Solucionando el problema de la mochila en el Modelo Sticker

Ernesto Mancebo

Febrero 2020

Abstract

Este documento ***

1 Introducción

2 Modelo Sticker

El Modelo Sticker es un modelo de computación inpirado en cadenas de ADN propuesto por Sam Roweis **citation-13218946**, donde tal modelo se basa en procedimientos de filtrado y con memoria de acceso aleatorio. En ese mismo orden, se distingue de otros modelos computacionales por la manera en que representa la información.

2.1 Conceptop de Cadena de Memoria

El eje central de este modelo son las cadenas de memoria, tales cadenas consisten en hebras simples de ADN configuradas de forma (n, k, m), tal que $n \geq k.m$, siendo

n la cantidad de sub-cadenas que soportra, k las sub-cadenas dentro de n, y m la longitud de cada sub-cadena p. Las cadenas de memoria también se les conocen como moléculas y son representadas por sigma (σ).

En ese orden, las operaciones realizadas sobre cada hebra o complejo de memoria asocia un sticker a una región k, transformando la cadena en una hebra doble *parcial*, tal que en cada región donde haya un sticker complementándole se dice que está *encendido*, mientras que la región donde se carezca de sticker se dice está *apagado*. Tal abstracción binaria también se puede expresar de modo *0/1*.

2.2 Codificación de la Información

2.3 Concepto de Tubo

Dentro de este modelo, conocemos un tubo como un multiconjunto de complejos de memorias del mismo tipo. Tal concepto se puede ilustrar del modo: $\begin{bmatrix} \sigma_0 \\ \sigma_i \\ \sigma_{i+1} \end{bmatrix}$

En ese orden de ideas, las operaciones a principales dentro de este modelo son:

- *mezclar*(T_1, T_2): Retorna la unión de los dos tubos devolviendo un solo tubo el cual contiene cada complejo de memoria proveniente de ambos. También se nota como $(T_1 \bigcup T_2)$.
- *separar*(T, i): Para un tubo T y un i tal que $(1 \leq i \leq n)$, siendo n la cantidad de subcadenas para cada complejo de memoria, retorna un $+(T, i)$ y $-(T, i)$, de modo que el primer tubo contiene todos los complejos de memoria cuya región i esté encendida (*on* o 1), mientras que la segunda lo contrario (*off* o 0).
- *encender*(T, i): Para un tubo T y un i tal que $(1 \leq i \leq n)$, siendo n la cantidad de subcadenas para cada complejo de memoria, enciende la región i de cada complejo de memoria asignando un 1 u *on*.
- *apagar*(T, i): Opuesto a *encender*, asigna un 0 u *off*.

- *leer*(T): Dado un tubo $T \neq \emptyset$, lee su contenido.

2.4 Concepto de Librería

Una librería dentro de este modelo es lo conocido como complejos de memoria de modo (k, l) tal que, teniendo k sub cadenas/regiones y las primeras $k - l$ subcadenas están *on* u *off*, en todas sus posibles combinaciones.

3 Sub-Rutina Ordenado por Cardinalidad

Teniendo los siguientes elementos:

- $A = \{1, \cdots, p\}$
- $B = \{b_1, \cdots, b_s\} \subseteq A$
- $F = \{D_1, \cdots, D_t\} \subseteq P(A)$

Se busca ordenar F con respecto a su cardinalidad en B, o en otras palabras, la cantidad de elementos que conincidan de manera $B \cap D_i$.

Una vez teniendo claro el domino del problema planteado, el paper bajo estudio nos sugiere el implementar la codificación de cada familia de subconjuntos F, de forma que la familia mencionada sea una molécula σ para el tubo T_0, codificando cada molécula de forma $T_0 = \{\{\sigma : |\sigma| = p \wedge \exists j(\chi_{D_j} = \sigma)\}\}$; tal que χ_{D_j} es la función característica en A, siendo $(\chi_{D_j}(i) = 1$ si $i \in D_j$ de lo contrario $\chi_{D_j} = 0$ si $i \in A - D_j)$.

