Evaluación de la Mejora de Conjuntos de Casos de Prueba mediante la Prueba de Mutación Evolutiva

Pedro Delgado-Pérez, Inmaculada Medina-Bulo

Departamento de Ingeniería Informática, Universidad de Cádiz, España
{pedro.delgado, inmaculada.medina}@uca.es

Resumen La Prueba de Mutación Evolutiva (PME) es una técnica surgida recientemente para reducir el número de mutantes a generar en la prueba de mutaciones y, por consiguiente, su alto coste computacional. Esto se logra a través de un algoritmo genético, el cual trata de localizar la mayor cantidad posible de los mutantes con potencial para guiar a la mejora del conjunto de casos de prueba (denominados mutantes fuertes) en ese subconjunto de mutantes generado. La técnica ha sido evaluada precisamente respecto a esa capacidad de encontrar mutantes fuertes, pero tal análisis omite el hecho de que parte de esos mutantes fuertes puede no aportar a la mejora de las pruebas ya que son mutantes equivalentes. Por esa razón, en este artículo se propone una nueva metodología para la evaluación de la PME. Esta realiza una estimación del refinamiento conseguido del conjunto de pruebas a través de los mutantes seleccionados por el algoritmo genético. Esta metodología se emplea sobre cuatro programas en C++ que aplican orientación a objetos, mostrando que la PME es capaz de aumentar el conjunto de pruebas generando un porcentaje menor de mutantes que la selección aleatoria.

Keywords: Prueba de software, prueba de mutaciones, algoritmos genéticos

1. Introducción

La *prueba de mutaciones* [8,11] es una técnica de prueba de software basada en fallos, ya que en ella se procede insertando fallos de forma intencionada en el programa que se prueba. El conjunto de casos de prueba que se desarrolla debe estar orientado a detectar la existencia de fallos en el programa, así que esta práctica viene a evaluar la capacidad de dicho conjunto para revelar errores. Los fallos se insertan mediante lo que se conoce como *operadores de mutación*, los cuales producen diferentes tipos de mutaciones en el código del programa, generando versiones ligeramente distintas a la original que se conocen como *mutantes*.

En esta técnica se generan los mutantes para seguidamente ejecutarlos con respecto al mismo conjunto de casos de prueba que se aplica al programa original. La funcionalidad del mutante puede ser distinta debido al fallo inyectado y es

lo que el conjunto de casos de prueba debe poder detectar. En ese caso, se dice que el mutante está *muerto*, y en caso contrario, que el mutante sigue *vivo*. En ocasiones, es posible que la funcionalidad del programa no difiera a pesar de la mutación, lo que se conoce como mutante *equivalente*. Por todo lo dicho, podremos decir que disponemos de un buen conjunto de casos de prueba para un determinado programa cuando este es capaz de detectar todas las mutaciones que afecten al comportamiento del programa. Ese conjunto de casos de prueba es llamado *conjunto adecuado*.

Además de evaluar la habilidad de nuestro conjunto de pruebas en la detección de potenciales errores, la prueba de mutaciones nos puede servir para mejorar la calidad de ese conjunto de pruebas. Para tal fin es para lo que se emplea la *Prueba de Mutación Evolutiva* (PME) [6], una técnica basada en un algoritmo evolutivo enfocado en la búsqueda de los mutantes que pueden ayudarnos en esa mejora de las pruebas, conocidos como *mutantes fuertes*. Técnicas como la PME surgen por la necesidad de reducir el coste computacional de la prueba de mutaciones, de forma que se mantenga una efectividad similar pero generando un número más reducido de mutantes.

La PME se ha puesto en práctica mediante un algoritmo genético [5], dando buenos resultados tanto para composiciones WS-BPEL [6] como para sistemas orientados a objetos programados en C++ [4]. Esta afirmación se basa en las evaluaciones realizadas con esta técnica para hallar mutantes fuertes. Sin embargo, debemos tener en cuenta que el objetivo final es el de refinar nuestro conjunto de casos de prueba, y esto se logra a partir de la revisión de los mutantes fuertes. Ese refinamiento no está ligado de forma directa con dicha revisión de mutantes fuertes, por ejemplo, por la posibilidad de generar mutantes fuertes que resulten siendo equivalentes y que, por tanto, no nos ayuden en nuestra labor (no existe ningún caso de prueba que pueda detectar la presencia de esas mutaciones).

