

Computação e Programação

Projecto de Biocomputação

Departamento de Matemática, IST

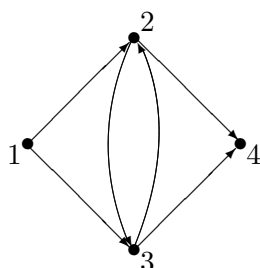
Outubro de 2007

A experiência de Adleman

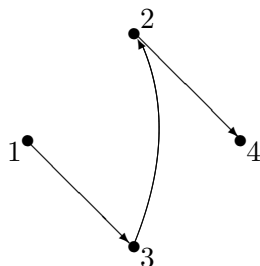
Em 1994, Leonard Adleman surpreendeu a comunidade científica quando publicou (Molecular computation of solutions to combinatorial problems, *Science* 226, 1021-1024, 1994) os resultados da experiência que veio a ficar conhecida com o seu nome.

Adleman resolveu com técnicas de biologia molecular o **problema HPP** (*Hamiltonian Path Problem*) a seguir descrito. Dado um grafo orientado e fixado um vértice origem e um vértice destino, pretende-se encontrar, caso exista, um caminho Hamiltoniano com essa origem e esse destino. Por caminho Hamiltoniano entende-se um caminho que passa por todos os vértices e em cada vértice uma e uma só vez.

Para ilustração, considere-se o grafo:



Neste grafo, existem dois caminhos Hamiltonianos. Por exemplo,



Este problema é difícil de resolver com meios computacionais convencionais, pois envolve a pesquisa exaustiva de todas as possíveis soluções. A solução proposta por Adleman permite tirar partido de um elevado nível de paralelismo para chegar à solução mais rapidamente. A ideia geral é simples - representam-se os vértices e as arestas por sequências de nucleótidos de tal maneira que a hibridação leve à concatenação de arestas contíguas no grafo.

Recorde-se que o DNA é constituído: A (*adenina*), C (*citocina*), T (*timina*) e G (*guanina*). Recorde-se também que A e T são complementares no sentido em que na hibridação eles se emparelham através de duas ligações de hidrogénio e que C e G são complementares com emparelhamento através de três ligações de hidrogénio.

Adleman codificou cada vértice por uma sequência de nucleótidos de comprimento par. Por exemplo, sejam os vértice u , v e w representados pelas sequências U_1U_2 , V_1V_2 e W_1W_2 , respectivamente, em que U_1 , U_2 , V_1 , V_2 , W_1 e W_2 são sequências de comprimento dez. Então, as arestas $u \rightarrow v$ e $v \rightarrow w$ deverão ser representadas pelas sequências $\overline{U_2V_1}$ e $\overline{V_2W_1}$, respectivamente. Aqui, dada uma sequência Z de nucleótidos, \overline{Z} é a sequência dos seus complementares. Por exemplo, se $Z = \text{ACATGG}$, então $\overline{Z} = \text{TGTACC}$.

Numa *sopa biomolecular* em que estejam presentes as sequências V_1V_2 , $\overline{U_2V_1}$ e $\overline{V_2W_1}$, por hibridação, acabará por se construir a trança aberta

$$\frac{V_1 V_2}{U_2 \overline{V_1} \overline{V_2} \overline{W_1}}.$$

Ou seja, surgirá o caminho $u \rightarrow v \rightarrow w$ por concatenação dos caminhos $u \rightarrow v$ e $v \rightarrow w$.

Dados um grafo orientado e fixado o vértice partida e o vértice de chegada, uma vez estabelecida esta codificação das cidades e das arestas, a experiência de Adleman procede do modo seguinte:

Passo 0: Preparação

Constitui-se uma sopa biomolecular num tubo de ensaio com grande quantidades de sequências de nucleótidos representando os vértices e as arestas do grafo orientado. Adleman usou 10^{13} cópias de cada vértice e de cada aresta.

