u) = 2$ o que é impossível por u fazer parte de

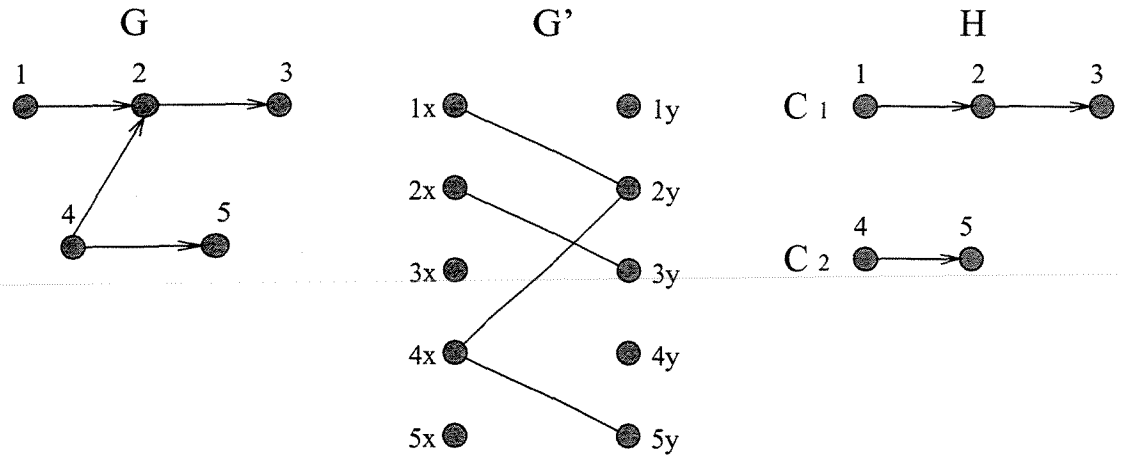


Figura 2.2: Emparelhamento máximo e caminhos associados.

um caminho em G. No segundo caso, considerando $a = u_x v_y$ e $b = t_x v_y$, teríamos $d_G^-(u) = 2$ o que também é impossível por v_y, da mesma forma, fazer parte de um caminho em G.

Quanto à equação, podemos escrever $|V(C)| = |V(G)| = |V(C_1)| + |V(C_2)| + ... + |V(C_n)|$, onde $\{C_1, C_2, ..., C_n\}$ são todos os n caminhos do conjunto C. Também é verdade que $|M| = |A(C)| = (|V(C_1)| - 1) + (|V(C_2)| - 1) + ... + (|V(C_n)| - 1) = |V(C_1)| + |V(C_2)| + ... + |V(C_n)| - n$. Se somarmos $|M|$ a $|C|$, teremos $|V(C_1)| + |V(C_2)| + ... + |V(C_n)| - n + n = |V(C_1)| + |V(C_2)| + ... + |V(C_n)| = |V(G)|$. □

Teorema 2.3 *Seja G um grafo orientado acíclico. Para encontrar uma família C de caminhos disjuntos nos vértices que cobrem $V(G)$ e tal que $|C|$ é mínima, basta encontrar um emparelhamento máximo em G'.*

Prova: Vimos que um emparelhamento em G' é um conjunto de caminhos em G, disjuntos nos vértices que cobrem $V(G)$. Vimos ainda que a equação do teorema 2.1 é válida. Vemos nessa equação que $|V(G)|$ é constante. Então na medida em que aumentamos $|M|$, $|C(H)|$ deve necessariamente diminuir, ou seja, para emparelhamentos cada vez maiores, temos um número de caminhos cada vez menor. □

A figura 2.2 mostra um exemplo simplificado que ilustra o exposto nesta seção.

2.4 Grafos Cíclicos

Na seção anterior mostramos como encontrar caminhos em grafos acíclicos através de emparelhamento máximo. Os resultados descritos a seguir, mostram que utilizando a mesma idéia para grafos cíclicos, o emparelhamento máximo corresponderá, neste caso, a caminhos e ciclos no grafo de sobreposição. Em nosso trabalho, ao invés de tentarmos buscar soluções evitando a formação de ciclos, que é exatamente o que caracteriza a NP-completude do MCP^r, deixamos que os ciclos se formem naturalmente e tentamos fazer a recombinação dos ciclos e caminhos de modo que, ao final, só sobrem caminhos na solução. Mais detalhes serão dados no próximo capítulo.

Um *ciclo* em um grafo orientado G é um subgrafo conexo C de G tal que para todo $t \in V(C)$, temos $d_C^-(t) = d_C^+(t) = 1$.

Teorema 2.4 *Seja G um grafo orientado, G' o seu dobro, M um emparelhamento em G' e H o subgrafo de G definido por $V(H) = V(G)$ e $A(H) = \beta^{-1}(M)$. Então o conjunto de componentes conexas $C(H)$, será um conjunto de caminhos e ciclos em G, disjuntos nos vértices, que cobrem $V(G)$. Além disso*

$$|M| + |C'(H)| = |V(G)|, \tag{2.1}$$

onde $C'(H)$ é o conjunto de caminhos de $C(H)$.

Prova: Vimos no teorema 2.1 que devido ao emparelhamento M, os graus de entrada e saída de qualquer vértice de H são no máximo 1. Seja C_i uma componente conexa de $C(H)$. Então se $|C_i| = 1$, temos um caminho. Se $|C_i| > 1$, há duas situações possíveis.

A primeira situação é a existência de algum vértice v de H, tal que $d_H^-(v) = 0$ e $d_H^+(v) = 1$ ou ainda, $d_H^-(v) = 1$ e $d_H^+(v) = 0$. Neste caso, esta componente conexa não pode representar um ciclo e pelo teorema 2.1 podemos afirmar que se trata de um caminho em G.

A outra situação é que se para todo $v \in C_i$, $d_H^-(v) = d_H^+(v) = 1$, temos a caracterização de um ciclo em G.

Para demonstrarmos a equação, podemos escrever $|V(H)| = |V(G)| = |V(C_1)| + |V(C_2)| + ... + |V(C_n)|$, onde $\{C_1, C_2, ..., C_n\}$ são todas as componentes conexas do conjunto $C(H)$. Podemos ainda, devido à bijeção β, escrever $|A(H)| = |M| = (|V(C_1)|-1)+(|V(C_2)|-1)+...+(|V(C_m)|-1)+|V(C_{m+1})|+|V(C_{m+2})|+...+|V(C_n)|$, onde as componentes $\{C_1, C_2, ..., C_m\}$ representam os m caminhos de $C'(H)$ e as componentes $\{C_{m+1}, C_{m+2}, ..., C_n\}$ representam os ciclos de $C(H)$. Podemos ainda dizer

que $|M| = |V(C_1)|+|V(C_2)|+...+|V(C_n)|-m$. Se somarmos $|M|$ à $|C'(H)|$, teremos $|V(C_1)|+|V(C_2)|+...+|V(C_n)|-m+m = |V(C_1)|+|V(C_2)|+...+|V(C_n)| = |V(G)|$. □

Teorema 2.5 *Seja G um grafo orientado, G' o seu dobro, C um conjunto de caminhos e ciclos em G, disjuntos nos vértices, que cobrem $V(G)$ e $M = \beta(A(C))$. Então M é um emparelhamento em G'. Além disso*

$$|M| + |C'| = |V(G)|,$$

onde C' é o conjunto de caminhos de C.

Prova: Se os caminhos e ciclos de C cobrem $V(G)$, então $V(G) = V(C)$ e como $M = \beta(A(C))$, então $M \subseteq E(G')$. Temos que provar que qualquer duas arestas de M não são adjacentes em G'. Suponha que as arestas $a, b \in M$ sejam adjacentes em G'. Isto poderia ocorrer pelas partições $X(G')$ ou $Y(G')$. No primeiro caso, considerando $a = u_x v_y$ e $b = u_x t_y$, teríamos $d_G^+(u) = 2$ o que é impossível, por u fazer parte de um caminho ou de um ciclo de C. No segundo caso, considerando $a = u_x v_y$ e $b = t_x v_y$, teríamos $d_G^-(u) = 2$ o que também é impossível por v_y, da mesma forma, fazer parte de um caminho ou de um ciclo de C.

Para demonstrarmos a equação, podemos escrever $|V(C)| = |V(G)| = |V(C_1)| + |V(C_2)| + ... + |V(C_n)|$, onde $\{C_1, C_2, ..., C_n\}$ são todas os elementos do conjunto C. Podemos ainda, devido à bijeção β, escrever $|A(C)| = |M| = (|V(C_1)|-1)+(|V(C_2)|-1)+...+(|V(C_m)|-1)+|V(C_{m+1})|+|V(C_{m+2})|+...+|V(C_n)|$, onde $\{C_1, C_2, ..., C_m\}$ representam os m caminhos de $C'(H)$ e $(C_{m+1}, C_{m+2}, ..., C_n)$ representam os ciclos de $C(H)$. Podemos ainda dizer que $|M| = |V(C_1)| + |V(C_2)| + ... + |V(C_n)| - m$. Se somarmos $|M|$ à $|C'(H)|$, teremos $|V(C_1)| + |V(C_2)| + ... + |V(C_n)| - m + m = |V(C_1)| + |V(C_2)| + ... + |V(C_n)| = |V(G)|$. □

O teorema a seguir é importante pois relaciona-se na prática com o caso onde temos cobertura suficiente da molécula de DNA original para gerar um só caminho.

Teorema 2.6 *Seja G um grafo orientado onde existe um caminho hamiltoniano. Então cada emparelhamento máximo M^* do dobro de G corresponderá a uma coleção de caminhos e ciclos onde o número de caminhos será no máximo um.*

Prova: Como há um caminho hamiltoniano em G, este caminho produz um emparelhamento M de cardinalidade $|V(G)| - 1$ em G', o dobro de G. Portanto, um emparelhamento máximo M^* em G' terá $|M^*| \geq |V(G)| - 1$ arestas. O resultado então segue da fórmula 2.1. □

2.5 Sumário

Neste capítulo fizemos algumas formalizações sobre grafos de sobreposição e mostramos alguns resultados relacionando emparelhamento máximo a caminhos em grafos acíclicos. Vimos que a mesma idéia pode ser aplicada a grafos cíclicos, mas obtendo como resultado não só caminhos mas também ciclos no grafo. Discorremos brevemente sobre algumas formalizações já propostas para a montagem de fragmentos como o MSC e vimos como esta dissertação se relaciona ao MCP^r. No próximo capítulo veremos como todos os resultados teóricos foram postos em prática.

Capítulo 3

Implementações

3.1 Introdução

Nosso principal objetivo, desde o início deste trabalho, foi não só desenvolver técnicas, mas efetivamente construir programas que pudessem auxiliar na montagem de fragmentos. Este capítulo é totalmente dedicado a descrever a implementação que realizamos para um conjunto de algoritmos constituintes da ferramenta que denominamos de `concam` (**con**tigs através de **cam**inhos em grafos de sobreposição). Estes algoritmos se baseiam no modelo do MCP^r descrito no capítulo anterior, mas diferenciando-se em diversos aspectos do trabalho de Ferreira, Souza e Wakabayashi como poderá ser visto adiante.

Ao longo das seções, o conjunto de fragmentos $\mathcal{F} = \{F1, F2, F3, F4, F5\}$ auxiliará no acompanhamento das descrições deste capítulo, onde a sequência de cada fragmento será $F1$ = `GGACCAGTGAAA`, $F2$ = `CTAACAGTGTAGTGCGCT`, $F3$ = `CCTGCGCCACTATAATGG`, $F4$ = `TAGTGTACC` e $F5$ = `TTGTGGGACGTTT`. Eventualmente, para definições em que este simples exemplo não se mostrar muito adequado, lançaremos mão de algum outro. Em todo o capítulo consideraremos sobreposições de tamanho pelo menos 3 para todos os exemplos.

3.2 Formato FASTA

Existe um padrão mundial para arquivos texto que representam sequências e informações de fragmentos chamado *formato* `FASTA` [10]. Já que faremos bastante uso deste padrão é fundamental defini-lo.

```
>F1
GGACCAGTGAAA
>F2
CTAACAGTGTAGTGCGCT
>F3
CCTGCGCCACTATAATGG
>F4
TAGTGTACC
>F5
TTGTGGGACGTTT
```

Figura 3.1: Exemplo de arquivo em formato FASTA.

Uma sequência em formato `FASTA` começa com uma linha de descrição, seguida por uma ou mais linhas com dados da sequência. A linha de descrição é diferenciada das linhas de dados através do caractere "`>`" na primeira coluna. Por sua simplicidade, um pequeno exemplo será suficiente para esclarecer esta representação. Na figura 3.1 podemos visualizar o arquivo `FASTA` para o conjunto $\mathcal{F}$. No decorrer desta seção mais exemplos serão ilustrados.

3.3 Obtenção e Preprocessamento das Sequências

3.3.1 Lendo Cromatogramas

Para que possamos dar início ao processo de montagem com nossa ferramenta, devemos primeiramente obter dois arquivos em formato FASTA. Um deles contendo as sequências dos fragmentos e o outro contendo as qualidades das bases. Qualidades são números inteiros entre 0 e 100. Quanto maior o número, maior a qualidade. A probabilidade de erro numa base está associada à sua qualidade pela fórmula $p = 10^{-q/10}$ [23, 8].

