

On the Correlation between Static Measures and Code Coverage using Evolutionary Test Case Generation

Javier Ferrer, Francisco Chicano and Enrique Alba

ferrer@lcc.uma.es

chicano@lcc.uma.es

eat@lcc.uma.es



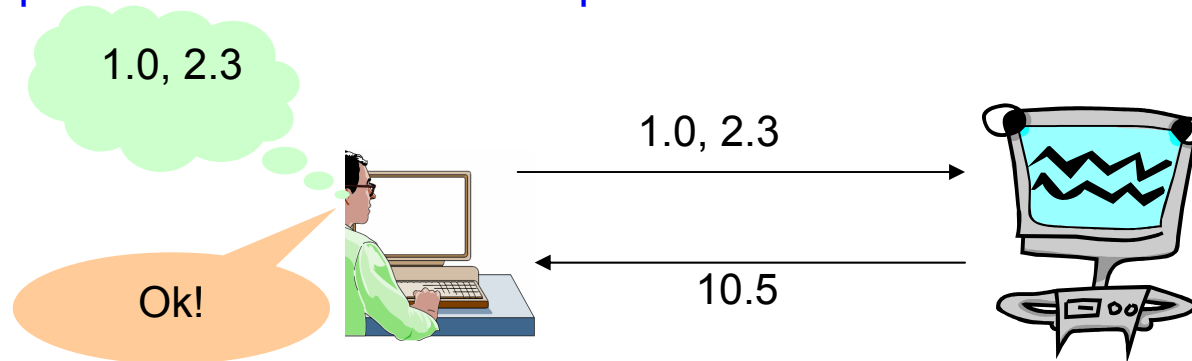
UNIVERSIDAD
DE MÁLAGA

Tabla de Contenidos

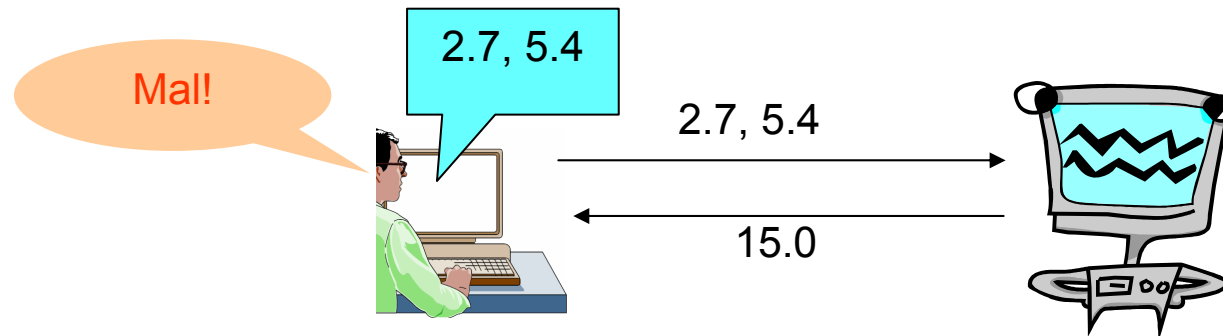
- 1 Introducción
- 2 Generador de Casos de Prueba
- 3 Algoritmo de Optimización
- 4 Experimentos
- 5 Conclusiones

Introducción

- Tras la codificación, el software requiere una **fase de prueba**
- El objetivo es comprobar que el software cumple la **especificación**
- Las empresas software dedican aprox. el **50%** de recursos a dicha fase

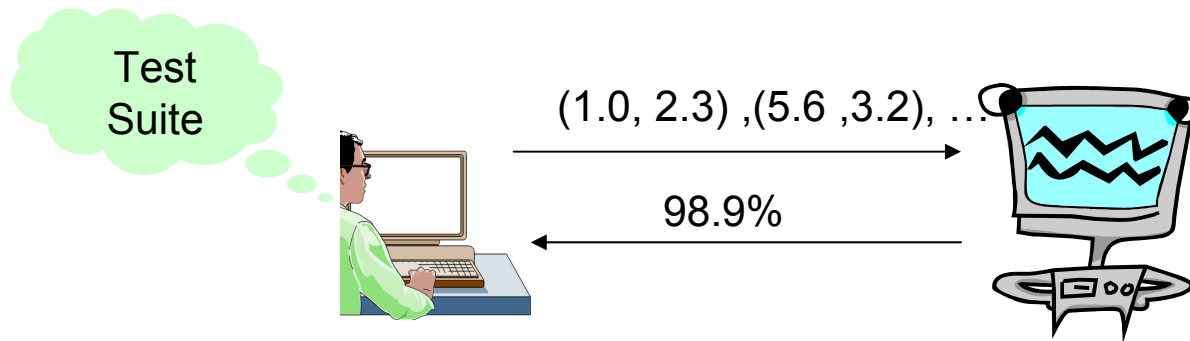


- Existen herramientas basadas en Metaheurísticas para **generar los casos de prueba** más adecuados para testear un programa



Introducción

- Cuando se usan Metaheurísticas la calidad de los casos de prueba se expresa mediante medidas de **cobertura de código**

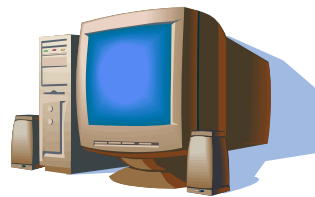


- ¿Hay **medidas estáticas** del código que se relacionen con la cobertura obtenida usando un generador de casos de prueba?