Passo 1: Hibridação

Deixa-se o mecanismo de hibridação actuar, o que conduz à constituição de muitas tranças representando caminhos possíveis no grafo orientado em questão. É neste passo que o método de Adleman introduz elevados níveis de paralelismo pois as tranças vão-se constituindo ao mesmo tempo.

Passo 2: Selecção dos caminhos com origem e destino pretendidos

Recorrendo à técnica *PCR* (de *Polymerase Chain Reaction*), eliminam-se as tranças que não têm a origem e o destino pretendidos.

Passo 3: Selecção dos caminhos com o comprimento pretendido

Recorrendo à técnica *GE* (de *Gel Electrophoresis*), eliminam-se as tranças que não têm comprimento correspondente ao número de vértices do grafo orientado em causa.

Passo 4: Selecção dos caminhos que passam em todos os vértices

Para cada vértice, recorrendo a uma técnica de *purificação por afinidade*, eliminam-se as tranças que não passam por esse vértice.

Passo 5: Observação do resultado

Procede-se à sequenciação das tranças de DNA que tenham sobrevivido aos passos anteriores. Qualquer delas representa um caminho Hamiltoniano no grafo orientado dado com a origem e destino pretendidos.

Para mais detalhes sobre a experiência de Adleman, recomenda-se a visita à página <http://arstechnica.com/reviews/2q00/dna/dna-1.html> e a leitura de um texto especializado, por exemplo, G. Paun, G. Rozenberg e A. Salomaa, *DNA Computing*, Springer, 1998.

Simulação da experiência de Adleman

O objectivo do projecto é desenvolver em F um programa que o resolva o *HPP* (*Hamiltonian Path Problem* - Problema do caminho Hamiltoniano) através de uma simulação abstracta da experiência de Adleman, representando a hibridação por simples concatenação de caminhos, sem entrar nos detalhes da codificação em sequências de nucleótidos.

Mais concretamente, a simulação deve consistir em:

Passo 0: Preparação

Distribui-se de modo aleatório uniforme no cubo $[0, 1]^3$ as κ cópias das arestas do grafo orientado γ dado. Para o efeito recorre-se à subrotina `random_number` da linguagem F que tem um parâmetro de saída no qual é devolvida uma observação de um número pseudo-aleatório.

Passo 1: Hibridação

Procede-se a uma simulação sequencial muito abstracta do processo de hibridação repetindo ν vezes a concatenação dos dois caminhos compatíveis mais próximos em $[0, 1]^3$.

Passo 2: Selecção dos caminhos com origem e destino pretendidos

Eliminam-se os caminhos que não têm a origem σ e o destino τ pretendidos.

Passo 3: Selecção dos caminhos com o comprimento pretendido

Eliminam-se os caminhos que não têm comprimento igual ao número de vértices de γ .

Passo 4: Selecção dos caminhos que passam em todos os vértices

Para cada vértice de γ , eliminam-se os caminhos que não passam por esse vértice.

Passo 5: Observação do resultado

Apresentam-se os caminhos que tenham sobrevivido aos passos anteriores. Qualquer deles é um caminho Hamiltoniano no grafo orientado γ com origem σ e destino τ .

Desenvolva o programa de simulação seguindo o *método de programação modular por camadas centradas nos dados*:

- a) Comece por identificar os objectos de trabalho, nomeadamente caminhos, posições, indivíduos e sopas biomoleculares.
- b) Desenvolva de seguida o programa abstracto pretendido sobre a camada que disponibiliza estes objectos.
- c) Implemente esta camada sobre a camada básica da linguagem F.
- d) Integre o programa obtido em (b) com os módulos desenvolvidos em (c) para obter o programa final.
- e) Experimente o programa com diversos conjuntos de dados.

Dados fornecidos

O programa deve receber interactivamente os dados seguintes:

- γ – grafo orientado representado por um vector (η) contendo a lista de nós do grafo, representados por números inteiros, e uma matriz (α) $n \times 2$, sendo n é o número de arestas, em que cada linha contém um par de vértices correspondendo a uma aresta;
- σ – vértice origem do caminho Hamiltoniano pretendido;
- τ – vértice destino;

- κ – número de cópias a utilizar (muito inferior ao número adoptado por Adleman - 10^{13});
- ν – número de concatenações de caminhos.

