# Hacia una Estrategia Óptima para la Depuración Algorítmica[1]

David Insa[2] Josep Silva[3]

*Departamento de Sistemas Informáticos y Computación*
*Universitat Politècnica de València*
*E-46022 Valencia, Spain.*

**Abstract**

Durante la depuración algorítmica el usuario debe contestar preguntas sobre la validez de las computaciones hechas por el programa que se está depurando. Reducir el número de preguntas realizadas es fundamental para mejorar el tiempo total de depuración, y por esta razón se ha investigado mucho en estrategias de depuración que tratan de minimizar el número de preguntas hechas por el depurador. Durante tres décadas se ha pensado que la estrategia Divide & Query introducida por Shapiro es óptima si sólo se considera la estructura del árbol de ejecución usado durante la depuración. En este artículo demostramos que Divide & Query no es óptimo e introducimos las bases para definir una estrategia óptima.

*Keywords:* Depuración de software, Depuración Algorítmica, Divide & Query.

## 1 Introducción

La *depuración algorítmica* [10] es una técnica de depuración semi-automática que ha sido extendida prácticamente a todos los paradigmas [11]. La técnica se basa en las respuestas del programador a una serie de preguntas generadas automáticamente por el depurador algorítmico. Las preguntas siempre están relacionadas con la validez de una (sub)computación realizada con unos valores de entrada determinados. Las respuestas le indican al depurador la validez de las (sub)computaciones del programa; y el depurador utiliza esta información

[1] Este trabajo ha sido parcialmente subvencionado por el *Ministerio de Ciencia e Innovación* con referencia TIN2008-06622-C03-02 y por la *Generalitat Valenciana* con referencia PROMETEO/2011/052.

[2] Email: dinsa@dsic.upv.es

[3] Email: jsilva@dsic.upv.es

*This paper is electronically published in*
*Electronic Notes in Theoretical Computer Science*
*URL:* www.elsevier.nl/locate/entcs

para guiar la búsqueda del error hasta que se aísla la porción de código que lo contiene.

**Ejemplo 1.1** Considérese este programa escrito en Haskell que ordena una lista utilizando el algoritmo de ordenamiento MergeSort.

```
main = mergeSort [2,1,3]

merge [] lista = lista
merge lista [] = lista
merge (x:xs) (y:ys) | x <= y    = x : merge xs (y:ys)
                    | otherwise = merge (x:xs) ys

mergeSort [] = []
mergeSort [x] = [x]
mergeSort lista = merge (mergeSort izq) (mergeSort der)
          where mitad = (div (length lista) 2)
                izq = take mitad lista
                der = drop mitad lista
```

A continuación veremos una sesión de un depurador algorítmico para este programa (las respuestas `SÍ` y `NO` son proporcionadas por el programador):

```
Empezando la Sesión de Depuración...
(1)  mergeSort [2,1] = [2]? NO
(2)  merge [2] [1] = [2]? NO
(3)  merge [2] [] = [2]? SÍ

Error encontrado en la regla:
merge (x:xs) (y:ys) | otherwise = merge (x:xs) ys
```

El depurador muestra la parte del código que contiene el error. En este caso debería ser `otherwise` = `y` : .... Nótese que para depurar el programa el programador solo contesta preguntas; no es necesario que vea el código.

Normalmente, los depuradores algorítmicos tienen un *front-end* que produce una estructura de datos que representa la ejecución de un programa—el llamado *árbol de ejecución* (ET) [8]—; y un *back-end* que utiliza el ET para hacer preguntas y procesar las respuestas del programador para encontrar el error. Por ejemplo, el ET del programa del Ejemplo 1.1 está representado en la Figura 1.

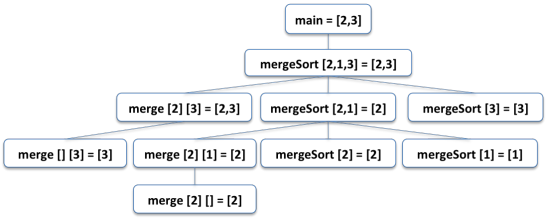



Fig. 1. ET del programa del Ejemplo 1.1

La estrategia usada para decidir cuáles son los nodos del ET que deben ser preguntados es crucial para el rendimiento de la técnica. Desde la definición de la depuración algorítmica, ha habido mucha investigación respecto a la definición de nuevas estrategias con el fin de minimizar la cantidad de preguntas [11].

En la práctica, la estrategia que necesita realizar menos preguntas para encontrar el error es la versión mejorada de Divide & Query propuesta por Hirunkitti y Hogger [4]. Desde su definición hace más de tres décadas, esta estrategia ha sido considerada óptima con respecto al número de preguntas generadas. Por esta razón, ha sido implementada en practicamente todos los depuradores algorítmicos (véase, e.g., [9,1,3,7,2,5]). En este árticulo mostramos que esta estrategia no solamente no es óptima, sino que no es ni completa ni correcta. Para ello presentamos contraejemplos que lo demuestran y explicamos las causas en cada uno de ellos. En una segunda parte, introducimos los principios que debe seguir una estrategia óptima, y proporcionamos una aproximación a ésta.

El resto del artículo ha sido estructurado de la siguiente manera. En la Sección 2 recordamos y formalizamos la estrategia D&Q y mostramos con contraejemplos que no es correcta ni completa. A continuación, en la Sección 3, mostramos los principios que debe seguir una estrategia óptima y proporcionamos una aproximación a la misma. En la Sección 4 mostramos el trabajo futuro. Finalmente, la Sección 5 presenta las conclusiones.

