Solucionando el problema de la mochila en el Modelo Sticker

Ernesto Mancebo

Febrero 2020

Abstract

Este artículo es un estudio del paper *Solving Knapsack Problems in a Sticker Based Model* de los profesores Mario Pérez-Jiménez y Fernando Sancho-Caparrini,[1] el cual tiene como objetivo la resolución de los problema *suma de subconjuntos* y *porblema de la mochila*, para éste último en sus versiones acotada y la no acotada, utilizando como apoyo para la resolución de ambos problemas las subrutinas *ordenado por cardinalidad* y *llenado en paralelo*.

[1]Mario J. Pérez-Jiménez and Fernando Sancho-Caparrini. "Solving Knapsack Problems in a Sticker Based Model". In: *DNA Computing*. Ed. by Natasa Jonoska and Nadrian C. Seeman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 161–171. ISBN: 978-3-540-48017-4.

Contents

1 Modelo Sticker **4**

1.1 Concepto de Cadena de Memoria 4

1.2 Concepto de Tubo 5

1.3 Concepto de Librería 6

2 Subrutina Ordenado por Cardinalidad **6**

2.1 Algoritmo 7

2.2 Otras Notaciones 8

2.3 Proposiciones 8

2.3.1 Proposición 1 8

2.3.2 Proposición 2 9

2.4 Corolarios 9

2.4.1 Corolario 1 - Robustez del Algoritmo 9

2.4.2 Corolario 2 - Completitud del Algoritmo 9

2.5 Traza Subrutina Ordenado por Cardinalidad 10

3 Subrutina de Llenado **11**

3.1 Algoritmo 11

3.2 Lemas 12

3.2.1 Lema 1 12

3.3 Proposiciones 13

3.3.1 Proposición 3 13
3.4 Corolarios 13
3.4.1 Corolario 3 13
3.5 Traza Subrutina de Llenado 14

4 Problema de Suma de Subconjuntos 15

4.1 Algoritmo 15
4.2 Diseño 15
4.3 Teoremas 16
4.3.1 Theorema 1 - Robustez 16
4.3.2 Theorema 2 - Completitud 16
4.4 Traza 17

5 Problema de la Mochila Acotado 18

5.1 Algoritmo 19
5.2 Diseño 20

6 Problema de la Mochila No Acotado 21

6.1 Algoritmo 21
6.2 Diseño 22

7 Conclusión 23

1 Modelo Sticker

El Modelo Sticker es un modelo de computación inpirado en cadenas de ADN propuesto por Sam Roweis,[2] donde tal modelo se basa en procedimientos de filtrado, con memoria de acceso aleatorio y una nueva manera de codificar la información. En ese mismo orden, éste modelo se distingue de otros modelos computacionales por la última característica enlistada, la manera en que codifica la información.

1.1 Concepto de Cadena de Memoria

El eje central de este modelo son las cadenas de memoria, tales cadenas consisten en hebras simples de ADN del tipo (n, k, m), tal que $n \geq k.m$, siendo n la longitud de la cadena, k las cantidad de subcadenas o regiones, y m la longitud de cada subcadena.

Asimismo, un sticker es un elemento asociado a una región en la cadena de memoria, una vez asociado se dice que la región está complementada, transformando la cadena en una hebra doble parcial. En ese sentido, se dice que en cada región donde haya un sticker complementándole, esta región está *activada*, mientras que la región donde se carezca de sticker se dice está *desactivada*. Tal abstracción binaria también se puede expresar de modo *0/1*.

[2]Sam Roweis et al. "A Sticker-Based Model for DNA Computation". In: *Journal of computational biology : a journal of computational molecular cell biology* 5 (Feb. 1998), pp. 615–29. DOI: 10.1089/cmb.1998.5.615.

Figure 1: Cadenas de Memoria

En esta imágen se ilustran dos cadenas de memoria, la primera cadena de memoria tiene un sticker en la primera y tercera región, por lo que estas están *activadas*, leyéndose 101000; mientras que para la segunda cadena de memoria no existe sticker alguno asociado, por ello todas las regiones están *desactivadas*, leyéndose 000000.

1.2 Concepto de Tubo

Dentro del modelo sticker un tubo consiste en un multiconjunto cuyo contenido son complejos de memorias del mismo tipo.

