An Extended Algorithm using a Divide-and-Rule Approach for Boundary Value Analysis and Equivalence Class Partitioning

Danh Nguyen-Cong

Faculty of Software Engineering, College of Information and Communication Technology, Can Tho University, Can Tho, Vietnam

Correspondence: Danh Nguyen-Cong, ncdanh@cit.ctu.edu.vn

Received: 31/07/2024, revised: 23/09/2024, accepted: 11/10/2024

Digital Object Identifier: 10.32913/mic-ict-research-vn.v2024.n3.1311

Abstract: An extended algorithm using a divide-and-rule approach is proposed in this paper. Instead of generating test cases by only using Boundary Value Analysis and an experience-based technique for combination of input variables with functional dependency like the original algorithm, the extended algorithm applies both Boundary Value Analysis and Equivalence Class Partitioning techniques as well as various techniques for combination of input values. In addition, a new tool provides advantages when compared with the original one. It provides users with more choices in generating test cases.

Keywords: *Software testing, boundary value analysis, equivalence class partitioning, functional dependency*

Tiêu đề: **Một thuật toán mở rộng sử dụng phương pháp chia để trị cho kỹ thuật phân tích giá trị biên và phân vùng tương đương**

Tóm tắt: Bài báo này đề xuất một thuật toán mở rộng sử dụng phương pháp chia để trị. Thay vì tạo các trường hợp kiểm thử chỉ bằng cách sử dụng kỹ thuật Phân tích giá trị biên và kỹ thuật dựa trên kinh nghiệm để kết hợp các biến đầu vào có phụ thuộc hàm như thuật toán gốc, thuật toán mở rộng áp dụng cả kỹ thuật Phân tích giá trị biên và Phân vùng tương đương cũng như nhiều kỹ thuật kết hợp khác để kết hợp các giá trị đầu vào. Ngoài ra, một công cụ mới cung cấp nhiều ưu điểm hơn khi so sánh với công cụ gốc. Nó cung cấp cho người dùng nhiều lựa chọn hơn trong việc tạo các trường hợp thử nghiệm.

Từ khóa: *Software testing, boundary value analysis, equivalence class partitioning, functional dependency*

I. INTRODUCTION

Generating effective test cases to thoroughly evaluate the functionality of developing software is a challenging aspect of the software testing process. Numerous functional testing methods are employed for this purpose [1–4, 6]. Among the various techniques used in black-box testing, Equivalence Class Partitioning (ECP) and Boundary Value Analysis (BVA) are the most prevalent techniques for designing test cases [5, 7, 8]. These techniques are also widely adopted by various software testing tools to automate the creation and evaluation of test cases.

ECP is a software testing technique that divides input data into equivalent partitions to derive test cases. Each partition is tested at least once to uncover error classes, reducing the overall number of test cases and thus shortening the testing time [9].

BVA is a software testing technique that focuses on input values at edges of their valid ranges. The assumption is that errors are more likely to occur near these extreme values [4, 5]. This method involves testing input values at the minimum, slightly above the minimum, a typical value, slightly below the maximum, and the maximum. These values can be denoted as min, min+, nom, max-, and max, respectively.

However, both ECP and BVA are effective for programs with independent variables but fall when variables are interdependent. For instance, in the NextDate function, the variables Month and Day range from 1-12 and 1-31, respectively. Consequently, using ECP and BVA does not generate test cases for dates like February 28th and 29th.

Vij K. et al. [10] developed a testing tool based on an algorithm that uses a divide-and-rule approach to eliminate dependencies among variables, creating independent variable sets. The algorithm showed the advantages over the traditional boundary value analysis technique.

In this paper, a new testing tool is developed based on a new algorithm that extends the algorithm proposed in [10]. Our extended algorithm considers the integrated use of BVA, ECP and different techniques for combination of input values. Section II presents the divide-and-rule approach including the extended algorithm and an example. Section III introduces the implementation of a new testing tool. The demonstration of the new testing tool is given in Section IV. Finally, Section V summarizes the paper with a conclusion and our future plans.

II. EXTENDED DIVIDE-AND-RULE APPROACH

The divide-and-rule approach was presented in detail in [10]. The algorithm breaks the dependencies among variables and produces multiple independent sets of variables. It ensures that variables within these new sets do not affect each other's boundary values. This allows for traditional boundary value analysis to be performed on each independent set.

In a set of interdependent variables, boundaries of a variable are influenced by the values of other variables, a relationship known as boundary dependency. For the divide-and-rule approach, these variables are categorized into the following three types: 1) *dependent variables*: variables whose boundary values are affected by the values of other variables, 2) *boundary-determining variables*: variables that influence the boundary values of other variables, and 3) *independent variables*: variables that neither affect nor are affected by the boundary values of other variables.

