Montagem de Fragmentos de DNA

Aluno: Thiago Teixeira Santos
Professor: Carlos Eduardo Ferreira

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

19 de Julho de 1999

Projeto PIBIC/CNPq

Conteúdo

Lista de Figuras

1 Introdução

O *seqüenciamento de moléculas de DNA* se tornou prática comum na comunidade científica. Tal processo determina a seqüência de *nucleotídeos* (as unidades químicas que constituem o DNA) formadora da molécula. Hoje, existem inúmeros projetos que visam mapear o genoma de organismos inteiros, inclusive o do próprio ser humano. Estes trabalhos são conhecidos como *seqüenciamento em larga escala*.

Porém, há um grande problema pelo qual estas técnicas de seqüenciamento passam: elas só são capazes de mapear moléculas com comprimento entre 500 e 700 nucleotídeos. Uma vez que uma molécula de DNA pode ter centenas de milhares de nucleotídeos, a solução é fragmentá-la em cadeias menores, que possam ser seqüenciadas. Ao final, estes fragmentos mapeados são usados como peças para montar a molécula original. Este é o trabalho dos *programas para montagem de fragmentos*.

Neste trabalho, abordaremos superficialmente os processos laboratoriais de seqüenciamento. Munidos desta base mínima para a compreensão da realidade biológica, iniciaremos o estudo computacional envolvido, formalizando o problema e buscando uma heurística para seu tratamento. Por fim, apresentaremos outras heurísticas conhecidas dentro da Ciência da Computação.

2 Do Laboratório...
O Seqüenciamento de DNA

2.1 O método *dideoxi*

A biotecnologia possui várias ferramentas para seqüenciar DNA. Um dos métodos mais usado no seqüenciamento em larga escala é o *dideoxi*, o qual iremos descrever em linhas gerais:

A enzima *DNA polimerase* é a responável pela replicação de DNA. Ela utiliza a molécula a ser replicada como molde e vai incorporando nucleotídeos à cópia. Cada nucleotídeo é capaz de duas ligações: uma delas incorpora-o à nova molécula enquanto a outra agrega novos nucleotídeos, prosseguindo com o processo de replicação.

A idéia do método *dideoxi* é inserir no meio nucleotídeos sintetizados especialmente para interromper a replicação. Seja A^* um nucleotídeo *Adenina* capaz de se incorporar à molécula mas incapaz de agregar novos nucleotídeos. Ao iniciarmos uma processo de replicação em um ambiente com nucleotídeos A^*, iremos gerar subcadeias de DNA cuja terminação se dá em A (Figura 1). Com uma *eletroforese em gel*, podemos separar essas cadeias por tamanho, concluindo a posição de cada nucleotídeo A na molécula original.

Podemos sintetizar nucleotídios G^*, T^* e C^* com as mesmas propriedades de A^* e, com isso, seqüenciar todo o fragmento de DNA. A Figura 2 mostra um filme produzido pela eletroforese em gel, que permite definir a ordem dos nucleotídeos na molécula.

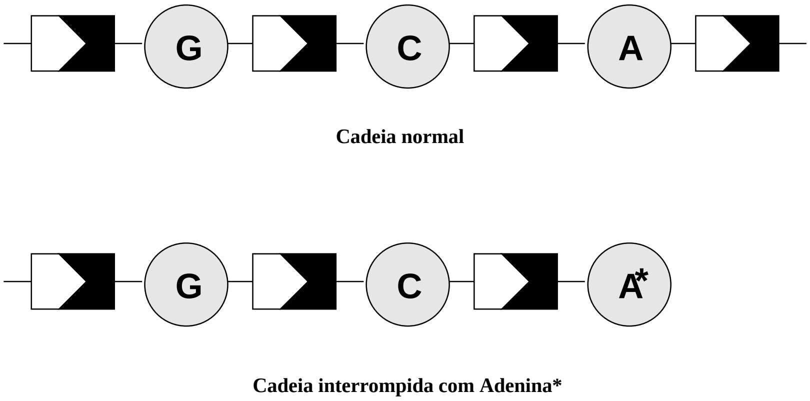

Figura 1: Seqüenciamento *dideoxi*.

Porém, há um sério problema neste método: quanto maior o comprimento da molécula sendo seqüenciada, menor é a resolução do filme gerado pela eletroforese. Isso faz com que o método *dideoxi* seja útil para cadeias de 500 a 700 nucleotídeos, um grande problema se tratando de moléculas de DNA, que podem chegar a milhares de pares de bases.

2.2 Contornando o problema: fragmentação de DNA

A solução para o problema acima é fragmentar o DNA, obtendo trechos de comprimentos suficientemente pequenos de modo que possam ser seqüenciados. Porém, como desejamos obter posteriormente a seqüência original, este processo de fragmentação deve atender a certos requisitos.

Como observarmos na Figura 3, os fragmentos obtidos devem cobrir toda a extensão da molécula original. Além disso, é necessária uma sobreposição suficientemente grande entre os fragmentos para que eles possam, futuramente, durante o processo de montagem, serem arranjados de modo a obtermos o DNA inicial.