## 2 Divide & Query

En esta sección formalizamos la estrategia D&Q inicialmente definida por Shapiro [10] y posteriormente mejorada por Hirunkitti y Hogger [4]. Empezamos por la definición de *árbol de ejecución marcado*, el cual es un ET donde algunos nodos pueden haber sido podados porque han sido marcados como correctos (i.e., contestados SÍ), algunos nodos pueden haber sido marcados como incorrectos (i.e., contestados NO) y la validez del resto de nodos está indefinida.

**Definición 2.1** [Árbol de Ejecución Marcado] Un *Árbol de Ejecución Marcado* (MET) es un árbol $T = (N, E, M)$ donde $N$ son los nodos, $E : N \times N$ son los arcos, y $M : N \to V$ es una función de marcado que asigna a todos los nodos de $N$ un valor en el dominio $V = \{Incorrecto,\ Desconocido\}$.

Inicialmente, todos los nodos del MET están marcados como *Desconocido*. Pero con cada respuesta del usuario, se produce un nuevo MET. Concretamente, dado un MET $T = (N, E, M)$ y un nodo $n \in N$, la respuesta del usuario a la pregunta de $n$ produce un nuevo MET tal que: (i) si la respuesta es SÍ, entonces este nodo y su subárbol son podados del MET; (ii) si la respuesta es NO, entonces todos los nodos del MET son podados excepto este nodo y sus descendientes. [4]

[4] También es posible aceptar *No lo sé* como respuesta del usuario. En este caso el depurador simplemente selecciona otro nodo [5]. Por simplicidad supondremos que el usuario solamente responde *Correcto* o *Incorrecto*.

**Algorithm 1** Algoritmo general para la depuración algorítmica

**Entrada:** Un MET $T = (N, E, M)$
**Salida:** Un nodo-causa o $\perp$ si éste no existe
**Precondiciones:** $\forall n \in N$, $M(n) = Desconocido$
**Inicialización:** $nodoCausa = \perp$

**begin**
(1) **do**
(2) nodo = seleccionarNodo($T$)
(3) respuesta = preguntarNodo(nodo)
(4) **if** (respuesta = *Incorrecto*)
(5) **then** $M$(nodo) = *Incorrecto*
(6) nodoCausa = nodo
(7) $N = \{n \in N \mid (nodo \rightarrow n) \in E^*\}$
(8) **else** $N = N \backslash \{n \in N \mid (nodo \rightarrow n) \in E^*\}$
(9) **while** ($\exists n \in N, M(n) = Desconocido$)
(10) **return** nodoCausa
**end**

Por lo tanto, el tamaño del MET se reduce gradualmente con cada respuesta. Si podamos todos los nodos del MET entonces el depurador concluye que no se ha encontrado ningún error. Si, por el contrario, terminamos con un MET compuesto por un único nodo marcado como incorrecto, a este nodo se le llama *nodo-causa* y es señalado como el responsable del error del programa.

Todo este proceso está definido en el Algoritmo 1 donde la función $seleccionarNodo$ selecciona un nodo del MET para ser preguntado al usuario con la función $preguntarNodo$. Por lo tanto, $seleccionarNodo$ es el punto central de este trabajo porque implementa la estrategia de depuración algorítmica. En adelante utilizaremos $E^*$ para referirnos al cierre reflexivo-transitivo de $E$.

D&Q supone que el peso individual de un nodo es siempre 1. Por lo tanto, dado un MET $T = (N, E, M)$, el peso de un subárbol cuya raíz es $n \in N$, $w_n$, es definido como su número de descendientes incluyéndose a él mismo (i.e., $1 + \sum (w_{n'} \mid (n \rightarrow n') \in E)$.

D&Q intenta simular una búsqueda dicotómica seleccionando el nodo que mejor divide el MET en dos subMETs con pesos similares. Por lo tanto, dado un MET con $n$ nodos, D&Q busca el nodo cuyo peso más se acerque a $\frac{n}{2}$. El algoritmo D&Q selecciona siempre:

- el nodo más pesado $n'$ cuyo peso más se acerque a $\frac{n}{2}$ siendo $w_{n'} \leq \frac{n}{2}$, o
- el nodo más ligero $n'$ cuyo peso más se acerque a $\frac{n}{2}$ siendo $w_{n'} \geq \frac{n}{2}$

### *2.1 Limitaciones de Divide & Query*

Una estrategia de depuración algorítmica es óptima si el número medio de preguntas que realiza para cualquier MET es mínimo. Podemos calcular el número de preguntas realizadas suponiendo que el error puede estar en cualquier nodo del árbol y, por tanto, calculando las secuencias de preguntas

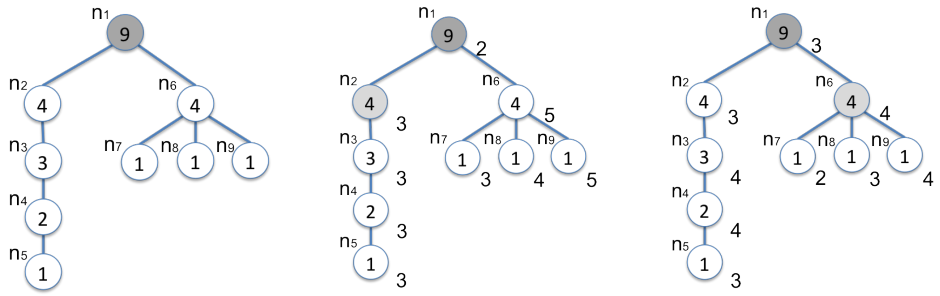



Fig. 2. Contraejemplo

que haría el depurador en cada nodo. Evidentemente, pueden existir varias estrategias óptimas diferentes. En adelante, dado un MET, llamaremos *nodo óptimo* al primer nodo preguntado por una estrategia óptima.

