УДК 004.896

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ДЛЯ АНАЛІЗУ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОДУ

Гончар Л.І.[1)], **Томашівський І.М.**[2)], **Олійник А.П.**[3)], **Ядчишин О.В.**[4)], **Опалько О.О.**[5)]
Західноукраїнський національний університет
[1)]к.е.н., доцент; [2)]магістрант; [3)]аспірант;[4)]аспірант; [5)]аспірант

I. Постановка проблеми

Генерація тестових даних – складний та трудомісткий процес, що вимагає великих зусиль. Тому автоматизація цього процесу, хоча б часткова, є актуальним завданням, вирішення якого могло б підвищити ефективність тестування програмного забезпечення. Однією з цілей автоматичної генерації тестових даних є створення таких тестових наборів, що забезпечило б достатній рівень якості кінцевого продукту шляхом перевірки більшої частини різних шляхів коду, тобто забезпечило б максимальне покриття коду у відповідність до обраних критеріїв оптимальності (наприклад, критерії покриття операторів чи гілок) [1-4]. Підібрати такі набори даних вручну трудомістке завдання, тому в роботі пропонується автоматизація цього процесу з використанням генетичного алгоритму [5,6].

II. Мета роботи

Метою дослідження є розробка методів та засобів для аналізу та тестування програмного коду з використанням генетичного алгоритму.

III. Метод генерації тестових наборів даних із використанням генетичного алгоритму

На рисунку 1 представлений тестований код, написаний мовою C#, а також побудований для нього граф потоків управління з вагами, визначеними на основі метрики оцінки складності коду NOD зпочатковим значенням 100. В результаті випадкової генерації початкової популяції було отримано 4 набори тестових даних (хромосом): (10,5,12);(3,4,10); (25,30,11); (5,3,17).

```
0.  public static void Main(int m, int i, int n) {
1.  int sum = 0, avg = 0;
2.  if (m <i)
3.  {
4.  use_m = m;
5.  }
6.  int[] a = new int[m];
7.  Console.WriteLine("Enter the Array Elements ");
8.  for (j = 0; j< m; j++)
9.  {
10. a[j] = i;
11. if (a[j] > n)
12. {
13. a[j] = n
14. }
15. }
16. for (k = 0; k < m; k++)
17. {
18. sum += a[k];
19. }
20. avg = sum / m;
21. Console.WriteLine("Average is {0}", avg);
22. Console.ReadLine();
23. }
```

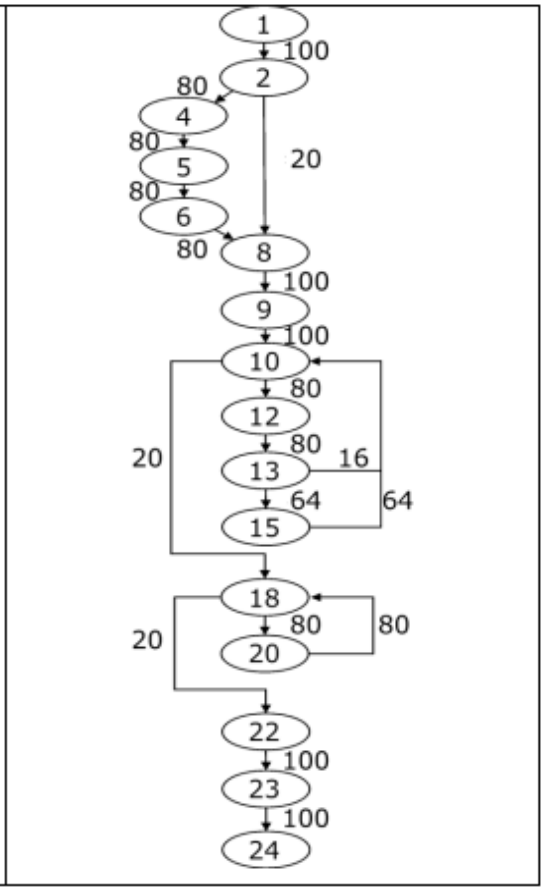

Рисунок 1 –Результати побудови графа потоків для тестованого програмного коду

Для кожної хромосоми розраховується значення функції пристосованості, після чого вони сортуються у порядку зменшення їїзначень. У таблиці 1 показані тестові набори, значення функції пристосованості та ранг. Найкращі хромосоми виділені курсивом і будуть використовуватися для схрещування. Внаслідок відбору для схрещування буливибрані набори 2 і 3. Два інші набори виключаються, а популяція буде доповнена нащадками відібраних хромосом. Для простоти ілюстрації тут немає використовується механізм змішування.