1. Extended Algorithm

The algorithm proposed by Vij K. et al. [10] can create test cases by only using BVA and an experience-based technique for combination of input variables. To provide flexibility and allow users to design effective test cases, we made some changes to this algorithm. Our extended algorithm can apply BVA, ECP and some various techniques for combination of input values. More particularly, we extend Steps 3, 4, 6, 8, 9 and 10 in the algorithm proposed in [10]. All the steps are presented in detail below.

Step 1: Categorize variables into three sets:

- D: set of dependent variables,
- B: set of boundary-determining variables,
- I: set of independent variables.

Step 2: For each dependent variable $d \in D$, create a set $V_{B,d}$ of its determining variables. Every element in $V_{B,d}$ will also belong to set B. The boundary values of d are influenced by all elements in $V_{B,d}$.

Step 3: For each dependent variable $d \in D$, create distinct sets representing all possible boundary value ranges for d. Label these sets d_1 through d_x. If BVA is used, these sets will include the boundary values (min, min+, nom, max-, max). On the other hand, if ECP is used, these sets will only contain the nominal values (nom). Creating these sets in such a way that d can assume, based on its boundary-determining variables.

Step 4: For each boundary-determining variable $b \in B$, create sets of values that impact the boundary values of its dependent variable. These sets are labeled b_1 to b_y. It is important to note that if BVA is applied, sets of values used for each boundary-determining variable b will include the boundary values (min, min+, nom, max-, max). In contrast, the sets of values used for b will only contain the nominal values (nom) when ECP is applied. The relationship between boundary-value sets (d_x) and boundary-determining sets (b_y) is such that if b takes a value from b_y, then d can only take a value from the corresponding d_x. The details of these relationships will be outlined in Step 6.

Step 5: For each dependent variable $d \in D$, create all possible combinations of its boundary-determining sets. For example, if e and f are elements of $V_{B,d}$ with e having 2 boundary-determining sets (e_1, e_2) and f having 3 boundary-determining sets (f_1, f_2, f_3), the possible combinations would be: e_1f_1, e_1f_2, e_1f_3, e_2f_1, e_2f_2, e_2f_3.

Step 6: Depending on whether either BVA or ECP is used, either a boundary-value set or a nominal-value set is assigned to each possible combination of determining sets. The relationship between a combination of determining sets and a boundary value set is such that if the determining variables e and f take values from a combination e_pf_q, then the dependent variable d can only assume a value from the corresponding d_r only.

Step 7: Create all possible combinations of the boundary-determining sets for each boundary-determining variable. For example, if e, f, and g are elements of B with 2, 3, and 2 determining sets respectively, this will result in 12 possible combinations.

$$e_1f_1g_1,\ e_1f_1g_2,\ e_1f_2g_1,\ e_1f_2g_2,$$
$$e_1f_3g_1,\ e_1f_3g_2,\ e_2f_1g_1,\ e_2f_1g_2,$$
$$e_2f_2g_1,\ e_2f_2g_2,\ e_2f_3g_1,\ e_2f_3g_2.$$

Step 8: For each combination created in Step 7, assign the boundary-value sets or nominal-value sets for each of the dependent variables according to the associations established in Step 6.

Step 9: Assign values to all the independent variables for each combination, generating all possible sets of independent variables.

Step 10: Conduct traditional BVA or ECP on each set of independent variables to create test cases.

In our study, the technique for combination of input values is one of the following techniques: experience-based, weak, and strong combination.

- Experience-based combination is referred to as the combination technique applied in [10].
- Weak combination (a.k.a. 1-way) chooses one variable value from each partition subset (i.e., equivalence class) such that all classes are covered.
- Strong combination (a.k.a. n-way) is based on the Cartesian product of partition subsets, i.e., testing all interactions of all equivalence classes.

2. NextDate Example

Using the NextDate function as an example, we can demonstrate the outcomes of each step outlined in Section II.1. The NextDate function is a function that takes three variables: Month, Day, and Year, and returns the date of the day after the input date. In this function, the boundary values for Day depends on the values of Month and Year.

Step 1: $D = \{Day\}, B = \{Month, Year\}, I = \emptyset$.

Step 2: $V_{B,Day}$= {*Month*, *Year*}.

Step 3: If BVA is used, sets can be created as follows:
$Day_1 = \{1 \ldots 31\}$, $Day_2 = \{1 \ldots 30\}$,
$Day_3 = \{1 \ldots 28\}$, $Day_4 = \{1 \ldots 29\}$.
If ECP is used, sets can be created as follows:
$Day_1 = \{15\}$, $Day_2 = \{15\}$,
$Day_3 = \{15\}$, $Day_4 = \{15\}$.