Las operaciones principales a ser realizadas sobre un tubo 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 tubos. También se nota como $(T_1 \cup T_2)$.

- *separar*(T, i): Para un tubo T y un índice i tal que $(1 \leq i \leq n)$, retorna un $+(T, i)$ y $-(T, i)$, de modo que el primer tubo contiene todos los complejos de memoria cuya región i-ésima esté *activada*, mientras que el segundo contiene los complejos cuya región i-ésima *desactivada*.

- *encender*(T, i): Para un tubo T y un índice i tal que $(1 \leq i \leq n)$, retorna un tubo T' con la región i-ésima *activada* en cada complejo de memoria.

- *apagar*(T, i): Para un tubo T y un índice i tal que $(1 \leq i \leq n)$, retorna un tubo T' con la región i-ésima *desactivada* en cada complejo de memoria.
- *leer*(T): Dado un tubo $T \neq \emptyset$, lee su contenido.

1.3 Concepto de Librería

Una librería dentro del modelo sticker consiste en complejos de memoria de modo (k, l), teniendo k sub cadenas y las primeras l subcadenas están *activadas* o *desactivadas* en todas sus posibles combinaciones, mientras que las regiones $k - l$ restantes están *desactivadas*.[3]

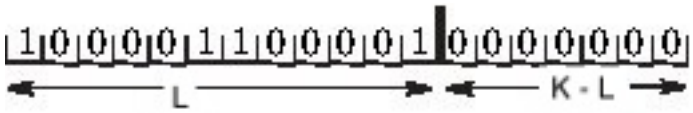

Figure 2: Librería(k, l)

2 Subrutina Ordenado por Cardinalidad

Problema: Sea $A = \{1, \ldots, p\}$, $B = \{b_1, \ldots, b_s\} \subseteq A$, $\mathcal{F} = \{D_1, \ldots, D_t\} \subseteq \mathcal{P}(A)$. Ordene los conjuntos de $\mathcal{F}$ de acuerdo a su cardinalidad en B, o en otras palabras, la cantidad de elementos de $B \cap D_i$.

El paper bajo estudio nos sugiere el implementar un programa molecular definido con las siguientes características:

- La entrada será un tubo T_0 conteniendo complejos de memoría codificando cada conjunto de la familia $\mathcal{F}$. Para esto, cada complejo de memoria en T_0 se representará mediante la función booleana $T_0 = \{\{\sigma : |\sigma| = p \ \wedge \ \exists j \ (\chi_{D_j} = \sigma)\}\}$; tal que χ_{D_j} es la función característica de D_j en A, siendo $(\chi_{D_j}(i) = 1$ si $i \in D_j$ de lo contrario $\chi_{D_j}(i) = 0$ si $i \in A - D_j)$.

[3] Hassan Taghipour, Mahdi Rezaei, and Heydar Ali Esmaili. "Solving the 0/1 Knapsack Problem by a Biomolecular DNA Computer". In: *Advances in Bioinformatics* 2013 (2013), pp. 1–6. DOI: 10.1155/2013/341419. URL: https://doi.org/10.1155/2013/341419.

- El programa consistirá en un bucle principal realizando s pasos, donde para el paso i-ésimo se tendrá $i+1$ tubos $(T_0, T_1, \ldots, T_i)$, tales tubos generados verificando la condición $\forall\sigma\ (\sigma \in T_j \rightarrow |\sigma \cap \{b_1, \ldots, b_i\}| = j)$. Los tubos se compondrán de la siguiente manera:

$$\begin{cases} T_0 = -(T_0, b_{i+1}) \\ T_j = +(T_{j-1}, b_{i+1}) \cup -(T_j, b_{i+1}) \quad (1 \leq j \leq i) \\ T_{i+1} = +(T_i, b_{i+1}) \end{cases}$$

2.1 Algoritmo

Las ideas previas nos indican el siguiente algoritmo:

```
1: procedure Cardinal_Sort(T_0, B)
2:     for i = 1 to s do
3:         (T_0, T'_0) ← separar(T_0, b_i)
4:         for j = 0 to i − 1 do
5:             (T'_{j+1}, T''_j) ← separar(T_j, b_i)
6:             T_j ← mezclar(T'_j, T''_j)
7:         end for
8:         T_i ← T'_i
9:     end for
10:    return T_0, ..., T_s
11: end procedure
```

2.2 Otras Notaciones

Para usos posteriores dentro de este informe, podemos notar el uso de *Cardinal_Sort* de modo:

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

2.3 Proposiciones

2.3.1 Proposición 1

$\forall i\ (1 \leq i \leq s \rightarrow \forall j \leq i\ \forall \sigma(\sigma \in T_{i,j} \rightarrow |\sigma \cap B_i| = j))$.