Comece por experimentar o programa com os seguintes conjuntos de dados:

Conjunto 1 (grafo considerado acima)

- $\eta = [1, 2, 3, 4]$;

$$\bullet \alpha = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 2 & 3 \\ 2 & 4 \\ 3 & 2 \\ 3 & 4 \end{bmatrix}$$

- $\sigma = 1$;
- $\tau = 4$;
- $\kappa = 20$;
- $\nu = 200$.

Conjunto 2 (grafo considerado por Adlman)

- $\eta = [0, 1, 2, 3, 4, 5, 6]$;

$$\bullet \alpha = \begin{bmatrix} 0 & 1 \\ 0 & 3 \\ 0 & 6 \\ 1 & 2 \\ 1 & 3 \\ 2 & 1 \\ 2 & 3 \\ 3 & 2 \\ 3 & 4 \\ 4 & 1 \\ 4 & 5 \\ 5 & 2 \\ 5 & 6 \end{bmatrix}$$

- $\sigma = 0$;
- $\tau = 6$;
- $\kappa = 30$;
- $\nu = 500$.

Resultados pretendidos

O programa deve apresentar o conjunto dos caminhos Hamiltonianos encontrados.

Por solicitação do utilizador, o programa deverá permitir fornecer também resultados intermédios de cada passo.

Devem ser apresentados os resultados obtidos com os dois conjuntos de dados obrigatórios e ainda com, pelo menos, um terceiro conjunto de dados de escolha livre.

Apêndice A

Serão disponibilizados os módulos seguintes:

- **Módulo `mpath`:** disponibiliza o tipo de dados `path` e as seguintes funções e subrotinas sobre este tipo de dados:
 - `make_path(v1,v2)`: função que recebe nos parâmetros de entrada `v1` e `v2` de tipo `integer` dois vértices do grafo γ e devolve o caminho (de tipo `path`) constituído por estes dois vértices, que começa em `v1` e termina em `v2`.
 - `first(w)`: função que recebe no parâmetro `w` de tipo `path` um caminho e devolve o primeiro vértice desse caminho.
 - `last(w)`: função que recebe no parâmetro `w` de tipo `path` um caminho e devolve o último vértice desse caminho.
 - `path_length(w)`: função que recebe no parâmetro `w` de tipo `path` um caminho e o comprimento desse caminho¹.
 - `compQ(w1,w2)`: função que recebe nos parâmetros de entrada `w1` e `w2` de tipo `path` dois caminhos e devolve `.true.` se esses caminhos forem compatíveis e `.false.` caso contrário. Dois caminhos `w1` e `w2` dizem-se *compatíveis* se o último vértice de `w1` for igual ao primeiro vértice de `w2`.
 - `glue(w1,w2,w3)`: subrotina que recebe nos parâmetros de entrada/saída `w1` e `w2` de tipo `path` dois caminhos e devolve no parâmetro de saída `w3` também de tipo `path` o resultado de *colar* `w1` com `w2`, caso estes caminhos sejam compatíveis. Note que o caminho resultante não inclui a repetição do vértice comum, i.e., se `w1` for o caminho $1 \rightarrow 3 \rightarrow 2$ e `w2` for o caminho $2 \rightarrow 4$ o resultado de colar estes dois caminhos (compatíveis) é o caminho $1 \rightarrow 3 \rightarrow 2 \rightarrow 4$.
 - `crossesQ(w,v,b)`: subrotina que recebe no parâmetro de entrada/saída `w` de tipo `path` um caminho e no parâmetro de entrada `v` de tipo `integer` um vértice e devolve no parâmetro de saída `b` de tipo `logical` o valor `.true.` se o caminho `w` passa pelo vértice `v` e `.false.` caso contrário.
 - `delete(w)`: subrotina que recebe no parâmetro de entrada/saída `w` de tipo `path` um caminho e o apaga.
 - `path_show(w)`: subrotina que recebe no parâmetro de entrada/saída `w` de tipo `path` um caminho e imprime a lista de vértices por onde esse caminho passa.
- **Módulo `mind`:** disponibiliza o tipo de dados `ind` e as seguintes funções e subrotinas sobre este tipo de dados:

¹Por comprimento de um caminho entende-se o número de vértices por onde esse caminho passa.