2.2.1 Enzimas de Restrição

Enzimas de restrição são enzimas capazes de "cortar" DNA em pontos específicos. A enzima *Eco RI*, por exemplo, corta as moléculas quando encontra a seqüência GAATTC. Há centenas de enzimas conhecidas que reconhecem diversos padrões. Todos os padrões são *palíndromos*, ou seja, são seqüências iguais ao seu complemento reverso. No caso da *Eco RI*:

$$\mathsf{GAATTC}^{CR} = \mathsf{GAATTC}$$

Mas, justamente pelo fato de fazer os cortes em sítios específicos, as enzimas de restrição não são a melhor forma de obter uma fragmentação satisfatória,

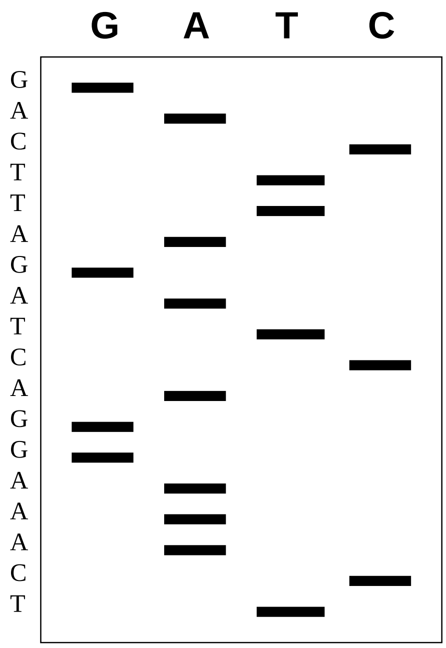

Figura 2: Filme produzido pela eletroforese em gel.

nos termos colocados acimas. Tais enzimas podem gerar fragmentos demasiadamente longos, o que não ajuda em nada quando procuramos por fragmentos com 500-700 bases. Por outro lado, elas podem gerar fragmentos muito pequenos, que também não são úteis. Por estas razões, em seqüenciamento em larga escala, prefere-se o método *shotgun* (canhão).

2.2.2 O *shotgun*

Neste processo, o DNA é submetido a algum processo físico (exposição à ondas ultrassônicas, centrifugação). Isto gera uma fragmentação que atende melhor aos requisitos do processo de montagem (Figura 3). Os fragmentos resultantes são selecionados por tamanho, replicados e seqüenciados.

O maior problema do *shotgun* é que ele gera fragmentos potencialmente únicos. Uma vez que o processo de seqüenciamento necessita de várias cópias da mesma molécula, são realizadas *clonagens* dos fragmentos. Isto é feito inserindo o fragmento de DNA em um microorganismo *vetor* (vírus ou bactérias). Durante a sua reprodução, o vetor cria novas cópias da seqüência, constituindo um método eficiente de ampliação. Porém, mutações e contaminações pelo DNA hospedeiro podem gerar cópias não fiéis ao original.