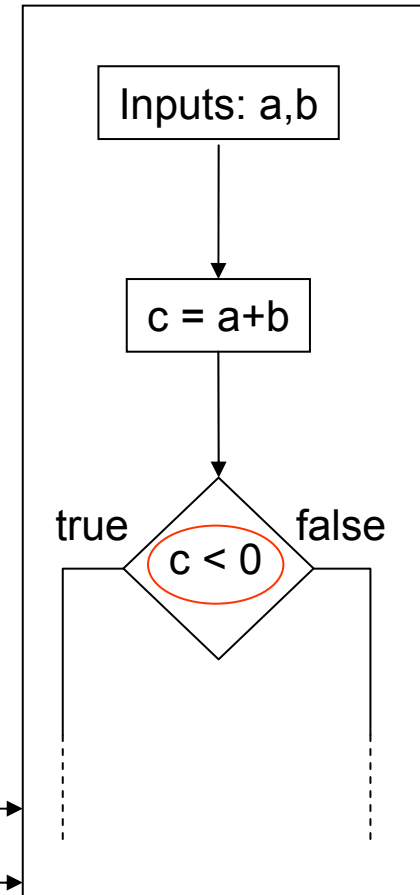
Generador

- **Generación dinámica** de casos de prueba:

- Se generan casos de prueba
- Se prueban los casos generados
- Se obtiene información de la traza
- Se generan nuevos casos de prueba



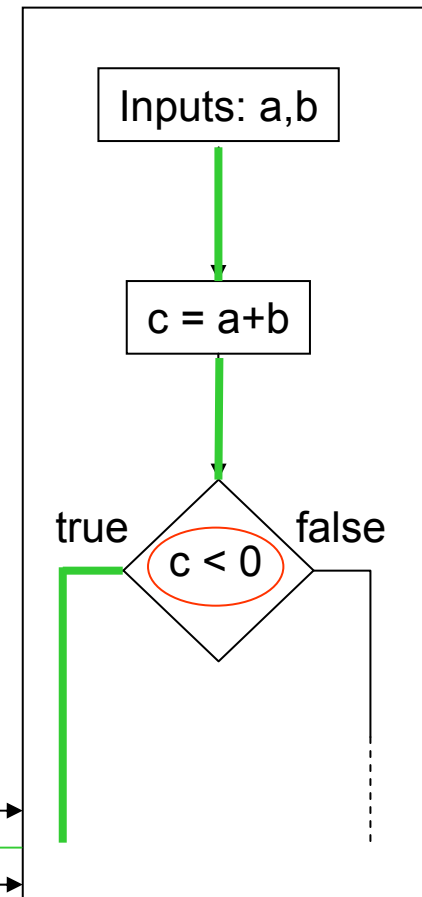
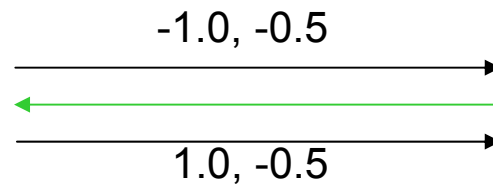
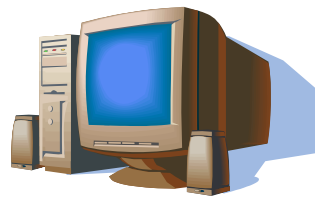
-1.0, -0.5
1.0, -0.5



Generador

- **Generación dinámica** de casos de prueba:

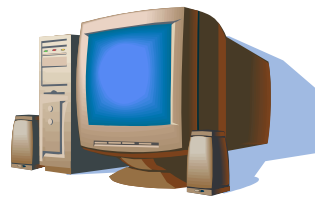
- Se generan casos de prueba
- Se prueban los casos generados
- Se obtiene información de la traza
- Se generan nuevos casos de prueba