Por lo anteriormente dicho, un análisis más en línea con el objetivo final de la PME es el de cuantificar la mejora que se produce en las pruebas. En este artículo se presenta una nueva metodología para estimar cuánto nos acercamos al conjunto adecuado si empleamos el subconjunto de mutantes generados con la PME para la mejora de las pruebas. Esto nos permitirá tener una mejor evidencia de que estamos logrando lo que esperábamos al utilizar el algoritmo genético.

Una vez definida, la metodología ha sido aplicada con los mismos operadores de mutación de clase [3] que se emplearon previamente sobre programas en C++ [4]. La PME ha sido entonces comparada con una técnica de selección aleatoria de mutantes. Los resultados revelan que, al igual que ocurrió en los experimentos previos para detectar mutantes fuertes, la PME supera a la selección aleatoria, es decir, es capaz de alcanzar diferentes incrementos en el tamaño del conjunto de casos de prueba generando un porcentaje menor del total de mutantes. Esto ocurre en todos los programas y, en especial, en aquellos en los que se genera un mayor número de mutantes y cuanto más buscamos acercarnos al conjunto adecuado.

El artículo se estructura de la siguiente manera. En la siguiente sección, se expondrán brevemente los principios fundamentales de la PME y se ahondará en

la motivación de establecer una nueva manera de calcular el rendimiento de la técnica. En la Sección 3 se explicará la metodología propuesta, usando un ejemplo ilustrativo del proceso a seguir. En la Sección 4 se detallará el procedimiento experimental que se aplica a los programas seleccionados, y se mostrarán y analizarán los resultados empíricos obtenidos. Finalmente, la Sección 5 se dedica a las conclusiones y a las posibilidades de trabajo futuro.

2. Prueba de Mutación Evolutiva

A consecuencia del alto coste que potencialmente puede suponer la aplicación de la prueba de mutaciones, han surgido diversas técnicas con el objetivo de reducir el número de mutantes a crear pero sin que esto afecte demasiado a la efectividad. Entre ellas podemos citar la mutación por muestreo [1], la mutación por agrupamiento [7] o la mutación selectiva [9]. Entre estas técnicas de reducción del coste, la Prueba de Mutación Evolutiva es la única enfocada por completo a la mejora del conjunto de casos de prueba desarrollado y no a la evaluación del mismo. Se trata de una técnica de búsqueda de aquellas mutaciones que tienen la capacidad para ayudar a diseñar nuevos casos de prueba. Como se ha dicho anteriormente, a estos mutantes se les conoce como fuertes. Los mutantes seleccionados al finalizar la ejecución de la PME podrán ser, en una fase posterior, utilizados para la creación de nuevas pruebas. Existen otros trabajos que, como la PME, utilizan un algoritmo genético para la generación de un subconjunto de mutantes, siguiendo diferentes enfoques [10].

Para localizar a esos mutantes fuertes, la PME se sirve de un *algoritmo genético* en el que se van seleccionando subconjuntos reducidos de mutantes en diversas generaciones, de tal manera que los mutantes seleccionados en una generación dependen de la información de los mutantes generados previamente. El proceso es el siguiente:

- A cada mutante se le asigna un valor en base a una función de aptitud.
- Se aplica un método para seleccionar mutantes, favoreciendo a los de mejor aptitud.
- Se generan nuevos mutantes mediante operadores reproductivos aplicados a esos mutantes seleccionados de la generación anterior.

La *función de aptitud* valora a cada mutante por dos factores:

- El número de casos de prueba que lo matan (cuantos menos mejor).
- El número de mutantes a los que esos casos de prueba matan al mismo tiempo (igualmente, cuantos menos mejor).

Es decir, que la función de aptitud le adjunta el mayor valor a aquellos mutantes que parecen necesitar de casos de prueba más específicos para ser detectados, ya que esos son los que tienen mayor potencial para inducir casos de prueba faltantes (mutantes fuertes). De esta manera, se consideran mutantes fuertes aquellos que son matados por un único caso de prueba que no mata

a ningún mutante más (*difíciles de matar*) y a aquellos que permanecen sin ser detectados (*potencialmente equivalentes*). Se espera por tanto que cada vez se produzcan mutantes más específicos en subsiguientes generaciones, llegando en última instancia a producir en mayor proporción mutantes potencialmente equivalentes.