**Definición 2.2** [Estrategia óptima] Sea $\epsilon$ una estrategia de depuración algorítmica. Dado un MET $T = (N, E, M)$, sea $s_n^\epsilon$ la secuencia de preguntas realizadas por el Algoritmo 1 usando la estrategia $\epsilon$ y suponiendo que el único nodo-causa de $T$ es $n \in N$. Sea $t_\epsilon = \sum_{n_i \in N} |s_{n_i}^\epsilon|$. Decimos que $\epsilon$ es óptima si para cualquier MET se cumple que $\nexists \epsilon' \,.\, t_\epsilon > t_{\epsilon'}$.

En esta sección mostramos que la estrategia D&Q no es óptima. La razón es que no es cierta la hipótesis inicial de D&Q que presupone que dividir el árbol en dos subárboles con la misma cantidad de nodos convierte la búsqueda en una búsqueda dicotómica.

Para demostrarlo utilizaremos tres contraejemplos en los que veremos como la información que D&Q tiene en cuenta para seleccionar el nodo óptimo no es suficiente. Los ejemplos están basados en el coste (medido en número de preguntas realizadas por el depurador) asociado a los nodos seleccionados por D&Q. Para poder medir dicho coste, usaremos la siguiente definición de secuencia de preguntas:

**Definición 2.3** [Secuencia de preguntas] Dado un MET $T = (N, E, M)$ y dados dos nodos $n_1, n_2 \in N$, la *secuencia de preguntas* de $n_1$ con respecto a $n_2$, $sp(n_1, n_2)$, está formada por todas las preguntas realizadas por el Algoritmo 1 suponiendo que el primer nodo escogido por la función $seleccionarNodo(T)$ es $n_2$ y que el único nodo-causa de $T$ es $n_1$.

Intuitivamente, $sp(n_1, n_2)$ se compone de las preguntas que el depurador haría para determinar que $n_1$ es un nodo-causa suponiendo que la primera pregunta es la asociada al nodo $n_2$. Esto significa que la secuencia de preguntas depende completamente de la estrategia utilizada. Para el caso concreto de D&Q, en el MET de la Figura 2 (izquierda) podemos ver que:

$$sp(n_2, n_6) = [n_6, n_3, n_2] \qquad sp(n_6, n_2) = [n_2, n_6, n_7, n_8, n_9]$$
$$sp(n_2, n_2) = [n_2, n_4, n_3] \qquad sp(n_6, n_6) = [n_6, n_7, n_8, n_9]$$

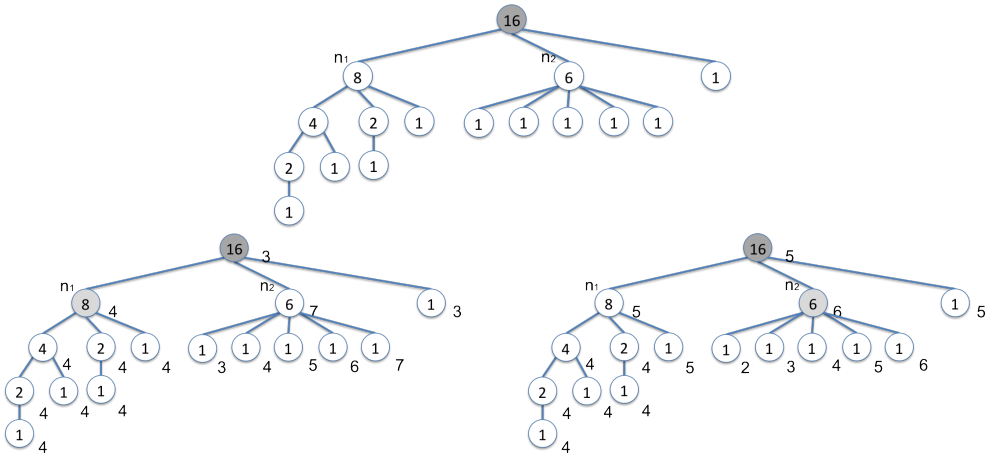


Fig. 3. Segundo contraejemplo

En los METs del centro y la derecha de la Figura 2, así como en las siguientes figuras, el número inferior derecho de un nodo representa el tamaño de la secuencia de preguntas de ese nodo con respecto al nodo gris claro. Es decir, el número de preguntas que debe hacerse para encontrar el error suponiendo que éste se encuentre en ese nodo y que se empieza a preguntar por el nodo gris claro.

**Ejemplo 2.4** Considérese el MET de la Figura 2 (izquierda). En él, el nodo raíz está marcado como incorrecto y D&Q selecciona como nodos óptimos a $n_2$ y a $n_6$. Al sumar el número de preguntas que se han hecho en el MET de la Figura 2 (centro), donde se ha empezado a preguntar por el nodo $n_2$, vemos que se hacen un total de 31 preguntas, haciendo una media de $\frac{31}{9} = 3,44$ preguntas para encontrar el error en cualquiera de los 9 nodos del árbol. Sin embargo en el MET de la Figura 2 (derecha), en el que se ha empezado a preguntar por el nodo $n_6$, se hacen 30 preguntas en total, por lo que tiene una media de $\frac{30}{9} = 3,33$ preguntas.

Puede observarse que, contrariamente a la hipótesis de D&Q, empezar seleccionando el nodo que divide el árbol en dos subárboles con la misma cantidad de nodos no es suficiente para determinar el nodo óptimo. En este ejemplo ambos nodos tienen el mismo peso (4) y ambos dividen el árbol en la misma cantidad de nodos. Pero hemos comprobado que empezando a preguntar por el nodo $n_6$ se hacen menos preguntas de media que si se empieza a preguntar por $n_2$. Siendo, por lo tanto, el nodo $n_6$ óptimo y no el nodo $n_2$. Para explicar el motivo utilizaremos el siguiente ejemplo.

