Computação de DNA*

Índice

.

* *Material baseado nas notas de aula do Prof. Leandro Nunes de Castro (Mackenzie/SP). Reprodução de conteúdo autorizada pelo autor. Material revisado pelo Prof. Romis R. F. Attux em 2007.*

1. Introdução

- A *computação de DNA* é uma das sub-áreas de uma linha de pesquisa mais ampla denominada de *computação biomolecular*.
- Em linhas gerais, a computação molecular emprega (bio)moléculas e operações para a manipulação destas (bio)moléculas para resolver problemas e realizar computação.
- Questões importantes a serem verificadas:
  - Qualquer algoritmo pode ser "simulado" via computação de DNA?
  - Quais as dificuldades em se projetar um computador de DNA?
- Diversos modelos de computação de DNA vêm sendo propostos para responder estas e outras questões. Estes modelos podem ser divididos em dois grandes grupos:
  - Modelos baseados em filtragem;
  - Modelos formais.

- De maneira simplificada, a computação de DNA emprega moléculas de DNA como estrutura de dados e manipula estas moléculas de forma a realizar computação.

1.1. Computação de DNA x Computadores Tradicionais

- A computação de DNA utiliza DNA como estrutura de dados. Alfabeto quaternário {A,C,T,G} em vez de binário {0,1}.
- Computadores de DNA operam de forma massivamente paralela.
- A computação de DNA opera em nível molecular, um limite difícil de ser atingido pela indústria de semicondutores.
- Os computadores de DNA demandam muito pouca energia e são altamente econômicos na armazenagem de informação.
- Os computadores de DNA são eficientes na resolução de problemas NP-completos.

2. Conceitos Básicos de Biologia Molecular

2.1. A Molécula de DNA

- Toda a informação genética em organismos celulares está armazenada no DNA, que consiste em *cadeias de polímeros*, usualmente conhecidas como *cadeias de DNA*.

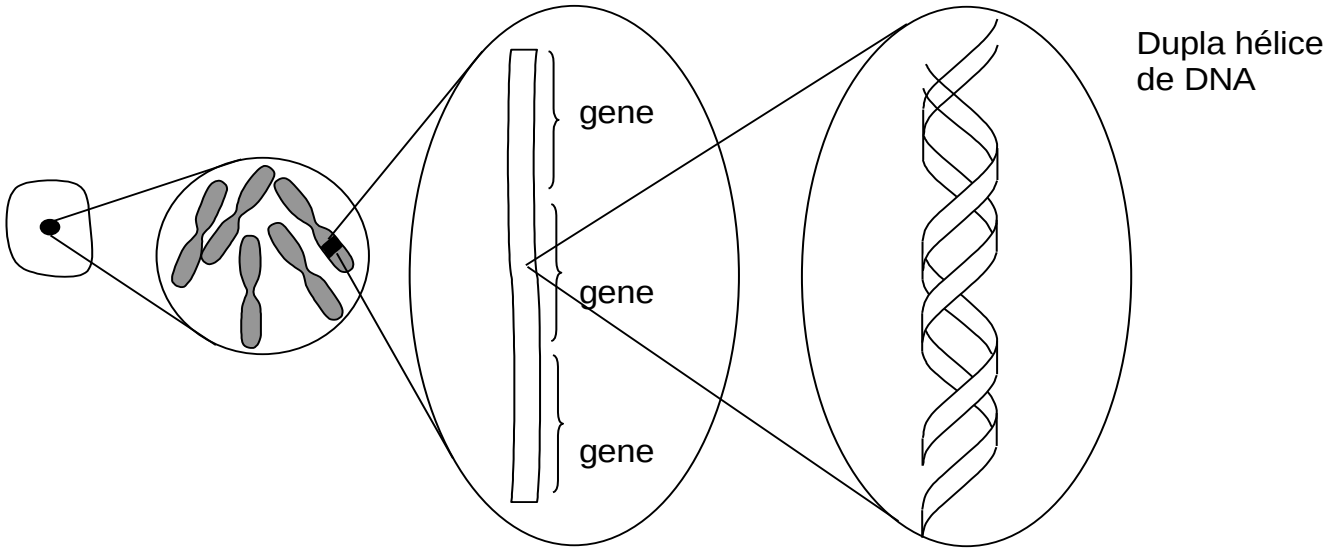


Figura 1: A molécula de DNA encontra-se no núcleo das células.

- As cadeias de DNA são formadas por quatro unidades de ácidos nucléicos, chamados de *desoxirribonucleotídeos* ou simplesmente *nucleotídeos*.
- Existem quatro nucleotídeos no DNA, e cada nucleotídeo é composto por três partes: uma *molécula base*, um *açúcar* e um *grupo fosfato*.
- As quatro bases são: *adenina* (A), *citosina* (C), *guanina* (G) e *timina* (T).
- Como os nucleotídeos diferem apenas pelas bases, eles são geralmente denominados *bases*.
- Números de 1′ a 5′ são usados para denotar os cinco átomos de carbono do açúcar do nucleotídeo. O grupo de fosfato se liga ao átomo de carbono 5′, e a base se liga ao átomo de carbono 1′.
- Cada cadeia possui, por convenção química, um terminal 5′ e um terminal 3′. Portanto, cada cadeia possui uma orientação.

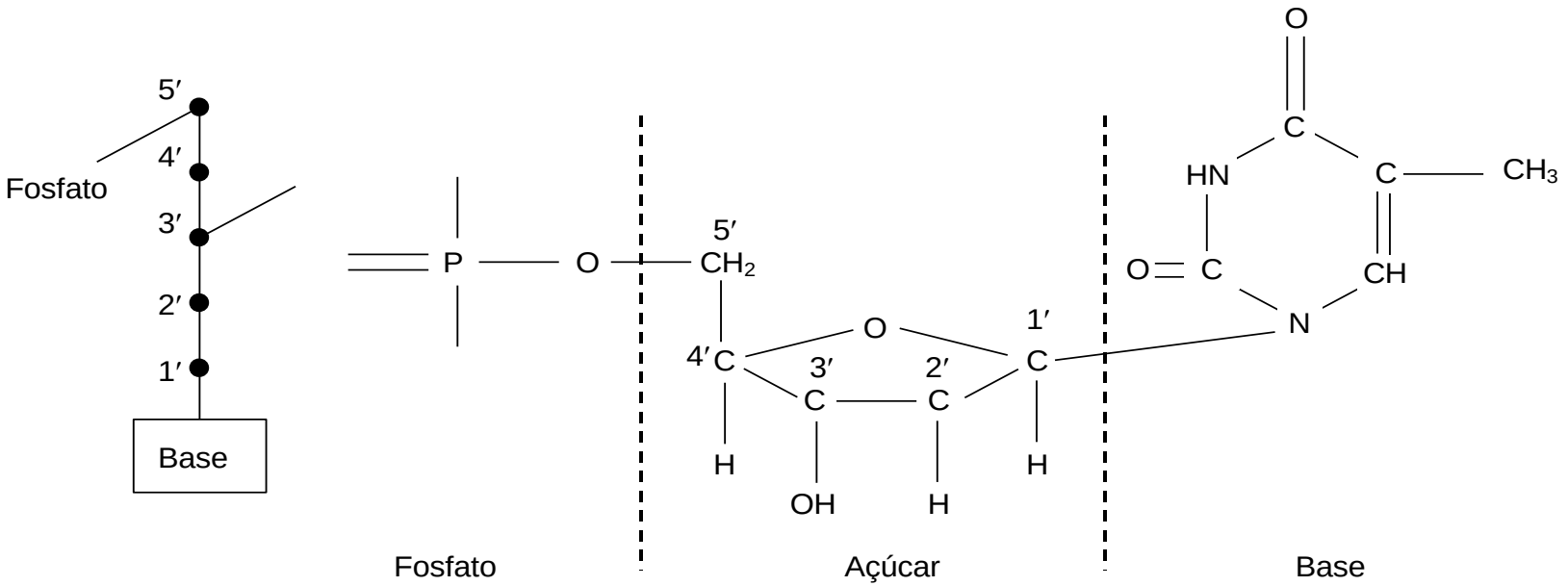


Figura 2: Estrutura química do nucleotídeo e uma de suas representações.

- Os nucleotídeos podem se ligar de duas formas distintas:
  - O grupo de fosfato 5′ de um nucleotídeo se junta ao grupo de hidroxila 3′ de outro nucleotídeo formando uma *ligação covalente*;
  - A base de um nucleotídeo interage com a base de outro para formar uma *ponte de hidrogênio*, que é uma ligação mais fraca que uma ligação covalente.