Ya con las restrinciones definidas, se plantea en el paper una subrutina que en el paso i se generen $i + 1$ tubos verificando la condición $\forall\sigma(\sigma \in T_j \rightarrow |\sigma \in \{b_1, \cdots, b_i\}| = j)$, leyéndose esta condición: sólo existirán moléculas σ en T_j cuya cantidad de regiones encendidas sean igual a j.

3.1 Algoritmo

En ese sentido, el paper ** sugiere el siguiente algoritmo:

Algorithm 1 Ordena los elementos en T_0 con respecto a los elementos de B presentes en cada σ

procedure $Cardinal\_Sort(T_0, B)$
  for $i = 1$ to s **do**
    $(T_0, T_0') = separar(T_0, b_i)$
    for $j = 0$ to $i - 1$ **do**
      $(T_{j+1}', T_j'') = separar(T_j, b_i)$
      $T_j = mezclar(T_j', T_j'')$
    end for
    $T_i = T_i'$
  end for
  Return $[T_0, ..., T_s]$
end procedure

3.2 Traza Sub-Rutina Ordenado por Cardinalidad

A modo de ilustrar el comportamiendo y los conceptos empleados en el algoritmo concebido, tenemos:

- $A : \{0, 1, 2, 3, 4, 5, 6\}$
- $B : \{1, 2, 4\}$
- $F : \{[2, 6], [3], [4], [2, 4]\}$

Observemos que los elementos que cumplen $B \cap D_j$ en F son: $\{[2, 6], [3], [4], [2, 4]\}$, cuyos elementos resaltados nos sugieren en qué posición del tubo de salida tras ser evaluados por *Cardinal_Sort*. Codificando F para llevarlo a un tubo tendíamos:

$$\begin{bmatrix} [0, 0, 1, 0, 0, 0, 1] \\ [0, 0, 0, 1, 0, 0, 0] \\ [0, 0, 0, 0, 1, 0, 0] \\ [0, 0, 1, 0, 1, 0, 0] \end{bmatrix}$$

Este T_0 codificando F tras la ejecución de *Cardinal_Sort* tendremos:

$$\begin{bmatrix} T_0 & : [[0,0,0,1,0,0,0]] \\ T_1 & : [[0,0,0,0,1,0,0],[0,0,1,0,0,0,1]] \\ T_2 & : [[0,0,1,0,1,0,0]] \\ T_3 & : [] \\ T_4 & : [] \\ T_5 & : [] \\ T_6 & : [] \end{bmatrix}$$

Cumpliendo de esta manera la condición de $\forall\sigma(\sigma \in T_j \rightarrow |\sigma \in \{b_1, \cdots, b_i\}| = j)$.

3.3 Asunciones

Para usos posteriores dentro de este informe, podemos notar la llamada a *Cardinal_Sort* de modo:

- $Cardinal\_Sort(T_0)$ cuando $B = A$.
- $Cardinal\_Sort(T_0, l, k)$ cuando $B = \{l, l+1, \cdots, k\}$.

4 Sub-Rutina de Llenado

Una vez pensado en el concepto de codificar en F las moléculas σ tal que $B \cap D_i$, podemos considerar la idea de segmentar cada molécula σ en T_0 en posibles regiones como:

- $(A\sigma) = \sigma(1) \cdots \sigma(p)$
- $(L\sigma) = \sigma(p+1) \cdots \sigma(p+r)$
- $(F\sigma) = \sigma(p+r+1) \cdots \sigma(p+r+q_f)$
- $(R\sigma) = \sigma(p+r+q_f+1) \cdots$

El propósito de la sub-rutina de llenado es manipular para el tubo T_0 las moléculas en $(\sigma(i))$ y codifiquen su valor con respecto a f en el sub-conjunto A, ambos se definen como:

- $A = \{1, \cdots, p\}$
- $r \in \mathbb{N}$
- $f : A \to \mathbb{N}$

Para este escenario denotamos que si $B \subseteq A$, decimos que $f(B) = \sum_{i \in B} f(i)$; o en otras palabras, $f(i)$ aplicada a un subconjunto B, retorna la suma de todos los elementos $\sum_0^i B$, cuya suma para $\sum_0^s B$ indica el espacio a reservar en el complejo de memoria T_0 para codificar B. Por otra parte, definimos que $q_f = f(A)$, tal que $A_i = \{0, \cdots, i\}$ siendo $(i \leq i \leq p)$. En ese mismo orden, r corresponde a la región comprendida entre p y $f(B)$ la cual está reservada para codificar algún otro atributo dentro del complejo de memoria para la molécula σ . Finalmente, calificamos T_0 un multiconjunto de forma (n, k, m) de complejos de memorías σ cuyo $k \geq p + r + q_f$.

4.1 Algoritmo

Algorithm 2 Asigna valores para cada complejo de memoria σ a partir de la región especificada en p y r

1: **procedure** $Parallel\_Fill(T_0, f, p, r)$
2: **for** $i = 1$ to p **do**
3: $(T^+_{i,0}, T^-_i) = separar(T_{i-1}, i)$
4: **for** $j = 1$ to $f(i)$ **do**
5: $T^+_{i,j} = encender(T^+_{i,j}, p + r + f(A_{i-1}) + j)$
6: **end for**
7: $T_i = mezclar(T^+_{i,f(i)}, T^-_i)$
8: **end for**
9: Return T_0
10: **end procedure**

Para cada $i(1 \leq i \leq p)$ se consiredan las regiones:
$R_i = \{p + r + f(A_{i-1}) + 1, \cdots, p + r + f(A_i)\}$.

4.2 Traza Sub-Rutina de Llenado

Con tal de ilustrar la sub-rutina bajo estudio, utilicemos como entrada:

- $A : \{1, 2, 3, 4\}$
- $B : \{2, 3, 4\}$
- $T_0 : \{[2], [4], [3]\}$

Una vez codificado el tubo T_0, tenemos:

$$\begin{bmatrix} [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0] \\ [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0] \\ [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0] \end{bmatrix}$$

Nótese que las primeras cuatro regiones resaltadas en rojo corresponden al tamaño p, reservado para codificar el valor de cada B. Una vez ilustrado el tubo T_0, una vez procesado por $Parallel\_Fill$, tendremos:

$$\begin{bmatrix} [0,1,0,0,1,1,1,0,0,0,0,0,0,0,0] \\ [0,0,0,1,1,1,1,1,1,1,1,1,1,1,1] \\ [0,0,1,0,1,1,1,1,1,1,0,0,0,0,0] \end{bmatrix}$$

5 Problema de Suma de Sub-Conjuntos

El problema de la suma de sub-conjuntos busca determinar si existe en B un sub-conjunto cuyo valor equivalga a k. En ese sentido, definimos $A = \{1, \cdots, p\}, k \in \mathbb{N}, w : A \to \mathbb{N}$, siendo w la función peso tal que $k \leq w(A) = q_w$.

5.0.1 Algoritmo

5.0.2 Traza

Tomando como referencia el tubo T_0 del apartado anterior:

$$\begin{bmatrix} [0,1,0,0,1,1,1,0,0,0,0,0,0,0,0] \\ [0,0,0,1,1,1,1,1,1,1,1,1,1,1,1] \\ [0,0,1,0,1,1,1,1,1,1,0,0,0,0,0] \end{bmatrix}$$

Algorithm 3 Retorna los complejos de memoria tal que la suma de sus $R\sigma$ sean igual a k

1: **procedure** $Subset\_Sum(p, w, k)$
2: $\quad q_w = \sum_{i=1}^{p} w(i)$
3: $\quad T_0 = \text{Libería}(p + q_w, p)$
4: $\quad T_1 = Parallel\_Fill(T_0, w, p, 0)$
5: $\quad T_k = Cardinal\_Sort(T_1, p + 1, p + q_w)[k]$
6: $\quad leer(T_k)$
7: **end procedure**

Para un $k = 10$, una vez el T_0 sea procesado por $Subset\_Sum$ tendríamos como salida: $[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, pues si contamos los bits en la región azul nos da un total de 10.