El funcionamiento de este algoritmo genético se describió en mayor profundidad en un trabajo previo [4]. En este trabajo además se realizaron experimentos para evaluar la reducción en cuanto al número de mutantes a generar hasta lograr localizar diferentes porcentajes del total de mutantes fuertes en diversas aplicaciones, al igual que también se había realizado en el trabajo en el que se definió la PME por primera vez [6]. En ese trabajo previo [4], en el cual se analizó la técnica respecto a operadores de mutación de clase para C++ en tres programas, la diferencia entre aplicar PME y realizar una selección aleatoria de mutantes era en torno al 6.5 % y 4 % para localizar un 75 % y un 90 % del total de mutantes fuertes.

Esos resultados, sin embargo, no ofrecían información sobre el uso que se les da después a esos mutantes para incrementar el conjunto de casos de prueba con nuevos escenarios. En principio, esto no sería ningún problema si cada uno de los mutantes potencialmente equivalentes encontrados proveyese un nuevo caso de prueba. Pero esto no ocurre así por dos factores:

- En primer lugar, es posible que un único caso de prueba sea suficiente para matar a varios de esos mutantes que permanecían vivos.
- En segundo lugar, y más importante, un porcentaje de esos mutantes potencialmente equivalentes pueden, como su nombre indica, resultar siendo equivalentes. El número de mutantes equivalentes depende de las características del programa y de los operadores de mutación aplicados y, en general, no es posible automatizar su detección.

Es por ello que el análisis del número de mutantes fuertes detectados y el número de nuevos casos de prueba puede diferir. Este hecho da pie a ir un paso más allá en la evaluación del rendimiento de la PME, proponiendo una nueva metodología que se enfoque a la mejora del conjunto de casos de prueba en lugar de a los mutantes seleccionados.

3. Metodología

El objetivo de esta investigación es la de conocer, a través de la experimentación, si los mutantes que la PME selecciona son capaces de inducir al refinamiento del conjunto de casos de prueba que se está evaluando. Como se ha comentado anteriormente, los experimentos en este artículo van un paso más allá que los realizados previamente, donde se analizaba la capacidad de esta técnica para encontrar mutantes fuertes pero no se conocía hasta qué punto esos mutantes podían ayudar en la mejora del conjunto de casos de prueba.

En esta sección se explica la metodología que seguimos para tal fin. Esta metodología se basa en una simulación de la generación o modificación de casos

de prueba a través de la información que nos proporcionan los mutantes seleccionados por la PME. Decimos simulación porque, como veremos más adelante, los casos de prueba no se añaden como consecuencia directa de revisar el código de los mutantes seleccionados, sino que nos apoyamos en información previamente generada para poder automatizar el proceso. Además, dado que los nuevos casos de prueba se van a generar manualmente, la no intervención por nuestra parte durante el proceso de mejora evita cualquier posible sesgo por tratar de favorecer los resultados que reporta la técnica.

Para explicar la metodología, nos apoyaremos en un ejemplo que nos ayude visualmente a comprender cómo se lleva a cabo el proceso. Para ello nos basaremos en el concepto de *matriz de ejecución*. Una matriz de ejecución contiene la información de la aplicación de cada uno de los casos de prueba (en las columnas) a cada uno de los mutantes generados (en las filas). Por ejemplo, la PME maneja internamente matrices de ejecución para poder calcular la función de aptitud en cada generación. En cada una de las celdas de esta matriz podemos encontrar los siguientes valores:

- **Valor 0**: El caso de prueba no es capaz de matar al mutante.
- **Valor 1**: El caso de prueba sí es capaz de matar al mutante.
- **Valor 2**: La mutación insertada no es válida.

De esta manera, podemos saber el estado de un mutante a partir de la información en la fila que lo representa en la matriz:

- **Vivo**: Todos los valores de la fila son 0 (ningún caso de prueba puede detectar al mutante).
- **Muerto**: Existe al menos una celda con valor 1 (algunos de los casos de prueba pueden detectar al mutante).
- **Inválido**: Como el mutante no compila, no puede ser ejecutado. Así que todos los valores de la fila serán 2.