Prueba. Por inducción en i, veamos que $\forall j \leq 1\ \forall \sigma \in T_{1,j}(|\sigma \cap B_1| = j)$.

- Sea $\sigma \in T_{1,0} = +(T_{0,-1}, b_1) \cup -(T_{0,0}, b_1)$. Partiendo de que $T_{0,-1} = \emptyset$ y $T_{0,0} = T_0$, resultando que $\sigma \in -(T_0, b_1)$; esto es, $b_1 \notin \sigma$; por lo que $|\sigma \cap B_1| = 0$.
- Sea $\sigma \in T_{1,1} = +(T_{0,0}, b_1) \cup -(T_{0,1}, b_1)$. Partiendo de que $T_{0,1} = \emptyset$ y $T_{0,0} = T_0$, resultando que $\sigma \in -T_{0,0} = T_0$ y $b_1 \in \sigma$; por lo que $|\sigma \cap B_1| = 1$.

Sea $i(1 \leq i < s)$ tal que $\forall j \leq i\ \forall \sigma \in T_{i,j}(|\sigma \cap B_i| = j)$. Veamos que el resultado verifica que para $i+1$. Por ello, se procede por inducción en j: $\forall j \leq i+1$ $\forall \sigma \in T_{i+1,j}\ (|\sigma \cap B_{i+1}| = j)$.

- Sea $\sigma \in T_{i+1,0} = +(T_{i,-1}, b_{i+1}) \cup -(T_{i,0}, b_{i+1})$. Siendo $T_{i,-1} = \emptyset$, resulta que $\sigma \in T_{i,0}$ y $b_{i+1} \notin \sigma$, por hipótesis de inducción deducimos que $|\sigma \cap B_i| = 0$. Por lo que $|\sigma \cap B_{i+1}| = 1$, partiendo de que $b_{i+1} \notin \sigma$.
- Sea $j > 0$ y $\sigma \in T_{i+1,j} = +(T_{i,j-1}, b_{i+1}) \cup -(T_{i,j}, b_{i,-1})$, entonces:
  - Si $\sigma \in T_{i,j}$ y $b_{i+1} \in \sigma$, por hipótesis de inducción tenemos que $|\sigma \cap B_i| = j-1$. Partiendo de que $b_{i+1} \in \sigma$, se concluye en que $|\sigma \cap B_{i+1}| = j-1+1 = j$.
  - Si $\sigma \in T_{i,j}$ y $b_{i,j} \notin \sigma$, por hipótesis de inducción tenemos que $|\sigma \cap B_i| = j$. Tal que $b_{i+1} \notin \sigma$, tenemos que $|\sigma \cap B_{i+1}| = j$.

2.3.2 Proposición 2

$\forall \sigma \in T_0 \; \forall i \; (0 \leq i \leq s \rightarrow \sigma \in T_{i,|\sigma \cap B_i|})$

Prueba. Por indución en i. Para $i = 0$, el resultado toca la trivialidad. Asumimos que el resultado mantiene que para i $(0 \leq i < s)$, se comprueba esto para $i + 1$.

- Si $b_{i+1} \in \sigma$, tenemos que $|\sigma \cap B_{i+1}| = 1 + |\sigma \cap B_i|$. Por hipótesis de inducción, $\sigma \in T_{i,|\sigma \cap B_{i+1}|}$, entonces $\sigma \in +(T_{i,|\sigma \cap B_i|,b_{i+1}}) \subseteq T_{i+1,|\sigma \cap B_i|+1}$.
- Si $b_{i+1} \notin \sigma$, entonces $|\sigma \cap B_{i+1}| = |\sigma \cap B_i|$. Por hipótesis de inducción $\sigma \in T_{i,|\sigma \cap B_i|}$, entonces $\sigma \in -(T_{i,|\sigma \cap B_i|,b_{i+1}}) \subseteq T_{i+1,|\sigma \cap B_i|} = T_{i+1,|\sigma \cap B_{i+1}|}$.

