Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

L′OP-ART: A linear-time adaptive random testing algorithm for object-oriented programs[☆]

Jinfu Chen[a,b,*], Jingyi Chen[a,b], Lili Zhu[a,b], Chengying Mao[c], Qihao Bao[a,b], Rubing Huang[d]

[a] School of Computer Science and Communication Engineering, Jiangsu University, 301 Xuefu Road, Zhenjiang, 212013, Jiangsu, China
[b] Jiangsu Key Laboratory of Security Technology for Industrial Cyberspace, Jiangsu University, 301 Xuefu Road, Zhenjiang, 212013, Jiangsu, China
[c] School of Software and IoT Engineering, Jiangxi University of Finance and Economics, Nanchang, 330013, Jiangxi, China
[d] School of Computer Science and Engineering, Macau University of Science and Technology, Taipa, 999078, Macao Special Administrative Region of China

ARTICLE INFO

Keywords:
Object-oriented software testing
Random testing
Adaptive random testing
Testing effectiveness
Test case generation

ABSTRACT

Object-oriented (OO) programming offers useful and desirable characteristics in the resulting code. These OO characteristics, however, demand an effective testing method. Adaptive random testing (ART) serves as an effective testing technique where test cases are distributed uniformly according to a proper metric for difference (distance) between test cases. Recently proposed object and method invocation sequence similarity (OMISS) distance metric was incorporated into a new ART algorithm (OMISS-ART) for testing OO software. Unfortunately, OMISS-ART's high effectiveness in finding faults comes at significantly higher computational overheads. To alleviate the problem, we present in this paper a linear-time ART algorithm called L′OP-ART (**L**inear-time **O**bject-oriented **P**rogram testing by **ART**) based on a lightweight OMISS metric. L′OP-ART designs specific data structures that store information about executed test cases, thus, the differences between candidates and executed test cases can be obtained with reduced overheads. Empirical studies show that L′OP-ART detects failures more effectively than random testing (RT), and has comparable fault-detection effectiveness to other ART algorithms for OO testing (according to the F-measure, or the expected number of test case executions required to find a failure). Furthermore, the test case generation overhead of L′OP-ART is low, close to that of RT.

1. Introduction

Object-oriented (OO) programming has become one of the most popular approaches to programming, producing software with distinctive characteristics of method invocations, inheritance, and polymorphism. However, while these OO characteristics greatly improve software reusability, extensibility, and interoperability, they also raise many challenges in software testing (Binder, 1996).

Software testing is an important part of software quality assurance, and has seen important research achievements over the past few decades. Many approaches and techniques for testing OO software have been proposed (Chen et al., 2021; Bertolino, 2007; Arcuri and Yao, 2008; Binder, 2000; Inkumsah and Xie, 2008; Le Traon et al., 2000; Chen and Tse, 2013; Ciupa et al., 2008; Pacheco and Ernst, 2007). Amongst these approaches, random testing (RT) is particularly popular for its simplicity and ease of use (Ciupa et al., 2008; Pacheco et al., 2007). Based on evidence that well-spread and evenly distributed test cases are more effective at finding failures, adaptive random testing (ART) (Chen et al., 2010; Huang et al., 2021; Chen et al., 2004b, 2020; Ackah-Arthur et al., 2019; Hui et al., 2021; Huang et al., 2022) has been widely investigated as an enhancement to RT. In general, an ART implementation selects from randomly-generated candidate test cases based on their distance to those that have already been executed. ART has been applied in numerical input domains, where a simple distance measure, such as the Euclidean distance, can be used (Chen et al., 2010), as well as non-numerical input domains, where more distance measurements are used. Barus et al. (2016) proposed ARTsum, a linear-time ART algorithm that applies the *max-sum* selection criterion to structured inputs based on a category-choice distance metric.

OO software, partly due to its unique characteristics, presents a number of challenges to software testers wanting to conduct thorough testing. Its complex structure and classes interactions, for example, may render the calculation of the distance between test cases impractical or

[☆] Editor: A. Bertolino.

[*] Corresponding author at: School of Computer Science and Communication Engineering, Jiangsu University, 301 Xuefu Road, Zhenjiang, 212013, Jiangsu, China.

E-mail addresses: jinfuchen@ujs.edu.cn (J. Chen), 2112208004@stmail.ujs.edu.cn (J. Chen), lilizhu@ujs.edu.cn (L. Zhu), maochy@yeah.net (C. Mao), qihaobao@ujs.edu.cn (Q. Bao), rbhuang@must.edu.mo (R. Huang).

https://doi.org/10.1016/j.jss.2024.111970
Received 26 May 2023; Received in revised form 27 December 2023; Accepted 12 January 2024
Available online 18 January 2024

Nomenclature

Acronyms and Abbreviations

OO	Object-oriented
RT	Random testing
SUT	Software under test
ART	Adaptive random testing
RefV	Reference variable
NRefV	Non-reference variable
OBJ	Objects set
MINV	Method invocations
LenD	Length difference
MsD	Method set difference
SD	Sequence difference

Notations

E	Executed test case set
C	Candidate set
α	Coefficients
A	Set of attributes
N	NRefV attributes
R	RefV attributes
M	Set of methods
p, q	Any two objects
Q	A set of objects
$Q.M$	The set of methods in Q
$p.A.R$	p's reference attributes
$p.A.N$	p's non-reference attributes
$Q.A.R$	A set of $q.A.R$
$Q.A.N$	A set of $q.A.N$
t	A test case
$t.OBJ$	The list of objects in t
$t.MINV$	The method invocation sequence in t
$k_{t.OBJ}$	The number of objects in $t.OBJ$
$k_{t.MINV}$	The length of $t.MINV$
$k_{p.M}$	The number of methods in $p.M$
$k_{p.A.N}$	The number of non-reference attributes in $p.A.N$
pG	*NRefV* attributes sets for $p.A.N$
$pG[i][j]$	The jth non-reference variable of $pG[i]$
$PL_s(qG[i],\ j)$	The jth value in the sth permutation of $qG[i]$
$dist(\mathrm{x},\ \mathrm{y})$	The distance between x and y
S, M, N, W	Structure tuples
g	The maximum number of methods in the MINV associated with an element of E
h	The total number of different methods in E
s_0^0	The number of test cases whose MINV is not null
s_{0i}^0	The number of test cases in which the length of MINV less than or equal to i
s_{0i}^1	The sum of the lengths of MINV that are less than or equal to i
s_i^x	The number of test cases with the length of MINV being i, and with the method with label x appearing in this sequence
s_i^0	The number of test cases in which the length of MINV is i
$s_{i,j}^x$	The number of test cases in which the length of MINV is i, and the jth method in this sequence is with label x
c	A candidate test case
$ut.MINV$	The reduced $MINV$ created by removing repeated methods in $t.MINV$
$R(t.MINV)$	The sequence of method labels in $t.MINV$
k_t	The length of $t.MINV$
$k_{E.OBJ}$	The length of object sets in $E.OBJ$
ko_t	The number of objects in $t.OBJ$
k_{ut}	The length of $ut.MINV$
$t.MINV[j]$	The jth method in $t.MINV$
$ut.MINV[j]$	The jth method in $ut.MINV$
τ	The number of test cases in which the lengths of MINV less than or equal to k_c
$\langle x_1, x_2, \dots, x_n \rangle$	A permutation of $\{1, 2, \dots, n\}$
$\{a_1, a_2, \dots, a_b\}$	A subset of $\{1, 2, \dots, n\}$
$up.M$	The reduced behavior section of p
Y	A single collection
$Y.d_x^0$	The number of elements (in Y) that are greater than x
$Y.d_x^{-0}$	The number of elements (in Y) that are less than x
$Y.d_x^1$	The sum of all the elements (in Y) that are greater than x
$Y.d_x^{-1}$	The sum of all the elements (in Y) that are less than x
$\ell_{i,j}^1$	The sum of positive values in Y
$\ell_{-i,j}^1$	The sum of negative values in Y
ℓ_i^0	The number of positive values whose digits are i
ℓ_i^1	The sum of positive integers whose digits are i
ℓ_{-i}^0	The number of negative values whose digits are i
ℓ_{-i}^1	The sum of negative integers whose digits are i
F_m	The number of test inputs required to find the first failure

even impossible. However, given OO software's status as both mainstream and popular, the potential to apply ART test case generation technology to its testing is attractive and useful, and research in this direction has already seen success. The key is finding an appropriate distance measurement or metric. Ciupa et al. (2006, 2008) made the first attempt to propose a metric that measured the distance between two objects from the same class or derived from the same class. They created an ART tool named ARTOO using this metric and found that it outperformed RT in terms of the F-measure (the expected number of test cases executed to find the first failure). In order to calculate the distance between objects, their proposed algorithm tracks the used objects and available objects. In other words, ARTOO maintains lists of objects for method calls, which can be updated each time when the input is tested.

Ciupa et al.'s metric could only be used to generate test cases for a single class method at a time. Lin et al. (2009) proposed an improved approach that could test a group of class methods, and incorporated it in a new tool called ARTGen. Experimental results showed that ARTGen could find more faults than ARTOO.

Nevertheless, there is a problem common to ARTOO and ARTGen. Their metrics do not consider the even spreading of inputs to OO software in terms of method invocation sequence distances, object distances, and different types of elementary distances (including string, integer, character, and Boolean distances). To address this issue, Chen et al. (2016) proposed the object and method invocation sequence similarity (OMISS) metric to compute the distances among such inputs. They also proposed a corresponding ART algorithm called OMISS-ART. Although OMISS-ART has been shown to have higher failure-detection effectiveness than both ARTOO and ARTGen in terms of the F-measure, it has also been found to incur an $O(n^2)$ computational overhead. This indicates its low efficiency and difficulty to handle large-scale testing.

To reduce the computational overhead, in this paper, we present an innovative method and the related Lightweight distance metrics for calculating the sum of OMISS distances between a candidate and a set of test cases. Also we prove the computational overhead of proposed lightweight distance metrics is linear order in theory. And adapting the *max-sum* selection criterion, we propose a linear-time ART algorithm called linear-time object-oriented programming ART (L′OP-ART). The performance of L′OP-ART is evaluated on several subject programs, measured by the standard effectiveness metric (F-measure) and its test case generation time. We open the source codes and subject programs of the proposed algorithm at https://github.com/JyC00/LOP-ART.

The rest of this paper is organized as follows: Section 2 summarizes the background information about adaptive random testing and the OMISS metric. Section 3 introduces a lightweight version of the OMISS metric, and present L′OP-ART in detail. Section 4 presents the empirical study of L′OP-ART. Section 5 presents and analyzes the experimental results. Section 6 provides an analysis of the threats to validity. Section 7 further discusses the results. Section 8 reviews the related work. Section 9 concludes the paper, and points out the future work.

2. Background and preliminaries

2.1. Adaptive random testing

Chen et al. reported that failure-causing inputs tend to form contiguous regions (Chen et al., 2013). Based on this observation, adaptive random testing (ART) (Anand et al., 2013; Chen et al., 2009, 2013; Chen and Tse, 2013; Chen et al., 2004b; Chan et al., 2006b; Chen et al., 2004c,a; Liu et al., 2011; Mayer, 2005; Shahbazi et al., 2012; Chen et al., 2019a) was proposed to improve the effectiveness of random testing by evenly spreading test cases across the input domain. A number of ART algorithms have been proposed in the past decade (Pacheco and Ernst, 2007; Ciupa et al., 2008; Chen and Tse, 2013; Chen et al., 2004b; Chan et al., 2006b; Chen et al., 2004c,a; Liu et al., 2011; Mayer, 2005; Shahbazi et al., 2012; Chen et al., 2019b; Omari et al., 2019), among which the fixed-size-candidate-set implementation of ART (FSCS-ART) (Chen and Tse, 2013) is of particular popularity for its easy understanding and implementation. FSCS-ART employs two sets of test cases to perform testing: *executed set* (denoted E); and *candidate set* (denoted C), whose elements are generated randomly when a new test case is required. For each test, it selects a test case from C that is the most different from E, according to a specific distance criteria, such as the *max–min* criterion where the selected candidate is the one whose *smallest* distance to E is the *largest*. Another popular selection criterion is the *max-sum* criterion, which give the highest priority to the candidate whose *sum* of distances to E is the *largest* (Barus et al., 2016).

There are two main challenges in the application of ART to OO software testing: (1) what distance metric to use to deal with OO inputs; and (2) how to manage the overheads incurred by the calculation of distance between candidate test cases and executed test cases? The former concerns the effectiveness and the latter the efficiency. As the number of executed test cases increases, the number of distance calculations increases, which leads to higher computational overheads and lower the practicality of ART. Reducing such overheads is a critical issue in OO software testing with ART.

2.2. Objects and test cases in OO software

An object is an individual, identifiable entity, which has a value for each visible property (called attribute value) and a set of visible behaviors (called methods). Suppose the class "Account" has objects *LucyAccount* and *LinusAccount*. In *LucyAccount*, for example, the value of the *id* attribute is 001, the value of the *owner* attribute is Lucy, and the value of the *balance* attribute is $0.05. It has methods, i.e., *debit* and *credit*.

An object structure is the model that represents the object. It consists of the attribute values and the methods. For example, the object structure of *LucyAccount* comprises attribute values *id* = 001, *owner* = Lucy, and *balance* = $0.05 as well as methods *debit* and *credit*.

A class is a blueprint or template that defines a common structure and behavior for a group of objects, which share the same set of methods and attributes, possibly with different values for those attributes. For example, objects *LucyAccount* and *LinusAccount* are two of the instances of class "Account". Each object of class "Account" has values for attributes *id*, *owner*, and *balance*, and methods *debit* and *credit*.

A class structure is the model that represents a class. It consists of the attributes and the methods defined for the class. For example, the class structure for Account comprises attributes *id*, *owner*, and *balance* as well as methods *debit* and *credit*.

Method invocations, inheritance, and polymorphism are the core elements of OO programming. Most object-oriented programming languages use the concept of class for encapsulation, and use class derivation to enable a class (derived class) to inherit the attributes and methods from another class (base class). Polymorphism allows software to decide at runtime whether to use a method defined in the base class or its overridden version defined in the derived class.

Fig. 1 shows a typical object structure, consisting of an attribute section and a behavior section — both contain self-defined elements (defined within the class itself) and inherited ones (derived from the base class). An attribute can be either a *reference variable* or a *non-reference variable*: a reference variable (abbreviated as *RefV*) refers to another attribute, while a non-reference variable (abbreviated as *NRefV*) stores the value of a primitive data type, such as *integer*, *real precision* number, *Boolean* value, *character*, and *string*.

Classes provide semantic information that can help with the design of test cases for the OO software. A test case t often consists of two parts: $t.OBJ$ and $t.MINV$, where $t.OBJ$ is a list of objects and $t.MINV$ is the method invocation sequence in the test case. If a method is non-static, its invocation requires an object, with both the object and method referring to the same class. Therefore, for any non-static method called in $t.MINV$, the corresponding invoker object must be in $t.OBJ$. In general, neither the objects section($t.OBJ$) nor the method invocations section($t.MINV$) of a test case is null. However, when testing only the correctness of the constructors, the method invocations section is null. When testing the correctness of static methods, the objects section is null.

2.3. The OMISS metric

Chen et al. (2016) proposed the OMISS metric to measure the dissimilarity of test cases involving multiple objects and multiple methods. If a test case t consists of an object set ($t.OBJ$) and a method invocation sequence ($t.MINV$), the distance (*TestcaseDistance*) between test cases t_1 and t_2 is defined as the sum of their method invocation sequence distance (*TCmSeqDist*) and their objects distance (*TCobjDist*):

$$\begin{aligned} TestcaseDistance(t_1, t_2) = {} & TCmSeqDist(t_1.MINV, t_2.MINV) \\ & + TCobjDist(t_1.OBJ, t_2.OBJ) \end{aligned} \tag{1}$$

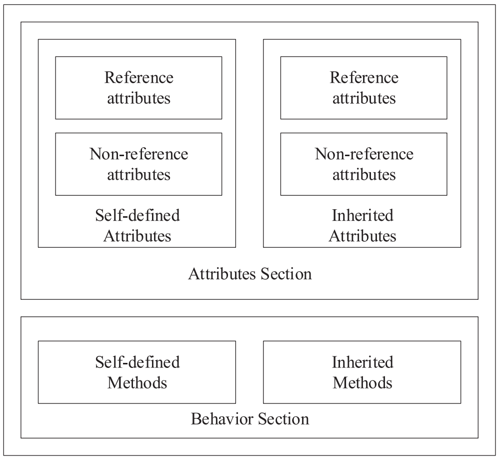


Fig. 1. Object structure.

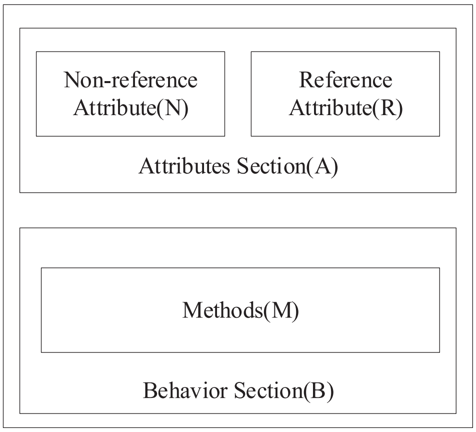


Fig. 2. Key object elements in our distance formula.

2.3.1. *Distance between method invocation sequences*

The method invocation sequence distance is a measure of how different two method invocation sequences are. It measures the difference between two sequences in length (*LenD*), in common methods (*MsD*), and in the ordered sequence (*SD*). Given a test case t, let $k_{t.MINV}$ denote the length of $t.MINV$. We have:

$$\begin{aligned}
&TCmSeqDist(t_1.MINV,\ t_2.MINV)\\
&\quad = LenD(t_1.MINV,\ t_2.MINV) + MsD(t_1.MINV,\ t_2.MINV)\\
&\quad + SD(t_1.MINV,\ t_2.MINV),\\
&\text{where}\quad LenD(t_1.MINV,\ t_2.MINV) = |k_{t_1.MINV} - k_{t_2.MINV}|,\\
&MsD(t_1.MINV,\ t_2.MINV)\\
&= \begin{cases} 1 - \frac{|t_1.MINV \cap t_2.MINV|}{|t_1.MINV \cup t_2.MINV|} & \text{if } \min(k_{t_1.MINV},\ k_{t_2.MINV}) > 0\\ 0 & \quad\text{otherwise}\end{cases}\\
&\text{and} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (2)\\
&SD(t_1.MINV,\ t_2.MINV)\\
&= \begin{cases} \frac{\sum_{i=1}^{k} d_i}{k}, \text{where } d_i = \begin{cases} 1 \text{ if } (t_1.MINV[i] \neq t_2.MINV[i])\\ 0 \text{ otherwise}\end{cases}\\ \qquad \text{and} \quad k = \min(k_{t_1.MINV},\ k_{t_2.MINV}),\\ \qquad \text{if} \quad \min(k_{t_1.MINV},\ k_{t_2.MINV}) > 0\\ 0 \qquad \text{otherwise}\end{cases}
\end{aligned}$$

where, for any test cases t_1 and t_2, $t_1.MINV \cap t_2.MINV$ is the set of methods appearing in both t_1 and t_2, and $t_1.MINV \cup t_2.MINV$ is the set of methods appearing in either t_1 or t_2.

2.3.2. *Distance between objects*

The object distance is a measure of how different two objects are.Unlike the general object structure shown in Fig. 1, the original OMISS metric does not differentiate between inherited and self-defined elements, and uses a simpler structure (shown in Fig. 2). In Fig. 2, $A = N \cup R$ denotes a list of *NRefV* and *RefV* attributes, and M denotes a list of methods associated with the object. The distance (*ObjDist*) between two objects p and q is defined as the sum of their behavior distance (*BehDist*) and attribute distance (*AttDist*):

$$ObjDist(p,\ q) = \begin{cases} BehDist(p.M,\ q.M) + AttDist(p.A,\ q.A)\\ \qquad\qquad \text{if both } p \text{ and } q \text{ are not null}\\ 0 \qquad\qquad \text{if both } p \text{ and } q \text{ are null}\\ 2 \qquad\qquad \text{otherwise}\end{cases} \tag{3}$$

where $p.M$ and $q.M$ denote the behavior sections of p and q, respectively; and $p.A$ and $q.A$ denote their attribute sections. Behavior distance is defined as the sum of the difference in the number of methods (*BLenD*) and the distance between their behavior sections (*BMsD*). Let $k_{p.M}$ denote the number of methods in $p.M$. We have:

$$\begin{aligned}
&BehDist(p.M,\ q.M) = BLenD(p.M,\ q.M) + BMsD(p.M,\ q.M),\\
&\text{where}\quad BLenD(p.M,\ q.M) = |k_{p.M} - k_{q.M}|,\\
&\text{and } BMsD(p.M,\ q.M) = \begin{cases} 1 - \frac{|p.M \cap q.M|}{|p.M \cup q.M|}\\ \qquad \text{if} \quad \min(k_{p.M},\ k_{q.M}) > 0\\ 0 \qquad \text{otherwise}\end{cases}
\end{aligned} \tag{4}$$

The attribute section of an object p, denoted by $p.A$, consists of the reference ($p.A.R$) and non-reference attributes ($p.A.N$), respectively. Therefore, the attribute distance is measured based on the reference and non-reference distances (*refDist* and *nonRefDist*):

$$\begin{aligned}&AttDist(p.A,\ q.A)\\ &= refDist(p.A.R,\ q.A.R) + nonRefDist(p.A.N,\ q.A.N)\end{aligned} \tag{5}$$

The reference distance is calculated by using the following formula recursively:

$$refDist(p.A.R,\ q.A.R) = TCobjDist(p.A.R,\ q.A.R) * \alpha, \tag{6}$$

where α is a non-negative constant 1/2, consistent with the set-up in Chen et al. (2016). (*TCobjDist* that represents the distance between two reference objects will be discussed in Section 2.3.3.)

The non-reference distance contains two kinds of calculations, one based on types (*typeDist*) and the other based on values (*secDist*), represented by (a, b)[1]:

$$\begin{aligned}&nonRefDist(p.A.N,\ q.A.N)\\ &= (typeDist(p.A.N,\ q.A.N), secDist(p.A.N,\ q.A.N)).\end{aligned} \tag{7}$$

Because of the two-layer measurement related to non-reference distance, the test input distance also has primary and secondary distances, in the form of (a, b). Given any two distances $dist_1 = (a, b)$ and $dist_2 = (c, d)$, the two basic operations of addition and multiplication, and the comparison between any two distances $dist1 = (a, b)$ and $dist2 = (c, d)$ are defined as follows:

- Operation 1. **Addition** (+): $dist_1 + dist_2 = (a,\ b) + (c,\ d) = (a + c,\ b + d)$; $dist_1 + n = (a + n,\ b)$.
- Operation 2. **Multiplication** (×): $dist_1 \times n = (a \times n,\ b \times n)$, where n is an integer.

[1] Note that Chen et al. used the alternative notation $a + b\mathbf{i}$ instead of $(a,\ b)$ in the original OMISS metric (Chen et al., 2016).

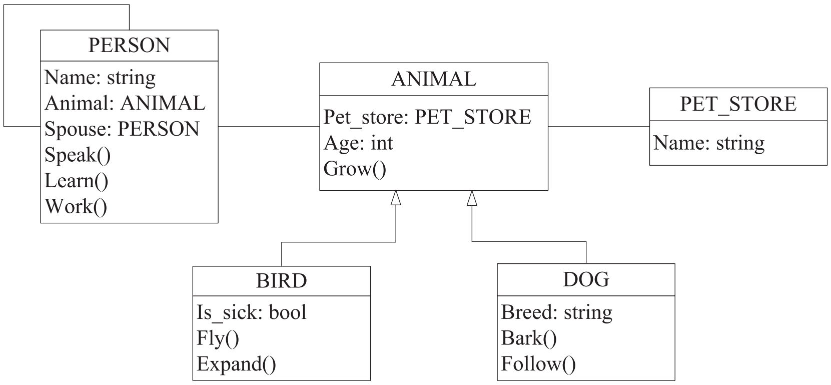


Fig. 3. Class diagram for an animal system with behavior methods.

Table 1
Test cases for the animal system in Fig. 3.

Test case (t_1)	Test case (t_2)	Test case (t_3)

{
PET_STORE Ps1(”Store1”);
BIRD Bird1(3, &Ps1,true);
Bird1.Grow();
Bird1.Expand();
Bird1.Fly();
}
	{
	PET_STORE Ps1(”Store1”);
	BIRD Bird2(1, &Ps1,false);
	Bird2.Grow();
	Bird2.Expand();
	Bird2.Expand();
	Bird2.Fly();
	}
		{
		PET_STORE Ps2(”Store2”);
		DOG Dog1(10, &Ps2,“good”);
		Dog1.Grow();
		Dog1.Bark();
		Dog1.Follow();
		}

- Operation 3. **Comparison** (=, >): $dist_1 = dist_2$ iff ($a = c$) and ($b = d$); $dist_1 > dist_2$ iff ($a > c$) or ($a = c$ and $b > d$).

The type distance (*typeDist*) is the sum of the difference in the number of non-reference attributes (*TSizeDiff*) and the distance between the non-reference attribute sections (*TSetDiff*). If $k_{p.A.N}$ denotes the number of non-reference attributes in $p.A.N$, we have:

$$\begin{aligned}&typeDist(p.A.N,\ q.A.N)\\&= TSizeDiff(p.A.N,\ q.A.N) + TSetDiff(p.A.N,\ q.A.N),\\&\text{where}\ \ TSizeDiff(p.A.N,\ q.A.N) = |k_{p.A.N} - k_{q.A.N}|,\\&\text{and}\ TSetDiff(p.A.N,\ q.A.N)\\&\quad = |p.A.N \cup q.A.N| - |p.A.N \cap q.A.N|.\end{aligned} \tag{8}$$

The distance in attribute values (data variables) is calculated as follows:

$$\begin{aligned}&(1)\ \text{For integers and real numbers: } dist(x,\ y) = \frac{|x-y|}{Range1}.\\&(2)\ \text{For characters: } dist(x,\ y) = \frac{|ASCII(x) - ASCII(y)|}{Range2}.\\&(3)\ \text{For Booleans: } \begin{cases} dist(x,\ y) = 0 & \text{if } x = y;\\ dist(x,\ y) = 1 & \text{otherwise.}\end{cases}\end{aligned} \tag{9}$$

(4) For strings: the Levenshtein distance (also known as edit distance) (Levenshtein, 1966),

$$dist(x,\ y) = \frac{editDistance(x,\ y)}{Range3}.$$

where $dist(x,\ y)$ denotes the distance between two values x and y that are of the same type, $ASCII(x)$ represents the ASCII code of the given character x, and *editDistance*$(x,\ y)$ is the edit distance between strings x and y — the minimum number of single-character edits (insertion, deletion, or substitution) required for conversion of one string into the other. In the four definitions of distance, *Range1*, *Range2*, and *Range3* are non-negative constants within specific ranges based on the requirements specification or user input.

Before calculating the elementary distance between objects p and q, the non-reference variables in both need to be grouped according to their data type (numbers, characters, Booleans, or strings), as shown in Eq. (9): pG={$pG[1]$, …, $pG[4]$} for $p.A.N$; and qG={$qG[1]$, …, $qG[4]$} for $q.A.N$, where $pG[x]$ and $qG[x]$ are groups of attributes of the same type from $p.A.N$ and $q.A.N$, respectively. The non-reference attribute distance in values is:

$$\begin{aligned}&secDist(p.A.N,\ q.A.N)\\&= \sum_{i=1}^{4} \min\Big(\bigcup_{s=1}^{m!}\{\sum_{j=1}^{m} dist(pG[i][j],\ PL_s(qG[i],\ j))\}\Big).\end{aligned} \tag{10}$$

where $m = \max(size(pG[i]),\ size(qG[i])) > 0$, and $PL_s(qG[i], j)$ is the jth value in the sth permutation of attribute value set $qG[i]$. For more details about the original OMISS metric, please refer to Chen et al. (2016).

2.3.3. Distance between object sets

The objects distance is a measure of the difference between two sets of objects, and is defined as the sum of the distances between each pair of objects, one from each set. If the sets have different numbers of objects, the smaller set is augmented by adding adequate *null* objects so that it will have the same size as the larger set (k). There are thus $k!$ permutations, each with k pairs of objects. For example, if $t_1.OBJ$={$p1$, $p2$, $p3$}, and $t_2.OBJ$={$q1$, $q2$}, then t_2 is augmented to become {$q1$, $q2$, *null*}, and then the $3! = 6$ permutations are determined:

$$\begin{aligned}P1 &= \{\{p1, q1\}, \{p2, q2\}, \{p3, null\}\},\\P2 &= \{\{p1, q1\}, \{p2, null\}, \{p3, q2\}\},\\P3 &= \{\{p1, q2\}, \{p2, q1\}, \{p3, null\}\},\\P4 &= \{\{p1, q2\}, \{p2, null\}, \{p3, q1\}\},\\P5 &= \{\{p1, null\}, \{p2, q1\}, \{p3, q2\}\},\\P6 &= \{\{p1, null\}, \{p2, q2\}, \{p3, q1\}\}.\end{aligned}$$

The distance between the two sets of objects is the smallest sum of distances for each pair of objects (*ObjDist*). For any test case t, if $k_{t.OBJ}$ denotes the number of objects in $t.OBJ$, for any two test cases t_1 and

Table 2
Distance between objects.

Between the pair of two sets of objects	Permutations	Between the pair of two objects	*ObjDist*	Distance of permutations	*TCobjDist*
$(t_1.OBJ,\ t_2.OBJ)$	1	(Ps1, Ps1)	(0, 0)	(0, 1.02)	(0, 1.02)
		(Bird1, Bird2)	(0, 1.02)		
	2	(Ps1, Bird2)	(6, 0)	(12, 0)	
		((Bird1, Ps1))	(6, 0)		
$(t_1.OBJ,\ t_3.OBJ)$	1	(Ps1, Ps2)	(0, 0.1)	(2.8, 0.22)	(2.8, 0.22)
		(Bird1, Dog1)	(2.8, 0.12)		
	2	(Ps1, Dog1)	(4, 0.5)	(10, 0.5)	
		(Bird1, Ps2)	(6, 0)		
$(t_2.OBJ,\ t_3.OBJ)$	1	(Ps1, Ps2)	(0, 0.1)	(2.8, 0.24)	(2.8, 0.24)
		(Bird2, Dog1)	(2.8, 0.14)		
	2	(Ps1, Dog1)	(4, 0.5)	(10, 0.5)	
		(Bird2, Ps2)	(6, 0)		

Table 3
Distance between method invocation sequences.

Between the pair of	Classified distance			*TCmSeqDist*
	LenD	*MsD*	*SD*	
$(t_1.MINV,\ t_2.MINV)$	1	0	0.33	1.33
$(t_1.MINV,\ t_3.MINV)$	0	0.80	0.67	1.47
$(t_2.MINV,\ t_3.MINV)$	1	0.84	0.67	2.51

t_2, we have:

$$TCobjDist(t_1.OBJ,\ t_2.OBJ) = \begin{cases} \min\left(\bigcup_{i=1}^{k!} \sum_{j=1}^{k} ObjDist(t_1.obj_j,\ PL_i(t_2.OBJ,\ j))\right) & \\ \qquad \text{where } \ k = \max(k_{t_1.OBJ},\ k_{t_2.OBJ}), & \\ \qquad \text{if } \ \max(k_{t_1.OBJ},\ k_{t_2.OBJ}) > 0 & \\ 0 \qquad \text{otherwise} & \end{cases} \tag{11}$$

where $PL_i(t_2.OBJ,\ j)$ denotes the jth element of the ith permuted list of $t_2.OBJ$.

Example 1

Ciupa et al. (2006) used a PET_STORE example to explain ARTOO, which was extended by Chen et al. (2016) to illustrate the characteristics of method invocations for the original OMISS metric — as shown in Fig. 3.

Assuming three program inputs t_1, t_2, and t_3 (as shown in Table 1), we set $Range1 = 100$ and $Range3 = 10$ for the calculation of attribute value distance.

For easy reference, Table 2 shows the distance between any two objects in the three inputs, and

Table 3 shows the distance between any two method invocation sequences.

The distance between each pair of the three inputs can thus be calculated as:

TestcaseDistance$(t_1,\ t_2) = (1.33,\ 1.02)$,
TestcaseDistance$(t_1,\ t_3) = (4.27,\ 0.22)$,
TestcaseDistance$(t_2,\ t_3) = (5.31,\ 0.24)$.

The computed distances are consistent with our intuition that t_2 is closer to t_1 than t_3, and t_1 is closer to t_3 than t_2.

Testing OO software with ART requires consideration of both the distance metric and the computational overheads. The original OMISS metric can be used to calculate the distance between two OO software inputs involving multiple objects and methods. Based on this metric, Chen et al. (2016) proposed OMISS-ART as an implementation of FSCS-ART using a *max–min* selection criterion. To reduce the $O(n^2)$ computational overheads incurred by OMISS-ART, they applied the random forgetting strategy (Chan et al., 2006a) — choosing a fixed number of the already executed test cases, and only using these in distance calculations to determine the best candidate.