Step 4: If BVA is used, sets can be created as follows:
$Month_1 = \{1, 3, 5, 7, 8, 10, 12\}$ (31 days),
$Month_2 = \{2\}$ (28 or 29 days),
$Month_3 = \{4, 6, 9, 11\}$ (30 days).
$Year_1 = \{1904, 1908, 1912, \ldots, 2000\}$ (leap years),
$Year_2 = \{1900, 1901, 1902, 1903, 1905, \ldots, 1999\}$. (non-leap years)
If ECP is used, sets can be created as follows:
$Month_1 = \{7\}$ (31 days),
$Month_2 = \{2\}$ (28 or 29 days),
$Month_3 = \{9\}$ (30 days).
$Year_1 = \{1948\}$ (leap years),
$Year_2 = \{1950\}$ (non-leap years).

Step 5: Create all possible combinations of boundary-determining sets for each dependent variable. In the case of the NextDate function, the only dependent variable is Day. Its determining value sets are as follows:
$Month_1Year_1$, $Month_1Year_2$,
$Month_2Year_1$, $Month_2Year_2$,
$Month_3Year_1$, $Month_3Year_2$.

Step 6: If BVA is used, assign a boundary value set or if ECP is used, assign a nominal-value set to each combination of determining sets. Table I presents an assignment of the value sets of Day to each combination of determining sets {Month, Year}.

Table I
ASSIGNMENT OF VALUE SETS OF DAY TO EACH COMBINATION OF DETERMINING SETS.

Determining value sets/Combinations	Boundary-value set/Nominal-value sets
$Month_1Year_1$	Day_1
$Month_1Year_2$	Day_1
$Month_2Year_1$	Day_4
$Month_2Year_2$	Day_3
$Month_3Year_1$	Day_2
$Month_3Year_2$	Day_2

Step 7: The possible combinations for all determining sets are identical to those listed in Step 5.

Step 8: Assign boundary-value sets or nominal-value sets to each combination from Step 7. The following sets are created:
$Month_1Year_1Day_1$, $Month_1Year_2Day_1$,
$Month_2Year_1Day_4$, $Month_2Year_2Day_3$,
$Month_3Year_1Day_2$, $Month_3Year_2Day_2$.

Step 9: This step is omitted due to the absence of independent variables.

Step 10: The results, obtained by applying the testing tool, are presented at the end of Section IV.

III. TESTING TOOL IMPLEMENTATION

Our automated test case generation tool was developed in C# using the .NET framework. Screens of the modules are provided to a user as forms. The generated test cases are displayed in reports. Figure 1 illustrates the process of generating test cases using our new testing tool.

The tool offers six screens to the user, with each screen representing a different module of the testing tool. Table II presents a comparison of the modules between the tool proposed in [10] and our new tool. It lists changes between two tools.

There are eight modules in our tool. These modules are: getVariables, getRelations, getNumOfSetsAndTechnique, getBounds, getNoms, getSetRelationsAndCombinationTechnique, generateBVATestCases and generateECPTestCases. Each module collects input from the user and forwards its output to the next module in sequence.

In our tool, two modules getNumOfSetsAndTechnique and getSetRelationsAndCombinationTechnique extends two modules getNumOfSets and getSetRelations in the tool proposed in [10], respectively. This extension allows a user to be able to make a choice in applying BVA or ECP and

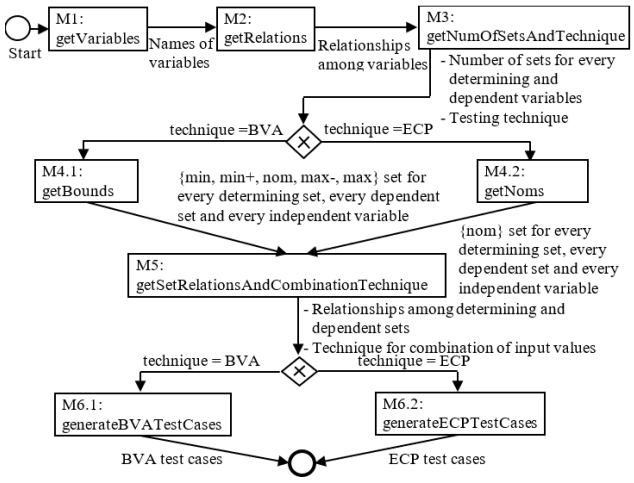

Figure 1. Process of generating test cases using the new testing tool

Table II
COMPARISON OF THE TOOL PROPOSED IN [10] AND THE NEW TOOL.