**Ejemplo 2.5** En la Figura 3 podemos observar en el MET superior como el subárbol cuya raíz es $n_1$ es un (sub)MET completamente balanceado [6] (i.e., en el peor caso puede depurarse el subárbol con un número de preguntas logarítmico). Mientras que el subárbol cuya raíz es $n_2$ es un (sub)MET

completamente desbalanceado (i.e., en el peor caso habría que preguntar por todos los nodos para encontrar el error).

D&Q trata de hacer una búsqueda dicotómica preguntando por el nodo que divide el árbol en dos partes con pesos similares, en este caso $\frac{16}{2} = 8$ por lo que determina al nodo $n_1$ como nodo óptimo. En el MET de la Figura 3 (izquierda) puede verse el número de preguntas que se hacen al empezar por este nodo; un total de 70 preguntas, es decir $\frac{70}{16} = 4,38$ preguntas de media para encontrar el error en cualquiera de los 16 nodos. Sin embargo, si empezamos a preguntar por el nodo $n_2$ se hacen la misma cantidad de preguntas. Puede verse el número de preguntas en el MET de la Figura 3 (derecha). A pesar de ser igual de óptimo que $n_1$, este nodo nunca será seleccionado por D&Q puesto que su peso dista más de $\frac{16}{2}$ que el de $n_1$.

La razón por la que seleccionar el nodo $n_2$ resulta igual de óptimo que el nodo $n_1$ es muy simple. Viendo el árbol derecho de la Figura 3 podemos observar que al empezar a preguntar por $n_2$ y si el error no se encuentra en ese subárbol, la naturaleza balanceada del subárbol del nodo $n_1$ permite que, en la segunda pregunta, se divida mejor el árbol. Como puede verse comparando el árbol izquierdo con el árbol derecho de la Figura 3 la gran mayoría de los nodos del subárbol del nodo $n_1$ no incrementan su número de preguntas por haberse preguntando antes el nodo $n_2$. Sin embargo, viendo el árbol izquierdo de la Figura 3 podemos ver que al empezar a preguntar por el nodo $n_1$ y si el error no se encuentra en ese subárbol, la naturaleza amplia del subárbol del nodo $n_2$ no permite, en una segunda pregunta, dividir el árbol; teniendo que preguntar por todos los nodos para encontrar el error. Obsérvese que en este caso todos los nodos del subárbol del nodo $n_2$ sí incrementan su número de preguntas por haberse preguntando antes el nodo $n_1$.

Todo esto nos lleva a la conclusión de que no solamente es importante podar subárboles grandes del MET, sino que también es importante podar árboles desbalanceados que son más difíciles de explorar. Esto último es ignorado por D&Q y es la causa de que no sea una estrategia óptima.

Finalmente mostramos un ejemplo donde D&Q no sólo no es capaz de encontrar todos los nodos óptimos como en el caso anterior, sino que además no es capaz de encontrar ninguno. El ejemplo es el mismo árbol del Ejemplo 2.4 en el que se han añadido 59 nodos en profundidad al nodo $n_2$ y 46 nodos en anchura al nodo $n_6$.

**Ejemplo 2.6** Considérese el MET de la Figura 4 (izquierda) donde D&Q determinaría que el nodo óptimo se encuentra en la rama izquierda del árbol; concretamente el nodo $n_1$ puesto que es el nodo cuyo peso más se acerca a $\frac{114}{2} = 57$. Sin embargo, el hijo derecho $n_2$ es el nodo óptimo puesto que produce menos preguntas que el nodo $n_1$ a pesar de que su peso dista mucho más de 57.

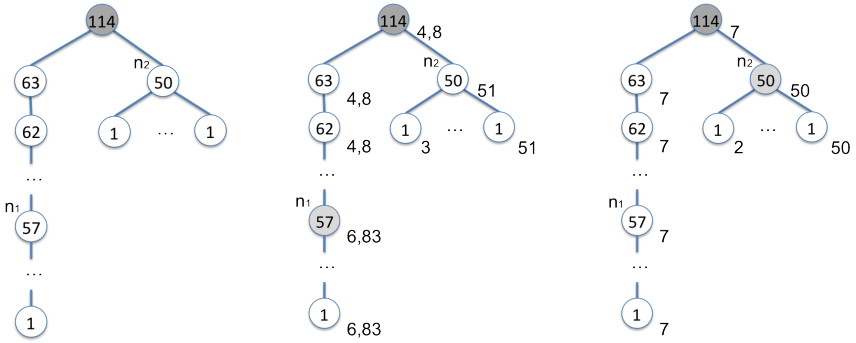



Fig. 4. Tercer contraejemplo

En el árbol central de la Figura 4 se ha empezado preguntando por el nodo que D&Q selecciona como óptimo, $n_1$. Si el error estuviera en el subárbol del nodo $n_1$, puesto que se trata de una sucesión de nodos en profundidad, se harían $\log_2 57 = 5,83$ preguntas de media a cada nodo (+1 por haber preguntado inicialmente por el nodo $n_1$). Si el error no estuviera en el subárbol del nodo $n_1$ tras preguntar por éste se empezará a explorar el subárbol del nodo $n_2$. Si el error se encuentra en ese subárbol se deberá preguntar por todos los nodos hasta encontrar el error, y todos estos nodos llevan acarreada la pregunta hecha al nodo $n_1$, en total se hacen $(\sum_{i=3}^{51} i)+51 = 1374$ preguntas. Si finalmente tampoco estuviera en la rama derecha sólo quedan los 7 nodos superiores de la rama izquierda (incluyendo la raíz). En los que se hacen $\log_2 7 = 2,8$ preguntas de media a cada nodo (+2 por haber preguntado anteriormente al nodo $n_1$ y al nodo $n_2$). En total se hacen $57 * 6,83 + 1374 + 7 * 4,8 = 1796,91$ preguntas, y una media de $\frac{1796,91}{114} = 15,76$ preguntas.