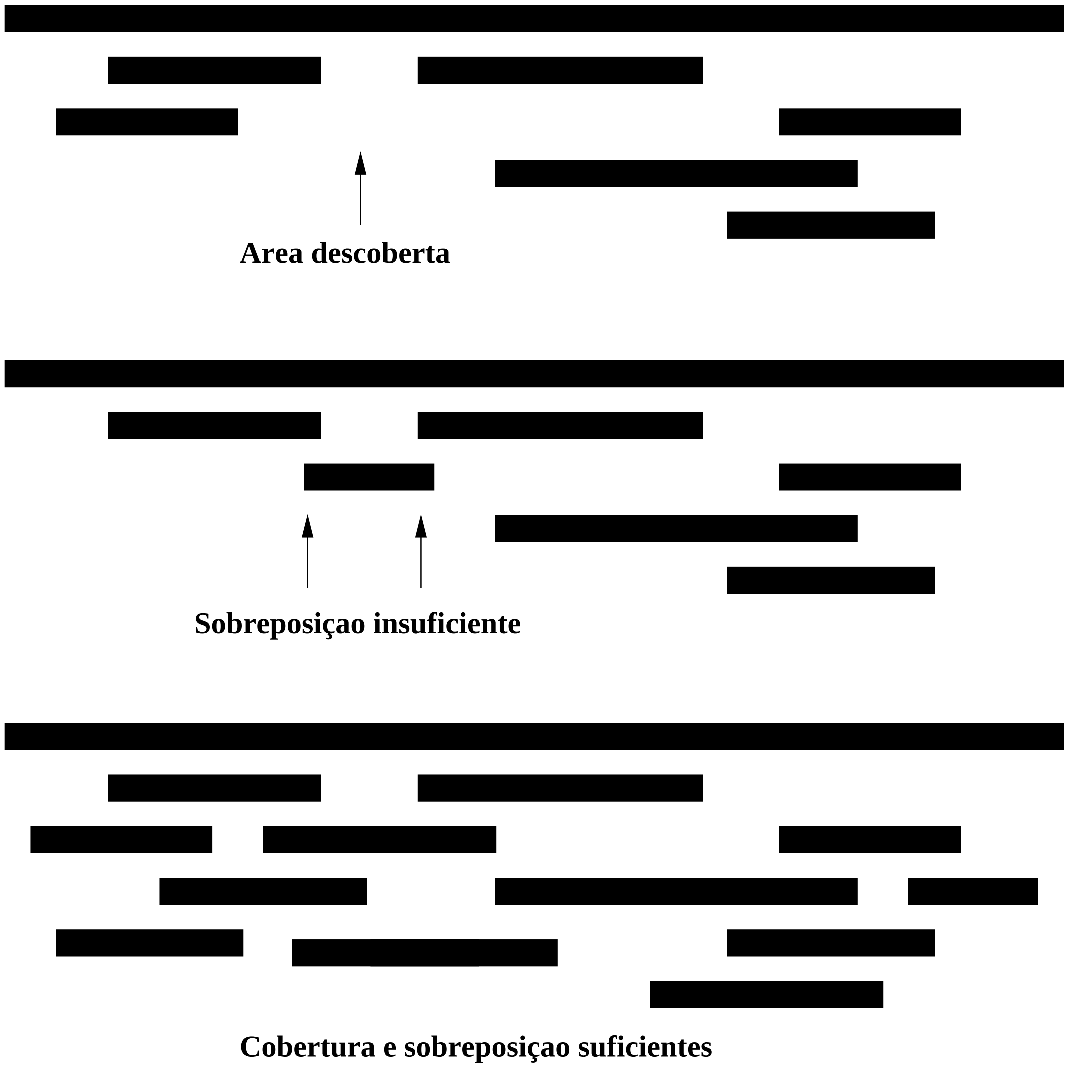

Figura 3: Fragmentação de DNA.

3 ... ao Computador

O Problema da Montagem de Fragmentos

3.1 Formalizações

Através dos fragmentos de DNA seqüenciados em laboratório, queremos obter o DNA original. É natural associar esse problema ao:

Problema 1 (Mínima Supercadeia Comum - MSC) *Dado um conjunto de strings s_1, s_2,..., s_k sobre um alfabeto Σ, encontrar a seqüência mais curta possível S tal que cada s_i seja um fator de S, ou seja, $S = us_iv$ para certos u e v sobre Σ.*

Este problema é NP-difícil. Embora pareça bastante adequado no contexto da montagem de fragmentos, há algumas ressalvas que devem ser feitas:

Erros na entrada: Como vimos, o processo de seqüenciamento pode gerar fragmentos contaminados pelo DNA do vetor e/ou alterados por mutações (embora este último seja um evento raro). O **MSC** não leva em conta tais fatos.

Orientação: O seqüenciamento é feito em DNA de fita simples. É muito comum que os fragmentos gerados sejam oriundos de fitas diferentes. Porém, cada molécula possui extremidades bem definidadas (nomeadas 3' e 5'). Cada uma das fitas do DNA-duplo se encontra em um a orientação diferente (3'-5' e 5'-3'). Logo, a molécula gerada na montagem deve cobrir cada fragmento ou seu complemento reverso.

Cobertura insuficiente: Embora o *shotgun* procure minimizar este problema, é possível que os fragmentos não cubram a extensão total da molécula original (Figura 3). Devemos, nestes casos, tentar encontrar uma ou mais seqüências que cubram todos os fragmentos.

Comprimento da molécula original: O esforço do **MSC** em minimizar o comprimento da resposta é para evitar respostas triviais (como a simples concatenação dos fragmentos). No problema de montagem, os biólogos podem, com grande precisão, determinar o comprimento da molécula alvo.

Assim, podemos redefinir o problema, tornando-o mais próximo da realidade biológica:

Problema 2 (Montagem de Fragmentos) *Dado um conjunto de strings s_1, s_2,..., s_k sobre um alfabeto $\Sigma = \{A, C, T, G\}$, encontrar a seqüência S tal que, para cada s_i:*

$$Sim(s_i, S) > \alpha$$
$$ou$$
$$Sim(s_i^{CR}, S) > \alpha$$

Onde $Sim(f,F)$ é uma função que retorna um valor associado a similaridade entre as duas seqüências e α é a similaridade mínima requerida.

Um versão mais completa faria uma restrição sobre S, para que seu comprimento fosse próximo do tamanho estimado pelos biólogos. Por hora, trataremos desta versão mais simplificada do problema.

3.2 Definindo uma Heurística

Quando especificamos o que seria uma boa fragmentação, dissemos que, além de uma boa cobertura, era necessária uma sobreposição suficiente entre os fragmentos. Tais sobreposições constituem as "ligações" entre um fragmento e outro, de modo que assim podemos agregá-los em seqüências maiores, obtendo por fim a molécula original.