Module in the tool proposed in [10]	**Module in new tool**
Module 1: getVariables	Module 1: getVariables (no change)
Module 2: getRelations	Module 2: getRelations (no change)
Module 3: getNumOfSets	Module 3: getNumOfSetsAndTechnique (extended)
Module 4: getBounds	Module 4.1: getBounds (no change) Module 4.2: getNoms (new)
Module 5: getSetRelations	Module 5: getSetRelationsAndCombinationTechnique (extended)
Module 6: generateBVATestCases	Module 6.1: generateBVATestCases (no change) Module 6.2: generateECPTestCases (new)

a combination technique (experience-based, weak or strong combination). In addition, two new modules getNoms and generateECPTestCases will be used by the user to collect boundary or nominal values. The design of the modules is briefly described below.

Module 1: getVariables. This module prompts the user to input the names of up to 10 variables.

Module 2: getRelations. This module allows the user to specify the relationships between variables (using checkboxes), enabling the tool to identify boundary-determining, boundary-dependent, and independent variables.

Module 3: getNumOfSetsAndTechnique. This module asks the user to specify the number of sets that can be created for each boundary-dependent, boundary-determining and independent variables. These sets are known as boundary-dependent, boundary-determining and independent sets. Besides, the user is required to choose a black-box testing technique (i.e., BVA or ECP).

Module 4.1: getBounds. This module asks the user to provide the {min, min+, nom, max-, max} values for each independent variable, as well as for all boundary-dependent and boundary-determining sets.

Module 4.2: getNoms. This module asks the user to enter the {nom} values for each independent variable, and for all boundary-dependent and boundary-determining sets.

Module 5: getSetRelationsAndCombinationTechnique. This module creates all possible combinations of boundary-determining sets for each boundary-dependent variable. It prompts the user to link each combination of boundary-determining sets with a boundary-dependent set. For each combination, the module displays *n* radio buttons, where n is the total number of sets for the corresponding boundary-dependent variable. In addition, the user is required to choose a combination technique.

Module 6.1: generateBVATestCases and Module 6.2.: generateECPTestCases. These modules create all possible combinations of boundary-determining sets. They then match each combination with a boundary-dependent set for each boundary-dependent variable, according to user-defined associations and the combination technique specified in Module 5. Next, they link all combinations with the independent variables. Finally, Modules 6.1 and 6.2 generate test cases.

IV. DEMONSTRATION OF THE TOOL

We will use the NextDate function to illustrate an application of the tool. We follow the process of generating test cases as introduced in Section III.

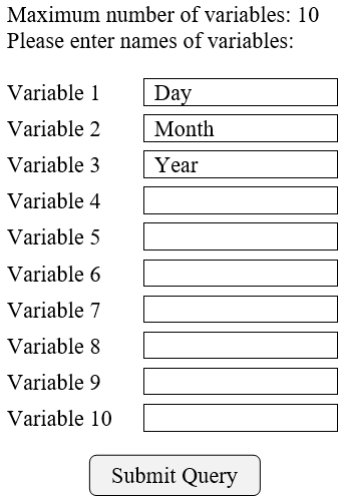

Figure 2. User Screen 1 - getVariables

Figure 2 displays User Screen 1 (getVariables), where the user enters the variable names such as Day, Month, and Year, then clicks "Submit Query".

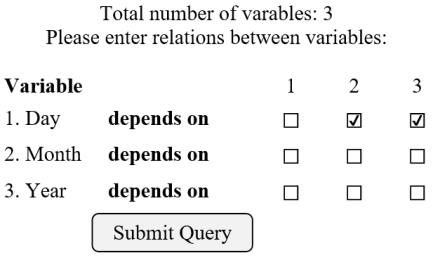

Figure 3. User Screen 2 - getRelations

Figure 3 depicts User Screen 2 (getRelations), where the user defines the relationships between the input variables. For instance, Day is dependent on Month and Year.

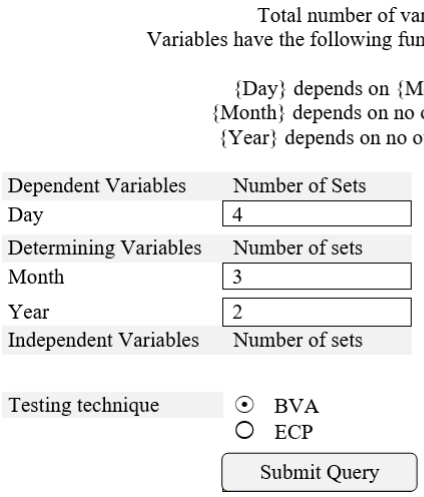

Figure 4. User Screen 3 - getNumOfSetsAndTechnique

Figure 4 illustrates User Screen 3 (getNumOfSetsAndTechnique). This screen prompts the user to input the number of possible sets for each boundary-dependent, boundary-determining and independent variable. There is no independent variable in this example. The testing technique is also chosen by the user.