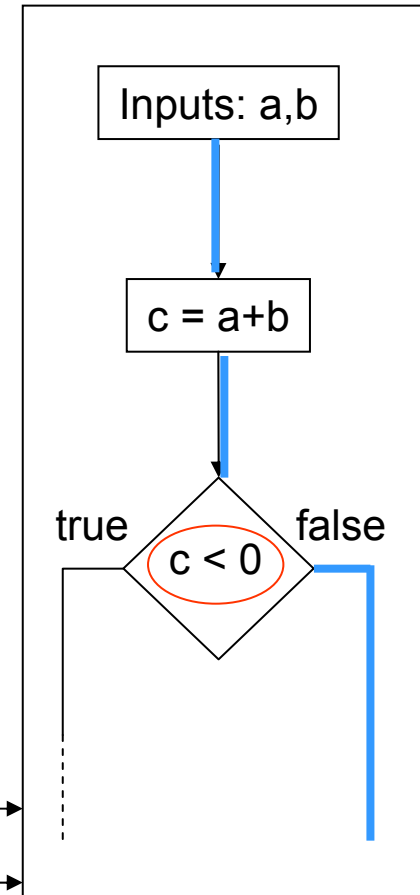
Generador

- **Generación dinámica** de casos de prueba:

- Se generan casos de prueba
- Se prueban los casos generados
- Se obtiene información de la traza
- Se generan nuevos casos de prueba

