УДК 004.02

ЖДАНОВА О.Г.
ПАДАЛКО О.Б.

ПРО ПРОБЛЕМУ ГЕНЕРАЦІЇ ПЛАНІВ ТЕСТУВАННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ ВЕРИФІКАЦІЇ ПРОГРАМНОГО ПРОДУКТУ

Робота присвячена задачі автоматизації тестування – побудові планів тестування програмних продуктів за їх діаграмами станів. Обговорюються проблеми, що виникають при вирішенні задачі, та шляхи їх подолання. Дається теоретичне обґрунтування розроблених алгоритмів. Подається опис можливостей програмного забезпечення, створеного згідно матеріалу роботи.

Работа посвящена задаче автоматизации тестирования – созданию планов тестирования программных продуктов по их диаграммам состояний. Обсуждаются проблемы, которые возникают при решении задачи, и пути их преодоления. Дается теоретическое обоснование разработанных алгоритмов. Описываются возможности программного обеспечения, созданного в соответствии с материалами работы.

Research is dedicated to the software testing automation task – application testing plan generation based on their statechart diagram. Theoretical background is included. Problems that appear while solving the task are discussed along with the way to overcome them. It includes feature list of created application software which was developed based on the research material.

Вступ

Створення програмної системи – складний та багатогранний процес. Для підвищення якості та надійності програмних продуктів потрібно значну увагу приділяти тестуванню. Тестування програмного продукту являє собою перевірку адекватності результатів його роботи початковим вимогам, тобто відповідності створеної функціональності вимогам технічного завдання. Для великих програмних продуктів практично не можливо виконати повне тестування без застосування засобів автоматизації цього процесу. Отже, постає проблема розробки плану тестування з метою покращення якості тестування.

Переважна більшість сучасних розробок орієнтована на перевірку правильності програмного коду [1, 2]. В даній роботі розглядається підхід до тестування програмної системи за відповідною їй діаграмою станів, яка створюється на етапі проектування. Така діаграма має один вхід (початкова точка програми) та один або декілька виходів, при чому можлива наявність циклів та петель. Такий підхід до тестування є зручним та надійним, оскільки на діаграмі відображені усі можливі переходи між станами системи, що може бути враховано при побудові плану. Далі діаграму станів будемо представляти у вигляді орієнтованого графу (рис. 1).

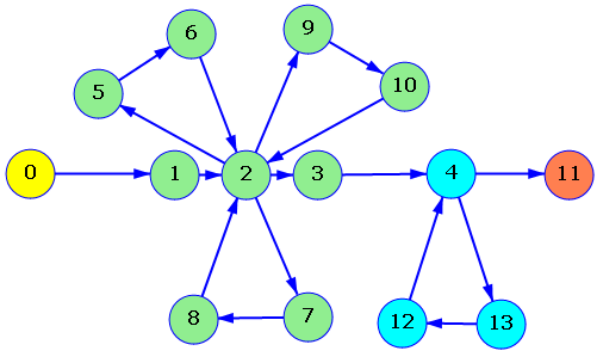

Рис. 1

Кожному тесту ставиться у відповідність шлях, який починається в початковій вершині і завершується в одній з кінцевих. Під планом тестування розуміється сукупність тестів, яка задовольняє певним умовам. В залежності від цих умов можливі декілька підходів до побудови плану тестування по діаграмі (графу) станів системи. Поширеними є покриття гілками (Branch Coverage або Edge Coverage), покриття шляхами - Path Coverage (яке має багато різновидностей, тому надалі під Path Coverage будемо мати на увазі Edge-Pair Coverage [2]). З перерахованих план тестування за покриттям типу Path Coverage дає найбільш високу якість.

При побудові планів тестування може бути застосовано декілька стратегій. З точки зору тестувальника (реалізації процесу тестування) виділяються такі стратегії: „довга”, „коротка” та стратегія, що враховує вагу дуг.

Під „довгою” стратегією розуміється знаходження плану тестування, який включає в себе найбільш довгі тести. „Коротка” стратегія вимагає знаходження плану тестування, який складався б з більш коротких тестів.