3. Our methodology

Barus et al. (2016) proposed a linear-time ART algorithm named ARTsum that uses the *max-sum* criterion and the category-choice distance metric to reduce the computational overhead for distance computing. In ARTsum, the selection of the next test case takes constant time to calculate the sum of distances between a candidate and the already executed test case set. This is achieved through the use of an integer tuple that stores the information about each executed test case, transforming the distance calculations between a candidate and n executed test cases to an equivalent calculation involving the candidate and this tuple.

Inspired by their work, in this paper, we propose L′OP-ART, linear-time ART algorithm based on the *max-sum* criterion and a lightweight OMISS metric (to be presented in Section 3.1).

3.1. Lightweight OMISS

Since FSCS-ART using the *max-sum* criterion and the original OMISS metric, the sum of distances from a candidate c to the executed test case set $E = \{t_1, t_2, \ldots, t_n\}$, denoted *sum_dist*$(c, E)$, can be computed using formula $\sum_{i=1}^{n} TestcaseDistance(c, t_i)$. An implementation using this formula (which requires $O(n)$ time) would incur an $O(n^2)$ overhead when selecting test cases. To obtain a linear-time implementation of FSCS-ART with the *max-sum* criterion, we propose a lightweight OMISS and specifically designed data structures to collectively store information of the already executed test cases. According to Eq. (14), when calculating the distance ($sum\_TestcaseDistance$) between a test case and a set of test cases, it includes the distance of object set $sum\_TCobjDist$ and the distance between method invocation sequences $sum\_TCmSeqDist$. $sum\_TCmSeqDist$ includes the distance between two sequences in length (sum_LenD), in common methods (sum_MsD), and in the ordered sequence (sum_SD). $sum\_TCobjDist$ includes the behavior distance ($sum\_BehDist$) and the attribute distance ($sum\_AttDist$). $sum\_BehDist$ includes the sum of the difference in the number of methods (sum_BLenD) and the distance between their behavior sections (sum_BMsD). Lightweight OMISS can be obtained by transforming sum_MsD in $sum\_TCmSeqDist$ and sum_BMsD in $sum\_TCobjDist$.

Let $E = \{t_1, t_2, \ldots, t_n\}$ be the set of executed test cases and c be a candidate test case. In the original OMISS,

$$sum\_MsD(c.MINV, E.MINV) = \left(1 - \frac{|c.MINV \cap t_1.MINV|}{|c.MINV \cup t_1.MINV|}\right) + \left(1 - \frac{|c.MINV \cap t_2.MINV|}{|c.MINV \cup t_2.MINV|}\right) + \cdots + \left(1 - \frac{|c.MINV \cap t_n.MINV|}{|c.MINV \cup t_n.MINV|}\right)$$

In this paper, we extend the original OMISS to contain the concept of reduced method invocation sequences, obtained by removing the repetitive methods in the respective test cases. Let $\{ut_1, ut_2, \dots, ut_n\}$ be the reduced method invocation sequences of $\{t_1, t_2, \dots, t_n\}$, and uc be the reduced method invocation sequences of c. We define

$$\begin{aligned} & sum\_MsD(c.MINV, E.MINV) \\ & = (1 - \frac{|uc.MINV \cap ut_1.MINV|}{|uc.MINV \cup ut_1.MINV|}) \\ & + (1 - \frac{|uc.MINV \cap ut_2.MINV|}{|uc.MINV \cup ut_2.MINV|}) + \dots + \\ & (1 - \frac{|uc.MINV \cap ut_n.MINV|}{|uc.MINV \cup ut_n.MINV|}). \end{aligned}$$

Let $\langle x_1, x_2, \dots, x_n \rangle$ be a permutation of $\langle 1, 2, \dots, n \rangle$. Let $\{ua_1, ua_2, \dots, ua_{ub}\}$ $(1 \le ub \le n)$ be a subset of $\{1, 2, \dots, n\}$ such that $ua_1 < ua_2 < \dots < ua_{ub}$. Suppose further that

$$\begin{aligned} & |ut_{x_1}.MINV| = |ut_{x_2}.MINV| = \dots = |ut_{x_{ua_1}}.MINV| \\ & < |ut_{x_{ua_1+1}}.MINV| = |ut_{x_{ua_1+2}}.MINV| = \dots = |ut_{x_{ua_2}}.MINV| \\ & < \dots < |ut_{x_{ua_{ub-1}+1}}.MINV| = |ut_{x_{ua_{ub-1}+2}}.MINV| = \dots = \\ & |ut_{x_{ua_{ub}}}.MINV|. \end{aligned}$$

For any $r \in \{1, 2, \dots, ub\}$, our lightweight OMISS approximates each of $|uc.MINV \cup ut_{x_{ua_{r-1}+1}}.MINV|$, $|uc.MINV \cup ut_{x_{ua_{r-1}+2}}.MINV|$, …, $|uc.MINV \cup ut_{x_{ua_r}}.MINV|$ by their average value. In other words, we substitute each of them by $\frac{\sum_{j=ua_{r-1}+1}^{ua_r} |uc.MINV \cup ut_{x_j}.MINV|}{ua_r - ua_{r-1}}$, where ua_0 is set to 0. Thus:

$$\begin{aligned} & sum\_MsD(c.MINV, E.MINV) \\ & = s_0^0 - \sum_{r=1}^{ub} \sum_{i=ua_{r-1}}^{ua_r} \frac{|uc.MINV \cap ut_{x_i}.MINV|}{|uc.MINV \cup ut_{x_i}.MINV|} \\ & = s_0^0 - \sum_{r=1}^{ub} \sum_{i=ua_{r-1}}^{ua_r} \frac{|uc.MINV \cap ut_{x_i}.MINV|}{\frac{\sum_{j=ua_{r-1}+1}^{ua_r} |uc.MINV \cup ut_{x_j}.MINV|}{ua_r - ua_{r-1}}} \\ & = s_0^0 - \\ & \sum_{r=1}^{ub} (ua_r - ua_{r-1}) \frac{\sum_{i=ua_{r-1}+1}^{ua_r} |uc.MINV \cap ut_{x_i}.MINV|}{\sum_{j=ua_{r-1}+1}^{ua_r} |uc.MINV \cup ut_{x_j}.MINV|} \end{aligned} \tag{12}$$

where s_0^0 represents the number of test cases whose MINV is not null.

Let $Q = \{q_1, q_2, \dots, q_n\}$ be the set of existing objects and p be a candidate object. In the original OMISS,

$$\begin{aligned} & sum\_BMsD(p.M, Q.M) \\ & = (1 - \frac{|p.M \cap q_1.M|}{|p.M \cup q_1.M|}) + (1 - \frac{|p.M \cap q_2.M|}{|p.M \cup q_2.M|}) + \dots + (1 - \frac{|p.M \cap q_n.M|}{|p.M \cup q_n.M|}). \end{aligned}$$

Let $\langle x_1, x_2, \dots, x_n \rangle$ be a permutation of $\langle 1, 2, \dots, n \rangle$. Let $\{a_1, a_2, \dots, a_b\}$ $(1 \le b \le n)$ be a subset of $\{1, 2, \dots, n\}$ such that $a_1 < a_2 < \dots < a_b$. Suppose further that

$$\begin{aligned} & |q_{x_1}.M| = |q_{x_2}.M| = \dots = |q_{x_{a_1}}.M| \\ & < |q_{x_{a_1+1}}.M| = |q_{x_{a_1+2}}.M| = \dots = |q_{x_{a_2}}.M| \\ & < \dots < |q_{x_{a_{b-1}+1}}.M| = |q_{x_{a_{b-1}+2}}.M| = \dots = |q_{x_{a_b}}.M|. \end{aligned}$$

For any $r \in \{1, 2, \dots, b\}$, our lightweight OMISS approximates each of $|p.M \cup q_{x_{a_{r-1}+1}}.M|, |p.M \cup q_{x_{a_{r-1}+2}}.M|, \dots, |p.M \cup q_{x_{a_r}}.M|$ by their average value. In other words, we substitute each of them by $\frac{\sum_{j=a_{r-1}+1}^{a_r} |p.M \cup q_{x_j}.M|}{a_r - a_{r-1}}$, where a_0 is set to 0. Thus:

$$\begin{aligned} & sum\_BMsD(c.MINV, E.MINV) \\ & = m_0^0 - \sum_{r=1}^{b} (a_r - a_{r-1}) \frac{\sum_{i=a_{r-1}+1}^{a_r} |p.M \cap q_{x_i}.M|}{\sum_{j=a_{r-1}+1}^{a_r} |p.M \cup q_{x_j}.M|} \end{aligned} \tag{13}$$

where m_0^0 represents the number of objects in Q whose behavior sections are not null.

Throughout this paper, we will refer to the lightweight OMISS simply as "LW-OMISS".

3.2. Some important properties

In this section, we present the properties of the LW-OMISS for the sum of distances between a candidate and n executed test cases. We use specifically designed data structures to store the n executed test cases.

3.2.1. Distance between a candidate and a test set

Because a test case t comprises both $t.MINV$ and $t.OBJ$, in this paper, we also define a set E of executed test cases to be composed of both $E.MINV$ and $E.OBJ$, where $E.MINV$ is a method invocation sequence set, and $E.OBJ$ is a set of object sets. For example, consider $E = \{t_1, t_2\}$, where $t_1.OBJ = \{o_{11}, o_{12}, o_{13}\}$, $t_1.MINV = \{m_{11}, m_{12}\}$, $t_2.OBJ = \{o_{21}, o_{22}, o_{23}\}$, and $t_2.MINV = \{m_{21}, m_{22}, m_{23}\}$. If the distance between $t_1.OBJ$ and $t_2.OBJ$ is calculated when $t_1.OBJ$ and $t_2.OBJ$ are in the orders o_{11}, o_{12}, o_{13} and o_{21}, o_{22}, o_{23}, respectively, then $E.OBJ = \{\{o_{11}, o_{21}\}, \{o_{12}, o_{22}\}, \{o_{13}, o_{23}\}\}$, and $E.MINV = \{m_{11}, m_{12}, m_{21}, m_{22}, m_{23}\}$.

$$\begin{aligned} & sum\_TestcaseDistance(c,\ E) \\ & = sum\_TCmSeqDist(c.MINV,\ E.MINV) \\ & + sum\_TCobjDist(c.OBJ,\ E.OBJ). \end{aligned} \tag{14}$$

As shown in Eq. (14), the sum of distances between a candidate test case c and the executed test case set E is computed by adding up the sum of distances between $c.MINV$ and $E.MINV$ and the sum of distances between $c.OBJ$ and $E.OBJ$.

The object set distance is a measure of the difference between two sets of objects, and involves individually comparing all the objects in one set with those in the other, and summing these distances. When one of the sets has fewer objects than the other, the smaller set is expanded by adding *null* objects or *null* object sets — for example, if the number of objects in the smaller set is k_1, and the larger set has k_2 objects, $(k_2 - k_1)$ *null* objects or *null* object sets are added to the smaller set, giving a total of $k_2!$ permutations, where each permutation has k_2 comparisons of object and object set pairs. The sum of distances between $c.OBJ$ and $E.OBJ$ is the minimum sum of distances amongst all the possible object and object set pairs (as will be explained in Section 3.2.3):

$$\begin{aligned} & sum\_TCobjDist(c.OBJ,\ E.OBJ) \\ & = \begin{cases} \min(\bigcup_{i=1}^{k!} \sum_{j=1}^{k} sum\_ObjDist(c.obj_j,\ PL_i(E.OBJ,\ j))) \\ \quad \text{where}\ \ k = \max(k_{c.OBJ},\ k_{E.OBJ}), \\ \quad \text{if}\ \ \max(k_{c.OBJ},\ k_{E.OBJ}) > 0 \\ 0 \quad \text{otherwise} \end{cases} \end{aligned} \tag{15}$$

3.2.2. Distance for method invocation sequences

The sum of the method invocation sequence distances between a candidate c and an executed test set E involves summing three different parts:the differences in length;the number of common methods; and the distances between method invocation sequences as follows:

$$
\begin{aligned}
&sum\_TCmSeqDist(c.MINV,\ E.MINV)\\
&= sum\_LenD(c.MINV,\ E.MINV)\\
&+\ sum\_MsD(c.MINV,\ E.MINV) + sum\_SD(c.MINV,\ E.MINV)
\end{aligned}
$$

where

$$
\begin{aligned}
&sum\_LenD(c.MINV,\ E.MINV)\\
&= \sum_{i=1}^{n} LenD(c.MINV,\ t_i.MINV)\\
&sum\_MsD(c.MINV,\ E.MINV)\\
&= \sum_{i=1}^{n} MsD(c.MINV,\ t_i.MINV)
\end{aligned}
$$

$$
sum\_SD(c.MINV,\ E.MINV) = \sum_{i=1}^{n} SD(c.MINV,\ t_i.MINV) \tag{16}
$$

As explained in Section 3.1, the executed test case information is stored in specifically designed data structures (tuples), allowing method invocation sequence calculations to be independent of object set calculations. Given an executed test set E, the information associated with its method invocation sequences ($E.MINV$) is recorded in the following structured tuple **S**.

Definition of Global Method Invocation Structure S

$$
\begin{aligned}
\mathbf{S} = (s_0^0,\ s_{01}^0,\ s_{01}^1, s_1^0, s_1^1,\ s_1^2,\ \ldots,\ &s_1^h,\\
&s_{1,1}^1,\ s_{1,1}^2,\ \ldots,\ s_{1,1}^h,\\
s_{02}^0,\ s_{02}^1,\ s_2^0,\ s_2^1,\ s_2^2, \ldots,\ &s_2^h,\\
&s_{2,1}^1,\ s_{2,1}^2,\ \ldots,\ s_{2,1}^h,\\
&s_{2,2}^1,\ s_{2,2}^2,\ \ldots,\ s_{2,2}^h,\\
\ldots,&\\
\ldots,&\\
\ldots,&\\
s_{0g}^0,\ s_{0g}^1,\ s_g^0,\ s_g^1,\ s_g^2, \ldots,\ &s_g^h,\\
&s_{g,1}^1,\ s_{g,1}^2,\ \ldots,\ s_{g,1}^h,\\
&s_{g,2}^1,\ s_{g,2}^2,\ \ldots,\ s_{g,2}^h,\\
&\ldots,\\
&s_{g,g}^1,\ s_{g,g}^2,\ \ldots,\ s_{g,g}^h),
\end{aligned}
$$

where

- g is the maximum number of methods in the method invocation sequence associated with all elements of E;
- h is the number of different methods in the program under test;
- s_0^0 represents the number of test cases whose method invocation sequences are not null;
- s_{0i}^0 denotes the number of test cases in which the length of the method invocation sequence is shorter than or equal to i;
- s_{0i}^1 denotes the sum of the lengths of method invocation sequences that are shorter than or equal to the length i;
- s_i^x denotes the number of test cases with the length of method invocation sequence i, and with the xth method appearing in this sequence;
- s_i^0 denotes the number of test cases in which the length of the method invocation sequence is i;
- $s_{i,j}^x$ denotes the number of test cases in which the length of the method invocation sequence is i and the jth method in this sequence has label x.

As a reminder, if a method appears twice, it is counted twice in s_{0i}^0, s_{0i}^1, and $s_{i,j}^x$ (which are used in *sum_LenD*, *sum_SD*); but is only counted once in s_i^x and s_i^0 (which are used in *sum_MsD*) ($1 \le i,\ j \le g,\ 1 \le x \le h$).

Let $E = \{t_1, t_2, \ldots, t_n\}$ be a set of executed test cases. For any $t_r \in E$, we record the information associated with its method invocation sequence $t_r.MINV$ in a structured tuple $\mathbf{S}_r$, whose format is the same as that of **S**. An element s of $\mathbf{S}_r$ is referred as $\mathbf{S}_r.s$. The number of different methods in the program under test (denoted by h) is the same for both $\mathbf{S}_r$ and **S**. The number of methods invoked (denoted by g), as captured by $t_r.MINV$ in $\mathbf{S}_r$, is shorter than or equal to that captured by $E.MINV$ in **S**. The values of other elements of $\mathbf{S}_r$ are computed according to t_r. If the context is obvious and there is no confusion, $\mathbf{S}.s_0^x$, $\mathbf{S}.s_i^x$, $\mathbf{S}.s_{0j}^x$ and $\mathbf{S}.s_{i,j}^x$ are abbreviated to s_0^x, s_i^x, s_{0j}^x and $s_{i,j}^x$, respectively.

To illustrate this, consider again the PET_STORE system in Fig. 3. It has eight methods, i.e., $h = 8$. For ease of presentation, we refer to methods “Grow()”, “Fly()”, “Expand()”, “Bark()”, “Follow()”, “Speak()”, “Learn()”, and “Work()” with labels 1, 2, 3, 4, 5, 6, 7, and 8, respectively. Assume that E has three test cases t_1, t_2, and t_3 (as specified in Table 1). Because the longest sequence of method calls is 4 in Table 1, we have $g = 4$, and **S** is as follows:

S = (

$$
\begin{aligned}
\mathbf{S} =(\ &3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0,\\
&2, 6, 3, 3, 2, 2, 1, 1, 0, 0, 0,\\
&2, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 1, 1, 0, 0, 0, 0,\\
&0, 1, 0, 0, 1, 0, 0, 0,\\
&3, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0,\\
&1, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 1, 0, 0, 0, 0, 0,\\
&0, 0, 1, 0, 0, 0, 0, 0,\\
&0, 1, 0, 0, 0, 0, 0, 0\ ).
\end{aligned}
$$

Since $\mathbf{S}_3$ only stores the information of $t_3.MINV$, which has three methods, the relevant g is 3 and $\mathbf{S}_3$ is as follows:

$$
\begin{aligned}
\mathbf{S}_3 =(&1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 0, 0, 0, 0, 0,\\
&1, 3, 1, 1, 0, 0, 1, 1, 0, 0, 0,\\
&1, 0, 0, 0, 0, 0, 0, 0,\\
&0, 0, 0, 1, 0, 0, 0, 0,\\
&0, 0, 0, 0, 1, 0, 0, 0).
\end{aligned}
$$

From the definition of $\mathbf{S}_r$ ($r \in \{1, 2, \ldots, n\}$) and the $\mathbf{S}_3$ example above, we can see that, apart from $\mathbf{S}_r.s_{0i}^1$ (which may be equal to $k_{t_r.MINV}$ or 0, $\mathbf{S}_r.s_0^0$), $\mathbf{S}_r.s_{0i}^0$, $\mathbf{S}_r.s_i^x$, and $\mathbf{S}_r.s_{i,j}^x$ are all either 1 or 0 ($i \in \{1, 2, \ldots, g\}$, $j \in \{1, 2, \ldots, g\}$, $j \le i$, $x \in \{1, 2, \ldots, h\}$).

The following notation, definitions, and assumptions are used in the subsequent theorem and lemmas:

(a) Given any set of test cases $\{t_1, t_2, \ldots, t_n\}$, there exists a permutation $\langle x_1, x_2, \ldots, x_n\rangle$ of $\langle 1, 2, \ldots, n\rangle$ such that $k_{ut_{x_1}} \le k_{ut_{x_2}} \le \cdots \le k_{ut_{x_n}}$, where $k_{ut_{x_i}}$ is the length of $ut_{x_i}.MINV$. As a result, there exists some $ub \in \{1, 2, \ldots, n\}$ and a set $\{ua_1, ua_2, \ldots, ua_{ub}\} \subseteq \{1, 2, \ldots, n\}$ such that $ua_1 < ua_2 < \cdots < ua_{ub} = n$ and $k_{ut_{x_1}} = k_{ut_{x_2}} = \cdots = k_{ut_{x_{ua_1}}} < k_{ut_{x_{ua_1+1}}} = \cdots = k_{ut_{x_{ua_2}}} < \ldots < k_{ut_{x_{ua_{ub-1}+1}}} = \cdots = k_{ut_{x_{ua_{ub}}}}$. For any $r \in \{1, 2, \ldots, ub\}$, because

$ut_{x_{ua_{r-1}+1}}$, $ut_{x_{ua_{r-1}+2}}$, …, $ut_{x_{ua_r}}$ are of the same length, we will devote all of $k_{ut_{x_{ua_{r-1}+1}}}$, $k_{ut_{x_{ua_{r-1}+2}}}$, …, $k_{ut_{x_{ua_r}}}$ by k_{ua_r} when there is no ambiguity.

(b) For any test case $t_i \in \{t_1, t_2, \dots, t_n\}$, and method $j \in \{1, 2, \dots, k_{t_i}\}$, we define $d_j(t_i) = 1$ if $c.MINV[j] \neq t_i.MINV[j]$; and $d_j(t_i) = 0$ otherwise. Defining $\overline{d}_j(t_i) = 1 - d_j(t_i)$, we have: $\overline{d}_j(t_i) = 1$ if $d_j(t_i) = 0$; and $\overline{d}_j(t_i) = 0$ otherwise.

(c) Given any set of test cases $\{t_1, t_2, \dots, t_n\}$, let $\langle v_1, v_2, \dots, v_n\rangle$ be a permutation of $\langle 1, 2, \dots, n\rangle$ such that $k_{t_{v_1}} \le k_{t_{v_2}} \le \dots \le k_{t_{v_n}}$, where $k_{t_{v_i}}$ is the length of $t_{v_i}.MINV$.[2] As a consequence, there exists some $b \in \{1, 2, \dots, n\}$ and a set $\{a_1, a_2, \dots, a_b\} \subseteq \{1, 2, \dots, n\}$ such that $a_1 < a_2 < \dots < a_b = n$ and

$$\begin{aligned} & k_{t_{v_1}} = k_{t_{v_2}} = \ldots = k_{t_{v_{a_1}}} \\ & \quad < k_{t_{v_{a_1+1}}} = k_{t_{v_{a_1+2}}} = \ldots = k_{t_{v_{a_2}}} \\ & \quad < \ldots < k_{t_{v_{a_{b-1}+1}}} = k_{t_{v_{a_{b-1}+2}}} = \ldots = k_{t_{v_{a_b}}}. \end{aligned}$$

For any $r \in \{1, 2, \dots, b\}$, because $t_{v_{a_{r-1}+1}}$, $t_{v_{a_{r-1}+2}}$, …, $t_{v_{a_r}}$ are of the same length, we will represent all of $k_{t_{v_{a_{r-1}+1}}}$, $k_{t_{v_{a_{r-1}+2}}}$, …, $k_{t_{v_{a_r}}}$ by k_{a_r} when there is no ambiguity.

Given any candidate test case c, let τ $(0 \le \tau \le n)$ denote the number of executed test cases in which the lengths of method invocation sequences are shorter than or equal to k_c. We have $k_c \le k_{t_{v_1}}$ if $\tau = 0$; $k_{t_{v_n}} \le k_c$ if $\tau = n$; and $k_{t_{v_1}} \le k_{t_{v_2}} \le \dots \le k_{t_{v_\tau}} \le k_c < k_{t_{v_{\tau+1}}}$ otherwise.

We further define $\tau\tau = 0$ if $k_c < k_{t_{v_{a_1}}}$; $\tau\tau = b$ if $k_{t_{v_{a_b}}} \le k_c$; and $\tau\tau = r$ if $k_{t_{v_{a_r}}} \le k_c < k_{t_{v_{a_{r+1}}}}$ for some $r \in \{1, 2, \dots, b-1\}$. Then, we have $a_{\tau\tau} = 0$ if $\tau\tau = 0$; and $a_{\tau\tau} = \tau$ otherwise.

(d) Given any objects $\{q_1, q_2, \dots, q_n\}$, let $\langle v_1, v_2, \dots, v_n\rangle$ be a permutation of $\langle 1, 2, \dots, n\rangle$ such that $k_{q_{v_1}.M} \le k_{q_{v_2}.M} \le \dots \le k_{q_{v_n}.M}$, where $k_{q_{v_i}.M}$ is the length of $q_{v_i}.M$. As a consequence, there exists some $b \in \{1, 2, \dots, n\}$ and a set $\{a_1, a_2, \dots, a_b\} \subseteq \{1, 2, \dots, n\}$ such that $a_1 < a_2 < \dots < a_b = n$ and

$$\begin{aligned} & k_{q_{v_1}.M} = k_{q_{v_2}.M} = \ldots = k_{q_{v_{a_1}}.M} \\ & \quad < k_{q_{v_{a_1+1}}.M} = k_{q_{v_{a_1+2}}.M} = \ldots = k_{q_{v_{a_2}}.M} \\ & \quad < \ldots < k_{q_{v_{a_{b-1}+1}}.M} = k_{q_{v_{a_{b-1}+2}}.M} = \ldots = k_{q_{v_{a_b}}.M} \end{aligned}$$

For any $r \in \{1, 2, \dots, b\}$, because $q_{v_{a_{r-1}+1}}.M$, $q_{v_{a_{r-1}+2}}.M$, …, $q_{v_{a_r}}.M$ are of the same length, we will denote all of $k_{q_{v_{a_{r-1}+1}}.M}$, $k_{q_{v_{a_{r-1}+2}}.M}$, …, $k_{q_{v_{a_r}}.M}$ by k_{a_r} when there is no ambiguity.

Let τ $(0 \le \tau \le n)$ denote the number of objects whose behavior sections have $k_{p.M}$ or fewer methods. We have $k_{p.M} < k_{q_{v_1}.M}$ if $\tau = 0$; $k_{q_{v_n}.M} \le k_{p.M}$ if $\tau = n$; and $k_{q_{v_1}.M} \le k_{q_{v_2}.M} \le \dots \le k_{q_{v_\tau}.M} \le k_{p.M} < k_{q_{v_{\tau+1}}.M}$ otherwise.

We further define $\tau\tau = 0$ if $k_{p.M} < k_{q_{v_{a_1}}.M}$; $\tau\tau = b$ if $k_{q_{v_{a_b}}.M} \le k_{p.M}$; and $\tau\tau = r$ if $k_{q_{v_{a_r}}.M} \le k_{p.M} < k_{q_{v_{a_{r+1}}}.M}$ for some $r \in \{1, 2, \dots, b-1\}$. Then, we have $a_{\tau\tau} = 0$ if $\tau\tau = 0$; and $a_{\tau\tau} = \tau$ otherwise.

(e) Given any objects $\{q_1, q_2, \dots, q_n\}$, let $\langle x_1, x_2, \dots, x_n\rangle$ be a permutation of $\langle 1, 2, \dots, n\rangle$ such that $k_{uq_{x_1}.M} \le k_{uq_{x_2}.M} \le \dots \le k_{uq_{x_n}.M}$, where $k_{uq_{x_i}.M}$ is the length of $uq_{x_i}.M$. As a result, there exists some $ub \in \{1, 2, \dots, n\}$ and a set $\{ua_1, ua_2, \dots, ua_{ub}\} \subseteq \{1, 2, \dots, n\}$ such that $ua_1 < ua_2 < \dots < ua_{ub} = n$ and

$$\begin{aligned} & k_{uq_{x_1}.M} = k_{uq_{x_2}.M} = \ldots = k_{uq_{x_{ua_1}}.M} \\ & \quad < k_{uq_{x_{ua_1+1}}.M} = k_{uq_{x_{ua_1+2}}.M} = \ldots = k_{uq_{x_{ua_2}}.M} \\ & \quad < \ldots < k_{uq_{x_{ua_{ub-1}+1}}.M} = k_{uq_{x_{ua_{ub-1}+2}}.M} = \ldots = k_{uq_{x_{ua_{ub}}}.M} \end{aligned}$$

For any $r \in \{1, 2, \dots, ub\}$, because $uq_{x_{ua_{r-1}+1}}.M$, $uq_{x_{ua_{r-1}+2}}.M$, …, $uq_{x_{ua_r}}.M$ are of the same length, we will denote all of $k_{uq_{x_{ua_{r-1}+1}}.M}$, $k_{uq_{x_{ua_{r-1}+2}}.M}$, …, $k_{uq_{x_{ua_r}}.M}$ by k_{ua_r} when there is no ambiguity.

(f) Given any objects $\{q_1, q_2, \dots, q_n\}$, let $\langle v_1, v_2, \dots, v_n\rangle$ be a permutation of $\langle 1, 2, \dots, n\rangle$ such that $k_{q_{v_1}.A.N} \le k_{q_{v_2}.A.N} \le \dots \le k_{q_{v_n}.A.N}$, where $k_{q_{v_i}.A.N}$ is the length of $q_{v_i}.A.N$. As a consequence, there exists some $b \in \{1, 2, \dots, n\}$ and a set $\{a_1, a_2, \dots, a_b\} \subseteq \{1, 2, \dots, n\}$ such that $a_1 < a_2 < \dots < a_b = n$ and

$$\begin{aligned} & k_{q_{v_1}.A.N} = k_{q_{v_2}.A.N} = \ldots = k_{q_{v_{a_1}}.A.N} \\ & \quad < k_{q_{v_{a_1+1}}.A.N} = k_{q_{v_{a_1+2}}.A.N} = \ldots = k_{q_{v_{a_2}}.A.N} \\ & \quad < \ldots < k_{q_{v_{a_{b-1}+1}}.A.N} = k_{q_{v_{a_{b-1}+2}}.A.N} = \ldots = k_{q_{v_{a_b}}.A.N} \end{aligned}$$

For any $r \in \{1, 2, \dots, b\}$, because $q_{v_{a_{r-1}+1}}.A.N$, $q_{v_{a_{r-1}+2}}.A.N$, …, $q_{v_{a_r}}.A.N$ are of the same length, we will denote all of $k_{q_{v_{a_{r-1}+1}}.A.N}$, $k_{q_{v_{a_{r-1}+2}}.A.N}$, …, $k_{q_{v_{a_r}}.A.N}$ by k_{a_r} when there is no ambiguity.

Let τ $(0 \le \tau \le n)$ denote the number of objects whose non-reference attribute sections have $k_{p.A.N}$ or fewer attributes. We have $k_{p.A.N} < k_{q_{v_1}.A.N}$ if $\tau = 0$; $k_{q_{v_n}.A.N} \le k_{p.A.N}$ if $\tau = n$; and $k_{q_{v_1}.A.N} \le k_{q_{v_2}.A.N} \le \dots \le k_{q_{v_\tau}.A.N} \le k_{p.A.N} < k_{q_{v_{\tau+1}}.A.N}$ otherwise.

We further define $\tau\tau = 0$ if $k_{p.A.N} < k_{q_{v_{a_1}}.A.N}$; $\tau\tau = b$ if $k_{q_{v_{a_b}}.A.N} \le k_{p.A.N}$; and $\tau\tau = r$ if $k_{q_{v_{a_r}}.A.N} \le k_{p.A.N} < k_{q_{v_{a_{r+1}}}.A.N}$ for some $r \in \{1, 2, \dots, b-1\}$. Then, we have $a_{\tau\tau} = 0$ if $\tau\tau = 0$; and $a_{\tau\tau} = \tau$ otherwise.

We have three major results, stated as Theorems 1, 2, and 3: Theorem 1 is related to the method invocation sequence between a candidate test case (c) and a set of executed test cases (E); Theorem 2 is related to the object distance between c and E; and Theorem 3 is related to the non-reference attribute distance between c and E.

We will present the main contributions of the proposed metric in the following paragraphs. Before considering Theorem 1, we need the following lemmas:

Lemma 1. Given any candidate test case c, and any permutation $\langle v_1, v_2, \dots, v_n\rangle$ of $\langle 1, 2, \dots, n\rangle$, we have $s^0_{0k_c} = \tau$, $s^1_{0k_c} = \sum_{i=1}^{\tau} k_{t_{v_i}}$, $s^1_{0g} = \sum_{i=1}^{n} k_{t_{v_i}}$, $s^0_{0g} = n$, and $\sum_{i=\tau+1}^{n} k_{t_{v_i}} = s^1_{0g} - s^1_{0k_c}$.

Lemma 2. Let c be any candidate test case and uc be the corresponding reduced candidate test case. Let t_{x_i} be any executed test case and ut_{x_i} be the corresponding reduced candidate test case. $|c.MINV \cap t_{x_i}.MINV| = |uc.MINV \cap t_{x_i}.MINV| = |c.MINV \cap ut_{x_i}.MINV| = |uc.MINV \cap ut_{x_i}.MINV|$, and $|c.MINV \cup t_{x_i}.MINV| = |uc.MINV \cup t_{x_i}.MINV| = |c.MINV \cup ut_{x_i}.MINV| = |uc.MINV \cup ut_{x_i}.MINV|$.

Lemma 3. Given any candidate test case c, and any permutation $\langle x_1, x_2, \dots, x_n\rangle$ of $\langle 1, 2, \dots, n\rangle$, we have $|uc.MINV \cap ut_{x_i}.MINV| = \sum_{l=1}^{k_{uc}} \mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ut_{x_i}}}$.

Lemma 4. For any $r \in \{1, 2, \dots, ub\}$ and $l \in \{1, 2, \dots, k_{ua_r}\}$, where k_{ua_r} is specified in Definition (a), we have $\mathbf{S}.s^{uc.MINV[l]}_{k_{ua_r}} = \sum_{j=ua_{r-1}+1}^{ua_r} \mathbf{S}_{x_j}.s^{uc.MINV[l]}_{k_{ua_r}}$.

Lemma 5. Given any candidate test case c, and any $r \in \{1, 2, \dots, ub\}$, we have $\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cap t_{x_i}.MINV| = \sum_{l=1}^{k_{uc}} s^{uc.MINV[l]}_{k_{ua_r}}$, where k_{ua_r} is specified in Definition (a).

