ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2019 Управление, вычислительная техника и информатика № 47

УДК 519.7
DOI: 10.17223/19988605/47/12

**С.А. Поттосина**

# ПРИМЕНЕНИЕ ЗАДАЧИ О КРАТЧАЙШЕМ ПОКРЫТИИ ПРИ ТЕСТИРОВАНИИ ПРОГРАММНОГО ПРОДУКТА

Рассматриваются две задачи тестирования программного продукта. Одна из них связана с нахождением минимального набора тестов при регрессионном тестировании функционала программы. Другая задача относится к нахождению минимального диагностического теста при классификации дефектов в программном продукте. Показано, как обе эти задачи сводятся к нахождению кратчайшего столбцового покрытия некоторой булевой матрицы.

**Ключевые слова:** регрессионное тестирование; задача о покрытии; дефекты программного продукта; диагностический тест; матрица различий.

Многие комбинаторные оптимизационные задачи сводятся к задаче о кратчайшем покрытии, которая ставится следующим образом. Пусть даны некоторое множество $A = \{a_1, a_2, \ldots, a_n\}$ и совокупность его подмножеств $B_1, B_2, \ldots, B_m$, т.е. $B_i \subseteq A$, $i = 1, 2, \ldots, m$, причем $B_1 \cup B_2 \cup \ldots \cup B_m = A$. Требуется среди данных подмножеств выделить такую совокупность $B_{i_1}, B_{i_2}, \ldots, B_{i_k}$ с минимальным $k$, чтобы каждый элемент из $A$ попал хотя бы в одно из $B_{i_j}$ ($j = 1, 2, \ldots, k$), т.е. $B_{i_1} \cup B_{i_2} \cup \ldots \cup B_{i_k} = A$.

Удобно рассматривать матричную формулировку данной задачи, при которой совокупность $B_1, B_2, \ldots, B_m$ задается в виде булевой матрицы, строки которой соответствуют подмножествам из данной совокупности, а столбцы – элементам множества $A$. Элемент $i$-й строки и $j$-го столбца имеет значение 1, если и только если $a_j \in B_i$. В этом случае говорят, что $i$-я строка покрывает $j$-й столбец. Требуется найти такое множество строк данной матрицы, чтобы каждый ее столбец имел единицу хотя бы в одной строке из этого множества, и при этом мощность выбранного множества должна быть минимальной. В такой интерпретации имеем дело с задачей поиска кратчайшего строчного покрытия булевой матрицы.

Задача о кратчайшем покрытии булевой матрицы нашла применение в теории графов, в теории логического проектирования дискретных устройств, при решении задач технической диагностики, в некоторых алгоритмах минимизации логических функций [1]. Встал вопрос, может ли успешно применяться данная задача при тестировании программного обеспечения.

Проблема качества программных продуктов (ПП) становится сегодня все более острой, особенно по мере расширения использования информационных технологий и роста сложности ПП, при работе большой команды разработчиков. Высокое качество продуктов дает разработчикам не только конкурентные преимущества и кредит доверия клиентов, но и облегчает сопровождение и развитие ПП. Одним из ключевых элементов обеспечения качества ПП является тестирование [2, 3]. Для тестирования ПП необходима разработка тестов. Разработчики ПП проводят тестирование своих продуктов в несколько этапов, которые отличаются видами выполняемых работ и привлекаемыми ресурсами. Фактически тестирование начинается еще в процессе кодирования очередной версии программы.

Затраты на тестирование вносят существенный вклад в общую стоимость сопровождения программы. Выборочный подход к регрессионному тестированию измененной программы требует исключения некоторых тестов из существующего набора, что приводит к уменьшению затрат. При проведении тестирования важно выбрать подмножество тестов минимальной мощности, обеспечивающее достижение целей регрессионного тестирования.

В научной литературе подчеркивается важность комплексного подхода [2, 3]. Под комплексным подходом к управлению качеством ПП имеется в виду необходимость поиска и устранения дефектов на каждом этапе жизненного цикла ПП [4]. Основными методами поиска дефектов являются отладка и тестирование. Методы поиска дефектов имеют различную эффективность и стоимость. Классификация дефектов по типам позволит отправлять их на устранение определенным разработчикам, что приведет к уменьшению затрат.

