Computing and Informatics, Vol. 40, 2021, 469–488, doi: 10.31577/cai_2021_3_469

CONCEPT SIMILARITY IN FORMAL CONCEPT ANALYSIS WITH MANY-VALUED CONTEXTS

Anna Formica

Istituto di Analisi dei Sistemi ed Informatica (IASI)
National Research Council
Via dei Taurini 19, I-00185, Rome, Italy
e-mail: anna.formica@iasi.cnr.it

Abstract. Formal Concept Analysis (FCA) is a mathematical framework which can also support critical activities for the development of the Semantic Web. One of them is represented by Similarity Reasoning, i.e., the identification of different concepts that are semantically close, that allows users to retrieve information on the Web more efficiently. In order to model uncertainty information, in this paper FCA with many-valued contexts is addressed, where attribute values are intervals, which is referred to as FCA with Interordinal scaling (IFCA). In particular, a method for evaluating concept similarity in IFCA is proposed, which is a problem that has not been adequately investigated, although the increasing interest in the literature in this topic.

Keywords: Formal concept analysis, similarity reasoning, many-valued contexts, FCA with interordinal scaling

1 INTRODUCTION

Formal Concept Analysis (FCA) is a formal framework based on lattice theory which is commonly used for data analysis [15, 28]. In the basic setting, FCA attributes are crisp, i.e., any object either has or does not have an attribute of a given context. This is the case of the so-called *one-valued* contexts. However, in real life most of attributes are fuzzy rather than crisp, i.e., "it is a matter of degree to which an object has a (fuzzy) attribute" [2]. In other words, an object may have different attributes with different values, and an attribute may apply to different objects with different values. This is the case of *many-valued* contexts [15]. *Fuzzy Formal Concept*

Analysis (FFCA) is a generalization of FCA which provides a formal framework for structuring, analyzing and visualizing data in the presence of uncertainty information [32]. In particular, in FFCA contexts are many-valued, and the attribute values are real numbers in the range $[0, 1]$. In this paper, this kind of FCA is referred to as OFCA, in line with the notion of FCA with *Ordinal scaling* defined in [15].

Regarding the notion of fuzzy sets [7], *Type-1 Fuzzy Sets* (T1 FSs) and *Type-2 Fuzzy Sets* (T2 FSs) were both introduced by Zadeh, the former in 1965 in the seminal paper [36], and the latter in 1975 in [37]. T2 FSs provide a way to overcome one of the early objections made about T1 FSs, i.e., that "it sounds contradictory for something that is *fuzzy* to have a perfectly defined membership function" [26]. In this paper we focus on *Interval Type-2 FSs* (IT2 FSs) [22], which represent a simplification about T2 FSs that is receiving much attention in the literature in different research areas, with different purposes, as for instance in [1, 8] just to mention a couple of examples.

Similarity Reasoning, i.e., the identification of syntactically different concepts that are semantically close, is fundamental in several research areas such as Cognitive Science, Artificial Intelligence, Software Engineering, and in the Semantic Web [4, 17]. Concept similarity in the framework of FCA with *Interordinal scaling* (IFCA), i.e., in many-valued contexts where attribute values are intervals, is a problem that has been marginally investigated in the literature, despite of an increasing interest in this topic.

In this paper, a concept similarity measure for IFCA is proposed which is novel because it combines the IT2 FS framework, with regard to concept extents, and the *information content* approach [23], with regard to concept intents. The latter has been extensively investigated and experimented in the literature, and has a higher correlation with human judgment with respect to the traditional approaches. In particular, this paper is a short and revised version of the results presented in [14]. Furthermore, in this work both the basic notions and the overall approach are presented informally, by providing simple examples, in order to reach a broad audience of readers, and not only specialists working in the area.

The paper is organized as follows. In the next section, the Related Work is given, and in Section 3 the basic notions related to FCA and IFCA are recalled. In Section 4 the notion of IFCA concept similarity is presented. In particular, first the similarity between IT2 FSs is recalled and, successively, the information content similarity is addressed. Then, they are combined in order to define the similarity between IFCA concepts. Finally, in Section 5 an evaluation of the method is given, and Section 6 concludes.

2 RELATED WORK

FCA techniques and tools have been employed in different research fields, such as, Information Retrieval, e-Learning, Expert Systems, etc., and in the development of the Semantic Web [32]. Similarity reasoning is fundamental in several research areas

such as Cognitive Science, Artificial Intelligence, Software Engineering and, recently, also in the Semantic Web [13]. In the development of the Semantic Web, similarity reasoning supports all the activities that, in general, require human interaction (which are time-consuming and error-prone), such as web service discovery, query refinement techniques for search engines, extractions of patterns and trends in web users behaviors, etc. Therefore, the similarity method proposed in this paper can be employed in all the research fields that can benefit from the combination of FCA techniques and similarity reasoning.

Some research challenges about the contribution of FCA in the Semantic Web development are illustrated in [18], and concern the automatic or semi-automatic generation of ontologies (ontology *engineering*), and the critical problem of identifying the overlapping knowledge in a common domain (also referred to as ontology *mapping*, *merging*, *integration*, or *alignment*). In particular, due to the presence in the web of large and specialized ontologies, FCA has been employed for more than one decade for reusing and combining independently developed domain ontologies, see for instance [31].

FCA concept similarity has been addressed in [10], by relying on human domain expertise, and in [11, 33], according to the information content approach, but in both cases within one-valued contexts. Many-valued contexts have been addressed in [13], but in the case of FCA with *Ordinal scaling* (OFCA). Therefore, in both the mentioned papers, IT2 FSs have not been addressed and a similarity measure has been proposed, based on T1 FSs, which has been experimented and compared with the relevant similarity measures proposed in the literature. It has been used as a basis for the definition of the similarity measure proposed in this paper for FCA with *Interordinal scaling* (IFCA), as illustrated in the next sections. Note that in [3] and [12] different problems related to OFCA have been addressed, by relying on Rough Set Theory.

With regard to IFCA, a formal framework, referred to as *L-Fuzzy concept theory*, has been defined in [5] which is probably the first research paper providing a theoretical foundation about it. Successively, some interesting works have been defined in the literature which have investigated and deepened the mathematics underlying specific aspects of IFCA, as for instance [6].

In [30] the need for IT2 fuzzy analytical systems for the development of the Semantic Web is emphasized, and a similarity measure for IFCA is proposed. It is based on the similarity measure for IT2 FSs defined in [35], the approach presented in [11], and relies on the experimental results given in [13]. In Section 5, a discussion about the evaluation of the proposed method is given.

As mentioned, the proposed *T2ConSim* combines the similarity of the concept extents and the concept intents. Concept extents are evaluated according to *IT2FSim*, which is the widely accepted crisp similarity measure for IT2 FSs defined in [35]. It is used in most applications of general T2 FSs due to the simpler underlying mathematics. Such a notion has been adopted here because it allows a relevant simplification about the definition of similarity between general T2 FSs, in line with the scope of this paper which is intended for non-specialist readers.

3 FCA WITH MANY-VALUED CONTEXTS

In this section the basic notions related to *Formal Concept Analysis* (FCA) are briefly recalled. In order to illustrate them, the context named *Sardinia Hotels* presented in [14] is used, which also allows us to introduce, in the next section, the notions underlying IFCA in an intuitive way.

In FCA [15], a *one-valued context* (*context* for short) is a triple (O, A, R), where O is a set of *objects*, A is a set of *attributes*, and R is a binary relation between O and A. In the *Sardinia Hotels* context mentioned below, the set O is defined by the following six objects representing six different hotels:

$$O = \{H1, H2, H3, H4, H5, H6\},$$

and the set A is defined by the three following attributes:

$$A = \{SwPool, Sea, Meal\}$$

where *SwPool* stands for swimming pool. Furthermore, the relation R among hotels and attributes is defined by Table 1.

A concept of the *Sardinia Hotels* context is, for instance, the pair (E, I) where E is a set of objects, referred to as concept *extent*, and I is a set of attributes, referred to as concept *intent*, defined as follows:

$$((H1, H3, H5), (Sea, Meal))$$

since the objects $H1$, $H3$, and $H5$ have both the attributes Sea and $Meal$, and vice versa, both these attributes apply to the objects $H1$, $H3$, and $H5$.

Intuitively, we can say that concepts correspond to maximal rectangles of crosses in the context, after appropriate permutations of rows and columns. It is possible to establish an *inheritance relation* ($\leq$) between concepts of a context, say (E_1, I_1), (E_2, I_2), as follows:

$$(E_1, I_1) \leq (E_2, I_2) \text{ iff } E_1 \subseteq E_2 (\textit{iff } I_2 \subseteq I_1).$$

In particular, (E_1, I_1) is called *subconcept* of (E_2, I_2) and (E_2, I_2) is called *superconcept* of (E_1, I_1). For instance, the concept $((H1, H2, H3, H5), (Meal))$ is a superconcept of the previous one, i.e.:

$$((H1, H3, H5), (Sea, Meal)) \leq ((H1, H2, H3, H5), (Meal))$$

and, vice versa, the former concept is a subconcept of the latter.

Given a context (O, A, R), consider the set of all the concepts of this context, indicated as $\mathcal{L}(O, A, R)$. Then:

$$(\mathcal{L}(O, A, R), \leq)$$

	SwPool	**Sea**	**Meal**
H1		×	×
H2	×		×
H3		×	×
H4	×	×	
H5		×	×
H6	×	×	

Table 1. The FCA *Sardinia Hotels* context

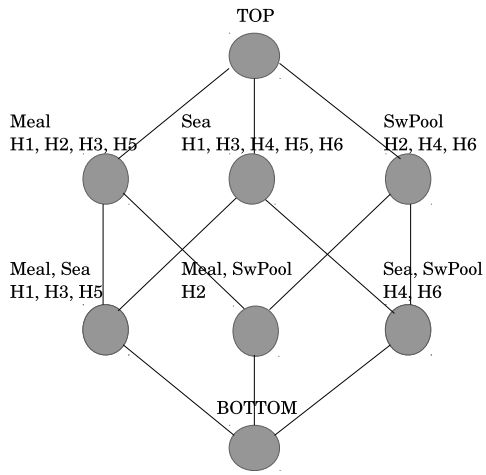


Figure 1. Concept Lattice of the *Sardinia Hotels* context [14]

is a complete lattice called *Formal Concept Lattice* (*Concept Lattice* for short), i.e., for each subset of concepts, the greatest lower bound (the greatest common subconcept) and the least upper bound (the least common superconcept) exist. For instance, the Concept Lattice constructed from the context of Table 1 is shown in Figure 1. Note that the Concept Lattice has two special nodes, the maximum and minimum nodes, grouping all the objects and the attributes of the context, respectively. The number of objects in the concept extent (the cardinality) is also referred to as the *support* the concept [19], therefore the concept corresponding to the maximum node has maximum support.