Lemma 6. Given any candidate test case c, and any $r \in \{1, 2, \dots, ub\}$, we have $\frac{\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cup t_{v_i}.MINV|}{ua_r - ua_{r-1}} = k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s^{uc.MINV[l]}_{k_{ua_r}}}{ua_r - ua_{j-1}}$, where k_{ua_r} is specified in Definition (a).

Lemma 7. Given any candidate test case c, and any $ub \in \{1, 2, \dots, n\}$, we have

[2] We will use $k_{t_{v_i}}$ in place of the original $k_{t_{v_i}.MINV}$ because of the need for a shorthand notation in the mathematical statements and proofs.

$\sum_{r=1}^{ub} \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{k_{uc}+k_{ua_r}-\frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{s_{k_{ua_r}}^{0}}} =$ $\sum_{r=1}^{g} \frac{\sum_{l=1}^{k_{uc}} s_{r}^{uc.MINV[l]}}{k_{uc}+r-\frac{\sum_{l=1}^{k_{uc}} s_{r}^{uc.MINV[l]}}{\max(s_r^0,1)}}$, where ua_r and k_{ua_r} are specified in Definition (a).

Lemma 8. Given any candidate test case c and any permutation $\langle v_1, v_2, \dots, v_n\rangle$ of $\langle 1, 2, \dots, n\rangle$ as specified in Definition (c), we have $\sum_{l=1}^{k_{t_{v_i}}} \overline{d}_l(t_{v_i}) = \sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}$, and $\sum_{l=1}^{k_c} \overline{d}_l(t_{v_i}) =$ $\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}$.

Lemma 9. Given any candidate test case c, and any $l \in \{1, 2, \dots, k_c\}$, we have $\mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]} =$ $\sum_{i=1}^{n} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]} = \sum_{j=a_{r-1}+1}^{a_r} \mathbf{S}_{v_j}.s_{k_{a_r},l}^{c.MINV[l]}$, where v_{a_r} and k_{a_r} are specified in Definition (c).

Lemma 10. Let c be any candidate test case.

(a) If $k_c < k_{a_1}$, $\sum_{r=1}^{k_c} \frac{\sum_{l=1}^{r} s_{r,l}^{c.MINV[l]}}{r} = 0$ and $\sum_{r=1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c} =$ $\sum_{r=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{r,l}^{c.MINV[l]}}{k_c}$;

(b) If $k_{a_b} \le k_c$, $\sum_{r=1}^{b} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}} = \sum_{r=1}^{k_c} \frac{\sum_{l=1}^{r} s_{r,l}^{c.MINV[l]}}{r}$ and $\sum_{r=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{r,l}^{c.MINV[l]}}{k_c} = 0$;

(c) If $k_{a_{\tau\tau}} \le k_c < k_{a_{\tau\tau+1}}$, $\sum_{r=1}^{\tau\tau} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}} = \sum_{r=1}^{k_c} \frac{\sum_{l=1}^{r} s_{r,l}^{c.MINV[l]}}{r}$ and $\sum_{r=\tau\tau+1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c} = \sum_{r=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{r,l}^{c.MINV[l]}}{k_c}$ where k_{a_r} $(r \in 1, 2, \dots, b)$ and $\tau\tau$ are specified in Definition (c).

We are now ready to state Theorem 1. The proofs of the theorem and its lemmas (1-10) can be found in Appendices A.1–A.11.

Theorem 1. *Consider an executed test set* $E = \{t_1, t_2, \dots, t_n\}$, *the associated global method invocation structure* $\mathbf{S}$, *and a candidate test case* c. *The three types of method invocation sequence distances have the following properties:*

$$sum\_LenD(c.MINV,\ E.MINV) = (2 * s_{0k_c}^{0} - s_{0g}^{0}) * k_c + s_{0g}^{1} - 2 * s_{0k_c}^{1} \tag{16a}$$

$$sum\_MsD(c.MINV,\ E.MINV) = \begin{cases} s_0^0 - \sum_{i=1}^{g} \frac{\sum_{l=1}^{k_{uc}} s_i^{uc.MINV[l]}}{k_{uc}+i-\frac{\sum_{l=1}^{k_{uc}} s_i^{uc.MINV[l]}}{max(s_i^0,1)}} & \textit{if } k_c > 0 \\ 0 & \textit{otherwise} \end{cases} \tag{16b}$$

$$sum\_SD(c.MINV,\ E.MINV) = \begin{cases} s_0^0 - \left(\sum_{l=1}^{k_c} \frac{\sum_{i=1}^{l} s_{l,\ i}^{c.MINV[i]}}{l} + \sum_{l=k_c+1}^{g} \frac{\sum_{i=1}^{k_c} s_{l,\ i}^{c.MINV[i]}}{k_c}\right) & \textit{if } k_c > 0 \\ 0 & \textit{otherwise} \end{cases} \tag{16c}$$

To illustrate this, consider again the three test cases t_1, t_2, and t_3 in Table 1, which have eight different methods (Fig. 3), and hence $h = 8$. For ease of presentation, we label the methods as $Speak = 1$; $Learn = 2$; $Work = 3$; $Grow = 4$; $Fly = 5$; $Expand = 6$; $Bark = 7$; and $Follow = 8$. Furthermore, $g = 4$, $R(t_1.MINV) = (4,\ 6,\ 5)$, $R(t_2.MINV) = (4,\ 6,\ 6,\ 5)$, and $R(t_3.MINV) = (4,\ 7,\ 8)$. Suppose that $E = \{t_1,\ t_2\}$, and t_3 is a candidate test case. We have: $s_{03}^0 = 1$, $s_{03}^1 = 3$, $s_{04}^0 = 2$, $s_{04}^1 = 7$, $s_3^0 = 2$, $s_3^4 = 2$, $s_3^5 = 2$, $s_3^6 = 2$, $s_{3,1}^4 = 1$, $s_{3,2}^6 = 1$, $s_{3,3}^5 = 1$, $s_{4,1}^4 = 1$, $s_{4,2}^6 = 1$, $s_{4,3}^6 = 1$, $s_{4,4}^5 = 1$, and all other values in $\mathbf{S}$ equal to 0. According to Theorem 1, we have:

$$sum\_LenD(t_3.MINV,\ E.MINV) = (2 * s_{03}^0 - s_{04}^0) * 3 + s_{04}^1 - 2 * s_{03}^1 = (2 * 1 - 2) * 3 + 7 - 2 * 3 = 1.$$

$$sum\_MsD(t_3.MINV,\ E.MINV) = 2 - (0 + 0 + \frac{2}{3} + 0) = 1.33.$$

$$sum\_SD(t_3.MINV,\ E.MINV) = 2 - (0 + 0 + \frac{1+0+0}{3} + \frac{1+0+0}{3}) = 1.33.$$

If Eqs. (16a), (16b), and (16c) are used to compute $sum\_LenD(c.MINV,\ E.MINV)$, $sum\_MsD(c.MINV,\ E.MINV)$, and $sum\_SD(c.MINV,\ E.MINV)$, respectively, then, because the lengths of the tuples in $\mathbf{S}$, $\mathbf{S}_r$ $(r = 1, 2, \dots, n)$, and $R(c.MINV)$ are independent of the test set size, the overheads for distance computations for method invocation sequences can be reduced to linear time.

3.2.3. Distance for objects

The calculation of the sum of distances between an object p and a set of objects Q involves calculating the sum of the distances between p's behavior section and that of each object in Q; and the sum of the distances between p's attribute section and that of each of the objects in Q. Accordingly, we have

$$sum\_ObjDist(p,\ Q) = \begin{cases} sum\_BehDist(p.M,\ Q.M)\ +\ sum\_AttDist(p.A,\ Q.A) & \text{if } p \text{ is not null} \\ 2 * num & \text{if } p \text{ is null,} \end{cases} \tag{17}$$

where num is the number of objects in Q that are not null.

Similar to the method invocation sequence distance (Section 3.2.2), the sum of distances between the behavior sections of object p and a set of objects Q also involves the difference in the length and the number of common methods.

$$sum\_BehDist(p.M,\ Q.M) = sum\_BLenD(p.M,\ Q.M) + sum\_BMsD(p.M,\ Q.M) \tag{18}$$

Given a set of objects Q $(\{q_1, q_2, \dots, q_n\})$, the information associated with their behavior sections $(Q.M{=}\{q_1.M, q_2.M, \dots, q_n.M\})$ can be recorded in the structured tuple $\mathbf{M}$.

Definition of Behavior Structure M of A Object Set

$$\begin{aligned}\mathbf{M} = (&m_0^0,\ m_{01}^0,\ m_{01}^1,\ m_1^0,\ m_1^1,\ m_1^2,\ \dots,\ m_1^h,\\ &m_{02}^0,\ m_{02}^1,\ m_2^0,\ m_2^1,\ m_2^2,\ \dots,\ m_2^h,\\ &\dots,\\ &m_{0c}^0,\ m_{0c}^1,\ m_c^0,\ m_c^1,\ m_c^2,\ \dots,\ m_c^h),\end{aligned}$$

where c is the maximum number of methods for an object's associated class; h denotes the total number of different methods in the program under test; m_0^0 represents the number of objects in Q whose behavior sections are not null; m_{0i}^0 denotes the number of objects with i methods or fewer; m_{0i}^1 denotes the sum of the lengths of such method sections; m_i^x denotes the number of objects that have i as the length of their behavior sections and method x appearing in their behavior sections; and m_i^0 denotes the number of objects with i methods $(1 \le i \le c; 1 \le x \le h)$.

Before we consider Theorem 2, we need the following lemmas:

Lemma 11. $m^0_{0k_{p.M}} = \tau$, $m^1_{0k_{p.M}} = \sum_{i=1}^{\tau} k_{q_{v_i}.M}$, $m^1_{0c} = \sum_{i=1}^{n} k_{q_{v_i}.M}$, $m^0_{0c} = n$, and $\sum_{i=\tau+1}^{n} k_{q_{v_i}.M} = m^1_{0c} - m^1_{0k_{p.M}}$, where τ and v_i are defined in Definition (d) of Section 3.2.2.

Lemma 12. $|p.M \cap q_{x_i}.M| = |up.M \cap q_{x_i}.M| = |p.M \cap uq_{x_i}.M| = |up.M \cap uq_{x_i}.M|$, and $|p.M \cup q_{x_i}.M| = |up.M \cup q_{x_i}.M| = |p.M \cup uq_{x_i}.M| = |up.M \cup uq_{x_i}.M|$.

Lemma 13. $|up.M \cap uq_{x_i}.M| = \sum_{l=1}^{k_{up.M}} \mathbf{M}_{x_i}.m^{up.M[l]}_{k_{uq_{x_i}.M}}$, where $x_i \in \{x_1, x_2, \ldots, x_n\}$ with $\langle x_1, x_2, \ldots, x_n\rangle$ as defined in Definition (e) of Section 3.2.2.

Lemma 14. For any $r \in \{1, 2, \ldots, ub\}$ and $l \in \{1, 2, \ldots, k_{ua_r}\}$, where k_{ua_r} and ub are specified in Definition (e) of Section 3.2.2, we have $\mathbf{M}.m^{up.M[l]}_{k_{ua_r}} = \sum_{i=ua_{r-1}+1}^{ua_r} \mathbf{M}_{x_i}.m^{up.M[l]}_{k_{ua_r}}$.

Lemma 15. For any j $(1 \le j \le ub)$, we have $\sum_{i=ua_{j-1}+1}^{ua_j} |p.M \cap q_{x_i}.M| = \sum_{l=1}^{k_{up.M}} m^{up.M[l]}_{k_{ua_j}}$, where k_{ua_j} and ub are defined in Definition (e) of Section 3.2.2.

Lemma 16. For any j $(1 \le j \le ub)$, we have $\frac{\sum_{i=ua_{j-1}+1}^{ua_j} |p.M \cup q_{x_i}.M|}{ua_j - ua_{j-1}} = k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m^{up.M[i]}_{k_{ua_j}}}{ua_j - ua_{j-1}}$, where k_{ua_j} and ub are defined in Definition (e) of Section 3.2.2.

Lemma 17. $\sum_{j=1}^{ub} \frac{\sum_{i=1}^{k_{up.M}} m^{up.M[i]}_{k_{ua_j}}}{k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m^{up.M[i]}_{k_{ua_j}}}{m^0_{k_{ua_j}}}} = \sum_{j=1}^{c} \frac{\sum_{i=1}^{k_{up.M}} m^{up.M[i]}_j}{k_{up.M} + j - \frac{\sum_{i=1}^{k_{up.M}} m^{up.M[i]}_j}{\max(m^0_j, 1)}}$,

where k_{ua_j} is defined in Definition (e) of Section 3.2.2, and c is the maximum number of methods for an object's associated class.

We are now ready to state Theorem 2. The proofs of the theorem and its lemmas (11-17) can be found in Appendices A.12–A.18.

Theorem 2. *Given a set Q of n objects, its associated $\mathbf{M}$, and a candidate object p, if $up.M$ represents the reduced behavior section of p (created by removing the repeated methods in $p.M$), $k_{p.M}$ and $k_{up.M}$ represent the lengths of $p.M$ and $up.M$ respectively, and $up.M[j]$ represents the jth method in $up.M$ $(1 \le j \le k_{up.M})$, we have:*

$$
\begin{aligned}
&sum\_BLenD(p.M,\ Q.M) \\
&= (2 * m^0_{0k_{p.M}} - m^0_{0c}) * k_{p.M} + m^1_{0c} - 2 * m^1_{0k_{p.M}}
\end{aligned}
\tag{18a}
$$

$$
sum\_BMsD(p.M,\ Q.M) = \begin{cases} m^0_0 - \sum_{i=1}^{c} \frac{\sum_{j=1}^{k_{up.M}} m^{up.M[j]}_i}{k_{up.M} + i - \frac{\sum_{j=1}^{k_{up.M}} m^{up.M[j]}_i}{max(m^0_i, 1)}} & \textit{if } p.M \textit{ is not null} \\ 0 & \textit{otherwise} \end{cases}
\tag{18b}
$$

Because the attribute distance is equal to the sum of reference and non-reference attribute distances (which are independent of each other), the sum of attribute distances between p and Q is:

$$
\begin{aligned}
&sum\_AttDist(p.A,\ Q.A) \\
&\quad = sum\_nonRefDist(p.A.N,\ Q.A.N) \\
&\quad + sum\_refDist(p.A.R,\ Q.A.R)
\end{aligned}
\tag{19}
$$

Because reference attributes can be considered as objects, we can use *sum_TCobjDist* (Eq. (15)) to calculate $sum\_refDist(p.A.R, Q.A.R)$ as follows:

$$
\begin{aligned}
&sum\_refDist(p.A.R,\ Q.A.R) \\
&= \begin{cases} sum\_TCobjDist(p.A.R,\ Q.A.R)\ *\ \alpha \\ \quad \text{where } \alpha = \frac{1}{2}, \text{if } \textit{either } p.A.R \text{ or } Q.A.R \text{ is not null} \\ 0 \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}
\end{aligned}
\tag{20}
$$

Eq. (20) serves the same purpose as Ciupa et al.'s recursive distance formula (Ciupa et al., 2006). Similar to Ciupa et al. (2006), we also assigned $1/2$ to α to attenuate the impact of deeper recursive calculations.

For non-reference variable (*NRefV*) attributes, the distance calculations calculated based on their data types and value differences:

$$
\begin{aligned}
&sum\_nonRefDist(p.A.N,\ Q.A.N) \\
&= (sum\_typeDist(p.A.N,\ Q.A.N), \\
&\quad sum\_secDist(p.A.N,\ Q.A.N))
\end{aligned}
\tag{21}
$$

The calculation of the sum of distances between the non-reference attribute section of p and those of Q is defined in Eq. (22), which also consists of the difference in the length and in the number of their common attributes. We will explain how to compute *sum_secDist* in Section 3.2.4. Next, we show how to compute *sum_typeDist*$(p.A.N, Q.A.N)$:

$$
\begin{aligned}
&sum\_typeDist(p.A.N,\ Q.A.N) \\
&= sum\_TSizeDiff(p.A.N,\ Q.A.N) \\
&+ sum\_TSetDiff(p.A.N,\ Q.A.N), \\
&\text{where} \\
&sum\_TSizeDiff(p.A.N,\ Q.A.N) \\
&= \sum_{q \in Q} TSizeDiff(p.A.N,\ Q.A.N), \ \text{ and} \\
&sum\_TSetDist(p.A.N,\ Q.A.N) \\
&= \sum_{q \in Q} TSetDist(p.A.N,\ Q.A.N)(\text{by Eq. } (8))
\end{aligned}
\tag{22}
$$

The non-reference attribute information for Q $(Q.A.N)$ can be recorded in the structured tuple $\mathbf{N}$.

Definition of Non-reference Attribute Structure N of A Object Set

$$
\begin{aligned}
\mathbf{N} =(\ & n^0_{01},\ n^1_{01},\ n^0_1,\ n^1_1,\ n^2_1,\ \ldots,\ n^a_1, \\
& n^0_{02},\ n^1_{02},\ n^0_2,\ n^1_2,\ n^2_2,\ \ldots,\ n^a_2, \\
& \ldots, \\
& n^0_{0r},\ n^1_{0r},\ n^0_r,\ n^1_r,\ n^2_r,\ \ldots,\ n^a_r),
\end{aligned}
$$

where r is the maximum number of non-reference attributes for any object's associated class; a is the total number of non-reference attributes for all the classes in the program under test; n^0_{0i} denotes the number of objects who have i non-reference attribute sections or fewer; n^1_{0i} denotes the sum of the lengths of such non-reference attribute sections; n^x_i denotes the number of objects that have i non-reference attributes and contain the attribute with label x; and n^0_i denotes the number of objects with i corresponding non-reference attributes $(1 \le i \le r;\ 1 \le x \le a)$.

Given $Q = \{q_1, q_2, \ldots, q_n\}$, for every $Q_i = \{q_i\}$ $(i = 1, 2, \ldots, n)$, we define $\mathbf{N}_i$ to be of the same structure as $\mathbf{N}$, with h being the same as in $\mathbf{N}$. However, because r denotes the number of attributes in $q_i.A.N$, r in $\mathbf{N}_i$ is smaller than or equal to that in $\mathbf{N}$, and the values for elements in $\mathbf{N}_i$ are only computed according to $Q_i = \{q_i\}$. The element x of $\mathbf{N}_i$ is referred to as $\mathbf{N}_i.x$. Let $up.A.N$ denote the reduced non-reference attribute section of p (created by removing duplicate attributes in $p.A.N$); $k_{p.A.N}$ and $k_{up.A.N}$ denote the lengths of $p.A.N$ and $up.A.N$, respectively, and $up.A.N[j]$ denote the jth non-reference attribute in $up.A.N$ $(1 \le j \le k_{up.A.N})$.

Before we consider Theorem 3, we need the following lemmas:

Lemma 18. $n^0_{0k_{p.A.N}} = \tau$, $n^1_{0k_{p.A.N}} = \sum_{i=1}^{\tau} k_{q_{v_i}.A.N}$, $n^1_{0r} = \sum_{i=1}^{n} k_{q_{v_i}.A.N}$, $m^0_{0r} = n$, and $\sum_{i=\tau+1}^{n} k_{q_{v_i}.A.N} = n^1_{0r} - n^1_{0k_{p.A.N}}$, where τ and v_i are defined in Definition (f) of Section 3.2.2.

Lemma 19. $|p.A.N \cap q_i.A.N| = |up.A.N \cap uq_i.A.N| = \sum_{l=1}^{k_{up.A.N}} N_i.n^{up.A.N[l]}_{k_{uq_i.A.N}}$.

Lemma 20. $\sum_{i=1}^{n} |p.A.N \cap q_i.A.N| = \sum_{j=1}^{k_{up.A.N}} \sum_{i=1}^{r} n_i^{up.A.N[j]}$.

We are now ready to state Theorem 3. The proofs of the theorem and its lemmas (18–20) can be found in the Appendices A.20–A.23.

Theorem 3. *With the integer tuple* $\mathbf{N}$ *of* Q, $p.A.N$ *and its reduced non-reference attribute section* $up.A.N$*, the two types of non-reference attribute distances have the following properties:*

$$\sum_{q \in Q} TSizeDiff(p.A.N,\ q.A.N) = (2 * n^0_{0k_{p.A.N}} - n^0_{0r}) * k_{p.A.N} + n^1_{0r} - 2 * n^1_{0k_{p.A.N}} \tag{22a}$$

$$\sum_{q \in Q} TSetDiff(p.A.N,\ q.A.N) = n * k_{up.A.N} + n^1_{0r} - 2 * \sum_{j=1}^{k_{up.A.N}} \sum_{i=1}^{r} n_i^{up.A.N[j]} \tag{22b}$$

3.2.4. Distance for basic data types

As discussed in Section 2.3, the calculation of the elementary distance involves each object's non-reference attribute section partitioned into four categories. Similarly, each object in Q also has four categories, but the elements in each category are integrated collections (sets), and each integrated collection stores values of the same kind (integers or characters), where each value comes from a different object.

In this section, we explain how the distance calculations within each of the four categories of non-reference attributes can be handled by comparing a single value x with a set of values $(y_1, y_2, \dots, y_n)$. By considering $(y_1, y_2, \dots, y_n)$ as a single collection, denoted Y, the calculation time can be made independent of n.

(1) Distance calculations for numbers (integers and floating point numbers)

Let $\langle k_1, k_2, \dots, k_n \rangle$ be a permutation of $\{1, 2, \dots, n\}$ satisfying $y_{k_1} \le y_{k_2} \le \cdots \le y_{k_n}$. Assume an integer a ($0 \le a \le n$) that satisfies $y_{k_a} \le x < y_{k_{a+1}}$ (when $a = 0, y_{k_0} = x$ and when $a = n$, $y_{k_{n+1}} = x$), and a non-negative constant $Range1$ denoting the length for a specific range. We have

$$\begin{aligned}
sum\_dist(x,\ Y) &= \sum_{i=1}^{n} dist(x,\ y_i) \\
&= \sum_{i=1}^{n} \frac{|x - y_i|}{Range1} = \frac{\sum_{i=1}^{n} |x - y_i|}{Range1} \qquad \text{(by Eq. (9))} \\
&= \frac{\sum_{i=1}^{n} |x - y_{k_i}|}{Range1} = \frac{\sum_{i=1}^{a} |x - y_{k_i}|}{Range1} + \frac{\sum_{i=a+1}^{n} |x - y_{k_i}|}{Range1} \\
&= \frac{\sum_{i=1}^{a} (x - y_{k_i})}{Range1} + \frac{\sum_{i=a+1}^{n} (y_{k_i} - x)}{Range1} \\
&= \frac{a * x - \sum_{i=1}^{a} y_{k_i} + \sum_{i=a+1}^{n} y_{k_i} - x * (n - a)}{Range1}
\end{aligned}$$

For a given set Y and a test candidate x, we define the following notation:

(i) d_x^0 denotes the number of elements (in Y) that are greater than or equal to x;

(ii) d_x^{-0} denotes the number of elements (in Y) that are less than x;

(iii) d_x^1 denotes the sum of all the elements (in Y) that are greater than or equal to x;

(iv) d_x^{-1} denotes the sum of all the elements (in Y) that are less than x.

Based on the notation above, there is $d_x^0 = n - a$, $d_x^{-0} = a$, $d_x^1 = \sum_{i=a+1}^{n} y_{k_i}$, and $d_x^{-1} = \sum_{i=1}^{a} y_{k_i}$. Therefore, we have $sum\_dist(x, Y) = \frac{d_x^1 - d_x^0 * x + d_x^{-1} * x - d_x^{-0}}{Range1}$.

An alternative way of calculating the values of d_x^0, d_x^{-0}, d_x^1, and d_x^{-1} involves a data structure $\mathcal{L}$ as follows: If i denotes the number of digits in the integer part of the number (left of the decimal point), and j denotes the number formed by combining the q most significant digits (from the left, including the decimal point), we have the following notations:

(i) ℓ_i^0 — the number of positive values whose integral parts have i digits;

(ii) ℓ_{-i}^0 — the number of negative values whose integral parts have i digits;

(iii) ℓ_i^1 — the sum of positive integers whose integral parts have i digits;

(iv) ℓ_{-i}^1 — the sum of negative integers whose integral parts have i digits;

(v) $\ell_{i,j}^0$ — the number of positive values in Y whose integral parts have i digits and have prefixes matching the j;

(vi) $\ell_{-i,j}^0$ — the number of negative values in Y whose integral parts have i digits and have prefixes matching the j;

(vii) $\ell_{i,j}^1$ — the sum of positive values in Y whose integral parts have i digits and have prefixes matching the j;

(viii) $\ell_{-i,j}^1$ — the sum of negative values in Y whose integral parts have i digits and have prefixes matching the j.

For example, if y=123.45 is to be added to Y, then the values of $\ell_{3,1}^0$, $\ell_{3,12}^0$, $\ell_{3,123}^0$, $\ell_{3,123.4}^0$, $\ell_{3,123.45}^0$ (i.e., $j = 1, 12, 123, 123.4, 123.45$) need to be updated.

Suppose that the maximum number of digits in any number's integer part is g, and let sum denote the sum of all the numbers in Y. It is updated whenever a new test case is added in Y.

When adding a new integer or floating point number x to Y, the values in $\mathcal{L}$ will be updated accordingly.

If k is the length of the integral part of x (excluding the positive or negative sign), and kk is the length of its decimal part, $s(x,i)$ contains the first i leftmost digits in x, excluding the sign and dot characters.

When x is negative, all l^0_{-k}, $l^0_{-k,s(x,i)}$, $l^0_{-k,s(x,j+k+1)}$ should be increased by 1, and all l^1_{-k}, $l^1_{-k,s(x,i)}$, $l^1_{-k,s(x,j+k+1)}$ should be increased by x, where $1 \le i \le k$ and $1 \le j \le kk$.

When x is positive, all l_k^0, $l^0_{k,s(x,i)}$, $l^0_{k,s(x,j+k+1)}$ should be increased by 1, and all l_k^1, $l^1_{k,s(x,i)}$, $l^1_{k,s(x,j+k+1)}$ should be increased by x, where $1 \le i \le k$ and $1 \le j \le kk$.

Therefore, adding a new value x takes $2 * (1 + k + kk)$ steps. Because both k and kk are constants, updating $\mathcal{L}$ takes constant time.

Let k represent the digits in x's integer part; kk represent the digits in x's decimal part; $x[i]$ represent the ith digit in x (excluding the decimal point); and $f[i]$ represent the number consisting of the first i digits in x (ignoring any positive/negative sign). For example, if $x = 34.25$, then $x[3] = 2$, $f[3] = 34.2$, and $f[3] - x[3]/10 + 0.3 = 34.3$. Similarly, if $x = -6.78$, then $x[2] = 7$, $f[2] = 6.7$, and $f[2] - x[2]/10 + 0.4 = 6.4$.

If x is negative, we have:

$$\begin{aligned}
d_x^1 = \sum_{i=1}^{g} \ell_i^1 &+ \sum_{i=1}^{k-1} \ell_{-i}^1 \\
&+ \sum_{i=1}^{k} \Big(\sum_{j=0}^{x[i]-1} \ell^1_{-k, f[i]-x[i]+j} \Big) \\
&+ \sum_{i=1}^{kk} \sum_{j=0}^{x[i+k]-1} \ell^1_{-k, f[i+k]-x[i+k]+\frac{j}{10^i}}.
\end{aligned}$$

$$d_x^0 = \sum_{i=1}^{g} \ell_i^0 + \sum_{i=1}^{k-1} \ell_{-i}^0 + \sum_{i=1}^{k}(\sum_{j=0}^{x[i]-1} \ell^0_{-k,f[i]-x[i]+j}) + \sum_{i=1}^{kk} \sum_{j=0}^{x[i+k]-1} \ell^0_{-k,f[i+k]-x[i+k]+\frac{j}{10^i}}.$$

$$d_x^{-1} = sum - d_x^1.$$

$$d_x^{-0} = n - d_x^0.$$

If x is positive, we have:

$$d_x^1 = \sum_{i=k+1}^{g} \ell_i^1 + \sum_{i=1}^{k}(\sum_{j=x[i]+1}^{9} \ell^1_{k,f[i]-a[i]+j}) + \sum_{i=1}^{kk}(\sum_{j=x[i+k]+1}^{9} \ell^1_{k,f[i+k]-x[i+k]+\frac{j}{10^i}}).$$

$$d_x^0 = \sum_{i=k+1}^{g} \ell_i^0 + \sum_{i=1}^{k}(\sum_{j=a[i]+1}^{9} \ell^0_{k,f[i]-x[i]+j}) + \sum_{i=1}^{kk}(\sum_{j=x[i+k]+1}^{9} \ell^0_{k,f[i+k]-x[i+k]+\frac{j}{10^i}}).$$

$$d_x^{-1} = sum - d_x^1.$$

$$d_x^{-0} = n - d_x^0.$$

Thus, the calculation of the sum of distances between x and Y is independent of the number of values in Y.

(2) Distance calculations for Boolean values

If v_0 is the number of 0s in Y, and v_1 is the number of 1s in Y, then:

$$sum\_dist(x, Y) = \sum_{i=1}^{n} dist(x, y_i) = \begin{cases} n - v_0 & \text{if } x \text{ is } 0 \\ n - v_1 & \text{if } x \text{ is } 1 \end{cases}$$

(3) Distance calculations for strings

If $Y = \{y_1, y_2, \dots, y_n\}$ is a collection of strings, we can use an integer tuple $\mathbf{W}(Y)$ to store Y's information.

Definition of W

$$\begin{aligned}\mathbf{W}(Y) =(& w_0,\ w_0^{c[0]},\ w_0^{c[1]},\ w_0^{c[2]},\ \dots,\ w_0^{c[h-1]},\\ & w_1,\ w_1^{c[0]},\ w_1^{c[1]},\ w_1^{c[2]},\ \dots,\ w_1^{c[h-1]},\\ & \dots,\\ & w_{r-1},\ w_{r-1}^{c[0]},\ w_{r-1}^{c[1]},\ w_{r-1}^{c[2]},\ \dots,\ w_r^{c[h]}),\end{aligned}$$

where h is the number of all the possible characters in Y; $c[0], c[1], \dots, c[h-1]$ represent different characters; r is the maximum length of strings in Y; w_i is the number of strings (in Y) whose length is greater than i; $w_i^{c[j]}$ is the number of strings (in Y) whose length is greater than i, and whose ith character is $c[j]$ $(0 \le i < r,\ 0 \le j < h)$.

For example, if $Y = \{$ "abc", "adc" $\}$, then $h = 4$, $c[0] =' a', c[1] =' b', c[2] =' c', c[3] =' d'$, $r = 3$, and

$$\begin{aligned}\mathbf{W}(Y) =(& 2,\ 2,\ 0,\ 0,\ 0,\\ & 2,\ 0,\ 1,\ 0,\ 1,\\ & 2,\ 0,\ 0,\ 2,\ 0)\end{aligned}$$

First, we need the following definitions and results:

$$\text{(a) } d^{(x,y_i,j)} = \begin{cases} 1 & \text{if } x[j] = y_i[j] \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

$$\text{(b) } \overline{d}^{(x,y_i,j)} = \begin{cases} 0 & \text{if } x[j] \neq y_i[j] \text{ or } j \geq k_{y_i} \\ 1 & \text{if } x[j] = y_i[j] \text{ and } j < k_{y_i} \end{cases} \tag{24}$$

Therefore, $\sum_{i=1}^{n} \overline{d}^{(x,y_i,j)}$ is the number of strings (in Y) whose length is greater than j and whose jth character is $x[j]$. It follows from the definition of $w_j^{x[j]}$ that $\sum_{i-1}^{n} \overline{d}^{(x,y_i,j)} = w_j^{x[j]}$.

$$\text{(c) } z^{(x,y_i,j)} = \begin{cases} 1 & \text{if } j < k_{y_i} \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

Therefore, $\sum_{i=1}^{n} z^{(x,y_i,j)}$ is the number of strings (in Y) whose length is greater than j. It follows from the definition of w_j that $\sum_{i=1}^{n} z^{(x,y_i,j)} = w_j$.

$$\begin{aligned}\text{(d) } & (\sum_{j=0}^{\min(k_x,k_{y_i})-1} (1 - \overline{d}^{(x,y_i,j)})) + |k_x - k_{y_i}| \\ & = \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} z^{(x,y_i,j)}\end{aligned} \tag{26}$$

The proof of Eq. (26) is given in the Appendix A.24.