- `make_ind(w,x,y,z,i)`: subrotina que recebe nos parâmetros de entrada `w` de tipo `path` e `x`, `y`, e `z` de tipo `real`, respectivamente, um caminho e três coordenadas reais devolve no parâmetro de saída `i` de tipo `ind` um indivíduo com estas características.
 - `ipath(i,w)`: subrotina que recebe no parâmetro de entrada `i` de tipo `ind` um indivíduo e devolve no parâmetro de saída `w` de tipo `path` o caminho desse indivíduo.
 - `xpos(i)`: função que recebe no parâmetro de entrada `i` de tipo `ind` um indivíduo e devolve a coordenada `x` desse indivíduo.
 - `ypos(i)`: função que recebe no parâmetro de entrada `i` de tipo `ind` um indivíduo e devolve a coordenada `y` desse indivíduo.
 - `zpos(i)`: função que recebe no parâmetro de entrada `i` de tipo `ind` um indivíduo e devolve a coordenada `z` desse indivíduo.
 - `dist(i1,i2)`: função que recebe nos parâmetros `i1` e `i2` de tipo `ind` dois indivíduos e devolve a distância, em $[0, 1]^3$, entre as posições ocupadas por eles.
 - `ind_eq(i1,i2)`: função que recebe nos parâmetros `i1` e `i2` de tipo `ind` dois indivíduos e devolve `.true.` se estes forem iguais e `.false.` caso contrário.
- **Módulo `msoup`**: disponibiliza o tipo de dados `soup` e as seguintes funções e subrotinas sobre este tipo de dados:
 - `new_soup()`: função sem parâmetros que devolve a sopa biológica vazia.
 - `insert(i,s)`: subrotina que recebe no parâmetro de entrada `i` de tipo `ind` um indivíduo e no parâmetro de entrada/saída `s` de tipo `soup` uma sopa biológica e devolve em `s` a sopa com o novo indivíduo inserido.
 - `next(s,i1,i2,b)`: subrotina que recebe no parâmetro de entrada/saída `s` de tipo `soup` uma sopa biológica e devolve nos parâmetros de saída `i1` e `i2` de tipo `ind` os próximo indivíduos que se vão recombinar, isto é, os indivíduos com caminhos compatíveis que se encontram mais próximos, caso existam. Devolve ainda no parâmetro de saída `b` de tipo `logical` o valor `.true.` caso existam indivíduos nestas condições e `.false.` caso contrário.
 - `take(i,s)`: subrotina que recebe no parâmetro de entrada `i` de tipo `ind` um indivíduo e no parâmetro de entrada/saída `s` de tipo `soup` uma sopa biológica e devolve em `s` a sopa sem o indivíduo `i`;
 - `an_ind(s,i)`: subrotina que recebe no parâmetro de entrada/saída `s` de tipo `soup` uma sopa biológica e devolve no parâmetro de saída `i` de tipo `ind` um indivíduo dessa sopa.
 - `emptyQ(s)`: função que recebe no parâmetro `s` de tipo `soup` uma sopa biológica e devolve `.true.` se essa sopa estiver vazia, e `.false.` caso contrário.
 - `soup_size(s)`: função que recebe no parâmetro `s` de tipo `soup` uma sopa biológica e devolve o seu tamanho (número de indivíduos).
 - `soup_show(s)`: subrotina que recebe no parâmetro de entrada/saída `s` de tipo `soup` uma sopa biológica e imprime a lista dos seus indivíduos.