Для проведения регрессионного тестирования реальных проектов и устранения в них дефектов можно предложить две задачи, решение которых сводится к нахождению кратчайшего покрытия некоторой булевой матрицы. Одна из них связана с нахождением минимального набора тестов при регрессионном тестировании функционала программы. Вторая задача позволяет получить минимальный диагностический тест при классификации дефектов в программном продукте. Обе задачи решаются с помощью минимаксного алгоритма, одним из приближенных методов решения задачи о кратчайшем покрытии.

## 1. Задача о нахождении минимального набора тестов при регрессионном тестировании функционала программы

Регрессионное тестирование – это выборочное тестирование, позволяющее убедиться, что изменения не вызвали нежелательных побочных эффектов или что измененная система по-прежнему соответствует требованиям. После каждой модификации программы необходимо удостовериться, что на функциональность программы не оказал влияния модифицированный код. Регрессионное тестирование – дорогостоящий род деятельности: процесс регрессионного тестирования может включать исполнение достаточно большого количества тестов на скорректированной программе, даже если изменений очень мало. Несмотря на то, что усилия, требуемые для внесения небольших изменений, как правило, минимальны, они могут требовать достаточно больших усилий для проверки качества измененной программы. Тем не менее проведение регрессионного тестирования необходимо. Надежная и эффективная разработка и дальнейшее сопровождение программного обеспечения невозможны без регрессионного тестирования. Выполнить полное регрессивное тестирование вряд ли возможно. По мере роста и расширения становится все сложнее тестировать отдельные части системы программного обеспечения. Эта проблема усложняется из-за частоты создания сборок программ. Необходимо тестировать предыдущую функциональность, чтобы обеспечить возможность тестирования новых исправлений и новых функций.

В настоящее время выделяют следующие методы отбора тестов для регрессионного тестирования: случайные методы; безопасные методы; методы минимизации; методы, основанные на покрытии кода. Каждый из представленных методов имеет свои достоинства и недостатки [4].

Применение задачи о кратчайшем покрытии булевой матрицы для отбора тестов при проведения регрессионного тестирования безопасным выборочным методом позволяет сократить временные затраты на проведение регрессионного тестирования без потери качества программного продукта.

В качестве булевой матрицы выступает матрица $R$, строки которой соответствуют тестируемым функционалам программы, а столбцы соответствуют тестам, предназначенным для этого. Элемент матрицы $r_{ij}$ равен 1, если для тестирования функционала $f_i$ пригоден тест $t_j$, иначе элемент $r_{ij.}$ матрицы $R$ равен 0.

В качестве примера рассмотрен реальный проект «Сайт-платформа сравнения цен на услуги по страхованию автомобиля». Для этого выделены 17 ключевых элементов системы (тестируемый функционал), безотказная работа которых была критична для данного приложения. К ним относятся:
$a$ – вход на сайт; $b$ – выход; $c$ – регистрация; $d$ – идентифицированный пользователь; $e$ – пользователь-студент; $f$ – работающий пользователь; $g$ – безработный пользователь; $h$ – пользователь с дополнительным водителем; $i$ – анонимный пользователь; $k$ – пользователь с истекшим временем покупки; $m$ – проверка уведомлений об ошибке; $n$ – пользователь, вернувшийся на сайт после 11 месяцев;

*o* – активный пользователь; *p* – пользователь с дополнительной работой; *r* – индивидуальный предприниматель; *s* – страница с результатами покупки; *t* – рассылка информации на почту.

Далее был создан тестовый набор регрессионного тестирования, который содержал проверки заявленной функциональности (14 тестов): 1 – «приходящий через 11 месяцев»; 2 – анонимный посетитель; 3 – дополнительный водитель; 4 – базовый тест; 5 – граничный тест; 6 – редактирование; 7 – пользовательский отчет; 8 – встроенный вход; 9 – навигация по сайту; 10 – сброс пароля; 11 – страница с результатами; 12 – повторный покупатель; 13 – проверка ошибок, 14 – рассылка на почту.

Затем была построена диагностическая матрица (таблица). Строки матрицы соответствуют функциональным модулям программы, а столбцы – тестам. Элемент матрицы C на пересечении *i*-й строки и *j*-го столбца имеет значение 1, если для тестирования *i*-го модуля необходим *j*-й тест. В противном случае элемент матрицы равен 0. Далее к данной матрице был применен алгоритм поиска кратчайшего столбцового покрытия. Результаты оказались достаточно успешными: удалось сократить тестовый набор на 14,3% без потери процента покрытия функциональных возможностей программы. Два теста, а именно тесты с номерами 3 и 4, оказались излишними.

