Desarrollo de un Algoritmo Computacional para la Linealización de Cadenas de Decaimiento y Transmutación

Carlos Antonio Cruz López y Juan Luis François Lacouture
Universidad Nacional Autónoma de México, Facultad de Ingeniería,
Departamento de Sistemas Energéticos
Paseo Cuauhnáhuac 8532, Col. Progreso, C.P. 62550, Jiutepec Morelos
cacl@ier.unam.mx; jlfl@fi-b.unam.mx

Resumen

Una de las metodologías más utilizadas para resolver las ecuaciones de Bateman, en el problema de quemado, es el método TTA (*Transmutation Trayectory Analysis*). En dicho método se descompone una red de decaimientos en elementos lineales conocidos como trayectorias, mediante un proceso conocido como linealización. En el presente trabajo se muestra un algoritmo alternativo para encontrar y construir dichas trayectorias, el cual considera tres aspectos de la linealización: la información "a priori" sobre los elementos que integran una red de decaimiento y transmutación, el uso de una nueva notación, y en las funciones para el tratamiento de cadenas de texto (que son comunes en la mayoría de los lenguajes de programación). Una de las principales ventajas del algoritmo es que puede condensar la información de una red de decaimientos y transmutación en sólo dos vectores. De estos últimos es posible determinar cuántas cadenas lineales pueden extraerse de la red e incluso su longitud (en el caso de que no sean cíclicas). A diferencia del método de Búsqueda en Profundidad (Deep First Search) que es ampliamente utilizado para el proceso de linealización, el método propuesto en el presente trabajo no tiene una rutina de retroceso y en su lugar ocupa un proceso de compleción, pues completa fragmentos de cadena en lugar de retroceder hasta el inicio de las trayectorias. El algoritmo desarrollado puede aplicarse de forma general a la búsqueda de información y a la linealización de las estructuras de datos computacionales conocidas como árboles. También puede aplicarse a problemas de ingeniería donde se busque calcular la concentración de alguna sustancia en función del tiempo, partiendo de ecuaciones diferenciales lineales de balance.

1. INTRODUCCIÓN

Durante la operación de un reactor nuclear, el combustible experimenta cambios en su composición debido a tres procesos principalmente: el de fisión, el de captura, y el proceso de decaimiento nuclear. Debido a que las modificaciones que experimenta el combustible se originan cuando se produce energía, al estudio de dichos cambios se la ha dado tradicionalmente el nombre de análisis del quemado de combustible [1].

Para modelar el cambio en el tiempo de los núclidos que componen el combustible, se plantea un conjunto de ecuaciones diferenciales conocido como ecuaciones de Bateman [2]. La solución más simple proviene de dividir el problema de una red de decaimientos y transmutación, en cadenas lineales simples, en las que no existan ramificaciones. Estas cadenas simples suelen llamarse

trayectorias, y constituyen la principal característica del método TTA (*Trayectory Transmutation Analysis*).

Dicho método resuelve las ecuaciones para cada trayectoria, y al final realiza una superposición de los resultados para cada una, asegurándose de sumar todas las contribuciones para encontrar la concentración de un núclido específico. Resulta claro que el proceso de linealización o de búsqueda de trayectorias es una parte fundamental del método TTA.

En los recientes trabajos [3][4][5][6] que consideran o mencionan el método TTA, no se ha indagado mucho acerca de la construcción de las trayectorias, y a lo más se recomienda usar el método de búsqueda en profundidad (*Deep First Search*, DFS) para la linealización de redes de decaimiento y transmutación. Esta falta de interés está justificada debido a que la mayoría de los códigos definen sus cadenas y trayectorias una sola ocasión, y en base al interés que se tenga sobre un número particular de núclidos, tal como lo hace MCNP [7].

Resulta claro que no pueden considerarse todos los posibles núclidos que surgen en una red de decaimiento y transmutación, porque esto afecta de forma considerable el tiempo de cómputo. Además, que muchos núclidos no tienen un interés o una aplicación práctica que motive su estudio o seguimiento en alguna trayectoria en específico [1]. Sin embargo, en los últimos años ha surgido un nuevo paradigma sobre los cálculos y modelos, que apuesta por dejar los métodos conservadores o simplificados, para incorporar nuevas técnicas y tecnologías computacionales que pueden hacer frente al problema del tiempo de cómputo [8].

Por lo anterior, el presente trabajo tiene como motivación el presentar un método alternativo que permita la construcción y linealización de las cadenas de decaimiento y transmutación. Dicho método propone un formato para representar las redes de decaimiento bajo una notación de guiones e índices, que, usando las funciones de ordenamiento estándar en lenguajes de programación, permite reducir de forma considerable las operaciones realizadas por otros métodos de búsqueda.

Finalmente, el formato para representar las redes es también una contribución que condensa la información en solo dos líneas de texto (o vectores), de la misma forma en que formatos como el de Newick lo hacen [9].

El presente trabajo se divide en 6 secciones. En la sección 2 se discuten de forma general las ecuaciones de Bateman. En la sección 3 se describe el proceso de búsqueda de trayectorias, y en la sección 4 se habla del algoritmo más usado para realizar dichas búsquedas: el DFS. La sección 5 contiene una descripción del método propuesto y de su notación, y la sección 6 establece posibles criterios de comparación entre el método propuesto y el DFS.

2. ECUACIONES DE BATEMAN

En 1910 el matemático Harry Bateman [2] planteó una solución al problema de una cadena de decaimiento lineal, como la que se muestra en la Figura 1. Aquí el término lineal no hace referencia al tipo de ecuación diferencial resuelta, sino a la estructura de la cadena, donde salvo el elemento inicial X_1 y el final, X_n: todos los demás tienen sólo un sucesor y un solo antecesor.

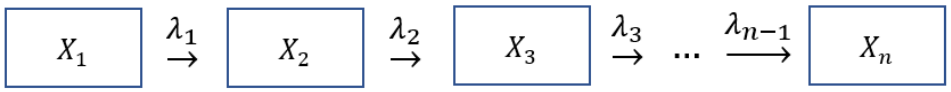

Figura 1. Cadena de decaimiento lineal propuesta por Bateman. Los X_i representan los elementos de la cadena, y las λ_i son sus constantes de decaimiento.

La solución de Bateman supone que la tasa de pérdida de un elemento X_i, con respecto al tiempo, es directamente proporcional a la cantidad de dicho elemento en dicho instante de tiempo. Es decir:

$$\frac{d}{dt}X_i(t) = -\lambda_i X_i(t) \tag{1}$$

Donde λ_i es la constante de proporcionalidad entre la tasa de pérdida y la concentración de un elemento de la cadena. Esta constante también se conoce como constante de decaimiento.