1 Постановка задачі

Дано орієнтований граф з однією початковою і декількаами кінцевими вершинами. Побудувати покриття типу Path Coverage, тобто знайти таку (мінімальну за потужністю) множину шляхів, яка покриває усі комбінації пар дуг вхід–вихід для кожної вершини графу (далі цю умову будемо позначати як умову УКПДВВ) і відповідає заданій стратегії.

2 Загальна схема розв’язання

Якщо граф системи не містить циклів, то шуканим планом тестування буде сукупність усіх шляхів з початкової точки в кінцеві. Цей окремий випадок є тривіальним і не є для нас цікавим. Наявність циклів різко збільшує кількість можливих планів тестування.

Для розв’язання поставленої задачі пропонується така схема:

Етап 1. Формування множини шляхів з початкової точки в кінцеві.

Етап 2. Формування множини тріад.

Етап 3.Розв’язання задачі.

3 Формування множини шляхів з початкової точки в кінцеві

Цей етап складається з трьох кроків:

Крок 1. Знаходження множини простих шляхів та множини циклів.

Крок 2. Формування послідовностей циклів.

Крок 3. Комбінування множин шляхів і послідовностей циклів.

3.1 Знаходження множини простих шляхів та множини циклів

На цьому кроці алгоритмом обходу графу в глибину знаходиться множина W_0 простих шляхів (шляхів, що не містять циклів), множина C циклів та точки входу в цикли.

Точка входу в цикл – вершина шляху, що є спільною для цього шляху і деякого циклу. Результатом роботи цього кроку є дерево простих шляхів та циклів (рисунок 2). (Зауважимо, що цей крок є спільним для побудови усіх видів покриттів).

3.2 Формування послідовностей циклів та множині розширених шляхів

Після визначення множин простих шляхів та циклів необхідно виконати їх комбінування (вбудування циклів в шляхи). У випадку, коли мають місце цикли, задача забезпечення основної вимоги Path Coverage має ряд нетривіальних особливостей. Розглянемо ці особливості на прикладі графу, що наведений на рисунку 1 (відповідне дерево шляхів та циклів показано на рис. 2).

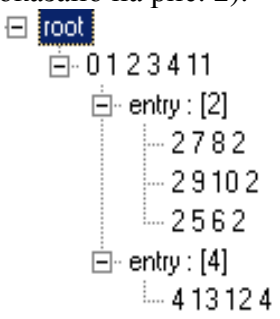

Рис. 2

Вершини 2 та 4 мають по декілька вхідних та вихідних дуг. Для забезпечення УКПДВВ у результуючий множині шляхів:

– через вершину 2 (вершину, яка має чотири входи та чотири виходи) необхідно забезпечити проходження шістнадцятьма підшляхами 1-2-5, 1-2-9, 1-2-3, 1-2-7, 6-2-5, 6-2-9, 6-2-3, 6-2-7, 10-2-5, 10-2-9, 10-2-3, 10-2-7, 8-2-5, 8-2-9, 8-2-3, 8-2-7;

– через вершину 4 (у неї два входи та два виходи) необхідно забезпечити проходження чотирма підшляхами 3-4-11, 3-4-13, 12-4-11, 12-4-13.

Далі такі підшляхи будемо називати ***тріадами*** (Edge-Pair [2])

3.2.1 Послідовності циклів

Якщо на деяку вершину i «навішано» один або декілька циклів, то для виконання умови УКПДВВ необхідно сформувати ***послідовності циклів*** (ПЦ), які б забезпечили покриття усіх тріад, що містять у собі цю вершину i. Проілюструємо сказане на прикладі графу з рисунку 1. Цей граф має один простий шлях 0–1–2-3-11 та чотири незалежних цикли (цикли називаються ***незалежними***, якщо вони не мають спільних дуг):

цикл1: 2-7-8-2;

цикл2: 2-9-10-2;

цикл3: 2-5-6-2;

цикл4: 4-13-12-4.