**Диагностическая матрица**

| Функционал | Тесты | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| *a* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| *b* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| *c* | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| *d* | | | | | | | 1 | 1 | | 1 | 1 | 1 | 1 | 1 |
| *e* | | | | 1 | | 1 | | | 1 | | 1 | | | |
| *f* | | | 1 | | | | 1 | | 1 | | 1 | | | |
| *g* | | | | | 1 | 1 | | | | | | | | 1 |
| *h* | | | 1 | | | 1 | | 1 | | 1 | | 1 | | |
| *i* | | 1 | | | | | | | | | | | | |
| *k* | | | | | | | 1 | | | | | | | |
| *m* | | | | | | | | | | | | | | 1 |
| *n* | 1 | | | | | | | | | | | | | |
| *o* | | | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| *p* | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| *r* | | 1 | | | | | 1 | | | | | | | |
| *s* | | | | | | 1 | | | 1 | | 1 | 1 | | |
| *t* | | | | | | | | | | | | | | 1 |

Аналогично данный метод был применен для другого проекта – «Сайта страхования недвижимости». По реализуемой задаче оба приложения похожи и имеют небольшое количество отличительных черт. Поэтому и тестовый набор во многом одинаков. Во втором случае удалось добиться уменьшения первоначального тестового набора на 21,4%.

## 2. Задача о нахождении минимального диагностического теста при классификации дефектов в программном продукте

Автоматизированное тестирование подразумевает под собой разработку и использование специального программного обеспечения для запуска и контроля выполнения тестовых сценариев и сравнения реальных и запланированных результатов согласно спецификации. При организации автоматизации тестирования больших проектов, когда необходимо разделение специалистов по тестированию на определенные группы, появляется задача о классификации дефектов.

Опишем распределение ролей в достаточно крупной команде специалистов, занимающихся автоматизацией тестирования. Сама по себе автоматизация крайне редко применяется на краткосрочных или часто изменяющихся проектах, что связано с постоянно растущими затратами на поддержку и написание автоматических тестов. Проект с применением автоматизации имеет большое количе-

ство вариантов использования. Это можем быть работа, связанная с тестированием интерфейсов (внешнего отображения), сервисов, производительности и нагрузки, стрессовым тестированием, тестированием взаимодействия с внешними приложениями по отношению к данному и т.д. Само собой разумеется, что и за дефекты, возникающие в процессе тестирования и работы приложения, будут отвечать специалисты, занимающиеся поддержкой данной деятельности на проекте и являющиеся в ней наиболее квалифицированными. Работа многих систем отслеживания и устранения дефектов организована при помощи данных классификаций дефекта [5].

Используется следующая классификация дефектов с точки зрения степени влияния (Severity) на работоспособность программы:

– блокирующий (Blocker). Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате чего дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможной. Решение проблемы необходимо для дальнейшего функционирования системы;

– критический (Critical). Критическая ошибка, неправильно работающая ключевая бизнес-логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой;

– значительный (Major). Значительная ошибка, часть основной бизнес-логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки;

– незначительный (Minor). Незначительная ошибка, не нарушающая бизнес-логику тестируемой части приложения, очевидная проблема пользовательского интерфейса;

– тривиальный (Trivial). Тривиальная ошибка, не касающаяся бизнес-логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

Градация дефектов с точки зрения приоритетности исправления (Priority):

– высокий (High): ошибка должна быть исправлена как можно быстрее, так как ее наличие является критической для проекта;

– средний (Medium): ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения;

– низкий (Low): ошибка должна быть исправлена, ее наличие не является критичной и не требует срочного решения.