- Uma característica importante da ligação de nucleotídeos (ligação covalente) é que qualquer nucleotídeo pode se ligar para formar uma sequência.
- Por outro lado, as ligações entre as bases só ocorrem pela atração entre pares específicos de bases:
  - A se liga com T
  - C se liga com G
- Estas ligações espelham a *complementaridade de Watson-Crick*.

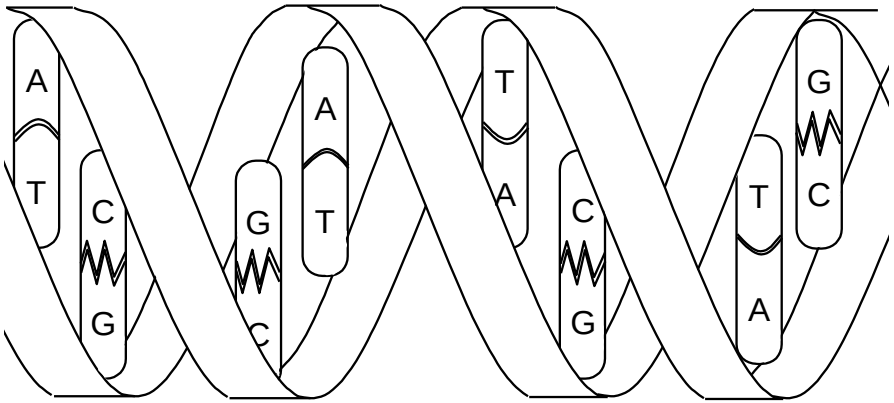
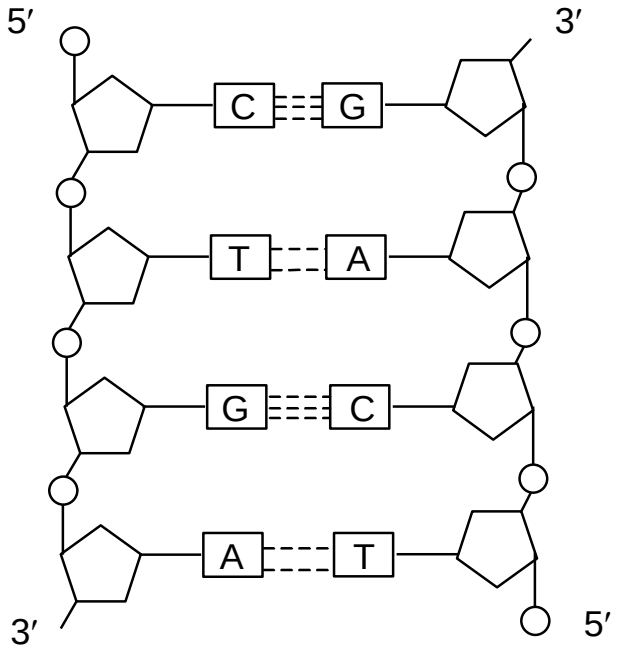


Figura 3: Molécula de DNA ilustrando a complementaridade de Watson-Crick.

- Algumas representações alternativas:



5′ – T C G A T T G A A C C – 3′
3′ – A G C T A A C T T G G – 5′

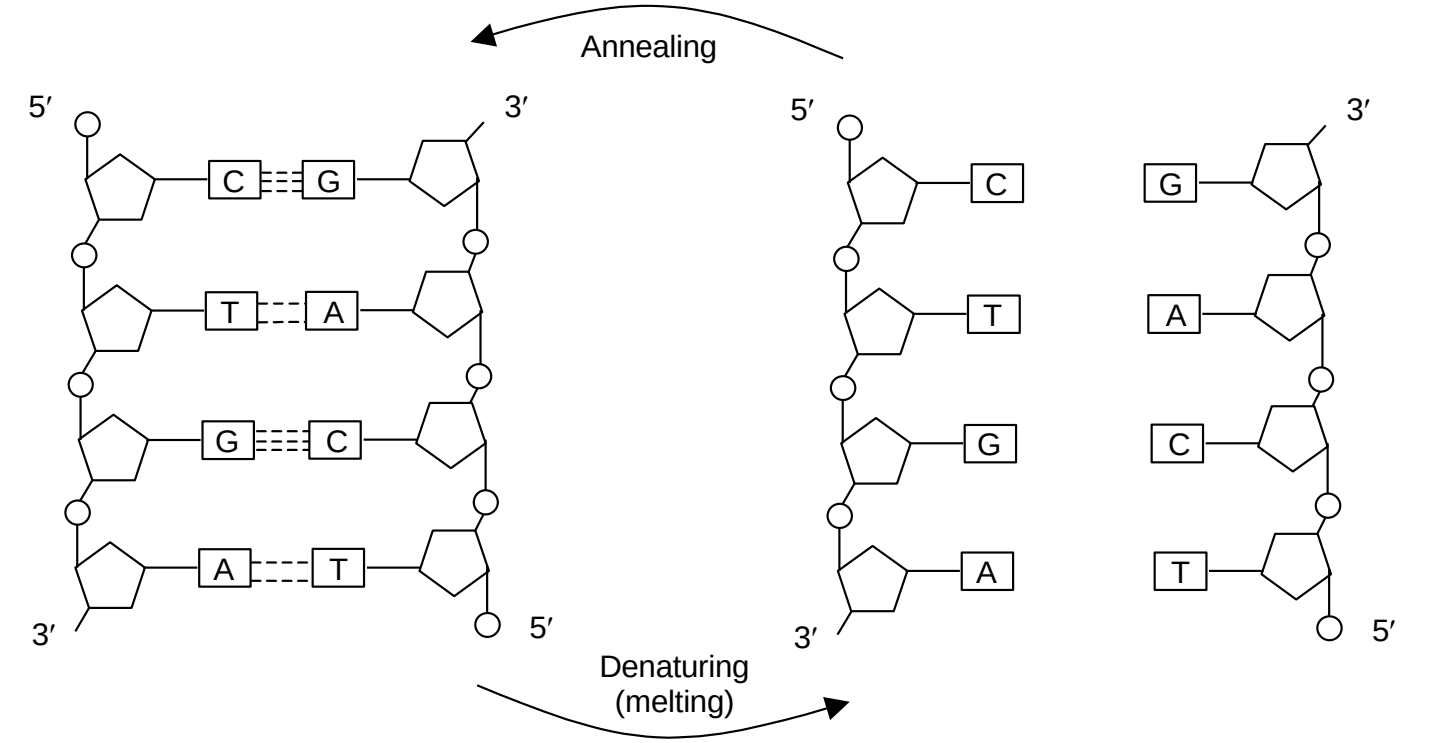
5′ – T C G A T T G A – 3′

3′ – A A C T T G G – 5′

(Sticky ends)

2.2. Manipulando o DNA

- Todas as técnicas de computação de DNA envolvem a aplicação de um conjunto específico de operações biológicas a um conjunto de moléculas.
  - *Desnaturação*: separa cadeias de DNA (separa as bases)
  - *Annealing*: junta cadeias de DNA (une pelas bases)
  - *Extensão de polimerase*: completa cadeias incompletas
  - *Degradação por nuclease*: encurta cadeias de DNA
  - *Endonucleases*: cortam moléculas de DNA (separa pelas ligações covalentes)
  - *Ligação*: une moléculas de DNA (une pelas ligações covalentes)
  - *Modificação de nucleotídeos*: insere ou deleta pequenas sequências
  - *Amplificação* (PCR): multiplica moléculas de DNA
  - *Eletroforese de gel*: mede o comprimento de moléculas de DNA
  - *Filtragem*: separa ou extrai moléculas específicas
  - *Síntese*: cria moléculas de DNA
  - *Sequenciamento*: lê a sequência de uma molécula de DNA



5′ – T C G A T T G A A – 3′ (single strand)

3′ – A A C T T C – 5′ (single strand)

↓ (Annealing)

5′ – T C G A T T G A A – 3′
3′ – A A C T T C – 5′

Figura 4: Desnaturação e annealing.