3.2.1 Comparando Fragmentos

As sobreposições podem ser avaliadas com o uso de um algoritmo por nós tratado em [SF 98]. Trata-se de uma adaptação do:

Problema 3 (Maior Subseqüência Comum) *Dadas duas seqüências $X = \langle x_1, x_2, ..., x_m\rangle$ e $Y = \langle y_1, y_2, ..., y_n\rangle$, encontrar a maior subseqüência comum entre X e Y, ou seja, a maior seqüência estritamente crescente $\langle i_1, i_2, ..., i_k\rangle$ tal que, para todo $j = 1, 2, ..., k$, $x_{i_j} = y_{i_j}$.*

Um algoritmo de programação dinâmica para a resolução do problema encontra-se na Figura 4. Ele devolve dois vetores b e c. O vetor b atua como um *log* das operações, permitindo reconstruir a máxima subseqüência comum. Já $c[i,j]$ nos dá o comprimento da maior subseqüência entre $\langle x_1, ..., x_i\rangle$ e $\langle y_1, ..., y_j\rangle$.

Uma modificação de $\mathsf{MaxSC(X, Y)}$ nos dá o algoritmo para a função $Sim(X, Y)$ (vista anteriormente na definição do Problema 2). Tal algoritmo (Figura 5) leva em consideração pequenos erros nas strings de entrada. A função $p(x_i, y_j)$ compara os dois caracteres x_i e y_j, retornando 1 caso sejam iguais ou -1 caso contrário. Assim, torna-se possível alinhar nucleotídeos diferentes, penalizando a similaridade entre os fragmentos de DNA toda vez que isso ocorre. Além disso, é possível inserir "buracos" nos fragmentos, de modo a maximizar a similaridade. Porém, quando um buraco é inserido, uma penalização g é removida da similaridade.

Este algoritmo, bem como o problema a ele associado, possui várias variantes. Maiores detalhes podem ser encontrados em [MS 94] e [SF 98].

3.2.2 As Fases da Montagem

Podemos considerar três passos para a montagem:

Comparação Rápida: Imagine um conjunto de 500 strings sobre $\Sigma = \{A, T, C, G\}$ geradas como saída do *shotgun*. Espera-se que cada um destes fragmentos se sobreponha a um certo número de outras seqüências. Este número, porém, deve ser bem menor do que 500.

```
MaxSC(X, Y)

m ← lenght[X]
n ← lenght[Y]
para i ← 1 ate m
     faça c[i, 0] ← 0
para j ← 0 ate n
     faça c[0, j] ← 0
para i ← 1 ate m
     faça para j ← 1 ate n
               faça se x_i = y_j
                      entao c[i, j] ← c[i − 1, j − 1] + 1
                            b[i, j] ← "↖"
                      senao se c[i − 1, j] ≥ c[i, j − 1]
                               entao c[i, j] ← c[i − 1, j]
                                     b[i, j] ← "↑"
                               senao c[i, j] ← c[i, j − 1]
                                     b[i, j] ← "←"
retorne c e b
```

Figura 4: Algoritmo para encontar a Maior Subseqüência Comum.

```
Sim(X, Y)

m ← lenght[X]
n ← lenght[Y]
para i ← 1 ate m
     faça c[i, 0] ← i ∗ g
para j ← 0 ate n
     faça c[0, j] ← j ∗ g
para i ← 1 ate m
     faça para j ← 1 ate n
               faça se x_i = y_j
                      entao c[i, j] ← c[i − 1, j − 1] + p(x_i, y_j)
                      senao se c[i − 1, j] ≥ c[i, j − 1]
                               entao c[i, j] ← c[i − 1, j] − g
                               senao c[i, j] ← c[i, j − 1] − g
retorne c[m, n]/min(m, n)
```

Figura 5: Algoritmo para encontrar a similaridade entre duas seqüências.

A comparação por Sim(X, Y) é bastante cara em termos de tempo de processamento, pois o algoritmo é $O(mn)$. Se os 500 fragmentos tiverem comprimento em torno de 600 bases cada um, o custo da comparação dois a dois seria $C_2^{500} * (600 * 600) \sim 4.5 * 10^{10}$!

Torna-se necessário um meio mais barato de comparação, tendo como objetivo indicar para os algoritmos da próxima fase quais são os fragmentos que valem a pena serem analisados por Sim(X, Y), isto é, os fragmentos que têm uma semelhança mínima entre si de modo a valer a pena compará-los por programação dinâmica.

O objetivo desta fase será, portanto, gerar um grafo $G = (V, E)$ onde cada vértive $v \in V$ representa um fragmento de entrada e só existe uma aresta $(u, v) \in E$ se e somente há uma semelhança mínima entre os fragmentos representados por u e v.

Refinamento: O objetivo aqui é refinar os dados obtidos na fase anterior. Seja $s[v]$ o fragmento associado ao vértice $v \in V$. Para cada aresta $(u, v) \in E$, chamamos Sim$(s[u], s[v])$. Se a similaridade entre os fragmentos estiver acima de um mínimo pré-estabelecido, a aresta (u, v) é mantida. Caso contrário, removemos (u, v) de E.

