Generazione Automatica di Test in JUnit con EvoSuite

Andrea Pennati

Università degli Studi di Milano

5 luglio 2021

Indice

- Generazione Automatica di Test:
  - obiettivi,
  - funzionamento,
  - problemi.
- Una possibile ottimizzazione:
  - EvoSuite,
  - Whole test suite generation,
  - Mutation-based assertion generation,

Generazione Automatica di Test:

La generazione automatica di test nasce con l'idea di delegare al sistema la parte più costosa e laboriosa dello sviluppo di un software, ovvero il testing.

- prevede necessariamente l'esistenza di un sistema già funzionante,
- in contrapposizione rispetto allo sviluppo guidato dai test.

Generazione Automatica di Test: quando serve?

Gli obiettivi nell'utilizzare un software di questo tipo sono essenzialmente due:

- vogliamo aumentare il numero di test,
- massimizzare il livello di copertura del codice.

Generazione Automatica di Test: come funziona?

Preso in input un software il tool genera un insieme di test basandosi sul comportamento attuale del sistema:

- verificano il comportamento corrente, non quello desiderato,
- il programmatore deve verificare che i test siano corretti, ovvero che il comportamento corrente del sistema coincida con quello desiderato.

Generazione Automatica di Test: problema

Di cosa abbiamo bisogno?

- casi di test che eseguano il software in modo sistematico,
- oracoli che valutino la correttezza del comportamento osservato.

Problema dell'oracolo:

- Gli errori a volte possono essere rilevati automaticamente se portano ad arresti anomali del programma, deadlock o violano una specifica formale; tuttavia non è sempre vero,
- E' necessario l'intervento del programmatore che conosce la semantica del programma.

Generazione Automatica di Test

- Necessario fornire all'utente piccole suite di test che possano essere verificate manualmente.
- controllando che i test generati verifichino il sistema in modo corretto.

EvoSuite

- EvoSuite è uno strumento che automatizza questo compito producendo sistematicamente suite di test, più piccole possibili, che raggiungono un'elevata copertura del codice definendo asserzioni.
- è basato su un algoritmo evolutivo che si ispira al principio di evoluzione degli esseri viventi.
- sfrutta 2 approcci:
  - Whole test suite generation,
  - Mutation-based assertion generation.

EvoSuite: Whole test suite generation

- Questo approccio non produce casi di test che mirano a coprire un determinato obiettivo, ma si concentra sull' intera suite di test focalizzandosi su una copertura totale del codice,
- La tecnica si basa sull'implementazione di un algoritmo evolutivo, in particolar modo sfrutta il concetto della genetica.
- I concetti alla base di questo algoritmo sono:
  - Crossover,
  - Mutazione,
  - Selezione.

EvoSuite: Crossover, Mutazione e Selezione

- Inizialmente viene creata una popolazione(insieme di suite di test) completamente random,
- la popolazione di soluzioni candidate viene evoluta utilizzando operatori che imitano l'evoluzione naturale come crossover e mutazione,
- le possibili soluzioni vengono valutate mediante una funzione di fitness,
- quelle con la fitness migliore vengono prese e usate per generare la futura prole.

EvoSuite: Crossover, Mutazione e Selezione

- Crossover: genera due figli a partire da due suite di test genitori scambiando casi di test.