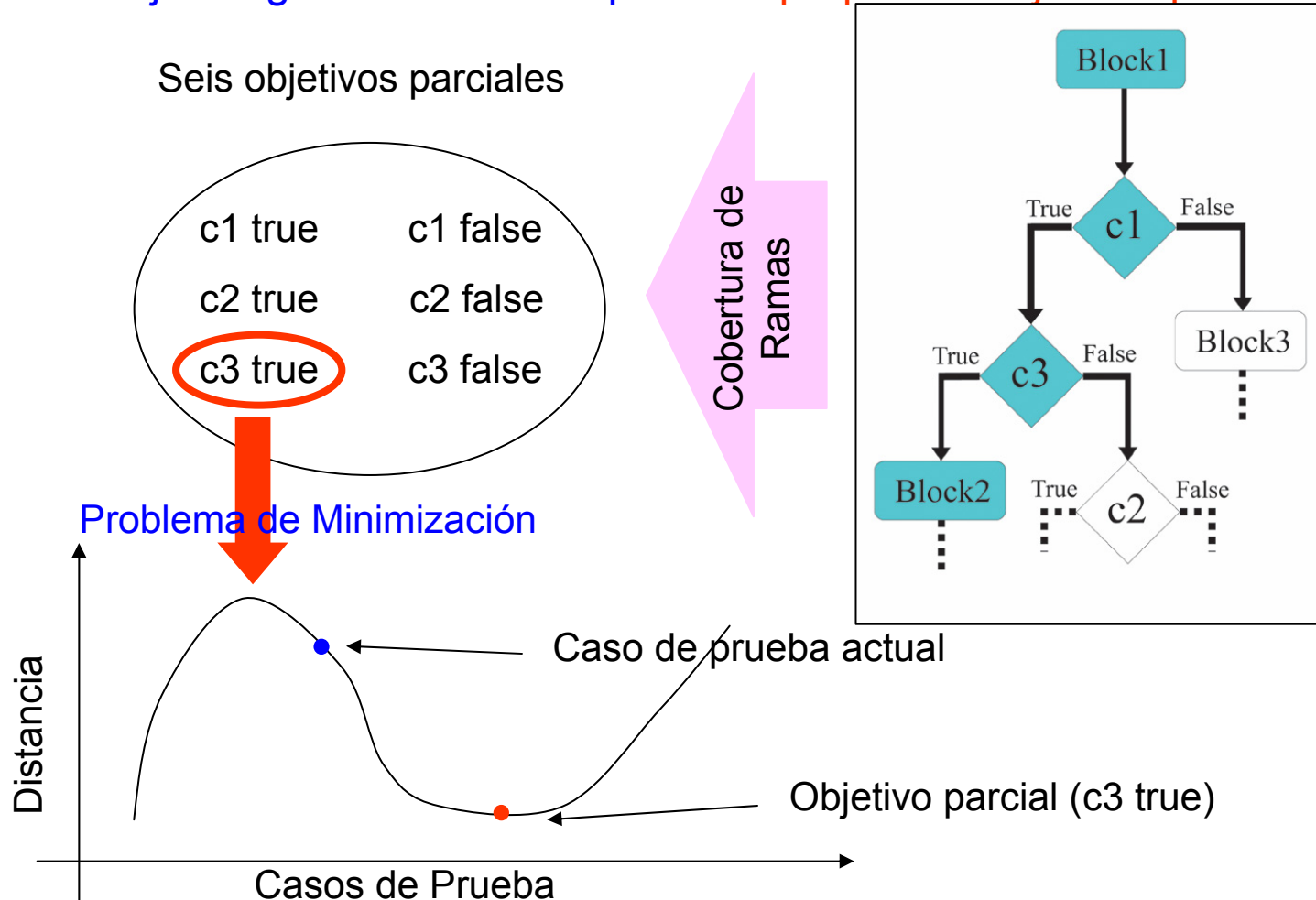


-1.0, -0.5
1.0, -0.5

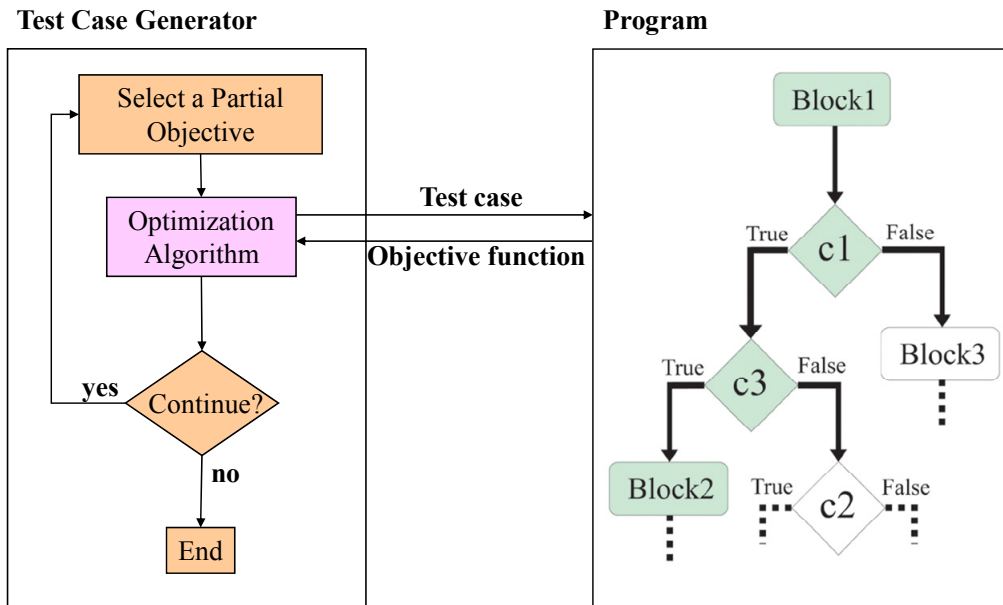


Generador

- El objetivo global se descompone en **pequeños objetivos parciales**



Generador



Evolutionary Strategy

```
t := 0;  
P(t) = Generate ();  
Evaluate (P(t));  
while not StopCriterion do  
    P'(t) := VariationOps (P(t));  
    Evaluate (P'(t));  
    P(t+1) := Replace (P'(t),P(t));  
    t := t+1;  
endwhile;
```

- El generador de casos de prueba selecciona el objetivo a cubrir en cada iteración buscando **maximizar la cobertura** de código.
- Nuestro generador crea una **tabla de cobertura** con los valores que satisfacen cada rama del programa.

Algoritmo de Optimización

• Estrategia Evolutiva

➤ Individuo

(0.2, -1.4, 3.5)

→ Vector Solución

(1.0, 10.3, 7.2)

→ Desviaciones estándar

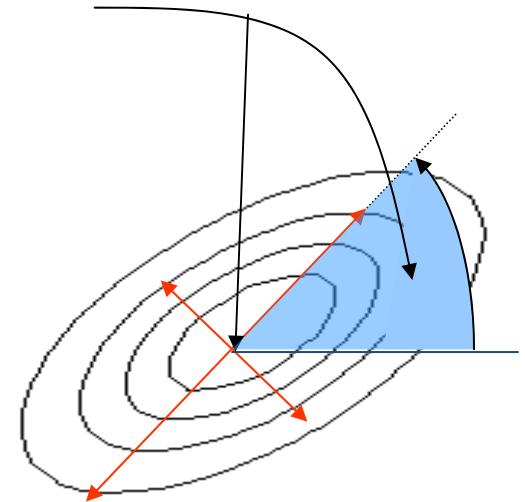
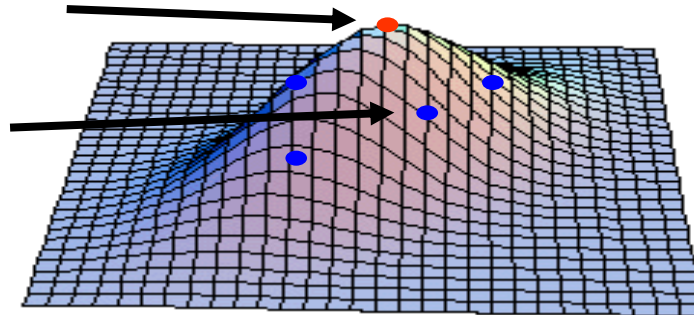
(1.7, 0.3, 2.1)