3.1 From One- to Many-Valued Contexts

In a one-valued context an attribute is a property that an object may have or may not have. For instance, according to the one-valued context *Sardinia Hotels* above, each of the attributes *SwPool*, *Sea*, and *Meal* applies or does not apply to each of the hotel objects. However, in real world, an attribute may apply to different objects with different values, i.e., it can be many-valued.

In FCA, a *many-valued context* is a quadruple (O, A, V, R), where O is a set of *objects*, A is a set of *many-valued attributes*, V is a set of *attribute values*, and R is a ternary relation among O, A, and V such that:

$$(o, a, v) \in R \text{ and } (o, a, w) \in R \Rightarrow v = w$$

where $(o, a, v) \in R$ can be read as "the attribute a has the value v for the object o". Note that (O, A, V, R) is referred to as *one-valued context* if V has one element [15].

Analogously to one-valued contexts, many-valued contexts can be represented by tables, where rows are labeled by objects and columns are labeled by attributes. Many-valued contexts can be transformed into one-valued contexts according to a *conceptual scaling* process [15]. In particular, in this process, each attribute of a many-valued context is interpreted by means of a context, referred to as *conceptual scale* (for details about the transformation process of a many-valued context into a one-valued context see [15]). Typical conceptual scales are *Nominal*, *Ordinal*, and *Interordinal* scales. Nominal scales are used for attribute values which mutually exclude each other, for instance in the case of the attribute values {*human, animal, plant*}. Ordinal scales are suitable when attribute values are ordered, and each value implies the weaker ones, e.g., {*extremely active, very active, active*}. *Interordinal scales* are used for attributes which have a range of possible values (intervals), e.g., {*fully, very much, very few, not at all*}. In the next subsection we focus on FCA with Interordinal scaling.

3.1.1 FCA with Interordinal Scaling

As mentioned above, in many-valued contexts attributes do not describe objects in a uniform way, i.e., a given attribute applies to different objects in different ways. For instance, in the *Sardinia Hotels* context above, consider the attribute $Meal$. In general, when reserving an hotel, we would like to know whether the hotel provides both lunch and dinner, or half-board. Without the introduction of fuzzy information, we have no way to specify how appropriate is an attribute to a given object.

In order to deal with fuzzy contexts, we need to recall the following definitions.

A *Type-1 Fuzzy Set* (T1 FS) A (also called *fuzzy set*) in a space of points X is characterized by a *membership function* $\mu_A(x)$ which associates each point x in X with a real number in the interval $[0, 1]$ representing the *grade of membership* of x in A [36]. Note that for an ordinary set, the membership function can take only the values 1 and 0, depending on x does or does not belong to A, respectively.

For instance, the following set A:

$$A = ((H1, 1.0), (H2, 0.5), (H3, 0.5), (H5, 1.0))$$

is a T1 FS in the space of point $X = \{H1, H2, H3, H5\}$.

An *Interval Type-2 Fuzzy Sets* (IT2 FS) $\tilde{A}$ in a space of points X is characterized by two membership functions, an *upper membership function* $\bar{\mu}_{\tilde{A}}$ and a *lower membership function* $\underline{\mu}_{\tilde{A}}$ which are both T1 FSs, such that each point x in X is associated with an interval $[\underline{\mu}_{\tilde{A}}(x), \bar{\mu}_{\tilde{A}}(x)]$ representing the *grade of membership* of x in $\tilde{A}$ [34].

For instance, the following set $\tilde{A}$:

$$\tilde{A} = ((H2, [0.6, 0.7]), (H4, [0.6, 0.8]), (H5, [0.4, 0.9]))$$

is a IT2 FS in the space of point $X = \{H2, H4, H5\}$.

In this subsection we address FCA contexts where grades of memberships are intervals, and in particular *words*. Indeed, words are closer to human judgment when we need to quantify "how much" an object is described by an attribute or, vice versa, an attribute applies to an object [5, 30]. Possible words representing grades of membership are:

$$\{Fully, Very\ Much, Very, Few, Very\ Few, Not\ at\ all\}.$$

For instance, consider the many-valued context *Sardinia Hotels* which is specified by the fuzzy relation given in Table 2 where crosses in Table 1 have been replaced by *words*, each allowing us to specify "how much" an object has, or is described by, an attribute, and vice versa an attribute applies to an object.

Consider for instance the hotel $H2$ in Table 2. It has the attribute *SwPool* with grade of membership *Fully*, which means that such an attribute fully applies to the hotel $H2$ (and vice versa the hotel $H2$ can be properly described by the attribute *SwPool*). Instead, the object $H2$ has the attribute *Meal* with a membership value *Very*, which means that such an attribute partially applies to this hotel (for instance it could provide meals just for lunch). In order to address only objects related to attributes with relevant grades of membership, a threshold is fixed such that the pairs with membership values under the threshold are ignored. For instance, assume that in the *Sardinia Hotels* context the intervals *Very Few* and *Not at all* are below the threshold. With this assumption, the pair $(H5, Sea)$ is ignored.

For instance, consider the IFCA context for the *Sardinia Hotels* shown in Table 2.

	SwPool	**Sea**	**Meal**
H1		Fully	Fully
H2	Fully		Very
H3		Very much	Very
H4	Fully	Fully	
H5		Very Few	Fully
H6	Fully	Very much	

Table 2. The IFCA *Sardinia Hotels* context, by using words

In order to elaborate such grades of membership, words are replaced by intervals (IT2 FS grades of membership). The association of words with intervals is a problem which has been extensively investigated in the literature and is still attracting a lot of attention [25], [27]. A simple association of words with intervals is shown in Table 3. Therefore, the context of Table 2 becomes the IFCA context shown in Table 4.

Not at all	$[0.0, 0.1]$
Very few	$[0.1, 0.3]$
Few	$[0.3, 0.5]$
Very	$[0.5, 0.7]$
Very much	$[0.7, 0.9]$
Fully	$[0.9, 1.0]$

Table 3. Mapping words to intervals

	SwPool	**Sea**	**Meal**
H1		$[0.9, 1.0]$	$[0.9, 1.0]$
H2	$[0.9, 1.0]$		$[0.5, 0.7]$
H3		$[0.7, 0.9]$	$[0.5, 0.7]$
H4	$[0.9, 1.0]$	$[0.9, 1.0]$	
H5		$[0.1, 0.3]$	$[0.9, 1.0]$
H6	$[0.9, 1.0]$	$[0.7, 0.9]$	

Table 4. The IFCA *Sardinia Hotels* context

In IFCA, Concept Lattices are defined similarly to FCA Concept Lattices. For instance, the IFCA Concept Lattice constructed from the context of Table 4 is shown in Figure 2. In IFCA an object of a concept is associated with an interval, standing for the related grade of membership. In the case two or more attributes apply to an object with different grades of membership (i.e., different intervals) the object is associated with the interval having, as lower bound and upper bound, the minimum between the lower bounds and the upper bounds, respectively. For instance, consider again the concept involving the attributes Sea and $Meal$, which in this case is defined as follows:

$$(((H1, [0.9, 1.0]), (H3, [0.5, 0.7])), (Sea, Meal))$$

since, as mentioned above, the pair ($(H5$,[0.1,0.3]), Sea) is not considered because under the threshold. According to the context shown in Table 4, Sea and $Meal$ apply to $H3$ with different intervals, that are $[0.7, 0.9]$ and $[0.5, 0.7]$, respectively. Since 0.5 is the minimum between their lower bounds, and 0.7 is the minimum between their upper bounds, in the concept above the object $H3$ has been associated with the interval $[0.5, 0.7]$. Indeed this interval represents the highest common grade of

membership that allows $H3$ to be described by both the attributes Sea and $Meal$ (and, vice versa, both the attributes Sea and $Meal$ to be applied to $H3$).

In the following section, the similarity between concepts in IFCA is addressed.

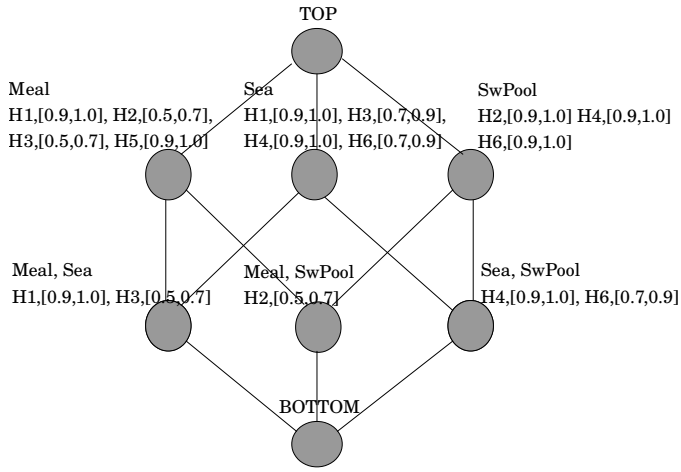


Figure 2. Concept Lattice of the IFCA *Sardinia Hotels* context [14]

4 IFCA CONCEPT SIMILARITY

In this section IFCA concept similarity is computed by combining the similarity of concept extents, i.e., the IT2 FSs of objects, and the similarity of concept intents, i.e., the sets of attributes.

4.1 Concept Extent Similarity

With regard to the similarity of concept extents, we need to recall a few basic notions about IT2 FSs. Note that in the literature, the notions of similarity between T2 FSs have been proposed by several authors, and are based on different underlying definitions (as for instance the notion of cardinality) [16, 34]. Below we focus on the definitions that are the most frequently used in the literature, which require a simpler mathematics with respect to the others.