Además, considera que la concentración que pierde un elemento X_i en la cadena, se transforma en la concentración del elemento sucesor X_{i+1}. Con ambas suposiciones se puede plantear el siguiente sistema de ecuaciones diferenciales lineales para la concentración del núclido X_i:

$$\frac{d}{dt}X_i(t) = -\lambda_i X_i(t) + \lambda_{i-1}X_{i-1}(t), \quad \text{con} \quad 1 \leq i \leq n \tag{2}$$

Existen dos soluciones propuestas por Bateman de acuerdo a las condiciones iniciales utilizadas. La más elegante viene de considerar que al tiempo $t = 0$, únicamente el primer elemento tiene una concentración distinta de cero. Es decir:

$$X_1(0) \neq 0 \quad \text{y} \quad X_j(0) = 0 \quad \forall j \neq 1 \tag{3}$$

La función que satisface las condiciones (1), (2) y (3) es:

$$X_i(t) = X_1(0) \prod_{j=1}^{i-1} \lambda_j \sum_{k=1}^{i} \frac{e^{-\lambda_k t}}{\prod_{\substack{j=1 \\ j\neq k}}^{i} (\lambda_j - \lambda_k)} \tag{4}$$

Por supuesto, en los problemas de quemado las condiciones de (3) no suelen cumplirse, puesto que existen bastantes elementos cuya concentración puede ser distinta de cero para un paso determinado. Sin embargo, aun así, es posible utilizar la ecuación (4) mediante un proceso de superposición. Éste proceso consiste en resolver primero el esquema que se muestra en la Figura 1 usando las condiciones de (3). Después se redefine la cadena lineal removiendo el primer elemento e iniciando en el primer núclido cuya concentración sea distinta de cero. Este proceso se repite cuantas veces sea necesario, tal como se muestra en la Figura 2. Al final se suman las concentraciones individuales encontradas.

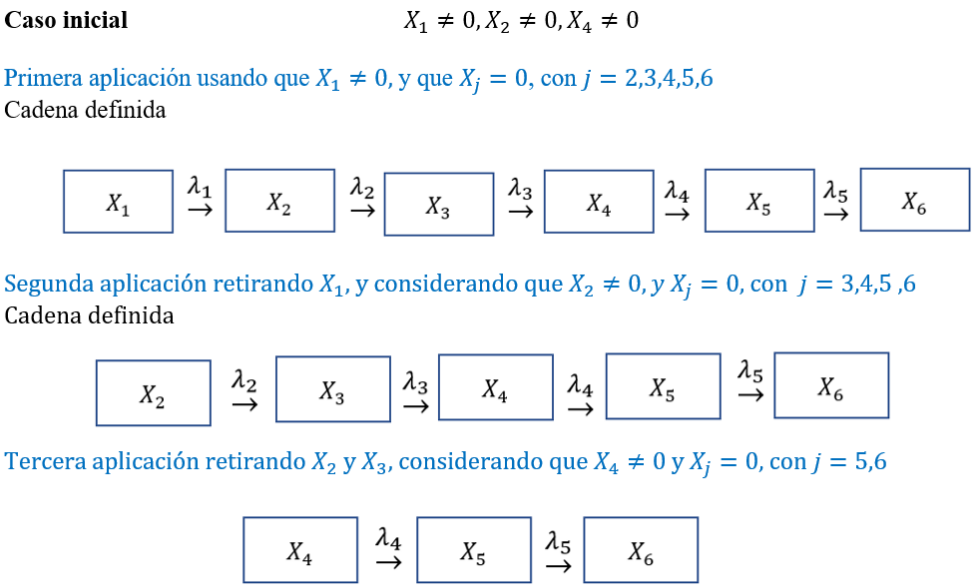

Figura 2. Ejemplo del proceso de superposición para un caso donde hay más de un núclido con una concentración distinta de cero.

Es claro que cuando se publicó la solución propuesta por Bateman, aún no se había desarrollado ni estudiado el concepto de ramificación, y mucho menos se había generalizado el proceso de pérdida para incluir a la fisión o a la reacción de captura. Particularmente las cadenas lineales como la que muestra la Figura 1 rara vez aparecen en los problemas de quemado. Más bien se tienen estructuras complejas debido a las ramificaciones que ocurren cuando un núclido se genera o pierde de varias formas, y se habla de una red de decaimientos o transmutación como la que se muestra en la Figura 3.

Afortunadamente, las ramificaciones pueden describirse a través de un conjunto de factores conocidos como razones de ramificación (*branching ratios*) que permiten conservar la linealidad en las ecuaciones diferenciales.

Por ejemplo, para núclidos que tienen varios tipos de decaimiento (es decir α, β, γ, etc.), puede definirse una constante λ_x parcial para cada una de estas reacciones, de tal forma que la constante total esté dada por:

$$\lambda = \lambda_\alpha + \lambda_\beta + \lambda_\gamma + \cdots \tag{5}$$

Esta definición puede generalizarse de tal forma que se relacionen dos núclidos i y j, siendo el primero el "padre" y el último el "hijo" en el proceso de decaimiento. Así, la fracción de decaimientos del núclido i que dan lugar al núclido j estaría caracterizado por $\lambda_{i,j}$. Con esto podemos calcular las razones de ramificación $b_{i,j}$ como:

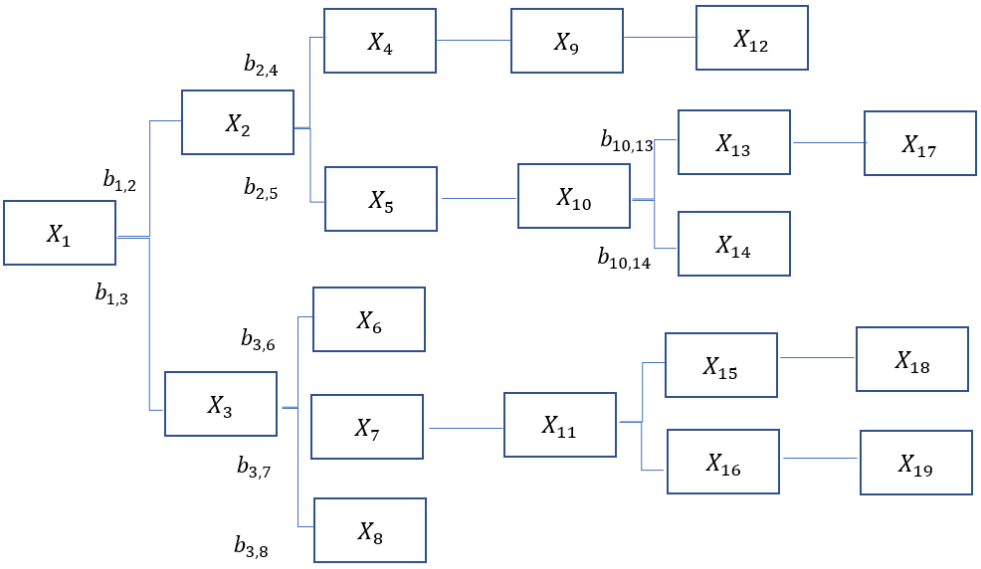

Figura 3. Esquema de una red de decaimiento y transmutación. Los factores de ramificación se muestran en cada bifurcación de la estructura.

$$b_{i,j} = \frac{\lambda_{i,j}}{\lambda_i} \tag{6}$$

En la Figura 3 pueden apreciarse estas razones de ramificación. Particularmente las librerías de decaimiento proporcionan la constante de decaimiento total λ_i y los valores de las razones de ramificación $b_{i,j}$, así que en la ecuación (4) en lugar de λ_k y λ_j, se debe utilizar el siguiente producto $b_i\lambda_{i,j}$.Para incluir los procesos de captura y el de fisión, puede definirse una constante efectiva de pérdida, dada por:

$$\lambda_i^{eff} = \lambda_i + \sum_g \sum_k \phi^g \sigma_k^g \tag{7}$$

Donde ϕ^g es el flujo para el grupo de energía g, y σ_k^g es la constante microscópica de la reacción del tipo k y para el grupo de energía g. Así mismo se debe definir una razón de ramificación efectiva, dada por:

$$b_{i,j}^{eff} = \frac{b_{i,j}\,\lambda_i + \sum_g \sum_k y_{i,j,k}^g \phi^g \sigma_{i,k}^g}{\lambda_i^{eff}} \tag{8}$$

Donde se ha incluido el término $y_{i,j,k}^g$, que hace referencia al número promedio de núclidos del tipo j, que son producidos por el núclido i en la reacción del tipo k, para el grupo de energía g. Con estas modificaciones puede generalizarse la solución de Bateman, quedando de la siguiente forma:

$$X_i(t) = X_1(0) \prod_{j=1}^{i-1} b_{j,j+1}^{eff} \lambda_j^{eff} \sum_{k=1}^{i} \frac{e^{-\lambda_k^{eff} t}}{\prod_{\substack{j=1 \\ j \neq k}}^{i} (\lambda_j^{eff} - \lambda_k^{eff})} \tag{9}$$

Las únicas dos condiciones para poder aplicar la ecuación (9) son que la cadena sea lineal (es decir, que no sea una red) y que no haya elementos repetidos en ella para que de esta forma no se anule el denominador.

3. LINEALIZACIÓN DE UNA RED DE DECAIMIENTO Y TRANSMUTACIÓN

Tal como se mencionó al final de la sección anterior, para poder aplicar la ecuación (9) es necesario contar con una cadena lineal, como la que aparece en la Figura 1. Así que se debe descomponer nuestra red de decaimientos en elementos lineales que no tengan ramificaciones. En otras palabras, en cadenas que solo tengan un inicio y un final. Teóricamente esto puede hacerse siempre y cuando no existan cadenas cíclicas. Por ejemplo, si en la Figura 3 luego del elemento X_{19} volviese a aparecer X_1, entonces tendríamos una cadena del tipo cíclico en la cuál sería imposible encontrar un final. Esto puede presentarse en las cadenas de decaimiento y transmutación con reacciones del tipo (n, γ) y $(n, 2n)$ por ejemplo:

$$^{235}U \xrightarrow{(n,\gamma)} {}^{236}U \xrightarrow{(n,2n)} {}^{235}U \xrightarrow{(n,2n)} {}^{234}U \xrightarrow{(n,\gamma)} {}^{235}U \tag{10}$$

La única forma de linealizar una cadena cíclica consiste en truncarla. Así, volviendo al caso de que después del elemento X_{19} aparaciera nuevamente el X_1, lo más recomendable sería seguir solo algunos elementos de esta cadena línea, e ignorar el resto, para de esta manera poder utilizar la ecuación (9). En la Figura 4 se muestra la linealización de la red de decaimiento y transmutación que aparece en la Figura 3. En dicha figura se indican las razones de ramificación, así como las constantes de decaimiento totales. Dado que hay núclidos que sólo tienen un tipo de reacción o decaimiento (por ejemplo, el núclido X_4 y X_5), esto implica que algunas razones de decaimiento son iguales a 1.

Cada una de las cadenas lineales se conoce como trayectorias, y pueden resolverse de forma independiente utilizando la ecuación (9). Este método recibe el nombre de Análisis de Trayectoria de Transmutación (TTA por el acrónimo en inglés), y sigue utilizándose en la actualidad, considerándose un método complementario [6] del novedoso método matricial CRAM. Finalmente, para que el proceso de linealización funcione, es necesario volver a usar un método de superposición y discriminación que sólo tome en cuenta ciertas contribuciones, para evitar que se esté sobrevalorando el resultado final.

4. ALGORITMOS DE BÚSQUEDA DFS PARA LA CONSTRUCCIÓN DE TRAYECTORIAS

Algunos autores [3] [4] sugieren utilizar el método de búsqueda en profundidad DFS para poder construir las trayectorias de una red de decaimientos y de transmutación. Dicho algoritmo consiste en recorrer una red creando vectores, cuyos elementos serán justamente los núclidos que vayan encontrándose en la trayectoria.