We next derive the formula for distance calculations for strings:

$$\begin{aligned} sum\_dist(x, Y) &= \frac{\sum_{i=1}^{n} dist(x, y_i)}{Range3} \\ &= \frac{\sum_{i=1}^{n}(\sum_{j=0}^{\min(k_x,k_{y_i})-1} d^{(x,y_i,j)} + |k_x - k_{y_i}|)}{Range3} \\ &= \frac{\sum_{i=1}^{n}(\sum_{j=0}^{\min(k_x,k_{y_i})-1} (1 - \overline{d}^{(x,y_i,j)}) + |k_x - k_{y_i}|)}{Range3} \\ &= \frac{\sum_{i=1}^{n}(\sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} z^{(x,y_i,j)})}{Range3} && \text{(by Eq. (26))} \\ &= \frac{\sum_{i=1}^{n} \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{i=1}^{n} \sum_{j=k_x}^{k_{y_i}-1} z^{(x,y_i,j)}}{Range3} \\ &= \frac{\sum_{j=0}^{k_x-1} \sum_{i=1}^{n} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} \sum_{i=1}^{n} z^{(x,y_i,j)}}{Range3} \\ &= \frac{\sum_{j=0}^{k_x-1} (n - \sum_{i=1}^{n} \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} w_j}{Range3} && \text{(by Eq. (25))} \\ &= \frac{\sum_{j=0}^{k_x-1} (n - w_j^{x[j]}) + \sum_{j=k_x}^{k_{y_i}-1} w_j}{Range3} && \text{(by Eq. (24))} \end{aligned}$$

In Eq. (27), $pG[i][j]$ and $QG[i][j]$ denote the jth non-reference variable of the ith data type in the group $pG[i]$ and $QG[i]$, respectively. This formula requires pairing the non-reference variables in $pG[i]$ and $QG[i]$ such that the minimum distance between these two sets will be returned. If one group ($pG[i]$ or $QG[i]$) has fewer variables than the other, it is filled with supplemental null values. Eq. (28) defines the sum of distances between attribute x and the set of attributes Y of the same type. Eq. (28a) handles the special case where a group of variables ($pG[i]$ or $QG[i]$) contains undefined or null values (to make a smaller group be the same size as the larger one). Eqs. (28b) to (28e) give the formulas for non-reference attribute value distance calculations (see Box I).

3.3. L′OP-ART algorithm and overheads analysis

Since OOS primarily revolves around classes, which contain valuable data for generating test inputs and measuring distances, the initial

$$sum\_secDist(p.A.N, q.A.N) = \sum_{q\in Q} secDist(p.A.N, q.A.N) = \sum_{i=1}^{4} \min(\bigcup_{s=1}^{m!}\{\sum_{j=1}^{m} sum\_dist(pG[i][j], PL_s(qG[i], j))\}), \tag{27}$$

where $m = \max(size(pG[i]), size(qG[i])) > 0$.

$sum\_dist(x, Y)$, where both x and the value in Y are of the same type. We let$n$denote the number of defined values in Y: (28)

If x and Y belong to one of the four commonly used types, and x is undefined (an unknown value), then sum_dist $(x, Y) = n$.**Note** : if all values in Y are undefined, then k_Y (the number of defined elements) = 0. (28a)

If both x and Y are *integers* (int, short int, long int, signed int, or unsigned int) or real numbers (float or double) :

$$sum\_dist(x, Y) = \frac{d_x^1 - d_x^0 * x + d_x^{-1} * x - d_x^{-0}}{Range1}, \text{ where } Range1 > 0\text{, and is a specific range of the input domain.} \tag{28b}$$

If both x and Y are *characters* : $sum\_dist(x, Y) = \dfrac{d^1_{ASCII(x)} - d^0_{ASCII(x)} * ASCII(x) + d^{-1}_{ASCII(x)} * ASCII(x) - d^{-0}_{ASCII(x)}}{Range2}$,

where $ASCII(x)$ represents the ASCII value of x, and $Range2$ ($>$ 0) is a specific range of the input domain. (28c)

If both x and Y are *strings* : $sum\_dist(x, Y) = \dfrac{\sum_{j=0}^{k_x-1}(n - w_j^{x[j]}) + \sum_{j=k_x}^{k_{y_i}-1} w_j}{Range3}$,

where $Range3 > 0$, and is a specific range of the input domain or the length of the longest user-set input string. (28d)

$$\text{If both } x \text{ and } Y \text{ are } Boolean: \; sum\_dist(x, Y) = \begin{cases} n - v_0, & \text{where } v_0 \text{is the number of 0s in } Y \quad \text{if } x \text{ is } 0 \\ n - v_1, & \text{where } v_1 \text{ is the number of 1s in } Y \quad \text{if } x \text{ is } 1 \end{cases}. \tag{28e}$$

Box I. The explanation of $sum\_dist(x, Y)$.

step involves extracting class-related information, like the class diagram, from the source code of the subject program. This information may include details such as the largest number of methods in the invocation sequence of any test case, the overall count of methods across all program classes, and the maximum number of objects. Once this class information is obtained, L′OP-ART can generate random test inputs, each comprising a collection of objects and a sequence of invoked methods. Nonetheless, numerous randomly generated inputs consist of improper method invocation sequences that could potentially cause the program to terminate unexpectedly. Consequently, for this research, we opted to eliminate these invalid sequences based on a customized data structure, resulting in a collection of valid test inputs, as illustrated in Algorithm 1–4.

Because an object has reference variables, non-reference variables, and methods, we use an *ObjectTuple* data structure to store an object's data information: One *ObjectTuple* contains one method tuple **S** (whose properties are described in Theorem 2), one behavior tuple **M** (whose properties are described in Theorem 3), and some data structures containing the values of the non-reference variables. Because an *ObjectTuple* represents an object, and an object may have reference attributes (which can also be objects), an *ObjectTuple* may include another *ObjectTuple*.

The purpose of Algorithm 1 is to implement the LW-OMISS using FSCS-ART with the max-sum criterion for generating test cases. As shown in Algorithm 1, the L′OP-ART algorithm first initializes the global tuple **S** (whose properties are described in Theorem 1), and then initializes the corresponding number of *ObjectTuples* (if needed) before adding the next selected test case into the test set, as shown in line 1. The relevant data structures are then updated. When adding a test case t to the set E, the algorithm updates **S** by incrementing s_0^0 and $s_{m_2}^0$ by 1 as shown in line 25, each s_{0x}^0 by 1 and each s_{0x}^1 by m_1 (where $x = m_1, \dots, g$); and each $s_{m_2}^{r_i}$ and $s_{m_1,j}^{r_j}$ by 1 (where $i = 1, 2, \dots, m_2$; $j = 1, 2, \dots, m_1$); and initializing $size(t.OBJ)$ number of *ObjectTuples* as shown in line 17-21 by invoking Algorithm 4. As shown in line 27, updating one *ObjectTuple* requires that one method tuple, one attribute tuple, and $size(o.A.N)$ data structures be updated by invoking Algorithm 2. If the corresponding object o has reference attributes, recursive updating is necessary.

The purpose of Algorithm 2 is to update a *ObjectTuple*. Its input consist of an object (called o) and a loop count parameter ($loopCount$). The algorithm first update the part elements of the method tuple **M** and the attribute tuple **N** according to the specific object o. Next, the algorithm categorizes the non-referenced attributes of o into four groups, namely integer and real number, character, boolean, and string. For the attributes in each group, the algorithm performs the corresponding update operations for obtaining the distance between these attributions. For the attributes that belong to integer and real number and characters, the UpdateInteger function is called for updating their value by invoking Algorithm 3. For Boolean attributes, the corresponding count is increased depending on whether the attribute value is 0. For string attributes, textbfW can be updated after obtaining the total number of such attributes. Repeat the above process until $loopCount$ decreases one by one to 0.

Algorithm 3 is used to update the non-reference attributes with integer type. The input parameters include an integer value $value$ and a $\mathcal{L}$ data structure. First, the algorithm defines a symbol $s(value, i)$ that denotes the former i digits in $value$ without positive or negative sign. Next, the algorithm computes the length of integral part of $value$ (i.e. k_1) and the length of fractional part of $value$ (i.e. k_2). Depending on the positive and negative values of the attributes, the distance between such attributes is calculated in the ways introduced in Section 3.2.4.

Algorithm 1, 2 and 4 show that updating an MINV tuple, a Method tuple, and an Attribute tuple involves less than $2 * (g{+}2)$, $2 * (c{+}2)$, and $2 * (r{+}2)$ steps, respectively (where g, c, and r in these algorithms have the same meaning as the corresponding symbols in the tuples **S**, **M**, and **N**). Algorithms 2 and 3 show that updating the data structures storing numbers or characters takes less than $2 * (1{+}k_1{+}k_2)$ steps, and updating data structures storing strings takes less than $2k + 1$ steps — where k,

Algorithm 1 Algorithm L′OP-ART

```
Initialize **S** (defined in Theorem 1) by setting $s_0^0$ and each $s_{0i}^x$, $s_i^y$, $s_{i,j}^z$ to 0, where $x \in \{0,1\}$; $y \in \{0,1,\dots,h\}$; $z \in \{1,2,\dots,h\}$; $i \in \{1,2,\dots,g\}$; and $j \in \{1,2,\dots,i\}$. Here, $g$ is the largest possible number of methods in the invocation sequence of any test case, and $h$ is the total number of methods in all the classes of the program under test.
Set **TD** = {};
Set $n = 0$ and $\mathbf{E} = \{\}$, $objectSetNum = 0$;
Define an integer $k > 0$ as the number of candidates to be generated;

while Termination condition is not satisfied do
  Increment $n$ by 1;
  if $n = 1$ then
    Randomly generate a test case $e_n$;
  else
    Randomly generate $k$ candidates $c_1, c_2, \dots, c_k$;
    for all $c_u$ $(u = 1, 2, \dots, k)$ do
      Calculate $sum\_TestcaseDistance(c_u, \mathbf{E})$ according to Eq. (14) to Eq. (28);
    end for
    Set $e_n = c_o$, where $\forall u$, $sum\_TestcaseDistance(c_o, \mathbf{E}) \geq sum\_TestcaseDistance(c_u, \mathbf{E})$;
  end if
  Add $e_n$ into **E**;
  if $objectSetNum < size(e_n.OBJ)$ then
    for $i = objectSetNum$ to $size(e_n.OBJ)$ do
      Initialize one ObjectTuple;
      add ObjectTuple into **TD**;
    end for
  end if
  $objectSetNum = size(e_n.OBJ)$;
  Set $m_1 = size(e_n.MINV)$, $m_2 = size(ue_n.MINV)$;
  Update **S** by incrementing $s_0^0$ and $s_{m_2}^0$ by 1 if $m_2 \neq 0$; each $s_{0x}^0$ by 1 and each $s_{0x}^1$ by $m_1$, where $x = m_1, \dots, g$; each $s_{m_2}^{ue_n.MINV(j)}$ and $s_{m_1,i}^{e_n.MINV(i)}$ by 1, where $i = 1, 2, \dots, m_1$; $j = 1, 2, \dots, m_2$;
  for $i$=0 to $size(e_n.OBJ) - 1$ do
    Update the ObjectTuple corresponding to $e_n.OBJ[i]$;
  end for
end while
```

Algorithm 2 Updating one *ObjectTuple*

```
**Update**(o,ObjectTuple,loopCount)
Set $\ell_1 = size(o.M)$, $\ell_2 = size(uo.M)$;
Update ObjectTuple.**M** by incrementing $m_0^0$ and $m_{\ell_1}^0$ by 1 if $\ell_1 \neq 0$; each $m_{0x}^0$ by 1 and each $m_{0x}^1$ by $\ell_1$, where $x \in \{\ell_1, \ell_1 + 1, \dots, c\}$; and each $m_{\ell_2}^{uo.M(i)}$ by 1, where $i \in \{1, 2, \dots, \ell_2\}$;
Set $\ell_{e_1} = size(o.A.N)$, $\ell_{e_2} = size(uo.A.N)$;
Update ObjectTuple.**N** by incrementing $n_0^0$ and $n_{\ell_{e_1}}^0$ by 1 if $\ell_{e_1} \neq 0$; each $n_{0x}^0$ by 1 and each $n_{0x}^1$ by $\ell_{e_1}$, where $x \in \{\ell_{e_1}, \ell_{e_1} + 1, \dots, r\}$; and each $m_{\ell_{e_2}}^{uo.A.N(i)}$ by 1, where $i \in \{1, 2, \dots, \ell_{e_2}\}$;
Classifying non-reference attributes of $o$ based on their types into four groups (denoted $OG[4]$): integer and real number; character; boolean; and string;
for $i$=0 to $size(OG[0]) - 1$ do
  **UpdateInteger**($OG[0][i]$, ObjectTuple.$\mathcal{L}[i]$);
end for
for $i$=0 to $size(OG[1]) - 1$ do
  **UpdateInteger**($OG[1][i]$, ObjectTuple.**DSRC**$[i]$);
end for
for $i$=0 to $size(OG[2]) - 1$ do
  if $OG[1][i] = 0$ then
    ObjectTuple.**BD**$[i].v_0 += 1$;
  else
    ObjectTuple.**BD**$[i].v_1 += 1$;
  end if
end for
for $i$=0 to $size(OG[3]) - 1$ do
  Set $k = size(OG[3][i])$;
  Update ObjectTuple.**W**.$w_i^{value[i]}$ and ObjectTuple.**W**.$w_i$ by 1, where $i = 0, 1, \dots, k - 1$;
end for
if $loopCount \neq 0$ then
  for $i = 0$ to $size(o.A.R) - 1$ do
    **Initialize**($o.A.R[i]$.ObjectTuple, $loopCount - 1$);
  end for
end if
```

k_1, and k_2 in these algorithms have the same meaning (the number of pairs of objects) as the corresponding symbols in the tuples **S**, **M**, and **N**. Because all the parameters used in Algorithms 2 and 3 are constants, the updating time in these algorithms is also constant (but may vary in different test cases). Although recursive updating is used in Algorithm 2 (which takes up most of the updating time), because the called inner operations only take constant time, Algorithm 2 still only takes constant time to complete. This overhead can be further reduced by reducing the number of cycles (however, this would also reduce the precision). Furthermore, the updating time required by Algorithm 3 can also be reduced by setting k_2 to be a small value (which would also result in a small loss of precision). The updating of **S** (Theorem 1) after execution of a test case and the method invocation sequence distance calculation for a candidate using Eq. (16) are independent of the number of tests. This means that the updating and calculation operations for the method invocation sequence require constant time. Although the method tuple **M** (Theorem 2) and attribute tuple **N** (Theorem 3) are similar to the method invocation sequence tuple **S**, they are also less complex than the structured tuple S. Thus, the updating and calculating operations of **M** and **N** also take constant time. The elementary distance calculation and the updating of the relevant data structures are dependent on the sizes of the elementary values (such as integers and strings), but such values are in pre-defined ranges. Thus, the time required for the elementary distances is also independent of the number of test cases.

In conclusion, the time analysis shows that the selection of a single test case takes a time that is unrelated to the number of test cases generated (n). However, the time required for some calculations is determined by some of the parameters of the candidate test case. For example, the time required for calculation of Eq. (16b) depends on the length of the method invocation sequence of the candidate test case — with longer method invocation sequences requiring more time — but is independent of the number of test cases already executed. Elementary distance calculations are also independent of the number of executed test cases, but do depend on the actual values involved: for integer distances, for example, the number of digits involved impacts on the time required, with more digits requiring more time. When the number of already executed test cases is not large, the original OMISS calculation for integers may take less time than using Eq. (28b). However, as n increases, the time required by the relevant formula in the original OMISS will increase beyond that for using Eq. (28b). Additionally, when calculating the distance for the *real number* data type, a specified upper bound on the number of digits in the variables is required. As a result, generating a set of n test cases takes linear time.

The space complexity for L′OP-ART is determined by the storage requirements of the Global Method Invocation Structure (S), Behavior Structure of an Object Set (M), Non-reference Attribute Structure of an Object Set (N), numbers structure (L), and strings structure (W). While

Algorithm 3 Updating the integer section

```
UpdateInteger($value$, $\mathcal{L}$)
Let $s(value, i)$ denote the former $i$ figures of $value$ without positive or negative sign (with the dot punctuation if necessary);
Set $k_1$ as the length of integral part of $value$ without positive or negative sign;
Set $k_2$ as the length of fractional part of $value$;
if $value < 0$ then
    $\ell^1_{-k_1}$ $+=value$;
    $\ell^0_{-k_1}$ $+=1$;
    for $i$=1 to $k_1$ do
        $\ell^1_{-k_1,s(value,\ i)}$ $+=value$;
        $\ell^0_{-k_1,s(value,\ i)}$ $+=1$;
    end for
    for $i$=1 to $k_2$ do
        $\ell^1_{-k_1,s(value,\ i+k_1+1)}$ $+=value$;
        $\ell^0_{-k_1,s(value,\ i+k_1+1)}$ $+=1$;
    end for
end if
if $value \geq 0$ then
    $\ell^1_{k_1}$ $+=value$;
    $\ell^0_{k_1}$ $+=1$;
    for $i$=1 to $k_1$ do
        $\ell^1_{k_1,s(value,\ i)}$ $+=value$;
        $\ell^0_{k_1,s(value,\ i)}$ $+=1$;
    end for
    for $i$=1 to $k_2$ do
        $\ell^1_{k_1,s(value,\ i+k_1+1)}$ $+=value$;
        $\ell^0_{k_1,s(value,\ i+k_1+1)}$ $+=1$;
    end for
end if
```

Algorithm 4 Initializing one *ObjectTuple*

```
Initialize(ObjectTuple,loopCount)
Set refNumMax = 5;
Initialize M (defined in Theorem 2) by setting $m_0^0$ and each $m_{0i}^x$ and $m_i^y$ to 0 for $x \in \{0,1\}$, $y \in \{0,1,2,\dots,h\}$, and $i \in \{1,2,\dots,c\}$, where $c$ is the largest possible number of methods in the associated class of any object;
Initialize N (defined in Theorem 3) by setting $n_0^0$ and each $n_{0i}^x$, $n_i^y$ to 0, where $x \in \{0,1\}$; $y \in \{0,1,\dots,a\}$; and $i \in \{1,2,\dots,r\}$, where $r$ is the largest possible number of non-reference attributes in the associated class of any object, and $a$ is the total number of non-reference attributes in all the classes of the program under test;
if loopCount $\neq 0$ then
    Set ObjectTupleSet={};
    for $i = 0$ to refNumMax do
        Initialize(ObjectTuple, loopCount − 1);
        Add ObjectTuple into ObjectTupleSet;
    end for
end if
Add {M, N, $\mathcal{L}[max]$, DSRC[$max$], BD[$max$], W[$max$], ObjectTupleSet} into ObjectTuple, where $max$ is the largest possible number of attributes in each group of objects;
```

L′OP-ART introduces additional structures, these do not significantly impact the asymptotic space complexity compared to OMISS-ART. The space complexity for both remains polynomial concerning the program's methods and attributes, which is typically overshadowed by the time complexity improvements. However, due to a trade-off between time and space complexity, L′OP-ART inevitably uses some additional memory to attain the presented performance improvements. The memory requirements imposed by the introduced data structure are not significantly correlated with the number of candidate test cases, but the spatial complexity of L′OP-ART is quadratic with respect to the number of methods that take place in method invocation sequence of the executed test case set that is commonly dependent on the system under test. Although the maximum number of methods in the method invocation of the executed test case set is usually not very large, it will inevitably require additional memory space compared to the previously proposed approach.

4. Empirical study

In this section, we report and analyze the results of the empirical study conducted to assess the computational overheads incurred by L′OP-ART and its cost-effectiveness in practice. We also evaluate the approach's failure-detection effectiveness against RT and other ART algorithms used for OO testing.

4.1. Subject programs

We studied ten subject programs, each of which had three different versions through injecting mutating operators. Table 4 summarizes the details of these subject programs. Calendar and SATM are small in size, with only a limited number (five and four, respectively) of faulty versions available. NSort, WaveletLibrary, EnterpriseManagement and ID3Manage are also relatively small and simple, but come with sets of faults in the form of mutants. IceChat, Foundation, Net, and XML are large programs for which mutation faults were generated. All programs were downloaded from open-source websites (Team, 2023; Microsoft, 2023; CodeProject, 2023; Media, 2023a; Engineering, 2023), and were implemented in C++ and C#. In all the subject programs, faults were seeded into randomly selected methods, with no method having more than a single fault. The 13 commonly used mutation operators (Jia and Harman, 2011) we used in our study are as follows:

1. arithmetic operators replacement
2. logical operators replacement
3. relational operators replacement
4. constant for scalar variable replacement
5. scalar variable for scalar variable replacement
6. scalar variable for constant replacement
7. array reference for constant replacement
8. new method invocation with child class type
9. argument order change
10. accessor method change
11. access modifier change
12. hiding variable deletion
13. property replacement with member field

4.2. Evaluation measures

In the study, we compared L′OP-ART with three other techniques: random testing (RT), Divergence-oriented ART (DO-ART) (Lin et al., 2009), and OMISS-ART (Chen et al., 2016).

Adaptive random testing (ART) attempts to improve the failure-detection effectiveness of RT by spreading test cases more evenly throughout the input domain. L′OP-ART makes use of innovative calculation formulas to achieve this widespread distribution (calculating sums of distances between one and n test cases based on the LW-OMISS metric). DO-ART with multiple methods invocation is one of the popular methods used for OOS testing. The comparison with RT and DO-ART is therefore a fundamental and effective way to present the improvements in L′OP-ART's failure-detection ability. For consistency

Table 4
Subject programs.

SUT ID	SUT name	Version	Lines of code	Num. of public classes	Num. of public methods	Num. of faults	Description
1	Calendar (Team, 2023)	V1 V2 V3	287	5	27	5 5 4	C++ library for *calendar* operation
2	SATM (Team, 2023)	V1 V2 V3	197	1	9	4 5 4	C++ library that simulates an *ATM*
3	WaveletLibrary (Microsoft, 2023)	V1 V2 V3	2406	12	84	15 10 7	C# library for wavelet algorithms
4	NSort (CodeProject, 2023)	V1 V2 V3	1118	18	61	14 9 7	C# library for sorting algorithms
5	IceChat Media (2023a)	V1 V2 V3	71000	101	271	22 23 24	C# program that implements an IRC (Internet Relay Chat) Client
6	poco-1.4.4: Foundation (Engineering, 2023)	V1 V2 V3	149547	641	4480	28 28 28	C++ library that contains a platform abstraction layer and a large number of utility classes
7	poco-1.4.4: Net (Engineering, 2023)	V1 V2 V3	52036	255	1568	22 21 20	C++ library that contains a platform abstraction layer and a large number of utility classes
8	poco-1.4.4: XML (Engineering, 2023)	V1 V2 V3	38544	184	952	17 15 13	C++ library that contains a platform abstraction layer and a large number of utility classes
9	EnterpriseManagement (Media, 2023b)	V1 V2 V3	1357	8	76	8 7 8	C# program for managing enterprise business
10	ID3Manage (Media, 2023b)	V1 V2 V3	4538	28	129	12 11 12	C# library for reading and writing of ID3 tags in MP3 files

with L′OP-ART, test cases handled by RT also consist of multiple objects and methods of multiple classes.

OMISS-ART was introduced by Chen et al. (2016) that achieves improved failure-detection effectiveness compared with ARTOO and ARTGen at a cost of high computational overheads : The selection of n test cases incurs an overhead of $O(n^2)$. L′OP-ART was motivated by the observation that because OMISS-ART's formula complexity was a main contributor to the large coefficient of the quadratic time overheads, strategies reducing these formula computations could significantly reduce the overheads. To facilitate the study, the "forgetting" technique (Chan et al., 2006a) with a fixed-size subset of the previously executed test cases was used to reduce the time required by OMISS-ART. To be consistent with the original OMISS-ART study (Chen et al., 2016), we also set the subset size to 20 for OMISS-ART. Due to the LW-OMISS was introduced in L′OP-ART, L′OP-ART need not use the forgetting technique.

For failure-detection effectiveness evaluation, we used three standard metrics: 1) F-measure (F_m) (Chen et al., 2010), i.e., the number of test cases required to detect the first failure (Chen et al., 2006), 2) E-measure (E_m), i.e. the expected value of the number of detected failures for a given test suite, and the ratio between the number of detected faults and the number of test cases (marked as *Ratio*).A testing algorithm with a low F_m or high E_m or a high *Ratio* has high failure-detection effectiveness. Besides, our method is to make OO test cases evenly distributed in the input domain. In order to verify the effect of our method, we use mutation testing to generate the possible faults of software in practice. Therefore, the automatic test prediction is not within the scope of our research. We already have the test prediction (source program) by default. To compare the actual selection overhead of L′OP-ART against the competing algorithms, we analyzed the execution time required for the four algorithms to generate test cases. We also developed a testing tool implementing L′OP-ART to enable verification of the failure-detection and cost effectiveness.

4.3. Experiment setup

In order to evaluate the effectiveness of L′OP-ART based on a lightweight OMISS metric, we developed a detailed experimental plan for rigorous and objective purposes. A description of the specific experimental setup is given below.

For each subject program, the F_m was averaged over 200 test runs for each test algorithm, each run using different seeds for the random number generation. To observe the changes in the time for test suite generation as the number of test cases (n) increased, the generation time was calculated for various n values $(100, 500, 1000, 1500, \ldots, 5000)$. To be consistent with the experimental settings in Chen et al. (2016), the maximum number of $t.OBJ$ and $t.MINV$ was set to 5, and the size of the candidate set (k) used by L′OP-ART and OMISS-ART was set to 10. Additionally, when calculating the recursive distance in L′OP-ART and OMISS-ART, the calculation depth was set to 4.

5. Data and analysis

5.1. F-measure

The F-measure results for each testing algorithm are given in Fig. 4 and Table 5. The boxplot generated by the F_m of four approaches over 200 test runs for each subject program is shown in Fig. 4, and the results in Table 5 are the averages over 200 test runs for each subject program. It can be seen from Table 5 that L′OP-ART and OMISS-ART both outperform RT for all the programs, and L′OP-ART outperforms OMISS-ART for all the programs except Calendar, WaveletLibrary and NSort (Programs #1, #3 and #4). Besides, for the detection of the first failure, L′OP-ART requires 4.25% fewer test cases than OMISS-ART on average. Fig. 4 indicates that under four programs (#5, #7,

Table 5
F-measures for L′OP-ART, OMISS-ART, DO-ART and RT.

	1_V1	1_V2	1_V3	2_V1	2_V2	2_V3	3_V1	3_V2	3_V3	4_V1	4_V2	4_V3	5_V1	5_V2	5_V3	6_V1	6_V2	6_V3	7_V1	7_V2	7_V3	8_V1	8_V2	8_V3	9_V1	9_V2	9_V3	10_V1	10_V2	10_V3	AVG
L′OP-ART	2.72	2.53	4.45	51.58	50.96	51.34	7.46	7.16	7.08	34.31	31.49	32.34	85.70	83.46	84.18	195.78	206.47	206.69	87.67	88.14	85.19	69.14	70.44	69.53	38.45	39.77	41.31	27.12	28.56	27.16	60.61
OMISS-ART	2.51	2.48	4.40	52.85	51.72	50.85	7.31	6.95	7.18	33.71	30.48	32.06	100.78	99.68	97.70	195.94	207.21	206.97	99.37	98.55	95.39	80.81	80.88	81.92	41.46	40.48	44.62	29.59	30.05	32.29	64.86
DO-ART	4.64	4.86	4.70	39.24	41.22	43.89	10.58	10.05	10.27	87.26	89.82	94.14	249.47	250.97	254.14	256.14	258.81	264.80	143.89	145.14	147.84	111.51	113.28	117.28	67.34	69.10	72.42	59.41	62.78	65.69	105.02
RT	3.30	3.30	4.89	57.44	58.16	59.07	7.87	8.06	7.55	41.04	40.57	41.57	145.63	150.35	141.71	228.02	231.84	233.21	141.74	142.1	147.66	123.73	125.05	123.6	47.80	52.61	52.55	40.14	43.59	43.67	84.93

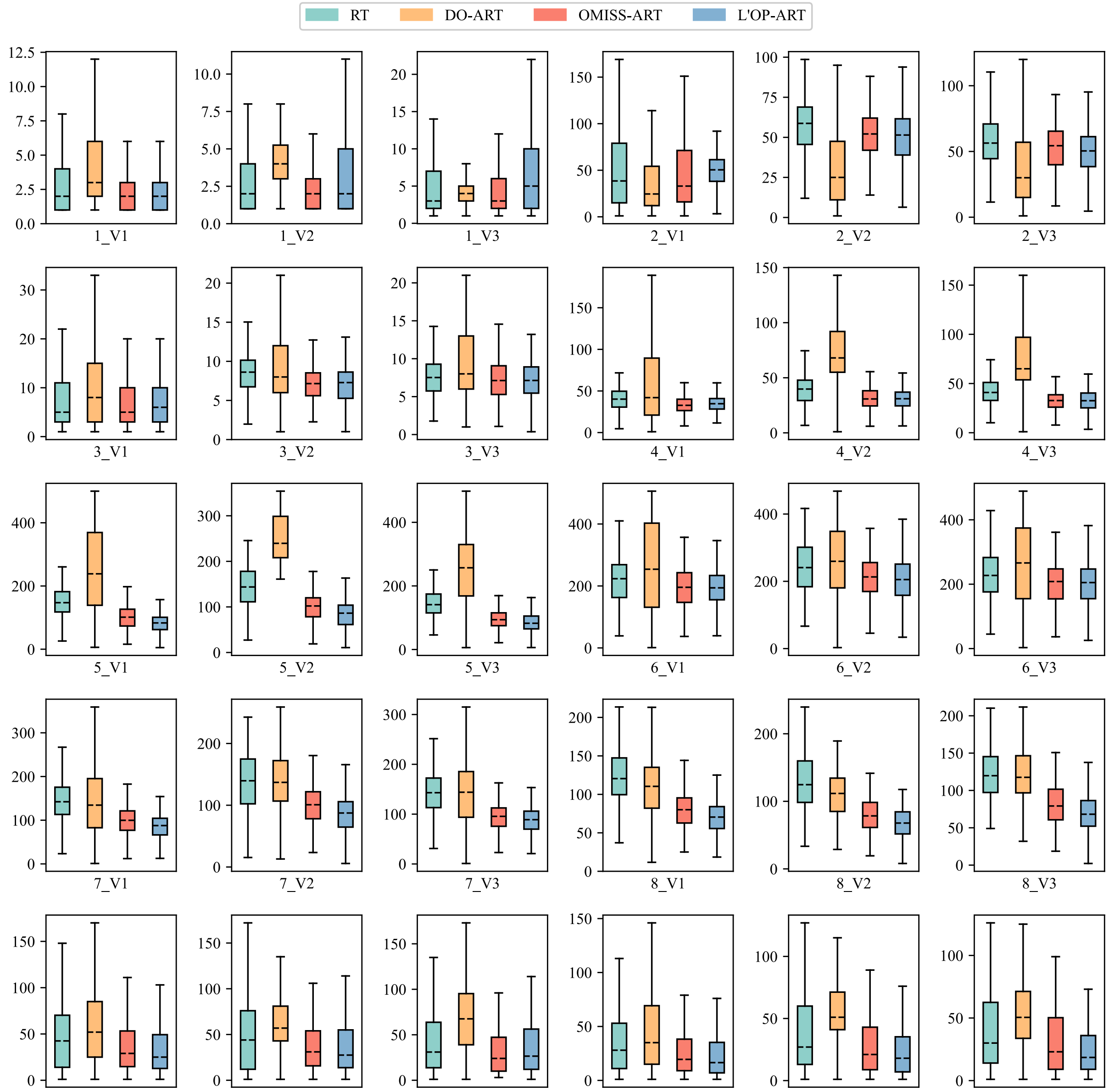


Fig. 4. F-measures for RT, DO-ART, OMISS-ART and L′OP-ART.

#8 and #10), L′OP-ART has the lower and more stable F_m than OMISS-ART. Besides, L′OP-ART and OMISS-ART have comparable performance on four programs (#2, #3, #4 and #9) in terms of the stability of F_m. On the other hand, DO-ART has the largest average F_m over 200 experiments compared to the other three methods, i.e. 105.02.

To facilitate a clearer comparison, the F-measure improvements achieved by L′OP-ART, OMISS-ART and DO-ART over RT are presented in Fig. 5 and Table 6. Overall, the average improvements achieved by L′OP-ART and OMISS-ART over RT are 28.64% and 23.63%, respectively.

To further study the significance of the differences in F-measures for each subject program, we report in Table 7 the p-value and effect size (ES) (Arcuri and Briand, 2014) for pairwise comparisons between the techniques. The p-value (probability value) is used to show the statistical significance of differences, with a p-value less than 0.05 meaning that there is a significant difference between the two compared methods (Arcuri and Briand, 2014). We also employed the non-parametric effect size (ES) measure to show the probability that one method is better than the other (Vargha and Delaney, 2000): Given an ES value for any two methods A and B, a higher ES value indicates a higher probability that A is better than B (Chen et al., 2018).

The results show that the difference in F_m between L′OP-ART and RT is significant (because the p-value is less than 0.05), except for program WaveletLibrary. The performance of L′OP-ART is comparable with that of OMISS-ART, and the difference is not significant for half of the subject programs (because the p-value is greater than 0.05).

Table 6
Improvements in F-measures achieved by L′OP-ART,DO-ART and OMISS-ART compared with RT.

	1_V1	1_V2	1_V3	2_V1	2_V2	2_V3	3_V1	3_V2	3_V3	4_V1	4_V2	4_V3	5_V1	5_V2	5_V3	6_V1	6_V2	6_V3	7_V1	7_V2	7_V3	8_V1	8_V2	8_V3	9_V1	9_V2	9_V3	10_V1	10_V2	10_V3	AVG
L′OP-ART(%)	17.58	23.33	9.00	10.20	12.38	13.09	5.21	11.17	6.23	16.40	22.38	22.20	41.15	44.49	40.60	14.14	10.94	11.37	38.15	37.97	42.31	44.12	43.67	43.75	19.56	24.41	21.40	32.44	34.49	37.81	28.64
OMISS-ART(%)	23.94	24.85	10.02	7.99	11.07	13.92	7.12	13.77	4.90	17.86	24.87	22.88	30.80	33.70	31.06	14.07	10.62	11.25	29.89	30.65	35.40	34.69	35.32	33.72	13.26	23.82	15.10	26.28	31.07	26.07	23.63
DO-ART(%)	−40.61	−47.27	3.89	31.69	29.13	25.70	−34.43	−24.69	−36.03	−112.62	−121.40	−126.46	−71.30	−66.92	−79.34	−12.33	−11.63	−13.55	−1.52	−2.14	−0.12	9.88	9.41	5.11	−40.88	−31.34	−37.81	−48.01	−44.02	−50.42	−23.66

Through a further analysis of the ES values for the various pairwise comparisons, we found that the results confirm L′OP-ART and OMISS-ART both outperform RT significantly — ES values greater than 0.5 indicate higher performance. The three columns titled “Better Method” in Table 7 highlight the better method between the two in comparison.