Pongamos pues el ejemplo de matriz de ejecución de la Figura 1 que resulta de la ejecución del conjunto de casos de prueba desarrollado (5 casos de prueba) y a los mutantes que pueden ser generados en nuestro programa (8 mutantes). Según la información que proporciona esta matriz, tenemos:

- **Mutantes vivos**: $mutante_2$, $mutante_4$, $mutante_6$ y $mutante_7$.
- **Mutantes muertos**: $mutante_1$, $mutante_3$ y $mutante_8$.
- **Mutantes inválidos**: $mutante_5$.

Al disponer de *toda la información* de todos los mutantes, podemos revisar aquellos mutantes vivos y obtener un conjunto de casos de prueba adecuado para este conjunto de mutantes. El resultado de aplicar ese conjunto adecuado a los mutantes es la matriz de ejecución de la Figura 2, a la que llamaremos *matriz adecuada*. Podemos observar en esta matriz que:

- Los mutantes $mutante_2$, $mutante_4$ y $mutante_7$ han sido matados, mientras que $mutante_6$ resultó ser equivalente.

- Se han añadido los nuevos casos de prueba $test_6$ y $test_7$ para matar al $mutante_2$ y $mutante_4$ respectivamente. En cuanto a $mutante_7$, en lugar de añadir un nuevo caso de prueba, se ha modificado el $test_4$ ya que simplemente faltaba realizar la comprobación oportuna para poder detectar a este mutante.

	test_1	test_2	test_3	test_4	test_5
mutante_1	1	0	1	0	0
mutante_2	0	0	0	0	0
mutante_3	0	1	1	0	0
mutante_4	0	0	0	0	0
mutante_5	2	2	2	2	2
mutante_6	0	0	0	0	0
mutante_7	0	0	0	0	0
mutante_8	0	0	0	0	1

Figura 1. Matriz de ejecución asociada al conjunto de casos de prueba original y al conjunto completo de mutantes.

	test_1	test_2	test_3	test_4	test_5	test_6	test_7
mutante_1	1	0	1	0	0	0	0
mutante_2	0	0	0	0	0	1	0
mutante_3	0	1	1	0	0	0	0
mutante_4	0	0	0	0	0	0	1
mutante_5	2	2	2	2	2	2	2
mutante_6	0	0	0	0	0	0	0
mutante_7	0	0	0	1	0	0	0
mutante_8	0	0	0	0	1	0	0

Figura 2. Matriz de ejecución asociada al conjunto de casos de prueba adecuado y al conjunto completo de mutantes (matriz adecuada).

Una vez en posesión de esta información, pasaremos a ejecutar la PME con respecto a nuestro conjunto de casos de prueba original para que, en lugar de generar todos los mutantes, tan solo se seleccione un subconjunto (que es el objetivo de esta técnica). Imaginemos que en las dos primeras generaciones se seleccionan los mutantes 1, 3, 4 y 7. Gracias a la información obtenida previamente al mejorar el conjunto de casos de prueba (Figura 2), podemos estimar qué mejora nos proporcionaría ese conjunto de mutantes seleccionados extrayendo la información de estos mutantes de la matriz adecuada. La información de esta matriz, a la que llamaremos *matriz seleccionada*, se muestra en la Figura 3.

	test_1	test_2	test_3	test_4	test_5	test_6	test_7
mutante_1	1	0	1	0	0	0	0
mutante_3	0	1	1	0	0	0	0
mutante_4	0	0	0	0	0	0	1
mutante_7	0	0	0	1	0	0	0

Figura 3. Matriz de ejecución asociada al conjunto de casos de prueba adecuado y al conjunto seleccionado de mutantes (matriz seleccionada).

Para realizar esta estimación, vamos a usar el concepto de *conjunto de casos de prueba mínimo*. Un conjunto de casos de prueba es mínimo para un conjunto de mutantes cuando, a partir de la matriz de ejecución asociada, se puede obtener una nueva matriz en la que se siga matando al mismo número de mutantes pero con el mínimo número de casos de prueba. La minimización de los conjuntos de casos de prueba para experimentos como este es apropiada ya que, de esta manera, se evita que casos de prueba redundantes pudiesen distorsionar los resultados. Definido este término, podemos sacar los siguientes conjuntos mínimos para la:

- **Matriz adecuada:** $test_3$, $test_4$, $test_5$, $test_6$ y $test_7$. Se pueden descartar el $test_1$ y $test_2$ porque los mutantes a los que matan también son matados por el $test_3$.
- **Matriz seleccionada:** $test_3$, $test_4$ y $test_7$ son suficientes para matar a los mutantes 1, 3, 4 y 7.