Si por el contrario empezásemos a preguntar por el nodo $n_2$ y el error se encuentra en ese subárbol, se hacen $(\sum_{i=2}^{50} i) + 50 = 1324$ preguntas puesto que no llevan preguntas acarreadas. Si el error se encuentra en la otra rama, tras preguntar por el nodo $n_2$ quedan 64 nodos en profundidad, por lo que en $\log_2 64 = 6$ preguntas (+1 por haber preguntado inicialmente por el nodo $n_2$) se encontrará el error. En total se hacen $1324 + 64 * 7 = 1772$ preguntas, y una media de $\frac{1772}{114} = 15,54$ preguntas.

Estos contraejemplos confirman que D&Q no siempre es capaz de encontrar todos los nodos óptimos en un MET, por lo que no es completo. Y además confirman que en determinados casos, D&Q selecciona un nodo que no es óptimo, por lo que no es correcto. Adicionalmente, los ejemplos han puesto de manifiesto que además de la cantidad de nodos, considerar la estructura del árbol de ejecución también es importante para obtener una estrategia que realice la menor cantidad de preguntas posibles al usuario. En la siguiente sección mostramos que dicha estrategia puede llegar a definirse puesto que encontrar todos los nodos óptimos de un MET es un problema decidible.

# 3 Una estrategia óptima para la depuración algorítmica: Divide *b*y Queries (D*b*Q)

En esta sección introducimos las bases que deben seguirse para obtener una estrategia óptima para la depuración algorítmica. En primer lugar mostramos que el problema abordado por dicha estrategia es decidible y posteriormente presentamos una aproximación del algoritmo que nos permite seleccionar un nodo óptimo.

**Teorema 3.1 (Decidibilidad)** *Dado un MET, encontrar todos sus nodos óptimos es un problema decidible.*

**Demostración.** Abordamos la prueba mostrando que al menos existe un método finito para encontrar todos los nodos óptimos. En primer lugar, sabemos que el tamaño del MET debe ser finito puesto que la pregunta de la raíz sólo puede completarse si la ejecución ha terminado y por consiguiente el número de subcomputaciones—y por tanto de nodos—es finito [6]. Por ser el árbol finito, sabemos (de acuerdo al Algoritmo 1) que cualquier secuencia de preguntas realizada por el depurador (sin importar la estrategia) también es finita ya que como máximo preguntará por todos los nodos del árbol. Por tanto el número de secuencias posibles también es finito. Esto garantiza que siempre es posible calcular todas las secuencias posibles y quedarse con las mejores de acuerdo a la ecuación de la Definición 2.2. Los nodos óptimos serán los primeros de las secuencias seleccionadas. □

Aunque el método presentado en la prueba del Teorema 3.1 es efectivo, también es muy costoso porque implica calcular todas las secuencias posibles de preguntas. En el resto de la sección presentamos una aproximación más eficiente para encontrar todos los nodos óptimos.

## *3.1 Cómputo de secuencias válidas de preguntas ($sp_n$)*

Por claridad en la explicación, en adelante cuando hagamos referencia a una secuencia de preguntas de un nodo, esta secuencia supondrá que ese nodo es incorrecto y estará formada por un conjunto de nodos que tras ser preguntados permitirán determinar si ese nodo es un nodo-causa o no lo es.

De acuerdo a la Definición 2.2, una secuencia de preguntas es óptima (es producida por una estratégia óptima) si la suma de todas las preguntas que se harían suponiendo que el error puede estar en cualquier nodo es mínima. Por tanto, como se ha explicado en la prueba del Teorema 3.1, un método para calcular la secuencia óptima es calcular todas las secuencias posibles y elegir la mejor. Sin embargo, no todas las secuencias posibles son válidas, y por tanto muchas de ellas pueden descartarse.

**Definición 3.2** [Secuencias válidas de preguntas de un nodo, $SP_n$] Sea $T = (N, E, M)$ un MET cuya raíz es $n \in N$. Una secuencia de preguntas $sp_n = [n_1, \ldots, n_m]$ para $n$ es válida si:

(i) $\forall n_i, n_j \in sp_n, 1 \leq i < j \leq m, (n_i \rightarrow n_j) \notin E^*$

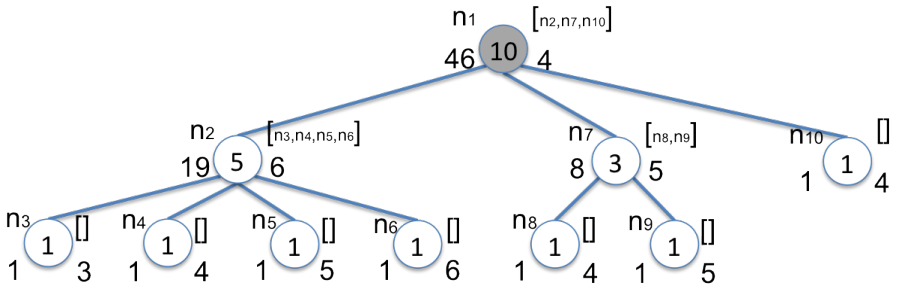
(ii) $N \backslash \{n_j \mid (n_i \rightarrow n_j) \in E^* \wedge n_i \in sp_n\} = \{n\}$

Denotamos con $SP_n$ el conjunto de las secuencias de preguntas válidas de $n$.

Intuitivamente las secuencias válidas de un nodo raíz $n$ son todas aquellas secuencias de nodos no repetidos que (1) un nodo de la secuencia no puede ser descendiente de un nodo anterior en la secuencia, y (2) tras podar todos los subárboles cuyas raíces son los nodos de la secuencia, el nodo $n$ se queda sin descendientes. Nótese que si consideramos un subábol, la misma definición puede ser utilizada para cualquier nodo del árbol original.