Another phenomenon worth in-depth analysis is that L′OP-ART achieves significant performance gains over other methods for 3 of the 10 subject systems (i.e. 5, 7, 8). In general, L′OP-ART is more suitable for large-scale programs in most cases, because large-scale programs usually involve complex “method-in-object” invocation relationships and long invocation sequences. Given such large-scale programs, valid test cases usually contain longer sequences of valid invocation. L′OP-ART is more capable of generating valid test cases due to its lower computational overhead, and reduces the waste of resources caused by invalid test cases. That is, the number of test cases invested in discovering the first defect is likely less, i.e., the Fm is smaller.

Although the 6th subject system (Foundation) is by far the largest one in terms of all the metrics (LOC, number of classes/methods), it acts as a base library, which contains a large number of relatively independent methods without complex internal method invocation relationships and long invocation sequences. The testing results of the Foundation library are similar to those of some smaller programs. L′OP-ART cannot show the advantages of handling such classes and methods independently without complex sequences of valid invocation. The method invocation sequences generated by L′OP-ART for the 7th subject program (Net), the 8th subject program (XML) those depend on the 6th subject program (Foundation) are relatively more complex, and the valid sequences are longer, which is a better reflection of the advantages of L′OP-ART in handling large-scale programs. The 5th subject program (IceChat) is a user-friendly program that can connect to multiple servers and channels simultaneously. When generating test cases for IceChat, more complex and valid invocation sequences are usually generated, demonstrating the advantages of L′OP-ART in handling large-scale programs.

In conclusion, our analysis indicates that while L′OP-ART is optimized for large-scale programs, its performance depends on the complexity of the “method-in-object” invocation relationships and the length of invocation sequences.

5.2. E-measure

Table 8 provides the E-measure results obtained by four approaches over 200 test runs for each subject program. It is evident that both L′OP-ART and OMISS-ART consistently outperform DO-ART and RT across nine tested programs, and the four approaches obtain the same E-measure under one program (#1). There are specific scenarios where DO-ART’s E-measure falls below that of RT, specifically for the program 4_V2 and 9_V1. Conversely, in all other cases, DO-ART consistently achieves higher E-measure compared to RT. Furthermore, Table 9 and Fig. 6 provide insights into the improvements in E-measure achieved by the four different approaches. On average, L′OP-ART, OMISS-ART, and DO-ART show improvements over RT of approximately 6.37%, 6.55%, and 2.84%, respectively. In summary, the results suggest that L′OP-ART and OMISS-ART consistently outperform RT, with minimal differences between them, while DO-ART’s performance varies depending on the specific program versions.

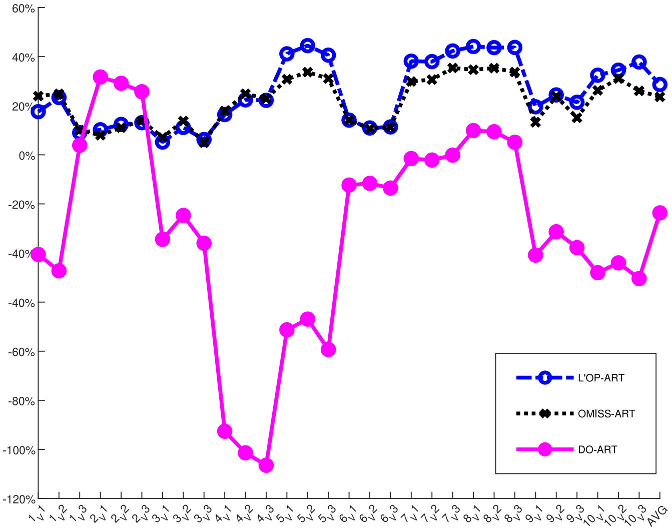


Fig. 5. Improvements in F-measures for L′OP-ART, OMISS-ART, and DO-ART compared with RT.

5.3. The connection between the number of detected faults and the number of test cases

Fig. 7 shows the relationship between the number of test cases and the number of detected faults of four different approaches on V1 of ten different programs. Under four programs (Calendar, SATM, WaveletLibrary, and XML), L′OP-ART detects the highest number of faults regardless of the number of test cases. For other five programs apart from EnterpriseManagement, the number of faults detected by L′OP-ART ranks first with other methods or in second place as the number of test cases increases. Besides, the number of faults detected by OMISS-ART exceeds that of the RT method for six programs when the number of test cases reaches 1,000 or more. Under all seven programs, DO-ART consistently identifies the fewest number of faults regardless of the quantity of test cases employed. Overall, it can be seen that different methods show different efficacy under different conditions of program and number of test cases. L′OP-ART possesses the best ability to detect faults, while in other cases OMISS-ART or RT may be more advantageous. DO-ART performs poorly in most cases.

5.4. Test suite generation time

Figs. 8 to 17 show the time taken by the four algorithms to generate test suites of various sizes for the subject programs. The results of OMISS-ART in these figures show the linear time complexity derived from OMISS-ART with “forgetting” techniques. Table 10 shows both the time taken by the four algorithms to generate 5000 test cases, and the comparisons in the relative times according to the versions of the programs (V1, V2, or V3). It can be seen that there is some variation in the relative time cost across different programs. As can be seen from these figures and Table 10, L′OP-ART and RT require linear time relative to the size of the generated test suite. This confirms our theoretical analysis in Section 3. The constant factors for OMISS-ART are greater than those for the other approaches.

Table 7
F-measure comparisons between various pairs of methods using *p*-value and effect size (ES).

SUT ID	L′OP-ART and RT			OMISS-ART and RT			L′OP-ART and OMISS-ART		
	p-value	ES	Better method	p-value	ES	Better method	p-value	ES	Better method
Calendar	0.003927922	0.5469028	L′OP-ART	0.02876791	0.5356583	OMISS-ART	0.3326464	0.515725	L′OP-ART
SATM	2.919725e−13	0.6215611	L′OP-ART	3.127814e−10	0.6048153	OMISS-ART	0.6184881	0.5082972	L′OP-ART
WaveletLibrary	0.09331011	0.5275028	L′OP-ART	0.0485372	0.5322847	OMISS-ART	0.7731636	0.4952861	OMISS-ART
NSort	1.047912e−66	0.7867472	L′OP-ART	3.489541e−76	0.8069889	OMISS-ART	0.0002277158	0.4390653	OMISS-ART
IceChat	1.410926e−159	0.9487611	L′OP-ART	3.823515e−164	0.9552222	OMISS-ART	8.054954e−12	0.6139875	L′OP-ART
Foundation	3.442655e−06	0.5774069	L′OP-ART	0.04319121	0.5337125	OMISS-ART	0.1026306	0.5272083	L′OP-ART
Net	2.434375e−184	0.9824	L′OP-ART	1.000739e−193	0.9946778	OMISS-ART	4.585467e−120	0.8879417	L′OP-ART
XML	1.465917e−196	0.9985042	L′OP-ART	4.243417e−190	0.9901736	OMISS-ART	4.574327e−90	0.8352611	L′OP-ART
EnterpriseManagement	6.76605e−05	0.566427778	L′OP-ART	0.000993654	0.554890278	OMISS-ART	0.298744628	0.517325	L′OP-ART
ID3Manage	9.85122e−12	0.613504167	L′OP-ART	2.21887e−06	0.578894444	OMISS-ART	0.021429965	0.538345833	L′OP-ART
SUT ID	L′OP-ART and DO-ART			OMISS-ART and DO-ART			DO-ART and RT		
	p-value	ES	Better method	p-value	ES	Better method	p-value	ES	Better method
Calendar	3.10158e−18	0.643765278	L′OP-ART	3.84774e−39	0.715569444	OMISS-ART	7.04605e−21	0.345329167	RT
SATM	5.77889e−33	0.300591667	DO-ART	1.41808e−23	0.33315	DO-ART	4.66798e−32	0.696491667	DO-ART
WaveletLibrary	1.90752e−13	0.622565278	L′OP-ART	2.61862e−13	0.621856944	OMISS-ART	6.38389e−06	0.424788889	RT
NSort	3.62033e−98	0.850613889	L′OP-ART	1.77272e−100	0.854797222	OMISS-ART	2.16735e−66	0.213019444	RT
IceChat	1.2178e−155	0.943152778	L′OP-ART	1.1894e−142	0.924002778	OMISS-ART	2.26958e−86	0.171572222	RT
Foundation	3.17586e−17	0.640725	L′OP-ART	1.6959e−15	0.632752778	OMISS-ART	1.3504e−05	0.427438889	RT
Net	4.89702e−58	0.767772222	L′OP-ART	3.79853e−38	0.715308333	OMISS-ART	0.547950208	0.510019444	DO-ART
XML	2.19371e−88	0.832322222	L′OP-ART	3.43061e−56	0.763341667	OMISS-ART	2.86688e−06	0.578036111	DO-ART
EnterpriseManagement	7.39489e−45	0.734298611	L′OP-ART	8.231e−43	0.728669444	OMISS-ART	6.17332e−19	0.351797222	RT
ID3Manage	5.90559e−57	0.765154167	L′OP-ART	1.56266e−43	0.7306625	OMISS-ART	1.08165e−18	0.352843056	RT

Table 8
E-measures for L′OP-ART, OMISS-ART,DO-ART and RT.

	1_V1	1_V2	1_V3	2_V1	2_V2	2_V3	3_V1	3_V2	3_V3	4_V1	4_V2	4_V3	5_V1	5_V2	5_V3	6_V1	6_V2	6_V3	7_V1	7_V2	7_V3	8_V1	8_V2	8_V3	9_V1	9_V2	9_V3	10_V1	10_V2	10_V3	AVG
L′OP-ART	5.00	5.00	5.00	4.00	5.00	4.00	14.99	9.97	6.97	12.93	8.91	6.91	16.92	18.05	19.87	17.21	17.33	17.42	20.79	20.41	19.50	16.28	14.39	12.79	8.00	7.00	8.00	10.56	10.23	11.59	11.83
OMISS-ART	5.00	5.00	5.00	4.00	5.00	4.00	14.99	9.99	6.98	12.90	8.89	6.93	17.39	18.27	19.83	17.39	17.53	17.60	21.05	20.56	19.58	16.33	14.42	12.77	8.00	7.00	8.00	10.48	10.19	11.16	11.87
DO-ART	5.00	5.00	5.00	3.92	4.93	3.95	14.72	9.95	6.96	12.57	8.24	6.83	16.37	16.89	18.95	17.21	15.96	17.08	20.45	19.83	19.12	15.94	13.85	11.64	7.14	6.95	7.97	9.75	9.45	10.85	11.42
RT	5.00	5.00	5.00	3.85	4.87	3.85	14.65	9.66	6.65	12.52	8.52	6.56	15.11	15.96	17.33	15.51	15.86	16.34	19.88	19.38	18.36	15.19	13.06	11.01	8.00	6.84	7.82	9.21	9.23	10.47	11.02

Table 9
Improvements in E-measures achieved by L′OP-ART and OMISS-ART compared with DO-ART.

	1_V1	1_V2	1_V3	2_V1	2_V2	2_V3	3_V1	3_V2	3_V3	4_V1	4_V2	4_V3	5_V1	5_V2	5_V3	6_V1	6_V2	6_V3	7_V1	7_V2	7_V3	8_V1	8_V2	8_V3	9_V1	9_V2	9_V3	10_V1	10_V2	10_V3	AVG
L′OP-ART(%)	0.00	0.00	0.00	3.90	2.67	3.90	2.32	3.21	4.81	3.27	4.58	5.34	11.98	13.10	14.66	10.96	9.27	6.61	4.58	5.31	6.21	7.18	10.18	16.17	0.00	2.34	2.30	14.66	10.83	10.70	6.37
OMISS-ART(%)	0.00	0.00	0.00	3.90	2.67	3.90	2.32	3.42	4.96	3.04	4.34	5.64	15.09	14.47	14.43	12.12	10.53	7.71	5.89	6.09	6.64	7.50	10.41	15.99	0.00	2.34	2.30	13.79	10.40	6.59	6.55
DO-ART(%)	0.00	0.00	0.00	1.82	1.23	2.60	0.48	3.00	4.66	0.40	−3.29	4.12	8.34	5.83	9.35	10.96	0.63	4.53	2.87	2.32	4.14	4.94	6.05	5.72	−10.75	1.61	1.92	5.86	2.38	3.63	2.84

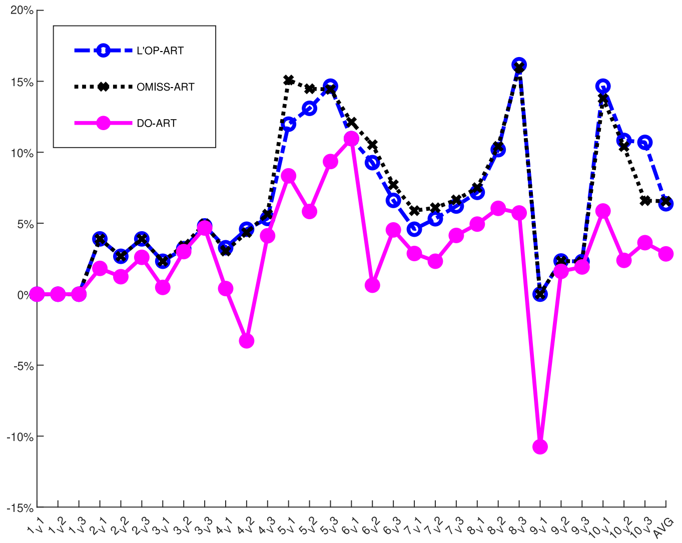


Fig. 6. Improvements in E-measures for L′OP-ART, OMISS-ART, and DO-ART compared with RT.

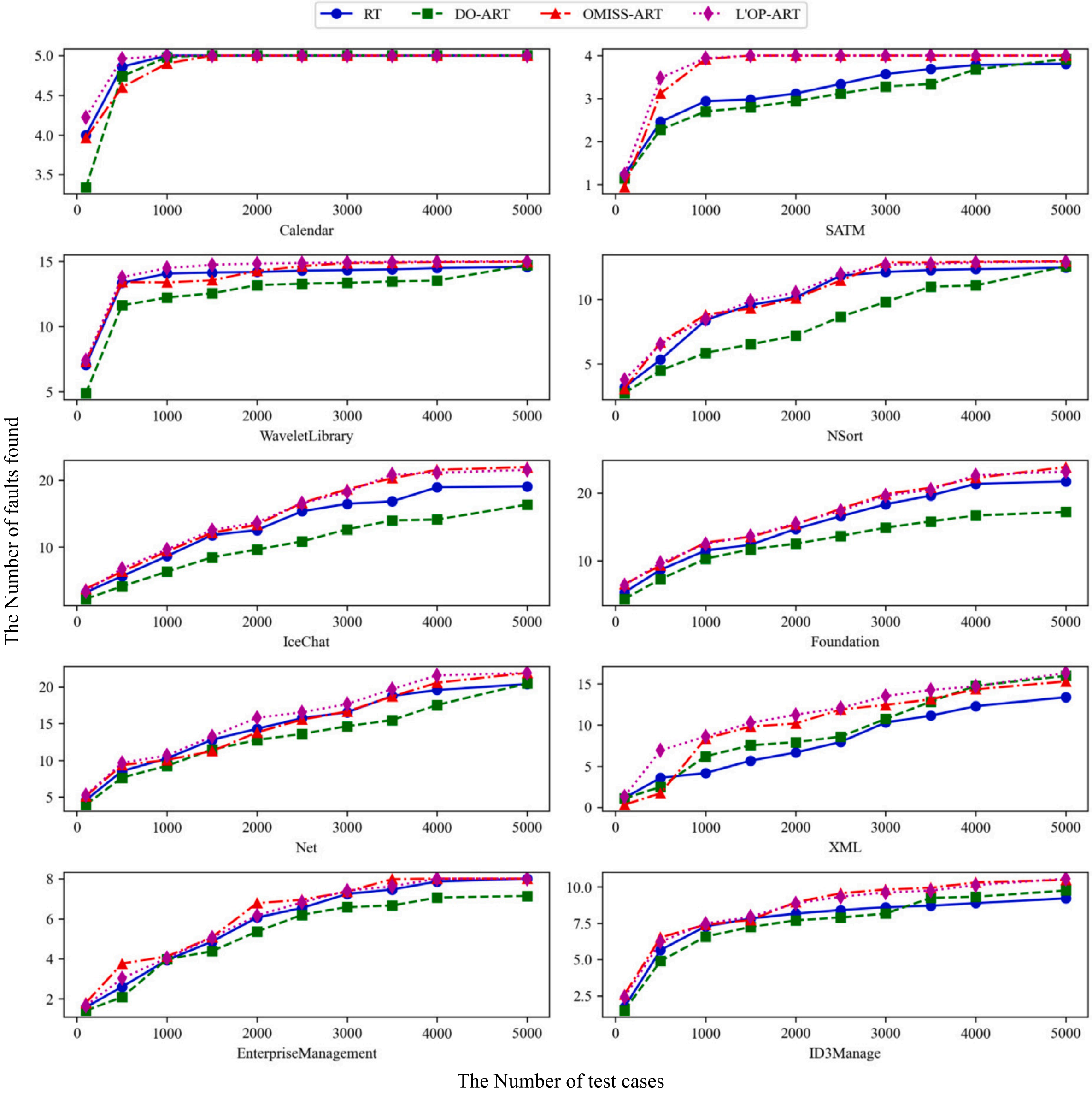


Fig. 7. *Ratio* of RT, DO-ART, OMISS-ART and L′OP-ART under Version 1.

5.4.1. Version 1 subject programs

For the Version 1 of the ten programs, to generate 5000 test cases, RT spends the least amount of average time, only 1.26 s, followed by L′OP-ART, which takes 4.36 s. OMISS-ART and DO-ART, on the other hand, spend a hundred times longer than RT and L′OP-ART. Across all ten programs, the average elapsed time of L′OP-ART is 4.18 times that of RT. OMISS-ART generally takes much more time than other three algorithms, specifically, between 1.39 and 1.77 times more than DO-ART, between 103.47 and 437.52 times more than RT, and between 52.91 and 152.08 times more than L′OP-ART (with averages of 1.52, 314.81 and 81.49 times longer, respectively).

5.4.2. Version 2 subject programs

For the 6 Version 2 subject programs, L′OP-ART takes less than four times the generation time taken by RT, while Foundation, Net, XML and ID3Manage take 5.87, 5.32, 6.03 and 4.01 times RT's generation time, respectively. Across all ten programs, the average elapsed time of L′OP-ART is 3.58 times that of RT. OMISS-ART generally takes much more time than other three algorithms, i.e., between 1.33 and 1.53 times more than DO-ART, between 98.74 and 383.30 times more than RT, and between 52.84 and 137.48 times more than L′OP-ART (with averages of 1.41, 280.96 and 79.55 times longer, respectively).

5.4.3. Version 3 subject programs

For the 7 Version 3 subject programs, L′OP-ART takes less than five times the average generation time of RT, while Foundation, Net, and XML use 5.88, 5.45, and 7.11 times the RT generation time, respectively. Across all ten programs, the average elapsed time of L′OP-ART is 4.04 times that of RT. OMISS-ART generally takes much more time than other three algorithms, between 1.27 and 1.69 times more than DO-ART, between 102.01 and 412.30 times more than RT, and between 47.00 and 129.71 times more than L′OP-ART (with averages of 1.45, 287.79 and 78.06 times longer, respectively).

To further study the significance of the differences in generation time for 5000 test cases for each subject program, Table 11 reports the p-value and effect size (ES) (Arcuri and Briand, 2014) for pairwise comparisons between the techniques. The results show that for all the subject programs, L′OP-ART outperforms OMISS-ART and DO-ART, but not RT. RT always takes the least time to generate 5000 test cases, which is consistent with the theoretical analysis presented in Section 3.

Table 10
Comparison of Time Required to Generate 5000 Random Inputs Using L′OP-ART, OMISS-ART, DO-ART, and RT.

(a) Generation time for Version 1 subject programs

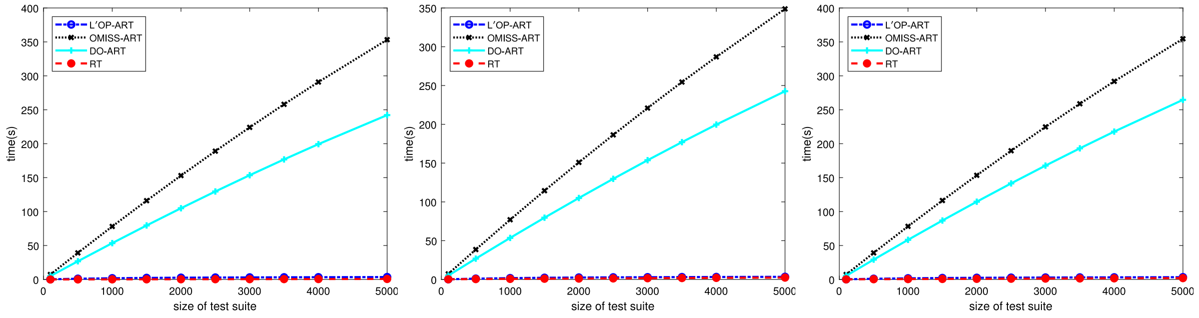
	Generation time (s)				Relative generation time					
	L′OP-ART	OMISS-ART	DO-ART	RT	L′OP-ART	OMISS-ART	DO-ART	OMISS-ART	L′OP-ART	OMISS-ART
					RT	RT	RT	DO-ART	DO-ART	L′P-ART
Calendar	3.54	353.27	242.15	0.75	4.72	471.03	322.87	1.46	0.01	**99.79**
SATM	2.22	218.76	128.74	0.50	4.44	437.52	257.48	1.70	0.02	**98.54**
WaveletLibrary	7.17	410.76	295.12	3.97	1.81	103.47	74.34	1.39	0.02	**57.29**
NSort	1.95	296.56	183.62	1.42	1.37	208.85	129.31	1.62	0.01	**152.08**
IceChat	4.31	329.48	215.45	0.96	4.49	343.21	224.43	1.53	0.02	**76.45**
Foundation	5.15	282.76	159.68	0.86	5.99	328.79	185.67	1.77	0.03	**54.90**
Net	5.42	286.78	195.24	1.26	4.30	227.60	154.95	1.47	0.03	**52.91**
XML	5.46	301.50	214.64	0.81	6.74	372.22	264.99	1.40	0.03	**55.22**
EnterpriseManagement	4.95	376.26	249.43	1.13	4.30	227.60	220.73	1.51	0.02	**52.91**
ID3Manage	3.41	312.73	223.48	0.97	6.74	372.22	230.39	1.40	0.02	**55.22**
Average	4.36	316.89	210.76	1.26	4.18	314.81	206.52	1.52	0.02	**81.49**

(b) Generation time for Version 2 subject programs

	Generation time (s)				Relative generation time					
	L′OP-ART	OMISS-ART	DO-ART	RT	L′OP-ART	OMISS-ART	DO-ART	OMISS-ART	L′OP-ART	OMISS-ART
					RT	RT	RT	DO-ART	DO-ART	L′OP-ART
Calenda	3.4	348.8	242.67	0.91	3.74	383.30	266.67	1.44	0.01	**102.59**
SATM	2.43	221.97	155.15	0.78	3.12	284.58	198.91	1.43	0.02	**91.35**
WaveletLibrary	7.22	408.77	287.05	4.14	1.74	98.74	69.34	1.42	0.03	**56.62**
NSort	2.07	284.58	192.36	1.79	1.16	158.98	107.46	1.48	0.01	**137.48**
IceChat	4.24	286.65	187.78	1.07	3.96	267.90	175.50	1.53	0.02	**67.61**
Foundation	5.22	276.79	203.98	0.89	5.87	311.00	229.19	1.36	0.03	**53.02**
Net	4.95	326.03	244.96	0.93	5.32	350.57	263.40	1.33	0.02	**65.86**
XML	5.19	274.22	195.21	0.86	6.03	318.86	226.99	1.40	0.03	**52.84**
EnterpriseManagement	4.64	382.49	283.91	1.31	3.54	291.98	216.73	1.35	0.02	**82.43**
ID3Manage	3.81	326.48	245.27	0.95	4.01	343.66	258.18	1.33	0.02	**85.69**
Average	4.317	313.678	223.834	1.363	3.85	280.96	201.24	1.41	0.02	**79.55**

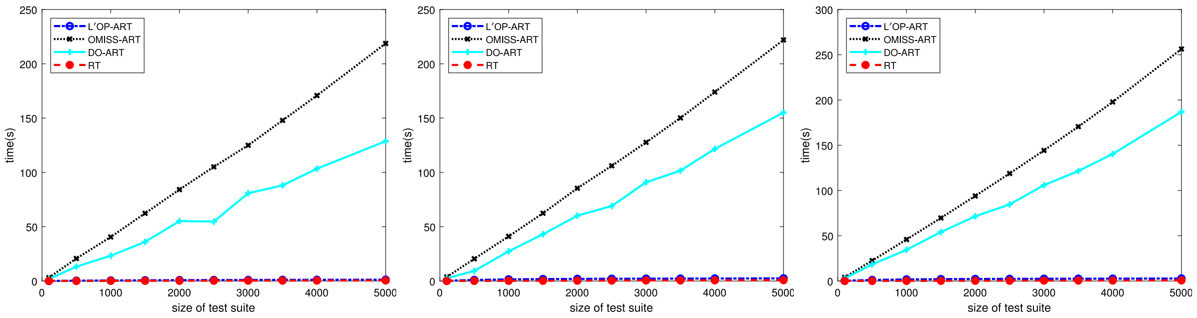
(c) Generation time for Version 3 subject programs

	Generation time (s)				Relative generation time					
	L′OP-ART	OMISS-ART	DO-ART	RT	L′OP-ART	OMISS-ART	DO-ART	OMISS-ART	L′OP-ART	OMISS-ART
					RT	RT	RT	DO-ART	DO-ART	L′OP-ART
Calenda	3.36	354.58	264.62	0.86	3.91	412.30	307.70	1.34	0.01	**105.53**
SATM	2.76	256.33	186.86	0.69	4.00	371.49	270.81	1.37	0.01	**92.87**
WaveletLibrary	6.55	401.93	237.33	3.94	1.66	102.01	60.24	1.69	0.03	**61.36**
NSort	2.31	299.63	236.74	1.57	1.47	190.85	150.79	1.27	0.01	**129.71**
IceChat	4.1	305.47	230.73	1.13	3.63	270.33	204.19	1.32	0.02	**74.50**
Foundation	6.88	385.35	235.89	1.17	5.88	329.36	201.62	1.63	0.03	**56.01**
Net	5.4	278.94	182.46	0.99	5.45	281.76	184.30	1.53	0.03	**51.66**
XML	5.97	280.57	193.59	0.84	7.11	334.01	230.46	1.45	0.03	**47.00**
EnterpriseManagement	5.13	395.17	282.63	1.24	4.14	318.69	227.93	1.40	0.02	**77.03**
ID3Manage	3.71	315.23	207.57	1.18	3.14	267.14	175.91	1.52	0.02	**84.97**
Average	4.617	327.32	225.842	1.361	4.04	287.79	201.39	1.45	0.02	**78.06**



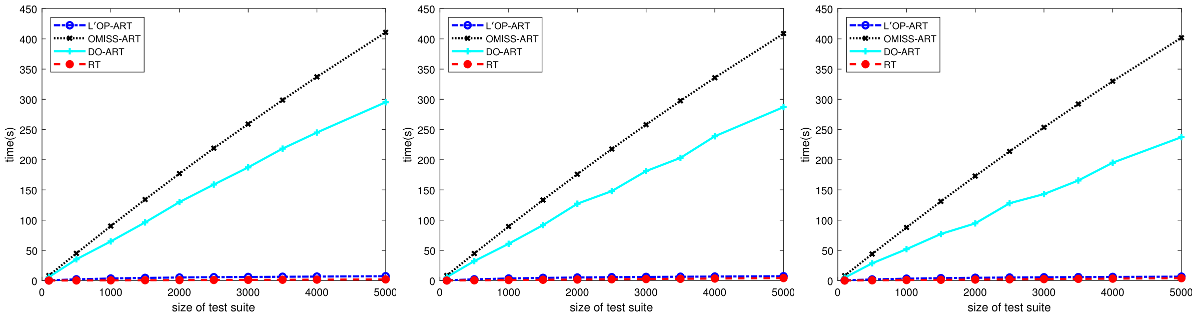
(a) 1_V1 **(b)** 1_V2 **(c)** 1_V3

Fig. 8. Time required to generate test suites for Calendar.



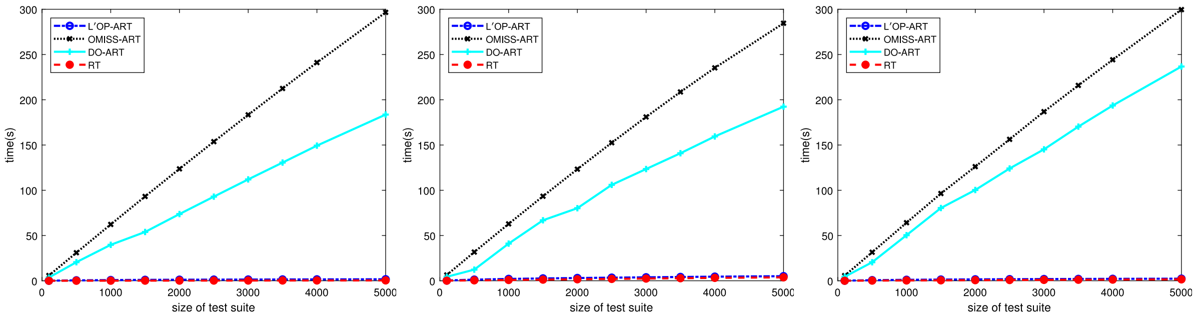
(a) 2_V1 **(b)** 2_V2 **(c)** 2_V3

Fig. 9. Time required to generate test suites for SATM.



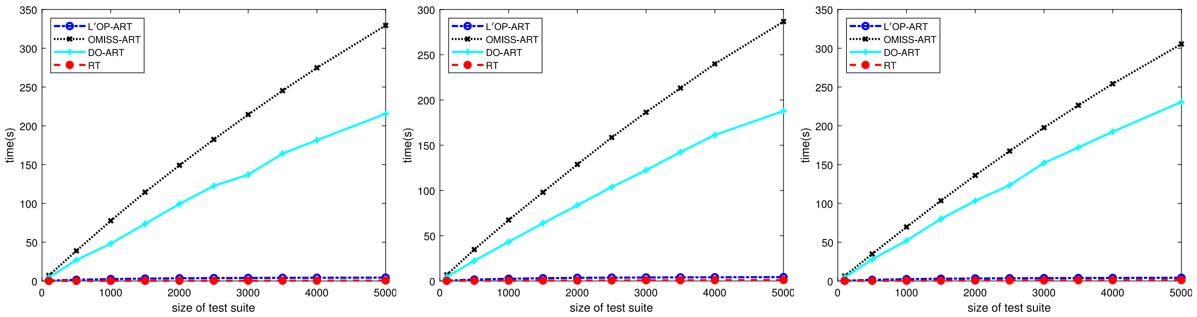
(a) 3_V1 **(b)** 3_V2 **(c)** 3_V3

Fig. 10. Time required to generate test suites for WaveletLibrary.



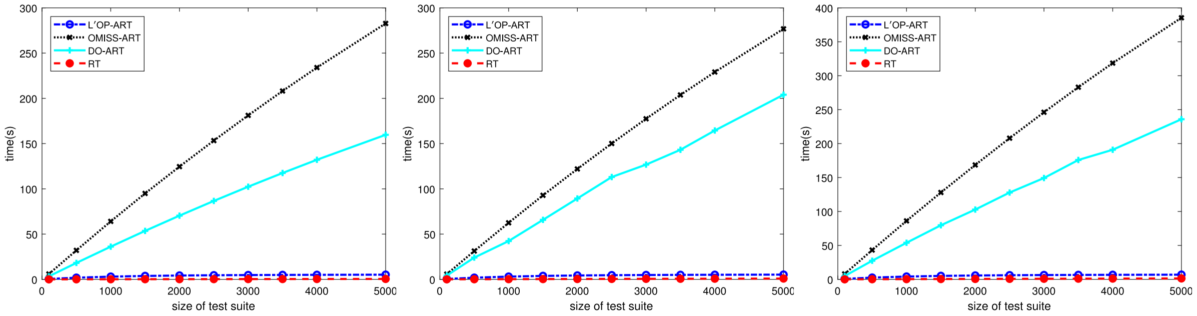
(a) 4_V1 **(b)** 4_V2 **(c)** 4_V3

Fig. 11. Time required to generate test suites for NSort.