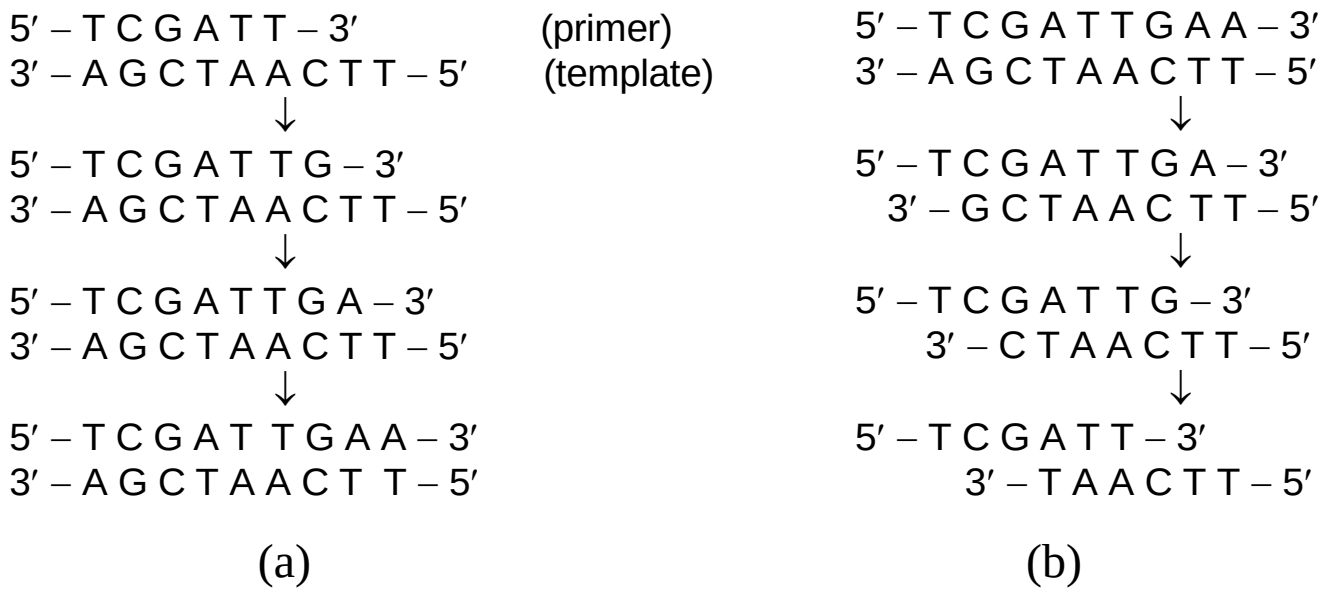


Figura 5: (a) Extensão de polimerase. (b) Degradação por nuclease.

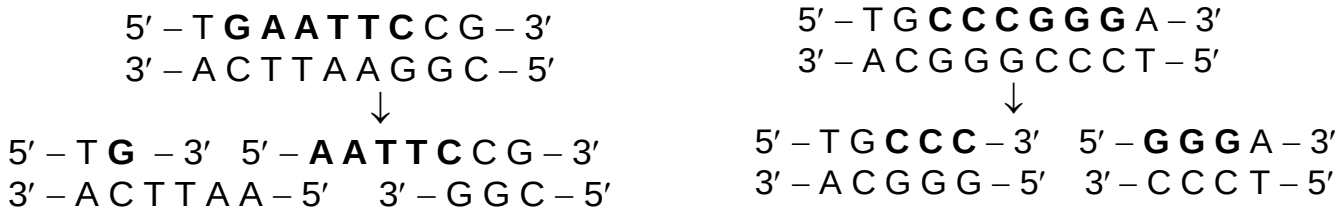


Figura 6: Corte por endonuclease.

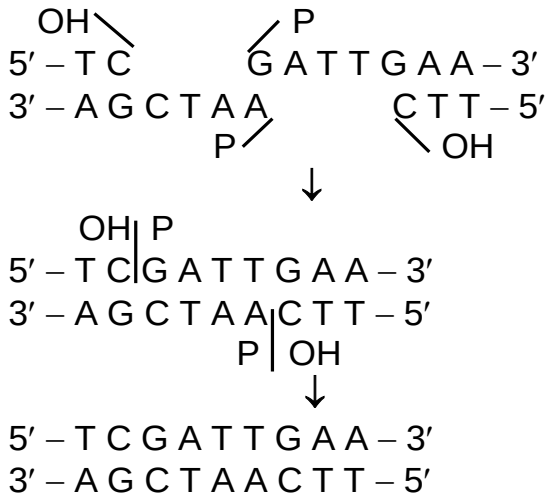


Figura 7: Ligação.

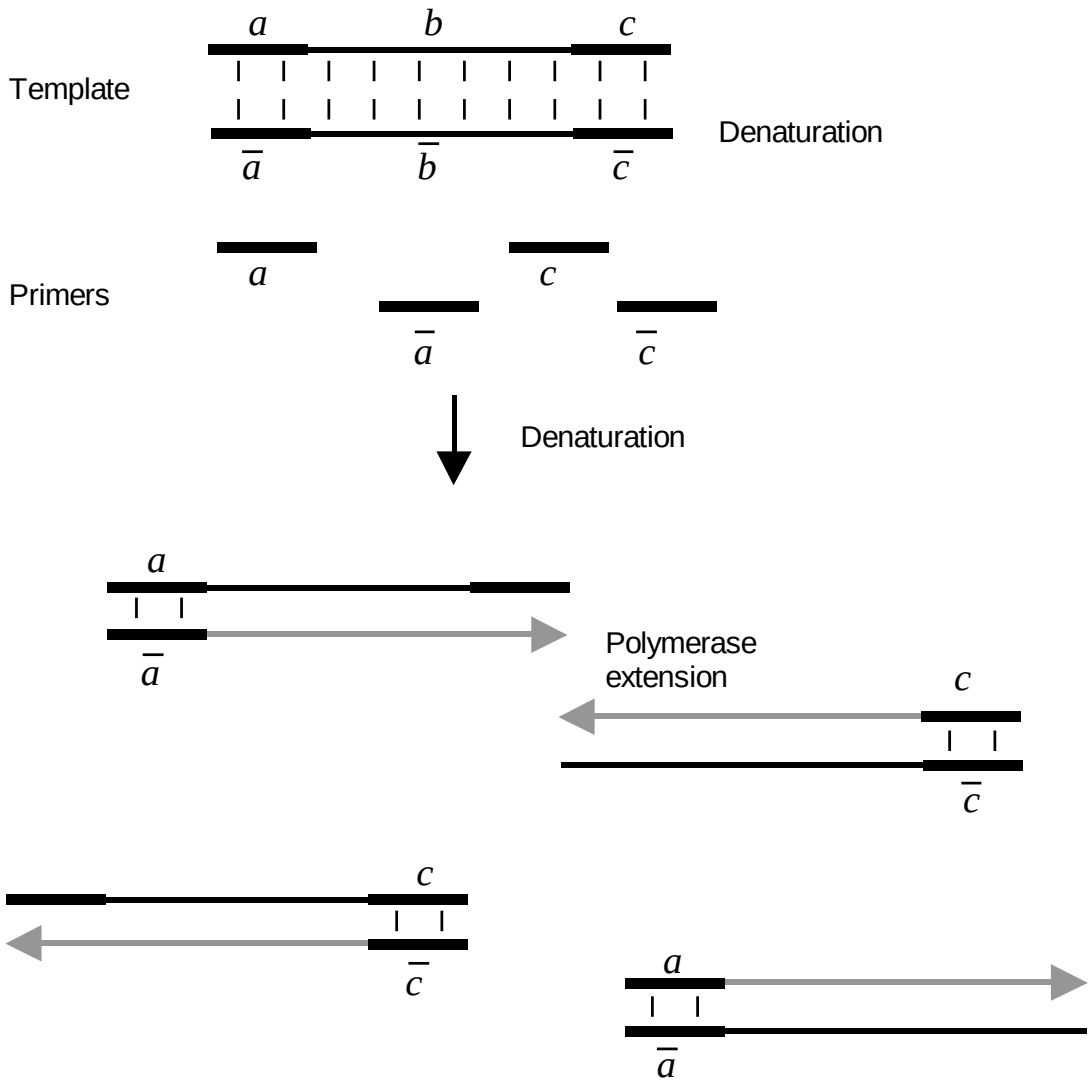


Figura 8: PCR (Polymerase Chain Reaction).