Antes de mais nada, devemos realizar `base calls` sobre os cromatogramas originados dos laboratórios. Para esta tarefa utilizamos o programa `phred` [23, 8, 7, 20]. Este programa gera, para cada cromatograma, um arquivo texto denominado de *arquivo phd* [23, 8, 7, 20], em determinado formato, contendo as bases do fragmento e as respectivas qualidades dessas bases.

Em seguida, executamos os programas `phd2fasta` e `cross_match` [20], onde o

```
Sequências:
>F1
XGACCAGTGAXX
>F2
XXXACAGTGTAGTGCGCX
>F3
XXTGCGCCAXTXXXATGX
>F4
TAGTGTACC
>F5
XXGTGGGACGTTT

Qualidades:
>F1
16 25 21 22 30 40 45 43 35 25 10 8
>F2
4 4 4 10 15 12 18 23 30 50 49 50 45 47 25 20 15 7
>F3
6 7 18 25 25 40 48 42 18 12 14 10 8 8 8 5 4 4
>F4
4 6 6 15 25 30 28 32 8
>F5
5 5 8 5 10 12 15 33 12 10 20 20 20
```

Figura 3.2: Arquivos de sequências com vetor indentificado e de qualidades.

primeiro, a partir dos arquivos phd, gera os dois arquivos FASTA que necessitamos. O segundo, partindo do arquivo de sequências gerado pelo phd2fasta, faz comparações aproximadas entre as sequências de cada fragmento e as sequências dos vetores utilizados para clonagem, de modo a identificar possíveis trechos nas extremidades desses fragmentos que pertençam ao DNA do vetor. É produzido então um arquivo de saída, também em formato FASTA, que corresponde ao mesmo arquivo de sequências que o phd2fasta produziu, com a única diferença de que as bases das sequências, identificadas como pertencentes ao DNA do vetor, serão substituídas pelo caractere "X". Na figura 3.2 temos os arquivos de sequências e qualidades relativos ao conjunto $\mathcal{F}$, considerando cromatogramas e vetores hipotéticos.

Uma maneira interessante de fazermos esta avaliação e que foi utilizada no programa é, partindo do início de uma das extremidades por vez de um determinado fragmento, percorrermos pequenos intervalos de tamanhos fixos, calculando a média das qualidades das bases relativas a cada intervalo. Se a média do primeiro intervalo for menor que um valor predefinido, calculamos a média de qualidade das bases do próximo intervalo, fazendo nova verificação. Quando houver algum intervalo cuja média de qualidade das bases seja maior ou igual a uma *qualidade mínima aceitável*, paramos e recomeçamos o mesmo teste para a outra extremidade, de modo que ao final tenhamos as posições à esquerda e à direita, de cada sequência, que demarcam o *trecho de boa qualidade* do fragmento. Aos pequenos intervalos utilizados para avaliação, daremos o nome de *janela de qualidade*.

Ao tentarmos fazer a identificação do trecho de boa qualidade de um determinado fragmento, caso na avaliação de alguma extremidade, percorrermos toda a sequência e não encontrarmos nenhuma janela de qualidade aceitável, então este fragmento é excluído da montagem por não apresentar confiabilidade. Outra situação que também causa a exclusão de um fragmento é quando definimos um suposto trecho de boa qualidade, porém ao verificarmos a média de qualidade das bases deste trecho, o valor é menor que a qualidade mínima aceitável ou ainda o tamanho do trecho é tão pequeno que seu alinhamento com trechos de outros fragmentos nem mesmo será reportado nas comparações dois a dois que ocorrerão posteriormente.

Teremos então cinco entradas para o programa `clip.pl`. A primeira e a segunda são os arquivos `FASTA` gerados pelo `phd2fasta`, mostrados na seção 3.3.1. A terceira e a quarta são o tamanho da janela e a qualidade mínima aceitável para os intervalos, tendo como valores padrões 10 e 20 respectivamente. Finalmente a última entrada é o tamanho mínimo para os trechos de boa qualidade dos fragmentos.

As saídas são dois arquivos, onde um contém, para cada sequência, o intervalo relativo ao trecho próprio, bem como o intervalo da região de boa qualidade. O segundo arquivo possui as sequências dos trechos próprios, que são as que realmente usaremos para montar, de fragmentos que não foram excluídos por baixa confiabilidade.

Executando o programa `clip.pl` nos arquivos da figura 3.2, mas considerando tamanho da janela e qualidade mínima aceitável como sendo respectivamente 3 e 20, teremos os dois arquivos da figura 3.3 como saída. Note que o fragmento $F5$ foi excluído, pois seu suposto intervalo de boa qualidade, que vai da posição 7 até a 13 possui média inferior a 20. Nosso novo conjunto após o corte do DNA dos vetores e exclusão dos fragmentos não confiáveis, se chamará C.

```
Intervalos:
>F1 2 10 2 10
>F2 4 17 7 17
>F3 3 9 3 9
>F4 1 9 4 9

Sequências Sem Vetores:
>F1
GACCAGTGA
>F2
ACAGTGTAGTGCGC
>F3
TGCGCCA
>F4
TAGTGTACC
```

Figura 3.3: Arquivos de intervalos e de sequências sem vetores.

3.4 Grafo de Sobreposição

Passaremos agora a descrever como colocamos em prática as definições da seção 2.2, através do programa `montaGrafo.pl` descrevendo funcionamento, entradas e saídas deste programa.

3.4.1 Comparação Dois a Dois

Para fazermos a montagem do grafo de sobreposição, surge a necessidade de um algoritmo para comparação dois a dois entre todos os fragmentos de nosso conjunto, definindo sobreposições aproximadas entre os pares. Novamente fazemos uso do programa `cross_match`. Os seguintes parâmetros são usados na execução deste programa:

```
cross_match -masklevel 101 -tags -alignments <arquivo-FASTA>
```

A opção `-masklevel 101` garante que todas os alinhamentos possíveis sejam reportados. Então, por exemplo, se o fragmento F_i alinha-se com o fragmento $\overline{F_j}$, então certamente F_j alinha-se com $\overline{F_i}$. Neste caso, nos interessam os dois alinhamentos. Este valor de 101 nos garante o aparecimento dos dois na saída do `cross_match`. No

entanto, quando F_i alinha-se com F_j, o `cross_match` não mostra os alinhamento de $\overline{F_j}$ com $\overline{F_i}$, havendo a necessidade, portanto, de criarmos isto por nossa própria conta.

A opção `-tags` serve para auxiliar a análise da saída em formato texto que o `cross_match` fornece, pois em cada linha desta saída onde são reportadas as posicões iniciais e finais do alinhamento entre os fragmentos, é incluída a palavra `ALIGNMENT`, facilitando a localização das informações que mais interessam.

A opção `-alignments` é para que no relatório sejam incluídos os alinhamentos entre os fragmentos, pois esses serão necessários mais à frente quando da montagem dos contigs. Abaixo, um exemplo bem simples do trecho principal da saída mostrada pelo `cross_match`, relativa ao par $F2$ e $F3$ do conjunto $\mathcal{C}$.

```
ALIGNMENT    4  0.00 0.00 0.00  F2     10    14 (0)  F3      1     5 (2)

  F2                    10 TGCGC 14

  F3                     1 TGCGC 5
```

A linha começando com a palavra `ALIGNMENT` diz, basicamente, que o alinhamento entre $F2$ e $F3$, em $F2$, começa na posição 10 e termina na posição 14, sobrando 0 bases e em $F3$, começa da posição 1, termina na posição 5, sobrando 2 bases no final. As sobras aparecem entre parênteses. Logo abaixo desta linha, os trechos alinhados de ambos os fragmentos são reportados. O valor 4 é uma pontuação ou *score* [27, 25, 24] associado à sobreposição. Os três valores 0.00 indicam, respectivamente, que houve 0.00% de substituições na região de sobreposição, 0.00% de deleções (na primeira sequência com relação à segunda) na região de sobreposição e 0.00% de inserções (na primeira sequência com relação a segunda) na região de sobreposição.

3.4.2 Vértices

Os vértices do grafo são definidos para o conjunto $\mathcal{C}$, de acordo com as descrições da seção 2.2.2, mas com tratamento um pouco diferenciado para identificar vértices de Steiner, de modo a contornar erros de sequenciamento.

Como já mencionado anteriormente, as extremidades dos fragmentos não são muito confiáveis. Então em muitos casos não teremos uma sequência sendo subcadeia de outra quando na verdade era para ser. Veja o alinhamento abaixo. O fragmento $F4$ deveria ser coberto por $F2$, no entanto o alinhamento entre os dois não reflete isto. Vemos que somente $66,67\%$ de $F4$ é coberto, pois existem *pontas indesejadas* à

direita e à esquerda deste fragmento, que ficam de fora da sobreposição. A primeira providência para contornar esta situação é analisar o intervalo de boa qualidade identificado anteriormente. Sabemos que este intervalo em $F4$ vai da posição 4 a 9. Sabemos ainda que o intervalo de sobreposição em $F4$, no alinhamento abaixo, vai de 2 a 7, de modo que uma base à esquerda e duas à direita ficaram de fora da sobreposição. Mas a base à esquerda encontra-se fora do intervalo de boa qualidade, então podemos simplesmente desconsiderá-la. Restam as duas bases da direita. Neste caso, as bases estão dentro do intervalo de boa qualidade. O que fazemos em nosso programa é admitir uma cobertura que não necessariamente seja de 100%, mas sim um valor passado por parâmetro denominado de *mínima cobertura*. No alinhamento de $F2$ com $F4$, se considerarmos cobertura de pelo menos 75%, $F4$ será considerado subcadeia de $F2$. Note que o tamanho considerado de $F4$ para este cálculo será de 8 e não de 9, pois a base da esquerda já foi desprezada por estar fora da região de boa qualidade. Mais adiante veremos quais são os bons valores de mínima cobertura, pois quando se considera uma porcentagem alta, deixamos de considerar muitos vértices como sendo de Steiner, quando eram para ser. O oposto também ocorre, uma porcentagem baixa causa a ocorrência de muitos vértices como sendo de Steiner quando na verdade não eram para ser.

```
F2    ACAGTGTAGTGCGC
F4    -TAGTGTACC----
```

3.4.3 Arcos

Para determinação dos arcos temos o mesmo problema com relação aos vértices de Steiner. Em vários casos um sufixo de uma sequência não será idêntico a um prefixo de alguma outra sequência em situações onde isto deveria acontecer. No alinhamento mostrado abaixo podemos ver um exemplo entre o par $F1$ e $F2$ do conjunto $\mathcal{C}$. Vemos que à direita de $F1$ e à esquerda de $F2$ sobraram pontas indesejadas de tamanho 1 que não tiveram suas bases casadas. Tratamos esses casos de maneira similar aos de vértices de Steiner. Verificamos se estas pontas estão fora do intervalo de boa qualidade de cada fragmento. Se isto for verdade, podemos desprezá-las. A base que sobrou à esquerda de $F2$ é um exemplo. Para a base que sobrou à direita de $F1$ isto não é verdade, então recorremos a outro parâmetro do programa `montaGrafo.pl`. Este parâmetro indica uma *tolerância* baseada na sobreposição em questão, de modo a obtermos um valor máximo aceitável para o tamanho das pontas indesejadas. Se a tolerância for de 20% então, visto que no caso abaixo a sobreposição é de tamanho 5, o valor máximo aceitável para as pontas indesejadas será de 1 e portanto podemos

considerar que para o alinhamento abaixo devemos acrescentar um arco no grafo de sobreposição.

```
F1    GACCAGTGA------
F2    --ACAGTGTAGTGCGC
```

3.4.4 Formato do Grafo

O programa `montaGrafo.pl` produz dois arquivos texto como saída. O primeiro contém as regiões de sobreposição de cada par, retiradas da saída do `cross_match`. As informações deste arquivo serão imprescindíveis na hora de montar os contigs referentes a cada caminho encontrado, como veremos em outras seções.

O outro arquivo produzido é uma representação do grafo de sobreposição. Seu formato consiste em três tipos de linhas. O primeiro tipo começa pelo caractere "d", indicando o número de vértices e arcos do grafo. Existem também linhas começando pelo caractere "f" com informações sobre os fragmentos. Cada uma informa o nome do fragmento, o rótulo s do vértice que representa este fragmento no grafo (deverá ficar subentendido que o rótulo do vértice que representará o complemento reverso do fragmento será $\overline{s}$) e se este vértice é terminal ou de Steiner (do mesmo modo para $\overline{s}$). O terceiro e último tipo de linha começa com o caractere "a", onde cada uma representa um arco no grafo.

Veja na figura 3.4 o formato do grafo de sobreposição para o conjunto $\mathcal{C}$ referente ao desenho da figura 3.5. Note que utilizamos o caractere "t" para indicar que o vértice é terminal e "s"para indicar que o vértice é de Steiner. Além disso, como exemplos de arcos, a linha `a 1 2 u u` indica o arco $(1, 2)$ e a linha `a 2 1 c c` indica o arco $(\overline{2}, \overline{1})$. Os caracteres "u" e "c" vêm do inglês *uncomplemented* e *complemented* respectivamente.