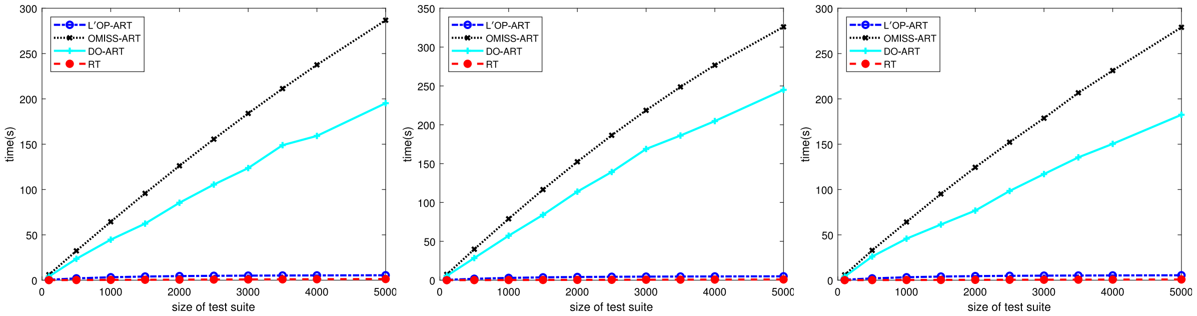
(a) 5_V1 **(b)** 5_V2 **(c)** 5_V3

Fig. 12. Time required to generate test suites for IceChat.



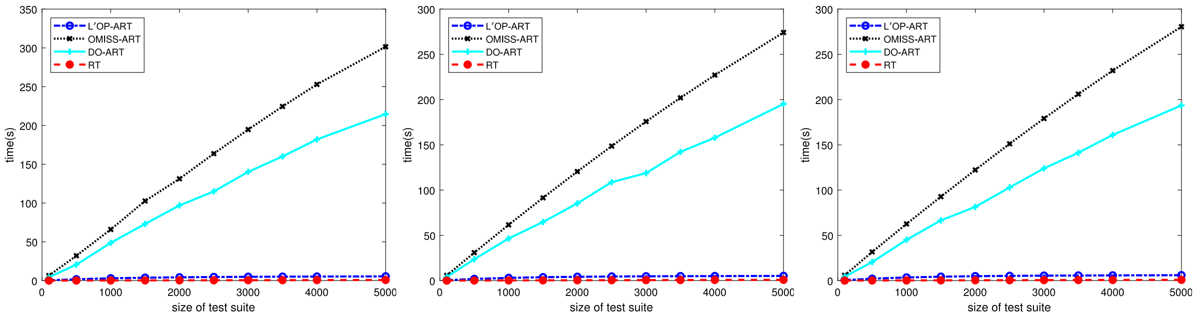
(a) 6_V1 **(b)** 6_V2 **(c)** 6_V3

Fig. 13. Time required to generate test suites for Foundation.



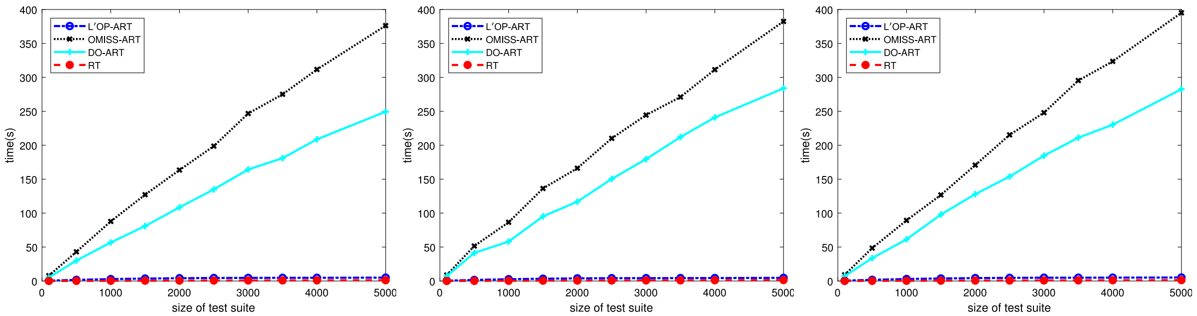
(a) 7_V1 **(b)** 7_V2 **(c)** 7_V3

Fig. 14. Time required to generate test suites for Net.



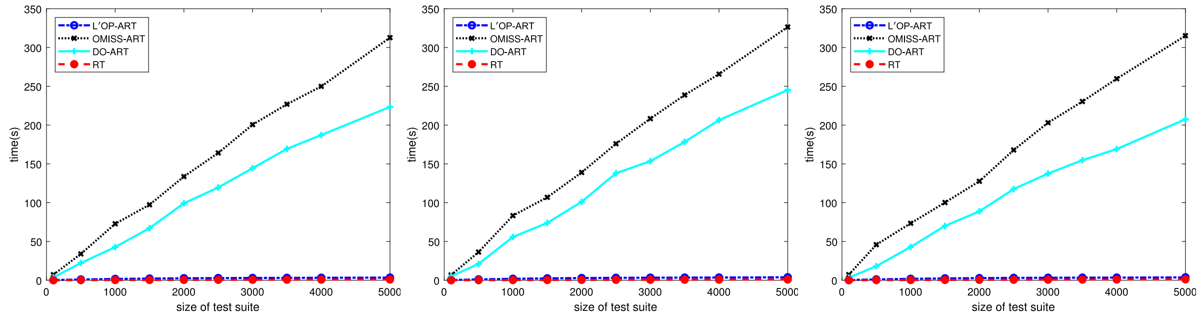
(a) 8_V1 **(b)** 8_V2 **(c)** 8_V3

Fig. 15. Time required to generate test suites for XML.



(a) 9_V1 **(b)** 9_V2 **(c)** 9_V3

Fig. 16. Time required to generate test suites for EnterpriseManagement.



(a) 10_V1 **(b)** 10_V2 **(c)** 10_V3

Fig. 17. Time required to generate test suites for ID3Manage.

Table 11
Comparisons in test case generation time between various pairs of methods using *p*-value and effect size (ES).

SUT ID	L′OP-ART and RT			OMISS-ART and RT			L′OP-ART and OMISS-ART		
	p-value	ES	Better method	p-value	ES	Better method	p-value	ES	Better method
Calendar	1.602008e−121	0.1091389	RT	1.431568e−197	0	RT	1.431568e−197	1	L′OP-ART
SATM	1.113783e−192	0.006294444	RT	1.431566e−197	0	RT	1.431568e−197	1	L′OP-ART
WaveletLibrary	2.056594e−141	0.07786667	RT	1.431456e−197	0	RT	1.431568e−197	1	L′OP-ART
NSort	8.756243e−33	0.3011639	RT	1.431568e−197	0	RT	1.431568e−197	1	L′OP-ART
IceChat	2.64355e−197	0.0003416667	RT	1.430794e−197	0	RT	1.428745e−197	1	L′OP-ART
Foundation	1.44596e−197	5.555556e−06	RT	1.431566e−197	0	RT	1.431566e−197	1	L′OP-ART
EnterpriseManagement	4.97847e−09	0.06	RT	3.01041e−11	0	RT	3.01797e−11	1	L′OP-ART
ID3Manage	1.00667e−08	0.068888889	RT	3.01041e−11	0	RT	3.01797e−11	1	L′OP-ART
SUT ID	L′OP-ART and DO-ART			OMISS-ART and DO-ART			DO-ART and RT		
	p-value	ES	Better method	p-value	ES	Better method	p-value	ES	Better method
Calendar	1.69112e−17	1	L′OP-ART	0.07972745	0.367777778	DO-ART	1.69112e−17	0	RT
SATM	1.54738e−14	0.982222222	L′OP-ART	0.063228045	0.36	DO-ART	1.69112e−17	0	RT
WaveletLibrary	2.03983e−12	0.958888889	L′OP-ART	0.03712506	0.343333333	DO-ART	1.69112e−17	0	RT
NSort	1.13305e−15	0.991111111	L′OP-ART	0.063228045	0.36	DO-ART	3.38225e−17	0.001111111	RT
IceChat	1.69112e−17	1	L′OP-ART	0.05140145	0.353333333	DO-ART	1.69112e−17	0	RT
Foundation	3.56733e−11	0.941111111	L′OP-ART	0.023386606	0.33	DO-ART	1.69112e−17	0	RT
Net	2.47826e−12	0.957777778	L′OP-ART	0.044579003	0.348888889	DO-ART	1.69112e−17	0	RT
XML	4.20088e−11	0.94	L′OP-ART	0.063228045	0.36	DO-ART	1.69112e−17	0	RT
EnterpriseManagement	3.01797e−11	1	L′OP-ART	0.067630299	0.362222222	DO-ART	3.01041e−11	0	RT
ID3Manage	6.69147e−11	0.991111111	L′OP-ART	0.069922725	0.363333333	DO-ART	3.01041e−11	0	RT

6. Threats to validity

In this paper, we primarily introduce a linear-time ART algorithm named L′OP-ART (Linear-time Object-oriented Program Testing by ART), based on a lightweight OMISS metric. In the study, we compare L′OP-ART with two other techniques: random testing and OMISS-ART. The comparison with RT serves as a fundamental and effective approach to demonstrate the improved failure-detection capability of L′OP-ART. OMISS-ART exhibits enhanced failure-detection effectiveness but incurs a computational overhead of $O(n^2)$ when selecting n test cases. The main contribution of L′OP-ART is in mitigating the substantial computational overhead posed by OMISS-ART, while achieving slightly superior efficiency in fault localization compared to OMISS-ART.

For the sake of facilitating the study, we employed the “forgetting” technique (Chan et al., 2006a) and utilized a fixed-size set of previously executed test cases to reduce the runtime of OMISS-ART. Even though we maintained a consistent subset size (20) with the original OMISS-ART study, variations in the subset size could still potentially impact the results. Hence, it becomes imperative to consider the potential influence of subset size on the comparative outcomes of failure-detection capabilities between L′OP-ART and other techniques.

We employed mutation testing to generate potential faults in real-world software and thereby validate the effectiveness of our proposed method. However, it is important to acknowledge that mutation testing might not fully simulate the complexities found in real-world software systems. Real software systems can involve a multitude of intricate factors, such as diverse programming styles, library usage, and various dependency relationships. Hence, it is crucial to recognize the applicability of experimental results in more complex real-world software contexts, as well as the potential limitations of the performance of our proposed method in such intricate software environments.

7. Discussion

In the empirical study, we examined the failure-detection ability of each algorithm using the F-measure (F_m). We also compared the execution time taken by the three algorithms to generate test suites of various sizes for the subject programs.

Our studies show that, as expected, L′OP-ART and OMISS-ART are more effective than RT in the F-measure. L′OP-ART, however, incurs much lower selection overheads than OMISS-ART. This is L′OP-ART’s key advantage.

We found L′OP-ART and OMISS-ART were more effective when testing subject programs that are larger and more complex, e.g., IceChat, Net, and XML. As discussed in Section 1, testing OO software is challenging because of the need to deal with new problems introduced by the unique features of OO languages such as method invocations, inheritance, and polymorphism. Complex OO softwares usually consist of more classes and methods, generating more complex inheritance relationships between classes, and more invocations between the methods in these classes. The distance metrics employed by L′OP-ART and OMISS-ART calculated distances between two inputs involving multiple objects and multiple methods, and the distance was strongly impacted by the number of objects and method invocations. Thus, L′OP-ART and OMISS-ART are more appropriate for use when testing programs with multiple classes. In reality, of course, most OO softwares do contain multiple classes, objects, and methods, with tasks usually completed through invocation of multiple objects and multiple methods (Binder, 1996). Additionally, because OMISS-ART uses a forgetting technique (Chan et al., 2006a) to generate test cases, but not L′OP-ART, L′OP-ART's performance measured by F_m was, as expected, higher than that of OMISS-ART.

We have shown that L′OP-ART, using our proposed distance calculation formulas, outperforms RT in terms of failure-detection effectiveness. The empirical study also showed that L′OP-ART and OMISS-ART have comparable effectiveness. In addition, L′OP-ART incurs much lower computational overheads than OMISS-ART, taking time comparable to RT to generate test cases. For its high failure-detection effectiveness and lower computational overheads, we can conclude that L′OP-ART with the *max-sum* criterion is more cost-effective than OMISS-ART.

8. Related work

Due to the dynamic and complex characteristics of object-oriented programming language (OOP), such as abstraction, encapsulation and polymorphism, automatic test generation is particularly challenging for unit class testing. Many methods have been proposed to solve the problem of automatic test data generation. OO software testing based on RT techniques is a topic of great interest in the software testing community (Pacheco et al., 2007, 2008; Ciupa et al., 2007; Ceccato et al., 2012; Zheng et al., 2010; Godefroid et al., 2005; Pacheco and Ernst, 2007). Random testing is widely applicable, easy to implement, and has no bias against input. However, it does not use any useful information in selecting input, resulting in low code coverage. In order to overcome the disadvantage of randomness, the technology based on dynamic symbolic execution is proposed, such as PEX (Tillmann and Halleux, 2008), a popular tool for unit test. It uses dynamic symbolic execution to generate input values for parametric test cases, but it is limited because classes need complex method sequences. However, these strategies do not meet the applicability and execution speed of random testing. Our approach aims to help evenly spread test cases for OO software based on random testing. Similar to our algorithm, there are other OO software testing algorithms that also use adaptive random testing.

d'Amorim et al. (2006) identified automated unit testing as an important, common and substantial part of software development. They reported that unit testing generally involved two main techniques: (1) automated test generation, which includes random generation and symbolic execution; and (2) automated test classification, which includes tests based on uncaught exceptions and violations of operational models inferred from manually provided tests. They developed a model-based symbolic testing approach for OO unit testing, and conducted empirical studies comparing the performance of their approach with that of three other techniques: model-based random testing, exception-based symbolic testing, and exception-based random testing. They concluded that the techniques were actually complementary, and recommended applying several techniques to the same code under test. Compared with their approach, our approach aims to help evenly spread test cases for OO software based on random testing. Similar to our algorithm, there are other OO software testing algorithms that also use adaptive random testing.

Ciupa et al. (2006, 2008) proposed a distance metric for calculating the distance between two objects, and applied it to adaptive random testing of OO software with a testing strategy called ARTOO that selects as inputs objects with the highest average distance to already used ones. Experimental studies showed that ARTOO outperformed RT in both F-measure and the number of faults uncovered. To test OO software with more diverse test cases, Lin et al. (2009) improved ARTOO by increasing the number of method invocations in each test case. They implemented a divergence-oriented ART tool named ARTGen and showed that it was able to find more faults with fewer test cases than RT. Unfortunately, neither ARTOO nor ARTGen is capable of handling OO test cases that involve a list of objects and a sequence of method invocations. To address this issue, we proposed a sophisticated distance metric called OMISS in Chen et al. (2016). Based on the OMISS metric, we developed a testing approach called OMISS-ART to test OO software. Compared with existing ART algorithms, OMISS-ART has achieved a higher failure-finding ability, but incurs high computational overheads.

There have been a number of attempts to reduce ART algorithms' test case selection overheads (Barus et al., 2016; Mao et al., 2017; Shahbazi et al., 2012), which can be used for both numeric and non-numeric inputs. For numeric inputs, Shahbazi et al. (2012) proposed a linear algorithm named RBCVT (Random Border Centroidal Voronoi Tessellations) to enhance the testing effectiveness by evenly distributing test cases across the input space. Based on a search algorithm, RBCVT reduces the computational complexity from a quadratic to a linear time order. Barus et al. (2016) introduced an ART application for software with non-numeric but structured input formats. Their distance measure, which is based on the concepts of category-partition and choices (Ostrand and Balcer, 1988), allows testers to use their knowledge of the specification or the program structure to identify their categories and choices. However, the failure-detection effectiveness can vary with different testers). Barus et al.'s implementation uses FSCS-ART with the *max-sum* criterion as the test case selection technique, resulting in an algorithm with a linear time overhead. However, neither RBCVT nor the approach designed by Barus et al. are suitable for testing OO software because their test inputs are totally different with those inputs of OO softwares. In this paper, we presented a new cost-effective ART algorithm named L′OP-ART that uses FSCS-ART with the *max-sum* criterion to generate test cases for OO software. Compared with OMISS-ART's quadratic selection overhead, L′OP-ART, which uses innovative distance calculations, incurs only linear selection overheads.

9. Conclusion and future work

At present, several researches, such as ARTOO and ARTGen, have been proposed to use ART for OO software testing, which has already seen success. Chen et al. (2016) proposed the object and method invocation sequence similarity (OMISS) metric to compute the distances among such inputs and a corresponding ART algorithm called OMISS-ART for addressing inadequate consideration of test case distance metrics in ARTOO and ARTGen. However, OMISS-ART has also been found to incur an $O(n^2)$ computational overhead. This indicates its low efficiency and difficulty to handle large-scale testing. To reduce the computational overhead, we first presented the original OMISS dissimilarity metric (Section 2.3) for calculating the distance between two test cases. In order to develop a more cost-effective algorithm, we improved the original calculation formulas for distances between one and n test cases, and also introduced a number of novel, innovative formulas to reduce the computational overheads, and proposed Lightweight OMISS (LW-OMISS) metric. We implemented the LW-OMISS using FSCS-ART with the max-sum criterion and proposed a new testing algorithm called L′OP-ART, and conducted experimental studies to evaluate its effectiveness and efficiency.

The experimental results show that, compared with RT and OMISS-ART, L′OP-ART is much more cost-effective, in both failure detection effectiveness and test case selection overhead. L′OP-ART and OMISS-ART have comparable failure-detection performance, with both outperforming RT in terms of the F-measure. In addition, the selection overhead of L′OP-ART is quite close to that of RT, and far lower than that of OMISS-ART.

In the future, we will extend the distance calculation to include other important data types (such as enumerated types), and explore the features and roles of method arguments in method invocation sequences. We will also investigate other ways to improve L′OP-ART's performance, and examine how changes to the test case dissimilarity metric could impact L′OP-ART's failure-detection effectiveness, with the aims of identifying a more effective distance metric and further improve L′OP-ART's performance. It is worth mentioning that this paper gives a rich proof process at the expense of many as well as complex notations and basic formulas out of not sacrificing rigor. In the future, we will continue to relentlessly explore ways to simplify these formulas and notations to ensure that this paper strikes a better balance between rigor and comprehensibility.

CRediT authorship contribution statement

Jinfu Chen: Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation, Funding acquisition, Conceptualization. **Jingyi Chen:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation. **Lili Zhu:** Writing – original draft, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Chengying Mao:** Writing – review & editing, Validation, Supervision, Software, Formal analysis. **Qihao Bao:** Visualization, Resources, Project administration, Data curation. **Rubing Huang:** Writing – review & editing, Resources, Project administration, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (NSFC grant numbers: 62172194, U1836116, 62172195, 61872167), the China Postdoctoral Science Foundation (Grant number: 2021M691310), the Postdoctoral Science Foundation of Jiangsu Province, China (Grant number: 2021K636C), the Leading-edge Technology Program of Jiangsu Natural Science Foundation (Grant number: BK20202001), the Postgraduate Research & Practice Innovation Program of Jiangsu Province (Grant number: KYCX23_3676), the Science and Technology Development Fund of Macau, Macau SAR under Grant Nos. 0021/2023/RIA1 and 0046/2021/A, and a Faculty Research Grant of Macau University of Science and Technology under Grant FRG-22-103-FIE.

Appendix

A.1. Proof for Lemma 1

Lemma 1. Given a candidate test case c, and any permutation $\langle v_1, v_2, \dots, v_n \rangle$ of $\langle 1, 2, \dots, n \rangle$, we have $s^0_{0k_c} = \tau$, $s^1_{0k_c} = \sum_{i=1}^{\tau} k_{t_{v_i}}$, $s^1_{0g} = \sum_{i=1}^{n} k_{t_{v_i}}$, $s^0_{0g} = n$, and $\sum_{i=\tau+1}^{n} k_{t_{v_i}} = s^1_{0g} - s^1_{0k_c}$, where τ and v_i are defined in Definition (c) of Section 3.2.2.

Proof. Since τ denotes the number of test cases in which the length of the method invocation sequence is shorter than or equal to k_c, it follows $s^0_{0k_c} = \tau$.

Since $\sum_{i=1}^{\tau} k_{t_{v_i}}$ is the sum of method invocation sequences that are shorter than or equal to k_c, it follows $s^1_{0k_c} = \sum_{i=1}^{\tau} k_{t_{v_i}}$.

Since g is the maximum number of methods in the method invocation sequence associated with an element of E, we have $k_{t_{v_1}} \le k_{t_{v_2}} \le \cdots \le k_{t_{v_n}} \le g$. Therefore, it follows $s^1_{0g} = \sum_{i=1}^{n} k_{t_{v_i}}$, $s^0_{0g} = n$.

We also have $\sum_{i=\tau+1}^{n} k_{t_{v_i}} = \sum_{i=1}^{n} k_{t_{v_i}} - \sum_{i=1}^{\tau} k_{t_{v_i}} = s^1_{0g} - s^1_{0k_c}$. ■

A.2. Proof for Lemma 2

Lemma 2. Let c be a candidate test case and uc be the corresponding reduced candidate test case. Let t_{x_i} be any executed test case and ut_{x_i} be the corresponding reduced candidate test case. $|c.MINV \cap t_{x_i}.MINV| = |uc.MINV \cap t_{x_i}.MINV| = |c.MINV \cap ut_{x_i}.MINV| = |uc.MINV \cap ut_{x_i}.MINV|$, and $|c.MINV \cup t_{x_i}.MINV| = |uc.MINV \cup t_{x_i}.MINV| = |c.MINV \cup ut_{x_i}.MINV| = |uc.MINV \cup ut_{x_i}.MINV|$.

Proof. It is straightforward from the definitions of $t.MINV$ and $ut.MINV$. ■

A.3. Proof for Lemma 3

Lemma 3. Given any candidate test case c, and any permutation $\langle x_1, x_2, \dots, x_n \rangle$ of $\langle 1, 2, \dots, n \rangle$, we have $|uc.MINV \cap ut_{x_i}.MINV| = \sum_{l=1}^{k_{uc}} \mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ut_{x_i}}}$.

Proof. By definition, $\mathbf{S}_{x_i}$ only stores the information of $t_{x_i}.MINV$. In particular, for any $l \in \{1, 2, \dots, k_{ut_{x_i}}\}$, $s^{uc.MINV[l]}_{k_{ut_{x_i}}}$ is the number of test cases with the length of method invocation sequence equivalent to $k_{ut_{x_i}}$ and with the $uc.MINV[l]$th method appearing in this sequence. Hence, we have $\mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ut_{x_i}}} = 1$ if $uc.MINV[l]$ appears in $ut_{x_i}.MINV$, and $\mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ut_{x_i}}} = 0$ otherwise. By definition, $|uc.MINV \cap ut_{x_i}.MINV|$ is equal to the number of methods in both $uc.MINV$ and $ut_{x_i}.MINV$. Therefore, it follows Lemma 2 that $|uc.MINV \cap ut_{x_i}.MINV| = \sum_{l=1}^{k_{uc}} \mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ut_{x_i}}}$. ■

A.4. Proof for Lemma 4

Lemma 4. For any $r \in \{1, 2, \dots, ub\}$ and $l \in \{1, 2, \dots, k_{ua_r}\}$, where k_{ua_r} is specified in Definition (a) of Section 3.2.2, we have $\mathbf{S}.s^{uc.MINV[l]}_{k_{ua_r}} = \sum_{j=ua_{r-1}+1}^{ua_r} \mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ua_r}}$.

Proof. By definition, $\mathbf{S}_{x_i}$ only stores the information of $x_i.MINV$. In particular, for any $l \in \{1, 2, \dots, k_{ua_r}\}$, $s^{uc.MINV[l]}_{k_{ua_r}}$ is the number of test cases with the length of method invocation sequence equivalent to k_{ua_r} and with the $uc.MINV[l]$th method appearing in this sequence. Furthermore, by Definition (a),

$$
\begin{aligned}
k_{ut_{x_1}} &= k_{ut_{x_2}} = \dots = k_{ut_{x_{a_1}}} (= k_{ua_1}) \\
&< k_{ut_{x_{ua_1+1}}} = k_{ut_{x_{ua_1+2}}} = \dots = k_{ut_{x_{ua_2}}} (= k_{ua_2}) \\
&< \dots < k_{ut_{x_{ua_{r-1}+1}}} = k_{ut_{x_{ua_{r-1}+2}}} = \dots = k_{ut_{x_{ua_r}}} (= k_{ua_r}) \\
&< \dots < k_{ut_{x_{ua_{ub-1}+1}}} = k_{ut_{x_{ua_{ub-1}+2}}} = \dots = k_{ut_{x_{ua_{ub}}}} (= k_{ua_{ub}}).
\end{aligned}
$$

Hence, we must have $\mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ua_r}} = 0$ for any $i \in \{1, 2, \dots, ua_{ub}\} \backslash \{ua_{r-1} + 1, ua_{r-1} + 2, \dots, ua_r\}$. Therefore, $\mathbf{S}.s^{uc.MINV[l]}_{k_{ua_r}} = \sum_{i=1}^{n} \mathbf{S}_{x_i}.s^{uc.MINV[l]}_{k_{ua_r}} = \sum_{j=ua_{r-1}+1}^{ua_r} \mathbf{S}_{x_j}.s^{uc.MINV[l]}_{k_{ua_r}}$ ■

A.5. Proof for Lemma 5

Lemma 5. Given a candidate test case c, and any $r \in \{1, 2, \ldots, ub\}$, we have $\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cap t_{x_i}.MINV| = \sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}$, where k_{ua_r} is specified in Definition (a) of Section 3.2.2.

Proof.

$$\begin{aligned}
&\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cap t_{x_i}.MINV| \\
&= \sum_{i=ua_{r-1}+1}^{ua_r} |uc.MINV \cap ut_{x_i}.MINV| && \text{(by Lemma 2)} \\
&= \sum_{i=ua_{r-1}+1}^{ua_r} \sum_{l=1}^{k_{uc}} \mathbf{S}_{x_i}.s_{k_{ut_{x_i}}}^{uc.MINV[l]} && \text{(by Lemma 3)} \\
&= \sum_{i=ua_{r-1}+1}^{ua_r} \sum_{l=1}^{k_{uc}} \mathbf{S}_{x_i}.s_{k_{ua_r}}^{uc.MINV[l]} && \text{(by Definition (a))} \\
&= \sum_{l=1}^{k_{uc}} \sum_{i=ua_{r-1}+1}^{ua_r} \mathbf{S}_{x_i}.s_{k_{ua_r}}^{uc.MINV[l]} \\
&= \sum_{l=1}^{k_{uc}} \mathbf{S}.s_{k_{ua_r}}^{uc.MINV[l]} && \text{(by Lemma 4)} \\
&= \sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]} \quad \blacksquare
\end{aligned}$$

A.6. Proof for Lemma 6

Lemma 6. Given a candidate test case c, and any $r \in \{1, 2, \ldots, ub\}$,we have $\frac{\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cup t_{v_i}.MINV|}{ua_r - ua_{r-1}} = k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{ua_r - ua_{j-1}}$, where k_{ua_r} is specified in Definition (a) of Section 3.2.2.

Proof.

$$\begin{aligned}
&\frac{\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cup t_{x_i}.MINV|}{ua_r - ua_{r-1}} \\
&= \frac{\sum_{i=ua_{r-1}+1}^{ua_r} |uc.MINV \cup ut_{x_i}.MINV|}{ua_r - ua_{r-1}} && \text{(by Lemma 2)} \\
&= \frac{\sum_{i=ua_{r-1}+1}^{ua_r} (|uc.MINV| + |ut_{x_i}.MINV| - |uc.MINV \cap ut_{x_i}.MINV|)}{ua_r - ua_{r-1}}
\end{aligned}$$

(immediately after the definition of intersection of two sets)

$$\begin{aligned}
&= k_{uc} + \frac{\sum_{i=ua_{r-1}+1}^{ua_r} |ut_{x_i}.MINV|}{ua_r - ua_{r-1}} \\
&\quad - \frac{\sum_{i=1}^{ua_r - ua_{r-1}} |uc.MINV \cap ut_{x_i}.MINV|}{ua_r - ua_{r-1}} \\
&= k_{uc} + k_{ua_r} - \frac{\sum_{i=ua_{r-1}+1}^{ua_r} |uc.MINV \cap ut_{x_i}.MINV|}{ua_r - ua_{r-1}} \\
&= k_{uc} + k_{ua_r} - \frac{\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cap t_{x_i}.MINV|}{ua_r - ua_{r-1}} && \text{(by Lemma 2)} \\
&= k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{ua_r - ua_{r-1}} && \text{(by Lemma 5)} \quad \blacksquare
\end{aligned}$$

A.7. Proof for Lemma 7

Lemma 7. Given a candidate test case c, and any $ub \in \{1, 2, \ldots, n\}$, we have $\sum_{r=1}^{ub} \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{k_{uc}+k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{s_{k_{ua_r}}^0}} = \sum_{r=1}^{g} \frac{\sum_{l=1}^{k_{uc}} s_r^{uc.MINV[l]}}{k_{uc}+r - \frac{\sum_{l=1}^{k_{uc}} s_r^{uc.MINV[l]}}{\max(s_r^0, 1)}}$, where ua_r and k_{ua_r} are specified in Definition (a) of Section 3.2.2.

Proof. For any $r \in \{ub+1, ub+2, \ldots, g\}$, $s_{k_{ua_r}}^0$ is the number of test cases with the length of method invocation sequence equivalent to k_{ua_r} and with $|t_i.MINV| \neq k_{ua_r}$ for any $i \in \{1, 2, \ldots, n\}$. Hence, $s_{k_{ua_r}}^0 = 0$.

For any $r \in \{ub + 1, ub + 2, \ldots, g\}$ and any $l \in \{1, 2, \ldots, k_{ua_r}\}$, $s_{k_{ua_r}}^{uc.MINV[l]}$ is the number of test cases with the length of method invocation sequence equivalent to k_{ua_r} and with the $uc.MINV[l]$th method appearing in this sequence. Furthermore, by Definition (a),

$$\begin{aligned}
&k_{ut_{x_1}} = k_{ut_{x_2}} = \ldots = k_{ut_{x_{a_1}}} (= k_{ua_1}) \\
&\quad < k_{ut_{x_{ua_1+1}}} = k_{ut_{x_{ua_1+2}}} = \ldots = k_{ut_{x_{ua_2}}} (= k_{ua_2}) \\
&\quad < \ldots < k_{ut_{x_{ua_{r-1}+1}}} = k_{ut_{x_{ua_{r-1}+2}}} = \ldots = k_{ut_{x_{ua_r}}} (= k_{ua_r}) \\
&\quad < \ldots < k_{ut_{x_{ua_{ub-1}+1}}} = k_{ut_{x_{ua_{ub-1}+2}}} = \ldots = k_{ut_{x_{ua_{ub}}}} (= k_{ua_{ub}}).
\end{aligned}$$

Hence, $s_{k_{ua_r}}^{uc.MINV[l]} = 0$ for any $i \in \{1, 2, \ldots, ua_{ub}\} \setminus \{ua_{r-1}+1, ua_{r-1}+2, \ldots, ua_r\}$. Therefore, $\sum_{r=ub+1}^{g} \sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]} = 0$.

Since $ub \leq g$, and since $1 \leq k_{ua_1} < k_{ua_2} < \cdots < k_{ua_{ub}} \leq g$, $\{k_{ua_1}, k_{ua_2}, \ldots, k_{ua_{ub}}\}$ must be a subset of $\{1, 2, \ldots, g\}$. Let $k_{ua_{ub+1}}$, $k_{ua_{ub+2}}, \ldots, k_{ua_g}$ denote the elements in $\{1, 2, \ldots, g\} \setminus \{k_{ua_1}, k_{ua_2}, \ldots, k_{ua_{ub}}\}$ such that $k_{ua_{ub+1}} \neq k_{ua_{ub+2}} \neq \ldots \neq k_{ua_g}$. It trivially follows that $\{k_{ua_1}, k_{ua_2}, \ldots, k_{ua_g}\} = \{1, 2, \ldots, g\}$.

Thus, we have:

$$\begin{aligned}
&\sum_{r=1}^{ub} \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{s_{k_{ua_r}}^0}} \\
&= \sum_{r=1}^{ub} \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{\max(s_{k_{ua_r}}^0, 1)}}
\end{aligned}$$

(because when $r \in \{1, 2, \ldots, ub\}$, $s_{k_{ua_r}}^0 \geq 1$, givingmax$(s_{k_{ua_r}}^0, 1) = s_{k_{ua_r}}^0$; and when $r \in \{ub + 1, ub + 2, \ldots, g\}$, $s_{k_{ua_r}}^0 = 0$, giving $\max(s_{k_{ua_r}}^0, 1) = 1$)

$$\begin{aligned}
&= \sum_{r=1}^{ub} \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{\max(s_{k_{ua_r}}^0, 1)}} + \\
&\quad \sum_{r=ub+1}^{g} \sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}
\end{aligned}$$

(because $\sum_{r=ub+1}^{g} \sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]} = 0$)

$$= \sum_{r=1}^{g} \frac{\sum_{l=1}^{k_{uc}} s_r^{uc.MINV[l]}}{k_{uc} + r - \frac{\sum_{l=1}^{k_{uc}} s_r^{uc.MINV[l]}}{\max(s_r^0, 1)}}$$

(because $\{k_{ua_1}, k_{ua_2}, \ldots, k_{ua_g}\} = \{1, 2, \ldots, g\}$) $\blacksquare$

A.8. Proof for Lemma 8

Lemma 8. Given any candidate test case c and any permutation $\langle v_1, v_2, \ldots, v_n \rangle$ of $\langle 1, 2, \ldots, n \rangle$ as specified in Definition (c), we have $\sum_{l=1}^{k_{t_{v_i}}} \overline{d}_l(t_{v_i}) = \sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}}, l}^{c.MINV[l]}$, and $\sum_{l=1}^{k_c} \overline{d}_l(t_{v_i}) = \sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}}, l}^{c.MINV[l]}$.