4.1.1 Similarity Between IT2 FSs

In this subsection, the notions of cardinality and *average cardinality* of an IT2 FS are recalled [16]. To this end, we first need to remind that the *cardinality* of a T1 FS A in a space of points X, also referred to as *power* of the T1 FS A, and denoted

as $p(A)$, is given by the sum of all membership grades, i.e.:

$$p(A) = p(\mu_A(x)) = \sum_{i=1}^{N} \mu_A(x_i). \tag{1}$$

For instance, the cardinality of the set A:

$$A = ((H1, 1.0), (H2, 0.5), (H3, 0.5), (H5, 1.0))$$

is:

$$p(A) = 1.0 + 0.5 + 0.5 + 1.0 = 3.0.$$

Given an IT2 FS $\tilde{A}$, the *cardinality* of $\tilde{A}$, denoted as $P(\tilde{A})$, is an interval defined as follows:

$$P(\tilde{A}) = [p(\underline{\mu}_{\tilde{A}}(x)), p(\bar{\mu}_{\tilde{A}}(x))] \tag{2}$$

where $p(\underline{\mu}_{\tilde{A}})$, and $p(\bar{\mu}_{\tilde{A}})$ are the cardinalities of the lower and upper membership functions, respectively, which are T1 FSs. The *average cardinality* of an IT2 FS $\tilde{A}$, indicated as $AC(\tilde{A})$, is defined as the average of its minimum and maximum cardinalities, i.e.:

$$AC(\tilde{A}) = \frac{p(\underline{\mu}_{\tilde{A}}(x)) + p(\bar{\mu}_{\tilde{A}}(x))}{2}. \tag{3}$$

For instance, consider the IT2 FS $\tilde{A}$:

$$\tilde{A} = ((H2, [0.6, 0.7]), (H4, [0.6, 0.8]), (H5, [0.4, 0.9])).$$

The cardinality of $\tilde{A}$, $P(\tilde{A})$, is the interval having as lower and upper bounds the sums of the grades of the lower and upper membership functions, respectively, therefore:

$$P(\tilde{A}) = [1.6, 2.4]$$

because:

$$p(\underline{\mu}_{\tilde{A}}) = 0.6 + 0.6 + 0.4 = 1.6$$

and:

$$p(\bar{\mu}_{\tilde{A}}) = 0.7 + 0.8 + 0.9 = 2.4.$$

Then, the average cardinality $AC(\tilde{A})$ is the following:

$$AC(\tilde{A}) = (1.6 + 2.4)/2 = 2.$$

Let us now address the intersection and union of IT2 FSs. The intersection, $\tilde{A} \cap \tilde{B}$, and union, $\tilde{A} \cup \tilde{B}$, of the IT2 FSs $\tilde{A}$ and $\tilde{B}$ are both IT2 FSs. In particular, the membership grades of an element x are intervals defined, respectively, according to the lower and upper membership functions as follows:

$$\tilde{A} \cap \tilde{B}(x) = [\min(\underline{\mu}_{\tilde{A}}(x_i), (\underline{\mu}_{\tilde{B}}(x_i)), \min(\bar{\mu}_{\tilde{A}}(x_i), (\bar{\mu}_{\tilde{B}}(x_i))], \tag{4}$$

$$\tilde{A} \cup \tilde{B}(x) = [\max(\underline{\mu}_{\tilde{A}}(x_i), (\underline{\mu}_{\tilde{B}}(x_i)), \max(\bar{\mu}_{\tilde{A}}(x_i), (\bar{\mu}_{\tilde{B}}(x_i))]. \tag{5}$$

On the basis of these notions, we are now able to recall the similarity between IT2 FSs. In particular, we follow the crisp similarity measure proposed by [35], here referred to as *IT2FSim*, which is defined below:

$$IT2FSim(\tilde{A}, \tilde{B}) = \frac{AC(\tilde{A} \cap \tilde{B})}{AC(\tilde{A} \cup \tilde{B})} \tag{6}$$

and, therefore:

$$IT2FSim(\tilde{A}, \tilde{B}) = \frac{\sum_{i=1}^{N} \min(\bar{\mu}_{\tilde{A}}(x_i), (\bar{\mu}_{\tilde{B}}(x_i)) + \sum_{i=1}^{N} \min(\underline{\mu}_{\tilde{A}}(x_i), (\underline{\mu}_{\tilde{B}}(x_i))}{\sum_{i=1}^{N} \max(\bar{\mu}_{\tilde{A}}(x_i), (\bar{\mu}_{\tilde{B}}(x_i)) + \sum_{i=1}^{N} \max(\underline{\mu}_{\tilde{A}}(x_i), (\underline{\mu}_{\tilde{B}}(x_i))}. \tag{7}$$

For instance, consider the previous set $\tilde{A}$, and the set $\tilde{B}$ below:

$$\tilde{B} = ((H1, [0.4, 0.9]), (H2, [0.7, 0.8]), (H5, [0.3, 1.0])).$$

Then:

$$\tilde{A} \cap \tilde{B} = ((H2, [0.6, 0.7]), (H5, [0.3, 0.9])),$$
$$AC(\tilde{A} \cap \tilde{B}) = ((0.6 + 0.3) + (0.7 + 0.9))/2 = 1.25.$$

and:

$$\tilde{A} \cup \tilde{B} = ((H1, [0.4, 0.9]), (H2, [0.7, 0.8]), (H4, [0.6, 0.8]), (H5, [0.4, 1.0])),$$
$$AC(\tilde{A} \cup \tilde{B}) = ((0.4 + 0.7 + 0.6 + 0.4) + (0.9 + 0.8 + 0.8 + 1))/2 = 2.8.$$

Therefore:

$$IT2FSim(\tilde{A}, \tilde{B}) = \frac{AC(\tilde{A} \cap \tilde{B})}{AC(\tilde{A} \cup \tilde{B})} = \frac{1.25}{2.8} = 0.45,$$

which is a crisp measure of the similarity between the IT2 FSs $\tilde{A}$, and $\tilde{B}$. Such a measure is used in order to evaluate the similarity of concept extents in IT2 Fuzzy Concept Lattices.

4.2 Concept Intent Similarity

In order to address the similarity of concept intents, we need to briefly recall the notion of *information content similarity*. It is based on the well-known notion of *information content*, which has been extensively investigated in the literature [23].

4.2.1 Information Content Similarity

Let us consider a *lexical database for the English language* as, for instance, *WordNet* [9]. Besides English concept nouns, *WordNet* contains verbs, adjectives and

adverbs, each associated with the related natural language definition and *frequency*. Frequencies are estimated using noun frequencies from large text corpora, as for instance the *Brown Corpus of American English*. Concept nouns are organized according to the *ISA* and *PartOf* relationships, and for each concept noun, a set of synonyms is given. In order to deal with the *information content* approach, below we focus on (fragments of) *WordNet ISA* hierarchies and, for the sake of simplicity, without addressing sets of synonyms. The *probability* of a concept noun c, $p(c)$, is defined as:

$$p(c) = \frac{freq(c)}{M} \tag{8}$$

where $freq(c)$ is the frequency of c from a text corpus, and M is the total number of observed instances of nouns in the corpus. In this paper probabilities have been assigned according to the *SemCor* project, which labels subsections of the *Brown Corpus* to senses in the *WordNet* lexicon. In Figure 3, the simple fragment of *ISA* hierarchy presented in [13] is recalled, where each concept is associated with the related probability.

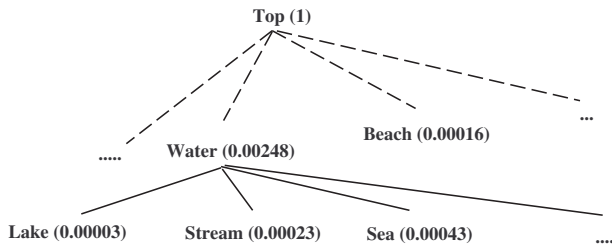


Figure 3. A fragment of *ISA* hierarchy from *WordNet* [13]

The *information content* of a concept noun c is defined as $-\log p(c)$, that is, as the probability of a concept noun increases, the informativeness decreases, therefore the more abstract a concept noun, the lower its information content. The similarity between hierarchically organized concept nouns is given by the maximum information content shared by the concepts, that is, the more information two concepts share, the more similar they are. Given a hierarchy of concept nouns organized according to a tree (also referred to as *taxonomy*), consider two concept nouns of this hierarchy, say c_1, c_2. Then, the maximum information content shared by c_1, c_2 in the taxonomy is provided by the superconcept of c_1, c_2 whose information content is maximum, i.e., the *least common superconcept* (*lcs*). In this paper we focus on concept hierarchies which are trees, therefore the *lcs* of two concept nouns always exists. Starting from these assumptions, the *information content similarity* (*ics*) of two concept nouns is defined by the maximum information content shared by the concepts divided by the information contents of the comparing concepts [23].

For instance, in the case of *Lake* and *Sea*, *Water* is their *lcs* in the hierarchy, and therefore:

$$ics(Lake, Sea) = \frac{2 \log p(Water)}{\log p(Lake) + \log p(Sea)} = \frac{2 \cdot 8.66}{14.85 + 11.18} = 0.67.$$

Below, the *ics* is used in order to compute the similarity between sets of concept nouns, i.e., between concept intents.

4.2.2 Similarity Between Sets of Attributes

In the following, since concept intents are defined by sets of attributes, we refer to attributes rather than concept nouns. The comparison between concept intents is performed according to the *Hungarian algorithm* in polynomial time [21]. Informally, given a lexical database for the English language, consider two sets of attributes, say I_1, I_2, defined in the lexical database. Let a *candidate set of pairs* be a subset of $I_1 \times I_2$ such that there are no two pairs in the set sharing an element. For instance, assume that I_1 and I_2 represent a set of boys and a set of girls, respectively, a candidate set of pairs defines a possible set of marriages (when polygamy is not allowed). Within all possible candidate sets of pairs, consider (one of) the set(s) such that the sum of the information content similarity (*ics*) of the pairs is maximal (*maximum weighted matching* problem in bipartite graphs [11]). Such a sum is indicated as $\mathcal{M}(I_1, I_2)$. Then, the similarity between the sets of attributes I_1, and I_2, $ASim(I_1, I_2)$ is defined as follows:

$$ASim(I_1, I_2) = \frac{\mathcal{M}(I_1, I_2)}{n} \tag{9}$$

where n is the greatest between the cardinalities of I_1, and I_2.