3.5 Caminhos

3.5.1 Retirada dos Vértices de Steiner

Uma importante decisão deste trabalho foi a de como tratar os vértices de Steiner. A definição da seção 2.2.2 diz que se o vértice s é de Steiner é porque existe algum outro vértice t que o cobre. Então só precisamos cobrir t em nossa solução. No trabalho de Ferreira, Souza e Wakabayashi esta questão foi abordada buscando-se soluções que

```
d 8 4

f F1 1 t
f F2 2 t
f F3 3 t
f F4 4 s

a 1 2 u u
a 2 1 c c
a 2 3 u u
a 3 2 c c
```

Figura 3.4: Formato texto do grafo de sobreposição.

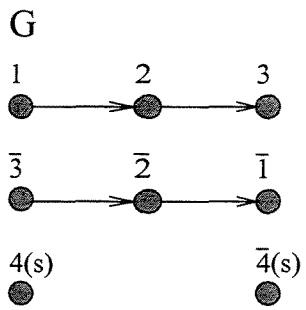


Figura 3.5: Grafo de Sobreposição.

não obrigassem a cobertura dos vértices de Steiner. Nós optamos por simplesmente retirar do grafo o conjunto de todos os vértices de Steiner e os respectivos arcos que tenham alguma extremidade neste conjunto de forma que sobrassem apenas os vértices terminais. Esta escolha não afetou os resultados, uma vez que conseguimos obter boas soluções em muitos dos exemplos testados e além disso a diminuição do grafo acarreta maior velocidade na execução dos programas. No grafo da figura 3.5 então, serão desconsiderados os vértices 4 e $\overline{4}$.

3.5.2 Emparelhamento Máximo

Para encontrar emparelhamentos máximos em grafos bipartidos fizemos uso de um programa chamado `bim_fifo` que faz parte de um conjunto de programas implementados no trabalho realizado por Setubal, Stolfi, Cherkassky, Martin e Goldberg [3].

Para utilizarmos o programa `bim_fifo` temos que primeiramente gerar o grafo dobro do nosso grafo de sobreposição, após a retirada dos vértices de Steiner, e colocar em formato que o programa entenda. Para resolver esta questão criamos o programa `inBim.pl` que em primeiro lugar faz a leitura do arquivo que representa o grafo, já retirando os vértices de Steiner e seus respectivos arcos. Do grafo resultante desta retirada, o programa cria o respectivo grafo dobro e o escreve em um arquivo de saída em formato adequado para o `bim_fifo`, que é executado gerando um novo arquivo contendo o emparelhameto encontrado. A figura 3.6 mostra o grafo dobro do conjunto $\mathcal{C}$ já sem os vértices de Steiner. É importante destacar que pode haver mais de um emparelhamento máximo para o mesmo grafo dobro, portanto mais de uma solução. Neste trabalho pegamos a primeira solução que o `bim_fifo` fornece sem verificar outras possibilidades, pois acreditamos que na prática isto não prejudique o resultado final. Os testes que veremos mais tarde dão sinais disto.

Daqui para frente, ao nos referirmos a $\mathcal{C}$, já consideraremos que a subcadeia $F4$ foi eliminada deste conjunto.

3.5.3 Retirada dos Vértices Complementares

No trabalho de Ferreira, Souza e Wakabayashi, é exigido que os caminhos no grafo de sobreposição cubram os vértices terminais ou seus complementos reversos. Então se o vértice s pertence a algum caminho, o vértice $\overline{s}$ certamente ficará de fora da solução. Em nosso caso, quando optamos por utilizar emparelhamento máximo, a resposta que o `bim_fifo` fornece produzirá caminhos que contenham tanto o vértice s quanto o $\overline{s}$.

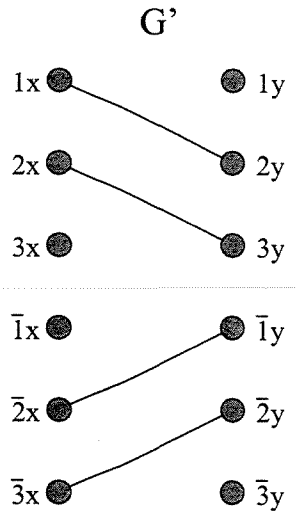


Figura 3.6: Grafo dobro.

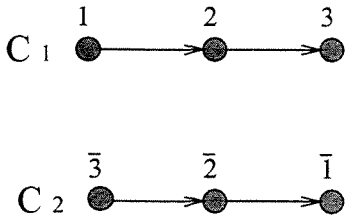


Figura 3.7: Caminhos derivados do emparelhamento máximo.

O programa `caminhos.pl`, que faz partes de nossas implementações, lê o emparelhamento fornecido pelo `bim_fifo`, definindo os caminhos no grafo de sobreposição e então executa uma heurística para retirada de vértices complementares que são cobertos por estes caminhos. Veja na figura 3.7 os caminhos relativos ao emparelhamento máximo do grafo dobro da figura 3.6.

A heurística funciona da seguinte forma. Seja o vértice s que participa do caminho C_i e seu complementar $\bar{s}$ que participa do caminho C_j. Se $C_i \neq C_j$ verificamos qual dos dois caminhos possui menor número de vértices. Supondo que seja o caminho C_j, retiraremos deste, o vértice $\bar{s}$ de forma que C_j se partirá em C_j e C_k. A idéia de escolher o menor caminho, para a retirada do vértice, é valorizar os caminhos mais longos, pois estaremos partindo o caminho mais curto, enquanto que o grande permanecerá intacto. Se por outro lado $C_i = C_j$, o vértice a ser retirado será o que for mais externo, ou seja, o mais próximo do vértice inicial ou do vértice final do caminho. Isto é feito mais uma vez pensando na valorização de caminhos longos, pois se o vértice retirado estiver bem próximo aos extremos do caminho, teremos como

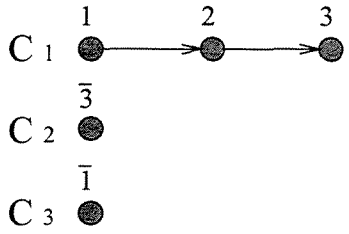


Figura 3.8: Eliminação de vértice complementar.

resultado da partição, um caminho pequeno e outro bem longo. Seguimos fazendo a verificação para todos os vértices até que só sobrem caminhos que cubram os vértices ou seus complementares. A figura 3.8 mostra como o caminho C_2 da figura 3.7 foi quebrado em C_2 e C_3 devido a retirada do vértice $\overline{2}$. Posteriormente, com a eliminação dos vértices $\overline{3}$ e $\overline{1}$, só sobrará o caminho C_1, que o programa `caminhos.pl` escreverá em um arquivo de saída no formato mostrado abaixo.

```
>p1
ve      1       2       3
```

3.5.4 Recombinando Ciclos e Caminhos

A maior dificuldade no problema de montagem de fragmentos é, sem dúvida, a ocorrência de regiões repetidas na molécula de DNA. Essas regiões causam ciclos no grafo de sobreposição e encontrar coberturas mínimas por caminhos de vértices disjuntos em grafos cíclicos é NP-difícil [9, 4]. Neste trabalho, tentamos encontrar uma forma de contornar o custo computacional destes casos.

A primeira idéia sobre como tratar casos cíclicos, foi identificar as componentes fortemente conexas [17] do grafo e condensá-las em um único vértice de modo que o grafo de sobreposição se tornasse acíclico. Deste grafo resultante encontraríamos os caminhos como descrito anteriormente, determinando uma solução S_1. Após isto, para cada componente condensada, encontraríamos uma solução individual S_i, unindo-a à S_1, resultando em uma única solução global. Efetivamos a implementação do algoritmo para identificar as componentes fortemente conexas, no entanto a resolução de problemas separados para cada componente não nos pareceu muito simples, ao passo que uma outra alternativa descrita a seguir se mostrava menos complicada e mais promissora, fazendo com que optassemos pela segunda abordagem.

Como vimos no teorema 2.6, se determinarmos o emparelhamento máximo no dobro de um grafo hamiltoniano, obteremos no máximo um caminho e o restante ciclos.

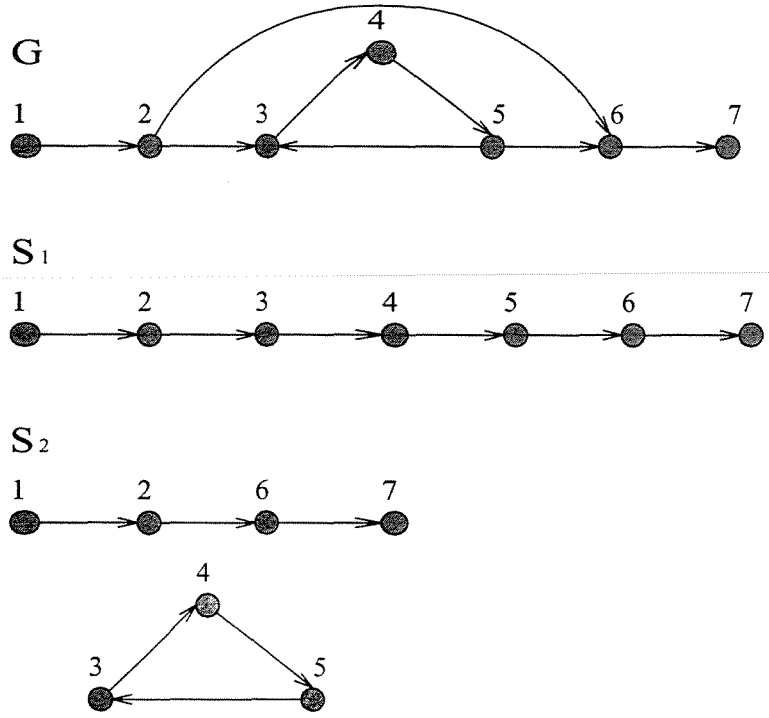


Figura 3.9: Exemplo de solução com ciclo.

A idéia então é tentar rearranjar esta solução de modo que os ciclos se recombinem ao caminho, caso haja algum. através de arcos que ficaram de fora da solução encontrada. Para entender melhor o que isto significa, a figura 3.9 mostra um grafo de sobreposição G, com duas soluções possíveis S_1 e S_2. Quanto a S_1 não há nenhum problema. No entanto, na solução S_2 podemos perceber o aparecimento de um ciclo. A figura 3.10 mostra como este ciclo pode ser recombinado ao caminho, utilizando os arcos $(2,3)$ e $(5,6)$, que não participaram da solução S_2, e eliminando os arcos $(2,6)$ e $(5,3)$.

Na prática, as componentes conexas dos grafos que montamos dificilmente serão grafos hamiltonianos devido à dificuldade, vista na seção 3.4.3, para a determinação de arcos. Haverá geralmente então, mais de um caminho nas soluções para cada componente. No entanto, a idéia permanece a mesma. Quando uma solução contém caminhos e ciclos, tentamos recombinar os ciclos nos caminhos, sendo que, em nossas implementações, para um ciclo c qualquer e um caminho p da solução, analisamos se c pode ser recombinado a p. Em caso negativo passaremos a verificação para outro caminho da solução. Se existir a possibilidade, no entanto, fazemos a recombinação de c com p sem verificar se há outro caminho que também possa ser recombinado, ou seja, o primeiro caminho que se identifique com nossas averiguações será utilizado

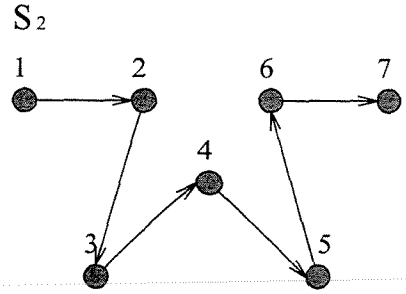


Figura 3.10: Recombinação de caminho e ciclo.

na recombinação. A análise prática neste caso também apresenta sinais de que esta atitude não causou danos ao resultado final.

Quando ocorre o caso em que algum ciclo não possa ser recombinado, este será aberto em algum vértice aleatório tornando-se um caminho. Veremos nos testes realizados que esta abordagem apresentou bons resultados. Além disso, poderemos notar uma certa tendência do número de ciclos ser baixo, de modo que a fase de recombinação não onere o tempo total de processamento. A idéia então não é evitar o surgimento de ciclos na solução. Deixamos que isto ocorra normalmente e em seguida aplicamos uma heurística para desfazer os ciclos.

3.6 Transformando Caminhos em Contigs

Uma vez determinados os caminhos no grafo, nossa tarefa passa a ser a obtenção dos respectivos contigs. Para isto fizemos uso dos programas `assemble` e `align` implementados por Moriya, Cruvinel e Jerozolimski [5]. Na verdade, houve a necessidade de realizarmos algumas adaptações no `assemble` para que nossos propósitos fossem atingidos.