По-перше: немає необхідності формувати шляхи, що включають в себе тільки один

прохід по циклу. Це означає, що немає сенсу формувати шлях, що проходить по циклу 1 тільки один раз:

0–1–цикл1-3-11 (0-1-2-7-8-2-3-11)

оскільки при цьому не буде покрита тріада 8-2-7 (кінцева дуга циклу 1 + початкова дуга циклу 1). Отже, для покриття такого типу тріад необхідно сформувати такі послідовності циклів (ПЦ), що є дублями циклів:

по точці входу 2:

цикл1–цикл1, цикл2–цикл2, цикл3–цикл3;

по точці входу 4:

цикл4–цикл4.

По-друге, якщо на деяку вершину i «навішано» декілька циклів (в нашому випадку це вершина 2), то для неї необхідно сформувати змішані ПЦ, які б забезпечили покриття усіх тріад, що є перехідними з одного циклу в другий. Тобто, необхідно, щоб деякий шлях (деякі шляхи) включали в себе такі ПЦ:

цикл1–цикл2, цикл1–цикл3,

цикл2–цикл1, цикл2–цикл3,

цикл3–цикл1, цикл3–цикл2.

Слід зауважити, що для забезпечення умови УКПВВ можна сформувати ПЦ, що включає у себе усі можливі комбінації циклів:

цикл1–цикл2–цикл1–цикл3–цикл2–цикл3–цикл1.

Отже, якщо цикли, навішані та точку входу, є незалежними, то процедура повинна передбачати комбінацію усіх можливих пар циклів.

Алгоритм генерації послідовності циклів

Нехай на деяку вершину i «навішано» декілька циклів Послідовність циклів повинна покривати усі можливі переходи між ними. Алгоритм, що досягає цього є простим. Наведемо результат його роботи для кількості незалежних циклів $k = 2...8$ (таблиця 1). При цьому, при формуванні ПЦ для кількості циклів k $(k > 2)$ в якості початкової використовується ПЦ, що отримана для $k-1$.

Табл. 1

k	Послідовності пар циклів	Результуюча послідовність
2	21 12	212
3	…23 31 13 32	…3132
4	…24 41 14 43 34 42	…414342
5	…25 51 15 53 35 54 45 52	…51535452
6	…26 61 16 63 36 64 46 65 56 62	…6163646562
7	…	…717374757672
8	…	…81838485868782

Для більшого контролю над результуючою множиною шляхів можна впливати на довжину отримуваної послідовності – довгі ПЦ розбивати на декілька підпослідовностей. В результаті варіюється максимальна та середня довжини шляхів у результуючий множині, що є додатковим інструментом збільшення відповідності шуканих покриттів обраній стратегії.

3.2.2 Наявність дочірніх циклів

Розглянемо випадок, в якому один з циклів має так званий дочірній цикл (цикл, який не має спільних вершин з будь-яким простим шляхом). Так у графі, що наведений на рисунку 3, дочірнім є цикл c_3.

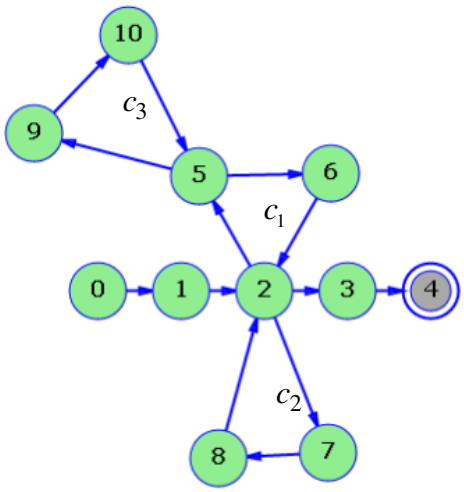

Рис. 3

Необхідно побудувати ПЦ, яка б забезпечила виконання УКПВВ для точки входу у цей цикл. Для цього батьківський цикл розглядається як простий шлях і для нього будується ПЦ за правилами, описаними в п. 3.2.1.

Для того, щоб уникнути дублювання точок входу (і необґрунтованого збільшення довжин шляхів), у ПЦ батьківський цикл включається один раз.