- Day values can be categorized into four sets: {Day: $1 \leq$ Day ≤ 31}, {Day: $1 \leq$ Day ≤ 30}, {Day: $1 \leq$ Day ≤ 28} and {Day: $1 \leq$ Day ≤ 29}.
- Month values can be categorized into three sets: {Month: Month has 31 days}, {Month: Month is February with 28 or 29 days} and {Month: Month has 30 days}.
- Year values can be categorized into two sets: {Leap years: (Year mod 4 = 0 and Year mod 100 != 0) or (Year mod 400 = 0)} and {Non-leap years: (Year mod 4 != 0) or (Year mod 100 = 0)}.

If the BVA option has been chosen, User Screen 4 (getBounds) is displayed as shown in Figure 5. Here, the boundary values {min, min+, nom, max-, max} for all

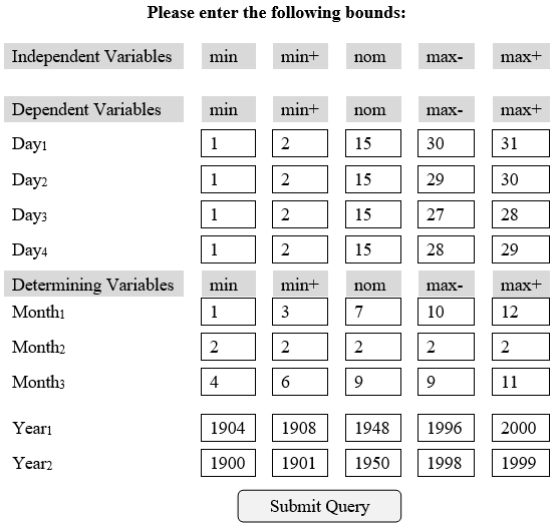

Figure 5. User Screen 4 - getBounds

the boundary-dependent and boundary-determining sets are identified. If the ECP option has been chosen, User Screen 5 (getNoms) is displayed as presented in Figure 6. The nominal values {nom} are specified for all the boundary-dependent and boundary-determining sets.

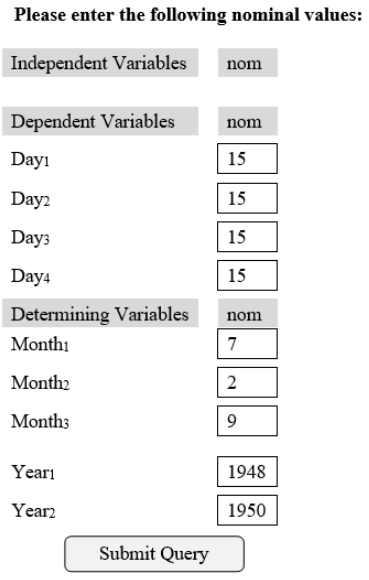

Figure 6. User Screen 5 - getNoms

Figure 7 presents User Screen 6 (getSetRelationsAndCombinationTechnique), where all possible combinations of the boundary-determining variables are generated. Each combination is linked to a boundary-dependent set. The resulting sets are created for either BVA or ECP.

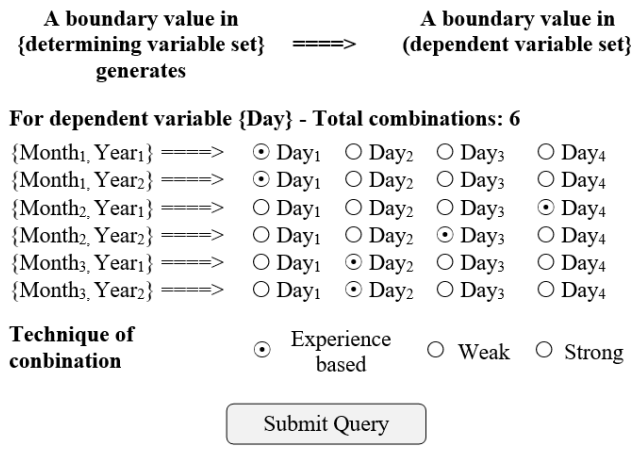

Figure 7. User Screen 6 - getSetRelationsAndCombinationTechnique

$\{Month_1, Year_1, Day_1\}$
$\{Month_1, Year_2, Day_1\}$
$\{Month_2, Year_1, Day_4\}$
$\{Month_2, Year_2, Day_3\}$
$\{Month_3, Year_1, Day_2\}$
$\{Month_3, Year_2, Day_2\}$

BVA and Experience-based Combination: If the user selects two options BVA and experience-based combination and the boundary values collected as shown in Figure 5, test cases generated as below. These test cases are identical to the ones shown in [10].