En el siguiente ejemplo mostramos que si se asocia a cada nodo de un MET una secuencia válida de preguntas, entonces es posible saber exactamente cuántas preguntas es necesario hacer para encontrar el nodo-causa del MET sea éste el nodo que sea.

**Ejemplo 3.3** Considérese el siguiente árbol formado por dos subárboles de profundidad 2 y un subárbol de profundidad 1.




Además del peso y el identificador del nodo, cada nodo está etiquetado arriba a la derecha con una secuencia de preguntas válida y abajo a la derecha con el número de preguntas necesario para encontrar el error en ese nodo. El lector puede ignorar por el momento los números etiquetados abajo a la izquierda de cada nodo.

Existen múltiples secuencias de preguntas posibles para el nodo $n_1$, (e.g. $[n_2, n_7, n_{10}]$, $[n_2, n_{10}, n_7]$, $[n_3, n_2, n_7, n_{10}]$, etc.). Siguiendo la secuencia de la figura ($[n_2, n_7, n_{10}]$) y teniendo en cuenta que se empieza a preguntar por el nodo raíz [5], podemos determinar el número de preguntas necesario para en-

[5] El hecho de que la secuencia de preguntas de un nodo suponga que la primera pregunta realizada es el propio nodo no es una limitación del método. Nótese que dado un MET cualquiera, siempre es posible añadir un nodo padre ficticio a la raíz y calcular la secuencia de nodos óptima de ese nodo ficticio. Esta secuencia coincide con la secuencia óptima de preguntas para encontrar el error en el MET original sin ese nodo ficticio.

contrar el error en cualquier nodo. En adelante, para un nodo $n$, llamaremos a este número $q_n$. Por ejemplo, para encontrar el error en el nodo $n_1$ se harían 4 preguntas ($[n_1, n_2, n_7, n_{10}]$). De manera similar, $q_{n_7} = 5$ ($[n_1, n_2, n_7, n_8, n_9]$). Nótese que al llegar al nodo $n_7$ y marcar el nodo como incorrecto se ha continuado con la secuencia de preguntas de ese nodo ($[n_8, n_9]$). De igual modo, $q_{n_4} = 4$ ($[n_1, n_2, n_3, n_4]$).

A partir del cálculo de las preguntas necesarias para encontrar el error en cada nodo ($q_n$), es posible obtener el número de preguntas totales del MET (en adelante $Q_n$). Este número es indicado para cada nodo del ejemplo anterior abajo a la izquierda. Obsérvese por ejemplo que el nodo raíz acumula un total de 46 preguntas ($Q_{n_1} = 46$), $Q_{n_2} = 19$ y los nodos hoja acumulan 1 pregunta (ese mismo nodo). En determinados casos, es fácil calcular $Q_n$. Por ejemplo, un momento de reflexión convencerá al lector de los siguientes cálculos de $Q_{n_1}$ usando las secuencias $[n_2, n_7, n_{10}]$, $[n_2, n_{10}, n_7]$, $[n_7, n_2, n_{10}]$:

$$\begin{aligned}
[n_2, n_7, n_{10}] \rightarrow Q_{n_1} &= (Q_{n_2} + w_{n_2}) + (Q_{n_7} + 2 * w_{n_7}) + (Q_{n_{10}} + 3 * w_{n_{10}}) + (4) \\
&= (19 + 5) + (8 + 6) + (1 + 3) + (4) = 46 \\
[n_2, n_{10}, n_7] \rightarrow Q_{n_1} &= (Q_{n_2} + w_{n_2}) + (Q_{n_{10}} + 2 * w_{n_{10}}) + (Q_{n_7} + 3 * w_{n_7}) + (4) \\
&= (19 + 5) + (1 + 2) + (8 + 9) + (4) = 48 \\
[n_7, n_2, n_{10}] \rightarrow Q_{n_1} &= (Q_{n_7} + w_{n_7}) + (Q_{n_2} + 2 * w_{n_2}) + (Q_{n_10} + 3 * w_{n_{10}}) + (4) \\
&= (8 + 3) + (19 + 10) + (1 + 3) + (4) = 48
\end{aligned}$$

Puesto que el objetivo es minimizar $Q_n$, utilizando el cálculo de $Q_{n_1}$ para cada secuencia, podemos concluir que la secuencia óptima del nodo $n_1$ es $[n_2, n_7, n_{10}]$.

En resumen, toda la información recogida y etiquetada en los nodos del MET puede consultarse en la Figura 5.

En el resto de la sección mostramos una técnica para calcular $Q_n$ para cualquier nodo de un MET. Empezamos definiendo dos casos base:

- Árbol de profundidad 1 ($n$ es una hoja): $Q_n = 1$
- Árbol de profundidad 2 ($n$ tiene $m$ hijos): $Q_n = (\sum_{i=2}^{m+1} i) + m + 1$

Estos casos base se corresponden con los siguientes árboles.

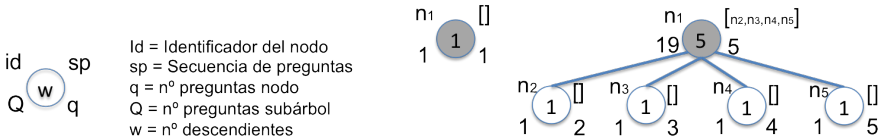



Fig. 5. Casos base

En el caso base 1 (árbol central de la Figura 5) sólo existe un nodo y por lo

tanto, al empezar preguntando por éste, en una única pregunta se encuentra el error. En el caso base 2 (árbol de la derecha de la Figura 5) se empieza a preguntar por el nodo raíz, por lo que todos los hijos llevan acarreada una pregunta (la del nodo raíz), seguidamente se preguntan los hijos de izquierda a derecha. Si el nodo-causa es el nodo raíz tendrán que preguntarse y descartarse todos los hijos, por lo que el nodo raíz tiene un $q_n$ de 5.