For instance in our running example, assume $I_1 = \{SwPool, Sea\}$, and $I_2 = \{SwPool, Lake\}$. In this simple case, within the two possible sets of pairs of attributes that can be formed with I_1 and I_2 as described above, the set of pairs with maximal sum is the following:

$$\{(SwPool, SwPool), (Sea, Lake)\},$$

because, of course, $ics(SwPool, SwPool) = 1$, and $ics(Sea, Lake) = 0.67$. Therefore:

$$\mathcal{M}((SwPool, Sea), (SwPool, Lake)) = 1.67,$$

whereas the other possible set of pairs:

$$\{(SwPool, Lake), (Sea, SwPool)\}$$

leads to a null value (the *ics* of both the pairs are null because, according to the *ISA* hierarchy of Figure 3, *SwPool* does not share any information content neither

with *Lake*, nor with *Sea*). As a result:

$$ASim(I_1, I_2) = \frac{\mathcal{M}(I_1, I_2)}{2} = 0.84.$$

Now we are able to evaluate the similarity between IFCA concepts, on the basis of the similarity of concept extents and the similarity of concept intents defined above.

4.3 Similarity Between IFCA Concepts

In this section, the notion of similarity between IFCA concepts, referred to as *T2ConSim*, is presented. It is essentially given by the weighted average between the similarity of the concept extents and the similarity of concept intents above. Formally, given two concepts of an IFCA Concept Lattice, namely $C_1 = (\tilde{E}_1, I_1)$, and $C_2 = (\tilde{E}_2, I_2)$, their similarity $T2ConSim(C_1, C_2)$ is defined as follows:

$$T2ConSim(C_1, C_2) = IT2FSim(\tilde{E}_1, \tilde{E}_2) \cdot w + ASim(I_1, I_2) \cdot (1 - w) \tag{10}$$

where *IT2FSim* is the similarity between the IT2 FSs $\tilde{E}_1, \tilde{E}_2$, $ASim(I_1, I_2)$ is the similarity between the sets of attributes I_1, and I_2, and w is a weight, $0 \leq w \leq 1$, defined by domain experts depending on the characteristics of the application domain. In the case of very small values of w, concept similarity is evaluated by taking into account mainly the concept intents, i.e., the sets of attributes associated with the objects of the application domain whereas, in the opposite case, values of w very close to 1 mean that the computation of similarity is performed by focusing on the specific objects of the application domain, rather than their intensional descriptions.

For instance, in our running example assume $w = \frac{1}{2}$, and consider the concept:

$$C_1 = (((H4, [0.9, 1.0]), (H6, [0.7, 0.9])), (SwPool, Sea))$$

of the Concept Lattice of Figure 2. Furthermore, consider the concept C_2 below, defined as follows:

$$C_2 = (((H2, [0.8, 1.0]), (H4, [0.7, 0.9]), (H7, [0.6, 0.8])), (SwPool, Lake))$$

and suppose C_2 belongs to a different context (it contains the object $H7$ and the attribute *Lake* which do not belong to the context *Sardinia Hotels*). The similarity between C_1 and C_2 is computed as follows:

$$IT2FSim(\tilde{E}_1, \tilde{E}_2) = \frac{AC(\tilde{E}_1 \cap \tilde{E}_2)}{AC(\tilde{E}_1 \cup \tilde{E}_2)} = \frac{0.8}{3.35} = 0.24$$

because:

$$\begin{aligned}
\tilde{E_1} \cap \tilde{E_2} &= (H4, [0.7, 0.9]), \\
AC(\tilde{E_1} \cap \tilde{E_2}) &= (0.7 + 0.9)/2 = 0.8, \\
\tilde{E_1} \cup \tilde{E_2} &= ((H2, [0.8, 1.0])(H4, [0.9, 1.0]), (H6, [0.7, 0.9]), (H7, [0.6, 0.8])), \\
AC(\tilde{E_1} \cup \tilde{E_2}) &= ((0.8 + 0.9 + 0.7 + 0.6) + (1.0 + 1.0 + 0.9 + 0.8))/2 = 3.35.
\end{aligned}$$

We have seen that:

$$ASim(I_1, I_2) = \mathcal{M}((SwPool, Sea), (SwPool, Lake))/2 = 0.84.$$

Therefore:

$$T2ConSim(C_1, C_2) = \frac{1}{2} \cdot 0.24 + \frac{1}{2} \cdot 0.84 = 0.54.$$

5 EVALUATION AND DISCUSSION

In line with the work for non-fuzzy concepts presented in [11], the information content approach and the use of a lexical database for the English language lead to a fundamental difference with respect to other proposals. In fact, in the absence of them, the evaluation of the attribute similarity (independently of the related objects), such as *Sea* and *Lake* in the example of the previous section, requires "additional knowledge" which, in general, is provided by a panel of experts in the given application domain [10]. Furthermore, note that *T2ConSim* is not a distance-based similarity measure and, in line with the notion of *information content similarity* on which it relies, the *triangle inequality* does not hold [23].

Finally, it is important to note that setting the parameter w in *T2ConSim* is a complex problem whose definition is, in general, left to the domain expert according to the context, which plays a crucial role when measuring concept similarity [20]. Within similarity measures, this topic has been addressed by several authors in different research areas as, for instance, in [29] where the problem of determining features' relevance in the context of Geographical Information Systems has been analyzed. However, the definition of (semi-)automatic criteria to evaluate context-dependent parameters is still a challenging topic which requires human expertise (and goes beyond the scope of this paper).

With regard to concept intents, which are non-fuzzy sets, it is important to recall that their similarity can also be evaluated by following several different approaches defined in the literature, as for instance *Dice*, *Jaccard*, *Cosine* [24], etc. Here we only recall the *Jaccard* measure since it is the one on which the similarity between IT2 FSs is based (of course reformulated for crisp sets), and we show the reason why it is not indicated in order to evaluate the similarity of concept intents.

Let I_1, and I_2 be two concept intents, the *Jaccard* similarity, $Jaccard(I_1, I_2)$, is defined on the basis of the cardinalities of their intersection and union sets as follows:

$$Jaccard(I_1, I_2) = \frac{\mid I_1 \cap I_2 \mid}{\mid I_1 \cup I_2 \mid}. \tag{11}$$

For instance, in our running example, consider the intents:

$$I_1 = (SwPool, Sea),$$

$$I_2 = (SwPool, Lake),$$

then, according to their intersection and union sets:

$$Jaccard((SwPool, Sea), (SwPool, Lake)) = 1/3 = 0.33.$$

Note that *Sea* and *Lake* do not contribute to the intersection since they are evaluated as different strings, independently of their semantics. Vice versa, in Subsection 4.2.2 we have seen that, according to the information content approach, the *ics* between *Sea* and *Lake* is:

$$ics(Sea, Lake) = 0.67$$

and:

$$ASim((SwPool, Sea), (SwPool, Lake)) = 0.84,$$

which is closer to human judgment. Indeed, Lin's approach has been extensively experimented in the literature and shows a higher correlation with human judgment than other methods such as Resnik, Wu and Palmer, etc., and the traditional *edge-counting* approach [23].

The impact about the use of the information content approach within OFCA has been experimented in [13]. In the mentioned paper, the experimental results show that the correlation with human judgment has an average increment of about 0.3, with respect to the compared proposals. Besides the use of the information content approach, this significant increment is due to the combination of the concept extent and the concept intent similarities.

This strong imbalance in favor of the measure proposed within OFCA, on which this approach is based, makes us optimistic for future possible experimentations and comparisons with forthcoming proposals within IFCA.

6 CONCLUSION AND FUTURE WORK

In this paper a similarity measure for IFCA concepts has been proposed. It essentially combines the similarity of concept extents, that are IT2 FSs, and the similarity of concept intents, that are sets of concept nouns. In particular, concept extents are compared according to the *IT2FSim*, that is the widely accepted crisp similarity measure for IT2 FSs, that allows a relevant simplification about the definition of similarity between general T2 FSs. Concept intents are evaluated according to the

information content approach, which has been extensively experimented in the literature and has a higher correlation with human judgment. This combination makes us confident about future comparisons with forthcoming proposals. In addition, in this paper both IT2 FS theory and IFCA have been recalled, by providing simple examples which allow to reach a broad audience of non-specialist readers.

As future work, we are arranging a wide experiment involving the students of our Institute in order to quantify the correlation of the proposed measure with human judgment that, unfortunately, is not an easy job due to the complexity of Concept Lattices.

REFERENCES

[1] Ashrafi, M.—Prasad, D. K.—Quek, C.: IT2-GSETSK: An Evolving Interval Type-II TSK Fuzzy Neural System for Online Modeling of Noisy Data. Neurocomputing, Vol. 407, 2020, pp. 1–11, doi: 10.1016/j.neucom.2020.03.065.

[2] Belohlávek, R.: What is a Fuzzy Concept Lattice? II. In: Kuznetsov, S. O. et al. (Eds.): Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC 2011). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6743, 2011, pp. 19–26, doi: 10.1007/978-3-642-21881-1_4.

[3] Benítez-Caballero, M. J.—Medina, J.—Ramírez-Poussa, E.—Ślęzak, D.: Rough-Set-Driven Approach for Attribute Reduction in Fuzzy Formal Concept Analysis. Fuzzy Sets and Systems, Vol. 391, 2020, pp. 117–138, doi: 10.1016/j.fss.2019.11.009.

[4] Berners-Lee, T.—Hendler, J.—Lassila, O.: The Semantic Web. Scientific American, Feature Article, May 2001.

[5] Burusco, A.—Fuentes-González, R.: The Study of the Interval-Valued Contexts. Fuzzy Sets and Systems, Vol. 121, 2001, No. 3, pp. 439–452, doi: 10.1016/S0165-0114(00)00059-2.

[6] Djouadi, Y.—Prade, H.: Interval-Valued Fuzzy Formal Concept Analysis. In: Rauch, J. et al. (Eds.): Foundations of Intelligent Systems, International Symposium on Methodologies for Intelligent Systems (ISMIS 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5722, 2009, pp. 592–601, doi: 10.1007/978-3-642-04125-9_62.