- BVA and Experience-based Combination Set #1 $\{Month_1, Year_1, Day_1\}$: (1, 1948, 15), (3, 1948, 15), (7, 1948, 15), (10, 1948, 15), (12, 1948, 15), (7, 1904, 15), (7, 1908, 15), (7, 1996, 15), (7, 2000, 15), (7, 1948, 1), (7, 1948, 2), (7, 1948, 30), (7, 1948, 31).
- BVA and Experience-based Combination Set #2 $\{Month_1, Year_2, Day_1\}$: (1, 1950, 15), (3, 1950, 15), (7, 1950, 15), (10, 1950, 15), (12, 1950, 15), (7, 1900, 15), (7, 1901, 15), (7, 1998, 15), (7, 1999, 15), (7, 1950, 1), (7, 1950, 2), (7, 1950, 30), (7, 1950, 31).
- BVA and Experience-based Combination Set #3 $\{Month_2, Year_1, Day_4\}$: (2, 1948, 15), (2, 1948, 15), (2, 1948, 15), (2, 1948, 15), (2, 1948, 15), (2, 1904, 15), (2, 1908, 15), (2, 1996, 15), (2, 2000, 15), (2, 1948, 1), (2, 1948, 2), (2, 1948, 28), (2, 1948, 29).
- BVA and Experience-based Combination Set #4 $\{Month_2, Year_2, Day_3\}$: (2, 1950, 15), (2, 1950, 15), (2, 1950, 15), (2, 1950, 15), (2, 1950, 15), (2, 1900, 15), (2, 1901, 15), (2, 1998, 15), (2, 1999, 15), (2, 1950, 1), (2, 1950, 2), (2, 1950, 27), (2, 1950, 28).
- BVA and Experience-based Combination Set #5 $\{Month_3, Year_1, Day_2\}$: (4, 1948, 15), (6, 1948, 15), (9, 1948, 15), (9, 1948, 15), (11, 1948, 15), (9, 1904, 15), (9, 1908, 15), (9, 1996, 15), (9, 2000, 15), (9, 1948, 1), (9, 1948, 2), (9, 1948, 29), (9, 1948, 30).
- BVA and Experience-based Combination Set #6 $\{Month_3, Year_2, Day_2\}$: (4, 1950, 15), (6, 1950, 15), (9, 1950, 15), (9, 1950, 15), (11, 1950, 15), (9, 1900, 15), (9, 1901, 15), (9, 1998, 15), (9, 1999, 15), (9, 1950, 1), (9, 1950, 2), (9, 1950, 29), (9, 1950, 30).

BVA and Weak Combination: If the user selects BVA and weak combination options and enters the boundary values as shown in Figure 5, test cases generated as below.

- BVA and Weak Combination Set #1 $\{Month_1, Year_1, Day_1\}$: (1, 1904, 1), (3, 1908, 2), (7, 1948, 15), (10, 1996, 30), (12, 2000, 31).
- BVA and Weak Combination Set #2 $\{Month_1, Year_2, Day_1\}$: (1, 1900, 1), (3, 1901, 2), (7, 1950, 15), (10, 1998, 30), (12, 1999, 31).
- BVA and Weak Combination Set #3 $\{Month_2, Year_1, Day_4\}$: (2, 1904, 1), (2, 1908, 2), (2, 1948, 15), (2, 1996, 28), (2, 2000, 28).
- BVA and Weak Combination Set #4 $\{Month_2, Year_2, Day_3\}$: (2, 1900, 1), (2, 1901, 2), (2, 1950, 15), (2, 1998, 27), (2, 1999, 28).
- BVA and Weak Combination Set #5 $\{Month_3, Year_1, Day_2\}$: (4, 1904, 1), (6, 1908, 2), (9, 1948, 15), (9, 1996, 29), (11, 2000, 30).
- BVA and Weak Combination Set #6 $\{Month_3, Year_2, Day_2\}$:(4, 1900, 1), (6, 1901, 2), (9, 1950, 1), (9, 1998, 29), (11, 1999, 30).

BVA and Strong Combination: If BVA and strong combination options are selected and the boundary values are collected as shown in Figure 5, test cases generated as below.