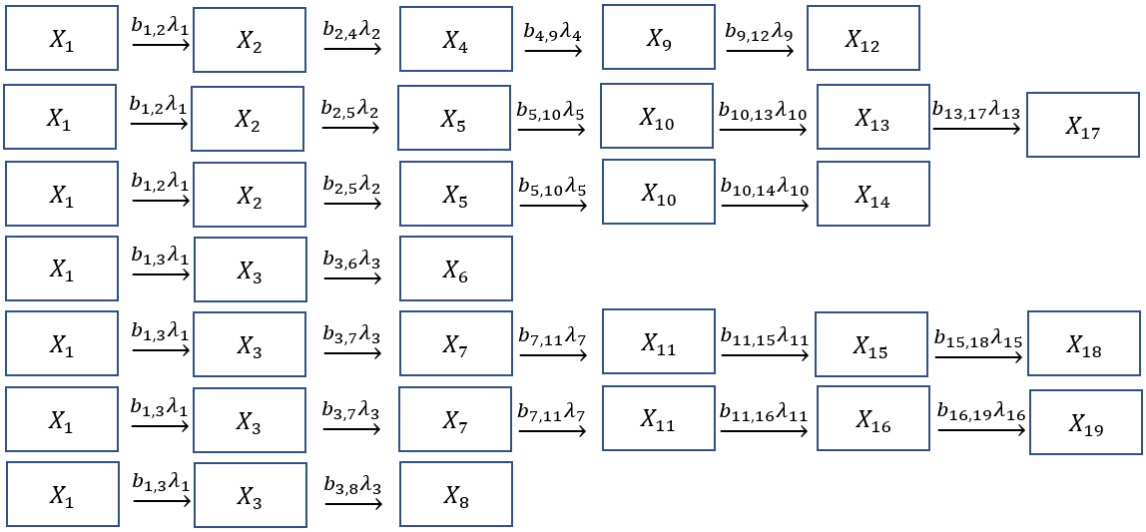

Figura 4. Linealización de cadenas de la red de decaimientos que se muestra en la Figura 3.

Comienza en el inicio de la red (por ejemplo, el elemento X_1 de la Figura 3), preguntando si dicho elemento tiene ramificaciones - es decir, si no es estable. Si la respuesta es negativa, esto implicará que se trata de una "hoja", y por lo tanto es el final del algoritmo. Si por el contrario el núclido no es estable, entonces tendrá un determinado número de "hijos" o "ramas" y cada una de éstas últimas dará lugar nuevamente a la búsqueda.

Si el algoritmo continúa, entonces seleccionará una de las ramas hijas, por ejemplo: el elemento X_2, almacenando a la otra rama X_3 al principio de una lista o pila que nos indicará qué elementos hace falta explorar. Luego, de forma recursiva buscará las ramas hijas de X_2. Si resulta que no existen éstas, entonces el algoritmo termina nuevamente. En caso contrario, vuelve a tomar una de las ramas hijas y continúa hacia adelante o a mayor profundidad, agregando al vector solución cada elemento que se encuentre en las paradas que realice.

Cuando el algoritmo termina en una hoja, todos los elementos almacenados en el vector serán una trayectoria o cadena lineal. Después viene un proceso de retroceso o "backtracking" [10], donde regresamos en nuestros pasos hasta encontrar una rama que tenga hijos que aún falte explorar. Esta búsqueda se hace en una pila o lista que almacene las "ramificaciones" inexploradas.

Por lo discutido en secciones anteriores sobre las cadenas cíclicas, es fácil saber cuándo el DFS no tendrá fin. Es otras palabras, el problema de Halting [10] se puede resolver fácilmente para nuestro caso, ya que si no existen cadenas cíclicas nuestra búsqueda siempre continuará. En caso contrario es necesario considerar un límite en la profundidad.

Existen una variedad de estudios sobre el DFS que involucran tópicos como la eficiencia, la programación en paralelo y la redistribución de grafos [11] [12] [13] [14]. Es claro que otra línea de investigación sobre la construcción de trayectorias puede involucrar la implementación de las mencionadas mejoras, así como de las nuevas técnicas de programación en paralelo.

5. DESARROLLO DE UN NUEVO ALGORITMO PARA LA CONSTRUCCIÓN DE TRAYECTORIAS

Es posible saber "a priori" si un elemento es estable o no. Solo basta construir una lista o diccionario con la información de cada núclido extrayéndola de librerías de datos nucleares como la ENDF o la JEFF. Por lo tanto, cuando se ocupa el DFS resulta redundante preguntarse en cada paso sobre la estabilidad o ramificación de cada núclido, puesto que esta información puede disponerse desde el inicio. Además, a diferencia de otros problemas sobre grafos o redes, en nuestro caso es la propia información de los núclidos la que nos permite en primer lugar construir las redes de decaimiento y transmutación en la que estemos interesados. Es decir, no contamos originalmente con la red (como sucede en otros problemas donde se ha transformado un problema de la vida real al campo de los árboles o los grafos), sino que podemos irla construyendo con la información de las librerías antes mencionadas.

En el presente método se construye un vector con la información con la que se cuenta "a priori", y se construyen las relaciones entre "padres" e "hijos" a través de una notación que utiliza guiones y dígitos. Particularmente dicho algoritmo puede ser programado en cualquier lenguaje de programación que use variables de clase "string" o cadena, junto con las operaciones o métodos tales como concatenación y separación de dicha clase.

5.1 Colapso de la Información y Generación de Índices.

Se ejemplificará el algoritmo desarrollado con la red de decaimientos y transmutación representada en la Figura 3. Comencemos construyendo dos vectores. En el primero de ellos se agrega el primer núclido de la red, y en el segundo se agrega una cadena de texto con la forma "$P0$":

$$\text{Vector1} = [X_1]; \qquad \text{Vector2} = [\text{"P0"}], \tag{11}$$

Luego se definen dos funciones: "Agregaíndices" y "Agregaelementos", las cuáles se muestran a en las siguientes páginas. La primera función construirá una notación en la que radicará la mayor parte del trabajo de la construcción de la trayectoria.

Si aplicamos por primera vez la función "Agregaelementos" al Vector1, el resultado es el siguiente:

$$\text{Vector1} = [X_1, X_2, X_3]; \qquad \text{Vector2} = [\text{"P0","P0-1","P0-2"}] \tag{12}$$

Nótese que a través del Vector2 podremos identificar quiénes son los descendientes o sucesores de X_1. Para ello basta identificar la posición que tiene X_1 en Vector1, y a través de ella considerar el elemento "notación" en el Vector2; el cual estará en la misma posición, puesto que existe una biyección de la forma:

$$f: \text{Vector1} \rightarrow \text{Vector2} \tag{13}$$

Donde la regla de correspondencia será justamente el índice (o posición) en cada uno de los vectores. Una vez identificado el elemento correspondiente en Vector2, el número de guiones indica la "generación", y los números indicarán a quién pertenece el descendiente.