В зависимости от того, какой методологии придерживаются на проекте, распределение зарегистрированных дефектов между специалистами может быть различным. Однако на данный момент одним из наиболее распространенных и приоритетных способов является наличие одной общей «доски с дефектами», куда помещаются все найденные дефекты. Далее специалист по автоматизации открывает «доску» и смотрит на список дефектов, расположенных там. Разумеется, сложно с первого взгляда определить, к какому типу относится дефект и какому специалисту лучше поручить работу над ним. Поэтому на процесс анализа, классификации и дальнейшего устранения дефектов тратится достаточно большой объем времени. В подобных условиях применение автоматизированной классификации дефектов является не только рациональным, но желательным для организации эффективной деятельности команды. Это позволит уменьшить время простоев, которые неминуемо возникают в случае ошибочной передачи дефекта. Задача о нахождении минимального диагностического теста, предложенная в [1], может оказаться весьма полезной.

Рассмотрим множества $D = \{D_1, D_2, \ldots, D_n\}$ и $B = \{b_1, b_2, \ldots, b_n\}$. Множество $D$ является множеством возможных дефектов, множество $B$ – множеством внешних признаков (симптомов), которые возникают вследствие обнаружения дефекта. Симптом – свойство дефекта, позволяющее классифицировать дефекты по их типичному проявлению. Строится булева диагностическая матрица $\boldsymbol{C}$, которая показывает, какими признаками характеризуется тот или иной дефект. Элемент матрицы $c_{ij}$ равен 1,

если дефект $D_i$ влечет симптом $b_j$, иначе элемент $c_{ij}$ матрицы $\boldsymbol{C}$ равен 0. Для матрицы $\boldsymbol{C}$ строится матрица различий $\boldsymbol{R}$, строки которой соответствуют парам строк диагностической матрицы и показывают, какими компонентами отличаются строки в этих парах; при этом будем использовать покомпонентную операцию сложения по модулю два, выполняя ее над всеми парами строк диагностической матрицы.

Для полученной матрицы различий можно найти кратчайшее столбцовое покрытие. Множество признаков, соответствующих столбцам найденного покрытия, будет искомым решением задачи. При этом каждый дефект можно однозначно определить соответствующей строкой подматрицы $\boldsymbol{C}'$ матрицы $\boldsymbol{C}$. Это позволяет наиболее рациональным образом классифицировать дефекты на момент их возникновения по данным признакам.

В качестве примера рассмотрен реальный проект – веб-приложение для МТБанка «Halva.by», в котором были выделены такие симптомы дефектов, как:

1. Косметический симптом – визуально заметный недостаток интерфейса, не влияющий на функциональность приложения.

2. Некорректная операция – некорректное выполнение некоей операции.

3. Нереализованная функциональность – некая функция приложения не выполняется или не может быть вызвана.

4. Низкая производительность – выполнение неких операций занимает недопустимо большое время.

5. Крах системы – приложение прекращает работу или теряет способность выполнять свои ключевые функции.

6. Неожиданное поведение – в процессе выполнения некоторой типичной операции приложение ведет себя необычным образом.

7. Недружественное поведение – поведение приложения создает пользователю неудобства в работе.

8. Расхождение с требованиями – приложение ведет себя не так, как указано в требованиях.

9. Предложение по улучшению – приложение ведет себя согласно требованиям, но у специалиста по тестированию есть обоснованное мнение о том, как ту или иную функциональность можно улучшить.

При проведении тестирования в программном продукте найдены следующие дефекты:

$D_1$: отсутствует логотип компании при авторизации в системе.

$D_2$: отображается страница «Акции» при клике на раздел «Аксессуары».

$D_3$: отсутствует раздел «Избранные магазины».

$D_4$: не загружается страница «Техника» со списком соответствующих магазинов при одновременном пользовании сайтом количеством человек, равным 10 000.

$D_5$: крах плагина Adobe Flash при попытке просмотра видеозаписи.

$D_6$: номер телефона технической поддержки не удается набрать в мобильном браузере.

Для нахождения минимального диагностического теста построим диагностическую матрицу $\boldsymbol{C}$:

$$\boldsymbol{C} = \begin{array}{c} \\ D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \end{array}.$$

Ниже представлена матрица различий $\boldsymbol{R}$, строки которой соответствуют парам строк диагностической матрицы и показывают, какими компонентами отличаются строки в этих парах; при этом будем использовать покомпонентную операцию сложения по модулю два, выполняя ее над всеми парами строк диагностической матрицы. Столбцы матрицы соответствуют симптомам.