- BVA and Strong Combination Set #1 $\{Month_1, Year_1, Day_1\}$: (1, 1904, 1), (1, 1904, 2), (1, 1904, 15), (1, 1904, 30), (1, 1904, 31), (1, 1908, 1), (1, 1908, 2), (1, 1908, 15), (1, 1908, 30), (1, 1908, 31), (1, 1948, 1), (1, 1948, 2), (1, 1948, 15), (1, 1948, 30), (1, 1948, 31), (1, 1996, 1), (1, 1996, 2), (1, 1996, 15), (1, 1996, 30), (1, 1996, 31), (1, 2000, 1), (1, 2000, 2), (1, 2000, 15), (1, 2000, 30), (1, 2000, 31), (3, 1904, 1), (3, 1904, 2), (3, 1904, 15), (3, 1904, 30), (3, 1904, 31), (3, 1908, 1), (3, 1908, 2), (3, 1908, 15), (3, 1908, 30), (3, 1908, 31), (3, 1948, 1), (3, 1948, 2), (3, 1948, 15), (3, 1948, 30), (3, 1948, 31), (3, 1996, 1), (3, 1996, 2), (3, 1996, 15), (3, 1996, 30), (3, 1996, 31), (3, 2000, 1), (3, 2000, 2), (3, 2000, 15), (3, 2000, 30), (3, 2000, 31), (7, 1904, 1), (7, 1904, 2), (7, 1904, 15), (7, 1904, 30), (7, 1904, 31), (7, 1908, 1), (7, 1908, 2), (7, 1908, 15), (7, 1908, 30), (7, 1908, 31), (7, 1948, 1), (7, 1948, 2), (7, 1948, 15), (7, 1948, 30), (7, 1948, 31), (7, 1996, 1), (7, 1996, 2), (7, 1996, 15), (7, 1996, 30), (7, 1996, 31), (7, 2000, 1), (7, 2000, 2), (7, 2000, 15), (7, 2000, 30), (7, 2000,

31), (10, 1904, 1), (10, 1904, 2), (10, 1904, 15), (10, 1904, 30), (10, 1904, 31), (10, 1908, 1), (10, 1908, 2), (10, 1908, 15), (10, 1908, 30), (10, 1908, 31), (10, 1948, 1), (10, 1948, 2), (10, 1948, 15), (10, 1948, 30), (10, 1948, 31), (10, 1996, 1), (10, 1996, 2), (10, 1996, 15), (10, 1996, 30), (10, 1996, 31), (10, 2000, 1), (10, 2000, 2), (10, 2000, 15), (10, 2000, 30), (10, 2000, 31), (12, 1904, 1), (12, 1904, 2), (12, 1904, 15), (12, 1904, 30), (12, 1904, 31), (12, 1908, 1), (12, 1908, 2), (12, 1908, 15), (12, 1908, 30), (12, 1908, 31), (12, 1948, 1), (12, 1948, 2), (12, 1948, 15), (12, 1948, 30), (12, 1948, 31), (12, 1996, 1), (12, 1996, 2), (12, 1996, 15), (12, 1996, 30), (12, 1996, 31), (12, 2000, 1), (12, 2000, 2), (12, 2000, 15), (12, 2000, 30), (12, 2000, 31).

- Similarly, BVA and Strong Combination Set #2 {$Month_1$, $Year_2$, Day_1}, BVA and Strong Combination Set #3 {$Month_2$, $Year_1$, Day_4}, BVA and Strong Combination Set #4 {$Month_2$, $Year_2$, Day_3}, BVA and Strong Combination Set #5 {$Month_3$, $Year_1$, Day_2}, BVA and Strong Combination Set #6 {$Month_3$, $Year_2$, Day_2} are created using the Cartesian product of the corresponding sets.

ECP and Weak Combination: If the user selects ECP and weak combination options and enters the nominal values as shown in Figure 6, test cases generated as below:

- ECP and Weak Combination Set #1 {$Month_1$, $Year_1$, Day_1}: (7, 1948, 15).
- ECP and Weak Combination Set #2 {$Month_1$, $Year_2$, Day_1}: (7, 1950, 15).
- ECP and Weak Combination Set #3 {$Month_2$, $Year_1$, Day_4}: (2, 1948, 15).
- ECP and Weak Combination Set #4 {$Month_2$, $Year_2$, Day_3}: (2, 1950, 15).
- ECP and Weak Combination Set #5 {$Month_3$, $Year_1$, Day_2}: (9, 1948, 15).
- ECP and Weak Combination Set #6 {$Month_3$, $Year_2$, Day_2}: (9, 1950, 15).