Al minimizar ambos conjuntos, podemos decir que con los mutantes seleccionados por la PME podríamos obtener 3 de hasta 5 casos de prueba que, como mínimo, serían necesarios para matar a todos los mutantes que podrían ser generados sin aplicar ninguna técnica de reducción de mutantes. Esta diferencia entre tamaños de conjuntos mínimos es la que empleamos en el procedimiento experimental para establecer cuando el algoritmo genético debe parar. Se determina un porcentaje P de forma que el algoritmo seguirá produciendo generaciones de mutantes hasta que, con todos los mutantes generados hasta el momento, se cumpla la siguiente condición de parada:

$$|Matriz\ seleccionada| \geq |Matriz\ adecuada| \times P \tag{1}$$

Para concluir con el ejemplo, supongamos que establecemos $P = 60\,\%$. Es decir, queremos que se generen tantos mutantes como sean necesarios para obtener un conjunto de pruebas con tamaño igual o superior al 60 % del tamaño del conjunto adecuado y mínimo. Siendo así, el algoritmo pararía la ejecución al alcanzar la condición de parada, ya que $|Matriz\ seleccionada| = 3$, $|Matriz\ adecuada| = 5$ y $3 \geq 5 \times 60\,\%$. Por el contrario, si P fuera mayor de 60 %, el algoritmo genético tendría que producir al menos una nueva generación de mutantes.

4. Evaluación de los Resultados de los Experimentos

En esta sección se aplica la metodología presentada en la sección anterior, exponiendo en qué consisten los experimentos a realizar, los resultados y las implicaciones de los mismos.

4.1. Configuración del experimento

El procedimiento experimental busca comparar el rendimiento en la mejora del conjunto de casos de prueba de la PME y un algoritmo de selección de mutantes aleatorio (la probabilidad con la que se selecciona cada mutante es la misma). El objetivo es medir la cantidad de mutantes que cada una de las técnicas debe generar para lograr conjuntos de casos de prueba de diferentes tamaños (porcentaje respecto al conjunto adecuado). Recordar que el conjunto adecuado es conocido gracias a una ejecución previa de todos los mutantes y a un proceso de refinamiento del conjunto de casos de prueba hasta matar a todos los mutantes no equivalentes.

Para este experimento se van a emplear cuatro casos de estudio con diferente números de mutantes, y se usarán los mismos parámetros de configuración para el algoritmo genético que los usados en experimentos anteriores para encontrar diferentes porcentajes de mutantes fuertes [4]. En el Cuadro 1 y 2 incluimos los datos relevantes tanto de los casos de estudio como de los parámetros de configuración respectivamente. Se muestra también una tabla con la información sobre los tamaños de los conjuntos de prueba empleados (Cuadro 3): conjunto original (distribuido con el programa), casos de prueba nuevos y modificados (para mejorar el conjunto original), conjunto adecuado (al añadir los nuevos casos de prueba al conjunto original) y conjunto mínimo (al minimizar el conjunto adecuado).

Cuadro 1. Mutantes generados en los casos de estudio analizados

	Dolphin	**KMyMoney**	**Tinyxml2**	**QtDom**
Mutantes totales	219	322	614	1,146
Válidos	208	251	433	681
Fuertes	103	151	159	348
% Mutantes fuertes	49.5%	60.2%	36.7%	51.1%

Para la ejecución de la PME se empleará una combinación del algoritmo genético implementado en *GAmera* [5] y un sistema de mutaciones para C++ que incluye los operadores de mutación a nivel de clase [2]. Las modificaciones que se realizaron para adaptar la ejecución de ambas herramientas se explicó en un trabajo previo [4]. Se ejecutarán los experimentos para dos valores de P (condición de parada): alcanzar el 75 % y el 90 % del conjunto de pruebas adecuado y mínimo. El algoritmo se ejecuta 30 veces para evitar el sesgo que puede provocar la componente aleatoria.