Так для графу з рисунку 3 маємо:

– множина простих шляхів: W_0={0-1-2-3-4};

– множина циклів: $C = \{ c_1 = 2\text{-}5\text{-}6\text{-}2,\ c_2 = 2\text{-}7\text{-}8\text{-}2,\ c_3 = 5\text{-}9\text{-}10\text{-}5\}$;

– множина послідовностей циклів для простого шляху: $\{ c_1\text{-}c_1,\ c_2\text{-}c_2,\ c_1\text{-}c_2\text{-}c_1 \}$;

– послідовність, у якій в батьківський цикл c_1 двічі вбудовано дочірній цикл c_3: 2-c_3-c_3-6-2.

3.2.3 Формування послідовностей циклів, серед яких є залежні

Залежними будемо називати цикли, які мають спільні дуги.

При аналізі результатів формування ПЦ для реальних задач ми зіткнулися з наступною проблемою. Якщо деяка вершина є точкою входу у декілька циклів (позначимо цю кількість через k), то ПЦ, у якій передбачені усі можливі комбінації циклів, містить у собі (k^2-k+1) циклів. З урахуванням того, що кожен з циклів включає в себе не менш ніж дві дуги, то кількість вершин отриманої ПЦ може бути дуже великою.

Якщо цикли, навішані на точку входу, є залежними, то після застосування вищевказаної процедури (яка передбачає комбінацію усіх можливих сполучень пар циклів) отримана ПЦ буде мати підпослідовності переходів, що багатократно повторюються. Це призводить до значного (і необгрунтованого) збільшення кількості вершин сформованої ПЦ.

Отже постала задача знаходження шляхів подолання цієї проблеми.

Розглянемо частину дерева шляхів та циклів (рисунок 4) деякого графу.

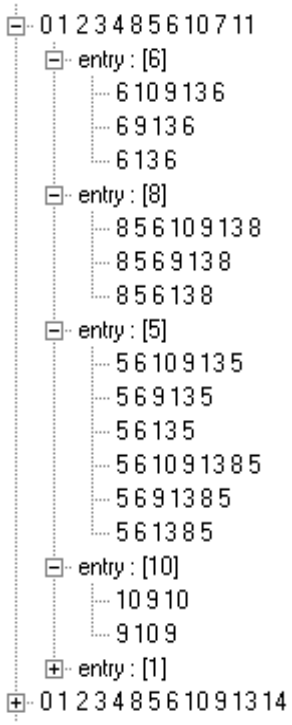

Рис. 4

Проаналізуємо особливості груп циклів, навішаних на вершини 6, 8, 5 та 10.

Особливість 1. В групі циклів 8 початкові та кінцеві дуги усіх циклів однакові. Відповідна тріада (13-8-5) покривається будь-яким продубльованим циклом. Крім того, достатньо продублювати тільки один з циклів, а з кожного іншого створити по послідовності, що включає їх по одному разу.

Особливість 2. Розглянемо групу циклів 5. Ця група розбивається на дві підгрупи і кожна з них володіє особливістю 1. Тому достатньо продублювати по одному циклу з кожної підгрупи.

Особливість 3. В групі циклів 6 усі (три) цикли мають різні початкові дуги. Отже, при побудові підпослідовності циклів будуть використані усі ці цикли.

Для визначення мінімальної множини циклів (яку достатньо буде продублювати) сформулюємо допоміжну оптимізаційну задача. Суть цієї задачі полягає в наступному. Для аналізованої групи циклів вершини відомі (визначаються) множина початкових дуг і множина кінцевих дуг. Необхідно визначити мінімальний набір циклів, які в б сукупності покривали (включали) всі задані початкові і кінцеві дуги. Тим самим, згенерована підпослідовність, безнадлишково покриє всі можливі тріади.

Математична постановка цієї задачі така. Нехай C_v – множина циклів, вершини v, $n=|C_v|$; B_v – множина початкових дуг циклів з C_v; $m=|B_v|$; F_v – множина кінцевих дуг циклів з C_v; $k=|F_v|$;

$$b_{ij}=\begin{cases}1, \text{ якщо початк. дуга } i \text{ входить до циклу } j\\ 0, \text{ якщо початк. дуга } i \text{ не входить до циклу } j\end{cases},\ i=\overline{1,m},\ j=\overline{1,n};$$

$$f_{ij}=\begin{cases}1, \text{ якщо кінц. дуга } i \text{ входить до циклу } j\\ 0, \text{ якщо кінц. дуга } i \text{ не входить до циклу } j\end{cases},\ i=\overline{1,k},\ j=\overline{1,n}.$$