[7] Dubois, D.—Prade, H.: Fundamentals of Fuzzy Sets. Springer Science and Business Media, 2012.

[8] Fateminia, S. H.—Sumati, V.—Fayek, A. R.: An Interval Type-2 Fuzzy Risk Analysis Model (IT2FRAM) for Determining Construction Project Contingency Reserve. Algorithms, Vol. 13, 2020, No. 7, Art. No. 163, pp. 1–22, doi: 10.3390/a13070163.

[9] Fellbaum, C.: WordNet: An Electronic Lexical Database. The MIT Press, Cambridge, MA, 1998.

[10] Formica, A.: Ontology-Based Concept Similarity in Formal Concept Analysis. Information Sciences, Vol. 176, 2006, No. 18, pp. 2624–2641, doi: 10.1016/j.ins.2005.11.014.

[11] Formica, A.: Concept Similarity in Formal Concept Analysis: An Information Content Approach. Knowledge-Based Systems, Vol. 21, 2008, No. 1, pp. 80–87, doi: 10.1016/j.knosys.2007.02.001.

[12] Formica, A.: Semantic Web Search Based on Rough Sets and Fuzzy Formal Concept Analysis. Knowledge-Based Systems, Vol. 26, 2012, pp. 40–47, doi: 10.1016/j.knosys.2011.06.018.

[13] Formica, A.: Similarity Reasoning for the Semantic Web Based on Fuzzy Concept Lattices: An Informal Approach. Information Systems Frontiers, Vol. 15, 2013, No. 3, pp. 511–520, doi: 10.1007/s10796-011-9340-y.

[14] Formica, A.: Similarity Reasoning in Formal Concept Analysis: From One- to Many-Valued Contexts. Knowledge and Information Systems, Vol. 60, 2019, No. 2, pp. 715–739, doi: 10.1007/s10115-018-1252-4.

[15] Ganter, B.—Wille, R.: Formal Concept Analysis – Mathematical Foundations. Springer, 1999.

[16] Hao, M.—Mendel, J. M.: Similarity Measures for General Type-2 Fuzzy Sets Based on the α-Plane Representation. Information Sciences, Vol. 277, 2014, pp. 197–215, doi: 10.1016/j.ins.2014.01.050.

[17] Hitzler, P.—Krötzsch, M.—Rudolph, S.: Foundations of Semantic Web Technologies. Chapman and Hall/CRC, Taylor and Francis Group, 2009, doi: 10.1201/9781420090512.

[18] Hitzler, P.: What's Happening in Semantic Web ... and What FCA Could Have to Do with It. In: Valtchev, P., Jäschke, R. (Eds.): Formal Concept Analysis (ICFCA 2011). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6628, 2011, pp. 18–23, doi: 10.1007/978-3-642-20514-9_2.

[19] Jay, N.—Nuemi, G.—Gadreau, M.—Quantin, C.: A Data Mining Approach for Grouping and Analyzing Trajectories of Care Using Claim Data: The Example of Breast Cancer. BMC Medical Informatics and Decision Making, Vol. 13, 2013, Art. No. 130, pp. 1–9, doi: 10.1186/1472-6947-13-130.

[20] Kessler, C.: Similarity Measurement in Context. In: Kokinov, B. et al. (Eds.): Modeling and Using Context (CONTEXT 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4635, 2007, pp. 277–290, doi: 10.1007/978-3-540-74255-5_21.

[21] Kuhn, H. W.: The Hungarian Method for the Assignment Problem. Naval Research Logistics Quarterly, Vol. 2, 1955, No. 1-2, pp. 83–97, doi: 10.1002/nav.3800020109.

[22] Liang, Q.—Mendel, J. M.: Interval Type-2 Fuzzy Logic Systems: Theory and Design. IEEE Transactions on Fuzzy Systems, Vol. 8, 2000, No. 5, pp. 535–550, doi: 10.1109/91.873577.

[23] Lin, D.: An Information-Theoretic Definition of Similarity. Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98), Madison, Wisconsin, USA, Morgan Kaufmann, 1998, pp. 296–304.

[24] Maarek, Y. S.—Berry, D. M.—Kaiser, G. E.: An Information Retrieval Approach for Automatically Constructing Software Libraries. IEEE Transactions on Software Engineering, Vol. 17, 1991, No. 8, pp. 800–813, doi: 10.1109/32.83915.

[25] Mendel, J. M.: Computing with Words and Its Relationship with Fuzzistics. Information Sciences, Vol. 177, 2007, No. 4, pp. 988–1006, doi: 10.1016/j.ins.2006.06.008.

[26] Mendel, J. M.: Type-2 Fuzzy Sets and Systems: A Retrospective. Informatik-Spektrum, Vol. 38, 2015, No. 6, pp. 523–532, doi: 10.1007/s00287-015-0927-4.

[27] Mendel, J. M.—Wu, D.: Perceptual Reasoning for Perceptual Computing. IEEE Transactions on Fuzzy Systems, Vol. 16, 2008, No. 6, 1550–1564, doi: 10.1109/TFUZZ.2008.2005691.

[28] Rocco, C. M.—Hernandez-Perdomo, E.—Mun, J.: Introduction to Formal Concept Analysis and Its Applications in Reliability Engineering. Reliability Engineering and System Safety, Vol. 202, 2020, Art. No. 107002, doi: 10.1016/j.ress.2020.107002.

[29] Rodríguez, A.—Egenhofer, M. J.: Comparing Geospatial Entity Classes: An Asymmetric and Context-Dependent Similarity Measure. International Journal of Geographical Information Science, Vol. 18, 2004, No. 3, pp. 229–256, doi: 10.1080/13658810310001629592.

[30] Safaeipour, H.—Fazel Zarandi, M. H.—Turksen, I. B.: Developing Type-2 Fuzzy FCA for Similarity Reasoning in the Semantic Web. Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS), IEEE, 2013, pp. 1477–1482, doi: 10.1109/IFSA-NAFIPS.2013.6608620.

[31] Stumme, G.—Maedche, A.: FCA-MERGE: Bottom-Up Merging of Ontologies. Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI '01), Seattle, Washington, USA, Vol. 1, 2001, pp. 225–230. ISBN: 1-55860-777-3.

[32] Tho, Q. T.—Hui, S. C.—Fong, A. C. M.—Cao, T. H.: Automatic Fuzzy Ontology Generation for Semantic Web. IEEE Transactions on Knowledge and Data Engineering, Vol. 18, 2006, No. 6, pp. 842–856, doi: 10.1109/TKDE.2006.87.

[33] Wang, F.—Wang, N.—Cai, S.—Zhang, W.: A Similarity Measure in Formal Concept Analysis Containing General Semantic Information and Domain Information. IEEE Access, Vol. 8, 2020, pp. 75303–75312, doi: 10.1109/ACCESS.2020.2988689.

[34] Wu, D.—Mendel, J. M.: Uncertainty Measures for Interval Type-2 Fuzzy Sets. Information Sciences, Vol. 177, 2007, No. 23, pp. 5378–5393, doi: 10.1016/j.ins.2007.07.012.

[35] Wu, D.—Mendel, J. M.: A Comparative Study of Ranking Methods, Similarity Measures and Uncertainty Measures for Interval Type-2 Fuzzy Sets. Information Sciences, Vol. 179, 2009, No. 8, pp. 1169–1192, doi: 10.1016/j.ins.2008.12.010.

[36] Zadeh, L. A.: Fuzzy Sets. Information and Control, Vol. 8, 1965, No. 3, pp. 338–353, doi: 10.1016/S0019-9958(65)90241-X.

[37] Zadeh, L. A.: The Concept of a Linguistic Variable and Its Application to Approximate Reasoning – I. Information Sciences, Vol. 8, 1975, No. 3, pp. 199–249, doi: 10.1016/0020-0255(75)90036-5.

Anna Formica received her degree (Hons.) in mathematics from the University of Rome "La Sapienza", in 1989. She is currently Senior Researcher with the "Istituto di Analisi dei Sistemi ed Informatica" (IASI) "Antonio Ruberti", Italian National Research Council (Consiglio Nazionale delle Ricerche – CNR), Rome, where she manages the "Software and Knowledge-Based Systems" (SaKS) Group. She tooks part in various research projects of the European framework programs and bilateral projects with international institutions. Her current research interests include semantic web, similarity reasoning, formal specification and validation of domain ontologies, fuzzy formal concept analysis, geographical information systems, and e-learning. She serves as a referee for several international journals and conferences.

Computing and Informatics, Vol. 40, 2021, 489–521, doi: 10.31577/cai_2021_3_489

BIG DATA STORAGE TOOLS USING NOSQL DATABASES AND THEIR APPLICATIONS IN VARIOUS DOMAINS: A SYSTEMATIC REVIEW

Amen Faridoon

University College Dublin, Dublin, Ireland
e-mail: amen.faridoon@ucdconnect.ie

Muhammad Imran

National Centre for Physics, Islamabad, Pakistan
&
CERN, Geneva, Switzerland
e-mail: muhammad.imran@cern.ch

Abstract. Over the past few years, data has been growing significantly due to the advent of new connected devices, availability of bandwidth, and the emergence of new applications which utilize cloud computing infrastructure in the data centers. This increased amount of data faces many problems in terms of storage, transmission, management, and processing, etc. Therefore, the term big data has gained significant attention from researchers in recent years. The rapidly growing quantity, velocity, and variety of data require more probable and logical tools for its storage. For this purpose, the industry is highly emphasizing the development of more viable tools for the storage of big data. The traditional big data storage tools are unsuccessful in storing an enormous amount of data. Hence, the structural modifications of management mechanisms of conventional storage systems such as SQL databases to NoSQL databases technology are necessary to cope up with drastically increasing requirements of big data storage. The primary objective of this paper is to concentrate exclusively on designing a road map for NoSQL big data storage technologies, evaluate current evidence, research progresses in NoSQL data storage systems and their applications in various domains. We conducted a systematic literature review (SLR) of various studies published in recent years. We propose a framework to classify selected articles on the basis of various factors such as motivations behind big data storage, NoSQL techniques used for storing big data, and significant ap-

plications of big data in different domains. Furthermore, we also discuss research issues and define an outline for future research in the big data storage domain for NoSQL databases.