Cuadro 2. Parámetros utilizados en la ejecución del algoritmo genético

Parámetro	**Valor**
Tamaño población	5%
Individuos generados aleatoriamente	10%
Individuos generados por mutación o cruce	90%
- Probalididad de mutación	30%
- Probabilidad de cruce	70%

Cuadro 3. Tamaño de los conjuntos de casos de prueba analizados

	Dolphin	**KMyMoney**	**Tinyxml2**	**QtDom**
$\|Original\|$	61	241	57	46
$\|Nuevos\ tests\|$	9	7	5	10
$\|Tests\ modificados\|$	5	10	3	4
$\|Adecuado\|$	70	248	62	56
$\|Mínimo\|$	22	34	15	25

Los mismos experimentos descritos anteriormente (casos de estudio, condiciones de parada y número de ejecuciones) se realizan sobre el algoritmo aleatorio a fin de comparar los resultados que proporciona usar una u otra técnica. Este algoritmo consiste en una ordenación aleatoria de los mutantes antes de su ejecución (ya que los mutantes que se pueden generar son conocidos de antemano), para después ir seleccionando uno a uno los mutantes hasta que se cumpla la condición de parada que se persigue.

4.2. Resultados

El Cuadro 4 muestra diversas estadísticas de la ejecución del experimento para los distintos programas. En concreto, se muestra la media, mediana, porcentaje mínimo y máximo y desviación estándar de los mutantes generados por la PME y la técnica aleatoria hasta llegar al 75 % y 90 % del tamaño del conjunto de casos de prueba adecuado y mínimo. Notar por tanto que, a menor porcentaje, mejor es el comportamiento de la técnica. Se puede pues observar que, en todos los casos, la PME necesita producir menos mutantes que la selección aleatoria en promedio. Al final del Cuadro 4 se muestra la media total en estos programas del porcentaje generado por estas técnicas antes de alcanzar ambas condiciones de parada.

De estos resultados se pueden destacar los siguientes hechos:

- Los mejores resultados a favor de la PME se dan en *Tinyxml2* y *QtDom*, donde la diferencia máxima en promedio respecto al algoritmo aleatorio se da en *QtDom* para $P = 90\,\%$, y es en torno al 26 %.
- El peor resultado para PME se da en el par *KMyMoney* y $P = 75\,\%$, con un estrecho margen del 1 % de diferencia. Sin embargo, en este mismo programa la diferencia pasa a ser de más del 7 % con la condición de parada del 90 %.

Cuadro 4. Porcentaje de mutantes generados por la PME y la técnica aleatoria hasta conseguir un 75 % y 90 % del conjunto adecuado y mínimo.

P	***75%***		***90%***	
Técnica	*PME*	*Aleatoria*	*PME*	*Aleatoria*
Dolphin				
Media	49.75	54.10	66.33	71.08
Mediana	48.63	52.05	65.29	69.63
Mínimo	36.52	30.13	52.51	40.63
Máximo	74.42	84.01	84.93	88.58
D.E.	8.51	9.95	8.61	10.45
KMyMoney				
Media	46.89	47.84	68.30	75.48
Mediana	47.20	47.82	67.86	75.93
Mínimo	31.98	32.60	54.03	53.10
Máximo	63.97	65.83	83.22	86.33
D.E.	6.87	9.04	7.14	8.03
Tinyxml2				
Media	19.26	25.75	31.93	46.79
Mediana	18.48	24.34	32.25	43.24
Mínimo	10.58	11.88	20.52	24.75
Máximo	27.36	53.09	46.09	86.80
D.E.	4.38	8.98	7.13	15.21
QtDom				
Media	13.33	23.74	21.41	49.04
Mediana	13.00	21.90	21.16	46.68
Mínimo	7.85	11.16	11.95	26.96
Máximo	23.29	49.04	35.86	81.15
D.E.	3.35	7.99	5.00	12.85
Media total	32.31	37.86	46.99	60.60

Cuadro 5. Porcentaje de ahorro promedio al aplicar la PME con respecto a la generación de todos los mutantes (Completo) y respecto a la selección aleatoria (Aleatoria) para conseguir el 75 % y 90 % del conjunto adecuado y mínimo.