Змінні моделі:

$$x_j=\begin{cases}1, \text{ якщо обираємо цикл } j,\\ 0, \text{ якщо не обираємо цикл } j,\end{cases}\quad j=\overline{1,n}$$

Цільова функція (кількість обраних циклів):

$$\sum_{j=1}^{n} x_j \to \min.$$

Обмеження:

початкова дуга i повинна входити хоча б в один обраний цикл:

$$\sum_{j=1}^{n} b_{ij} x_j \geq 1; \quad i = \overline{1,m};$$

кінцева дуга i повинна входити хоча б в один обраний цикл:

$$\sum_{j=1}^{n} f_{ij} x_j \geq 1, \quad i = \overline{1,k}.$$

3.2.4 Схема формування множини послідовностей циклів для заданої вершини v

Нехай S_v – результуюча множина послідовностей для вершини v, що є точкою входу в цикли (початково $S_v = \varnothing$).

1 Визначити множину B_v початкових и множину F_v кінцевих дуг множини C_v циклів.

2 ВИРІШИТИ допоміжну оптимізаційну задачу (на її вхід подаються C_v, B_v, F_v), отримати множину C_{gen} циклів, що приймають участь у генерації.

3 ЗГЕНЕРУВАТИ послідовність циклів $s = gen(C_{gen})$; Додати її до множини S_v

4 ЦИКЛ по циклах c множини C_{gen}

СФОРМУВАТИ підпослідовність – дубль $s = c\text{-}c$, ДОДАТИ її до множини S_v

КІНЕЦЬ ЦИКЛУ 4

5 ЦИКЛ по циклам множини $C_v \setminus C_{gen}$

СФОРМУВАТИ підпослідовність $s = c$, ДОДАТИ її до множини S_v

КІНЕЦЬ ЦИКЛУ 5

Таким чином, множина S_v включає в себе три типи послідовностей:

- послідовність $gen(C_{gen})$;
- послідовності – дублі (їх в загальному випадку декілька);
- послідовності, що включають у себе одинарні цикли, які не приймали участь в генерації послідовності $gen(C_{gen})$ і не є батьківськими (таких послідовностей в загальному випадку також може бути декілька):

У наведеному вигляді цей алгоритм застосовується до вершин (точок входу), що входять до простих шляхів. У тому разі, коли вершина v належить одному з циклів, то знайдені для неї послідовності з дочірніх циклів (що будуються за цим же алгоритмом) вбудовуються в цей цикл і додаються до результуючої множини ПЦ.

3.2.5 Комбінування множин шляхів і послідовностей циклів

На цьому кроці виконується вбудовування отриманих на кроці 2 послідовностей циклів в шляхи.

Для кожної точки входу простого шляху відома множина послідовностей циклів (див. п. 3.2.4). Можливі декілька варіантів комбінування послідовностей зі шляхом.

Варіант 1. В шлях додається тільки одна послідовність з множини S_v і отриманий шлях включається у результуючу множину шляхів. Ця процедура повторюється для усіх послідовностей, що мають відношення до цього шляху. В результаті отримуємо $\sum_{v \in V} |S_v|$ нових шляхів, де V – множина точок входу шляху, $|S_v|$ – кількість послідовностей для точки входу v.

Варіант 2. У шлях вбудовуються усі можливі для нього послідовності і після цього отриманий шлях включається до результуючої множини. В результаті отримуємо один шлях з усіма можливими послідовностями.

Варіант 3. Є проміжним між першими двома варіантами – ПЦ вбудовуються в деякі точки входу шляху.

Вибір варіанту залежить від виду прийнятої стратегії побудови планів тестів.

4 Формування множини тріад. Формулювання задачі покриття множини як задачі булевого програмування

За заданим графом формується множина тріад. Після цього кожній тріаді ставиться у відповідність підмножина шляхів, що її покривають. Далі формулюється задача знаходження покриття.