→ Ángulos

➤ Mutación Gaussiana

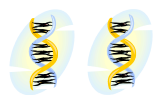
Solución Seleccionada

Nueva solución



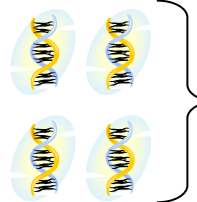
➤ Reemplazo → $(\mu + \lambda)$

Población previa

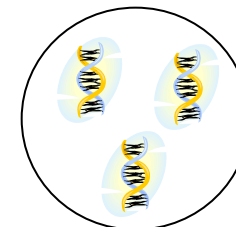


μ individuos

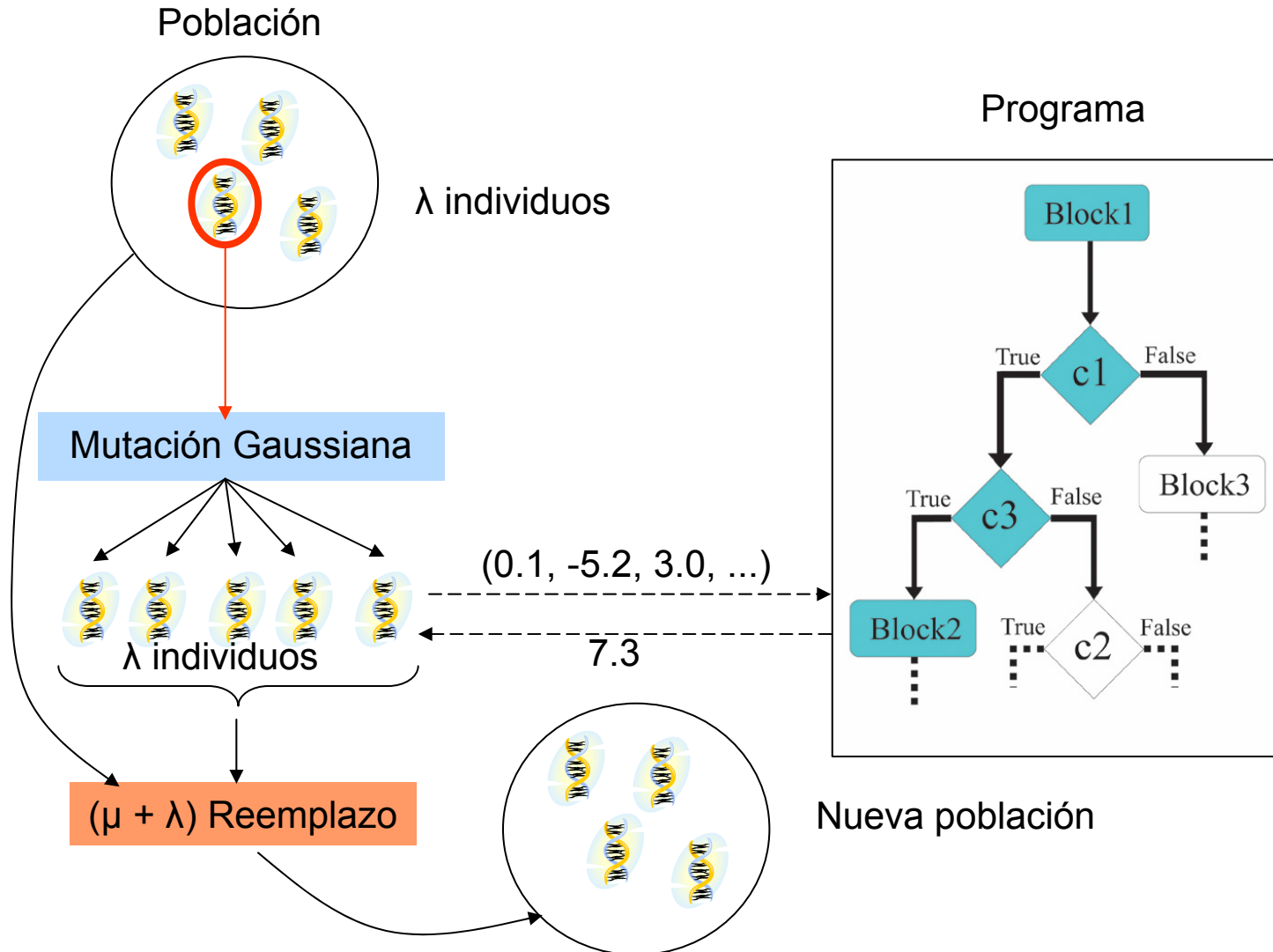
Nuevos individuos



Nueva población



Algoritmo de Optimización



Experimentos: Metodología

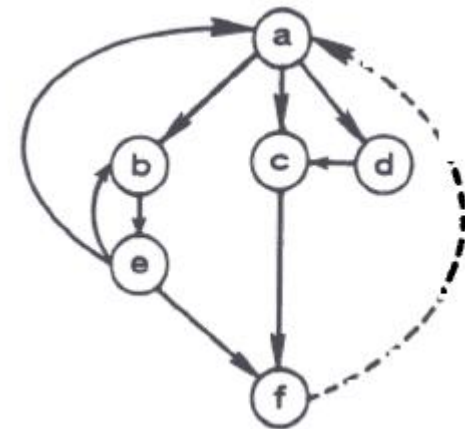
¿Hay alguna **medida estática** del programa que tenga una **correlación** con la **cobertura** obtenida?