Proof. By definition, $\mathbf{S}_{v_i}$ only stores the information of $t_{v_i}.MINV$. In particular, for any $l \in \{1, 2, \ldots, k_{t_{v_i}}\}$, $s_{k_{t_{v_i}}, l}^{t_{v_i}.MINV[l]}$ denotes the number of test cases in which the length of the method invocation sequence is $k_{t_{v_i}}$ and the lth method in this sequence is $t_{v_i}.MINV[l]$. Hence, we have $\mathbf{S}_{v_i}.s_{k_{t_{v_i}}, l}^{c.MINV[l]} = 1$ if $c.MINV[l] = t_{v_i}.MINV[l]$, and $\mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]} = 0$ otherwise. It follows from the definition that $\sum_{l=1}^{k_{t_{v_i}}} \overline{d}_l(t_{v_i}) = \sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}$, and $\sum_{l=1}^{k_c} \overline{d}_l(t_{v_i}) = \sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}$. ■

A.9. Proof for Lemma 9

Lemma 9. Given any candidate test case c, and any $l \in \{1, 2, \dots, k_c\}$, we have $\mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]} =$ $\sum_{i=1}^{n} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]} = \sum_{j=a_{r-1}+1}^{a_r} \mathbf{S}_{v_j}.s_{k_{a_r},l}^{c.MINV[l]}$, where v_{a_r} and k_{a_r} are specified in Definition (c) of Section 3.2.2.

Proof. By definition, $\mathbf{S}_{v_i}$ only stores the information of $t_{v_i}.MINV$. In particular, for any $l \in \{1, 2, \dots, k_{a_r}\}$, $s_{k_{a_r},l}^{c.MINV[l]}$ denotes the number of test cases in which the length of the method invocation sequence is k_{a_r} and the lth method in this sequence is $c.MINV[l]$. Furthermore, by Definition (c),

$$\begin{aligned}
k_{t_{v_1}} &= k_{t_{v_2}} = \dots = k_{t_{v_{a_1}}} (= k_{a_1}) \\
&< k_{t_{v_{a_1+1}}} = k_{t_{v_{a_1+2}}} = \dots = k_{t_{v_{a_2}}} (= k_{a_2}) \\
&< \dots < k_{t_{v_{a_{r-1}+1}}} = k_{t_{v_{a_{r-1}+2}}} = \dots = k_{t_{v_{a_r}}} (= k_{a_r}) \\
&< \dots < k_{t_{v_{a_{b-1}+1}}} = k_{t_{v_{a_{b-1}+2}}} = \dots = k_{t_{v_{a_b}}} (= k_{a_b}).
\end{aligned}$$

Hence, we must have $\mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]} = 0$ for any $i \in \{1, 2, \dots, a_b\} \setminus \{a_{r-1}+1, a_{r-1}+2, \dots, a_r\}$. Therefore, $\mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]} = \sum_{i=1}^{n} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]} = \sum_{j=a_{r-1}+1}^{a_r} \mathbf{S}_{v_j}.s_{k_{a_r},l}^{c.MINV[l]}$. ■

A.10. Proof for Lemma 10

Lemma 10. Let c be any candidate test case.

(a) If $k_c < k_{a_1}$, $\sum_{r=1}^{k_c} \frac{\sum_{l=1}^{r} s_{r,l}^{c.MINV[l]}}{r} = 0$ and $\sum_{r=1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c} = \sum_{r=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{r,l}^{c.MINV[l]}}{k_c}$;

(b) If $k_{a_b} \leq k_c$, $\sum_{r=1}^{b} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}} = \sum_{r=1}^{k_c} \frac{\sum_{l=1}^{r} s_{r,l}^{c.MINV[l]}}{r}$ and $\sum_{r=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{r,l}^{c.MINV[l]}}{k_c} = 0$;

(c) If $k_{a_{\tau\tau}} \leq k_c < k_{a_{\tau\tau+1}}$, $\sum_{r=1}^{\tau\tau} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}} = \sum_{r=1}^{k_c} \frac{\sum_{l=1}^{r} s_{r,l}^{c.MINV[l]}}{r}$ and $\sum_{r=\tau\tau+1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c} = \sum_{r=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{r,l}^{c.MINV[l]}}{k_c}$;

where k_{a_r} $(r \in \{1, 2, \dots, b\})$ and $\tau\tau$ are specified in Definition (c) of Section 3.2.2.

Proof. By definition, $s_{i,l}^{c.MINV[l]}$ denotes the number of test cases in which the length of the method invocation sequence is i and the lth method in this sequence is $c.MINV[l]$. Hence, for any $i \in \{1, 2, \dots, g\}$ such that $i \notin \{k_{a_1}, k_{a_2}, \dots, k_{a_b}\}$, we have $s_{i,l}^{c.MINV[l]} = 0$.

(a) Suppose $k_c < k_{a_1}$.

For any $i \in \{1, 2, \dots, k_c\}$, we must have $i \notin \{k_{a_1}, k_{a_2}, \dots, k_{a_b}\}$, giving $s_{i,l}^{c.MINV[l]} = 0$. Therefore, $\sum_{i=1}^{k_c} \frac{\sum_{l=1}^{i} s_{i,l}^{c.MINV[l]}}{i} = 0$.

For any $i \in \{k_c+1, k_c+2, \dots, g\}$ such that $i \notin \{k_{a_1}, k_{a_2}, \dots, k_{a_b}\}$, we have $s_{i,l}^{c.MINV[l]} = 0$. Thus, $\sum_{i=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{i,l}^{c.MINV[l]}}{k_c} = \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c}$.

(b) Suppose $k_{a_b} \leq k_c$.

For any $i \in \{1, 2, \dots, k_c\}$ such that $i \notin \{k_{a_1}, k_{a_2}, \dots, k_{a_b}\}$, we have $s_{i,l}^{c.MINV[l]} = 0$. Therefore, $\sum_{i=1}^{k_c} \frac{\sum_{l=1}^{i} s_{i,l}^{c.MINV[l]}}{i} = \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$.

For any $i \in \{k_c+1, k_c+2, \dots, g\}$, we must have $i \notin \{k_{a_1}, k_{a_2}, \dots, k_{a_b}\}$, giving $s_{i,l}^{c.MINV[l]} = 0$. Thus, $\sum_{i=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{i,l}^{c.MINV[l]}}{k_c} = 0$.

(c) Suppose $k_{a_{\tau\tau}} \leq k_c < k_{a_{\tau\tau+1}}$.

Since $1 \leq k_{a_1} < k_{a_2} < \cdots < k_{a_{\tau\tau}} \leq k_c < k_{a_{\tau\tau+1}} < k_{a_{\tau\tau+2}} < \cdots < k_{a_b} \leq g$, $\{k_{a_1}, k_{a_2}, \dots, k_{a_{\tau\tau}}\}$ must be a subset of $\{1, 2, \dots, k_c\}$, and $\{k_{a_{\tau\tau+1}}, k_{a_{\tau\tau+2}}, \dots, k_{a_b}\}$ must be a subset of $\{k_c+1, k_c+2, \dots, g\}$.

For any $i \in \{1, 2, \dots, k_c\}$ such that $i \notin \{k_{a_1}, k_{a_2}, \dots, k_{a_{\tau\tau}}\}$, we must have $i \notin \{k_{a_{\tau\tau+1}}, k_{a_{\tau\tau+2}}, \dots, k_{a_b}\}$ as well, giving $s_{i,l}^{c.MINV[l]} = 0$. Therefore, $\sum_{i=1}^{k_c} \frac{\sum_{l=1}^{i} s_{i,l}^{c.MINV[l]}}{i} = \sum_{r=1}^{\tau\tau} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$.

For any $i \in \{k_c+1, k_c+2, \dots, g\}$ such that $i \notin \{k_{a_{\tau\tau+1}}, k_{a_{\tau\tau+2}}, \dots, k_{a_b}\}$, we must have $i \notin \{k_{a_1}, k_{a_2}, \dots,$ $k_{a_{\tau\tau}}\}$ as well, giving $s_{i,l}^{c.MINV[l]} = 0$. Thus, $\sum_{i=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{i,l}^{c.MINV[l]}}{k_c} = \sum_{r=\tau\tau+1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c}$. ■

A.11. Proof for Theorem 1

Theorem 1. *Consider an executed test set $E = \{t_1, t_2, \dots, t_n\}$, the associated global method innovation structure $\mathbf{S}$, and a candidate test case c. The three types of method invocation sequence distances have the following properties:*

$$sum\_LenD(c.MINV,\ E.MINV) = (2 * s_{0k_c}^{0} - s_{0g}^{0}) * k_c + s_{0g}^{1} - 2 * s_{0k_c}^{1} \tag{A.16a}$$

$$sum\_MsD(c.MINV,\ E.MINV) = \begin{cases} s_0^0 - \sum_{i=1}^{g} \frac{\sum_{l=1}^{k_{uc}} s_i^{uc.MINV[l]}}{k_{uc}+i-\frac{\sum_{l=1}^{k_{uc}} s_i^{uc.MINV[l]}}{max(s_i^0,1)}} & \textit{if } k_c > 0 \\ 0 & \textit{otherwise} \end{cases} \tag{A.16b}$$

$$sum\_SD(c.MINV,\ E.MINV) = \begin{cases} s_0^0 - (\sum_{l=1}^{k_c} \frac{\sum_{i=1}^{l} s_{l,\ i}^{c.MINV[i]}}{l} + \sum_{l=k_c+1}^{g} \frac{\sum_{i=1}^{k_c} s_{l,\ i}^{c.MINV[i]}}{k_c}) & \textit{if } k_c > 0 \\ 0 & \textit{otherwise} \end{cases} \tag{A.16c}$$

Proof. (1) **Proof of** Eq. (A.16a).

$$\begin{aligned}
&sum\_LenD(c.MINV,\ E.MINV) \\
&= \sum_{i=1}^{n} LenD(c.MINV,\ t_i.MINV) \\
&= \sum_{i=1}^{n} |k_c - k_{t_i}| \\
&= \sum_{i=1}^{n} |k_c - k_{t_{v_i}}|
\end{aligned}$$

(because $\langle v_1, v_2, \dots, v_n \rangle$ is a permutation of $\langle 1, 2, \dots, n \rangle$, as defined in Definition (c) of Section 3.2.2)

$= \sum_{i=1}^{\tau}(k_c - k_{t_{v_i}}) + \sum_{i=\tau+1}^{n}(k_{t_{v_i}} - k_c)$($\tau$ is defined as in Definition (c) of Section 3.2.2)

$$\begin{aligned}
&= \tau * k_c - \sum_{i=1}^{\tau} k_{t_{v_i}} + \sum_{i=\tau+1}^{n} k_{t_{v_i}} - (n - \tau) * k_c \\
&= s_{0k_c}^{0} * k_c - s_{0k_c}^{1} + (s_{0g}^{1} - s_{0k_c}^{1}) - (s_{0g}^{0} - s_{0k_c}^{0}) * k_c \qquad \text{(by Lemma 1)} \\
&= (2 * s_{0k_c}^{0} - s_{0g}^{0}) * k_c + s_{0g}^{1} - 2 * s_{0k_c}^{1}.
\end{aligned}$$

Thus, Eq. (A.16a) is proved.

(2) **Proof of** Eq. (A.16b).

(a) Suppose $k_c > 0$.

$sum\_MsD(c.MINV,\ E.MINV)$

$$= s_0^0 - \sum_{r=1}^{ub} \frac{\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cap t_{x_i}.MINV|}{\frac{\sum_{i=ua_{r-1}+1}^{ua_r} |c.MINV \cup t_{x_i}.MINV|}{ua_r - ua_{r-1}}} \qquad \text{(by Eq. (12))}$$

$$= s_0^0 - \sum_{r=1}^{ub} \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{ua_r - ua_{r-1}}} \qquad \text{(by Lemmas 2, 5, and 6)}$$

$$= s_0^0 - \sum_{r=1}^{ub} \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{k_{uc} + k_{ua_r} - \frac{\sum_{l=1}^{k_{uc}} s_{k_{ua_r}}^{uc.MINV[l]}}{s_{k_{ua_r}}^0}}$$

(because $(ua_r - ua_{r-1})$ is the number of test case of which the length of method invocation sequence is k_{ua_r}, and it follows the)

$$s_{k_{ua_r}}^0 = ua_r - ua_{r-1}$$

$$= s_0^0 - \sum_{r=1}^{g} \frac{\sum_{l=1}^{k_{uc}} s_r^{uc.MINV[l]}}{k_{uc} + r - \frac{\sum_{l=1}^{k_{uc}} s_r^{uc.MINV[l]}}{\max(s_r^0, 1)}} \qquad \text{(by Lemma 7)}$$

$$= s_0^0 - \sum_{l=1}^{g} \frac{\sum_{r=1}^{k_{uc}} s_l^{uc.MINV[r]}}{k_{uc} + l - \frac{\sum_{r=1}^{k_{uc}} s_l^{uc.MINV[r]}}{\max(s_l^0, 1)}}$$

(b) Suppose $k_c = 0$.

Therefore, $sum\_MsD(c.MINV, E.MINV) = \sum_{i=1}^{n} MsD(c.MINV, t_i.MINV) = 0$

Thus, Eq. (A.16b) is proved.

(3) **Proof of** Eq. (A.16c).

(a) Suppose $k_c > 0$.

$$sum\_SD(c.MINV, E.MINV)$$

$$= \sum_{i=1}^{n} SD(c.MINV, t_i.MINV)$$

$$= \sum_{i=1}^{n} \begin{cases} \frac{\sum_{l=1}^{\min(k_c,\ k_{t_i})} d_l(t_i)}{\min(k_c,\ k_{t_i})} & \text{if } k_{t_i} > 0 \\ 0 & \text{otherwise} \end{cases}$$

(by Eq. (2) and Definition (b) of Section 3.2.2)

$$= \sum_{i=1}^{n} \begin{cases} \frac{\sum_{l=1}^{\min(k_c,\ k_{t_{v_i}})} d_l(t_{v_i})}{\min(k_c,\ k_{t_{v_i}})} & \text{if } k_{t_{v_i}} > 0 \\ 0 & \text{otherwise} \end{cases}$$

(because $\langle v_1, v_2, \ldots, v_n \rangle$ is a permutation of $\langle 1, 2, \ldots, n \rangle$)

Now, let us consider the following three possible cases.

Subcase (i). Assume that $0 \le k_c < k_{t_{v_1}}$. There is $\min(k_c,\ k_{t_{v_i}}) = k_c$. Thus, we have

$$\sum_{i=1}^{n} \frac{\sum_{l=1}^{\min(k_c,\ k_{t_{v_i}})} d_l(t_{v_i})}{\min(k_c,\ k_{t_{v_i}})}$$

$$= \sum_{i=1}^{n} \frac{k_c - \sum_{l=1}^{k_c} \overline{d}_l(t_{v_i})}{k_c} \qquad \text{(by Definition (b) of Section 3.2.2)}$$

$$= s_0^0 - \sum_{i=1}^{n} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c} \qquad \text{(by Lemma 8)}$$

$$= s_0^0 - \sum_{i=1}^{a_b} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c} \qquad \text{(because } a_b = n\text{)}$$

$$= s_0^0 - (\sum_{i=1}^{a_1} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c} + \sum_{i=a_1+1}^{a_2} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c} + \ldots + \sum_{i=a_{b-1}+1}^{a_b} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c})$$

$$= s_0^0 - \sum_{r=1}^{b} \sum_{i=a_{r-1}+1}^{a_r} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c} \text{(where } a_0 = 0\text{)}$$

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{i=a_{r-1}+1}^{a_r} \sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_c}$$

(because the constant common factor $\frac{1}{k_c}$ can be extracted before the second summation formula)

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_c} \sum_{i=a_{r-1}+1}^{a_r} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_c}$$

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_c} \mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]}}{k_c} \qquad \text{(by Lemma 9)}$$

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c}$$

(because $\mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]} = s_{k_{a_r},l}^{c.MINV[l]}$)

$$= s_0^0 - \sum_{i=1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_i},l}^{c.MINV[l]}}{k_c}$$

$$= s_0^0 - (\sum_{i=1}^{k_c} \frac{\sum_{l=1}^{i} s_{i,l}^{c.MINV[l]}}{i} + \sum_{i=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{i,l}^{c.MINV[l]}}{k_c})$$

(because $\sum_{i=1}^{k_c} \frac{\sum_{l=1}^{i} s_{i,l}^{c.MINV[l]}}{i} = 0$ and by Lemma 10(a))

Subcase (ii). Assume further that $0 < k_{t_{v_n}} \le k_c$. There is min(k_c, $k_{t_{v_i}}$)=$k_{t_{v_i}}$ and we have

$$\sum_{i=1}^{n} \frac{\sum_{l=1}^{\min(k_c,\ k_{t_{v_i}})} d_l(t_{v_i})}{\min(k_c,\ k_{t_{v_i}})}$$

$$= \sum_{i=1}^{n} \frac{\sum_{l=1}^{k_{t_{v_i}}} d_l(t_{v_i})}{k_{t_{v_i}}}$$

$$= \sum_{i=1}^{n} \frac{k_{t_{v_i}} - \sum_{l=1}^{k_{t_{v_i}}} \overline{d}_l(t_{v_i})}{k_{t_{v_i}}}$$

(by Definition (b) of Section 3.2.2)

$$= s_0^0 - \sum_{i=1}^{n} \frac{\sum_{l=1}^{k_{t_{v_i}}} \overline{d}_l(t_{v_i})}{k_{t_{v_i}}}$$

$$= s_0^0 - \sum_{i=1}^{n} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}} \qquad \text{(by Lemma 8)}$$

$$= s_0^0 - \sum_{i=1}^{a_b} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}} \qquad \text{(because } a_b = n\text{)}$$

$$= s_0^0 - (\sum_{i=1}^{a_1} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}} + \sum_{i=a_1+1}^{a_2} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}} + \ldots + \sum_{i=a_{b-1}+1}^{a_b} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}})$$

$$= s_0^0 - \sum_{r=1}^{b} \sum_{i=a_{r-1}+1}^{a_r} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}}$$

$$= s_0^0 - \sum_{r=1}^{b} \sum_{i=a_{r-1}+1}^{a_r} \frac{\sum_{l=1}^{k_{a_r}} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

(because$k_{a_r} = k_{t_{v_{a_{r-1}+1}}} = k_{t_{v_{a_{r-1}+2}}} = \ldots = k_{t_{v_{a_r}}}$)

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{i=a_{r-1}+1}^{a_r} \sum_{l=1}^{k_{a_r}} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

(because when r is fixed, k_{a_r} is a constant, and the constant common factor $\frac{1}{k_{a_r}}$ can be extracted before the second summation formula)

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_{a_r}} \sum_{i=a_{r-1}+1}^{a_r} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_{a_r}} \mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}} \qquad \text{(by Lemma 9)}$$

$$= s_0^0 - \sum_{r=1}^{b} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

(because $\mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]} = s_{k_{a_r},l}^{c.MINV[l]}$)

$$= s_0^0 - \sum_{i=1}^{b} \frac{\sum_{l=1}^{k_{a_i}} s_{k_{a_i},l}^{c.MINV[l]}}{k_{a_i}}$$

$$= s_0^0 - (\sum_{i=1}^{k_c} \frac{\sum_{l=1}^{i} s_{i,l}^{c.MINV[l]}}{i} + \sum_{i=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{i,l}^{c.MINV[l]}}{k_c})$$

(because $\sum_{i=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{i,l}^{c.MINV[l]}}{k_c} = 0$ and by Lemma 10(b))

Subcase (iii). Assume further that $0 < k_{t_{v_\tau}} \leq k_c < k_{t_{v_{\tau+1}}}$. Therefore, we have

$$\sum_{i=1}^{n} \frac{\sum_{l=1}^{\min(k_c,\ k_{t_{v_i}})} d_l(t_{v_i})}{\min(k_c,\ k_{t_{v_i}})}$$

$$= \sum_{i=1}^{\tau} \frac{\sum_{l=1}^{k_{t_{v_i}}} d_l(t_{v_i})}{k_{t_{v_i}}} + \sum_{i=\tau+1}^{n} \frac{\sum_{l=1}^{k_c} d_l(t_{v_i})}{k_c}$$

(because $k_{t_{v_1}} \leq k_{t_{v_2}} \leq \ldots \leq k_{t_{v_\tau}} \leq k_c < k_{t_{v_{\tau+1}}} \leq k_{t_{v_{\tau+2}}} \leq \ldots \leq k_{t_{v_n}}$)

$$= \sum_{i=1}^{\tau} \frac{k_{t_{v_i}} - \sum_{l=1}^{k_{t_{v_i}}} \overline{d}_l(t_{v_i})}{k_{t_{v_i}}}$$

$$+ \sum_{i=\tau+1}^{n} \frac{k_c - \sum_{l=1}^{k_c} \overline{d}_l(t_{v_i})}{k_c} \qquad \text{(by Definition (b) of Section 3.2.2)}$$

$$= s_0^0 - (\sum_{i=1}^{\tau} \frac{\sum_{l=1}^{k_{t_{v_i}}} \overline{d}_l(t_{v_i})}{k_{t_{v_i}}} + \sum_{i=\tau+1}^{n} \frac{\sum_{l=1}^{k_c} \overline{d}_l(t_{v_i})}{k_c})$$

$$= s_0^0 - (\sum_{i=1}^{\tau} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}}$$

$$+ \sum_{i=\tau+1}^{n} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c}) \qquad \text{(by Lemma 8)}$$

$$= s_0^0 - (\sum_{i=1}^{a_{\tau\tau}} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}}$$

$$+ \sum_{i=a_{\tau\tau}+1}^{a_b} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c})$$

(because $a_{\tau\tau} = \tau$ and $a_b = n$)

$$= s_0^0 - (\sum_{i=1}^{a_1} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}}$$

$$+ \sum_{i=a_1+1}^{a_2} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}} + \ldots$$

$$+ \sum_{i=a_{\tau\tau-1}+1}^{a_{\tau\tau}} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}}$$

$$+ \sum_{i=a_{\tau\tau}+1}^{a_{\tau\tau+1}} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c}$$

$$+ \sum_{i=a_{\tau\tau+1}+1}^{a_{\tau\tau+2}} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c} + \ldots$$

$$+ \sum_{i=a_{b-1}+1}^{a_b} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c})$$

$$= s_0^0 - (\sum_{r=1}^{\tau\tau} \sum_{i=a_{r-1}+1}^{a_r} \frac{\sum_{l=1}^{k_{t_{v_i}}} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_{t_{v_i}}}$$

$$+ \sum_{r=\tau\tau+1}^{b} \sum_{i=a_{r-1}+1}^{a_r} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{t_{v_i}},l}^{c.MINV[l]}}{k_c})$$

$$= s_0^0 - (\sum_{r=1}^{\tau\tau} \sum_{i=a_{r-1}+1}^{a_r} \frac{\sum_{l=1}^{k_{a_r}} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

$$+ \sum_{r=\tau\tau+1}^{b} \sum_{i=a_{r-1}+1}^{a_r} \frac{\sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_c})$$

(because $k_{a_r} = k_{t_{v_{a_{r-1}+1}}} = k_{t_{v_{a_{r-1}+2}}} = \ldots = k_{t_{v_{a_r}}}$)

$$= s_0^0 - (\sum_{r=1}^{\tau\tau} \frac{\sum_{i=a_{r-1}+1}^{a_r} \sum_{l=1}^{k_{a_r}} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

$$+ \sum_{r=\tau\tau+1}^{b} \frac{\sum_{i=a_{r-1}+1}^{a_r} \sum_{l=1}^{k_c} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_c})$$

(because when r is fixed, k_{a_r} is a constant, and the constant common factor $\frac{1}{k_{a_r}}$ can be extracted before the second summation formula)

$$= s_0^0 - (\sum_{r=1}^{\tau\tau} \frac{\sum_{l=1}^{k_{a_r}} \sum_{i=a_{r-1}+1}^{a_r} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

$$+ \sum_{r=\tau\tau+1}^{b} \frac{\sum_{l=1}^{k_c} \sum_{i=a_{r-1}+1}^{a_r} \mathbf{S}_{v_i}.s_{k_{a_r},l}^{c.MINV[l]}}{k_c})$$

$$= s_0^0 - (\sum_{r=1}^{\tau\tau} \frac{\sum_{l=1}^{k_{a_r}} \mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}}$$

$$+ \sum_{r=\tau\tau+1}^{b} \frac{\sum_{l=1}^{k_c} \mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]}}{k_c}) \qquad \text{(by Lemma 9)}$$

$$= s_0^0 - (\sum_{r=1}^{\tau\tau} \frac{\sum_{l=1}^{k_{a_r}} s_{k_{a_r},l}^{c.MINV[l]}}{k_{a_r}} + \sum_{r=\tau\tau+1}^{b} \frac{\sum_{l=1}^{k_c} s_{k_{a_r},l}^{c.MINV[l]}}{k_c})$$

(because $\mathbf{S}.s_{k_{a_r},l}^{c.MINV[l]} = s_{k_{a_r},l}^{c.MINV[l]}$)

$$= s_0^0 - (\sum_{i=1}^{\tau\tau} \frac{\sum_{l=1}^{k_{a_i}} s_{k_{a_i},l}^{c.MINV[l]}}{k_{a_i}} + \sum_{i=\tau\tau+1}^{b} \frac{\sum_{l=1}^{ck} s_{k_{a_i},l}^{c.MINV[l]}}{k_c})$$

$$= s_0^0 - (\sum_{i=1}^{k_c} \frac{\sum_{l=1}^{i} s_{i,l}^{c.MINV[l]}}{i} + \sum_{i=k_c+1}^{g} \frac{\sum_{l=1}^{k_c} s_{i,l}^{c.MINV[l]}}{k_c}) \qquad \text{(by Lemma 10(c))}$$

(b) Suppose $k_c = 0$.

Therefore, $sum\_SD(c.MINV, \quad E.MINV) = \sum_{i=1}^{n} SD(c.MINV, t_i.MINV) = 0$

Thus, Eq. (A.16c) is proved. ■

A.12. Proof for Lemma 11

Lemma 11. $m_{0k_{p.M}}^{0} = \tau$, $m_{0k_{p.M}}^{1} = \sum_{i=1}^{\tau} k_{q_{v_i}.M}$, $m_{0c}^{1} = \sum_{i=1}^{n} k_{q_{v_i}.M}$, $m_{0c}^{0} = n$, and $\sum_{i=\tau+1}^{n} k_{q_{v_i}.M} = m_{0c}^{1} - m_{0k_{p.M}}^{1}$, where τ and v_i are defined in Definition (d) of Section 3.2.2.

Proof. Since τ denotes the number of objects in which the length of the behavior section is shorter than or equal to $k_{p.M}$, it follows the definition of $m_{0k_{p.M}}^{0}$ that $m_{0k_{p.M}}^{0} = \tau$.

Since $\sum_{i=1}^{\tau} k_{q_{v_i}.M}$ is the sum of the lengths of behavior sections whose lengths are shorter than or equal to $k_{p.M}$, it follows the definition of $m_{0k_{p.M}}^{1}$ that $m_{0k_{p.M}}^{1} = \sum_{i=1}^{\tau} k_{q_{v_i}.M}$.

Since c is the maximum number of methods for an object's associated class, we have $k_{q_1.M} \leq k_{q_2.M} \leq \cdots \leq k_{q_n.M} \leq c$. Therefore, it follows the definitions of m_{0c}^{1} and m_{0c}^{0} that $m_{0c}^{1} = \sum_{i=1}^{n} k_{q_{v_i}.M}$, $m_{0c}^{0} = n$.

We also have $\sum_{i=\tau+1}^{n} k_{q_{v_i}.M} = \sum_{i=1}^{n} k_{q_{v_i}.M} - \sum_{i=1}^{\tau} k_{q_{v_i}.M} = m_{0c}^{1} - m_{0k_{p.M}}^{1}$. ■

A.13. Proof for Lemma 12

Lemma 12. $|p.M \cap q_{x_i}.M| = |up.M \cap q_{x_i}.M| = |p.M \cap uq_{x_i}.M| = |up.M \cap uq_{x_i}.M|$, and $|p.M \cup q_{x_i}.M| = |up.M \cup q_{x_i}.M| = |p.M \cup uq_{x_i}.M| = |up.M \cup uq_{x_i}.M|$.

Proof. It is straightforward after the definitions of $q.M$ and $uq.M$. ■

A.14. Proof for Lemma 13

Lemma 13. $|up.M \cap uq_{x_i}.M| = \sum_{l=1}^{k_{up.M}} \mathbf{M}_{x_i}.m_{k_{uq_{x_i}.M}}^{up.M[l]}$, where $x_i \in \{x_1, x_2, \dots, x_n\}$ with $\langle x_1, x_2, \dots, x_n \rangle$ as defined in Definition (e) of Section 3.2.2.

Proof. By definition, $\mathbf{M}_{x_i}$ only stores the information of $q_{x_i}.MINV$. In particular, for any $l \in \{1, 2, \dots, k_{uq_{x_i}}\}$, $m_{k_{uq_{x_i}}}^{up.M[l]}$ denotes the number of objects that have $k_{uq_{x_i}}$ methods, including the method $up.M[l]$. Hence, we have $\mathbf{M}_{x_i}.m_{k_{uq_{x_i}}}^{up.M[l]} = 1$ if $up.M[l]$ appears in $uq_{x_i}.M$, and $\mathbf{M}_{x_i}.m_{k_{uq_{x_i}}}^{up.M[l]} = 0$ otherwise. By definition, $|up.M \cap uq_{x_i}.M|$ is equal to the number of methods in both $up.M$ and $uq_{x_i}.M$. Therefore, it follows Lemma 2 that $|up.M \cap uq_{x_i}.M| = \sum_{l=1}^{k_{up.M}} \mathbf{M}_{x_i}.m_{k_{uq_{x_i}}}^{up.M[l]}$. ■

A.15. Proof for Lemma 14

Lemma 14. For any $r \in \{1, 2, \dots, ub\}$ and $l \in \{1, 2, \dots, k_{ua_r}\}$, where k_{ua_r} and ub are specified in Definition (e) of Section 3.2.2, we have $\mathbf{M}.m_{k_{ua_r}}^{up.M[l]} = \sum_{i=ua_{r-1}+1}^{ua_r} \mathbf{M}_{x_i}.m_{k_{ua_r}}^{up.M[l]}$.

Proof. As indicated in Definition (e) of Section 3.2.2 that $k_{uq_{x_1}.M} = k_{uq_{x_2}.M} = \cdots = k_{uq_{x_{ua_1}}.M} < k_{uq_{x_{ua_1+1}}.M} = \cdots = k_{uq_{x_{ua_2}}.M} < \dots < k_{ua_j} = k_{uq_{x_{ua_{j-1}+1}}.M} = k_{uq_{x_{ua_{j-1}+2}}.M} = \cdots = k_{uq_{x_{ua_j}}.M} < \dots < k_{uq_{x_{ua_{ub-1}+1}}.M} = \cdots = k_{uq_{x_{ua_{ub}}}.M}$, $\mathbf{M}_{x_i}$ only stores the information of $q_{x_i}.M$, and $m_{k_{ua_j}}^{up.M[l]}$ is the number of objects with the length of behavior section equivalent to k_{ua_j} and with the $uc.MINV[l]$th method appearing in this behavior section. Thus, we have $\mathbf{M}_{x_i}.m_{k_{ua_j}}^{up.M[l]} = 0$ $(i \in \{1, 2, \dots, ua_{j-1}, ua_j+1, ua_j+2, \dots, ua_{ub}\})$. Therefore $\mathbf{M}.m_{k_{ua_j}}^{up.M[l]} = \sum_{i=1}^{n} \mathbf{M}_{x_i}.m_{k_{ua_j}}^{up.M[l]} = \sum_{i=ua_{j-1}+1}^{ua_j} \mathbf{M}_{x_i}.m_{k_{ua_j}}^{up.M[l]}$. ■

A.16. Proof for Lemma 15

Lemma 15. For any j $(1 \leq j \leq ub)$, we have $\sum_{i=ua_{j-1}+1}^{ua_j} |p.M \cap q_{x_i}.M| = \sum_{l=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[l]}$, where k_{ua_j} and ub are defined in Definition (e) of Section 3.2.2.