O programa `assemble`, em sua versão original, realiza o alinhamento entre determinados pares (previamente selecionados por outro programa) através do uso de programação dinâmica [25, 24]. Para cada par uma avaliação é realizada no sentido de se afirmar se este par contribuirá na construção dos contigs sendo montados. Caso contribua, o par passa por uma etapa chamada de *unificação* [5] que ao juntar os fragmentos vai formando estruturas chamadas *grafos de base* [5], que são grafos acíclicos através dos quais são armazenados os fragmentos que se sobrepõem. Os nós do grafo são as bases de cada fragmento e as arestas possuem os rótulos que identificam os fragmentos aos quais as bases pertecem. Para cada contig a ser gerado, existirá um grafo de base. Todos os grafos deste tipo formados pelo `assemble` serão

escritos separadamente em arquivos de saída, que serão lidos pelo `align`. Este último simplesmente faz a tradução dos grafos de base em alinhamentos múltiplos. Cada alinhamento destes é escrito em um arquivo de saída, onde são reportados a sequência, a orientação e a posição inicial de cada fragmento no contig.

A modificação efetuada no `assemble` foi no seguinte sentido. Uma vez que o `cross_match` já fornece o alinhamento das regiões de sobreposição dos pares de fragmentos e além disso, já existe a definição de quais contigs devem ser formados (caminhos) e portanto quais pares devem ser utilizados, então bastaria tirar a etapa que realizava a programação dinâmica e ainda, aquela responsável pela escolha de quais pares seriam aceitos para fazerem parte dos contigs, pois já estaríamos fornecendo os alinhamentos dois a dois somente dos pares que realmente interessariam para nossa solução. Desta forma, garantidamente, só haveria unificação entre pares que fizessem parte do mesmo contig relativo aos caminhos determinados anteriormente, ou seja, haveria um grafo de base para cada caminho. A partir daí, o `align` executa sua função normalmente.

Para fornecer ao `assemble` os pares já alinhados para serem unificados, criamos o programa `inAssemb.pl` que lê o arquivo gerado pelo programa `caminhos.pl`. Para cada caminho o `inAssemb.pl` monta o alinhamento de cada par de vértices através das informações guardadas pelo `montaGrafo.pl` no arquivo que reporta as regiões de sobreposições dos fragmentos, fornececidas pelo `cross_match`. Abaixo podemos ver os alinhamentos que serão passados ao `assemble`, referentes ao caminho resultante de nosso exemplo do conjunto $\mathcal{C}$.

```
de F1 F2 1 2 u u 16 5 5
r1 GACCAGTGA-------
r2 --ACAGTGTAGTGCGC
de F2 F3 2 3 u u 16 5 4
r1 ACAGTGTAGTGCGC--
r2 ---------TGCGCCA
```

A linha que se inicia com "de", no primeiro caso, diz que o alinhamento é entre $F1$ e $F2$, relativo ao arco $(1,2)$ do caminho, pois as orientações estão indicadas como "u" e "u". Se fossem "c" e "c", por exemplo, o arco correspondente seria $(\overline{1},\overline{2})$. O número 16 é o tamanho total do alinhamento, o primeiro 5 é o tamanho da sobreposição e o último 5 o score do alinhamento calculado pelo `cross_match`. As linhas "r1" e "r2" mostram, respectivamente, os fragmentos $F1$ e $F2$ alinhados. No segundo alinhamento as mesmas informações são mostradas para o arco $(2,3)$.

O `inAssemb.pl` gera ainda um outro arquivo, em formato `FASTA`, para o programa `assemble`, contendo os fragmentos que foram cobertos pelos caminhos, quem em nosso exemplo são os fragmentos $\{F1, F2, F3\}$.

3.7 Obtenção e Visualização dos Consensos

Nesta seção descrevemos as idéias implementadas no programa `ace.pl` que tem como entradas os alinhamentos múltiplos representados pelos arquivos gerados pelo programa `align` e como saída um arquivo contendo os consensos e informações sobre estes em determinado formato, para a leitura de outro programa que realiza a visualização gráfica deste resultado.

3.7.1 Votação das Bases no Alinhamento Múltiplo

Definidos os alinhamentos múltiplos para cada caminho, nos resta agora um modo de descrever as sequências consenso correspondentes. Temos que, a partir dos alinhamentos múltiplos obtidos, criar um critério de votação que elegerá a melhor base para representar cada coluna desses alinhamentos. Em seguida mostraremos as idéias implementadas para resolver esta questão.

O que foi levado em maior consideração, na hora de definir a melhor base de cada coluna dos alinhamento múltiplos, foi a qualidade de cada base previamente calculada pelo programa `phred`. Escolhemos o seguinte critério inspirado no que faz o conceituado programa `consed` [11, 20]. Para uma determinada coluna i, seja uma base a, de qualidade q_1, pertencente a um fragmento A numa determinada orientação (complementado ou não). Seja ainda, para a coluna i, uma base b do mesmo tipo de a, com qualidade q_2, pertencente a um fragmento B, com orientação oposta à de A, tal que nenhuma outra base de i, que também seja do mesmo tipo de a e que pertença a algum outro fragmento também em orientação oposta à de A, possua qualidade maior que q_2. Então ao somarmos q_1 e q_2, obtemos o que chamaremos de *qualidade modificada* de a. Ao definirmos a qualidade modificada para todas as bases de i, a que possuir o maior valor será eleita.

Uma importante observação deve ser feita. Como mencionado anteriormente, sabemos de antemão a qualidade de todas as bases dos fragmentos que vamos montar, no entanto, por consequência dos alinhamentos realizados, houve o aparecimento de espaços nesses fragmentos. A esses espaços devem ser associadas qualidades para que possamos realizar o cálculo das qualidades modificadas de acordo com critério

```
G  A  C  C  A  G  T  G  A  (u)
25 21 22 30 40 45 43 35 25

      A  C  A  G  T  G  T  A  G  T  G  C  G  C  (u)
      10 15 12 18 23 30 50 49 50 45 47 25 20 15

                                    T  G  C  G  C  C  A  (u)
                                    18 25 25 40 48 42 18

-------------------------------------------------------

G  A  C  C  A  G  T  G  T  A  G  T  G  C  G  C  C  A
25 21 22 30 40 45 43 35 50 49 50 45 47 25 40 48 42 18
```

Figura 3.11: Qualidades modificadas.

de votação exposto acima. Seja um buraco e pertencente ao fragmento A. Sejam, neste mesmo fragmento, as bases a e b com qualidades q_1 e q_2, de modo que a é a base imediatamente à esquerda do buraco e e b a base imediatamente à direita do mesmo buraco. Então a qualidade qe de qualquer espaço que pertença ao buraco e, será calculada pela fórmula

$$qe = \left\lfloor \frac{q_1 + q_2}{2} \right\rfloor .$$

Na figura 3.11 mostramos o alinhamento múltiplo para os fragmentos $F1$, $F2$ e $F3$. O consenso se encontra abaixo da linha tracejada. Ao lado de cada sequência é indicada entre parênteses a orientação do fragmento.

A figura 3.12 exibe um exemplo mais completo para maior elucidação.

3.7.2 Visualização Gráfica dos Contigs - Arquivo ace

Desde o início do trabalho, existiu a preocupação de que, ao encontrar contigs para um determinado conjunto de fragmentos, houvesse a possibilidade de visualizar graficamente os consensos resultantes. Sabíamos que a análise gráfica seria muito mais eficiente que a textual, uma vez que facilita bastante a percepção de determinados detalhes que através de texto não detectamos tão facilmente. A vantagem se aplica até mesmo à aferição dos códigos produzidos, pois a constatação de possíveis erros também é melhor realizada de forma a se efetuar modificações nos programas o mais cedo possível.

```
                A  G  -  C  C  T  G  T  C  A  A  G  (u)
                6  8  11 15 18 25 23 25 22 17 12 9

          C  T  T  A  G  T  C  C  -  -  T  G  A  (u)
          9  8  10 15 16 19 25 22 14 14 7  8  5

                                  G  -  C  A  T  G  G  C  (c)
                                  12 13 15 23 18 12 15 10

                     T  C  C  T  G  T  C  C  T  G  (c)
                     8  5  10 15 15 19 24 18 14 8

                  G  T  T  T  T  G  T  G  C  T  G  G  C  A  T  (u)
                  7  5  10 7  12 18 26 23 28 20 15 10 8  11 6

   G  G  C  T  T  A  A  -  T  T  T  A  T  G  C  (u)
   10 8  9  15 17 20 18 22 27 23 19 11 15 10 7

-------------------------------------------------------------------

   G  G  C  T  T  A  A  T  C  C  T  G  T  C  C  T  G  G  C  A  T
   10 8  9  15 17 20 18 27 30 32 40 38 45 46 46 38 27 25 18 11 6
```

Figura 3.12: Qualidades modificadas para exemplo mais completo.

Para alcançar este objetivo, escolhemos o programa consed [11, 20] na versão 8.0 que lê arquivos texto chamados de ace [11] em um determinado formato. Estes arquivos devem possuir os consensos e suas principais informações, que o consed lê e traduz graficamente. É importante salientar, que esta versão do consed já está apta a ler o novo formato para os arquivos ace [11], que contém mais informações do que o formato anterior e em nosso trabalho já construímos neste novo padrão. Passemos às descrições e considerações que foram efetuadas para construir os arquivos ace.

Para facilitar a descrição do arquivo, as partes que o compõem serão chamadas de seções. Uma seção por vez será explicada e abreviações entre parênteses serão associadas às informações nela contida. Em seguida, o formato da seção será mostrado fazendo uso dessas abreviações.

Logo no início do arquivo vem a seção AS que fornece informações gerais como o número total de contigs (NTC) e o número total de fragmentos (NTF).

```
AS <NTC> <NTF>
```

Em seguida, para cada contig existente, um conjunto de seções é gerado a começar pela CO. Esta exibe informações sobre o contig como nome (NC), número de bases (NBC), número de fragmentos que o compõe (NF), número de segmentos de bases (NSB) que será explicado mais adiante, orientação do contig (OC), que pode ser U ou C do inglês *uncomplemented* e *complemented*, e finalmente a própria sequência consenso relativa ao contig (CTG). Duas observações devem ser feitas. Primeiro, fixamos OC sempre em U, pois o arquivo ace está sendo gerado pela primeira vez e ainda não sofreu nenhuma edição que pode ser feita via programa consed, como por exemplo a obtenção do complemento reverso do contig. Segundo, CTG é formatado com 50 colunas no máximo, ou seja cada linha possuirá no máximo 50 bases.

```
CO <NC> <NBC> <NF> <NSB> <OC>
<CTG>
```

Aparece então a seção BQ que exibe a qualidade de cada base do contig (QB) exceto para os eventuais espaços que ocorrerem no mesmo, pois o consed ao ler o arquivo já faz o cálculo das qualidades dos espaços automaticamente. A formatação mantém o padrão de 50 qualidades por linha, já que as mesmas se referem às bases do contig que também seguiram este procedimento, só que com a diferença de que há um espaço em branco entre os valores de qualidade.

```
BQ
<QB>
```

A seção AF é a próxima. Para cada fragmento que compõe o contig, surge uma linha começando com as letras AF seguidas das informações nome do fragmento (NOF), sua orientação U ou C (OF) e em que posição no contig ele começa (PC). Essas linhas aparecem na ordem crescente de PC.

```
AF <NOF> <OF> <PC>
```

A seguir, descrevemos os segmentos de bases do contig. É a seção BS. Para cada segmento ocorre uma linha, iniciando com as letras BS. Os segmentos de bases, são trechos do contig compostos de bases que foram eleitas do mesmo fragmento. Então se da posição i a j, no contig, com i menor ou igual a j, as bases vierem do mesmo fragmento A e tal que as bases das posições $i-1$ e $j+1$, se existirem, venham de outro fragmento diferente de A, temos um segmento de bases que começa em i (CS) e termina em j (FS), relativo a A (NOF). Os segmentos aparecem na ordem crescente de posições.

```
BS <CS> <FS> <NOF>
```

Neste ponto, para cada fragmento que fez parte da construção do contig, uma seção RD é gerada e para cada uma destas últimas são geradas as seções QA e DS. A RD informa o nome do fragmento (NOF), o número de bases (NBF), incluindo os eventuais espaços, dois outros campos (NE1 e NE2) que dizem respeito ao número de determinadas edições realizadas através do `consed` que serão fixados em zero, já que estamos construindo o arquivo pela primeira vez e por fim, mostra o fragmento (F) da maneira como aparece no alinhamento múltiplo, ou seja, com os espaços que possivelmente foram incluídos e na orientação usada para o alinhamento. Além disso, nas extremidades de F aparecerá, se for o caso, o DNA do vetor que inicialmente havia sido retirado. A formatação também será de no máximo 50 bases por linha.

```
RD <NOF> <NBF> <ED1> <ED2>
<F>
```

Na fase de formação dos contigs, alguns trechos nas extremidades de cada fragmento, por critérios de qualidade e/ou alinhamento, podem ser excluídos do processo de votação. A seção QA indica as posições iniciais (PIQ e PIA) e finais (PFQ e PFA) em F, dos trechos que devem ser levados em conta na votação segundo os dois critérios. Em nosso trabalho, consideramos que a única parte que deve ser excluída é, se ocorrer, o DNA do vetor como visto na seção 3.3.2. O restante deve participar da

construção do contig. Então apesar de utilizarmos os valores do arquivo de posições gerado pelo programa `clip.pl` para servir de informação na seção QA, não efetivamos o corte por qualidade (só fazemos uso dessas informações para construir o grafo como visto na seção 3.4).

```
QA <PIQ> <PFQ> <PIA> <PFA>
```

A seção DS, informa o cromatograma (AC) e o arquivo `phd` (AP) referentes a F, bem como a data e hora de geração do `phd` (DH).

```
DS CHROMAT_FILE: <AC> PHD_FILE: <AP> TIME: <DH>
```

Existem ainda, as seções WR, RT, CT e WA, mas estas só aparecem quando da edição do arquivo `ace` via programa `consed`. A figura 3.13 mostra o arquivo `ace` gerado para o conjunto $\mathcal{C}$. Temos ainda a figura 3.14 que mostra o resultado deste arquivo no programa `consed`, onde podemos ver o alinhamento entre os fragmentos do conjunto e o respectivo consenso.