Após o refinamento, um algoritmo para cálculo de árvores geradoras máximas é utilizado no grafo G, gerando uma árvore $A = (V_A, E_A)$. A idéia é manter um conjunto minimal de arestas que resultam na maior similaridade total.

Determinação do Consenso (Montagem): Resta agora obtermos o DNA original. Para isso, seguiremos a árvore gerada na fase anterior: para cada $u, v \in E_A$ tal que $v = filho[u]$, geramos um vértice w tal que $s[w]$ seja a seqüência consenso entre $s[u]$ e $s[v]$. Para cada aresta oriunda do vértice u, isto é, da forma $(u, z) \in E_A$, criamos uma aresta correspondente (w, z). Redefinimos $A = (V_A^*, E_A^*)$ onde:

$V_A^* = (V_A - \{u, v\}) \cup \{w\}$
e
$E_A^* = (E_A - E_u) \cup E_w$
sendo: $E_u = \{(x, y) \in E_A | x = u\}$ e $E_w = \{(x, y) \in E_A | x = w\}$

Repetimos este processo até restar um único vértice r na árvore A. A seqüência $s[r]$ será nosso consenso geral, ou seja, a molécula original.

Nas próximas seções detalharemos melhor cada uma destas fases.

3.3 Comparação Rápida

Nosso método de comparação rápida é inspirado no utilizado em [MS 94]. O primeiro passo é codificar os fragmentos de DNA na forma de dígitos. Definimos A= 0, C= 1, G= 2 e T= 3 (Figura 6 a).

Definimos, também, um tamanho para as *janelas*, isto é, subseqüências contínuas dos fragmentos. Vamos assumir, por exemplo, que nossas janelas tenham tamanho 8. A idéia é avaliar quantas janelas dois fragmentos, f_i e f_j, têm em comum. Se o número de janelas comum for suficientemente grande (acima de um parâmetro β), podemos dizer que os fragmentos são suficientemente parecidos para serem avaliados por programação dinâmica na fase posterior, ou seja, estes fragmentos têm grande probabilidade de serem similares ($Sim(f_i, f_j) > \alpha$).

Analisamos todas as janelas de cada fragmento f_i. Suponha, por exemplo, que encontremos a janela ATCATGCT, que codificada torna-se 03103213. Associamos uma tabela de hashing t_i a f_i, na forma de um vetor de bits de tamanho h. Para obtermos um bom espalhamento, h deve ser um número primo e razoavelmente grande, de modo que não haja muitas colisões. Para prosseguir com nosso exemplo, tomemos $h = 2003$. Então, para indicar a presença da janela ATCATGCT no fragmento f_i:

$t_i[pos] = 1$
onde
$pos = (03103213)_4 \bmod 2003 = 13543 \bmod 2003 = 1525$

Deste modo, obtemos tabelas de hashing para cada fragmento, como indicado na Figura 6 b. Para saber se dois fragmentos têm um número suficientemente grande de janelas comuns, aplicamos uma operação AND entre seus vetores de hashing e contamos o número de bits 1, verificando se este total é maior ou igual ao parâmetro β.

Podemos agora criar um grafo G tal que a cada vértice é associado um fragmento. Há uma aresta conectando dois vértices se seus fragmentos associados possuem um número satisfatório de janels comuns.

Note que nosso sistema de hashing não trata colisões, ou seja, duas janelas diferentes w_1 e w_2, presentes em f_i, poderiam alterar um mesmo bit em t_i. Porém, como estamos interessados em um método "grosseiro" de comparação, cuja função é apenas eliminar o uso da comparação por programação dinâmica entre seqüências muito diferentes, tal fato não se torna um problema.

3.4 Refinamento

A fase anterior nos deu um grafo G com uma série de arestas que não deveriam existir, pois não possuem uma similaridade mínima requerida. Estes *falso-positivos* devem ser removidos com uma comparação mais rigorosa dada por Sim(X, Y). O algoritmo da Figura 7 nos devolve um novo grafo G que é o refinamento do grafo de entrada. Note que $w(u, v)$ denota o peso associado à aresta (u, v).

Queremos trabalhar com um conjunto minimal de arestas, ou seja, o menor conjunto de arestas que conectam todos os vértices do grafo G. Isso torna mais fácil a montagem na próxima fase. Para obtermos uma qualidade maior,