Para árboles de profundidad 3 o superior, el cálculo de $Q_n$ requiere de un proceso más elaborado. Afortunadamente, dicho proceso es composicional (i.e., puede calcularse $Q_n$ de un nodo a partir de los $Q_n$ de sus descendientes) y por tanto, puede calcularse al mismo tiempo que se genera el ET.

### *3.2 Cómputo de secuencias óptimas de preguntas*

En la sección anterior hemos visto que el coste $Q_n$ de cada nodo solamente depende del subárbol cuya raíz es ese mismo nodo. Este número nos indica la cantidad de preguntas que habría que hacer para encontrar el error si sólo se considera ese subárbol y empezando a preguntar por su nodo raíz. En esta sección el objetivo es encontrar la secuencia $sp_n$ que minimice el $Q_n$ del nodo. El Algoritmo 2 obtiene esa secuencia.

**Algorithm 2** Calcular $sp_n$ óptima

**Entrada:** Un MET $T = (N, E, M)$ y un nodo $n \in N$
**Salida:** $(sp_n, Q_n)$
**Precondiciones:** *n.profundidad* devuelve el nivel de profundidad del nodo $n$

**begin**
(1) **if** ($n.profundidad = 1$)
(2) $spOptima = []$
(3) $QOptima = 1$
(4) **else if** ($n.profundidad = 2$)
(5) $spOptima = [n_a, ..., n_z] \mid n_a, ..., n_z \in N \wedge (n \to n_a), ..., (n \to n_z) \in E$
(6) $QOptima = (\sum_{i=2}^{m} i) + m \mid m = 1 + Card(spOptima)$
**else**
(7) $spOptima = sp_n \in SP_n \mid \nexists sp'_n \in SP_n, calcularQ(T, n, sp_n) > calcularQ(T, n, sp'_n)$
(8) $QOptima = calcularQ(T, n, spOptima)$
**end if**
(9) **return** $(spOptima, QOptima)$
**end**

A medida que el depurador construye el ET, el Algoritmo 2 puede usarse para generar (al mismo tiempo) las secuencias óptimas de preguntas de cada nodo generado. Para ello el algoritmo debe ejecutarse para cada nodo completado del árbol (i.e., éste ya no puede tener más descendientes), es decir, los descendientes del siguiente nodo a procesar siempre tendrán calculada previamente su secuencia óptima. Este algoritmo contempla los dos casos base para los árboles de profundidad 1 (líneas 2 y 3) y 2 (líneas 5 y 6). Si el árbol tuviera profundidad 3 o más se selecciona la mejor secuencia de entre las secuencias

válidas del nodo, escogiendo aquella cuyo $Q_n$ sea el más bajo de todas las secuencias válidas (línea 7). Para calcular el $Q_n$ de cada una de estas secuencias se utiliza el Algoritmo 3.

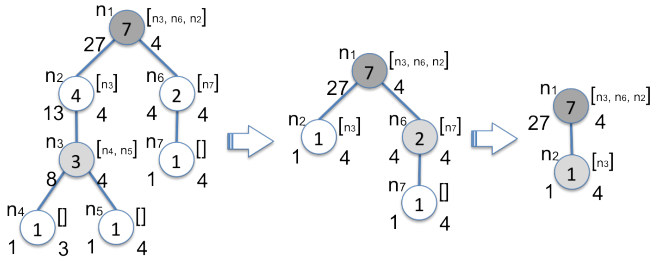
**Algorithm 3** Calcular $Q_n$ dada una secuencia de preguntas

**Entrada:** Un MET $T = (N, E, M)$, un nodo $n \in N$ y una secuencia de preguntas $sp_n \in SP_n$
**Salida:** $Q_n$
**Inicialización:** $preguntas = 0$
$pregAcarr = 0$

**begin**
(1) **while** $(\{n' | (n \to n') \in E^*\} \neq \{n\})$
(2) $\quad nodo = sp_n[pregAcarr]$
(3) $\quad pregAcarr = pregAcarr + 1$
(4) $\quad preguntas = preguntas + (Q_{nodo} + pregAcarr * w_{nodo})$
(5) $\quad T = ajustarNodosIntermedios(T, n, nodo)$
**end while**
(6) $preguntas = preguntas + (1 + pregAcarr)$
(7) **return** *preguntas*
**end**

El Algoritmo 3, tras cada evaluación del siguiente nodo de la secuencia (línea 4), incluye la llamada a una función que ajusta los costes $Q_n$ de los nodos intermedios entre $n$ y el último nodo preguntado $n'$ de la secuencia (línea 5). Esto se debe a que el coste $Q_n$ de los nodos que se encuentran entre $n$ y $n'$ incluían este subárbol. Sin embargo, ahora, cuando la estrategia pregunte por estos nodos se habrá podado el subárbol cuya raíz es $n'$. El Algoritmo 4 se encarga de ajustar el valor de $Q_n$ para estos nodos intermedios.

**Ejemplo 3.4** Considérese el siguiente árbol (izquierda) de profundidad 4, donde se quiere calcular el coste $Q_{n_1}$ asociada a la secuencia $[n_3, n_6, n_2]$.




El Algoritmo 3 evalúa en la primera iteración el nodo $n_3$. A continuación el Algoritmo 4 poda el subárbol del nodo $n_3$ y recalcula el valor de $Q_{n_2}$ y $w_{n_2}$. Como puede observarse, cuando el algoritmo evalúe este nodo en la tercera iteración (árbol de la derecha), éste debe tener un $Q_n$ acorde con la estructura que se encontrará la estrategia al preguntar por él al seguir la secuencia.