3.8 Novas Iterações

Encontrar bons caminhos em grafos de sobreposição, depende em parte de montarmos esses grafos de modo que representem o mais próximo possível as verdadeiras sobreposições entre os fragmentos. Se não obtivermos bons grafos, os caminhos não corresponderão a boas soluções.

Vimos que a principal dificuldade na determinação de arcos e vértices de Steiner é o fato de que, geralmente, as sequências dos fragmentos possuem extremidades de má qualidade. Nossa primeira tentativa para contornar este problema, foi a criação da mínima cobertura e da tolerância descritos na seção 3.4. O uso desses valores realmente foi fundamental para continuação do trabalho, mas ainda assim havia em nossas soluções um número muito grande de caminhos, o que indicava que havíamos dispensado arcos que não deveriam ser dispensados e ainda deixávamos de definir muitos vértices como sendo de Steiner quando não era o caso. A primeira providência foi aumentar a tolerância e diminuir a mínima cobertura, de modo que minimizássemos as situações possuindo pontas indesejadas. Isto realmente causou melhoras, mas ainda não foi o ideal uma vez que ocorreu o oposto, ou seja, acarretou o surgimento de arcos que não deveriam existir e dispensou fragmentos que deveríam estar presentes na montagem. O próximo passo foi tentar utilizar as informações de qualidade que

```
AS 1 3

CO Contig1 18 3 3 U
GACCAGTGTAGTGCGCCA

BQ
 25 21 22 30 40 45 43 35 50 49 50 45 47 25 40 48 42 18

AF F2 U 0
AF F1 U 0
AF F3 U 10
BS 1 8 F1
BS 9 14 F2
BS 15 18 F3

RD F2 18 0 0
XXXACAGTGTAGTGCGCX

QA 4 17 7 17
DS CHROMAT_FILE: F2 PHD_FILE: F2.phd.1 TIME: Mon Nov 29 20:39:51 1999

RD F1 12 0 0
XGACCAGTGAXX

QA 2 10 2 10
DS CHROMAT_FILE: F1 PHD_FILE: F1.phd.1 TIME: Mon Nov 29 20:39:51 1999

RD F3 18 0 0
XXTGCGCCAXTXXXATGX

QA 3 9 3 9
DS CHROMAT_FILE: F3 PHD_FILE: F3.phd.1 TIME: Mon Nov 29 20:39:51 1999
```

Figura 3.13: Exemplo de arquivo ace.

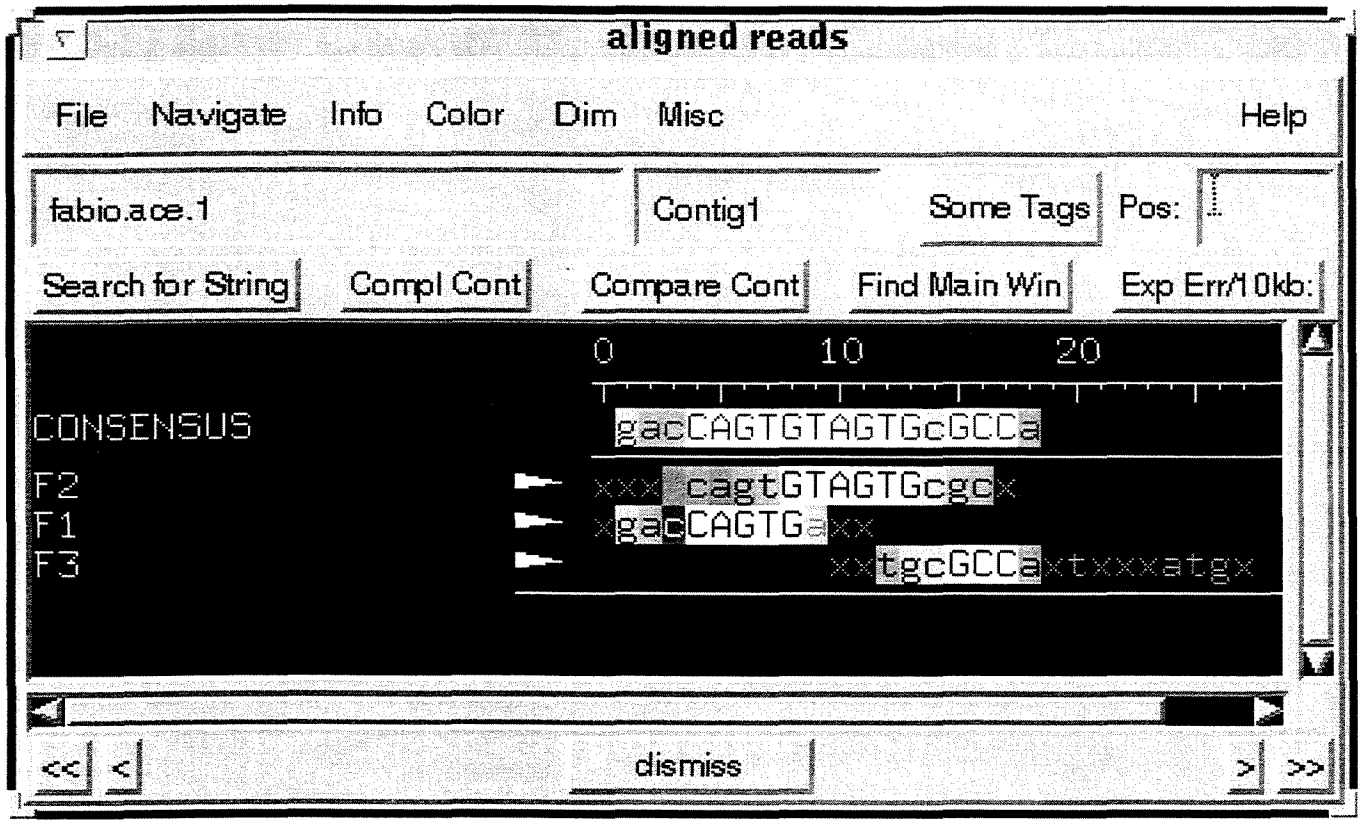


Figura 3.14: Visualização dos fragmentos e respectivo consenso.

o programa `phred` fornecia. Antes de começarmos a montagem, identificaríamos os extremos de baixa qualidade e faríamos o corte destes extremos, ficando somente com o trecho de boa qualidade dos fragmentos. Isto fez com que por um lado alguns arcos que não existiam em nossos exemplos passassem a existir. Mas por outro lado, muitas vezes apesar de termos extremidades com qualidade baixa em um determinado par de fragmentos, ainda assim existe coincidência entre as bases dos respectivos sufixo e prefixo, ou seja, o corte acabou por excluir muitos arcos que haviam antes, pois as extremidades que possibilitavam o surgimento desses arcos foram excluídas. Além disso, sequências que não deveriam ser subcadeias de outras, devido ao corte, passaram a ser e por isso foram indevidamente excluídas de nosso conjunto de montagem. Chegamos então à conclusão de que esses valores de qualidade não deveríam ser usados para fazer o corte, mas para somente auxiliar na decisão de afirmar se uma sobreposição representa arco, vértice de Steiner ou deve ser descartada. Com isso chegamos à solução que descrevemos na seção 3.4, onde mesclamos o uso das qualidades, a tolerância e a mínima cobertura.

A solução da seção 3.4 causou visíveis melhoras à determinação dos caminhos, mas estes continuavam numerosos e sabemos que o caso ideal é encontrarmos apenas um caminho, ou seja, apenas um contig. Mais uma vez, houve a tentativa de alterar alguns parâmetros, como a tolerância, a mínima cobertura, o tamanho da janela de qualidade e a qualidade mínima para as janelas. Nas várias experimentações, apesar de estarmos em melhor situação que nos métodos tentados anteriormente, ainda não tínhamos a situação ideal. Mais uma vez, para valores de tolerância muito baixos e de mínima cobertura muito alto, acabamos por dispensar muitos arcos e manter muitos vértices no grafo. Para valores altos de tolerância e baixos de mínima cobertura, aceitávamos arcos demais e descartávamos muitos vértices.

Todos estes experimentos nos levaram a acreditar que devido à natureza do DNA e aos erros de sequenciamento que descrevemos no primeiro capítulo, por mais que aferíssemos nosso método de montagem do grafo de sobreposição, sempre obteríamos um número de caminhos elevado em nossas soluções. Nos restou então a tentativa de unir os contigs relativos a esses caminhos.

Tentar fazer a união dos contigs resultou na idéia de realizar várias iterações na montagem. Na primeira execução, passaríamos a tolerância bem próximo de 0% e a mínima cobertura bem próxima de 100%, ou seja, estaríamos sendo bastante exigentes para definição de arcos e vértices de Steiner. Como dissemos anteriormente, essa atitude dispensará muitos arcos e manterá muitos vértices no grafo, portanto na primeira iteração obtemos um número elevado de contigs, mas estes terão grandes chances de não terem sofrido nenhuma distorção, já que os arcos e vértices que for-

maram o grafo são altamente confiáveis. Fazemos então uma nova iteração, onde o conjunto de fragmentos serão os consensos obtidos anteriormente. Para produzir os arquivos FASTA, um contendo as sequências dos consensos e outro as qualidades de suas bases, implementamos o programa `consenso2frag.pl` que lê o arquivo ace da primeira iteração e produz esses arquivos como saída. Na segunda iteração relaxamos os parâmetros anteriores, ou seja, aumentamos a tolerância e diminuímos a mínima cobertura de modo que o primeiro se distancie um pouco mais de 0% e o segundo se distancie um pouco mais de 100%. Desta forma alguns contigs que não puderam se unir na primeira iteração poderão fazê-lo na segunda e ainda, alguns contigs que deveriam ser subcadeias de outros na primeira execução, desta vez serão excluídos. Ainda assim não ocorrerão uniões e exclusões de todos. Então relaxamos mais ainda os parâmetros e realizamos nova iteração sobre os contigs da iteração anterior. Este processo continua com parâmetros que exijam cada vez menos exatidão na definição de arcos e vértices de Steiner até que obtenhamos um único contig ou até que se atinja um valor máximo de número de iterações. Note que a tolerância vai crescendo sem restrições, mas se a mínima cobertura chegar a 0, antes de terminar as iterações, permanecerá com esse valor até o fim. No próximo capítulo teremos oportunidade de verificar como os parâmetros afetam as soluções.

3.9 Sumário

Neste capítulo foram descritos todos os aspectos considerados no desenvolvimento do conjunto de programas da ferramenta `concam`. Vimos como os fragmentos são preprocessados antes do início da montagem. Em seguida mostramos a dificuldade para a montagem dos grafos de sobreposição devido às pontas indesejadas.

Discorremos ainda sobre como obter os caminhos, passando desde o programa utilizado para encontrar o emparelhamento máximo até a recombinação de ciclos que eventualmente apareçam na solução e a heurística utilizada para eliminação dos vértices complementares nos caminhos. Afirmamos que a retirada dos vértices de Steiner do grafo, não causa dano à solução e diminui o tamanho de nossas instâncias.

Descrevemos um pouco sobre os programas utilizados para produzir os alinhamentos múltiplos para cada contig e posteriormente mostramos o uso da qualidade modificada para calcular os consensos. Mostramos também o formato do arquivo ace utilizado pelo programa `consed` para visualização gráfica dos consensos obtidos.