Keywords: Big data, storage tools, NoSQL databases, systematic literature review

1 INTRODUCTION

Over the past few years, big data has gained significant attention of the researchers and industrial experts as world has faced challenges related to big data storage, transmission, management, processing, analysis, visualization, integration, architecture, security, quality and privacy. Big data is an abstract concept. People still have different opinions on the term "big data" but data scientists and experts explain it by five main characteristics which are called 5Vs [58]. These are volume, variety, velocity, veracity and value. The volume represents the quantity of data [45]. Many organizations already have tremendous quantity of archive data but they are not able to efficiently store and process it. Processing of a large amount of data is the main attraction of the big data analytics [36]. The variety refers to the format of the data [45]. The dataset format is divided into three main categories, i.e. structured, semi-structured and unstructured data. Various sources generate data in multiple formats, e.g. videos, audios, documents, comments, logs, tabular form and others. The velocity defines the increase in speed of data generation and processing [45]. Social media platforms and real time applications are some good examples that illustrate data generation with fast speed. The veracity covers the quality or accuracy of the data [5]. While dealing with the large quantity, velocity and diversity of the data, it is impossible that all of the data is 100 % correct rather there will be noisy data. The value is the very necessary aspect of the big data. The value basically states the worth of the data being extracted or extracting meaningful insight from the data. Having access to the big data is good but at the same time it would be useless if we are not able to turn it into the value. [5]

1.1 Challenges of Traditional Systems with Big Data

Traditional systems are not capable to store and process big data efficiently. This is because the traditional storage systems were not designed for such data. Society faces many problems with the traditional storage systems, some of them are

1. schema-on-write,
2. cost of storage,
3. cost of proprietary hardware,
4. complexity,

5. heterogeneous data,
6. causation and
7. bringing data to the programs.

Moreover, in order to reduce the fraction of the cost for traditional shared storage, we have to reduce the size of the data. As a result, after large volumes of the data thrown out causes reduction in the data to be analyzed which ultimately decreases accuracy and confidence of the results. Hence, most of the world has learned that the traditional storage systems are not upright for storing and analyzing huge amount of data.

1.2 Big Data Storage Technologies and Their Features

As the traditional storage systems fail in the era of the big data, the scientists have been looking up for modern systems to overcome this problem [53]. Big data storage is a storage infrastructure that is specifically designed for massive amount of data for storage, processing and retrieval. Big data storage tools are the basic driver for advance analysis and have the capacity to transmute the society. Some of the storage tools are:

1. BigTable,
2. HBase,
3. Cassandra,
4. MongoDB,
5. Neo4j, etc.

These tools meet the demand of storage and allow us to store heterogeneous data. These tools also have certain properties like scalability, access control, fault tolerance, failover recovery, real time query, SQL supported queries, distributed nature and much more [10]. Big data storage infrastructures provide the solutions for the problems that are faced in traditional systems. Solutions of the challenges discussed above in Section 1.1 are:

1. schema-on-read,
2. reducing the storage cost,
3. commodity hardware which overcomes the cost of proprietary hardware,
4. simplicity,
5. allow unstructured and semi-structured data to be stored and processed,
6. correlation and
7. bringing programs to the data.

1.3 Applications Using Big Data Storage Technologies

Many sources that include business applications, public web, social media, machine log data, transactions, sensor data and some real time applications generate big volume of the data continuously [13, 1]. Big data storage technologies cross the barriers and are used to handle the problems of big data. These technologies are not only important in IT-based sector but also have particular importance in non-IT-based sector like energy, health and finance etc. International data corporation (IDC) predicts that total amount of digital data created worldwide will grow from approximately 44 zettabytes in 2020 to 180 zettabytes in 2025 [55] due to the growing number of smart devices, sensors and availability of bandwidth [18]. Moreover, in one day, internet of things (IoT) and the use of connected devices generate huge amount of data. Approximately 500 million tweets and 294 billion emails are sent in a day. Facebook creates 4 petabytes of data. The 5 billion searches are made and 65 billion messages are sent on WhatsApp. The 4 petabytes of data are created from each connected car. In addition, weather channels receive 18 055 556 forecast requests. Venmo processed 51 892 peer-to-peer transactions. Uber riders take 45 788 trips and there are 600 new page edits to Wikipedia. These all happens in one day. The aggregate of space need to save one second video of a high quality is two thousand (2 000) times more than the space needed to save a plain text of page. Some of the predictions made by IDC regarding 2025 termed as IDC Data Age 2025 predictions are shown in Table 1.

Year	Numbers	Predictions	Organization
2025	>150 billion	More than 150 billion devices will be connected across the globe.	IDC Data Age 2025
2025	>6 billion	Consumers will interact with data every day.	IDC Data Age 2025
2025	90 zettabytes	In 2025 IoT devices will generate more than 90 zettabytes of data.	IDC Data Age 2025
2025	30 %	Comparing to 15 % in 2017 nearly 30 % of all data created will be real-time.	IDC Data Age 2025
2020	90 %	Large enterprises will generate revenue from data as a service.	IDC FutureScape: Worldwide IT Industry 2018 Predictions

Table 1. IDC Data Age 2025 predictions

1.4 Existing Issues of Big Data Storage Tools

Although NoSQL data tools have sorted out the issues of big data comprehensively but the velocity is the one area that still requires improvements. This is because of the ever increasing number of big data sources and heterogeneous data created from these sources [13]. Security and privacy are also the well-recognized challenges in the big data storage as the data is stored in the clusters of the data centers not in the users' personal devices. The compatibility and updating data are the requirements that big data storage tools must also fulfill [42]. Scalability is also a major challege which arises due to rapid change in the growth of data and this is handled by adding more storage to an existing node. Furthermore, the data consistency, single server storing of data and partitioning are the most imperative research challenges.

1.5 Our Contributions

Systematic literature review (SLR) uses systematic methods to collect secondary data, critically appraise research studies, and synthesize findings qualitatively or quantitatively. In this paper, we perform SLR on NoSQL big data storage tools and their applications in various domains. The primary objective of this SLR is to concentrate exclusively on designing a road map for NoSQL big data storage technologies, evaluate current evidences, research progresses in big data storage systems and their applications in various domains. We selected various publications from 2015 to 2020 on the basis of this SLR and classify these chosen articles on the basis of factors involved in the movement of SQL-to-NoSQL, frameworks used for storing massive data and significant applications in different industries. Furthermore, we also highlight research gaps and define outline for future research.

In summary, the main objectives of this SLR are following:

- To assist data scientist and researchers to understand and choose the storage framework that effectively suits to their requirements,
- To present the significance and applications of the NoSQL big data storage technologies to particular domain,
- To highlight issues or challenges of the NoSQL database systems,
- To discuss directions for future research in the big data storage.

This SLR entirely depends on framed questions, related studies, analyzing their findings, and gives a brief evidence through clear methodology. In this paper, we categorize the storage tools according to data model such as key-value, columnar, document oriented and graph stores. For evaluation, features or properties of the storage systems are being used. In this paper, we also highlight the use cases of these tools in different domains.

1.6 Needs for Conducting SLR

The need for an SLR entails to create a technology roadmap, analyzing current publications and the research progression in the NoSQL big data storage systems. It exclusively focuses on classification of storage technologies and their applications in different domains. To demonstrate that a similar review has not been already reported, we search the Compendex, IEEE Xplore, ACM, ScienceDirect, SpringerLink and Google Scholar digital libraries with the help of search string shown in Figure 1.

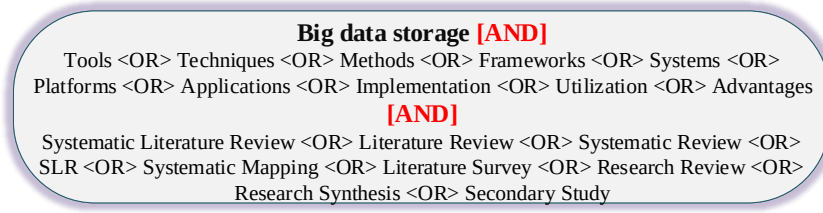


Figure 1. Search string for searching the similar SLRs

After searching the digital libraries of above mentioned databases by means of search string, we come across that none of the subsidiary studies was related to any of the research questions discussed in Section 2.

The remainder of the paper is organized as follows: Section 2 describes the investigation methodology, research questions and the extent to which we stretched our search for the studies. Section 3 provides answer to our research questions that we raised and focus on the results of our research methodology. Finally, we conclude in Section 4.

2 RESEARCH METHODOLOGY

SLR eliminates bias as compared to unstructured review process and also follows an extremely thorough and accurate sequence of steps to search literature. The primary aim of this SLR is to structure the existing literature in the domain of NoSQL big data storage systems and their applications. To conduct this SLR we adopted the guidelines presented in the study [8] with a three-step review process which includes planning, conducting and documenting, as shown in Figure 2. For the SLR to be effective – we obtain, analyze and document the outcome and evaluate our review protocols. The planning and conducting phases of the systematic literature review procedure are described below.

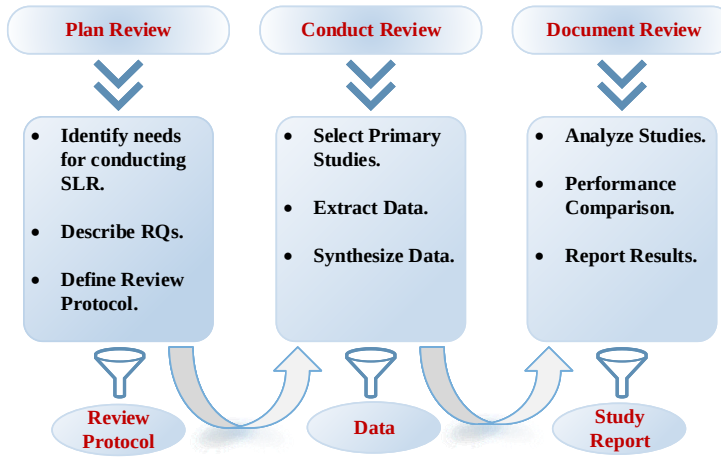


Figure 2. Outline of our research methodology

2.1 Review Plan

An essential element in conducting a systematic literature review is to establish a protocol for the study during the planning phase. Planning phase focuses the underlying principles for the recognition of the needs for a methodical review and results in a review protocol as follows.