$$\boldsymbol{R} = \begin{array}{c} \\ D_1D_2 \\ D_1D_3 \\ D_1D_4 \\ D_1D_5 \\ D_1D_6 \\ D_2D_3 \\ D_2D_4 \\ D_2D_5 \\ D_2D_6 \\ D_3D_4 \\ D_3D_5 \\ D_3D_6 \\ D_4D_5 \\ D_4D_6 \\ D_5D_6 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left[\begin{array}{ccccccccc} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \end{array}.$$

Используя минимаксный алгоритм поиска кратчайшего столбцового покрытия матрицы $\boldsymbol{R}$, получаем одно их возможных кратчайших покрытий $П$ = (1, 2, 3, 4, 6). Действительно, множество симптомов $П$ однозначно диагностирует каждый дефект данного программного продукта, поскольку все строки подматрицы $\boldsymbol{C}'$различны:

$$\boldsymbol{C}' = \begin{array}{c} \\ D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 6 \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}.$$

Так, дефект $D_1$ характеризуется только симптомом 1, дефект $D_2$ характеризуется наличием симптомов 2, 6 и отсутствием симптомов 1, 3, 4 и т.д.

Что касается практического применения данного подхода, то каждому из признаков (симптомов) дефекта может соответствовать определенная метка, которая будет однозначно идентифицировать дефект на «доске». проекта. Последнее ускорит поиск и распределение дефектов между специалистами, а в дальнейшем эффективно скажется на работе всего проекта. Также это имеет и обратную связь: в случае, если после исправления дефекта он снова был обнаружен, поиск причин такого поведения системы будет организован гораздо легче и быстрее.

## Заключение

Для автоматизации процесса мониторинга и контроля дефектов, регрессионного тестирования при разработке программных продуктов создана система управления задачами и отслеживанием ошибок, в которой представлены программные модули для реализации решения рассмотренных выше задач. Система может применяться, например, в начинающих организациях, которые не обладают средствами для обеспечения себя дорогостоящими продуктами, такими как JIRA, Trello, Yodiz. Бюджетные программные средства, использующиеся на данный момент организациями, имеют ограниченные возможности и не обладают достаточным набором функций.

Система выполняет следующие функции и задачи:

– добавление, изменение, просмотр и удаление проектов, задач, дефектов, а также сотрудников команды, работающей над созданием программного продукта;

– осуществление поиска задачи, дефектов, сотрудника по различным параметрам;

– отображение личного кабинета сотрудника компании;

– разграничение функционала в зависимости от роли: проектный менеджер (администратор) и специалист по тестированию или обнаружению дефектов (пользователь);

– определение минимального диагностического теста и минимального тестового набора при регрессионном тестировании.

Система реализована в виде распределенного приложения на языке Java с использованием технологий и фреймворков разработки веб-приложений. Веб-приложение представлено в виде клиент-серверного приложения, в котором клиентом выступает браузер, а сервером – веб-сервер.

## ЛИТЕРАТУРА

1. Закревский А.Д., Поттосин Ю.В., Черемисинова Л.Д. Логические основы проектирования дискретных устройств. М. : Физматлит, 2007. 592 с.
2. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем : пер. с англ. СПб. : Питер, 2004. 318 с.
3. Wang Sh., Wu Yu., Lu M., Li H. Software reliability accelerated testing method based on test // Proceedings «Annual Reliability and Maintainability Symposium». Lake Buena Vista, FL, USA. 24–27 Jan. 2011.
4. Котляров В.П. Основы тестирования программного обеспечения : учеб. пособие. М. : БИНОМ. Лаборатория знаний, 2006. 285 с.
5. Куликов С.С. Тестирование программного обеспечения : базовый курс. Минск : Четыре четверти, 2017. 312 с.

Поступила в редакцию 28 августа 2018 г.

Pottosina S.A. (2019) APPLICATION OF THE SHORTEST COVER PROBLEM IN TESTING A SOFTWARE PRODUCT. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie vychislitelnaja tehnika i informatika* [Tomsk State University Journal of Control and Computer Science]. 47. pp. 102–109

DOI: 10.17223/19988605/47/12

The combinatorial problem of shortest cover can be applied successfully in testing software. Two tasks whose solution is reduced to finding a shortest column cover of a Boolean matrix are considered in the paper. Both the tasks can be solved with the minimax algorithm.