2.4 Corolarios

2.4.1 Corolario 1 - Robustez del Algoritmo

Si $\forall j \; \forall \sigma \; (0 \leq j \leq s \wedge \sigma \in T_{s,j} \rightarrow |\sigma \cap B| = j)$. Dicho de otra manera, j debe de ser un valor entre 0 y s, y para cada complejo de memoria σ existente en $T_{s,j}$, tantas moléculas contenga para la cardinalidad $|\sigma \cap B|$,esta será igual a j.

2.4.2 Corolario 2 - Completitud del Algoritmo

Si $\sigma \in T_0$ y $|\sigma \cap B| = j$, entonces $\sigma \in T_{s,j}$. Esto es, si existe un complejo de memoria en el tubo de entrada, y la cardinalidad de la misma en B es j, pues el mismo complejo de memoria debe de aparecer en el tubo j-ésimo.

2.5 Traza Subrutina Ordenado por Cardinalidad

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

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

Identificando los elementos que cumplen $B \cap D_j$ en $\mathcal{F}$ le resaltamos en rojo de modo que tenemos: $\{\{2, 6\}, \{3\}, \{4\}, \{2, 4\}\}$. Codificando $\mathcal{F}$ 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 $\mathcal{F}$ tras la ejecución de *Cardinal_Sort* tendremos:

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

Cumpliendo de esta manera los corolarios 1 y 2.

3 Subrutina de Llenado

Sea $A = \{1, \ldots, p\}$, $r \in \mathbb{N}$ y $f : A \to \mathbb{N}$ una función. Si $B \subseteq A$, notamos $f(B) = \sum_{i \in B} f(i)$, por conveniencia deifnimos que para $f(0) = 0$. Sea $q_f = f(A)$, $A_i = \{0, \ldots, i\}$ $(0 \leq i \leq p)$, y T_0 un multiconjunto de complejos de memoria de forma (n, k, m) con $k \geq p + r + q_f$.

Si $\sigma \in T_0$, podemos asumir que tal complejo de memoria está formado por las regiones:

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

La subrutina opera sobre T_0 modificando sus elemento haciendo las moléculas del tubo de salida en (F_σ) el peso con respecto a f del subconjunto en A codificado en (A_σ), mientras que las zonas (R_σ) y (L_σ) no son modificadas, expresándose todo esto de forma:

$$\sum_{i=1}^{p} \sigma(i) f(i) \qquad \sum_{j=p+r+1}^{p+r+q_f} \sigma(j)$$

3.1 Algoritmo

```
1: procedure Parallel_Fill(T_0, f, p, r)
2:     for i = 1 to p do
3:         (T^+, T^-) ← separar(T_0, i)
4:         for j = 1 to f(i) do
5:             T^+ ← encender(T^+, p + r + f(A_{i-1}) + j)
6:         end for
7:         T_0 ← mezclar(T^+, T^-)
8:     end for
9:     return T_0
10: end procedure
```

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

3.2 Lemas

3.2.1 Lema 1

1. La zona inicial de la molécula (codificando el subconjunto de A) no cambia durante la ejecución del programa, siendo esto:

$$\forall \sigma \in T_0 \; \forall k \; (1 \leq k \leq p \rightarrow A\sigma = (A\sigma^k)) \tag{1}$$

2. Las moléculas que son obtenidas in el k-ésimo paso del búcle principal es almacenado en el tubo T_k, siendo esto:

$$\forall \sigma \in T_0 \; \forall k \; (1 \leq k \leq p \rightarrow \sigma^k \in T_k) \tag{2}$$

3. Cada molécula en el k-ésimo tubo proviene de alguna molécula del tubo inicial, siendo esto:

$$\forall k \; (1 \leq k \leq p \rightarrow \forall \tau \in T_k \; \exists \sigma \in T_0 (\sigma^k = \tau)) \tag{3}$$