Por último expomos alguns aspectos sobre a realização de novas iterações para a montagem, onde a cada iteração diminuímos o grau de exatidão para definirmos

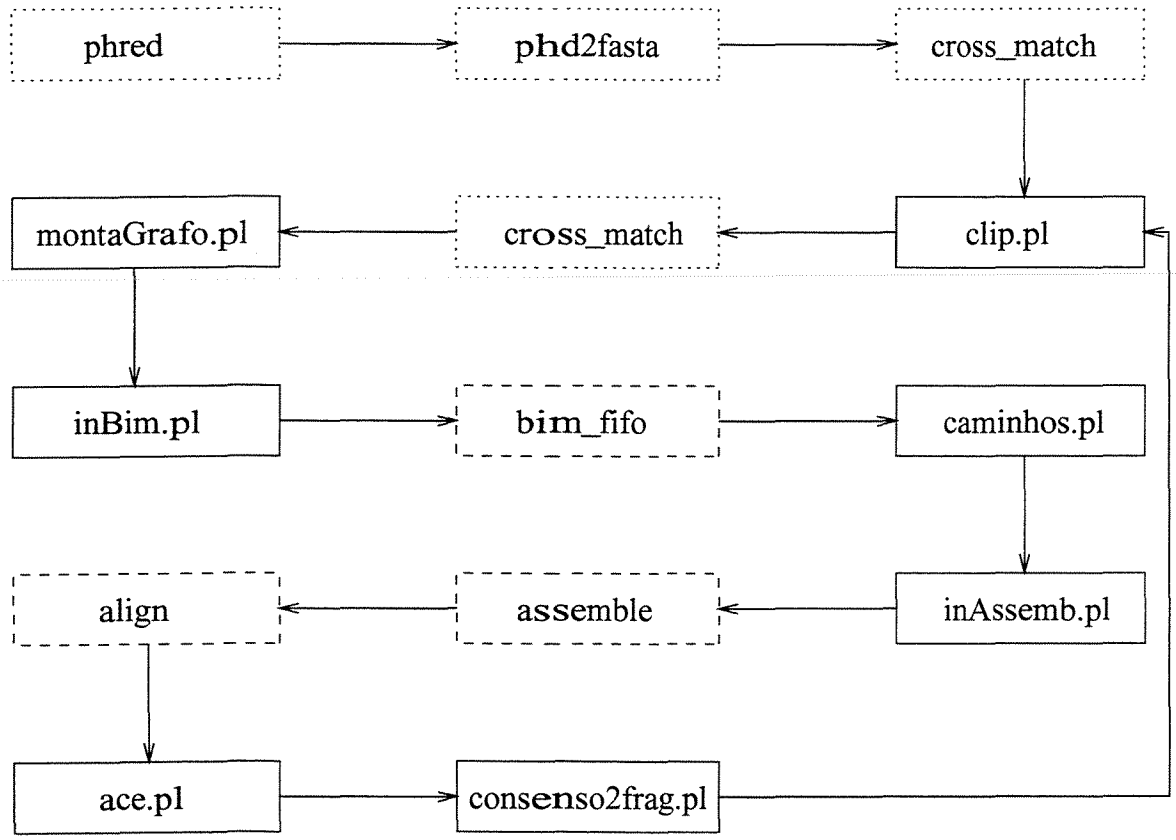


Figura 3.15: Esquema geral dos programas implementados e utilizados. Os retângulos de linha cheia indicam os programas implementados, os de linha tracejada mostram os programas auxiliares que fazem parte do concam e os de linha pontilhada definem também programas auxiliares mas que não fazem parte de nossa ferramenta. As setas indicam a ordem de execução dos programas.

arcos e vértices de Steiner. Na figura 3.15 mostramos todos os programas envolvidos no processo de montagem deste trabalho. Os retângulos de linha cheia indicam os programas implementados, os de linha tracejada mostram os programas auxiliares que fazem parte do concam e os de linha pontilhada definem também programas auxiliares mas que não fazem parte de nossa ferramenta. As setas indicam a ordem de execução dos programas.

Capítulo 4

Avaliação dos Algoritmos

4.1 Introdução

No capítulo anterior expomos alguns detalhes de implementação de nossos programas. Este capítulo tem por objetivo avaliar a ferramenta `concam`, mostrar como os algoritmos implementados, em conjunto com os programas auxiliares, se saíram na prática. Apresentamos alguns testes mostrando que para determinados parâmetros os resultados são bastantes significativos. Fazemos uma comparação textual e gráfica dos resultados do `concam` com os do conceituado programa `phrap` [20].

4.2 Complexidade dos Algoritmos

Analisando somente os pontos principais de nosso processo de montagem, podemos fazer uma divisão em oito etapas. Para discorrer sobre cada uma seguiremos a legenda

n - Número de cromatogramas.

n_2 - Número de fragmentos após o preprocessamento.

L - Tamanho máximo dos fragmentos.

T - Tamanho esperado da sequência original.

V - Número de vértices do grafo de sobreposição sem os vértices de Steiner.

A - Número de arcos do grafo de sobreposição sem os vértices de Steiner.

t_{ci} - Número de vértices terminais em ciclos.

t_{ca} - Número de vértices terminais em caminhos.

t_f - Número de vértices terminais após retirada de vértices complementares.

A primeira etapa é a leitura dos cromatogramas, de modo a obter os arquivos de sequências e qualidades. A complexidade é linear no tamanho da entrada, ou seja, $O(nL)$. Do mesmo modo será o preprocessamento dos fragmentos, que constitui a segunda etapa.

Na terceira etapa, relativa à comparação dois a dois dos fragmentos que não foram eliminados no preprocessamento, cada sequência é comparada a todas as outras e portanto temos $O(n_2^2 L^2)$. Na quarta, quando da montagem do grafo de sobreposição, temos a mesma complexidade já que lemos os alinhamentos reportados na fase anterior.

A quinta etapa é o cálculo do emparelhamento máximo pelo `bim_fifo`, para o grafo de sobreposição já sem os vértices de Steiner e seus respectivos arcos, que segundo consta no trabalho de Setubal, Stolfi, Cherkassky, Martin e Goldberg é $O(\sqrt{V}A)$ [3].

Após a definição do emparelhamento máximo e portanto dos caminhos e possíveis ciclos que este emparelhamento representa no grafo de sobreposição, passamos para a recombinação dos ciclos com os caminhos, representando a sexta etapa. Seja um ciclo C_i da solução. Até que consigamos encontrar um caminho para recombinar este ciclo, verificamos para cada vértice de C_i todos os vértices de um caminho P_i para concluir se é possível ou não a recombinação de C_i com P_i. Então para o pior caso podemos dizer que esta etapa é $O(t_{ci}t_{ca})$.

A sétima etapa é relativa à produção dos alinhamentos múltiplos para cada caminho resultante da fase anterior. No trabalho de Moriya, Cruvinel e Jerozolimski já que, em nossa adaptação, eliminamos o uso da programação dinâmica no programa `assemble`, o programa `align` passou a ser dominante, tendo complexidade de $O(t_f L + T \log T)$ [5].

A última etapa consiste no cálculo dos consensos. Para cada sequência de cada alinhamento múltiplo, percorremos as bases e qualidades de modo a calcular a maior qualidade modificada de cada coluna. Temos então $O(t_f L)$. A tabela 4.1 resume esta seção.

Tabela 4.1: Complexidades das principais etapas da montagem.

Etapas	**Complexidades**
Leitura dos cromatogramas	$O(nL)$
Preprocessamento dos fragmentos	$O(nL)$
Comparação dois a dois	$O(n_2^2L^2)$
Montagem do grafo de sobreposição	$O(n_2^2L^2)$
Emparelhamento máximo do grafo dobro	$O(\sqrt{V}A)$
Recombinação de ciclos	$O(t_{ci}t_{ca})$
Alinhamentos múltiplos	$O(t_fL + T\log T)$
Cálculo do consenso	$O(t_fL)$

4.3 Testes Comparativos

Os dados para testes são muito importantes no sentido de realizar uma avaliação confiável dos programas. Em todos os testes utilizamos conjuntos de fragmentos pertencentes ao repositório do projeto Genoma *Xylella fastidiosa*, portanto os dados estão sujeitos a todos os contratempos que descrevemos no primeiro capítulo e assim poderemos visualizar o comportamento dos programas implementados para casos reais de montagem. Serão usados quatro conjuntos de fragmentos, onde cada um representa regiões distintas no genoma. São eles `OOI03`, `07A01`, `07C03`, `11A02`, sendo o primeiro um *plasmídeo* [22, 25, 12] e os outros três *cosmídeos* [22, 25, 12]. Esses nomes são alusões aos vetores utilizados para clonar a região do genoma a que cada conjunto pertence. Os plasmídeos são utilizados para clones de regiões pequenas, enquanto que os cosmídeos para clones mais longos.

Em todos os testes a mínima sobreposição foi de 30, o número máximo de iterações foi 10, o tamanho da janela de qualidade foi 10, a mínima qualidade para as janelas foi 20 e o tamanho mínimo do trecho de boa qualidade para cada fragmento foi de 100. Exploramos somente os parâmetros mais críticos que são a tolerância, indicado por *tol*, a mínima cobertura, indicado por *min*, e a variação destes a cada iteração através de suas taxas de crescimento e decrescimento que indicaremos por *inc* e *dec* respectivamente. Na primeira iteração são atribuídos valores para *tol* e *min*, a partir de então uma determinada iteração i terá tolerância $tol_i = tol_{i-1}inc$ e mínima cobertura $min_i = 100 - (100 - min_{i-1})dec$. Na verdade mostraremos neste capítulo somente os resultados obtidos com os parâmetros que consideramos os mais apropriados. No apêndice A são exibidos mais testes variando *tol*, *min*, *inc* e *dec*.

A máquina utilizada para todos os testes desta dissertação foi uma `UltraSparc`

Tabela 4.2: Informações sobre os conjuntos utilizados para testes.

Nome	# Cromat.	Tam(bp)	Repet.	# Contigs phrap	T(s) phrap
00I03	17	1750	não	1	1
07A01	632	40000	não	1	49
07C03	1067	40000	sim	1	104
11A02	1037	40000	sim	1	103

Regiões repetidas:
751 0.00 0.00 0.00 07C03 14831 15588 (24804) 07C03 19820 20577 (19815)
682 11.71 0.09 0.09 11A02 35115 36190 (3887) 11A02 35490 36565 (3512)

rodando `SunOS 5.6`, com 2 processadores de 300MHz e 1GB de memória RAM.

Na tabela 4.2 temos algumas informações sobre os conjuntos, como o número de cromatogramas, o tamanho aproximado em pares de base da respectiva região do genoma, se há repetições ou não nesta região, o número de contigs produzidos pelo `phrap` e o tempo gasto pelo mesmo programa para produzir os contigs. Logo abaixo da tabela é mostrada a comparação feita pelo `cross_match` para os conjuntos `07C03` e `11A02` de seus respectivos contigs com eles próprios, para identificação de regiões repetidas. Vemos que para o conjunto `07C03`, por exemplo, há duas regiões repetidas de tamanho 757. Uma vai da posição 14831 até 15588 e a outra da posição 19820 até 20577. No caso do `11A02` as regiões repetidas não têm casamento exato, pois vemos que, por exemplo, houve 11.71% de substituições na região de sobreposição. A descrição de como interpretar a saída do `cross_match` encontra-se na seção 3.4.1 do capítulo 3.

Os resultados de alguns dos testes realizados podem ser visualizados nas tabelas 4.3, 4.4, 4.5 e 4.6. Todos foram executados com $tol = 1$, $min = 99$, $inc = 3$ e $dec = 3$ que são os parâmetros que apresentaram melhores resultados dentre as outras variações testadas. Isso se deve ao fato de que iniciando a execução com essa tolerância e mínima cobertura, apesar da possibilidade da perda de alguns arcos importantes e por conseguinte alguns contigs não se ligarem inicialmente, ainda assim poderemos afirmar que os arcos e vértices de Steiner definidos para o grafo são extremamente confiáveis diminuindo em muito a possibilidade de haver desvio de uma boa solução. Os valores de *inc* e *dec* iguais a 3, neste caso, auxiliam no relaxamento cada vez maior de *tol* e *min* nas iterações posteriores, fazendo com que os contigs se liguem o mais cedo possível, ou seja, com um número baixo de iterações. Certamente que essas taxas não podem ter valores muito maiores que 3, pois neste caso

a precisão na definição do grafo cairá muito rapidamente. Nos outros testes descritos no apêndice A, pode ser constatado, para alguns casos, que quando iniciamos a execução com valores de *tol* e *min* maior que 1 e menor que 99, respectivamente, algumas diferenças aparecem nas soluções obtidas pelo `concam` em relação às obtidas pelo `phrap`. Isto ocorre principalmente nos conjuntos `07C03` e `11A02` que são os que possuem maior número de fragmentos e ainda regiões repetidas.

Para as colunas das tabelas com os resultados dos testes, segue a legenda

It. - Iteração.

v - Número de vértices do grafo de sobreposição G.

a - Número de arcos de G.

t - Número de vértices terminais de G.

s - Número de vértices de Steiner de G.

V - Número de vértices de G após a retirada dos vértices de Steiner.

A - Número de arcos de G após a retirada dos vértices de Steiner.

Cíclico - Indica se o grafo sem os vértices de Steiner é cíclico ou não.

ci - Número de ciclos na solução.

Rec. - Indica se os eventuais ciclos puderam ser recombinados ou não.

C - Número de contigs produzidos.

T - Tempo de execução.

Cada linha contém informações de cada iteração. Abaixo das tabelas estão indicados o tempo total de execução e ainda a comparação realizada pelo `cross_match` entre o consenso gerado pelo `concam` e o gerado pelo programa `phrap`. O caractere "C", que em alguns casos aparece nesta comparação, indica que a sobreposição se deu com o complemento reverso do consenso gerado pelo `phrap`. Vemos que em todos os casos o consenso do `concam` cobre o consenso do `phrap`, ocorrendo maiores diferenças somente nas extremidades das soluções. Outro fato é que, exceto pelo conjunto `00I03`, as soluções também tiveram algumas diferenças nas regiões internas dos consensos gerados, por exemplo, comparando os resultados do conjunto `07A01` vemos que ocorreu 0.50% de substituições na região de sobreposição, 0.09% de deleções na

Tabela 4.3: Resultado da execução para o conjunto 00I03.