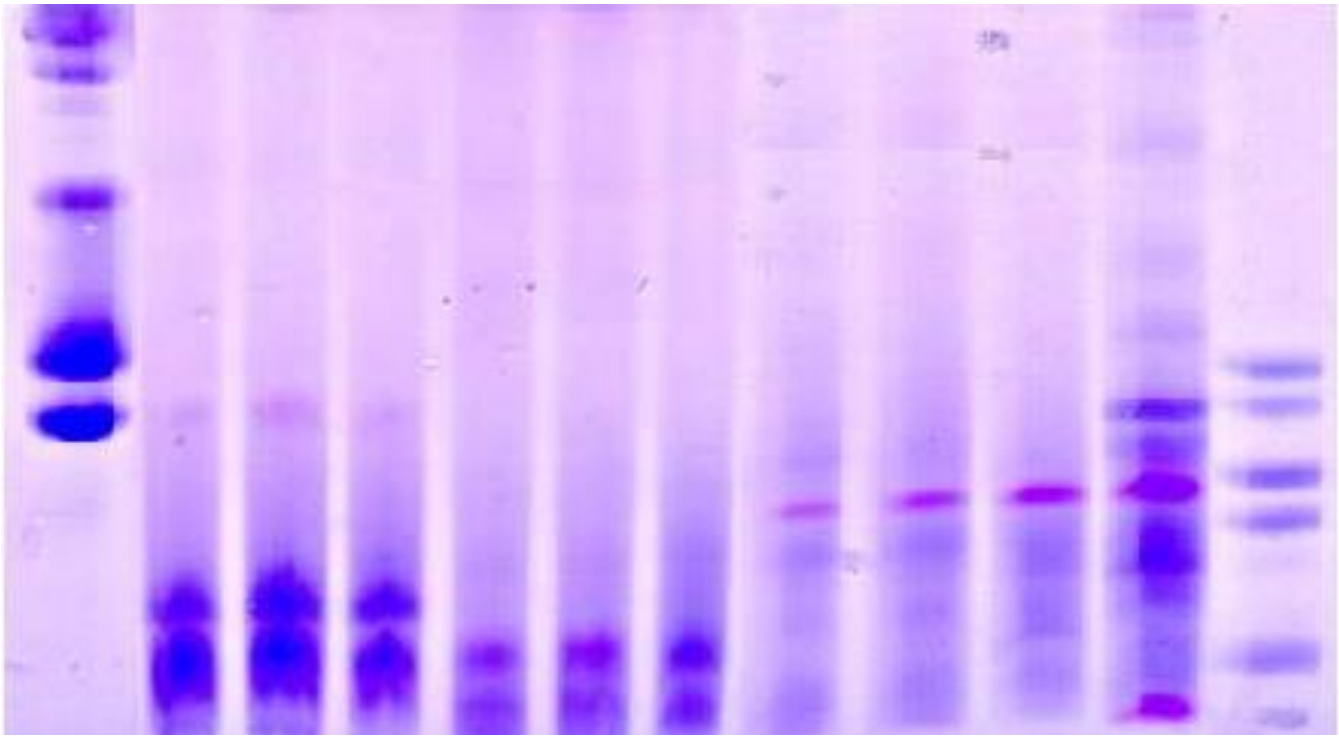


Figura 9: Fotografia de uma eletroforese de gel de uma molécula de proteína.

3. Modelos Baseados em Filtragem

- Em todos os modelos baseados em filtragem, um grande conjunto de *strings* é gerado e diversos processos de filtragem são aplicados para isolar *strings* que não podem ser solução do problema.

3.1. O Experimento de Adleman

- O primeiro experimento bem-sucedido na utilização de moléculas de DNA e técnicas de manipulação de DNA na solução de problemas foi apresentado por Adleman em 1994. Neste trabalho, Adleman resolveu uma pequena instância de um problema de caminho hamiltoniano (HPP).
  - Um grafo direcionado G com os nós de entrada e saída definidos, v_{in} e v_{out}, possui um caminho hamiltoniano se e somente se existe uma sequência de ramos direcionados e_1, e_2,…,e_z (caminho) que inicia em v_{in} e termina em v_{out}, passando por todos os nós do grafo.

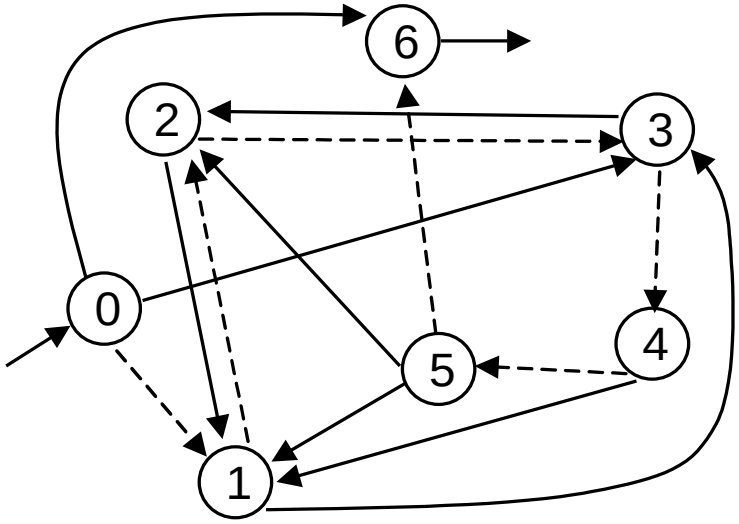


Figura 10: Grafo usado no experimento de Adleman.

- O HPP pode ser resolvido de forma exaustiva e, embora haja algoritmos capazes de resolver instâncias específicas de forma eficiente, todos os algoritmos de solução possuem complexidade exponencial ou fatorial, no pior caso.
  - Portanto, na prática, o HPP é um problema intratável usando as técnicas tradicionais de computação.

- Com a proposta de Adleman usando computação de DNA, o número de operações em laboratório a serem empregadas na solução do HPP é linear em função do tamanho do grafo (número de vértices) (o problema é NP-completo sob a perspectiva da computação digital).

- Algoritmo para resolver o problema:

Passo 1: Gere caminhos aleatórios pelo grafo.

Passo 2: Mantenha apenas aqueles que iniciam em v_{in} e terminam em v_{out}.

Passo 3: Se o grafo possui n vértices, mantenha somente aqueles caminhos de comprimento n.

Passo 4: Mantenha apenas aqueles que passam por cada vértice uma única vez.

Passo 5: Se um caminho permanecer, *aceite*; caso contrário, *rejeite*.

- Antes de aplicar o algoritmo determinístico acima para resolver este problema, é necessário "codificar" os possíveis caminhos utilizando moléculas de DNA.

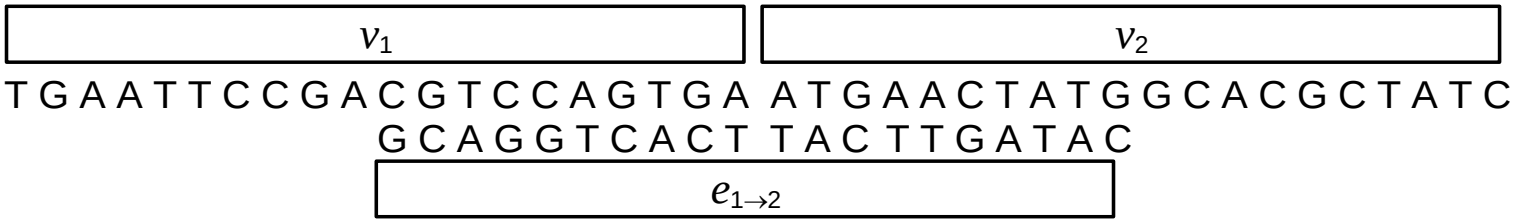
- Adleman codificou cada nó do grafo utilizando uma sequência de nucleotídeos (*single strand*) de comprimento 20.
- A codificação foi escolhida aleatoriamente e o comprimento de 20 bases foi adotado para garantir uma codificação diferente para cada nó.
- Uma grande quantidade de nucleotídeos foi gerada via PCR e colocada em um tubo de ensaio.
- Os ramos foram codificados da seguinte forma:

v_1 = 5′ – T G A A T T C C G A | C G T C C A G T G A – 3′
v_2 = 5′ – A T G A A C T A T G | G C A C G C T A T C – 3′
v_3 = 5′ – C A T A G T C C G A | T T A G C A G T A G – 3′

↓

$e_{1\to2}$ = 3′ – G C A G G T C A C T | T A C T T G A T A C – 5′
$e_{2\to1}$ = 3′ – C G T G C G A T A G | A C T T A A G G C T – 5′
$e_{1\to3}$ = 3′ – G C A G G T C A C T | G T A T C A G G C T – 5′

(a)



(b)

Figura 11: Método de codificação usado por Adleman para resolver o problema do caminho hamiltoniano.

Método de Solução

Passo 1: Gere caminhos aleatórios pelo grafo.

- Para ligar os vértices de modo a formar caminhos, oligonucleotídeos $\bar{O}_i$ complementares àqueles representando os nós (O_i) são gerados.
- Para unir as '*single strands*' e gerar os caminhos aleatórios pelo grafo foi feito o annealing e foi utilizada uma reação de ligação.

Passo 2: Mantenha apenas aqueles caminhos que iniciam em v_{in} e terminam em v_{out}.

- Em seguida, uma PCR empregando $\bar{O}_0$ e $\bar{O}_6$ como primers é utilizada para amplificar o resultado do passo anterior.
- Dessa forma, apenas as moléculas que iniciam no nó 0 e terminam no nó 6 foram amplificadas.
- Um processo de filtragem separa estas moléculas das demais.