4. La ejecución de un paso en el bucle principal no modifica las regiones correspondiente a los pasos previos, siento esto:

$$\forall \sigma \in T_0 \; \forall i \; \forall k \; (1 \leq i \leq k \leq p \rightarrow \sigma^i_{|R_i} = \sigma^k_{|R_i}) \tag{4}$$

5. Luego de la ejecución del i-ésimo paso del bucle principal, la región R_i en σ ha sido modificada para de acuerdo con el valor de $\sigma(i)$, siendo esto:

$$\forall \sigma \in T_0 \; \forall i \; \forall k \; (1 \leq i \leq k \leq p \rightarrow \sigma^k_{|R_i} \equiv \sigma(i)) \tag{5}$$

6. La ejecución de un paso en el bucle principal no altera las zonas $(L\sigma)$ ni $(R\sigma)$, siendo esto:

$$\forall \sigma \in T_0 \; \forall k \; (1 \leq k \leq p \rightarrow (L\sigma) = (L\sigma^k) \wedge (R\sigma) = (R\sigma^k)) \tag{6}$$

Lo resultante para estos lemas nos aseguran que el bucle principal modifique exclusivamente la región R_i en una molécula, codificando de esta manera el peso para cada una de estas.

3.3 Proposiciones

3.3.1 Proposición 3

Sea $B \subseteq A$ tal que $\sigma_B \in T_0$, entonces para cada k $(1 \leq k \leq p)$, tenemos que:

$$f(B \cap \{1, \ldots, k\}) = \sum_{j=p+r+1}^{p+r+f(A_k)} = \sigma_B^k(j)$$

Prueba. Partiendo de (1) se tiene que

$$f(B \cap \{1, \ldots, k\}) = \sum_{i=1}^{k} f(i).\sigma_B(i) = \sum_{i=1}^{k} f(i).\sigma_B^k(i)$$

Por otro lado, (1) y (5) asegura que:

$$f(i).\sigma_B^k(i) = \sum_{j \in R_i} \sigma_B^k(j) \ (1 \leq i \leq k)$$

3.4 Corolarios

3.4.1 Corolario 3

Para cada $B \subseteq A$ tal que $\sigma_B \in T_0$ existe algún $\tau \in T_p$ tal que $f(B) = \sum_{i=p+r+1}^{p+r+q_f} \tau(i)$.

Prueba. Dado un $B \subseteq A$, consideremos que la molécula asociada $\sigma_B \in T_0$. Ello satisface el considerar $\tau = \sigma_B^p$, partiendo de que $f(B) = f(B \cap \{1, \ldots, p\}) = \sum_{j=p+r+1}^{p+r+q_f} \sigma_B^p(j)$.

3.5 Traza Subrutina de Llenado

A fin de ilustrar el comportamiento para el algoritmo bajo estudio, tenemos:

- $A : \{1, 2, 3, 4\}$
- $B : \{1, 2, 4\}$
- $T_0 : \{\{1, 2, 3\}, \{1\}, \{2, 3, 4\}\}$
- $p = 4$
- $r = 1$
- $q_f = 20$

Definido lo previo, tendremos las siguientes zonas delimitadas de la siguiente manera:

- $(A\sigma) = \sigma(1) \dots \sigma(4)$
- $(L\sigma) = \sigma(4 + 1) \dots \sigma(4 + 1)$
- $(F\sigma) = \sigma(4+1+1) \dots \sigma(4+1+20)$
- $(R\sigma) = \sigma(4 + 1 + 20 + 1) \dots$

Para el tubo T_0 los valores que cumplen $B \cap A$ le resaltamos en rojo, teniendo $T_0 : \{\{1, 2, 3\}, \{1\}, \{2, 3, 4\}\}$, donde al codificar este con las delimitaciones previas, tenemos:

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

Nótese que que la región (A_σ) y (L_σ) se han resaltado para su apreciación, mientras que lo restante se reserva para la región (F_σ).