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	34	124	14	20	14	36	não	-	-	1	3.74

Número de Fragmentos eliminados no preprocessamento: 0

Tempo total: 3.74s

Comparação:
1728 0.00 0.00 0.00 concam 14 1761 (0) C phrap (0) 1748 1

região de sobreposição, na solução do `concam` com relação à solução do `phrap`, 0.10% de inserções na região de sobreposição, na solução do `concam` com relação à solução do `phrap`. Acontece que essas diferenças internas são muito pequenas de modo que podemos considerar bem próximas as soluções dos dois programas.

A figura 4.1 mostra as comparações graficamente para melhor compreensão. Para cada caso, o primeiro desenho corresponde ao consenso do `concam`, enquanto que no segundo mostramos a solução do `phrap`. As regiões sombreadas indicam as similaridades. Linhas ligam o ínicio e o fim da sobreposição nas duas soluções. Perceba que o conjunto `00I03` corresponde a um trecho do DNA muito menor que os trechos relativos aos outros conjuntos e por isso sua escala é distinta. A diferença, para este conjunto, que percebemos no desenho entre as respostas dos dois programas é relativa à 13 bases, como visto no resultado do programa `cross_match` na tabela 4.3.

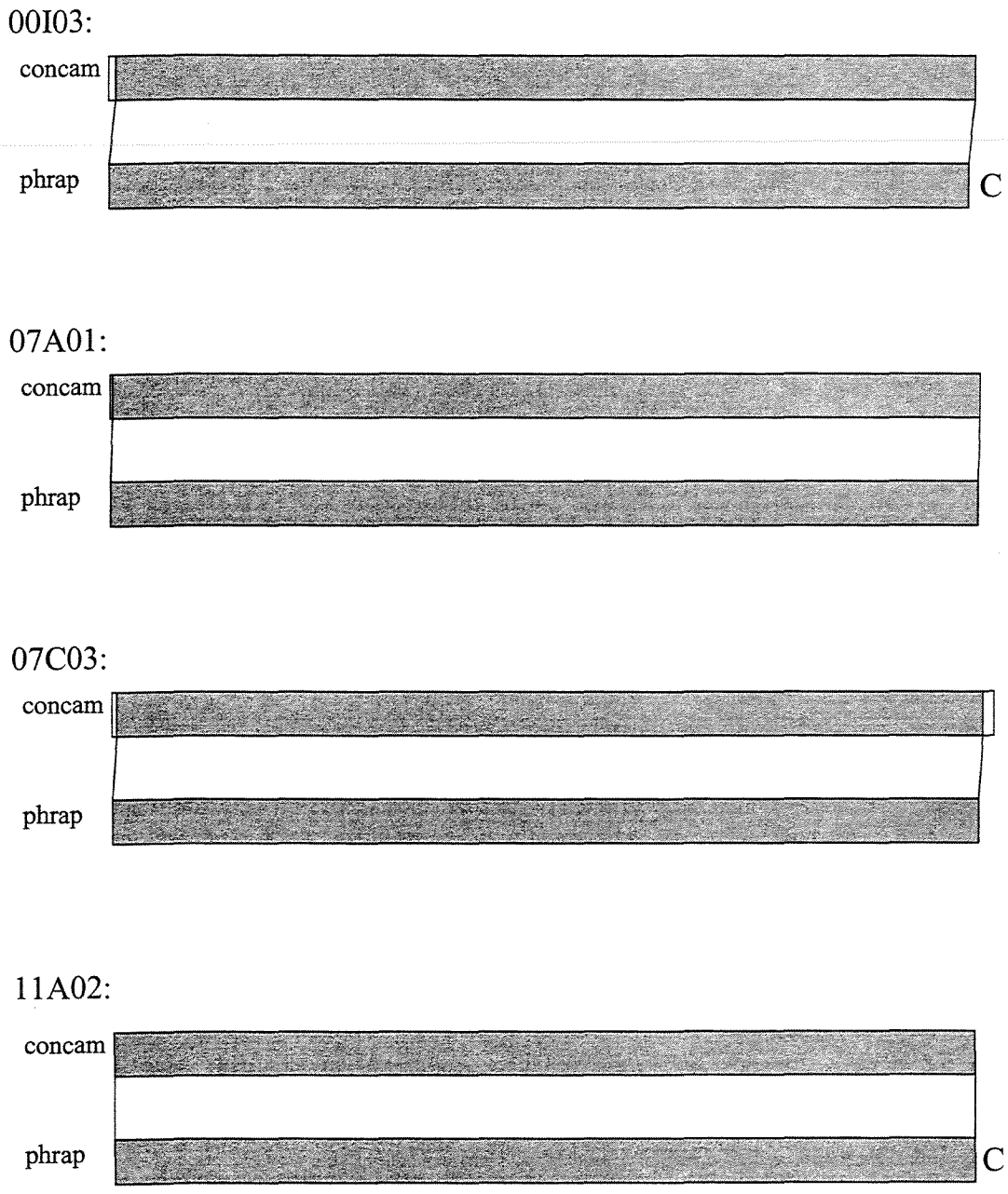


Figura 4.1: Comparações gráficas.

Tabela 4.4: Resultado da execução para o conjunto 07A01.

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1158	7690	556	602	556	1980	não	-	-	57	179.99
2	114	128	24	90	24	22	não	-	-	5	35.30
3	10	2	4	6	4	0	não	-	-	2	15.87
4	4	0	2	2	2	0	não	-	-	1	15.12

Número de Fragmentos eliminados no preprocessamento: 53

Tempo total: 246.28s

Comparação:
38836 0.50 0.09 0.10 concam 127 40415 (22) phrap 55 40339 (8)

Tabela 4.5: Resultado da execução para o conjunto 07C03.

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1848	19802	860	988	860	4118	sim	0	-	125	318.40
2	250	382	54	196	54	42	não	-	-	15	45.06
3	30	0	8	22	8	0	não	-	-	4	21.88
4	8	4	8	0	8	4	não	-	-	2	25.47
5	4	0	2	2	2	0	não	-	-	1	19.20

Número de Fragmentos eliminados no preprocessamento: 143

Tempo total: 430.01s

Comparação:
39441 0.56 0.08 0.04 concam 269 40591 (481) phrap 55 40392 (0)

Tabela 4.6: Resultado da execução para o conjunto 11A02.

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1706	18318	726	980	726	3672	sim	1	sim	83	310.97
2	166	244	32	134	32	20	não	-	-	8	39.62
3	16	0	10	6	10	0	não	-	-	4	18.39
4	10	0	6	4	6	0	não	-	-	3	17.14
5	6	2	4	2	4	2	não	-	-	1	21.65

Número de Fragmentos eliminados no preprocessamento: 184

Tempo total: 407.77s

Comparação:
38737 0.56 0.05 0.08 concam 1 40091 (7) C phrap (0) 40077 1

Capítulo 5

Conclusões

5.1 Pontos Principais

Um fato que consideramos importante nesta dissertação é o uso das qualidades das bases geradas pelo programa `phred`. Na fase de preprocessamento, as qualidades proporcionam a exclusão de muitos fragmentos não confiáveis, sendo que suas presenças no grafo poderiam acarretar distorções nas soluções.

A definição de arcos e vértices de Steiner também é bastante auxiliada pelas qualidades, pois em um determinado alinhamento, quando detectamos a existência de pontas indesejadas, uma vez concluído que essas pontas possuem qualidade baixa podemos simplesmente descartá-las de modo que não atrapalhem a análise. Como descrevemos anteriormente, se não utilizássemos as qualidades, a maneira que teríamos para não perder arcos importantes e ainda definir vértices como sendo de Steiner, seria aumentar a tolerância e diminuir a mínima cobertura, mas nesse caso, muitos alinhamentos que deveríam ser desconsiderados acarretariam arcos ou vértices de Steiner equivocadamente.

Para o cálculo dos consensos, mais uma vez as qualidades são muito importantes. O método por qualidade modificada descrito no capítulo 3 valoriza bases que têm qualidade alta, sendo mais confiável que, por exemplo, procurarmos a base que aparece em maior número em uma coluna.

Outro ponto importante de nossas abordagens foi a eliminação dos vértices de Steiner. Percebemos que essa exclusão, na prática, não prejudica as soluções e mais que isso pode viabilizar muitos casos cujo tamanho elevado do conjunto produza grafos muito grandes. Nos testes mostrados vimos que o tamanho do grafo se reduz

drasticamente com a retirada das subcadeias.

O método que apresentamos no capítulo 2 para encontrar caminhos em grafos de sobreposição, através de emparelhamento máximo, merece grande destaque, principalmente em grafos cíclicos aliado à nossa heurística de recombinação de ciclos e caminhos. Os testes revelam sinais de que pela característica dos grafos de sobreposição para problemas reais, este método é bastante eficiente uma vez que, naturalmente, parece fornecer soluções com um número bastante reduzido de ciclos. Nos exemplos cíclicos do capítulo 4 e ainda nos que se encontram no apêndice A, em muitos casos só ocorreram caminhos na solução e em outros no máximo dois ciclos foram reportados. Este baixo número de ciclos faz ainda com que a heurística implementada de recombinação não onere o tempo de execução. Nestes testes, em somente um caso não foi possível a recombinação, mas percebemos que neste exemplo iniciamos a execução dos algoritmos com tolerância igual a 10 e mínima cobertura igual a 90, mas os resultados gerais levam a crer que a primeira iteração deve ser sempre bastante restritiva. O que podemos destacar então destes fatos é que não houve a necessidade de, explicitamente, evitar ciclos na solução, pois a tendência é que esses ciclos não apareçam e quando o fazem é em número bastante reduzido, sendo rapidamente recombinados aos caminhos.

A maior contribuição deste trabalho foi a produção de algoritmos polinomiais que funcionam bem em casos práticos. As fases dominantes em termos de complexidade são a comparação dois a dois e a montagem dos grafos, que possuem complexidades quadráticas, mas nos testes obtivemos tempos bastante viáveis para casos de tamanhos elevados. Quanto à diferença de tempo de execução dos nossos algoritmos e o programa `phrap`, podemos destacar que todos os programas implementados foram escritos em `perl`, que se trata de uma linguagem interpretada, enquanto que o `phrap` é escrito em linguagem `C/C++`.

5.2 Extensões

Voltamos a enfatizar aqui a importância de se obter bons grafos de sobreposição. Vimos que o emparelhamento máximo foi bastante eficiente, mas se o grafo não condiz com a realidade no que se refere a ligar os fragmentos certos, não importa o método utilizado para encontrar caminhos, pois teremos soluções que não representarão bons consensos. Então é de grande valia estudar técnicas mais precisas para separar o que são fragmentos realmente bons para permanecerem no conjunto e ainda definir arcos que sejam bastante confiáveis.

No sentido de se obter bons grafos, podemos explorar a identificação de fragmentos quiméricos. Nesta dissertação não houve um tratamento explícito destes casos. Acreditamos que ao iniciarmos a execução da montagem com parâmetros bastante restritivos, como vimos no capítulo anterior, os efeitos desses fragmentos sejam minimizados, mas não necessariamente resolvidos. No trabalho de Huang [13], tanto a identificação de fragmentos quiméricos quanto estimativas de erros são feitas por um conceito denominado de *vetores de taxas de erro* [13]. Os fragmentos são divididos em regiões de tamanho r (a região mais à direita do fragmento pode ter tamanho menor que r), então a posição i do vetor contém a taxa de erro da região de tamanho r que começa na posição i do fragmento. Os valores do vetor são estimados de acordo com as sobreposições com outros fragmentos, fazendo uma comparação das regiões de similaridade. No caso dos fragmentos quiméricos, já que são formados de trechos não contínuos na molécula de DNA original, teremos uma ou mais regiões de tamanho r contendo pontos de separação que resultará em altas taxas de erro para aquela região.

Com a exclusão de fragmentos quiméricos na fase de preprocessamento, talvez possamos iniciar a montagem com parâmetros um pouco menos restritivos, portanto diminuindo o número de iterações e ainda obtendo bons resultados.

Uma outra questão que poderia ser abordada é com relação aos fragmentos cujos vértices foram considerados de Steiner e portanto não fizeram parte dos caminhos encontrados. Podemos fornecer ao programa `assemble` o alinhamento destes fragmentos com suas respectivas supercadeias de modo que façam parte dos alinhamentos múltiplos produzidos. A vantagem disto é que esses fragmentos contribuirão com suas qualidades no processo de votação, pois mesmo tendo conseguido boas soluções sem este feito, eventualmente, num caso em que haja equilíbrio nas qualidades modificadas de uma base que deveria ser escolhida para representar determinada coluna e outra que deveria ser descartada, a inclusão das subcadeias possa ajudar no desempate em prol da base mais adequada.