P	***75%***		***90%***	
Técnica	*Completo*	*Aleatoria*	*Completo*	*Aleatoria*
Dolphin	50.25	8.04	33.67	6.68
KMyMoney	53.11	1.99	31.70	9.51
Tinyxml2	80.74	25.20	68.07	31.76
QtDom	86.67	43.85	78.59	56.34
Media	67.69	19.77	53.00	26.07

- De forma generalizada, la distancia entre ambas técnicas se ensancha cuando la mejora buscada del conjunto de pruebas se hace más exigente (es decir, los resultados son mejores para la PME cuando $P = 90\,\%$).
- En todas las estadísticas, excepto en contadas ocasiones, PME obtiene mejores resultados, pudiendo destacar la desviación estándar, que refleja que el algoritmo genético se comporta siempre de una manera más estable.

Por último, en el Cuadro 5 se muestra el ahorro promedio obtenido al aplicar la PME con respecto a la generación de todos los mutantes y también con respecto a los resultados de la selección aleatoria. Los datos en esta tabla son relativos al 100 % de los mutantes para el primer caso, y al porcentaje de mutantes generado por la selección aleatoria para el segundo caso (ver Cuadro 4).

4.3. Discusión sobre los resultados

De los resultados obtenidos podemos extraer diversas conclusiones. En primer lugar, que la PME supera en rendimiento a la selección aleatoria a la hora de seleccionar mutantes para mejorar un conjunto de casos de prueba, ocurriendo esto en todos los casos de estudio. En segundo lugar, que los beneficios de usar la técnica se incrementan cuando se trata de conseguir una aproximación mayor al conjunto de casos de prueba adecuado, puesto que las diferencias entre ambas técnicas aumentan al pasar de $P = 75\,\%$ a $P = 90\,\%$. Como prueba, la media total muestra que la diferencia entre el algoritmo genético y el aleatorio es del 5.55 % para la primera condición de parada, pero sube hasta el 13.61 % para la condición de parada más exigente. En último lugar, parece que el algoritmo genético escala con el tamaño del programa, puesto que se dan mejores resultados para los programas con mayor número de mutantes en total.

No obstante, estos experimentos están sujetos a diversos factores que pueden tener influencia en los resultados:

- Estos experimentos tienen una gran dependencia del conjunto de casos de prueba original. Por una parte, hay algunos conjuntos más completos que otros en relación al conjunto adecuado. Esto se puede comprobar al observar la disparidad entre los resultados entre programas. Por otro lado, hay conjuntos con casos de prueba que presentan una mayor capacidad que otros para detectar a un mayor número de mutantes (dependiendo de lo específicos o genéricos que se hayan concebido los casos de prueba). De esta manera, seleccionar cualquiera de los mutantes a los que un caso de prueba mata puede inducir a la inclusión de ese caso de prueba en el conjunto. Esto puede perjudicar a la PME en estos experimentos, ya que esta técnica se centra en buscar mutantes matados por muy pocos casos de prueba (lo cual también incluye mutantes equivalentes).
- Los mutantes inválidos producen una diferencia con respecto a experimentos previos en los que se buscaba localizar mutantes fuertes. En este último caso, la condición de parada se veía afectada únicamente por mutantes que fuesen fuertes. Sin embargo, en los experimentos en este artículo, incluso

un mutante que sea de baja calidad (matado por muchos casos de prueba) puede provocar el aumento del tamaño del conjunto de casos de prueba. No obstante, aunque esto puede afectar el resultado del experimento, también demuestra que la PME es capaz de evitar la generación de mutantes inválidos en mayor grado que el algoritmo aleatorio.

En cuanto a los resultados del Cuadro 5, estos se muestran a fin de poder evaluar el ahorro al utilizar esta técnica en cuanto a coste de ejecución. De esta forma, podemos decir que, en promedio, usar la PME con respecto a no usarla nos ahorra un 67.69 % y un 53 % en cuanto a generación y ejecución de mutantes para $P = 75\,\%$ y $P = 90\,\%$ respectivamente. Una medida más justa del ahorro es la comparación con respecto a la selección aleatoria. Tomando como referencia el porcentaje de mutantes a generar con la selección aleatoria (Cuadro 4), vemos que la PME nos proporciona un ahorro del 19.77 % para alcanzar el 75 % del conjunto de casos de prueba, el cual se incrementa hasta el 26.07 % cuando se trata del 90 %.