Step 1: Characterization of review demands. We specify the research questions in Table 2 that provide the foundations for defining and assessing the review protocol. The research questions depend on our motivation to identify evidence about the distinct storage frameworks and their implementations used by the authors in the big data storage.

We describe the overall objectives and scope of our investigative study by showing them in Figure 3. We consider PICOC (population, intervention, comparison, outcome and context) criteria for this purpose [38].

Step 2: Analyze and specify review protocol. Based on the goals, we define the research questions and the scope of the review that lead us to design the search string for the extraction of the literature. Before the execution of the review protocol, we evaluated it by external experts. Therefore, we invited specialists/professionals to provide feedback who have capabilities for conducting and designing the SLRs in our area of investigation. Their assessment is reflected in the refined protocol.

	Research Questions	Objectives
RQ1	What are the main motivations behind big data storage?	We are interested to know what are main reasons that traditional storage systems failed in the era of big data.
RQ2	What are the existing NoSQL technologies and their key features used in big data storage?	The aim is to investigate and categorize the storage technologies according to database structure.
RQ3	What are the applications of NoSQL big data storage models in various domains?	The aim is to investigate the socio-economic influences of large scale storage systems of data.
RQ4	What are the current research problems and what should be the future research agenda in big data storage?	The main objective is to disclose the research flaws which need to be addressed and potential future directions in this area.

Table 2. Research questions mapped by objectives

Criteria	RQ1	RQ2	RQ3	RQ4
Population or problem	Practical motivation.	Storage technologies and their key features.	Application of storage technologies in various domain.	Research issues and future direction.
Intervention or Exposure	Internal/External validation; Extracting data and synthesis.			
Comparison	Perform comparison between applications of storage tools and evaluate the storage tools on the basic of their key features.			
Outcome	Categorize big data storage technologies. Grouping their applications. Hypotheses for future research.			
Context	A systematic investigation with an exclusive focus on storage technologies and their implementation in different areas.			

Figure 3. Define objective and scope of SLR via PICOC criteria

2.2 Review Conduction

The second phase of research methodology is the review conduction which is initiated by choosing the literature and its outcome is the extraction of data and synthesized information. This phase is further divided into three steps that are:

1. selection of primary research articles,
2. data extraction, and
3. synthesis of the results.

Step 1: Selection of primary research articles. By considering the research questions described in Table 2 and taking the guidelines from study [8] we developed a search string. After applying the search string to the digital libraries of five databases, i.e. IEEE Xplore, ScienceDirect, SpringerLink, ACM and Google Scholar,

we obtained 1 900 articles from 2015 to 2020. The search string and the results from various databases are shown in Figure 4.

		NAME	RESULTS
Storage	Migration <OR> Evaluation <OR> Switching <OR> Traditional Systems <OR> Transformation <OR> Modernization <OR> Relational Systems <OR> SQL-to-NoSQL **<OR>** Technologies <OR> Techniques <OR> Tools <OR> Platforms <OR> Frameworks <OR> Methods <OR> Approaches <OR> NoSQL Stores **<OR>** Applications <OR> Implementation <OR> Utilization <OR> Advantages <OR> Employment **<OR>** Issues <OR> Problems <OR> Difficulties <OR> Challenges **[AND]** Large Data <OR> Massive Data <OR> Macro Data <OR> Large Volume of Data <OR> Large Amount of Data <OR> Lots of Data	IEEE	552
		Springer	448
		Science Direct	138
Big Data		ACM	220
		Google Scholar	542
		Total	1900

Figure 4. Composition of search strings and search results

Initial Selection. This step comprises of inspecting the titles and abstracts of primary articles. In initial selection stage, we came across 136 papers. However further screening of these 136 papers was done against the inclusion/exclusion criteria which is presented in Table 3.

	Inclusion		Exclusion
I1	Scientific peer-reviewed papers are included.	E1	Studies that do not explicitly discuss storage technologies and their features.
I2	Studies that discuss technologies and their key features used in data storage.	E2	Studies that conduct survey on storage technologies.
I3	Studies that conduct experiment on storage technologies.	E3	Editorials, abstracts, courses and papers less than 5 pages.
I4	Studies that mention the issues or problems of big data storage technologies.	E4	Non-peer-reviewed articles.
		E5	Non-English manuscripts.
		E6	Thesis and book chapters.

Table 3. Inclusion/exclusion criteria

Final Selection. Final selection depends on our objectives and goals for big data storage areas which consist of storage systems and their key features, employment of storage systems in various domains, issues or challenges related to storage systems and reasons behind the failure of traditional storage systems for big data. After the completion of final selection process, we selected 33 studies.

We only considered articles for which full text is available and the studies having abstract only are discarded. We also exclude the publications related to text books, editorials, courses, non-peer-reviews, articles not written in English and the papers less than five pages.

Qualitative Assessment of Included Studies. After applying the above mentioned criteria, 33 research articles are filtered out from the database contained 136 papers. Then the screening of these 33 articles have been done through the quality assessment criteria presented in Figure 5. We categorize our quality assessment criteria into two portions that are "General characteristics for quality assessment" and "Specific characteristics for quality assessment". According to the general assessment criteria, we comprehensively reviewed the selected research papers, examined their purpose of the research, and observed that the study effectively described their proposed methodology according to their problem statement and presented results are inline with the contributions of the study. However, through the specific quality assessment criteria, we have to examine: do the research papers define the limitations of traditional storage systems over the dimensions of the big data?, do they clearly investigate the NoSQL big data storage technologies and their use cases in various domains?, and does the study mention the research problems of the storage systems?

	General characteristics for quality assessment	**Score**		
		Yes=2	Partially =1	No=0
G1	Is problem statement and research motivation clearly defined?			
G2	Are methodology of the research and its organization comprehensively described?			
G3	Is the research environment of the study clearly explained?			
G4	Are the presented results of the research inline with the contributions of the study?			
	Specific characteristics for quality assessment			
S1	Is the research successfully described the drawbacks of traditional data storage in the dimensions of the big data?			
S2	Are the research effectively investigate the existing big data storage technologies?			
S3	Are the study productively employed the big data storage systems in a particular domain?			
S4	Are the limitations of present research clearly indicated?			

Figure 5. Quality assessment criteria

3 DISCUSSION AND RESULTS

The Table 4 shows the detailed summary of selected 33 research articles. In this section, we present the discussion and findings on these selected papers on the basis of our research questions.

ID	Title	Year	Journal/ Conf.	RQs	Cs.
S1 [63]	Application and research of massive big data storage system based on HBase	2018	IEEE Xplore	RQ2	–
S2 [27]	Distributed storage system for electric power data based on HBase	2018	Big Data Mining and Analytics	RQ3	–
S3 [54]	A new method for time-series big data effective storage	2017	IEEE Access	RQ1, RQ2	12
S4 [28]	MongoDB-Based Repository Design for IoT-Generated RFID/Sensor Big Data	2015	IEEE Sensors Journal	RQ3	74
S5 [11]	An OAIS-based hospital information system on the cloud: Analysis of a NoSQL column-oriented approach	2017	IEEE Journal of Biomedical and Health Informatics	RQ3	16
S6 [62]	Big data storage and management in SaaS applications	2017	Journal of Communications and Information Networks	RQ1, RQ2	4
S7 [22]	Optimization strategy of Hadoop small file storage for big data in healthcare	2016	The Journal of Supercomputing	RQ2, RQ3	24
S8 [26]	An Approach to Security for Unstructured Big Data	2016	The Review of Socionetwork Strategies	RQ4	1
S9 [43]	Privacy preserving data publishing based on sensitivity in context of Big Data using Hive	2018	Journal of Big Data	RQ3, RQ4	–
S10 [59]	A Big Data platform for smart meter data analytics	2019	Computers in Industry	RQ3	2
S11 [16]	HB-File: An efficient and effective high-dimensional big data storage structure based on US-ELM	2017	Neurocomputing	RQ4	3
S12 [46]	Dynamic Preclusion of Encroachment in Hadoop Distributed File System	2015	Procedia Computer Science	RQ4	5
S13 [57]	A new reliability model in replication-based big data storage systems	2017	Journal of Parallel and Distributed Computing	RQ4	9
S14 [3]	Dynamic Merging based Small File Storage (DM-SFS) Architecture for Efficiently Storing Small Size Files in Hadoop	2018	Procedia Computer Science	RQ2, RQ4	1
S15 [35]	BigDimETL with NoSQL Database	2018	Procedia Computer Science	RQ2, RQ3	–
S16 [21]	Compression and Security in MongoDB without affecting Efficiency	2016	ICTCS	RQ2, RQ4	1

S17 [61]	RDBMS, NoSQL, Hadoop: A Performance-Based Empirical Analysis	2016	AMECSE	RQ1, RQ2, RQ3	5
S18 [20]	Performing OLAP over Graph Data: Query Language, Implementation, and a Case Study	2017	BIRTE	RQ2, RQ3	1
S19 [7]	Enabling Scientific Data Storage and Processing on Big-data Systems	2015	IEEE	RQ3	4
S20 [14]	Using the column oriented NoSQL model for implementing big data warehouses	2015	Semantic Scholar	RQ3	28
S21 [56]	Query-oriented Adaptive Indexing Technique for Smart Grid Big Data Analytics	2017	Journal of Signal Processing Systems	RQ3	2
S22 [60]	A Performance-improved and Storage-efficient Secondary Index for Big Data Processing	2017	2017 IEEE Inte. Conf. Smart Cloud	RQ3, RQ4	–
S23 [34]	A Versatile Event-Driven Data Model in HBase Database for Multi-Source Data of Power Grid	2016	2017 IEEE Inte. Conf. Smart Cloud	RQ3, RQ4	2
S24 [44]	A Framework for Migrating Relational Datasets to NoSQL	2015	Procedia Computer Science	RQ1	26
S25 [41]	Comparative Study of SQL and NoSQL Databases	2015	Inte. J. Adv. Res. Comp. Engi. Tech	RQ1, RQ2	9
S26 [49]	SQL Versus NoSQL Movement with Big Data Analytics	2016	Inte. J. Info. Tech. Comp.Sci	RQ1, RQ2	11
S27 [6]	Handling Big Data using NoSQL	2015	29th Inte. Conf. Adv. Info. Net. Appl.	RQ1, RQ2	37
S28 [51]	Aerospike: architecture of a real-time operational DBMS	2016	Proc.VLDB Endowment	RQ2	18
S29 [15]	A scalable generic transaction model scenario for distributed NoSQL databases	2015	Journal of Systems and Software	RQ2	12
S30 [52]	Analysis of various NoSql database	2015	ICGCIoT	RQ2	14
S31 [33]	Statistical analysis of tourist flow in tourist sports based on big data platform and DA-HKRVM algorithms	2020	Personal and Ubiquitous Computing	RQ2, RQ3	1
S32 [39]	Distributed Data Platform for Automotive Industry: A Robust Solution for Tackling Challenges of Big Data in Transportation Science	2019	ConTEL	RQ2, RQ3	1
S33 [30]	Multilevel Object Tracking in Wireless Multimedia Sensor Networks for Surveillance Applications Using Graph-Based Big Data	2019	IEEE Access	RQ2, RQ3	3