4.1 Математична постановка задачі

Задача знаходження покрить зводиться до класичної задачі мінімального покриття множини [3]. Початковими даними цієї задами є скінчена множина T тріад і множина W шляхів (кожен з яких визначається множиною покритих тріад, тобто шлях можна розглядати як підмножину T). Покриттям $\overline{W} \subseteq W$ називають множину найменшої потужності (найменшої вартості) підмножин, об'єднанням яких є T.

Ця задача відноситься до класу NP-складних і може бути сформульована як за-

дача булевого програмування. Нехай граф має m тріад (множина T) і в ньому можливі n шляхів від початкової до термінальних вершин (множина W). Для кожного шляху відомі тріади, що входять до нього, тобто для $i=\overline{1,m}$ та $j=\overline{1,n}$ задані

$$a_{ij}=\begin{cases}1, \text{ якщо тріада } i \text{ входить у шлях } j,\\ 0, \text{ якщо тріада } i \text{ не входить у шлях } j.\end{cases}$$

Вимагається знайти таку підмножину шляхів $\overline{W}\subseteq W$, щоб кожна тріада входила хоча б у один шлях і ця множина була б найкращою згідно обраної стратегії. За умови, що

$$x_j=\begin{cases}1, \text{ якщо } j-й \text{ шлях входить до множ. } \overline{W},\\ 0, \text{ якщо } j-й \text{ шлях не входить до множ. } \overline{W},\end{cases}$$

вказане обмеження формулюється наступним чином:

$$\sum_j a_{ij}x_j \ge 1,\ i=\overline{1,m}\ .$$

Щодо цільової функції. Оскільки "довга" стратегія передбачає знаходження множини планів тестування, які включають в себе найбільш довгі тести, то з цього виходить, що такий план буде включати в себе найменш можливу кількість шляхів, тобто $\sum_{j=1}^{n}x_j\to\min$. При цьому автоматично буде гарантуватися ненадлишковість знайденого покриття. (Покриття є ***ненадлишковим***, коли виключення з нього хоча б одного шляху приведе до недопустимості розв'язку). „Коротка" стратегія вимагає знаходження множини планів тестування, які включають в себе найбільш короткі тести. Тобто такий план буде включати в себе найбільш можливу кількість коротких шляхів (за відсутності надлишковості розв'язку). Для цього випадку цільова функція така: $\sum_{j=1}^{n}x_j\to\max$ (але у цьому випадку наведені обмеження не гарантують ненадлишковості розв'язку, для цього необхідно формулювати додаткові обмеження). У разі врахування вартості шляхів (c_j, $j=\overline{1,n}$) критерій ефективності можна записати так: $\sum_{j=1}^{n}c_jx_j\to\min$.

5 Знаходження розв'язку

Цей етап складається з двох кроків:

Крок 1. Оптимізація системи рівнянь (опціонально).

Крок 2. Розв'язання задачі.

В залежності від розмірності задачі та наявних обчислювальних ресурсів пропонується один з наступних методів:

– отримання повної множини покрить методом бектрекінгу;

– розв'язання задачі із застосуванням жадібного алгоритму;

– розв'язання задачі із застосуванням генетичного алгоритму.

5.1 Отримання повної множини покрить

Алгоритмом, в основу якого покладений метод бектрекінгу [4], знаходиться повна множина покриттів. Алгоритм полягає у послідовному виклику для кожного (починаючи з першого) елементу рівнянь рекурсивної процедури, яка додає до цього елементу елемент з наступного рівняння і так далі, поки не буде досягнуто останнє рівняння, що свідчитиме про отримання деякого допустимого розв'язку. При цьому забезпечується ненадлишковість розв'язків, що отримуються.

Знаходження повної множини покрить дозволяє в подальшому при виборі найкращого покриття застосовувати декілька критеріїв.

Для задачі побудови множини покрить типу Path Coverage час розв'язання зростає експоненційно з ростом кількості шляхів.