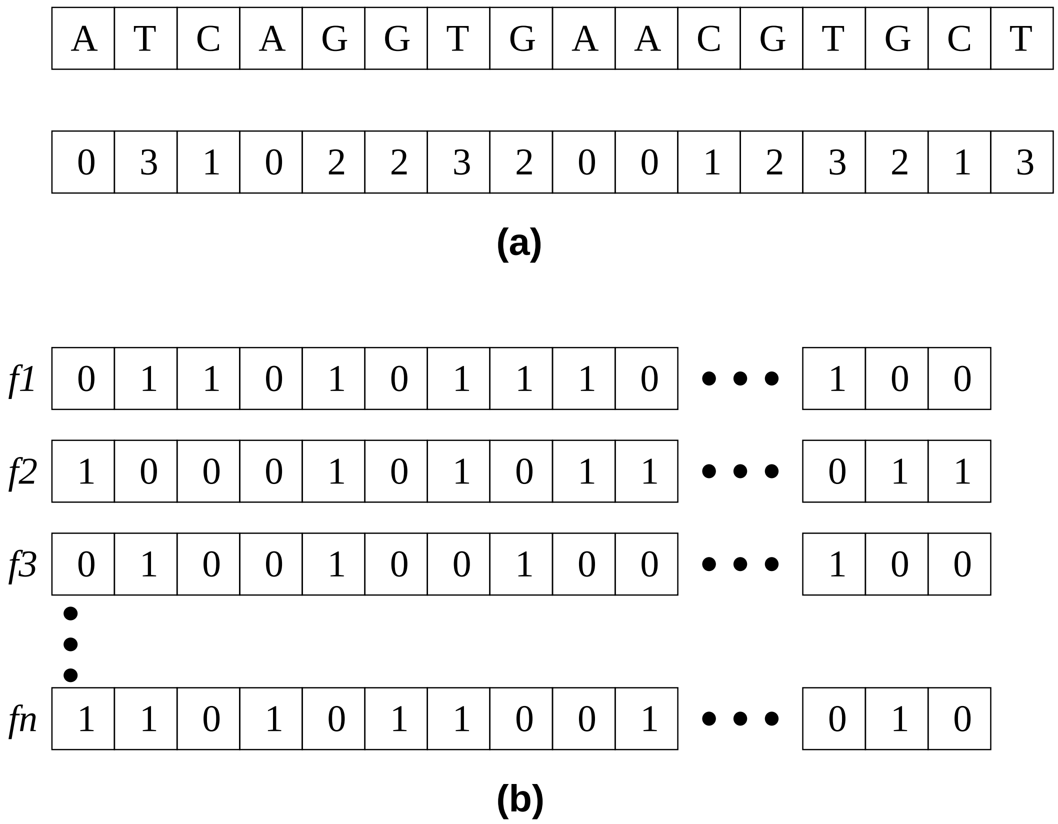

Figura 6: **(a)** Codificação em base 4. **(b)** Tabelas de Hashing.

Refina(G, w)

para cada $(u, v) \in E[G]$
    faça $w(u, v) \leftarrow \mathsf{Sim}(s[u], s[v])$
        se $w(u, v) < \mu$
            então $E[G] \leftarrow E[G] - \{(u, v)\}$
retorne G

Figura 7: Algoritmo para Refinamento de Grafos

escolheremos um conjunto minimal E_m de arestas tal que maximize:

$$\sum_{(u,v) \in E_m} Sim(s[u], s[v])$$

.

O grafo $A = (V, E_m)$ é uma árvore que pode ser obtida com um algoritmo para árvores de espalhamento máximas, como o algoritmo de Kruskal (Figura 8). A cada passo do *loop* de AEMax-Kruskal(G, w), conectam-se dois conjuntos de vértice através da maior aresta possível. Ao final, a árvore $A = (V, E_A)$ representa um único conjunto conexo de vértices, com peso total máximo.

AEMax-Kruskal(G, w)

$E_A \leftarrow \emptyset$
para cada $v \in V[G]$
    faça Make-Set(v)
sort $E[G]$ em ordem não decrescente de pesos w
para cada $(u, v) \in E[G]$
    faça se Find-Set(u) $\neq$ Find-Set(v)
        entao $E_A \leftarrow E_A \cup \{(u, v)\}$
            Union(u, v)
retorne E_A

Figura 8: Algoritmo para encontar a Árvore de Espalhamento Máxima

3.5 Determinação do Consenso (Montagem)

Munidos da árvore gerada pela fase de refinamento, podemos obter a molécula original. A cada vértice de V, associamos um *consenso*. Um *consenso* é um vetor tal que *consenso*$[i]$ é uma listas das candidatos a ocupar a posição i (Figura 9). Cala candidato é composto de uma base $B \in \Sigma = \{A, C, T, G, \square\}$,

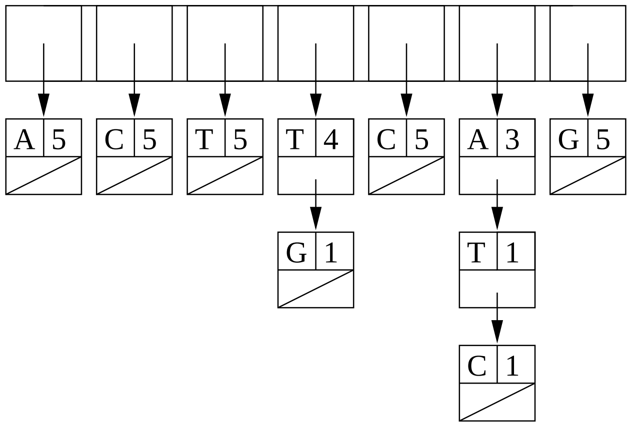

Figura 9: Fragmento Consenso.

onde □ denota um "buraco", isto é, um espaço vago na seqüência, e de um número inteiro que indica quantas vezes tal base foi indicada para ocupar esta posição.