Passo 3: Se o grafo possui *n* vértices, mantenha somente aqueles caminhos de comprimento *n*.

- A eletroforese de gel é utilizada para separar as moléculas (*double stranded*) de acordo com seus comprimentos.
- Apenas as cadeias com comprimento de 140 pares de base (7 vértices) foram mantidas.
- Ao final deste passo, existem diversas moléculas que iniciam no nó 0, terminam no nó 6 e passam por 7 nós.

Passo 4: Mantenha apenas aqueles que passam por cada nó uma única vez.

- Com um passo para cada vértice, foi possível verificar se as moléculas restantes possuíam estes vértices (filtragem).

Passo 5: Se um caminho permanecer, *aceite*; caso contrário, *rejeite*.

Discussão

- Adleman demorou 7 dias para completar seu experimento.
  - Entretanto, a quantidade de nucleotídeos necessária para resolver o problema cresce linearmente com o número de nós do problema.
- Portanto, um problema NP-completo (sob a perspectiva da computação digital), em que a demanda por recursos cresce exponencial ou fatorialmente com o tamanho do problema, pode ser resolvido em tempo linear devido ao paralelismo da computação de DNA.
- Uma das dificuldades do procedimento adotado por Adleman está relacionada à quantidade de *single strands* que devem ser geradas para codificar os diversos caminhos possíveis no grafo.
- Como o HPP é um problema NP-completo e ele foi resolvido por uma técnica de computação de DNA, em teoria é possível utilizar esta mesma estratégia para resolver qualquer problema da classe NP-completo.

- Entretanto, isso não significa que qualquer instância de um problema NP possa ser resolvida de forma factível por computação de DNA.
- Adleman resolveu o problema HPP usando a força bruta: ele projetou um sistema capaz de gerar e avaliar todas as possíveis soluções para uma dada instância do HPP.
- A característica marcante do experimento de Adleman foi o paralelismo massivo das moléculas de DNA.
- Em 1994, quando Adleman executou seu experimento, um computador do tipo desktop comum era capaz de executar 10^6 operações por segundo e o supercomputador mais rápido conhecido podia executar aproximadamente 10^{12} operações por segundo.
  - O computador de DNA de Adleman era capaz de executar 10^{14} operações por segundo, assumindo que cada ligação corresponde a uma operação. Escalonando o passo de ligação, talvez seja possível elevar este número para 10^{20}.

- Além disso, a quantidade de energia consumida era muito baixa, da ordem de 2×10^{19} operações por joule, um valor próximo do limite proposto pela segunda lei da termodinâmica (34×10^{19}).
- Os supercomputadores modernos operam na casa de 10^9 operações por joule.
- Por último, em um computador de DNA, um bit de informação pode ser armazenado em um nanômetro cúbico de DNA, o que era aproximadamente 10^{12} vezes mais eficiente que os dispositivos de armazenagem conhecidos na época.
- Em resumo, um computador de DNA podia ser, em 1994, 1.200.000 vezes mais rápido do que o supercomputador mais rápido conhecido, além de permitir um armazenamento de informação 10^{12} vezes mais eficiente e consumir 10^{10} vezes menos energia que os computadores existentes.

3.2. A Solução de Lipton para o Problema SAT

- Lipton mostrou como empregar procedimentos de DNA para resolver o problema denominado *satisfiability problem for propositional formulas* (SAT).

- SAT é um problema de busca NP-completo que pode ser definido como a seguir.
- Dado um conjunto finito de variáveis lógicas $E = \{e_1, e_2, \ldots, e_n\}$, defina um *literal* como sendo uma variável, e_i, ou seu complemento $\bar{e}_i$. Se e_i é verdadeira, então $\bar{e}_i$ é falsa, e vice-versa.
- Seja uma *cláusula* C_j um conjunto de literais $\{e_1^j, e_2^j, \ldots, e_l^j\}$.
- Uma instância I do problema SAT consiste em um conjunto de cláusulas, mais especificamente, uma fórmula booleana da forma $C_1 \wedge C_2 \wedge \ldots \wedge C_m$, onde cada cláusula é uma proposição que pode ser construída a partir de variáveis proposicionais e_i, $i = 1,\ldots,n$, e *conectivos lógicos* AND ($\wedge$), OR ($\vee$), e NOT ($\neg$).
- O problema SAT corresponde, portanto, a especificar um valor booleano para cada variável $e_i \in E$, $i = 1,\ldots,n$, tal que a fórmula booleana seja verdadeira.
- Um aspecto chave explorado por Lipton foi o fato de podermos representar o problema SAT como um problema de busca em grafos.

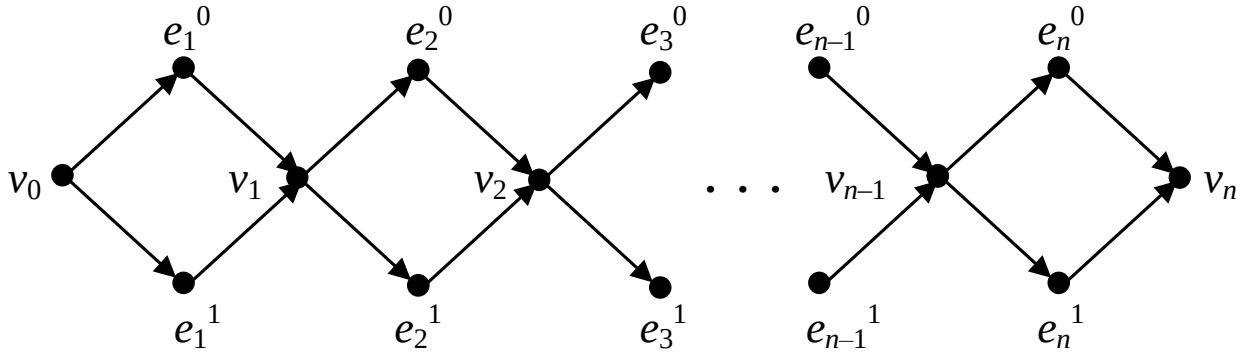
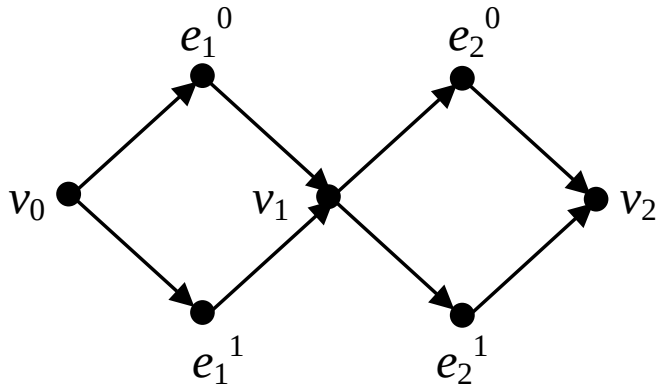


Figura 12: Representação em grafo do problema SAT.

- De acordo com a figura acima, um caminho genérico pode ser representado por uma sequência $v_0 e_1^{i1} v_1 e_2^{i2} \ldots v_{n-1} e_n^{in} v_n$, onde a variável e_j pode assumir o valor verdade i_j, $j = 1,\ldots,n$.
- Neste grafo, todos os caminhos que iniciam em v_0 e terminam em v_n correspondem a uma *string* binária.
  - Por exemplo, o caminho $v_0 e_1^1 v_1 e_2^0 v_2 e_3^0 \ldots v_{n-1} e_n^1 v_n$ codifica a *string* binária 100…1.

- Assim como Adleman, Lipton propôs um método composto por duas fases principais:
  - Gerar todas as soluções possíveis (caminhos no grafo);
  - Filtrar aquela(s) que satisfaz(em) os requisitos de solução.
- Lipton propôs codificar os grafos em um tubo de ensaio como feito por Adleman, e utilizou o mesmo esquema de codificação.
- Por outro lado, a forma de solução proposta por Lipton foi diferente, em essência.
- Ele propôs trabalhar com *operações em tubos de ensaio*.
- Para exemplificar, foi verificada a seguinte expressão: $F = (e_1 \vee e_2) \wedge (\bar{e}_1 \vee \bar{e}_2)$.



- Lipton construiu uma série de tubos de ensaio, onde o primeiro tubo, t_0, supostamente contém todas as possíveis soluções do problema.
- Ele propôs, dentre outras, uma operação de extração $E(t,i,a)$ que *extrai* todas as sequências no tubo t cujo i-ésimo bit é a, $a \in \{0,1\}$.
- Em seguida, ele propôs o seguinte algoritmo para resolver o problema:

Passo 1: Seja t_1 o tubo correspondente a $E(t_0,1,1)$. O tubo contendo o restante é t_1', e t_2 é $E(t_1',2,1)$. Junte o conteúdo de t_1 e t_2 produzindo o tubo t_3.