Таблиця 1

Початкові тестові набори даних (популяція)

№	Набір даних	F(X)	Ранг
3	(25,30,11)	1308	1
2	(3,4,10)	1196	2
1	(10,5,12)	896	3
4	(5,3,17)	896	3

Схрещування проводиться уніфікованим методом, тобто хромосомами з трьома змінними можуть бути розділені за першою та другою позиціями. Мутація кожного гена відбувається із ймовірністю 0.05 на інтервалі (0, 50). Очевидно, що при такому шансі мутації з малим розміром популяції такількістю змінних, він є недостатнім для забезпечення суттєвого розмаїття.

На цьому прикладі можна добре побачити недоліки використаннябезперервного генетичного алгоритму без механізму змішування. На популяції маленькоїрозмірності буквально за 1 покоління значення тестових наборів почали повторюватися. Друга змінна, при наступному схрещуванні, не буде змінюватися, а третє вагатиметься, приймаючи всього два можливі значення (10, 11). Через невеликий шанс мутації алгоритм, швидше за все, будеформувати лише невиразні хромосоми. Однак для цього прикладу це іневажливо, оскільки значення 1308 є оптимальним значенням функції пристосованості для прикладу, що розглядається, і воно і було отримано відразу на етапі випадкової генерації.

Для досягнення максимального покриття можна використовувати багаторазовий запуск алгоритму генерації тестових даних для одного шляху, тимсамим послідовно збільшуючи значення покриття наборами, що проходять повідмінним шляхам.

З кожною новою ітерацією значення функції пристосованості поступово знижується, тобто кожен запуск покриває дедалі менше новихоператорів. Винятком є п'ята ітерація, коли значення функції вище, у попередньому. Це говорить про те, що вихід на шлях, покритий п'ятим набором даних, обмежений сильнішими умовами. Також можлива ситуація, коли алгоритм при випадковій генерації отримав дані для виходу наменш складний шлях, і, при поточних параметрах ГА, йому не вистачило часу дляпошуку тестових наборів для виходу потенційно складний шлях. В такомуразі можливе встановлення інших параметрів ГА, щоб дати алгоритму більшечасу для пошуку даних для інших шляхів.

Висновок

Розроблено алгоритм генерації тестових даних для одного складного шляху програмного коду, що визначається вагами операторів, що знаходяться на ньому. Було виявлено, що при використанні цього методу формується велика кількість невиразних хромосом, що говорить про його неефективність при формуванns тестових наборів. Досліджено алгоритм багаторазового запуску алгоритму генерації тестових даних для одного шляху з метою досягнення повного покриття тестованої програми. На відміну від існуючих методів, алгоритм дозволяє згенерувати необхідну кількість тестових наборів для досягнення необхідного значення покриття.

Список використаних джерел

1. Agarwal M. Software Testing Basics: Types of Bugs and Why They Matter –Access mode: https://www.techbeamers.com/static-testing-vs-dynamic-testing/ (accessdate: 16.03.2023).
2. Крепич С.Я. Співак І.Я. Якість програмного забезпечення та тестування: базовий курс. Тернопіль, Паляниця В.А. 2020. – 479с.
3. Static Testing vs Dynamic Testing: What's the Difference? – Access mode: https://www.guru99.com/static-dynamic-testing.html (accessdate: 16.03.2023).
4. Spillner A., Linz T., Schaefer H. Software Testing Foundations. A Study Guide for the Certified Tester Exam // RockyNook – 2014 – 305 p.
5. Korel B. Automated software test data generation // IEEE Transactions on Software Engineering – 1990 – N16 – P. 870–879.
6. Bird D., Munoz C.C. Automatic generation of random self-checking test cases // IBM Systems Journal –1983 – N 22 – P.229–245.