- Analizamos `java.util.*` y calculamos las **medidas estáticas** del código fuente
- Usamos **medidas estáticas similares** a las obtenidas para generar el benchmark de programas.
- Generamos estos programas de forma **automática**
- Aplicamos nuestro generador de casos de prueba con configuraciones **(1+5)-ES** y **(25+5)-ES_c**, un total de 5 veces por programa

Experimentos: Benchmark

Benchmark de 3600 programas:

- **Generados automáticamente**
- **Parámetros conforme a una distribución de probabilidad**
 - Número de instrucciones
 - Número de condiciones atómicas por condición
 - Grado de anidamiento
 - Número de variables
- **Características medidas:**
 - Número de instrucciones: 26-388
 - Número de condiciones atómicas por condición: 1-7
 - Número total de condiciones: 1-39
 - Grado de Anidamiento: 1-7
 - Complejidad de McCabe: 4-142
 - E = Arcos del grafo
 - N = Nodos del grafo
 - P = Componentes conexos



$$v(G) = E - N + 2P;$$

$$v(G) = 9 - 6 + 2$$

Experimentos: Parámetros

- Parámetros de los algoritmos:

ES

Parámetro	Valor
Población	1
Selección	Aleatoria
Cruzamiento	No
Mutación	Gaussiana
Hijos (λ)	5
Reemplazo	$(\mu+\lambda)$
Parada	Obj. o 1000 evals.

ES_c

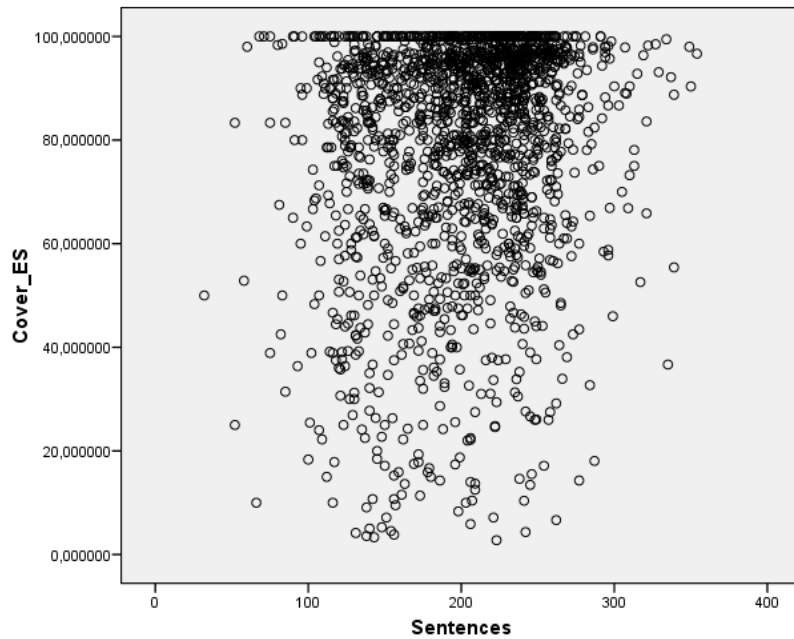
Parámetro	Valor
Población	25
Selección	Aleatoria
Cruzamiento	1 punto
Mutación	Gaussiana
Hijos (λ)	5
Reemplazo	$(\mu+\lambda)$
Parada	Obj. o 1000 evals.

- 3600 programas * 5 ejecuciones independientes * 2 algoritmos = 36000 ejecuciones

Experimentos

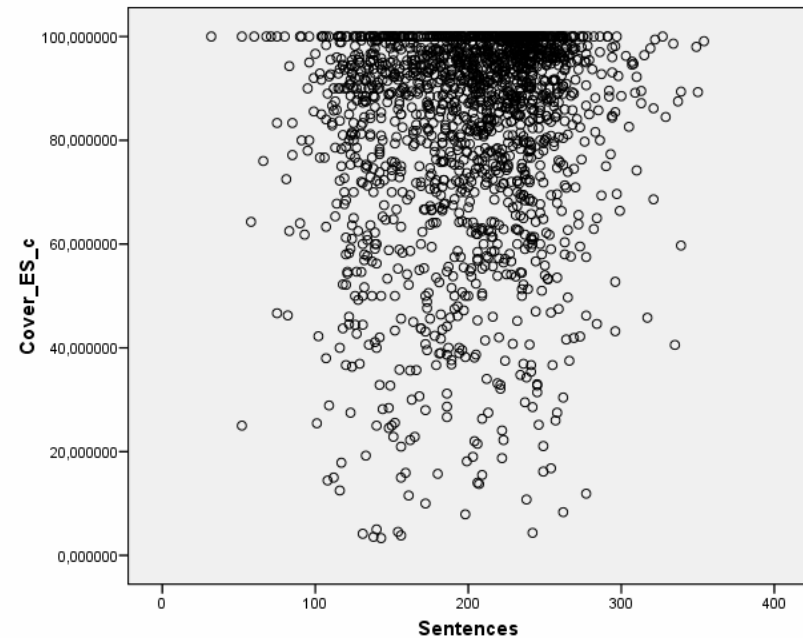
- Número de instrucciones:

ES



Correlación: 18,8 %

ES_c



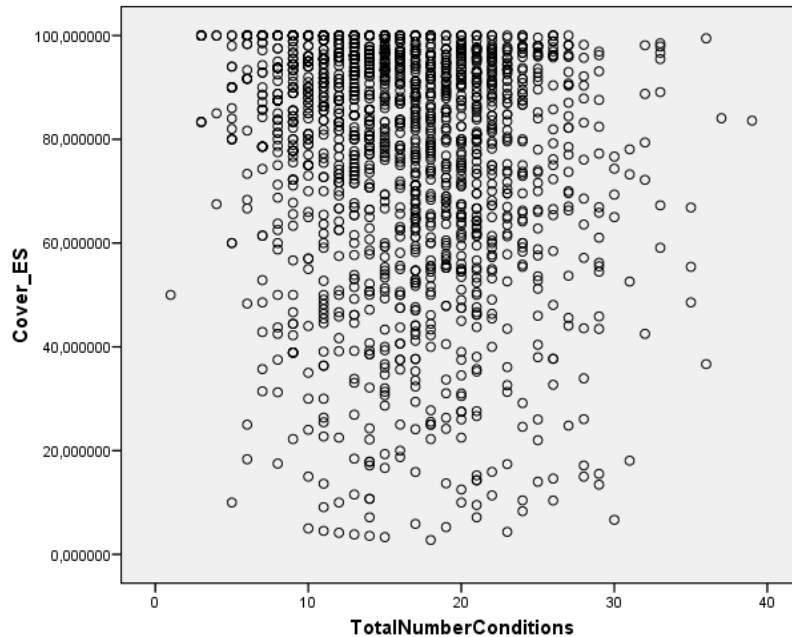
Correlación: 13,4 %

- No tiene influencia en la cobertura

Experimentos

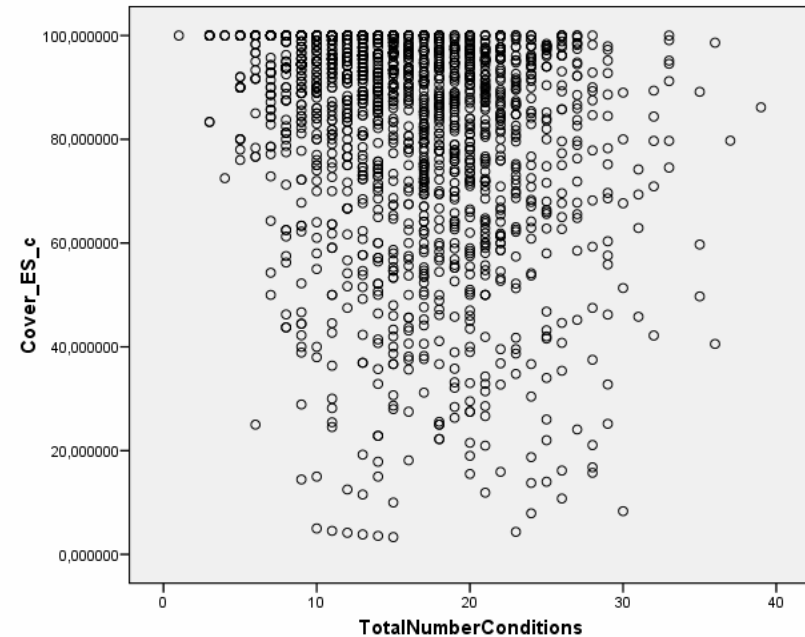
- Número total de condiciones:

ES



Correlación: -16,4 %

ES_c



Correlación: -21,6 %

- Programas con más condiciones son más difíciles de testear

Experimentos

- Número de condiciones atómicas por condición:

Conditions	ES_c	ES
1	83,59% _{21.69}	77,74% _{23.49}
2	82,85% _{20.33}	78,54% _{21.82}
3	85,15% _{19.82}	81,39% _{21.53}
4	88,04%_{16.42}	83,23%_{19.87}
5	85,06% _{19.19}	80,50% _{21.53}
6	84,16% _{19.58}	79,12% _{23.09}
7	81,24% _{21.18}	76,26% _{23.74}
r	-0,50 %	0,30 %

- No tenemos correlación lineal

Experimentos

- Grado de anidamiento:

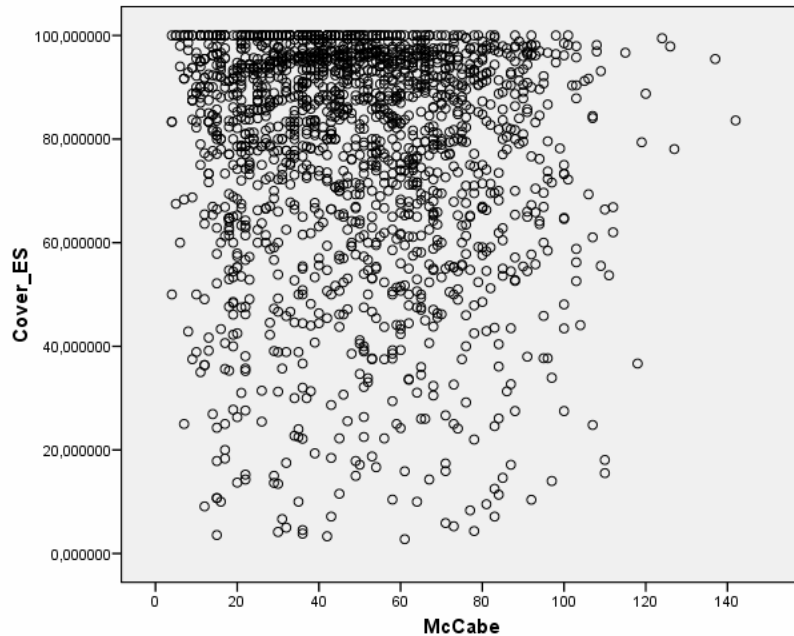
Nesting	ES_c	ES
1	96,03% _{05,74}	93,56% _{08.15}
2	94,07%08,85	89,97%11.29
3	90,02%13,67	85,29%16.71
4	85,06%16,43	80,09%19.18
5	77,83%22,53	72,02%23.90
6	73,02%24,80	67,47%24.66
7	64,45%25,96	58,41%27.16
r	-47,00 %	-45,70 %

- Tenemos correlación lineal inversa
- Las ramas anidadas suponen un gran reto para el generador

Experimentos

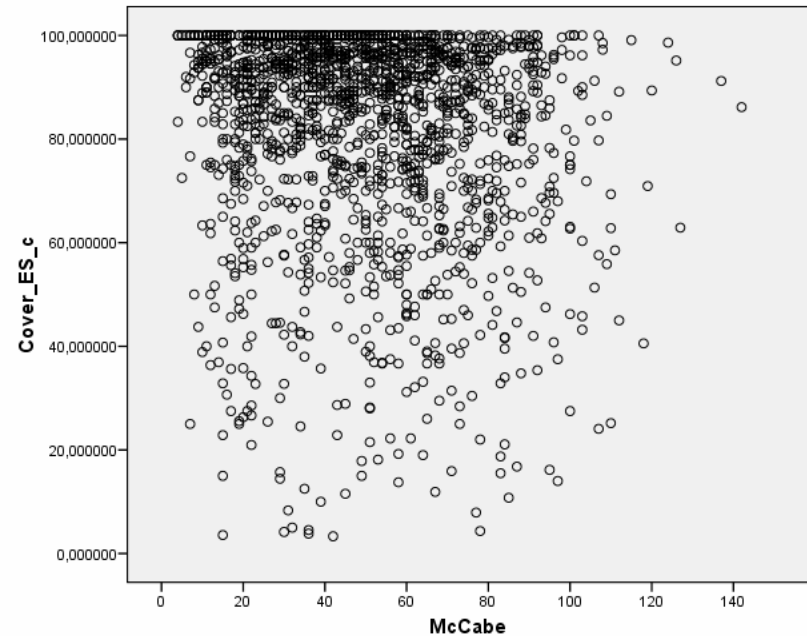
- McCabe:

ES



Correlación: -4,8 %

ES_c



Correlación: -8,6 %

- No está correlacionado: grado de anidamiento
- No podemos usar la complejidad de McCabe como indicador de dificultad para testear

Conclusiones

- Analizamos la correlación de la cobertura obtenida con 2 algoritmos diferentes y 5 medidas estáticas del código
- La cobertura no esta correlacionada con:
 - Número de instrucciones
 - Número de condiciones por condición
 - Número total de condiciones
 - Complejidad de McCabe
- La cobertura desciende si aumentamos el grado de anidamiento
- El grado de anidamiento es el factor más determinante
- La complejidad de McCabe es un mal indicador de la dificultad de testeo automático

Trabajo Futuro

- Analizar otras medidas estáticas y dinámicas con el fin de proponer una medida fiable sobre la dificultad de testear automáticamente un programa
- Proponer una herramienta que en base a medidas estáticas, parametrize el generador de casos de prueba
- Realizar un análisis exhaustivo con programas orientados a objetos para dar una medida fiable de su dificultad para testear
- Comparar esta medida que vamos a diseñar con otras existentes de complejidad, principalmente en software real

Gracias
por su atención

Javier Ferrer
ferrer@lcc.uma.es