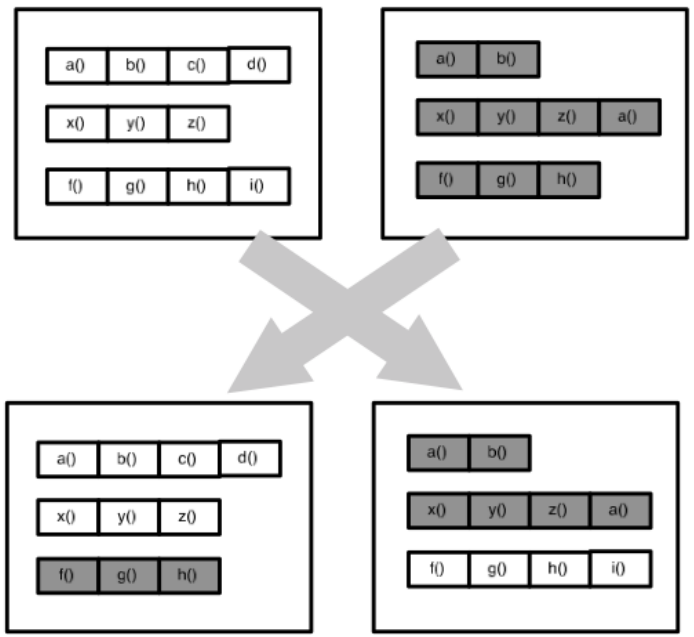

Figura: Crossover

EvoSuite: Crossover, Mutazione e Selezione

- Mutazione: i test generati vengono mutati, creando nuovi test con un comportamento leggermente diverso rispetto alla controparte originale. La mutazione avviene applicando con un certo grado di probabilità tre operazioni:
  - Remove: viene rimosso uno statement,
  - Change: viene cambiato uno statement.
  - Insert: viene aggiunto uno statement.
- In generale queste operazioni sono delicate perchè, andando a lavorare a livello di statement, bisogna tenere in considerazione le varie dipendenze.

EvoSuite: Crossover, Mutazione e Selezione

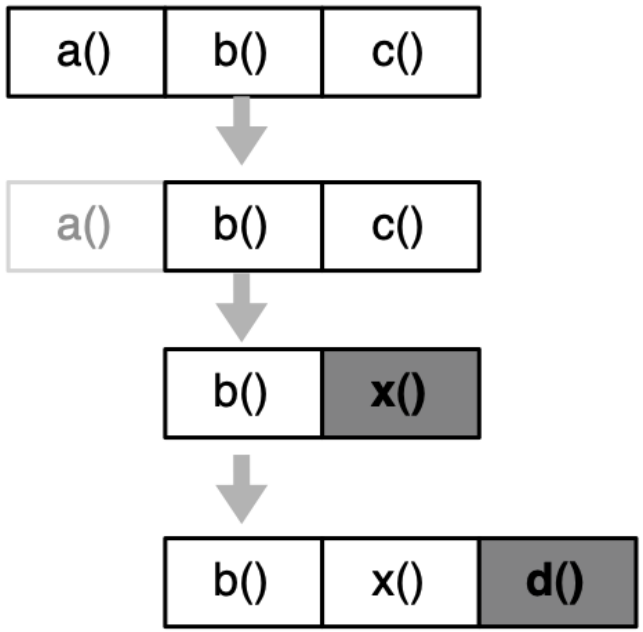

Figura: Mutazione

EvoSuite: Crossover, Mutazione e Selezione

- Selezione: operazione che seleziona i migliori elementi di una popolazione. La selezione viene effettuata valutando il risultato di una funzione chiamata *fitness*,

- Presa una suite di test T, il valore di fitness viene misurato eseguendo tutti i test $t \in T$ e tenendo traccia dell'insieme dei metodi eseguiti MT nonché della distanza minima dei rami $dmin(b, T)$ per ciascun ramo $b \in B$.

$$\text{fitness}(T) = |M| - |M_T| + \sum_{b_k \in B} d(b_k, T)$$

Figura: La funzione di fitness

EvoSuite: Crossover, Mutazione e Selezione

```
1  current_population ← generate random population
2  repeat
3     Z ← elite of current_population
4     while |Z| ≠ |current_population| do
5        P1,P2 ← select two parents with rank selection
6        if crossover probability then
7           O1,O2 ← crossover P1,P2
8        else
9           O1,O2 ← P1,P2
10       mutate O1 and O2
11       fP = min(fitness(P1),fitness(P2))
12       fO = min(fitness(O1),fitness(O2))
13       lP = length(P1) + length(P2)
14       lO = length(O1) + length(O2)
15       TB = best individual of current_population
16       if fO < fP ∨ (fO = fP ∧ lO ≤ lP) then
17          for O in {O1,O2} do
18             if length(O) ≤ 2 × length(TB) then
19                Z ← Z ∪ {O}
20             else
21                Z ← Z ∪ {P1 or P2}
22       else
23          Z ← Z ∪ {P1,P2}
24    current_population ← Z
25 until solution found or maximum resources spent
```