The first task is connected with minimization of a test set in regression testing of a program functional. As a Boolean matrix, there is matrix $\boldsymbol{R}$ whose rows correspond to the program functionals being tested and columns to the tests intended for it. An element $r_{ij}$ of the matrix $\boldsymbol{R}$ is equal to 1 if the test $t_j$ is suitable for testing the functional $f_i$, otherwise it is equal to 0.

As an example, the real project of "Site-platform of comparing costs for services on automobile insurance" was considered. For this purpose, the key elements of the system (the tested functional) are extracted whose faultless work was critical for this application. Those are the following: enter to a site; exit; registration; recognized user; user-student; user working; user unemployed; user with additional driver; anonymous user; user with elapsed time of a purchase; check of notification on a mistake; user with additional work; individual business owner; page with results of purchase; distribution of information to post. Then, a test set of regression testing was created that contained checks of declared functionality (11 tests). As a result of the search of a shortest cover of matrix $\boldsymbol{R}$, it was found that the initial test set could be 14.3% decreased (2 tests) without lost of cover percentage of the functional. Similarly, this method was applied for another project; it was "Site of property insurance". According to the solving problem, both the applications resemble and have a little number of distinctive features. Therefore the test set is the same in much. At the second case, the decrease of the initial test set was by 21.4%.

The second task concerns to the search for the minimal diagnostic test that can qualify defects in a software product. It is useful for automation of testing large products when testers are divided into certain groups. Let us consider two sets $D = \{D_1, D_2, \ldots, D_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. The set $D$ is the set of possible defects and $B$ is the set of outward signs (symptoms) that appear as a result of disclosure of a defect. A symptom is a characteristic of a defect that can help to qualify defects according to their typical exposure. The diagnostic Boolean matrix $\boldsymbol{C}$ is constructed that shows what characters one or another defect is characterized. Then, the distinction matrix $\boldsymbol{R}$ is constructed whose rows correspond to the row pairs of the diagnostic matrix and show by what components those pairs differ. At that, the component-wise operation of modulo 2 addition is used executing it over all the row pairs of the diagnostic matrix.

Let {1, 2, 3, 4, 5, 6, 7, 8, 9} be the set of symptoms. The defect $D_1$ is characterized by the symptoms 1 and 8 and negates the rest. The defect $D_2$ is characterized by the symptoms 2, 6, 7, 8 and negates the rest. Then the first row of the distinction matrix is $D_1D_2$ = (1 1 0 0 0 1 1 0 0).

For the obtained distinction matrix, a shortest column cover is found. The set of characters corresponding to the columns in the found cover is the desired solution of the problem of searching for a minimum unconditional diagnostic test. At that, any defect is defined unambiguously by the corresponding row of the obtained submatrix. It helps to classify the defects in the most rational way at the moment of their appearance according to the given characters. This problem can be useful when the automation of testing large projects is organized and it is necessary to divide the test experts into certain groups. Then the problem of defect classification appears.

Keywords: regression testing; short cover problem; defects of a software product; diagnostic test; distinction matrix.

*POTTOSINA Svetlana Anatolievna* (Candidate of Physical and Mathematical Sciences, Associate Professor, Belorussian State University of Informatics and Radio-Electronics, Minsk, Belarus).

REFERENCES

1. Zakrevskiy, A.D., Pottosin, Yu.V. & Cheremisinova, L.D. (2007) *Logicheskie osnovy proektirovaniya diskretnykh ustroystv* [Logical Fundamentals of Design of Digital Devices]. Mosocw: Fizmatlit.
2. Beizer, B. (2004) *Testirovanie chernogo yashchika. Tekhnologii funktsional'nogo testirovaniya programmnogo obespecheniya i sistem* [Black-box testing. Techniques for functional testing of software and systems]. Translated from English. ST. Petersburg: Piter.
3. Wang, Sh., Wu, Yu., Lu, M. & Li, H. (2011) Software reliability accelerated testing method based on test. *Annual Reliability and Maintainability Symposium*. Proc of the Conference. Lake Buena Vista, FL, USA. January 24–27, 2011.
4. Kotlyarov, V.P. (2006) *Osnovy testirovaniya programmnogo obespecheniya* [Fundamentals of software testing]. Moscow: BINOM.
5. Kulikov, S.S. (2017) *Testirovanie programmnogo obespecheniya* [Software testing]. Minsk: Chetyre chetverti.