Agregaíndices

La ***entrada*** son dos números enteros tomados como índices. Uno denotado por k y otro por j
La ***respuesta*** genera una nueva cadena de texto y la almacena en Vector2

PASO 1 Toma el elemento del Vector2 en la posición k
PASO 2 Crea la cadena de texto " $-j$"
PASO 3 Concatena la cadena creada en el PASO 2 al elemento del Vector2 en la posición k. Es decir:

$$\text{Vector2[k]} + " - j"$$

PASO 4 Almacena el resultado del PASO 3 en el Vector2. Es decir:

$$\text{Vector2} \leftarrow \text{Vector2[k]} + " - j"$$

Aplicando nuevamente "Agregaelementos", el resultado será el siguiente:

$$\text{Vector1} = [X_1, X_2, X_3, X_4, X_5]; \tag{14}$$

$$\text{Vector2} = \text{["P0","P0-1","P0-2","P0-1-1","P0-1-2"]}$$

Si nos interesara la trayectoria en la que estuviese un elemento, solo basta obtener su posición a través de la función (13), luego es posible obtener sus antecedentes y descendientes siguiendo los guiones e índices. Así sería relativamente sencillo retroceder y obtener la trayectoria:

$$("\text{P0}") \rightarrow ("P0-1") \rightarrow ("P0-1-2") \tag{15}$$

Aplicando una vez más el procedimiento se obtiene:

$$\text{Vector1} = [X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8]; \tag{16}$$

$$\text{Vector2} = \text{["P0","P0-1","P0-2","P0-1-1","P0-1-2","P0-2-1","P0-2-2","P0-2-3"]}$$

Debe notarse que la red de decaimiento no se recorre en profundidad. Particularmente ya aparecen "descendientes" de la ramificación X_3, pero el Vector2 de índices permite identificarlos claramente. Aplicando 10 veces el procedimiento obtenemos los siguientes vectores:

$$\text{Vector1} = [X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, \text{Fin}, X_{11}, \text{Fin}, X_{12}, X_{13}, X_{14}];$$

$$\text{Vector2} = \tag{17}$$

["P0", "P0-1", "P0-2", "P0-1-1", "P0-1-2", "P0-2-1", "P0-2-2", "P0-2-3", "P0-1-1-1", "P0-1-2-1", "P0-2-1-1", "P0-2-2-1", "P0-2-3-1", "P0-1-1-1-1", "P0-1-2-1-1", "P0-1-2-1-2"]

Agregaelementos

La ***entrada*** es el elemento j del Vector1, donde j es un índice
La ***respuesta*** agrega elementos al Vector1 e invoca la función **Agregaíndices**

Si Vector1[j] es distinto de "Fin":

- **PASO 1** Consulta los "hijos" del elemento j del Vector1
- **PASO 2** Cuenta el número de hijos y define este valor como n
- **Si** n=0**:**
  - Agrega "Fin" al Vector1:

    $$\text{Vector1} \leftarrow \text{"Fin"}$$

  - Ejecuta la función **Agregaíndices** con valores de entrada j e i:

    $$\text{Agregaíndices} \quad (j, i)$$

- **Si** n es distinto de cero:
  - **Para** $i = 1, \ldots, n$:
    - Agrega el hijo número i (de un total de n) del elemento j del Vector1 al Vector1:

      $$\text{Vector1} \leftarrow \text{hijo i del elemento Vector1[j]}$$

    - Ejecuta la función **Agregaíndices** con valores de entrada j e i:

      $$\text{Agregaíndices} \quad (j, i)$$

Por supuesto, se puede detener el algoritmo en el momento en el que se desee, aunque se deben establecer subrutinas para que el recorrido de la red sea uniforme (y no suceda, por ejemplo, que una rama se haya explorado en más detalle con la otra).

Finalmente, el Vector1 y el Vector2 contienen la información colapsada de la red. Y como se verá más adelante, podrán ser utilizados para crear un nuevo formato que permite reconstruir la estructura y que facilite la linealización y la búsqueda de trayectorias.

5.2 Ordenamiento y Formato a Base de Guiones.

En ejemplos anteriores se esquematizó cómo buscar ascendientes y descendientes de un elemento utilizando el Vector2. El proceso en esencia usa números y guiones para seguir ramificaciones. Así, dado un elemento X_k con índice P0-$\ldots -n-m-j$, el primer paso para encontrar su antecesor y sucesor inmediato será buscar en el Vector2 los elementos:

$$"P0-\cdots-n-m" \quad \text{y} \quad "P0-\cdots-n-m-j-1" \tag{18}$$

Una vez que se encuentren estos elementos, a través de sus índices pueden encontrarse sus partes correspondientes en el Vector1, lo cual permitirá trazar las trayectorias. Sin embargo, este proceso de búsqueda puede volverse lento e inclusive difícil de programar, y quizá no tendría ninguna ventaja sobre el método de búsqueda en profundidad y de hecho carecería de interés como algoritmo computacional de no existir una forma más fácil y rápida de ordenar los elementos.

La dificultad y el costo en tiempo resultan de la necesidad de descomponer los índices, que son variables del tipo "string", separar por guiones, contar elementos y el propio ordenamiento una vez que se encuentren los antecesores y sucesores. Y es en este punto donde el algoritmo adquiere una de sus principales ventajas. Sucede que por la forma en cómo se construyeron los índices, y usando funciones de ordenamiento y clasificación del tipo "sort", es posible ordenar el Vector2 sin descomponer la cadena, ni contar, ni comparar. Únicamente se necesita ordenar en base a su valor lexicográfico que por lo regular hace uso de ordinales bajo la definición del formato ASCII.

Por ejemplo, la cadena "A" tiene un ordinal de 65, la cadena "4" (es decir, el 4 visto como "string" y no como número) tiene un ordinal 54, el guion "-" tiene un ordinal 45 y así sucesivamente. La mayoría de los lenguajes de programación [15] utilizan un criterio de producto cartesiano para realizar esta comparación.

Así, si se cuenta con dos conjuntos A y B parcialmente ordenados, el ordenamiento lexicográfico sobre las parejas (a,b) y (a',b') del producto cartesiano $A\times B$ (con $a,a'\in A$ y $b,b'\in B$) se define como:

$$(a,b)\leq(a',b') \text{ si y solo sí } a<a' \text{ ó } (a=a' \text{ y } b\leq b') \tag{19}$$

Por lo anterior, usando una función nativa de los lenguajes de programación, es posible ordenar el Vector2 (denotándolo como Vector2* bajo el ordenamiento) como:

$$\text{Vector2}^* =$$

["P0", "P0-1", "P0-1-1", "P0-1-1-1", "P0-1-1-1-1", "P0-1-2", "P0-1-2-1"," "P0-1-2-1-1", "P0-1-2-1-2", "P0-2", "P0-2-1", "P0-2-1-1", "P0-2-2", "P0-2-2-1", "P0-2-3", "P0-2-3-1"]

Conviene ordenar el Vector1 bajo este esquema, utilizando la función definida en (13) y definiendo una tercera función de **Acomodo** que se muestra en la siguiente página. Esta última función vincularía el orden de Vector2* proyectándolo al orden de Vector1. Como comentario final en esta sección, habría que señalar que la función biyectiva f suele estar definida en términos del índice. Por ejemplo, en algunos lenguajes de programación existen funciones como "index" que devuelve el índice en una sucesión, cadena, lista o secuencia.