5. Conclusiones

En este artículo se ha desarrollado una nueva metodología para medir el rendimiento de la Prueba de Mutación Evolutiva. En esta metodología se estima la mejora que nos proporciona el conjunto de mutantes seleccionado por la técnica en términos de incremento del número de casos de prueba. Esta metodología será de gran utilidad en futuros experimentos respecto a la PME: la medición de la utilidad de la técnica en base al número de mutantes fuertes encontrados (como se ha realizado en experimentos previos) no ofrece información alguna sobre la manera en la que nos ayuda a completar el conjunto de casos de prueba. De esta manera, nuestra propuesta nos permitirá, por ejemplo, evaluar de manera apropiada la integración de mecanismos para la detección de mutantes equivalentes, tal y como se tiene previsto realizar. Es decir, si la detección es efectiva, tendrá su reflejo en la mejora alcanzada del conjunto de pruebas.

Se han llevado a cabo experimentos siguiendo esta metodología, comparando los resultados de la PME y la selección aleatoria de mutantes. Al comparar los porcentajes de mutantes a generar para lograr determinados niveles de mejora en los casos de prueba, se observa que la PME necesita producir un número menor de mutantes, especialmente en aquellos programas en los que se crean más mutantes. No obstante, estos experimentos también han revelado que esta metodología aún podría ser perfeccionada. Esto es así porque no existe una clara relación entre que los mutantes seleccionados incrementen el conjunto de pruebas en la simulación y que, en la práctica, esos mutantes pudiesen realmente inducir el diseño de esos casos de prueba. Por tanto, esta mejora supondría buscar métodos para analizar no solo el tamaño del conjunto de casos de prueba, sino también la calidad (o especificidad) de los casos de prueba que se producirían.

Agradecimientos: Este trabajo está parcialmente financiado por la beca de investigación PU-EPIF-FPI-PPI-BC 2012-037 de la Universidad de Cádiz, por los

fondos FEDER y por los proyectos nacionales del Ministerio de Economía y Competitividad DArDOS (TIN2015-65845-C3-3-R) y la Red de Excelencia SEBASENET (TIN2015-71841-REDT).

Referencias

1. Budd, T.A.: Mutation Analysis of Program Test Data. Ph.D. thesis, Yale University (1980)
2. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J.: Herramienta para la prueba de mutaciones en el lenguaje C++. In: XX Jornadas de Ingeniería del Software y Base de Datos, JISBD 2015. Santander, Spain (2015)
3. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J., García-Domínguez, A.: Operadores de mutación a nivel de clase para el lenguaje C++. In: XII Jornadas sobre Programación y Lenguajes, PROLE 2013. Madrid, Spain (2013)
4. Delgado-Pérez, P., Medina-Bulo, I., Segura, S., García-Domínguez, A., Domínguez-Jiménez, J.J.: Prueba de mutación evolutiva aplicada a sistemas orientados a objetos. In: XXI Jornadas de Ingeniería del Software y Base de Datos, JISBD 2016. Salamanca, Spain (2016)
5. Domínguez-Jiménez, J.J., Estero-Botaro, A., García-Domínguez, A., Medina-Bulo, I.: GAmera: an automatic mutant generation system for WS-BPEL compositions. In: Eshuis, R., Grefen, P., Papadopoulos, G.A. (eds.) Proceedings of the 7th IEEE European Conference on Web Services. pp. 97–106. IEEE Computer Society Press, Eindhoven, The Netherlands (Nov 2009)
6. Domínguez-Jiménez, J.J., Estero-Botaro, A., García-Domínguez, A., Medina-Bulo, I.: Evolutionary mutation testing. Information and Software Technology 53(10), 1108–1123 (Oct 2011), http://dx.doi.org/10.1016/j.infsof.2011.03.008
7. Hussain, S.: Mutation Clustering. Master's thesis, King's College London (2008)
8. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 37(5), 649 –678 (Oct 2011), http://dx.doi.org/10.1109/TSE.2010.62
9. Offutt, A.J., Rothermel, G., Zapf, C.: An experimental evaluation of selective mutation. In: Proceedings of 15th International Conference on Software Engineering, 1993. pp. 100–107 (May 1993), http://dx.doi.org/10.1109/ICSE.1993.346062
10. Silva, R.A., do Rocio Senger de Souza, S., de Souza, P.S.L.: A systematic review on search based mutation testing. Information and Software Technology (2016), http://dx.doi.org/10.1016/j.infsof.2016.01.017
11. Woodward, M.R.: Mutation testing - its origin and evolution. Information and Software Technology 35(3), 163–169 (Mar 1993)