Figura: L'algoritmo

EvoSuite: Whole test suite generation

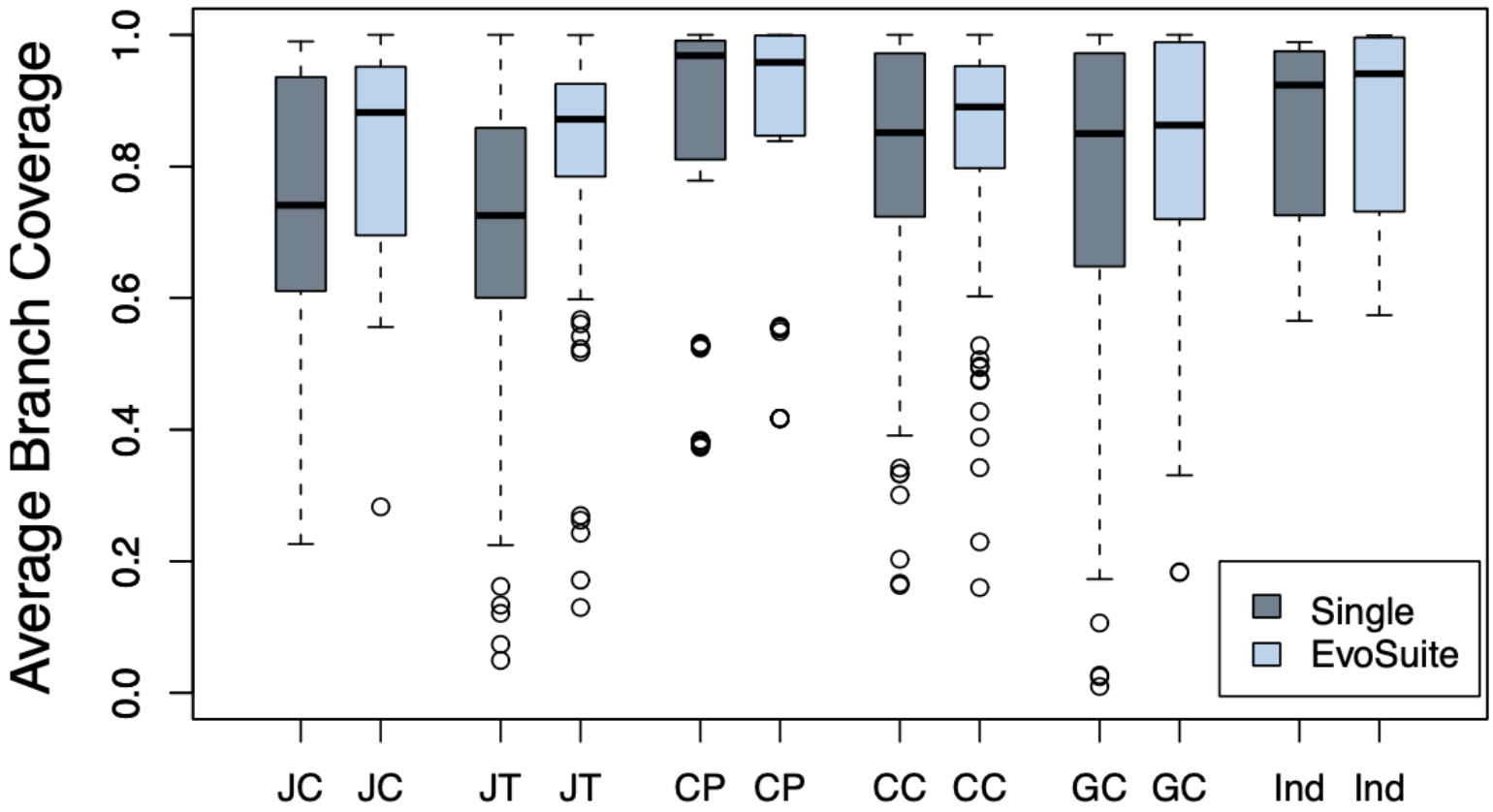

Figura: Copertura media raggiunta da EvoSuite su 6 casi di studio

EvoSuite: Mutation-based assertion generation

- A partire dal codice vengono generati dei mutanti. Viene fatta una valutazione sulla base di quanti mutanti riescono ad individuare le asserzioni:
  - se un test non è in grado di distingure il sistema originale dai suoi mutanti, questo viene considerato superfluo e quindi scartato,
  - in caso contrario viene tenuto.
- Infine viene applicato un processo di sfoltimento delle asserzioni che rende i testi più leggibili aiutando così il programmatore.