The process of generating test cases using the tool offers a variety of paths designed to provide flexibility in the choice of testing and combination techniques. Adaptive design allows the tool to adjust its behavior, process, and interface according to the user's needs. Exhaustive testing is not possible. In addition, each test case adds to the cost of testing. Therefore, the user may need a suitable technique to choose a subset of all the combinations.

The ECP technique allows the user to need less effort to collect input values and reduce the number of test cases to be executed due to just selecting one representative value from each equivalence class (subset) as test data. When the ECP and Weak Combination option is chosen, a minimal set of six test cases is created for six possible combinations of the boundary-determining variables. One variable value is selected from each equivalence class (one Day_i, $Month_i$, and $Year_i$) such that all classes are covered. However, the ECP option ignores testing boundary values.

On the other hand, the BVA technique focuses on testing the boundary or edge values of input ranges and thus needs a greater number of test cases. To test every boundary value at least once, we need five test cases {min, min+, nom, max-, max} from each equivalence class. When the BVA and Weak Combination option is chosen, a set of thirty test cases is created for six possible combinations of the boundary-determining variables. These test cases are good enough to cover all equivalence classes and all boundary values. Therefore, the BVA and Weak Combination option may be more cost-effective than the BVA and Experience-based Combination option (used in the original algorithm) and BVA and Strong Combination option.

V. CONCLUSION

This paper addresses some limitations on the testing case design algorithm using a divide-and-rule approach in terms of advancement to increase the flexibility of the algorithm. The core idea is that the extended algorithm considers the integrated use of BVA, ECP and different combination techniques. Applying the extended algorithm described in this paper allows a user more options in selecting input values and output test data sets. This may reduce the cost of testing. The tool for automated test case generation was developed in C# language and can be effortlessly utilized by navigating the user interface and supplying the required inputs. Currently, the extended algorithm supports test case generation from numerical values. As a future direction, we plan to employ this algorithm to verify a mixing of different data types such as Boolean, numeric and string variables.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their comments on an earlier version of this paper. The authors acknowledge Can Tho University for financial support of this work. This study is funded in part by the Can Tho University, Code: T2023-123.

REFERENCES

[1] Pressman R. and Maxim B. "Software Engineering - A Practitioner's approach.", *9th Edition. The McGraw Hill Companies, Inc.*, New York (2020).

[2] Rao C. et al. "Combinatorial Test Generation for Multiple Input Models With Shared Parameters.", *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2606-2628 (2022).

[3] Nie C. and Leung H. "A survey of combinatorial testing.", *ACM Comput. Surv.* 43, 2, Article 11 (2011).

[4] Chen T.Y. and Poon P.L., Tang S.F., and Tse T.H. "On the identification of categories and choices for categorypartition test case generation.", *Information and Software Technology*, 46 (13) pp. 887-898 (2004).

[5] Chen T.Y., Cheng M.Y., Poon P.L., Tse T.H., and Yu Y.T. "A study of input domain partitioning.", *Proceedings of the 20th IASTED International Multi-Conference on Applied Informatics (AI 2002)*, ACTA Press, Calgary, Canada, pp. 176-181 (2002).

[6] Lei Y., Kacker R., Kuhn D. R., Okun V., and Lawrence J. "IPOG: A General Strategy for T-Way Software Testing.", *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, Tucson, AZ, USA, pp. 549-556 (2007).

[7] Debnath N. and Haakenson J.R. "Automated testing design and description for certain functions.", *Proceedings of the Sixth IEEE International Conference on Electro/Information Technology, (IEEE-EIT 2006)*, Michigan State University, East Lansing, MI, USA (2006).

[8] Feng W. and Zhang Z. "Sequence algorithms for boundary value analysis with constrained input parameters.", *Proceedings of the ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2005)*, Toronto, Canada, pp. 255-260 (2005).

[9] Burnstein, I. "Practical Software Testing: A Process-Oriented Approach.", *Springer*, New York (2006).

[10] Vij K. and Feng W. "Boundary Value Analysis Using Divide-and-Rule Approach.", *Fifth International Conference on Information Technology: New Generations (itng 2008)*, Las Vegas, NV, USA, pp. 70-75 (2008).

Danh Nguyen-Cong received his B.S degree in Computer Science from Can Tho University in 2000. In 2006, he received his M.S. degree in Information Technology from King Mongkut's University of Technology North Bangkok, Thailand. In 2009, he worked as a researcher at the Software Engineering Research Group (AGSE), the Technical University of Kaiserslautern, Germany. He completed his PhD in Computer Science at CNRS, LIMOS UMR 6158, Clermont Auvergne University, France in 2018. His research interests include Big Data, Software Process, and Software Quality Assurance and Testing.

Email: ncdanh@cit.ctu.edu.vn