Una vez odificado el tubo T_0, éste al ser procesado por $Parallel\_Fill$, tendremos como salida un tubo de la siguiente manera:

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

4 Problema de Suma de Subconjuntos

Problema: Sea $A = \{1, \ldots, p\}$ y $\omega \to \mathbb{N}$ una función peso. Sea $k \in \mathbb{N}$, tal que $k \leq \omega(A) = q_\omega$, determinar si existe un subconjunto $B \subseteq A$, tal que la suma de los pesos en B sea exactamente k.

Definido una vez el problema, el paper bajo estudio presenta un diseño de un programa dentro del modelo sticker tal que resuelve el problema de la suma de subcojnuntos. El tubo de entrada T_0 conteniendo una librería de forma $(p + q_\omega, q)$.

La primera fase, la fase de llenado, cada molécula σ del tubo de entrada será llenado a fin de conseguir en los últimos componentes q el peso de los subcojnuntos que codifica. Los tubos obtenidos luego serán ordenados de acuerdo a su cardinalidad. Por último, el tubo k-ésimo es leído, donde tal tubo T_k representa el peso buscado.

4.1 Algoritmo

La descripción del planteamiento explicado sugiere un programa molecular como el que sigue:

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

4.2 Diseño

La implementación del algoritmo expuesto considera el uso de $4 + 2q$ tubos T y un coste de $2p+q+1+\frac{q.(q+3)}{2}$ operaciones o pasos. Asimismo, el diseño de éste programa no sólo soluciona un problema de desición pues a su vez retorna todas las soluciones posibles al problema propuesto.

4.3 Teoremas

4.3.1 Theorema 1 - Robustez

Si $T_{salida} \neq \emptyset$, entonces existe un $B \subseteq A$ tal que $\omega(B) = k$.

Prueba. Tomando $\tau \in T_{salida}$, aplicando (3) y la proposición 3, obtenemos $\sigma \in T_0$ verificamos que el resultado para B_σ.

4.3.2 Theorema 2 - Completitud

Sea $\sigma \in T_0$ tal que $\omega(B_\sigma) = k$, entonces $T_{salida} \neq \emptyset$.

Prueba. Sea $\sigma \in T_0$ tal qeu $\omega(B_\sigma) = k$, partiendo del corolario 3, luego de la ejecución de la subrutina de llenado, tenemos una molécula $\tau = \sigma^p \in T_1$ tal que $\omega(B_\sigma) = \sum_{i=p+1}^{p+q_\omega} \tau(i)$, entonces $\tau \in T_{salida}$.

4.4 Traza

A fin de ilustrar el comportamiento para el algoritmo bajo estudio y tomando como referencia el tubo del apartado anterior T_0 para simplicitud, tenemos:

- $A : \{1, 2, 3, 4\}$
- $B : \{1, 2, 4\}$
- $T_0 : \{\{1, 2, 3\}, \{1\}, \{2, 3, 4\}\}$
- $p = 4$
- $r = 1$
- $q_f = 20$
- $k = 4$

Una vez definido el tubo T_0, y éste procesado por $Parallel\_Fill$, tendríamos como salida el tubo T_1 codificado de la siguiente manera:

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

Este tubo T_1, una vez ordenado por $Cardinal\_Sort$ resultaría de la siguiente manera:

$T_0 : [\,]$
$T_1 : [\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,]$
$T_2 : [\,]$
$T_3 : [\,]$
$T_4 : [\,1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,]$
$T_5 : [\,]$
$T_6 : [\,]$
$\ldots$
$T_{13} : [\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,]$
$\ldots$

Finalmente, para un $k = 4$, el tubo a buscar es el T_4, por lo que tendríamos como salida: $1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0$; de modo que éste resultado da cumpliento con los teoremas 1 y 2.

5 Problema de la Mochila Acotado

Problema: sea $A = \{1, \ldots, p\}$ un conjunto finito no vacío, $\omega : A \to \mathbb{N}$ una función peso y $\rho : A \to \mathbb{N}$ una función valor. Sea $k, k' \in \mathbb{N}$ tal que $k \leq \omega(A) = q_\omega$ y $k' \leq \rho(A) = q_\rho$; determine si existe un subconjunto $B \subseteq A$ tal que $\omega(B) \leq k$ y $\rho(B) \geq k'$.

Esto nos lleva plantear el diseño de un programa molecular el cual resuelva el problema propuesto utilizando una librería$(p + q_\omega + q_\rho, p)$.