Seja $c[v], v \in V$ o consenso associado ao vértice v. Inicialmente, $c[v]$ é obtido de $s[v]$ de forma bastante simples: há apenas uma base candidata para cada posição com uma indicação apenas. A partir daí, iremos *comprimir* as arestas de A, como indicado na Figura 10.

Um vértice de A é dito *preto* quando ele não possui vértices filhos. Caso o vértice possua filhos e estes sejam todos *pretos*, dizemos que o vértice é *cinza*. Em último caso, quando o vértice possui filhos de mais de uma cor, dizemos que ele é *branco*. A cada vértice *preto*, iremos comprimir sua aresta com seu vértice pai. Continuaremos este processo até restar um único vértice *preto* sem pai.

Quando comprimimos a aresta entre dois vértices u e v, geramos um vértice w tal que $c[w]$ seja um consenso obtido de $c[u]$ e $c[v]$ através de um algoritmo de alinhamento, semelhante ao visto em [SF 98]. Como indicado na Figura 11, um alinhamento entre consensos pode reforçar os votos de certos candidatos ou até mesmos propor novos candidatos para uma certa posição i.

Ao final, quando obtivermos um único vértice r, definimos a molécula original como uma string s tal que:

$$s[i] = MaxBase(c[r][i])$$

onde MaxBase(L) é a base mais votada na lista L.

4 Outras Heurísticas

Algumas heurísticas para o problema da montagem de fragmentos são descritas em [MS 94]. Descreveremos brevemente duas delas:

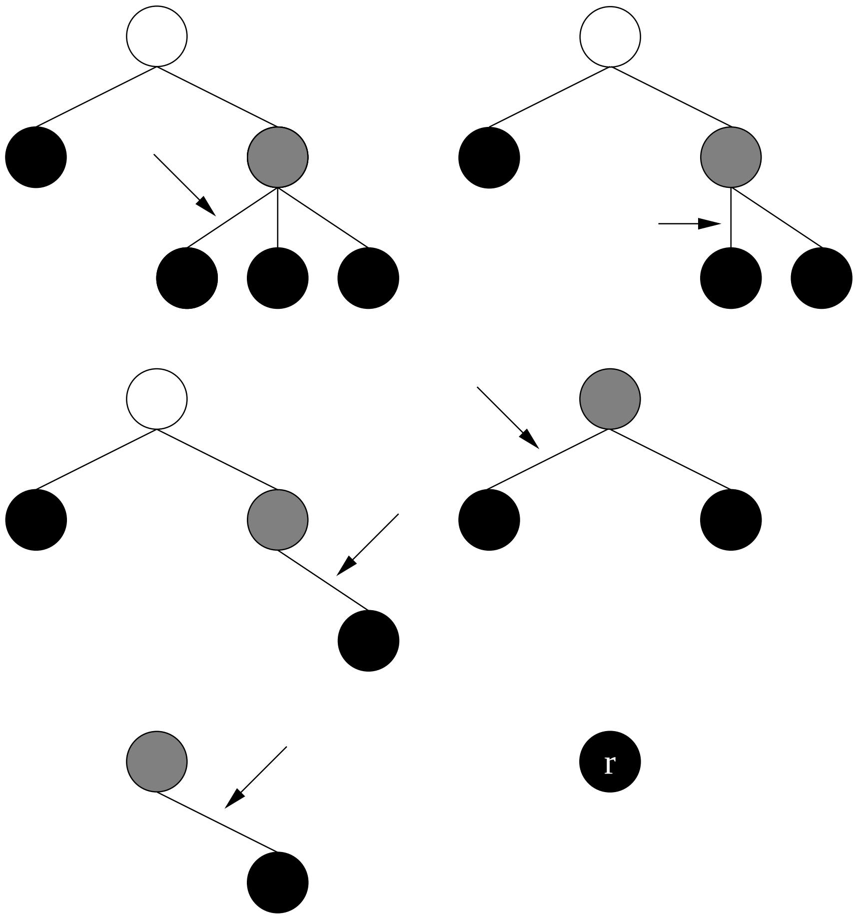

Figura 10: Colapso dos vértices na fase de montagem.