5.2 Шляхи прискорення обчислень

Природа NP-складних задач вимагає приділення особливої уваги проблемі зростання часу пошуку розв'язку у випадку росту розмірності задачі. Прискорення процесу обчислень було досягнуто такими шляхами:

– застосуванням критеріїв відсікання "безперспективних" підмножин, що в результаті дає деяку групу покриттів, що задовольняють певному критерію (згідно обраної стратегії);

– зменшенням вхідної множини шляхів: якщо, є два шляхи, таких, що $T_1\subseteq T_2$, де T_1, T_2 –множини тріад, які покриваються цими шляхами, то виключається шлях, якому відповідає множина T_1;

– оптимізацією системи рівнянь: на цьому етапі для збільшення швидкості подальших обчислень виконується оптимізація системи рівнянь (видалення рівнянь, які містять усі шляхи; видалення рівнянь, які містять тільки один шлях з подальшим включенням цього шляху до кожного знайденого

розв'язку; видалення рівнянь у випадку, якщо воно повністю включає в себе інше рівняння). Цей процес дозволяє значно зменшити розмірність задачі (у деяких випадках виграш становив десятки разів).

Поряд з алгоритмічною оптимізацією суттєвого збільшення швидкості обчислень можна досягти використанням технології відомої як GPGPU (general-purpose GPU) – перенесенням обчислень з центрального процесору на графічний чип. Найпоширенішими реалізаціями технології GPGPU є NVidia CUDA та AMD FireStream. Графічний процесор має більші обчислювальні потужності завдяки великій кількості процесорів (512 і більше) та швидкої пам'яті. Після перенесення обчислень на графічну карту вдалося збільшити швидкість роботи алгоритму в декілька разів. Недоліком цього підходу є висока залежність програмного забезпечення від апаратних засобів.

5.3 Розв'язання задачі із застосуванням жадібного алгоритму

Досить часто для реальних практичних NP-складних задач знайти точний розв'язок за прийнятний час не можливо. В такому випадку можна взяти ліпший розв'язок, який вдалося знайти за певний встановлений ліміт часу роботи точного алгоритму, або використовувати наближені або евристичні алгоритми. Жадібний алгоритм є одним з найпоширеніших зразків евристичних алгоритмів. Перевага цього алгоритму полягає в тому, що час знаходження рішення поліноміально зростає з ростом розмірності. Для задачі знаходження покриття графу шляхами розроблено та реалізовано наступний жадібний алгоритм. Спочатку маємо початкове покриття, яке не включає в себе жодного шляху. Доки це покриття не задовольняє умові УКПВВ додаємо до нього новий шлях, який покриває найбільш можливу кількість ще не покритих тріад. Цілком можливо, що отриманий розв'язок буде надлишковим. Надлишковості можна позбутися використовуючи алгоритм оптимізації, описаний у п. 5.1. Вихідна система рівнянь будується на основі шляхів, що увійшли у знайдений розв'язок. Після оптимізації можливе отримання декількох розв'язків, що дуже схожі між собою.

З урахуванням сутності дій алгоритму побудовані розв'язки будуть в більшості випадків задовольняти вимогам довгої стратегії.

5.4 Розв'язання задачі із застосуванням генетичного алгоритму

Структура множини допустимих розв'язків задачі знаходження покриття графу шляхами (кожний розв'язок є булевим вектором розмірності n) дозволяє розробити для цієї задачі генетичний алгоритм (ГА), що відноситься до так званих метаевристичних алгоритмів [5].

ГА включає в себе наступні основні оператори:

- генерування початкової популяції;
- схрещування обраних за певним критерієм особин;
- мутація окремих особин;
- локальне покращення особини.

Те, яким чином реалізовані ці оператори, впливає на якість отриманого результату. Розглянемо деякі особливості розроблених операторів. Оператори генерування та мутації базуються на випадковому додаванні\вилученні елементів з розв'язку із одночасним збереженням допустимості. Схрещування є "змішуванням" батьківських розв'язків (особин) у певних пропорціях – якась частина нової особини унаслідується від "матері", якась - від "батька". В операторі локального покращення виключається "найслабкіша ланка" – елемент розв'язку, що у найбільшій мірі не задовольняє обраному критерію оптимальності. Замість виключеного елементу знаходиться новий (нові), який максимально покращить значення критерію оптимальності для розв'язку не порушуючи допустимості. У всіх операторів є одна спільна риса – після створення нового розв'язку (особини) цілком можливим є те, що він є надлишковим. Щоб гарантувати ненадлишковість застосовується алгоритм, що описаний у п 5.1.