En la fase inicial, la fase de llenado, se procede como en el problema previo: cada complejo de memoria en el tubo inicial es llenado apropiadamente tal que codifique el *peso* asociado al subconjunto, seguido de la ordenación de los mismos de acuerdo a su cardinalidad de $\omega(A\sigma)$, empezando por obtener algunos tubos $T_0, \ldots, T_{q_\omega}$ tales que $\forall\sigma\ (\sigma \in T_j \Rightarrow |\omega(A\sigma)| = j)$.

Con los tubos $T_0 \cup \cdots \cup T_k$ se lleva a cabo una segunda operación de llenado, esta con respecto a los *valores*. Posteriormente, las moléculas σ del tubo resultante son ordenadas con respecto a su cardinalidad en $\rho(A\sigma)$, obteniendo nuevamente una serie de tubos de forma $T_0, \ldots, T_{q_\rho}$, tales que $\forall\sigma\ (\sigma \in T_j \Rightarrow |\rho(A\sigma)| = j)$.

Finalmente, se obtiene la solución al problema con tubos $T_{k'} \cup \ldots T_{q_\rho}$ y estos son leidos.

5.1 Algoritmo

Las directrices del apartado anterior guían a definir un programa molecular de la siguiente forma

---

1: **procedure** $Bounded\_Knapsack(p, w, \rho, k, k')$
2: $q_w \leftarrow \sum_{i=1}^{p} w(i)$; $q_\rho \leftarrow \sum_{i=1}^{p} \rho(i)$; $T_0 \leftarrow$Libería$(p + q_w + q_\rho, p)$
3: $T_0 \leftarrow Parallel\_Fill(T_0, w, p, 0)$
4: $T_0 \leftarrow Cardinal\_Sort(T_0, p + 1, p + q_w)$
5: $T_1 \leftarrow \emptyset$
6: **for** $i = 1$ to k **do**
7: $T_1 \leftarrow mezclar(T_1, Cardinal\_Sort(T_0, p + 1, p + q_w)[i])$
8: **end for**
9: $T_0 \leftarrow Parallel\_Fill(T_1, \rho, p, q_w)$
10: $Cardinal\_Sort(T_0, p + q_w + 1, p + q_w, q_\rho)$
11: $T_1 \leftarrow \emptyset$
12: **for** $i = k'$ to q_ρ **do**
13: $T_1 \leftarrow merge(T_1, Cardinal\_Sort(T_0, p + q_w + 1, p + q_w + q_\rho)[i])$
14: **end for**
15: $leer(T_1)$
16: **end procedure**

5.2 Diseño

6 Problema de la Mochila No Acotado

Problema: bajo las mismas condiciones del problema anterior, determinar si un subconjunto $B \subseteq A$ tal que $\rho(B) = max\{\rho(C) : C \subseteq A \ \wedge \ \omega(C) \leq k\}$.

Una solución molecular es obtenida a partir del problema la mochila acotado, mas un cambio en la fase final en donde se ordenan los tubos de salida con respecto a su valor y se elige sólamente el tubo con mayor valor.

6.1 Algoritmo

La nueva propuesta lleva a considerar un programa molecular similar al previo mas se ha de notar las variaciones en la salida del programa al retornar sólo el tubo con mayor valor.

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

6.2 Diseño

7 Conclusión

References

Pérez-Jiménez, Mario J. and Fernando Sancho-Caparrini. "Solving Knapsack Problems in a Sticker Based Model". In: *DNA Computing*. Ed. by Natasa Jonoska and Nadrian C. Seeman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 161–171. ISBN: 978-3-540-48017-4.

Roweis, Sam et al. "A Sticker-Based Model for DNA Computation". In: *Journal of computational biology : a journal of computational molecular cell biology* 5 (Feb. 1998), pp. 615–29. DOI: 10.1089/cmb.1998.5.615.

Taghipour, Hassan, Mahdi Rezaei, and Heydar Ali Esmaili. "Solving the 0/1 Knapsack Problem by a Biomolecular DNA Computer". In: *Advances in Bioinformatics* 2013 (2013), pp. 1–6. DOI: 10.1155/2013/341419. URL: https://doi.org/10.1155/2013/341419.