6 Problema de la Mochila Delimitado

El problema de la mochila es un problema considerado NP-completo, en el que se busca recolectar una serie de objetos cuyo peso sea menor que k y valor mayor o igual al k'. En ese sentido, consideramos los valores: $A = \{1, \cdots, p\}$ un conjunto no vacío, $w : A \to \mathbb{N}$ una función que codifica el valor para un A, $\rho : A \to \mathbb{N}$ una función que codifica el valor para un A, asimismo, $k, k' \in \mathbb{N}$, tal que $k \leq w(A) = q_w$ y $k' \leq \rho(A) = q_\rho$.

6.1 Algoritmo

***** definir las líneas

El algoritmo a presentar se apoya en los tres algoritmos propuestos anteriormente, donde se delimitan las regiones a utilizar, se ordenan y agrupan los complejos de memoria a modo de conseguir los que cumplan con las restricciones $w(B) \leq k$ y $\rho(B) \geq k'$ y posteriormente se mueven dichas moléculas en T_1.

Algorithm 4

procedure $Bounded\_Knapsack(p, w, \rho, k, k')$
    $q_w = \sum_{i=1}^{p} w(i); q_\rho = \sum_{i=1}^{p} \rho(i);$
    $T_0 =$Libería$(p + q_w + q_\rho, p)$
    $T_0 = Parallel\_Fill(T_0, w, p, 0)$
    $T_0 = Cardinal\_Sort(T_0, p + 1, p + q_w)$
    $T_1 = \emptyset$
    for $i = 1$ to k **do**
        $T_1 = mezclar(T_1, Cardinal\_Sort(T_0, p + 1, p + q_w)[i])$
    end for
    $T_0 = Parallel\_Fill(T_1, \rho, p, q_w)$
    $Cardinal\_Sort(T_0, p + q_w + 1, p + q_w, q_\rho)$
    $T_1 = \emptyset$
    for $i = k'$ to q_ρ **do**
        $T_1 = merge(T_1, Cardinal\_Sort(T_0, p + q_w + 1, p + q_w + q_\rho)[i])$
    end for
    $leer(T_1)$
end procedure

7 Problema de la Mochila No Delimitado

8 Conclusión

9 Bibliografía

Algorithm 5

1: **procedure** $Unbounded\_Knapsack(p, w, \rho, k, k')$
2: $q_w = \sum_{i=1}^{p} w(i)$;
3: $q_\rho = \sum_{i=1}^{p} \rho(i)$;
4: $T_0 =$Libería$(p + q_w + q_\rho, p)$
5: $T_0 = Parallel\_Fill(T_0, w, p, 0)$
6: $T_0 = Cardinal\_Sort(T_0, p + 1, p + q_w)$
7: $T_1 = \emptyset$
8: **for** $i = 1$ to k **do**
9: $T_1 = mezclar(T_1, Cardinal\_Sort(T_0, p + 1, p + q_w)[i])$
10: **end for**
11: $T_0 = Parallel\_Fill(T_1, \rho, p, q_w)$
12: $i = q_\rho; t = 0$
13: **while** $i \geq 1 \wedge t == 0$ **do**
14: $T' = Cardinal\_Sort(T_0, p + q_w + 1, p + q_w + q_\rho)[i]$
15: **if** $T' \neq \emptyset$ **then**
16: $leer(T')$
17: $t = 1$
18: **else**
19: $i = i - 1$
20: **end if**
21: **end while**
22: **end procedure**