EvoSuite: Mutation-based assertion generation

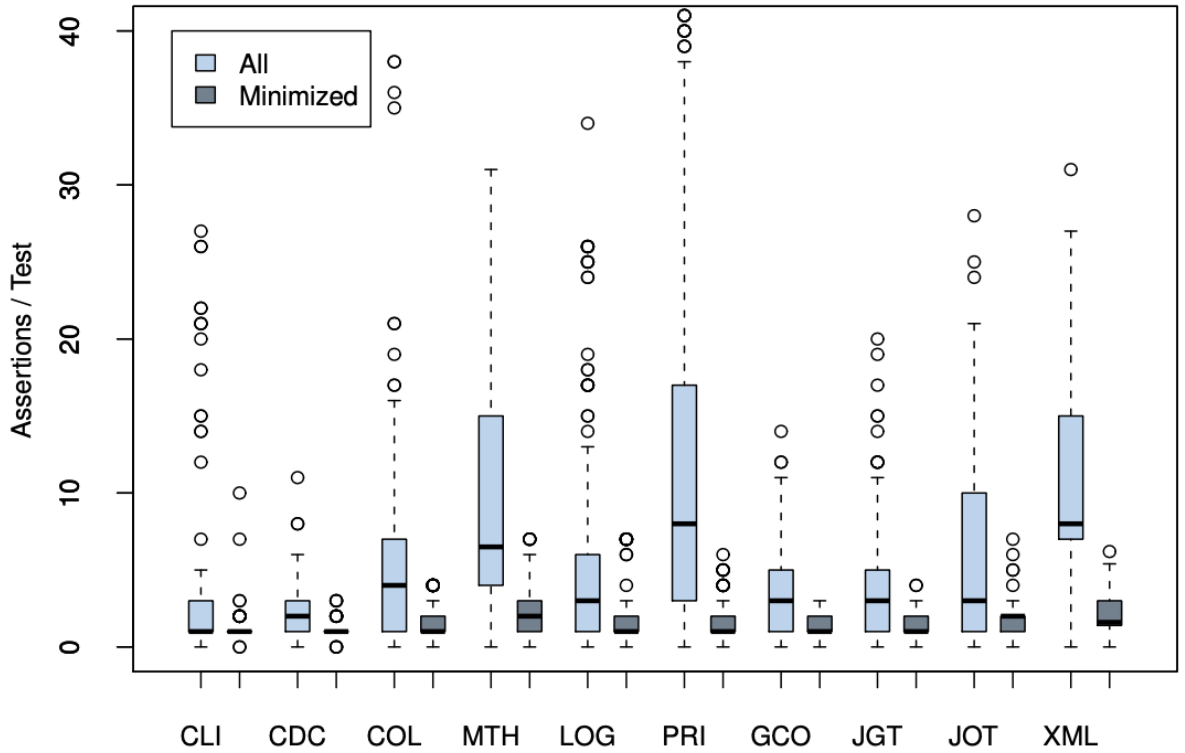

Figura: I boxplot riassumono le statistiche sulle asserzioni prima ("All") e dopo ("Minimized") applicando la minimizzazione basata sulla mutazione.

EvoSuite: Pro e Contro

- Vantaggi:
  - personalizzazione dei criteri di copertura,
  - generazione di test banali che potrebbero sfuggire al programmatore.
- Svantaggi:
  - non adatto al multi-threading,
  - risultati non ottimali in applicazioni dipendenti dal web o dal file system.

Bibliografia

- G. Fraser and A. Arcuri, *"EvoSuite: automatic test suite generation for object-oriented software,"* in Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, New York, NY, USA, 2011, pp. 416-419.

- Fraser and A. Arcuri, *"Evolutionary Generation of Whole Test Suites,"* in International Conference On Quality Software (QSIC), Los Alamitos, CA, USA, 2011, pp. 31-40.