5.3 Propiedades de la Notación y Algoritmo para Construir las Trayectorias

Por la forma en cómo se construyó la notación y bajo la definición (19) del orden lexicográfico, la notación propuesta del Vector2* presenta varias características que permiten construir las trayectorias de una forma más eficiente:

1) En una trayectoria, el número de guiones que tienen sus elementos es creciente. Así, si existen dos elementos consecutivos con un número creciente de guiones, entonces estos pertenecen a la misma trayectoria.
2) Si existen k elementos consecutivos con el mismo número de guiones, y consideramos que el último elemento tiene un índice j, esto implica que se duplicarán $k-1$ veces los segmentos de las trayectorias anteriores a la posición j.
3) Si el número de guiones disminuye entre dos elementos consecutivos j y k, entonces el elemento j es el último elemento de la trayectoria en curso, y el elemento k es el nodo de la siguiente trayectoria.

Estas características pueden deducirse y demostrarse usando la definición (19) y la naturaleza de los sucesores y antecesores de una red de decaimientos y transmutación. Estas tres propiedades ofrecen valiosa información únicamente al contar el número de guiones de los elementos del Vector2*. Al recorrer este vector se presentan sólo 3 casos en cuanto al número de guiones entre dos elementos consecutivos: que éste aumente, que sea el mismo o que disminuya.

Para la propiedad 3), cuando el número de guiones entre elementos consecutivos disminuye, estaremos encontrando "fragmentos de trayectorias". Estos fragmentos sólo estarán completos si comienzan en "P0".

En nuestro ejemplo anterior, usando el número de guiones pueden obtenerse los siguientes fragmentos de trayectorias que se encuentran en la Tabla I.

Tabla I. Fragmentos de trayectorias.

	Fragmento de Trayectoria de Vector2*	Equivalente en Vector1*
1	["P0", "P0-1", "P0-1-1", "P0-1-1-1", "P0-1-1-1-1]	$[X_1, X_2, X_4, X_9, X_{12}]$
2	["P0-1-2", "P0-1-2-1"," "P0-1-2-1-1","P0-1-2-1-2]	$[X_5, X_{10}, X_{13}, X_{14}]$
3	["P0-2", "P0-2-1", "P0-2-1-1"]	$[X_3, X_6$,Fin]
4	["P0-2-2", "P0-2-2-1"]	$[X_7, X_{11}]$

Se puede apreciar que sólo el primer fragmento está completo. Al resto le hacen falta elementos, que pueden "integrarse" o "agregarse" usando los guiones. Es posible identificar los fragmentos incompletos como aquellos que no comienzan con $P0$.

La propiedad 2 representa una ventaja sobre el método de búsqueda en profundidad. Resulta que cuando en un fragmento de cadena tenemos guiones repetidos una cantidad k de ocasiones en la posición j, esto implica que $k-1$ trayectorias pueden generarse, cuyo cuerpo "común" serán los elementos de las posiciones menores a j. Por ejemplo, en la segunda fila de la Tabla I se repite el número de guiones para los dos últimos elementos. Esto significa que estos dos elementos tienen el mismo "tronco", es decir:

["P0-1-2", "P0-1-2-1"] (20)

Ahora, para el último de los elementos con el mismo número de guiones repetidos (esto es "P0-1-2-1-2") pueden presentarse dos casos. El primero es que el elemento que venga a continuación de él tenga un número mayor de guiones.

Acomodo

La ***entrada*** es el Vector2*
La ***respuesta*** genera un Vector1* que contiene los mismos elementos que Vector1, pero en el orden obtenido a través de la notación de Vector2*

PASO 1 Determina la longitud del Vector2*, y denomina este número m
Para $i = 1, \ldots . m$:

- Encuentra la posición del elemento Vector2*[i] en Vector2, y denomina a este índice j
- Utiliza la función biyectiva f (Ecuación (16)) en el elemento Vector2[j], para obtener de esta forma el elemento Vector1[j]. Es decir:

$$f(\text{Vector2[j]}) = \text{Vector1[j]}$$

- Agrega este elemento al Vector1*

El segundo es que tenga un número menor. En el primero de los casos, significaría que la trayectoria a la cual pertenece "P0-1-2-1-2" continúa. En el segundo de los casos implicaría que "P0-1-2-1-2" es el último elemento de la trayectoria. En cualquier caso, los elementos anteriores al último generarían trayectorias donde cada uno de los elementos sería el final de la cadena lineal. Aplicando estas dos propiedades, las trayectorias obtenidas serían las que se muestran en la Tabla II.

Tabla II. Trayectorias finales

	Fragmento de Trayectoria de Vector2*	Equivalente en Vector1*
1	["P0", "P0-1", "P0-1-1", "P0-1-1-1", "P0-1-1-1-1]	$[X_1, X_2, X_4, X_9, X_{12}]$
2	["P0", "P0-1" ,"P0-1-2", "P0-1-2-1"," "P0-1-2-1-1"]	$[X_1, X_2, X_5, X_{10}, X_{13}]$
3	["P0", "P0-1" ,"P0-1-2", "P0-1-2-1"," "P0-1-2-1-2"]	$[X_1, X_2, X_5, X_{10}, X_{14}]$
4	["P0" , "P0-2", "P0-2-1", "P0-2-1-1"]	$[X_1, X_2, X_5, X_{10}, X_{13}]$
5	["P0" , "P0-2" ,"P0-2-2", "P0-2-2-1"]	$[X_1, X_3, X_6$,Fin]

5.4 Comparación con el elemento de Búsqueda profunda.

En términos computacionales, una adecuada comparación entre algoritmos suele hacer uso de la notación de Bachmann-Landau [10], también conocida como notación de cota superior asintótica y un análisis matemático-computacional. Dicha notación se usa para evaluar el tiempo de ejecución, así como para analizar las variables de almacenamiento y espacio en el caso de que los parámetros de entrada crezcan.