El Algoritmo 4 elimina el subárbol ya evaluado (línea 1 y 2) y además de calcular su nuevo valor $Q_n$ para los nodos intermedios (línea 5) también

**Algorithm 4** Ajustar nodos intermedios

**Entrada:** Un MET $T = (N, E, M)$ y dos nodos $n, n' \in N \mid n \neq n' \wedge (n \rightarrow n') \in E^*$
**Salida:** Un MET $T' = (N', E', M')$

**begin**
(1) $O = \{n'' \in N \mid (n' \rightarrow n'') \in E^*\}$
(2) $N = N \backslash O$
(3) $n' = n'' \mid (n'' \rightarrow n') \in E$
(4) **while** $(n' \neq n)$
(5) $\quad (\_, Q_{n'}) = calcularSpOptima(T, n')$
(6) $\quad w_{n'} = w_{n'} - |O|$
(7) $\quad n' = n'' \mid (n'' \rightarrow n') \in E$
**end while**
(8) **return** $T$
**end**

actualiza el peso de estos (línea 6) restándole al peso del nodo la cantidad de nodos eliminados del árbol.

## 4 Trabajo futuro

Los Algoritmos 2, 3 y 4 presentados en la Sección 3.2 asignan a cada nodo del ET su secuencia óptima, sin embargo esto se consigue obteniendo todas las secuencias válidas posibles y calculando cual de todas ellas es mejor (Algoritmo 2 línea 7). Obtener todas las secuencias válidas es una tarea costosa que podría optimizarse descartando dinámicamente aquellas secuencias que no pueden llegar a ser óptimas. Algunas de estas secuencias que se pueden descartar son:

- Aquellas que contienen alguna hoja siempre y cuando estos nodos no sean los últimos de la secuencia,
- Aquellas cuyo primer nodo sea un descendiente del primer nodo de la secuencia de alguno de sus hijos,
- La permutación de una secuencia (i.e., $[n_j, n_i, n_k, ...]$ con respecto a $[n_i, n_j, n_k, ...]$) donde el nodo $n_i$ sea más pesado que el nodo $n_j$.

## 5 Conclusiones

Tradicionalmente D&Q y sus variantes han tratado de dividir el ET en dos subárboles con el mismo peso, y hasta ahora se pensaba que esa manera de proceder garantizaba una secuencia de preguntas (en promedio) mínima para encontrar un error. Este trabajo demuestra que la estrategia D&Q no es óptima con respecto al número de preguntas que realiza al programador. Y además también demuestra que D&Q no es una estrategia correcta ni tampoco completa. Para ello proporciona contraejemplos razonando las causas del problema. Este problema radica fundamentalmente en el hecho de que

D&Q no tiene en cuenta la distribución de los nodos al podar el ET y esta información es fundamental para poder dividir el ET en dos subárboles que requieren el mismo esfuerzo para encontrar el error en ellos (y que no necesariamente implica que tengan el mismo peso).

Un resultado teórico del trabajo es la demostración de que encontrar una secuencia óptima de preguntas es un problema decidible. Además, se ha introducido la primera versión de una estrategia óptima. A pesar de ser óptima con respecto al número de preguntas, la estrategia propuesta tiene el problema de ser costosa, aunque bien es cierto que tiene la cualidad de ser composicional y por tanto la información que utiliza puede calcularse al mismo tiempo que se genera el ET. Para solucionar este problema, se han propuesto mecanismos que pueden ayudar a mejorar la eficiencia y que constituyen la base para una versión eficiente de la estrategia.

## References


[1] R. Caballero. A Declarative Debugger of Incorrect Answers for Constraint Functional-Logic Programs. In *Proc. of the 2005 ACM SIGPLAN Workshop on Curry and Functional Logic Programming (WCFLP'05)*, pages 8–13, New York, USA, 2005. ACM Press.

[2] R. Caballero, N. Martí-Oliet, A. Riesco, and A. Verdejo. A Declarative Debugger for Maude Functional Modules. *Electronic Notes in Theoretical Computer Science*, 238:63–81, June 2009.

[3] T. Davie and O. Chitil. Hat-delta: One Right Does Make a Wrong. In *Seventh Symposium on Trends in Functional Programming, TFP 06*, April 2006.

[4] V. Hirunkitti and C. J. Hogger. A Generalised Query Minimisation for Program Debugging. In *Proc. of International Workshop of Automated and Algorithmic Debugging (AADEBUG'93)*, pages 153–170. Springer LNCS 749, 1993.

[5] D. Insa and J. Silva. An Algorithmic Debugger for Java. *In Proc. of the 26th IEEE International Conference on Software Maintenance*, 0:1–6, 2010.

[6] D. Insa and J. Silva. Debugging with Incomplete and Dynamically Generated Execution Trees. In *Proc. of the 20th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR 2010)*, Austria, 2010.

[7] W. Lux. Declarative debugging meets the world. *Electronic Notes in Theoretical Computer Science*, 216:65 – 77, 2008. Proc. of the 16th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2007).

[8] H. Nilsson. *Declarative Debugging for Lazy Functional Languages.* PhD thesis, Linköping, Sweden, May 1998.

[9] B. Pope. *A Declarative Debugger for Haskell.* PhD thesis, The University of Melbourne, Australia, 2006.

[10] E. Shapiro. *Algorithmic Program Debugging.* MIT Press, 1982.

[11] J. Silva. A Comparative Study of Algorithmic Debugging Strategies. In *Proc. of the International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'06)*, pages 143–159. Springer LNCS 4407, 2007.