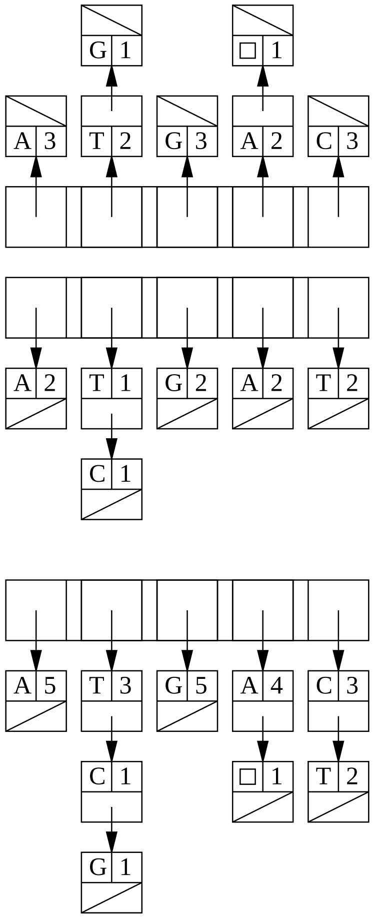

Figura 11: Alinhamento entre Consensos.

4.1 Os *contigs* de Staden

Um dos sistemas mais populares do mundo para montagem de fragmentos é o de R. Staden, pesquisador de Cambridge. Staden se baseia na idéia de *contig*: uma estrutura que engloba um conjunto de fragmentos e possui um *consenso* associado, isto é, uma seqüência ininterrupta de bases.

Os fragmentos são processados um a um. Procura-se um contig ao qual o fragmento seja semelhante. Encontrando-se um contig, o fragmento é inserido nele e recalcula-se o consenso. Caso contrário, um novo contig é criado. Desta forma, os contigs vão crescendo com o processamento. Fragmentos podem ainda relacionar dois contigs, fundindo-os.

Os consensos dos contigs representam a molécula original. Caso os fragmetos gerados no seqüenciamento não cubram toda a molécula, dois ou mais contigs serão obtidos. Mas, com cobertura e sobreposição suficientes, um único contig, representando toda a seqüência, pode ser devolvido.

4.2 Helsinki: os *grafos de intervalos*

Pesquisadores da Universidade de Helsinki, Finlândia, formalizaram a montagem de fragmentos como:

Problema 4 *Dados um conjunto de fragmentos* $f_1, f_2, ..., f_k$ *e uma constante* δ *entre 0 e 1, encontrar a seqüência mais curta possível* F *tal que, para cada* f_i*, exista uma subcadeia de* F *cuja distância de edição*[1] *a* f_i *seja, no máximo,* $\delta|f_i|$.

Para este problema, foi desenvolvido o SEQAIDS. A principal estrutura utilizada são os *grafos de intervalo*. Neste tipo de grafo, cada vértice representa um intervalo de elementos consecutivos $\langle e_i, e_{i+1}, ..., e_j \rangle$. Há uma aresta ligando dois vétices u e v se e somente se a intersecção entre seus intervalos correspondentes não for vazia. O peso associado à aresta é igual ao tamanho desta intersecção.

Usando um algoritmo de programação dinâmica, o sistema compara os fragmentos dois a dois. Este algoritmo foi modificado, de forma que leva tempo $O(\delta mn)$, onde m e n são os comprimentos dos fragmentos comparados e δ é uma constante proporcional a taxa de erro permitida. Um grafo G é criado, sendo cada um de seus vértices associado a um fragmento e cada aresta a indicação de sobreposição entre eles. SEQAIDS procura em G um subgrafo que seja uma grafo de intervalos e realiza a montagem a partir dele.

5 Programas

Um programa para avaliação de similaridade já foi desenvolvido por nós e pode ser encontrado em:

[1] O número de alterações (inserções, remoções ou substituições de bases) necessárias em uma seqüência para se obter a outra.

http://www.linux.ime.usp.br/~thsant/ic/

Estamos desenvolvendo um software para a montagem de fragmentos baseado em nossa heurística (definida na seção 3). A linguagem escolhida para a implementação é a JAVA[2]. Como tal tipo de software requer velocidade, linguagens como C ou C++ seriam escolhas mais naturais. Porém, como nosso interesse é didático e não comercial, JAVA foi escolhida por sua elegância e pela facilidade de exposição dos programas na *Web*, sob a forma de *applets* (programas cuja execução pode ser feita em *web browsers*).

[2]JAVA é *trademark* da Sun Microsystems.

Referências

[CLR 90] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[FSW 97] C. E. Ferreira, C. C. Souza, Y. Wakabayashi. *Rearrangement of DNA Fragments: a branch-and-cut algorithm*. Relatório Técnico. Instituto de Matemática e Estatística - Universidade de São Paulo, 1997.

[MS 94] J. Meidanis, J. C. Setúbal. *Uma Introdução à Biologia Computacional*. IX Escola de Computção, Recife, 1994.

[Shamir 98] R. Shamir. *Algorithms for Molecular Biology*. Leituras de Aula, 1998.

[SF 98] T. T. Santos (aluno), C. E. Ferreira (professor/orientador). *Algoritmos para Rearranjo de Seqüências de DNA*. Relatório Final de Atividades - PIBIC/CNPq, 1998.

[Web 1] http://www.linux.ime.usp.br/~thsant

[Web 2] http://www.math.tau.ac.il/~shamir