También es importante analizar si dicho algoritmo es estable, su recursividad y si es adaptable a la mayoría de los lenguajes de programación. Estos tópicos están fuera del alcance de este primer estudio exploratorio, y a lo más puede proponerse una comparación práctica (aunque en términos

de la matemática computacional, poco rigurosa) con base en las operaciones y comparaciones que realiza. Por ello consideramos este algoritmo como "alternativo" por ahora.

Es claro que una de las primeras ventajas que tiene este algoritmo es la notación que propone, en la que se pueden identificar elementos finales de las cadenas, trayectorias que tienen el mismo tronco, e inclusive el número de cadenas lineales generadas. Esto último se puede obtener de manera sencilla a partir de las propiedades mencionadas en la sección anterior. Cuando el número de guiones disminuya, entonces tenemos trayectorias distintas. Cuando el número de guiones se repita k veces, tendremos $k-1$ trayectorias que se generarán.

En computación existen varios formatos para representar estas estructuras que por lo regular reciben el nombre de árboles. Uno de los formatos más populares es el formato de Newick, que tiene múltiples características en cuanto a la identificación de sus elementos [9]. Así que la presente notación también viene a ser alternativa.

Otro resultado que podemos obtener de nuestra notación y formato es el número total de cadenas lineales que podemos extraer. Si denotamos como α el número de veces en que los elementos disminuyen de guiones, y existen m ocasiones en que se repiten los guiones, denotando $m_1, m_2, \dots, m_m$ la cantidad en que se repiten estos elementos, entonces el número total de cadenas lineales puede estimarse como:

$$\alpha + \sum_{i=1}^{m} m_i \tag{21}$$

Podemos realizar una comparación sencilla en cuanto a las operaciones realizadas, contra el método de búsqueda en profundidad. En primer lugar, este último método de búsqueda en profundidad realiza operaciones del tipo "if", para determinar si el elemento tiene sucesores o es estable. Esta operación es idéntica al algoritmo propuesto, que evalúa si el elemento agregado es distinto de "Fin" (véase el algoritmo de **"Agregaelementos"**). Sin embargo, el algoritmo propuesto realiza esta evaluación de forma lineal, siempre hacia adelante, mientras que el algoritmo de búsqueda en profundidad tiene que regresar y evaluar todos aquellos elementos que tienen ramificaciones, lo cual consume una mayor cantidad de tiempo.

Adicionalmente, gracias a la propiedad 2, una vez identificados los elementos donde los guiones se repiten, es posible realizar la "copia" de las trayectorias que comparten el mismo "tronco" de manera eficiente. Mientras que, con el método de búsqueda en profundidad, en cada caso tendría que evaluarse si cada elemento tiene un sucesor (aun cuando dicha información por lo regular ya se conoce "a priori"). En el proceso de retroceso o "backtracking", el método de búsqueda en profundidad debe identificar las ramificaciones que aún no se exploran e intentar cada una de ellas. En el caso del algoritmo desarrollado, esta información ya se encuentra resumida una vez que se ha aplicado el método de ordenamiento lexicográfico, que en un primer intento genera los fragmentos de trayectoria que aparecen en la Tabla I.

Además, en este punto el algoritmo propuesto tiene posibles mejoras en cuanto a su velocidad, considerándose de hecho el corazón del método el ordenamiento. Por ejemplo, si en lugar de usar una función estándar de ordenamiento, se usan las múltiples técnicas que se han desarrollado hasta ahora (por ejemplo el método *Quicksort*, que tiene un esquema de partición que podría fácilmente

adaptarse a la estructura propuesta), es claro que podría aumentarse la velocidad de manera significativa.

En las siguientes dos gráficas comparativas se realizaron ejecuciones del algoritmo de búsqueda en profundidad y del algoritmo propuesto para 2 casos particulares, usando el lenguaje de programación Python 2.7. Nótese que no se proponen tiempos de ejecución, porque esto está fuertemente condicionado por el hardware y arquitectura de un equipo, optándose mejor por ciclos (definidos fácilmente como una base normalizada), y dado que la red de decaimientos y transmutación puede afectar el resultado según su topología, se ha optado por considerar como variable la extensión de las cadenas lineales para una red que comienza en el U-235.

Los casos a) y b) muestran los ciclos en base a la profundidad para cadenas no uniformes (es decir, cuya longitud no es fija), para el método de búsqueda en profundidad y el método propuesto El caso a), no incluye cadenas cíclicas, el caso b) sí. En las comparaciones se aprecia un desempeño excelente del método propuesto, aunque como se dijo en líneas anteriores, hace falta un análisis matemático-computacional que incluya la notación Bachman-Landau, y responda si el método es estable o no.

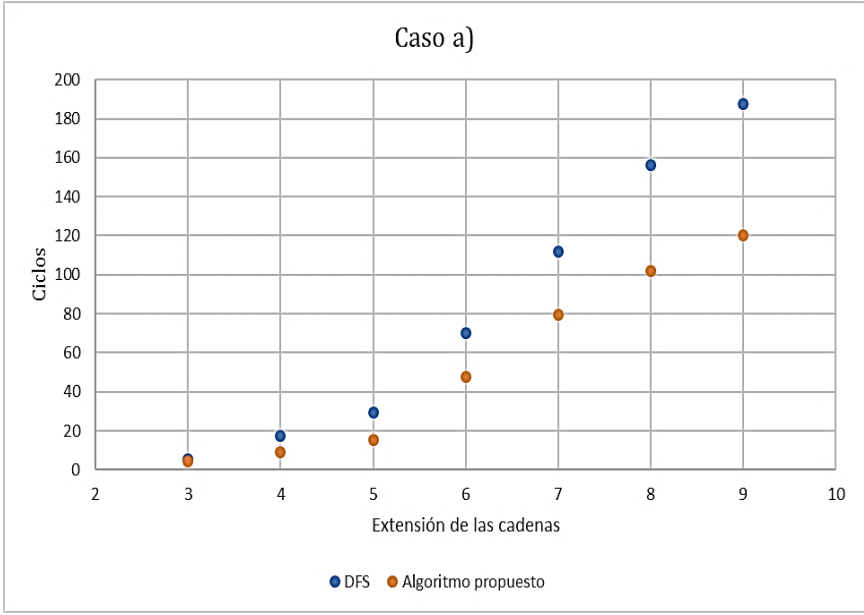

Figura 5. Gráfico comparativo entre el algoritmo propuesto y el DFS, para el caso a), donde no se consideran cadenas cíclicas. Las variables analizadas son los ciclos (tiempo) empleados y la extensión de las cadenas linealizadas.