Passo 2: Seja t_4 o tubo correspondente a $E(t_3,1,0)$. O tubo com o conteúdo restante é t_4', e t_5 é $E(t_4',2,0)$. Junte o conteúdo de t_4 e t_5 produzindo o tubo t_6.

Passo 3: Verifique se há alguma molécula de DNA no último tubo. Se houver, *aceite*; caso contrário, *rejeite*.

Tubo de ensaio	Strings presentes
t_0	00, 01, 10, 11
t_1	10, 11
t_1'	00, 01
t_2	01
t_3	01, 10, 11
t_4	01
t_4'	10, 11
t_5	10

Caso Genérico

- Qualquer problema SAT com n variáveis e m cláusulas pode ser resolvido com, no máximo, $O(m)$ operações de extração e uma operação de detecção.
- Sejam C_1, C_2,…, C_m as m cláusulas de uma fórmula proposicional.
- Construa m tubos, t_0, t_1,…, t_m, de forma que t_k seja o conjunto de números com n-bits tal que $C_1(e) = C_2(e) = \ldots = C_k(e) = 1$, onde $C_i(e)$ corresponde ao valor verdade da cláusula C_i sobre o conjunto de variáveis e.

- Para t_0, use todas as combinações possíveis de cláusulas.
- Dado t_k, construa t_{k+1} da seguinte forma. Assuma que C_{k+1} está na forma disjuntiva: $o_1 \vee ... \vee o_l$, onde o_i é um literal e $\bar{o}_i$ é o complemento de um literal.
- Para cada literal opere da seguinte forma:
  - Se $o_i = e_j$, então gere $E(t_k,j,1)$.
  - Senão, se $o_i = \bar{e}_j$, gere $E(t_k,j,0)$.
- Cada operação de extração é efetuada e o restante é colocado em um outro tubo.
- Junte todos os tubos e faça uma detecção. Se sobrar algo, então a fórmula é satisfeita.

Discussão

- Em essência, nenhum método é melhor do que a busca exaustiva na solução do SAT.

- Neste sentido, o método utilizado pela computação de DNA não é melhor do que os de busca exaustiva, porém ele faz uso do paralelismo massivo das moléculas de DNA e suas técnicas de manipulação.
- Um dos principais resultados da solução proposta por Lipton foi a verificação de que seu procedimento permite resolver qualquer problema SAT de *n* variáveis e *m* cláusulas com, no máximo, *m* passos de extração e uma detecção.

3.3. Linguagem de Programação de Tubos de Ensaio

- Os aspectos práticos das propostas de Lipton e Adleman dependem das tecnologias de manipulação de DNA disponíveis.
- Até mesmo os algoritmos utilizados para resolver os problemas poderiam ser mudados.
- O que é importante, neste caso, é provar que a computação é factível.

- Meios puramente bioquímicos foram empregados para resolver problemas NP-completos em um tempo linear em relação à quantidade de operações de laboratório.
- Estas operações, em uma formulação abstrata, são uma grande contribuição derivada da proposta de Adleman: delas resulta uma espécie de *linguagem de programação de tubos de ensaio* baseada em moléculas de DNA colocadas em tubos de ensaio e em mecanismos para manipulação destas moléculas.

O Modelo Irrestrito

- Um *tubo de ensaio* é um conjunto de moléculas de DNA, ou seja, um multi-conjunto de *strings* finitas construídas a partir de um alfabeto {A,C,G,T}.
- Dado um tubo, é possível realizar quatro operações básicas:

1. `Separate` (`extract`) dado um tubo t e uma palavra w (cadeia de símbolos w pertencentes ao alfabeto {A,C,G,T}), produza dois tubos $+(t,w)$ e $-(t,w)$, onde $+(t,w)$ consiste de todas as cadeias de DNA em t que contêm w como sub-sequência, e $-(t,w)$ consiste de todas as cadeias de DNA em t que não contêm w como sub-sequência.
2. `Merge`: dado um conjunto de tubos t_1, t_2, …, t_m, produza um tubo com o conteúdo de todos os tubos: $\cup(N_1,N_2,\ldots,N_m) = N_1 \cup N_2 \cup \ldots \cup N_m$.
3. `Detect`: dado um tubo t, *aceite* se t contém pelo menos uma molécula de DNA, e *rejeite* caso contrário.
4. `Amplify`: dado um tubo t, produza duas cópias t_1 e t_2: $t = t_1 = t_2$.

- Essas quatro operações podem ser empregadas para escrever programas que recebem como entrada um tubo e produzem como saída uma resposta *aceite* (YES), *rejeite* (NO) ou um conjunto de tubos.

- Além dessas operações, o experimento de Adleman utiliza a complementaridade de Watson-Crick e as seguintes modificações da operação `separate`:
  1. `Length-separate`: dado um tubo t e um inteiro n, produza o tubo $(t, \leq n)$ que consiste de todas as cadeias em t de comprimento menor ou igual a n.
  2. `Position-separate`: dado um tubo t e uma palavra w, produza um tubo $B(t,w)$ que possui todas as cadeias em t que iniciam com a palavra w; ou produza o tubo $E(t,w)$ que contém todas as cadeias em t e que terminam com a palavra w.
- Exemplos de aplicação:

```
procedure [out] = extract(t,A,T,G)
  t ← −(t,T)
  t ← −(t,G)
  t ← −(t,A)
  out ← detect(t)
end procedure
```

- O que o programa acima faz?

```
procedure [out] = HPP(t,vin,vout)
  t ← B(t,vin)
  t ← E(t,vout)
  t ← (t,≤ 140)
  for i=1 to 5 do
    t ← +(t,si)
  end for
  out ← detect(t)
end procedure
```

- Na proposta de Lipton para o problema SAT, uma operação extract $E(t,i,a)$ que extrai todas as sequências de um tubo t cujo i-ésimo bit é igual a a, foi definida:

  $E(t,i,a) = +(t,e_i^a)$,

  $E^-(t,i,a) = -(t,e_i^a)$,

  onde $E^-(t,i,a)$ extrai todas as sequências no tubo t cujo i-ésimo bit é complementar a a.

```
procedure [out] = SAT(t)
  t1 ← +(t,e1^1)
  t1' ← −(t,e1^1)
  t2 ← +(t1',e2^1)
  t3 ← merge(t1,t2)
  t4 ← +(t3,e1^0)
  t4' ← −(t3,e1^0)
  t5 ← +(t4',e2^0)
  t6 ← merge(t4,t5)
  out ← detect(t6)
end procedure
```

A Linguagem DNA Pascal

- Com o objetivo de fornecer um modelo em alto nível para a computação molecular, foi introduzida uma outra linguagem de programação combinando elementos de Pascal com operadores de manipulação de DNA.
- Nesta linguagem, denominada de DNA Pascal, tubos de ensaio com moléculas de DNA foram mapeados em variáveis contendo palavras do alfabeto {0,1}.
  - `Initialization`: preenche um conjunto de variáveis t com $\{0,1\}^n$, $t := \text{In}(n)$.
  - `Empty word`: $t := \{\varepsilon\}$.
  - `Union`: união de dois conjuntos de variáveis t_1 e t_2, $t := t_1 \cup t_2$.
  - `Extraction`: filtra todas as palavras de um conjunto de variáveis t_1 que possuem um padrão especial. Os autores propuseram dois tipos de procedimentos de extração: (1) uma extração de bits, e (2) uma extração de sub-palavras.

- `Bit extraction`: procura um bit especial b em uma posição particular k, $t := \mathrm{Bx}(t_1,b,k)$.
- `Sub word extraction`: a extração procura uma sub-palavra especial w em qualquer lugar da palavra, $t := \mathrm{Sx}(t_1,w)$.
- `Concatenation`: a concatenação de dois conjuntos de variáveis t_1 e t_2 é $t := t_1.t_2$.
- `Right cut`: $t := t_1/$, onde $t_1/ = \{z/ \mid z \in t_1\}$ e $za/ = z \ \forall a \in \{0,1\}$ e $\varepsilon/ = \varepsilon$.
- `Left cut`: $t := /t_1$, onde $/t_1 = \{/z \mid z \in t_1\}$ e $/za = z \ \forall a \in \{0,1\}$ e $/\varepsilon = \varepsilon$.
- `Right append`: $t := t_1.a$, onde $t_1.a = \{z.a \mid z \in t_1\}$.
- `Left append`: $t := a.t_1$, onde $a.t_1 = \{a.z \mid z \in t_1\}$.