Proof.

$$\sum_{i=ua_{j-1}+1}^{ua_j} |p.M \cap q_{x_i}.M|$$
$$= \sum_{i=ua_{j-1}+1}^{ua_j} |up.M \cap uq_{x_i}.M| \qquad \text{(by Lemma 12)}$$
$$= \sum_{i=ua_{j-1}+1}^{ua_j} \sum_{l=1}^{k_{up.M}} \mathbf{M}_{x_i}.m_{k_{uq_{x_i}}.M}^{up.M[l]} \qquad \text{(by Lemma 13)}$$
$$= \sum_{i=ua_{j-1}+1}^{ua_j} \sum_{l=1}^{k_{up.M}} \mathbf{M}_{x_i}.m_{k_{ua_j}}^{up.M[l]} \qquad \text{(by Definition (e) of Section 3.2.2)}$$
$$= \sum_{l=1}^{k_{up.M}} \sum_{i=ua_{j-1}+1}^{ua_j} \mathbf{M}_{x_i}.m_{k_{ua_j}}^{up.M[l]}$$
$$= \sum_{l=1}^{k_{up.M}} \mathbf{M}.m_{k_{ua_j}}^{up.M[l]} \qquad \text{(by Lemma 14)}$$
$$= \sum_{l=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[l]} \quad ■$$

A.17. Proof for Lemma 16

Lemma 16. For any j $(1 \leq j \leq ub)$, we have $\frac{\sum_{i=ua_{j-1}+1}^{ua_j} |p.M \cup q_{x_i}.M|}{ua_j - ua_{j-1}} = k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{ua_j - ua_{j-1}}$, where k_{ua_j} and ub are defined in Definition (e) of Section 3.2.2.

Proof.

$$\frac{\sum_{i=ua_{j-1}+1}^{ua_j} |p.M \cup q_{x_i}.M|}{ua_j - ua_{j-1}}$$
$$= \frac{\sum_{i=ua_{j-1}+1}^{ua_j} |up.M \cup uq_{x_i}.M|}{ua_j - ua_{j-1}} \qquad \text{(by Lemma 12)}$$
$$= \frac{\sum_{i=ua_{j-1}+1}^{ua_j} (|up.M| + |uq_{x_i}.M| - |up.M \cap uq_{x_i}.M|)}{ua_j - ua_{j-1}}$$

(immediately after the definition of intersection of two sets)

$$= k_{up.M} + \frac{\sum_{i=ua_{j-1}+1}^{ua_j} |uq_{x_i}.M|}{ua_j - ua_{j-1}} - \frac{\sum_{i=1}^{ua_j - ua_{j-1}} |up.M \cap uq_{x_i}.M|}{ua_j - ua_{j-1}}$$
$$= k_{up.M} + k_{ua_j} - \frac{\sum_{i=ua_{j-1}+1}^{ua_j} |up.M \cap uq_{x_i}.M|}{ua_j - ua_{j-1}}$$
$$= k_{up.M} + k_{ua_j} - \frac{\sum_{i=ua_{j-1}+1}^{ua_j} |p.M \cap q_{x_i}.M|}{ua_j - ua_{j-1}} \qquad \text{(by Lemma 12)}$$
$$= k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{ua_j - ua_{j-1}} \qquad \text{(by Lemma 15)} \quad ■$$

A.18. Proof for Lemma 17

Lemma 17. $\sum_{j=1}^{ub} \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{m_{k_{ua_j}}^{0}}} = \sum_{j=1}^{c} \frac{\sum_{i=1}^{k_{up.M}} m_{j}^{up.M[i]}}{k_{up.M} + j - \frac{\sum_{i=1}^{k_{up.M}} m_{j}^{up.M[i]}}{\max(m_j^0, 1)}}$,

where k_{ua_j} is defined in Definition (e) of Section 3.2.2, and c is the maximum number of methods for an object's associated class.

Proof. Since $1 \leq k_{ua_1} < k_{ua_2} < \cdots < k_{ua_{ub}} \leq c$, $ub \leq c$, and $\{k_{ua_1}, k_{ua_2}, \dots, k_{ua_{ub}}\}$ is a subset of $\{1, 2, \dots, c\}$. Define $1 \leq k_{ua_{ub+1}} < k_{ua_{ub+2}} < \cdots < k_{ua_c} \leq c$, $k_{ua_{ub+x}} \in \{1, 2, \dots, c\}$, $k_{ua_{ub+x}} \notin \{k_{ua_1}, k_{ua_2}, \dots, k_{ua_{ub}}\}$ $(x \in \{1, 2, \dots, c - ub\})$. Therefore $\{k_{ua_1}, k_{ua_2}, \dots, k_{ua_g}\} = \{1, 2, \dots, g\}$. Since $m_{k_{ua_j}}^{up.M[i]}$ denotes the number of objects with the length of behavior section being k_{ua_j}, and with method $up.M[i]$ appearing in this section; $m_{k_{ua_j}}^{0}$ denotes the number of objects with the length of behavior section being k_{ua_j}, and all $|q_l.M| \neq k_{ua_{ub+x}}$ $(l \in \{1, 2, \dots, n\},\ x \in \{1, 2, \dots, c - ub\})$. Therefore, $m_{k_{ua_j}}^{up.M[i]} = 0$, $m_{k_{ua_j}}^{0} = 0$ $(j \in \{ub+1, ub+2, \dots, c\})$, $\sum_{j=ub+1}^{c} \sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]} = 0$.

Since $\{k_{ua_1}, k_{ua_2}, \dots, k_{ua_c}\} = \{1, 2, \dots, c\}$, thus we have

$$\sum_{j=1}^{ub} \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{m_{k_{ua_j}}^0}}$$

$$= \sum_{j=1}^{ub} \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{\max(m_{k_{ua_j}}^0, 1)}}$$

(because when $j \in \{1, 2, \dots, ub\}$, $s_{k_{ua_j}}^0 \geq 1$, $\max(m_{k_{ua_j}}^0, 1) = m_{k_{ua_j}}^0$; when $j \in \{ub+1, ub+2, \dots, c\}$, $m_{k_{ua_j}}^0 = 0$, $\max(m_{k_{ua_j}}^0, 1) = 1$)

$$= \sum_{j=1}^{ub} \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{\max(m_{k_{ua_j}}^0, 1)}} + \sum_{j=ub+1}^{c}$$

$$\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}$$

(because $\sum_{j=ub+1}^{c} \sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]} = 0$, as previously proved)

$$= \sum_{j=1}^{c} \frac{\sum_{i=1}^{k_{up.M}} m_j^{up.M[i]}}{k_{up.M} + j - \frac{\sum_{i=1}^{k_{up.M}} m_j^{up.M[i]}}{\max(m_j^0, 1)}}.$$

(because $\{k_{ua_1}, k_{ua_2}, \dots, k_{ua_g}\} = \{1, 2, \dots, g\}$,as previously proved)

■

A.19. Proof for Theorem 2

Theorem 2. *Given an object set Q with n objects, its associated **M**, and a candidate object p, let $up.M$ represent the reduced behavior section of p (created by removing the repetitive methods in $p.M$), $k_{p.M}$ and $k_{up.M}$ represent the lengths of $p.M$ and $up.M$ respectively, and $up.M[j]$ represent the jth method in $up.M$ ($1 \leq j \leq k_{up.M}$). Then, we have*

$$sum\_BLenD(p.M,\ Q.M) = (2 * m_{k_{p.M}}^0 - m_c^0) * k_{p.M} + m_c^1 - 2 * m_{k_{p.M}}^1 \tag{A.18a}$$

$$sum\_BMsD(p.M,\ Q.M) = \begin{cases} m_0^0 - \sum_{i=1}^{c} \frac{\sum_{j=1}^{k_{up.M}} m_i^{up.M[j]}}{k_{up.M} + i - \frac{\sum_{j=1}^{k_{up.M}} m_i^{up.M[j]}}{\max(m_i^0, 1)}} & \textit{if } p.M \textit{ is not null} \\ m_0^0 & \textit{otherwise} \end{cases} \tag{A.18b}$$

Proof. (1) **Proof of** Eq. (A.18a).

$$sum\_BLenD(p.M,\ Q.M)$$
$$= \sum_{i=1}^{n} BLenD(p.M,\ q_i.M)$$
$$= \sum_{i=1}^{n} |k_{p.M} - k_{q_i.M}|$$
$$= \sum_{i=1}^{n} |k_{p.M} - k_{q_{v_i}.M}|$$

(because $\langle v_1, v_2, \dots, v_n \rangle$ is a permutation of $\langle 1, 2, \dots, n \rangle$, which are defined in Definition (d) of Section 3.2.2)

$$= \sum_{i=1}^{\tau} (k_{p.M} - k_{q_{v_i}.M}) + \sum_{i=\tau+1}^{n} (k_{q_{v_i}.M} - k_{p.M})$$

(τ is defined as in Definition (d) of Section 3.2.2)

$$= \tau * k_{p.M} - \sum_{i=1}^{\tau} k_{q_{v_i}.M} + \sum_{i=\tau+1}^{n} k_{q_{v_i}.M} - (n - \tau) * k_{p.M}$$
$$= m_{0k_{p.M}}^0 * k_{p.M} - m_{0k_{p.M}}^1 + (m_{0c}^1 - m_{0k_{p.M}}^1) - (m_{0c}^0 - m_{0k_{p.M}}^0) * k_{p.M}$$

(by Lemma 11)

$$= (2 * m_{0k_{p.M}}^0 - m_{0c}^0) * k_{p.M} + m_{0c}^1 - 2 * m_{0k_{p.M}}^1.$$

Thus, Eq. (A.18a) is proved.

(2) **Proof of** Eq. (A.18b).

(a) Suppose $p.M$ is not null.

$$sum\_BMsD(p.M,\ Q.M)$$

$$= m_0^0 - \sum_{j=1}^{b} \frac{\sum_{i=a_{j-1}+1}^{a_j} |p.M \cap q_{x_i}.M|}{\frac{\sum_{i=a_{j-1}+1}^{a_j} |p.M \cup q_{x_i}.M|}{a_j - a_{j-1}}} \quad \text{(by Eq. (13))}$$

$$= m_0^0 - \sum_{j=1}^{ub} \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{ua_j - ua_{j-1}}} \quad \text{(by Lemmas 12, 15, and 16)}$$

$$= m_0^0 - \sum_{j=1}^{ub} \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{k_{up.M} + k_{ua_j} - \frac{\sum_{i=1}^{k_{up.M}} m_{k_{ua_j}}^{up.M[i]}}{m_{k_{ua_j}}^0}}$$

(because $(ua_j - ua_{j-1})$ is the number of objects of with the length of behavior section is k_{ua_j}, and it follows from the definition of $m_{k_{ua_j}}^0$ that $m_{k_{ua_j}}^0 = ua_j - ua_{j-1}$)

$$= m_0^0 - \sum_{j=1}^{c} \frac{\sum_{i=1}^{k_{up.M}} m_j^{up.M[i]}}{k_{up.M} + j - \frac{\sum_{i=1}^{k_{up.M}} m_j^{up.M[i]}}{\max(m_j^0, 1)}} \quad \text{(by Lemma 17)}$$

$$= m_0^0 - \sum_{i=1}^{c} \frac{\sum_{j=1}^{k_{up.M}} m_i^{up.M[j]}}{k_{up.M} + i - \frac{\sum_{j=1}^{k_{up.M}} m_i^{up.M[j]}}{\max(m_i^0, 1)}}$$

(b) Suppose $k_{p.M} = 0$.

Therefore, $sum\_BMsD(p.M,\ Q.M) = \sum_{i=1}^{n} BMsD(p.M,\ q_i.M) = 2 * m_0^0$

Thus, Eq. (A.18b) is proved. ■

A.20. Proof for Lemma 18

Lemma 18. $n_{0k_{p.A.N}}^0 = \tau$, $n_{0k_{p.A.N}}^1 = \sum_{i=1}^{\tau} k_{q_{v_i}.A.N}$, $n_{0r}^1 = \sum_{i=1}^{n} k_{q_{v_i}.A.N}$, $n_{0r}^0 = n$, and $\sum_{i=\tau+1}^{n} k_{q_{v_i}.A.N} = n_{0r}^1 - n_{0k_{p.A.N}}^1$, where τ and v_i are defined in Definition (f) of Section 3.2.2.

Proof. Since τ denotes the number of objects in which the length of the behavior section whose length is shorter than or equal to $k_{p.A.N}$, it follows the definition of $n_{0k_{p.A.N}}^0$ that $n_{0k_{p.A.N}}^0 = \tau$.

Since $\sum_{i=1}^{\tau} k_{q_{v_i}.A.N}$ is the sum of the lengths of behavior sections whose lengths are shorter than or equal to $k_{p.A.N}$, it follows the definition of $n_{0k_{p.A.N}}^1$ that $n_{0k_{p.A.N}}^1 = \sum_{i=1}^{\tau} k_{q_i.A.N}$.

Since r is the maximum number of non-reference attributes for any objects associated class, we have $k_{q_{v_1}.A.N} \leq k_{q_{v_2}.A.N} \leq \dots \leq k_{q_{v_n}.A.N} \leq r$. Therefore, it follows from the definitions of n_{0r}^1 and n_{0r}^0 that $n_{0r}^1 = \sum_{i=1}^{n} k_{q_{v_i}.A.N}$, $n_{0r}^0 = n$.

We also have $\sum_{i=\tau+1}^{n} k_{q_{v_i}.A.N} = \sum_{i=1}^{n} k_{q_{v_i}.A.N} - \sum_{i=1}^{\tau} k_{q_{v_i}.A.N} = n_{0r}^1 - n_{0k_{p.A.N}}^1$. ■

A.21. Proof for Lemma 19

Lemma 19. $|p.A.N \cap q_i.A.N| = |up.A.N \cap uq_i.A.N| = \sum_{l=1}^{k_{up.A.N}} \mathbf{N}_i.n_{k_{uq_i.A.N}}^{up.A.N[l]}$.

Proof. Since $n_{k_{uq_i.A.N}}^{up.A.N[l]}$ is the number of objects whose non-reference attribute section has $k_{uq_i.A.N}$ attributes, where attribute $up.A.N[l]$ appears, we have $\mathbf{N}_i.n_{k_{uq_i.A.N}}^{up.A.N[l]} = 1$ if $up.A.N[l]$ appears in $uq_i.A.N$, and $\mathbf{N}_i.n_{k_{uq_i.A.N}}^{up.A.N[l]} = 0$ otherwise. By definition, $|up.A.N \cap uq_i.A.N|$ is equal to the number of attributes that exist in both $up.A.N$ and $up.A.N$. Therefore, $|up.A.N \cap uq_i.A.N| = \sum_{l=1}^{k_{up.A.N}} \mathbf{N}_i.n_{k_{uq_i.A.N}}^{up.A.N[l]}$. ■

A.22. Proof for Lemma 20

Lemma 20. $\sum_{i=1}^{n} |p.A.N \cap q_i.A.N| = \sum_{j=1}^{k_{up.A.N}} \sum_{i=1}^{r} n_i^{up.A.N[j]}$

Proof.

$$\sum_{i=1}^{n} |p.A.N \cap q_i.A.N|$$

$$= \sum_{i=1}^{n} |up.A.N \cap uq_i.A.N|$$

$$= \sum_{i=1}^{n} \sum_{l=1}^{k_{up.A.N}} \mathbf{N}_i.n_{k_{uq_i.A.N}}^{up.A.N[l]} \qquad \text{(by Lemma 19)}$$

$$= \sum_{l=1}^{k_{up.A.N}} \sum_{i=1}^{n} \mathbf{N}_i.n_{k_{uq_i.A.N}}^{up.A.N[l]}$$

$$= \sum_{j=1}^{k_{up.A.N}} \sum_{i=1}^{r} \mathbf{N}.n_i^{up.A.N[j]}$$

$$= \sum_{j=1}^{k_{up.A.N}} \sum_{i=1}^{r} n_i^{up.A.N[j]} \qquad ■$$

A.23. Proof for Theorem 3

Theorem 3. *With the integer tuple* $\mathbf{N}$ *of* Q, $p.A.N$ *and its reduced non-reference attribute section* $up.A.N$, *the two types of non-reference attribute distance have the following properties.*

$$\sum_{q \in Q} TSizeDiff(p.A.N,\ q.A.N) = (2 * n_{0k_{p.A.N}}^{0} - n_{0r}^{0}) * k_{p.A.N} + n_{0r}^{1} - 2 * n_{0k_{p.A.N}}^{1} \tag{A.22a}$$

$$\sum_{q \in Q} TSetDiff(p.A.N,\ q.A.N) = n * k_{up.A.N} + n_{0r}^{1} - 2 * \sum_{j=1}^{k_{up.A.N}} \sum_{i=1}^{r} n_i^{up.A.N[j]} \tag{A.22b}$$

Proof. (1) **Proof of** Eq. (A.22a).

$$sum\_TSizeDiff(p.A.N,\ Q.A.N)$$

$$= \sum_{i=1}^{n} TSizeDiff(p.A.N,\ q_i.A.N)$$

$$= \sum_{i=1}^{n} |k_{p.A.N} - k_{q_i.A.N}|$$

$$= \sum_{i=1}^{n} |k_{p.A.N} - k_{q_{v_i}.A.N}|$$

(because $\langle v_1, v_2, \dots, v_n \rangle$ is a permutation of $\langle 1, 2, \dots, n \rangle$, which are defined in Definition (f))

$$= \sum_{i=1}^{\tau} (k_{p.A.N} - k_{q_{v_i}.A.N}) + \sum_{i=\tau+1}^{n} (k_{q_{v_i}.A.N} - k_{p.A.N})$$

(τ is defined as in Definition (f) of Section 3.2.2)

$$= \tau * k_{p.A.N} - \sum_{i=1}^{\tau} k_{q_{v_i}.A.N} + \sum_{i=\tau+1}^{n} k_{q_{v_i}.A.N} - (n - \tau) * k_{p.A.N}$$

$$= n_{0k_{p.A.N}}^{0} * k_{p.A.N} - n_{0k_{p.A.N}}^{1} + (n_{0r}^{1} - n_{0k_{p.A.N}}^{1}) - (n_{0r}^{0} - n_{0k_{p.A.N}}^{0}) * k_{p.A.N} \qquad \text{(by Lemma 18)}$$

$$= (2 * n_{0k_{p.A.N}}^{0} - n_{0r}^{0}) * k_{p.A.N} + n_{0r}^{1} - 2 * n_{0k_{p.A.N}}^{1}.$$

Thus, Eq. (A.22a) is proved.

Let us prove Eq. (A.22b) now.

$$\sum_{i=1}^{n} (TSetDiff(p.A.N,\ q_i.A.N))$$

$$= \sum_{i=1}^{n} (|p.A.N \cup q_i.A.N| - |p.A.N \cap q_i.A.N|) \qquad \text{(by Eq. (8))}$$

$$= \sum_{i=1}^{n} |p.A.N \cup q_i.A.N| - \sum_{i=1}^{n} |p.A.N \cap q_i.A.N|.$$

$$= n * |p.A.N| + \sum_{i=1}^{n} |q_i.A.N| - 2 * \sum_{i=1}^{n} |p.A.N \cap q_i.A.N|$$

$$= n * k_{up.A.N} + n_{0r}^{1} - 2 * \sum_{j=1}^{k_{up.A.N}} \sum_{i=1}^{r} n_i^{up.A.N[j]} \qquad \text{(by Lemma 20)}$$

Eq. (A.22b) is therefore proved. ■

A.24. Proof for Eq. (26)

$$(\sum_{j=0}^{\min(k_x,k_{y_i})-1} (1 - \overline{d}^{(x,y_i,j)})) + |k_x - k_{y_i}|$$

$$= \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} z^{(x,y_i,j)}$$

Proof. Subcase (i). Suppose $k_x \le k_{y_i}$.

$$\sum_{j=0}^{\min(k_x,k_{y_i})-1} (1 - \overline{d}^{(x,y_i,j)}) + |k_x - k_{y_i}|$$

$$= \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + k_{y_i} - k_x$$

$$= \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} 1$$

$$= \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} z^{(x,y_i,j)}$$

Subcase (ii). Suppose $k_x > k_{y_i}$.

$$\sum_{j=0}^{\min(k_x,k_{y_i})-1} (1 - \overline{d}^{(x,y_i,j)}) + |k_x - k_{y_i}|$$

$$= \sum_{j=0}^{k_{y_i}-1} (1 - \overline{d}^{(x,y_i,j)}) + k_x - k_{y_i}$$

$$= \sum_{j=0}^{k_{y_i}-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_{y_i}}^{k_x-1} (1 - 0)$$

$$= \sum_{j=0}^{k_{y_i}-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_{y_i}}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)})$$

$$= \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + 0$$

$$= \sum_{j=0}^{k_x-1} (1 - \overline{d}^{(x,y_i,j)}) + \sum_{j=k_x}^{k_{y_i}-1} z^{(x,y_i,j)}$$

(because, as $k_{y_i} < k_x$, we must have $k_{y_i} - 1 < k_x$,giving $\sum_{j=k_x}^{k_{y_i}-1} z^{(x,y_i,j)} = 0$)

Thus, Eq. (26) is valid in both cases. ■

References

Ackah-Arthur, H., Chen, J., Towey, D., Omari, M., Xi, J., Huang, R., 2019. One-domain-one-input: Adaptive random testing by orthogonal recursive bisection with restriction. IEEE Trans. Reliab. 68, 1404–1428.

Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P., Bertolino, A., et al., 2013. An orchestrated survey of methodologies for automated software test case generation. J. Syst. Softw. 86, 1978–2001.

Arcuri, A., Briand, L., 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab. 24, 219–250.

Arcuri, A., Yao, X., 2008. Search based software testing of object-oriented containers. Inform. Sci. 178, 3075–3095.

Barus, A.C., Chen, T.Y., Kuo, F.C., Liu, H., Merkel, R., Rothermel, G., 2016. A cost-effective random testing method for programs with non-numeric inputs. IEEE Trans. Comput. 65, 3509–3523.

Bertolino, A., 2007. Software testing research: Achievements, challenges, dreams. In: Future of Software Engineering. FOSE'07, pp. 85–103.

Binder, R.V., 1996. Testing object-oriented software: A survey. Softw. Test. Verif. Reliab. 6, 125–252.

Binder, R., 2000. Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley Professional.

Ceccato, M., Marchetto, A., Mariani, L., Nguyen, C.D., Tonella, P., 2012. An empirical study about the effectiveness of debugging when random test cases are used. In: 2012 34th International Conference on Software Engineering. ICSE, pp. 452–462.

Chan, K.P., Chen, T., Towey, D., 2006a. Forgetting test cases. In: 30th Annual International Computer Software and Applications Conference. COMPSAC'06, pp. 485–494.

Chan, K.P., Chen, T.Y., Towey, D., 2006b. Restricted random testing: Adaptive random testing by exclusion. Int. J. Softw. Eng. Knowl. Eng. 16, 553–584.

Chen, J., Ackah-Arthur, H., Mao, C., Kudjo, P.K., 2019a. A taxonomic review of adaptive random testing: Current status, classifications, and issues. arXiv:1909.10879.

Chen, J., Bao, Q., Tse, T., Chen, T.Y., Xi, J., Mao, C., Yu, M., Huang, R., 2020. Exploiting the largest available zone: A proactive approach to adaptive random testing by exclusion. IEEE Access 8, 52475–52488.

Chen, J., Chen, H., Guo, Y., Zhou, M., Huang, R., Mao, C., 2021. A novel test case generation approach for adaptive random testing of object-oriented software using k-means clustering technique. IEEE Trans. Emerg. Top. Comput. Intell..

Chen, J., Kuo, F.C., Chen, T.Y., Towey, D., Su, C., Huang, R., 2016. A similarity metric for the inputs of oo programs and its application in adaptive random testing. IEEE Trans. Reliab. 66, 373–402.

Chen, T.Y., Kuo, F.C., Liu, H., 2009. Adaptive random testing based on distribution metrics. J. Syst. Softw. 82, 1419–1433.

Chen, T.Y., Kuo, F.C., Liu, H., Wong, W.E., 2013. Code coverage of adaptive random testing. IEEE Trans. Reliab. 62, 226–237.

Chen, T.Y., Kuo, F.C., Merkel, R., 2006. On the statistical properties of testing effectiveness measures. J. Syst. Softw. 79, 591–601.

Chen, T.Y., Kuo, F.C., Merkel, R.G., Ng, S.P., 2004a. Mirror adaptive random testing. Inf. Softw. Technol. 46, 1001–1010.

Chen, T.Y., Kuo, F.C., Merkel, R.G., Tse, T., 2010. Adaptive random testing: The art of test case diversity. J. Syst. Softw. 83, 60–66.

Chen, T.Y., Leung, H., Mak, I., 2004b. Adaptive random testing. In: Annual Asian Computing Science Conference. pp. 320–329.

Chen, T.Y., Merkel, R., Wong, P., Eddy, G., 2004c. Adaptive random testing through dynamic partitioning. In: Fourth International Conference onQuality Software, 2004. QSIC 2004. Proceedings. pp. 79–86.

Chen, H.Y., Tse, T., 2013. Equality to equals and unequals: A revisit of the equivalence and nonequivalence criteria in class-level testing of object-oriented software. IEEE Trans. Softw. Eng. 39, 1549–1563.

Chen, J., Zhou, M., Tse, T., Chen, T.Y., Guo, Y., Huang, R., Mao, C., 2019b. Toward a k-means clustering approach to adaptive random testing for object oriented software. Sci. China Inf. Sci. 62, 1–2.

Chen, J., Zhu, L., Chen, T.Y., Towey, D., Kuo, F.C., Huang, R., Guo, Y., 2018. Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering. J. Syst. Softw. 135, 107–125.

Ciupa, I., Leitner, A., Oriol, M., Meyer, B., 2006. Object distance and its application to adaptive random testing of object-oriented programs. In: Proceedings of the 1st international workshop on Random testing. pp. 55–63.

Ciupa, I., Leitner, A., Oriol, M., Meyer, B., 2007. Experimental assessment of random testing for object-oriented software. In: Proceedings of the 2007 international symposium on Software testing and analysis. pp. 84–94.

Ciupa, I., Leitner, A., Oriol, M., Meyer, B., 2008. Artoo: Adaptive random testing for object-oriented software. In: Proceedings of the 30th international conference on Software engineering. pp. 71–80.

CodeProject, 2023. Codeproject: For those who code. Retrieved from https://www.codeproject.com/.

d'Amorim, M., Pacheco, C., Xie, T., Marinov, D., Ernst, M.D., 2006. An empirical comparison of automated generation and classification techniques for object-oriented unit testing. In: 21st IEEE/ACM International Conference on Automated Software Engineering. ASE'06, pp. 59–68.

Engineering, A.I.S., 2023. Poco c++ libraries. Retrieved from https://pocoproject.org/.

Godefroid, P., Klarlund, N., Sen, K., 2005. Dart: directed automated random testing. vol. 40. pp. 213–223.

Huang, R., Sun, W., Chen, H., Cui, C., Yang, N., 2022. A nearest-neighbor divide-and-conquer approach for adaptive random testing. Sci. Comput. Programm. 215, 102743.

Huang, R., Sun, W., Xu, Y., Chen, H., Towey, D., Xia, X., 2021. A survey on adaptive random testing. IEEE Trans. Softw. Eng. 47, 2052–2083.

Hui, Wang, X., Huang, S., Yang, S., 2021. Mt-art: A test case generation method based on adaptive random testing and metamorphic relation. IEEE Trans. Reliab. 70, 1397–1421.

Inkumsah, K., Xie, T., 2008. Improving structural testing of object-oriented programs via integrating evolutionary testing and symbolic execution. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. pp. 297–306.

Jia, Y., Harman, M., 2011. An analysis and survey of the development of mutation testing. IEEE Trans. Softw. Eng. 37, 649–678.

Le Traon, Y., Jéron, T., Jézéquel, J.M., Morel, P., 2000. Efficient object-oriented integration and regression testing. IEEE Trans. Reliab. 49, 12–25.

Levenshtein, V.I., 1966. Binary codes capable of correcting deletions, insertions and reversals. Sov. Phys. Doklady 10, 707–710.

Lin, Y., Tang, X., Chen, Y., Zhao, J., 2009. A divergence-oriented approach to adaptive random testing of Java programs. In: 2009 IEEE/ACM International Conference on Automated Software Engineering. pp. 221–232.

Liu, H., Xie, X., Yang, J., Lu, Y., Chen, T.Y., 2011. Adaptive random testing through test profiles. Softw. - Pract. Exp. 41, 1131–1154.

Mao, C., Chen, T.Y., Kuo, F.C., 2017. Out of sight, out of mind: A distance-aware forgetting strategy for adaptive random testing. Sci. Chin. Inform. Sci. 60, 092106.

Mayer, J., 2005. Lattice-based adaptive random testing. In: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. pp. 333–336.

Media, S., 2023a. Sourceforge. Retrieved from https://sourceforge.net/.

Media, S., 2023b. Sourceforge-download, develop and publish free open source software. Retrieved from http://sourceforge.net.

Microsoft, 2023. Codeplex archive: open source project archive. Retrieved from https://archive.codeplex.com/.

Omari, M., Chen, J., Kudjo, P.K., Ackah-Arthur, H., Huang, R., 2019. Random border mirror transform: A diversity based approach to an effective and efficient mirror adaptive random testing. In: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security. QRS, pp. 54–61.

Ostrand, T.J., Balcer, M.J., 1988. The category-partition method for specifying and generating fuctional tests. Commun. ACM 31, 676–686.

Pacheco, C., Ernst, M.D., 2007. Randoop: feedback-directed random testing for java. In: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion. pp. 815–816.

Pacheco, C., Lahiri, S.K., Ball, T., 2008. Finding errors in net with feedback-directed random testing. In: Proceedings of the 2008 international symposium on Software testing and analysis. pp. 87–96.

Pacheco, C., Lahiri, S.K., Ernst, M.D., Ball, T., 2007. Feedback-directed random test generation. In: 29th International Conference on Software Engineering. ICSE'07, pp. 75–84.

Shahbazi, A., Tappenden, A.F., Miller, J., 2012. Centroidal Voronoi tessellations-a new approach to random testing. IEEE Trans. Softw. Eng. 39, 163–183.

Team, C.D., 2023. Codeforge. Retrieved from http://www.codeforge.com/.

Tillmann, N., Halleux, J.D., 2008. Pex–white box test generation for net. In: International conference on tests and proofs. Springer, pp. 134–153.

Vargha, A., Delaney, H.D., 2000. A critique and improvement of the CL common language effect size statistics of McGraw and wong. J. Educ. Behav. Stat. 25, 101–132.

Zheng, W., Zhang, Q., Lyu, M., Xie, T., 2010. Random unit-test generation with mut-aware sequence recommendation. In: Proceedings of the IEEE/ACM international conference on Automated software engineering. pp. 293–296.

Jinfu Chen received the Ph.D. degree in computer science and technology from Huazhong University of Science and Technology, Wuhan, China, in 2009. He is currently a full professor in the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China. His major research interests include software testing, software security, and trusted software. He has published more than 80 papers in some famous journals or conferences, including in IEEE Transactions on Reliability, Information Sciences, Journal of Systems and Software, Information and Software Technology, Software: Practice and Experience, IET Software, The Computer Journal, ISSTA, ASE, ISSRE, QRS, etc. He is a member of the IEEE and the ACM, and a member of the China Computer Federation.

Jingyi Chen received the B.E. degree in 2020 from Huaiyin Normal University, Huaian, China, and started her master's degree in 2021 at Jiangsu University, Zhenjiang, China, both in Computer Science and Technology. She is currently working toward the Ph.D. in the Computer Science and Technology, Jiangsu University, Zhenjiang, China. Her current research interests include fuzzing and adaptive random testing.

Lili Zhu received the B.E. degree in 2014 from Jiangsu University, Zhenjiang, China, and Master's degree in 2017 from Jiangsu University, Zhenjiang, China, both in Computer Science and Technology. Her research interests include vulnerability detection.

Chengying Mao received the B.E. degree in computer science and technology from Central South University, Changsha, China, in 2001 and the Ph.D. degree in computer software and theory from the Huazhong University of Science and Technology, Wuhan, China, in 2006. From 2006 to 2008, he was a Postdoctoral Researcher with the College of Management, Huazhong University of Science and Technology. He is currently a Full Professor with the School of Software and IoT Engineering, Jiangxi University of Finance and Economics, Nanchang, China. His current research interests include software engineering and service computing. He is a Member of ACM, IEEE Computer Society, and CCF.

Qihao Bao received the B.E. degree in 2017 from Jiangsu University, Zhenjiang, China, and Master's degree in 2020 from Jiangsu University, Zhenjiang, China, both in Computer Science and Technology. His research interests include vulnerability detection.

Rubing Huang received the Ph.D. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2013. From 2016 to 2018, he was a Visiting Scholar with the Swinburne University of Technology, Melbourne, VIC, Australia, and Monash University, Melbourne, VIC, Australia. He is currently an Associate Professor with the School of Computer Science and Engineering, Macau University of Science and Technology (MUST), China. Before joining MUST, he was an Associate Professor with Jiangsu University, Zhenjiang, China. He has more than 60 publications in journals and proceedings, including the IEEE Transactions on Software Engineering, IEEE Transactions on Reliability, IEEE Transactions on Emerging Topics in Computational Intelligence, Journal of Systems and Software, Information and Software Technology, Science of Computer Programming, IET Software, The Computer Journal, International Journal of Software Engineering and Knowledge Engineering, Information Sciences, ICSE, ISSRE, ICST, COMPSAC, QRS, SEKE, and SAC. His current research interests include AI for software engineering, software engineering for AI, software testing, debugging, and maintenance. He is a Senior Member of China Computer Federation and a Member of ACM.