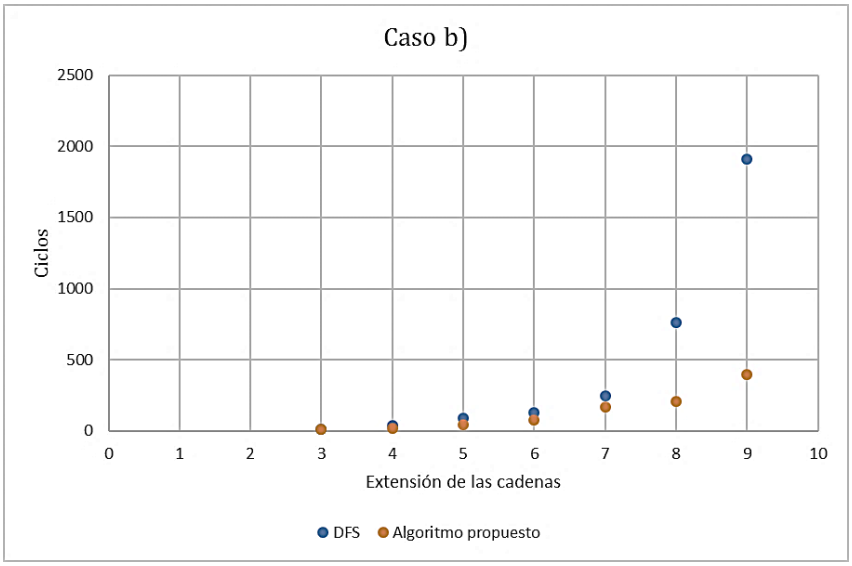

Figura 6. Gráfico comparativo entre el algoritmo propuesto y el DFS, para el caso b) donde se incluyen cadenas cíclicas. Las variables analizadas son los ciclos (tiempo) empleados y la extensión de las cadenas linealizadas.

4. CONCLUSIONES

Se desarrolló un método alternativo para la construcción de cadenas lineales que parte de información "a priori" obtenida de librerías. El algoritmo presenta características que le brindan ventajas en comparación con el método de búsqueda profunda, entre las que se encuentran: 1) la ausencia de un proceso de retroceso y por lo tanto una optimización en el tiempo de cómputo, 2) el uso de una notación que brinda información sobre la topología de la red de decaimiento y transmutación, y 3) la identificación de ramificaciones que aceleran el proceso de "repetición" de trayectorias. Se realizó una comparación en términos de tiempo de cómputo (que normalizado se ha traducido a ciclos), utilizando un ejemplo particular de una red de decaimiento obtenida a partir del U-235. La comparación está en función de la extensión de las cadenas lineales, sin considerar cadenas cíclicas para un caso, e incluyéndolas en otra. En ambos casos el método propuesto muestra una ventaja que es más pronunciada para el caso donde se incluyen cadenas cíclicas.

Aún es necesario indagar más, utilizando técnicas de conteo computacionales, y la notación de cota superior asintótica, concluyendo por el momento que este método es alternativo y bajo ciertas circunstancias más rápido que el método de búsqueda en profundidad.

AGRADECIMIENTOS

Al Consejo Nacional de Ciencia y Tecnología por brindar el apoyo económico a C.A. Cruz para la realización de este trabajo que forma parte de su investigación doctoral. Los autores agradecen el apoyo financiero recibido del proyecto estratégico No. 212602: "AZTLAN Platform: Desarrollo

de una plataforma mexicana para el análisis y diseño de reactores nucleares", del Fondo Sectorial de Sustentabilidad Energética CONACYT - SENER.

REFERENCIAS

1. Cacucci, Dan Gabriel (Editor); *Handbook of Nuclear Engineering*; Editorial Springer, **Vol. 1**; Karlsruhe, Germany (2010).
2. Bateman, Harry; "Solution of a system of differential equations occurring in the theory of radiactive transformations", Cambridge Philos. Soc., **Vol. 15**, p.423-427(1910).
3. Isotalo, Arno; "Computational Methods for Burnup Calculations with Monte Carlo Neutronics", Aalto University publications series; Doctoral Dissertation; Aalto University, Finlandia (2013).
4. Logan J.; Harr; "Precise Calculation of complex radioactive decay chains"; Air Force Institute of Technology; Master of Science Dissertation; Wright-Patterson Air Force Base, Ohio; Estados Unidos (2007).
5. Cetnar, Jerzy; "General solution of Bateman equations for nuclear transmutations", *Annals of Nuclear Energy,* **Vol. 33,** p. 640-645 (2006).
6. Huang, Kai; Wu, Hongchun; Cao, Liangzhi; Li, Yunzhao; Shen, Wei; "Improvements to the Transmutation Trajectory Analysis of depletion evaluation", *Annals of Nuclear Energy,* **Vol. 87,** p.637-647 (2015).
7. Pelowitz, Denise B. (Editor); *MCNPX USER'S MANUAL,* Los Alamos National Laboratory; Nuevo México, Estados Unidos (2008).
8. García-Herranz, Nuria; Herrero C., José J.; Cabellos De F., Oscar L.; "Optimization of multidimensional cross-section tables for few group calculations" *Annals of Nuclear Energy,* **Vol. 69,** p. 226-237 (2014).
9. Junier, Thomas; *Newick Utilities Tutorial,* Computational Evolutionary Genomics Group; Genova, Suiza (2009).
10. Atallah, Mikhail; Blanton, Marina; *Algorithms and Theory of Computation Handbook*, CRC Press, Florida, Estados Unidos (2009).
11. Holzmann, Gerald J.; Peled, Doron; Yannakakis, Mihalis; "On Nested Depth First Search", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. **Vol. 32**, p.23-31. (1997).
12. Cidon, Israel; "Yet another distributed depth-first-search algorithm", *Information Processing Letters*, **Vol. 26**, p.301-305 (1988).
13. Makki, S.A.M.; Havas, George, "Distributed algorithms for depth-first search", *Information Processing Letters,* **Vol. 60**, p.7-12 (1996).
14. Sharman, Mohan B.; Iyengar, Sitharama S.; "An efficient distributed depth-first-search algorithm", *Information Processing Letters,* **Vol.32**, p.183-186 (1989).
15. Harzheim, Egbert; *Ordered Sets (Advances in Mathematics)*, Springer, Germany (2005).