- Alguns testes condicionais também foram propostos:
  - `Subset test`: $t_1 \subseteq t_2$.
  - `Detect test`: $t = 0$.
  - `Membership test`: $x \in t$.

4. Um Breve Resumo dos Modelos Formais

- Virtualmente cada pesquisador em computação de DNA possui sua própria forma de utilizar DNA para computar.
- Isso indica que esta linha de pesquisa ainda está explorando as diversas possibilidades de implementação de um computador de DNA.
- Entretanto, diversos *modelos formais*, alguns introduzidos antes do experimento de Adleman, têm sido propostos com o objetivo de fornecer um estudo teórico sobre a computação de DNA. Dentre eles é possível citar os:
  - *sticker systems*: ROWEIS *et al.* (1996) introduziram um modelo de computação de DNA chamado de *sticker model*. Assim como os modelos de filtragem, este modelo emprega cadeias de DNA como o substrato físico para armazenar e processar informação. O modelo de *stickers* possui uma *memória de acesso aleatório* que não requer a extensão de cadeias de DNA, não utiliza enzimas, e (em teoria) utiliza material reaproveitável.

- *splicing systems* ou *sistemas H*: De forma simples, cortar (*splice*) duas *strings* corresponde a parti-las em pontos específicos e concatenar os fragmentos obtidos de uma forma similar à feita com cromossomos durante um *crossover*. Este modelo é baseado em linguagens formais e a operação de *splicing*.
- *insertion/deletion systems*: As operações de inserção e deleção são fundamentais em linguagens formais. Dado um par de palavras (*x*,*y*), denominado *contexto*, um operador de inserção permite inserir uma palavra *v* entre *x* e *y*.
- *modelo PAM (parallel associative memory model)*: Este modelo basicamente descreve um operador de ligação paralela e associativa. Ele também utiliza operadores comuns em computação de DNA, como união, extração e deleção.

5. Computadores Universais de DNA

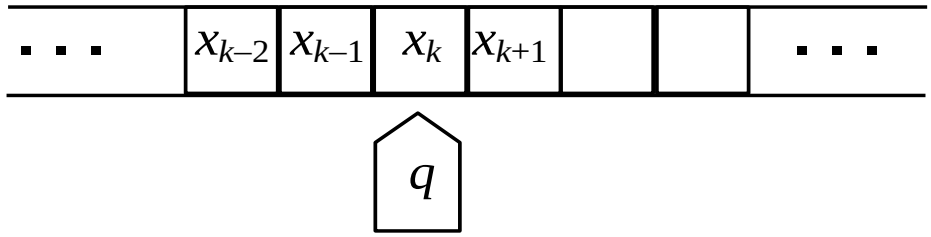
- A verificação da capacidade de computação universal de um computador de DNA tem sido feita de várias formas utilizando diferentes computadores universais, como máquinas de Turing, autômatos celulares, circuitos booleanos e gramáticas de Chomsky.
- Como uma máquina de Turing universal pode, em tese, computar qualquer função computável, projetar uma máquina de Turing universal utilizando DNA constitui um passo importante na direção de provar a universalidade da computação de DNA.
- Neste caso, é preciso especificar um computador molecular capaz de manter um estado e uma memória, e de executar uma quantidade indefinida de transições de estados.

- D. Beaver projetou uma máquina de Turing consistindo de uma única molécula de DNA, na qual os mecanismos químicos para a transição de estados permitem uma computação paralela, sincronizada e heterogênea.
- A cada passo do modelo proposto, uma molécula de DNA codifica uma configuração da máquina de Turing: o conteúdo da fita, seu estado atual e a posição do cabeçote.
- Cada transição de estado requer um esforço $O(1)$ em termos de passos de laboratório a serem executados.
- Como uma única cadeia de DNA é utilizada para codificar a configuração de uma máquina de Turing, Beaver primeiramente mostrou como implementar uma *substituição dependente de contexto* em uma molécula de DNA.
    - Isso foi feito porque simular uma computação (passo) de uma MT corresponde a substituir uma parte da configuração de uma MT.

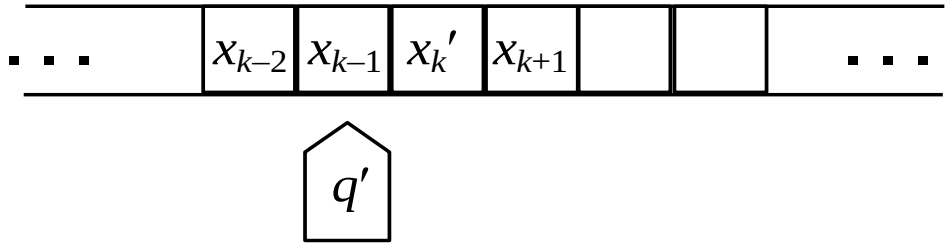
- A idéia é substituir a porção de DNA que irá sofrer a transição de estado. (ver figura)
- Configuração: $C = (x_1 \ldots x_{k-1} q x_k x_{k+1} \ldots x_m)$
- Codificação: $C^e = e(x_1,1) \ldots e(x_{k-1},k-1) e(q,k) e(x_k,k) \ldots e(x_m,m)$, onde $e(x_k,k)$ indica que o símbolo $x_k \in \sum$ está na k-ésima posição da fita e $e(q,k)$ indica o estado atual da máquina.
- Note que, neste esquema, o conteúdo da máquina de Turing, juntamente com o estado q atual da máquina são codificados um a um e concatenados para formar uma cadeia de DNA que codifica a configuração da máquina.
- Para que ocorra uma transição, a molécula é isolada em um tubo de ensaio de acordo com o estado q, posição do cabeçote k, símbolo x_k sendo lido atualmente e os símbolos x_{k-1} à esquerda e x_{k+1} à direita do cabeçote.
- Os valores de q e x_k determinam a transição de estados a ser realizada:
  - $\delta(q,x_k) = (q',x_k',L)$ movimento para a esquerda

- $\delta(q,x_k) = (q',x_k',R)$ movimento para a direita

- Beaver (1995) também estendeu este método para simular máquinas de Turing não-determinísticas.



$$C = (x_{k-1}qx_kx_{k+1});\ \ e(x_{k-1},k-1)e(q,k)e(x_k,k)$$



$$C = (x_{k-2}q'x_{k-1}x_k');\ \ e(q',k-1)e(x_{k-1},k-1)e(x_k',k)$$

Figura 13: Codificação de uma configuração da máquina de Turing.