Quando se fala em ferramentas para montagem de fragmentos, a visualização gráfica das soluções é muito importante. A produção de arquivos `ace` foi um importante segmento de nosso desenvolvimento. No entanto, optando por uma estratégia de várias iterações, geramos o arquivo `ace` relativo a cada iteração, mas não um que envolva o consenso final produzido e os fragmentos originais do início da montagem. Na primeira iteração por exemplo, todos os fragmentos que sobraram do preprocessamento e cujas sequências não são subcadeias de nenhuma outra, poderão ser visualizados no programa `consed` juntamente com os consensos. Na segunda iteração os fragmentos a serem montados serão os consensos da primeira iteração, então estes consensos também poderão ser visualizados e assim por diante, mas não há um arquivo `ace` da

solução final que inclua os fragmentos iniciais.

Produzir ferramentas cada vez mais adequadas ao uso dos profissinais de biologia e bioinformática é um grande desafio. Muitas são as possibilidades para o desenvolvimento de novas idéias acerca deste tema. Acreditamos que a continuação dos resultados alcançados no trabalho apresentado é de extrema importância visto que os paradigmas adotados deram bons resultados e sinais de prosperidade.

Apêndice A

Resultados dos Testes

Neste apêndice podem ser encontrados vários outros testes com maiores variações dos parâmetros *tol*, *min*, *inc* e *dec*. Para identificação do significado de cada coluna vide seção 4.3 do capítulo 4.

Tabela A.1: 07A01 ($tol = 1$, $min = 99$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1158	7690	556	602	556	1980	não	-	-	57	176.16
2	114	120	28	86	28	22	não	-	-	6	34.66
3	12	2	6	6	6	2	não	-	-	2	20.91
4	4	0	4	0	4	0	não	-	-	2	16.56
5	4	0	2	2	2	0	não	-	-	1	17.77

Tempo total: 266.06s

Comparação:
38802 0.51 0.10 0.11 concam 127 40414 (22) phrap 55 40339 (8)

Tabela A.2: 07A01 ($tol = 1$, $min = 99$, $inc = 5$, $dec = 5$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1158	7690	556	602	556	1980	não	-	-	57	178.02
2	114	136	22	92	22	24	não	-	-	2	36.75
3	4	0	2	2	2	0	não	-	-	1	15.76

Tempo total: 230.53s

Comparação:
38745 0.54 0.11 0.10 concam 127 40407 (22) phrap 55 40339 (8)

Tabela A.3: 07A01 ($tol = 2$, $min = 98$, $inc = 3$, $dec = 3$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1158	8162	534	624	534	2010	não	-	-	34	174.57
2	68	64	16	52	16	14	não	-	-	2	29.51
3	4	0	4	0	4	0	não	-	-	2	16.26
4	4	0	2	2	2	0	não	-	-	1	15.55

Tempo total: 235.89s

Comparação:
38288 0.88 0.08 0.18 concam 23 40345 (126) C phrap (8) 40339 55

Tabela A.4: 07A01 ($tol = 5$, $min = 95$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1158	8464	512	646	512	1948	não	-	-	30	169.15
2	60	52	14	46	14	10	não	-	-	2	29.51
3	4	0	2	2	2	0	não	-	-	1	15.05

Tempo total: 213.71s

Comparação:
38564 0.76 0.09 0.11 concam 23 40369 (79) C phrap (8) 40339 1

Tabela A.5: 07A01 ($tol = 10$, $min = 90$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1158	8560	476	682	476	1750	não	-	-	32	167.39
2	64	56	14	50	14	14	não	-	-	1	24.68

Tempo total: 192.07s

Comparação:
38617 0.73 0.05 0.11 concam 174 40441 (22) phrap 95 40339 (8)

Tabela A.6: 07C03 ($tol = 1$, $min = 99$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1848	19802	860	988	860	4118	sim	0	-	125	336.76
2	250	320	72	178	72	38	não	-	-	24	47.02
3	48	10	20	28	20	6	não	-	-	8	22.95
4	16	0	8	8	8	0	não	-	-	4	19.73
5	8	0	6	2	6	0	não	-	-	3	18.18
6	6	2	6	0	6	2	não	-	-	2	21.12
7	4	0	2	2	2	0	não	-	-	1	16.49

Tempo total: 482.25s

Comparação:
39707 0.35 0.08 0.03 concam 269 40583 (481) phrap 55 40392 (0)

Tabela A.7: 07C03 ($tol = 1$, $min = 99$, $inc = 5$, $dec = 5$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1848	19802	860	988	860	4118	sim	0	-	125	326.80
2	250	452	38	212	38	50	não	-	-	7	43.57
3	14	4	8	6	8	4	não	-	-	2	20.32
4	4	0	2	2	2	0	não	-	-	1	16.81

Tempo total: 407.50s

Comparação:
39750 0.31 0.09 0.03 concam 269 40582 (481) phrap 55 40392 (0)

Tabela A.8: 07C03 ($tol = 2$, $min = 98$, $inc = 3$, $dec = 3$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T**(s)**
1	1848	21104	778	1070	778	3952	sim	0	-	69	315.02
2	138	200	26	112	26	26	não	-	-	4	42.93
3	8	0	6	2	6	0	não	-	-	3	23.46
4	6	2	6	0	6	2	não	-	-	2	22.81
5	4	2	2	2	2	0	não	-	-	1	14.36

Tempo total: 418.58s

Comparação:
2082 1.77 0.09 0.35 concam 344 2607 (32988) C phrap (30) 40362 38105
7918 1.81 0.34 0.27 concam 2432 11041 (24554) C phrap (2727) 37665 29050
9669 1.15 0.16 0.18 concam 10385 20555 (15040) C phrap (10404) 29988 19820
14493 1.77 0.20 0.17 concam 19798 35381 (214) C phrap (24804) 15588 1

Tabela A.9: 07C03 ($tol = 5$, $min = 95$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T**(s)**
1	1848	22588	698	1150	698	3676	sim	0	-	21	359.61
2	42	36	10	32	10	6	não	-	-	2	37.21
3	4	0	4	0	4	0	não	-	-	2	26.19
4	4	0	2	2	2	0	não	-	-	1	19.47

Tempo total: 442.48s

Comparação:
39210 0.63 0.15 0.08 concam 215 40565 (492) phrap 1 40381 (11)

Tabela A.10: 07C03 ($tol = 10$, $min = 90$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1848	23282	660	1188	660	3406	sim	1	sim	59	310.57
2	118	178	14	104	14	12	sim	2	não	3	29.91
3	6	0	6	0	6	0	não	-	-	3	16.01
4	6	0	4	2	4	0	não	-	-	2	15.55
5	4	0	2	2	2	0	não	-	-	1	13.73

Tempo total: 385.77s

Comparação:
2135 1.83 0.00 0.13 concam 486 2776 (32667) C phrap (0) 40392 38105
17247 0.74 0.17 0.08 concam 2600 20429 (15014) C phrap (2726) 37666 19820
14944 0.86 0.21 0.02 concam 19672 35175 (268) C phrap (24804) 15588 55

Tabela A.11: 11A02 ($tol = 1$, $min = 99$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1706	18318	726	980	726	3672	sim	1	sim	83	300.65
2	166	234	36	130	36	20	não	-	-	10	39.76
3	20	0	12	8	12	0	não	-	-	6	18.15
4	12	0	10	2	10	0	não	-	-	5	17.48
5	10	0	8	2	8	0	não	-	-	4	17.21
6	8	0	6	2	6	0	não	-	-	3	16.64
7	6	0	2	4	2	0	não	-	-	1	16.37

Tempo total: 426.26s

Comparação:
38138 0.61 0.05 0.10 concam 1 39579 (524) C phrap (0) 40077 518

Tabela A.12: 11A02 ($tol = 1$, $min = 99$, $inc = 5$, $dec = 5$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1706	18318	726	980	726	3672	sim	1	sim	83	321.45
2	166	248	28	138	28	20	não	-	-	6	38.13
3	12	0	6	6	6	0	não	-	-	3	17.19
4	6	2	4	2	4	2	não	-	-	1	21.51

Tempo total: 398.28s

Comparação:
38737 0.56 0.05 0.08 concam 1 40091 (7) C phrap (0) 40077 1

Tabela A.13: 11A02 ($tol = 2$, $min = 98$, $inc = 3$, $dec = 3$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1706	18842	696	1010	696	3516	sim	0	-	52	300.98
2	104	114	16	88	16	10	não	-	-	4	36.55
3	8	0	6	2	6	0	não	-	-	3	19.18
4	6	0	2	4	2	0	não	-	-	1	15.76

Tempo total: 372.47s

Comparação:
4129 2.94 0.04 0.07 concam 1 4589 (35133) C phrap (0) 40077 35490
34269 0.75 0.04 0.08 concam 3498 39198 (524) C phrap (3872) 36205 518

Tabela A.14: 11A02 ($tol = 5$, $min = 95$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1706	19434	650	1056	650	3198	sim	0	-	50	294.15
2	100	108	16	84	16	14	não	-	-	4	32.32
3	8	2	6	2	6	2	não	-	-	2	20.04
4	4	0	4	0	4	0	não	-	-	2	16.75
5	4	2	4	0	4	2	não	-	-	1	23.36

Tempo total: 386.62s

Comparação:
32503 2.05 0.23 0.26 concam 8 35835 (3514) phrap 1 35815 (4262)
3669 3.67 0.12 0.17 concam 35132 39349 (0) phrap 35862 40077 (0)

Tabela A.15: 11A02 ($tol = 10$, $min = 90$, $inc = 2$, $dec = 2$).

It.	v	a	t	s	V	A	**Cíclico**	ci	**Rec.**	C	T(s)
1	1706	19762	618	1088	618	3014	sim	0	-	50	285.95
2	100	114	12	88	12	10	não	-	-	2	28.45
3	4	2	4	0	4	2	não	-	-	1	17.57

Tempo total: 331.97s

Comparação:
35069 0.56 0.02 0.05 concam 8 36220 (3497) phrap 1 36205 (3872)
4068 3.25 0.13 0.09 concam 35132 39717 (0) phrap 35490 40077 (0)

Bibliografia

[1] J. A. Bondy and U. S. R. Murty. *Graph Theory With Applications.* The Macmillan Press LTD, 1976.

[2] James K. Bonfield, Kathryn F. Smith, and Rodger Staden. A new DNA sequence assembly program. *Nucleic Acids Research*, pages 23:4992–4999, 1995.

[3] B.V. Cherkassky, A.V. Goldberg, P. Martin, J. C. Setubal, and J. Stolfi. Augment or push? a computational study of bipartite matching and unit capacity flow algorithms. *Proceedings of the Workshop on Algorithm Engineering*, pages 1–10, 1997.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1990.

[5] Augusto A. M. de Almeida, João Meidanis, and Alexandre Moriya. Um sistema para auxílio na montagem de fragmentos de DNA. *XXI Seminário Integrado de Software e Hardware*, pages 533–545, 1994.

[6] Richard Durbin and Simon Dear. Base qualities help sequencing software. *Genome Research*, pages 8:161–162, 1998.

[7] B. Ewing and P. Green. Base-calling of automated sequencer traces using phred. I. accuracy assessment. *Genome Research*, pages 8:175–185, 1998.

[8] B. Ewing and P. Green. Base-calling of automated sequencer traces using phred. II. error probabilities. *Genome Research*, pages 8:186–194, 1998.

[9] Carlos E. Ferreira, Cid C. de Souza, and Yoshiko Wakabayashi. Rearrangement of DNA fragments: a branch-and-cut algorithm. Technical report, IME-USP e IC-UNICAMP, 1997.

[10] National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov/BLAST/fasta.html.

[11] David Gordon, Chris Abajian, and Phil Green. Consed: A graphical tool for sequence finishing. *Genome Research*, pages 8:195–202, 1998.

[12] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

[13] X. Huang. An improved sequence assembly program. *Genomics*, pages 33:21–31, 1996.

[14] The National Human Genome Research Institute. http://www.nhgri.nih.gov/HGP/.

[15] J. D. Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction.* PhD thesis, University of Arizona, 1991.

[16] J. D. Kececioglu and E. W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, pages 13:7–51, 1995.

[17] Udi Manber. *Introduction to Algorithms.* Addison-Wesley, 1989.

[18] João Meidanis. *Algorithms for Problems in Computational Genetics.* PhD thesis, University of Wisconsin-Madison, 1992.

[19] E. W. Myers. Advances in sequence assembly. In M. A. Adams, C. Fields, and J. C. Venter, editors, *Automated DNA Sequencing and Analysis*, chapter XXXII, pages 231–238. Academic Press, 1994.

[20] The Phred/Phrap/Consed System Home Page. http://www.phrap.org/.

[21] H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. *In Information Processing 83: Proceedings of the International Federation for Information Processing (IFIP) Ninth World Computer Congress*, pages 53–64, 1983.

[22] *Xylella fastidiosa* Genome Project. http://www.dcc.unicamp.br/genoma/.

[23] Peter Richterich. Estimation of errors in "Raw" DNA sequences: A validation study. *Genome Research*, pages 8:251–259, 1998.

[24] João Setubal and João Meidanis. *Uma Introdução à Biologia Computacional.* CIP, 1994.

[25] João Setubal and João Meidanis. *Introduction to Computational Molecular Biology.* PWS, 1997.

[26] J. S. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, pages 83:1–20, 1989.

[27] Michael S. Waterman. *Introduction to Computational Biology.* Chapman and Hall, 1995.