Звичайно, час роботи ГА більший порівняно із жадібним, але результати, отримані за його допомогою, є значно кращими. Експерименти показали, що у більшості випадків для задач, для яких можна за прийнятний час отримати оптимальний розв'язок, цей же розв'язок був отриманий і ГА. Але самою сильною стороною генетичного алгоритму є можливість пошуку рішень за практично довільними критеріями – значно складнішими, за критерії довгої та короткої стратегій. При чому застосування нелінійних та комплексних критеріїв практично не позначається на швидкодії алгоритму.

5.5 Програмний продукт

За матеріалами роботи створений програмний продукт “Тріада”.

На рисунку 5 наведено головне вікно розробленого програмного продукту.

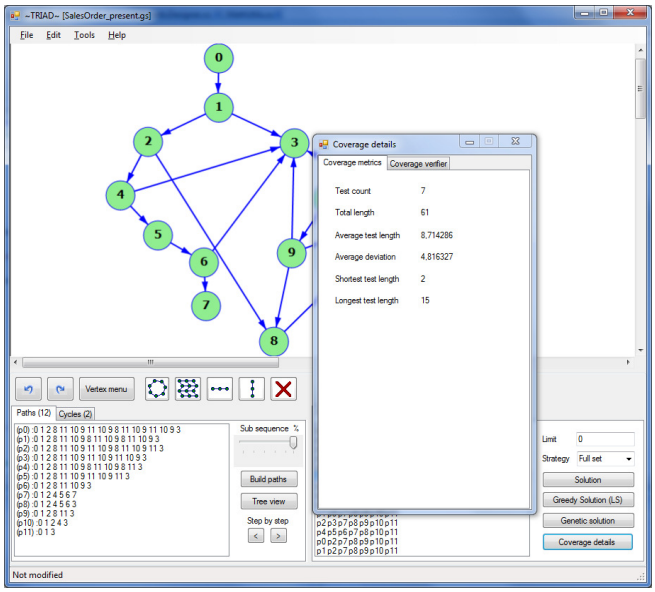

Рис. 5

В ньому реалізовано:

– WYSIWYG- редактор для створення графу з можливістю додавання, видалення та редагування дуг та вершин графу;

пошук усіх шляхів з початкової вершини у термінальні;

– пошук усіх циклів графу:

– побудова множини покрить шляхами з урахуванням обраної стратегії;

– пошук покриття жадібним алгоритмом;

– пошук покриття за допомогою ГА;

– додаткові інструменти: метрики отриманих покриттів, верифікатор покриття, дерево шляхів та циклів та ін.

Використовуючи систему плагінів, можна транслювати отриманий план у іншу предметну область, наприклад створити динамічну бібліотеку, яка буде викликати методи програми у порядку, що вказаний у плані тестування.

Висновки

В роботі досліджені особливості задачі побудови покриття типу Path Coverage. Розроблені точний та евристичні алгоритми розв’язання задачі. Запропоновані шляхи оптимізації процесу та зменшення часу обчислень.

Дослідження проблеми побудови покриття типу Path Coverage було тісно пов’язане з розробкою програмного продукту “Тріада”. Деякі особливості задачі вдалося дослідити саме завдяки інструментарію програмного продукту “Тріада”.

Розроблені та реалізовані такі методи розв’язання задачі, що дозволяють менеджеру (керівнику групи тестувальників) застосовувати їх в реальних умовах з можливістю впливу на параметри результуючого плану тестування в залежності від обраної стратегії.

Перелік посилань

1. Kolawa A., Huizinga D. (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. 254 p.
2. Ammann P., Offutt J.. Introduction to software testing. Cambridge University press - New York. 2008, – 322 p
3. Кристофидес Н. Теория графов. Алгоритмический подход. М.: Мир, 1978, – 432 с.
4. Кнут Д. Искусство программирования. В 3-х т. 3-е изд. – М.: «Вильямс», 2006. Т. 1– 720 с.
5. Сергиенко И.В., Шило В.П. Задачи дискретной оптимизации: проблемы, методы решения, исследования. – К. Наук. думка, 2003. – 264 с.

Поступила в редакцию 3.12.2009