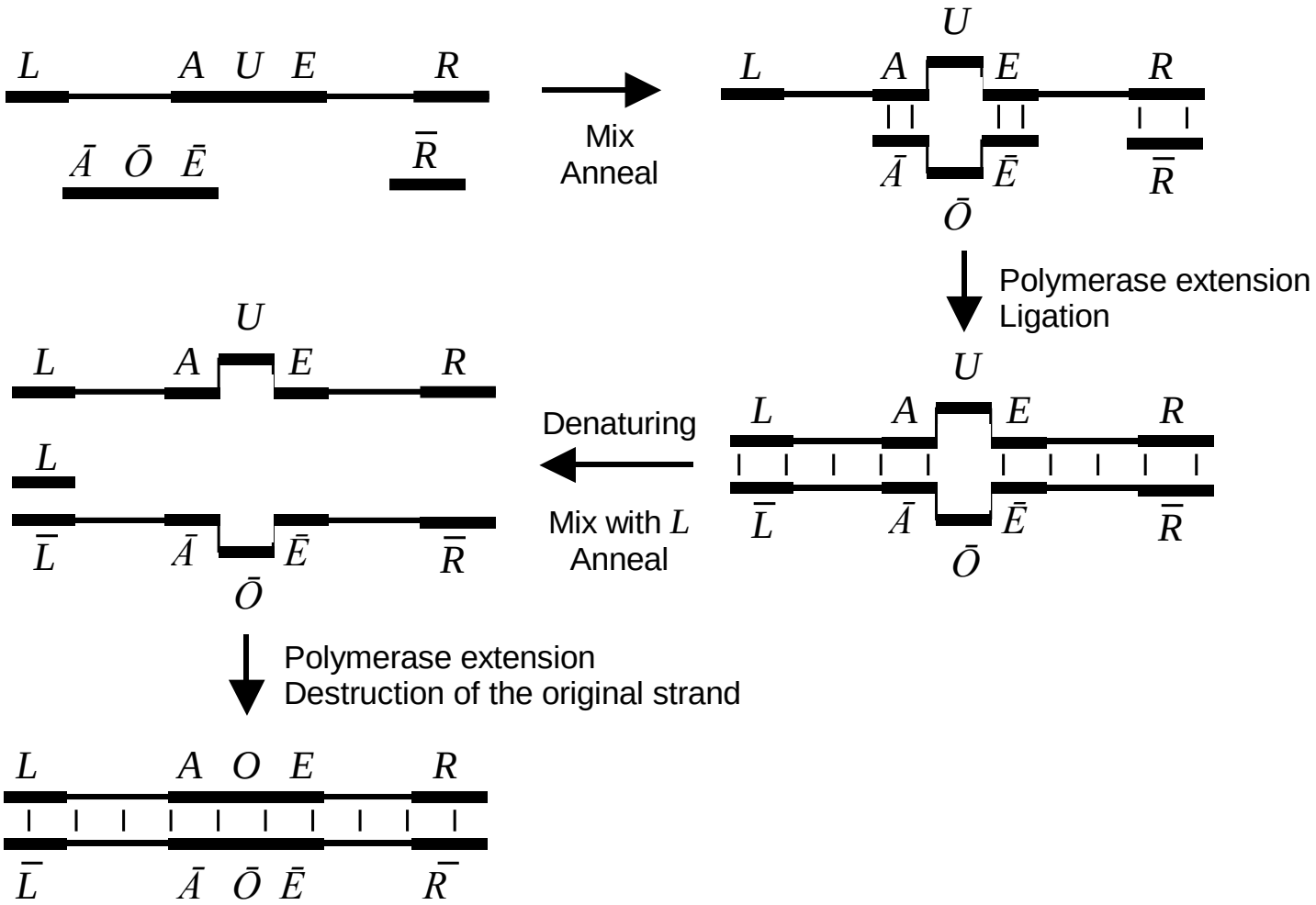


Figura 14: Substituição molecular de uma sequência de DNA empregada como transição de estados de uma "máquina de Turing de DNA". Exemplo: substituir *U* por *O*.

6. Escopo da Computação de DNA

- A computação de DNA foi inicialmente proposta para resolver problemas, sendo bem-sucedida na solução de problemas NP-completos.
- Como qualquer instância de um problema NP-completo pode ser expressa em termos de um outro problema NP-completo, as soluções baseadas em computação de DNA apresentadas fornecem implicitamente um poder computacional suficiente para resolver qualquer problema desta classe.
- Exemplos de outros problemas que podem ser resolvidos por DNA:
  - Graph coloring
  - Shortest common superstring
  - Integer factorization
  - Protein conformation
  - Maximum clique
  - E muitos outros.

- Também são encontrados trabalhos na literatura aplicando a ideia à geração de memórias associativas, à solução de problemas criptográficos, ao desenvolvimento de algoritmos (baseados em DNA) para adição de números e multiplicação de matrizes, ao projeto de máquinas paralelas, etc.
- O paralelismo massivo e a miniaturização do DNA sugerem uma vasta gama de problemas que são candidatos em potencial a serem resolvidos pela computação de DNA.
- Além destas aplicações computacionais da computação de DNA, ela também pode ser aplicada no domínio da biologia. Exemplos:
  - Processamento de DNA: sequenciamento e *fingerprinting*
  - Decodificação de material genético
  - Criação e busca em bases de dados de DNA
  - Detecção de mutações
- Possíveis resultados desta pesquisa em problemas da biologia:

- Erradicação de doenças
- Identificação de criminosos
- Desenvolvimento de biochips implantáveis

7. Discussão

- O experimento de Adleman foi rapidamente seguido por uma grande quantidade de generalizações e extensões para a solução de outros problemas NP-completos.
- Entretanto, é interessante notar que boa parte dos autores não implementaram a computação de DNA em laboratório, como feito por Adleman.
- Na verdade, boa parte das propostas de solução baseadas em DNA são de cunho teórico, uma atividade denominada de *menmology* (mental molecular biology) por Rozenberg.
- As questões em aberto sobre a computação de DNA não mais dizem respeito ao seu poder de processamento.

  - Em vez disso, a principal questão que permanece diz respeito ao projeto e construção de um computador de DNA.
  - Neste sentido são dois os problemas centrais: *correção de erros* e *realização e automação das técnicas de manipulação de DNA*.

- Um dos grandes problemas da computação de DNA é que erros são muito comuns em reações e processos biológicos. As operações de extração, annealing, merge e muitas outras são imprecisas.
- Tem sido grande o esforço no sentido de aproveitar conceitos de matemática e biologia molecular para projetar computadores de DNA.
- No estado atual, a computação molecular possui diversos desafios:
  - O material utilizado (DNA, RNA ou proteínas) não é reutilizável
  - Os componentes moleculares são especializados
  - Correção de erros

- Para que um computador de DNA seja eficiente, o algoritmo a ser utilizado deve ser o mais paralelizável possível
- A interface de entrada/saída é um tanto complicada
- O tempo experimental ainda é grande, mas isso pode ser remediado com um aprofundamento dos conhecimentos e tecnologias em biologia molecular e engenharia genética

- Entretanto, algumas características da computação de DNA servem para contrabalançar as dificuldades de projeto de um computador de DNA:
  - Alta velocidade de processamento paralelo quando automatizado (permite o uso da força bruta)
  - É economicamente barato sob o ponto de vista de consumo de energia, armazenagem e processamento de informação

8. Bibliografia

ADLEMAN, L.M. (1994) "Molecular Computation of Solutions to Combinatorial Problems", *Science*, vol. 226, November, pp. 1021-1024.

ADLEMAN, L.M. (1998) "Computing with DNA", Scientific American, vol. 279, no. 2, pp. 34-41.

AMOS, M. (2003), "Theoretical and Experimental DNA Computation", Springer-Verlag.

BAUMGARDNER, J., ACKER, K., ADEFUYE, O., CROWLEY, S.T., DELOACHE, W., DICKSON, J.O., HEARD, L., MARTENS, A.T., MORTON, N., RITTER, M., SHOECRAFT, A., TREECE, J., UNZICKER, M., VALENCIA, A., WATERS, M., CAMPBELL, A.M., HEYER, L.J., POET, J.L. & ECKDAHL, T.T. (2009) "Solving a Hamiltonian Path Problem with a bacterial computer", Journal of Biological Engineering, vol. 3, pp. 1-11.

BEAVER, D. (1995), "Molecular Computing", Technical Report TR 95-001, Penn. State University, Pennsylvania, USA, January.

BENENSON, Y., PAZ-ELIZUR, T., ADAR, R., KEINAN, E., LIVNEH, Z. & SHAPIRO E. (2001) "Programmable and autonomous computing machine made of biomolecules", Nature, vol. 414, no. 1, pp. 430-434.

BONEH, D., DUNWORTH, C., LIPTON, R.J. & SGALL, J. (1996) "On the computational power of DNA", Discrete Applied Mathematics, vol. 71, nos. 1-3, pp. 79–94.

CALUDE, C. S. & PĂUN, G. (2001), "Computing with Cells and Atoms", Taylor & Francis.

FRANCO, G. (2006) "Biomolecular computing – Combinatorial algorithms and laboratory experiments", Ph.D. Thesis, University of Verona, Italy.

FRANCO, G. & MARGENSTERN, M. (2008) "A DNA computing inspired computational model", Theoretical Computer Science, vol. 404, nos. 1-2, pp. 88-96.

GRAMß, T., BORNHOLDT, S., GROß, M., MITCHELL, M. & PELLIZZARI, T. (2001) "Non-Standard Computation: Molecular Computation – Cellular Automata – Evolutionary Algorithms – Quantum Computers", Wiley-VCH.

LIPTON, R. (1995) "DNA solution of hard computational problems", Science, vol. 268, pp. 542-545.

PĂUN, G. (2002) "Membrane Computing – An Introduction", Springer-Verlag.

PĂUN, G., ROZENBERG, G. AND SALOMA, A. (1998) "DNA Computing: New Computing Paradigms", Springer-Verlag.

PISANTI, N. (1998) "DNA Computing: A Survey", *Bulletin of the European Association for Theoretical Computer Science*, **64**, pp. 188–216.

ROWEIS, S., WINFREE, E., BURGOYNE, R., CHELYAPOV, N. GOODMAN, M., ROTHEMUND, P. AND ADLEMAN, L. (1996) "A Sticker Based Model for DNA Computation", In E. Baum, D. Boneh, P. Kaplan, R. Lipton, J. Reif and N. Seeman (eds.), *DNA Based Computers*, Proc. of the 2nd Annual Meeting, pp.1-27.

SHAPIRO, E. & BENENSON, Y. (2006) "Bringing DNA Computers to Life", Scientific American, vol. 294, no. 5, pp. 45-51.

SHAPIRO E. & RAN T. (2013) "DNA computing: Molecules reach consensus", Nature Nanotechnology, vol. 8, pp. 703-705.