Table 4. List of selected studies

3.1 What Are the Main Motivations Behind Big Data Storage? (RQ1)

Our classification of research articles has allowed to counter our first research question (RQ1). We identify main reasons behind big data storage focused by the researchers in their studies. On the ground of literature, we note that relational databases have lost their popularity because of well-structured nature. The motivation and need behind using the big data storage technologies is that the relational, well-structured, well-schema [S3, S6, S17] databases could not develop as rapid as big data. The large volume of data comes with their own challenges [50] such as real-time processing, fast recovery, fault tolerance and complex structure of data is not fulfilling expectations and needs for heterogeneous data [S3, S17, S26]. In addition, the schema of relational and structured databases does not assist the recurrent changes. Some of the well-known network data repositories are Google, Twitter, Amazon and Facebook. There, management and dynamic scalability requirements exceed the capabilities of relational databases [40]. Thus, frequently growing and developing data needs a better and satisfactory solution. In order to fulfill the needs of big data, NoSQL databases have emerged which not only overcome the problems of relational datasets, rather the also have become mainly adopted frameworks for storing large scale data. Unlike the traditional databases, these frameworks have capacity and ability to deal with multiple users interacting with big data simultaneously. They also provide assurance for some distinctive characters over relational datasets like distributed scalability, availability, fault tolerance, consistency, data replication, parallel data processing, flexibility, multiple servers, non-relational distributed data models, efficient, high performance, cost saving and secondary indexing.

Based on data synthesis, we identify four primary factors which are scalability (32 %), availability (21 %), schema less (21 %) and data replication (16 %) of the studies [S3, S6, S17, S24, S25, S26], that clearly mentioned the SQL verses NoSQL movement, as shown in Figure 6. We also perform comparison between relational database systems and NoSQL big data storage models and this is shown in Table 5. This analysis clearly demonstrates that big data storage technologies have more worth than relational database systems.

3.2 What Are the Existing NoSQL Technologies and Their Key Features Used in Big Data Storage? (RQ2)

We answer RQ2 by considering the NoSQL storage technologies from selected research articles. The remaining part regarding this question involves remarkable features of these storage technologies. Big data storage tools are referred to as the storage tools that in measure particularly address the quantity, speed, or verity of challenges and do not drop under the heading of traditional storage systems. This does not specify that traditional storage systems do not solve these problems but other storage tools like NoSQL databases (column oriented store, Key-value store, document oriented store and graph oriented store) [S17, S25, S26] are often more methodical and less expansive. These technologies offer scalable storage solutions

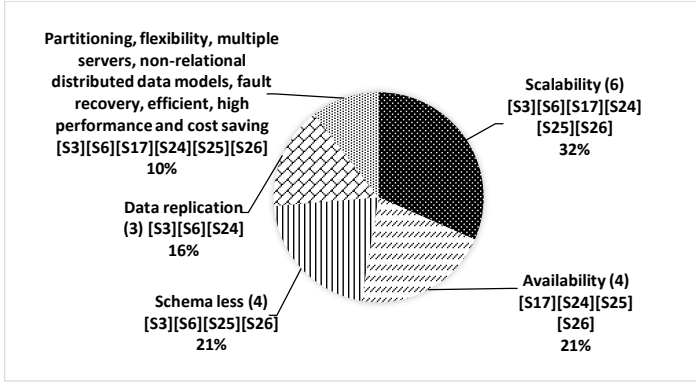


Figure 6. Percentage of studies with respect to various drivers behind big data storage

	Traditional Database Systems	NoSQL Storage Systems
Strengths	Store structured data. Vertical scalability. Extendable processing on a server. Particular schema. Particular DML (Data Manipulation Languages).	Store un-structured, semi-structured and structured data. Scalability. Extendable commodity servers. Parallel computing applications. High availability and fault tolerance. Reliability. Simultaneous accessibility and consistency.
Weaknesses	Bottleneck performance and delays in processing. With the growth of data, chances of occurring deadlock increases. Limited storage capacity. Difficulty in join operations for multidimensional data.	Due to scalability and performance not compliance with ACID (atomicity, consistency, isolation, durability).
Opportunities	Traditional data storage systems support complex queries. Built-in deployment. Atomicity in complex database transactions.	Simplicity in complex storage structures. Enhance response time for query.
Threats	With the dynamic growth of the data large volume for storage is required. Complex and Schema less data structures. Real-time processing and maintaining consistency for large number of storage servers are the main threats for relational storage systems.	Deployment of big data storage systems. Small files in large numbers is one more threat.

Table 5. SWOT analysis of traditional and big data storage architectures

for continuously growing big data with further improved data structures quality wise and support fault tolerance. After the keen survey of literature we categorize big data storage tools according to the classification of NoSQL data models. A brief summary and comparison of these tools are also presented in Table 6.

NoSQL databases. NoSQL database management systems are the most imperative family of big data storage technologies. When the storing and processing of large data is a primary requirement then the NoSQL is the preferred choice. NoSQL databases introduce data models from outside the relational world. These models are flexible in nature, provide horizontal scalability, schema-less and aim to manage large amount of data. NoSQL databases can be divided into four major categories based on their data model. [S25, S27, S30]

3.2.1 Key-Value Stores

This is the simplest NoSQL data store model. Key-value data models allow data to store in a schema-less way [S17, S25, S26, S27, S30]. Because of no schema, it is not compulsory that the data objects share the same structure. It may be completely structure or un-structure and can be assessed by a single key. In key-value store model, "Key" represents unique value to access row and value contain the information which corresponds to that key. Key-value model shows flexibility to add more records easily and to consume less memory storage because of different representations of values in records. Aerospike, Riak and Redis and many others are the examples of key-value database model.

Aerospike. Aerospike is the name of eponymous company that produces it. It is the open-source, in-memory, first flash-optimized key-value data storage software and is more suitable for storing real-time data[51]. It is written in C language and operates in three layers namely flash optimize data layer, distribution layer and cluster aware client layer. To ensure the consistency, the distribution layer is replicated across data centers. Whereas, client layer is responsible for managing communication in the server node and it is also used to track the cluster configuration in the database. The highlighted goals behind the development of Aerospike are to design a scalable and flexible framework for web applications and support reliability and consistency like a traditional database. [S27, S28, S29]

Riak. Riak [48] is also a NoSQL database storage system that provides high availability, fault tolerance, operational simplicity and scalability in a very low cast. In addition, it can be used to store data in memory, disk or both. It uses the term bucket and key for interaction with objects. The key is utilized to store the values whereas the bucket is for setting the bucket's properties. The Riak distributes the data throughout the cluster by computing the binary hash of each bucket/key pair and maps the calculated value to a location on an ordered ring of all such values. In multi-datacenters replication, one cluster acts as a primary

cluster and is responsible for handling requests from one or more secondary clusters. If the primary cluster goes down, then secondary cluster can take place of it. Moreover, the objective of Riak developers is to provide high availability of diverse data to applications. [S25, S26, S27, S30]

Redis. Original developer of the Redis is Salvatore Sanfilippo. Linux platform is selected for the development and testing of Redis [17]. It can provide powerful properties such as fast access to whole data resides in the memory, built-in persistency. Its distinctive feature is to support multiple datatypes. Thus, Redis is the most suitable option for heterogeneity in servers and application, and where in-memory data is the requirement. It is designed for efficiently supporting query operations and replication in a master-slave environment. [S17, S30]

3.2.2 Columnar Stores

Google's BigTable is the motivation behind the column family stores. The basic architecture of columnar store data model is rows and columns, and any number of key-value pairs can be stored within rows. Rows and columns both are split over multiple nodes to achieve scalability. Columns can be grouped to column families while rows are fragmented over nodes according to primary key. Columnar store systems provide efficient data compression and partition, and particularly perform well with aggregation queries such as sum, count, average, etc. The most important advantage of columnar store models is the scalability. They are well suited for parallel processing where data is distributed over thousands of clusters and are extremely fast in data loading. HBase, Cassandra, Hypertable, BigTable etc. are the examples of columnar stores. [S17, S25, S26, S27, S29, S30]

BigTable. BigTable is a compressed, high performance and proprietary data storage system developed by Google Inc [12]. The prominent properties of BigTable are flexibility, adoptability, reliability, high performance storage for a structured large-scaled data distributed over the commodity servers, and applicable storage and manage petabyte of data on thousands of machines. It is design for the distribution of highly scalable and structured data. BigTable is applicable to store large amount of structure data at google, web pages and many other google products. [S17, S26, S30]

HBase. HBase [29] is a column oriented, non-relational and distributed database model written in Java which is capable of managing structure and un-structure data. It was developed by Apache. The objective behind its designing is to handle big data storage needs in Apache project [19]. HBase runs on the top of Hadoop distributed file system (HDFS). It provides scalability, distribution, fault recovery and random read/write access to stored data. HBase architecture is composed of at least one master server responsible for management and assign regions to region servers and several slave servers to store data. The most prominent feature of HBase is support to read-intensive transactions. Motivation behind the development of HBase is to provide random, consistent and real-time

access to scalable BigTable with intensive read and write operations. LinkedIn is the use case of HBase. [S1, S17, S25, S26, S30, S31]

Hypertable. Hypertable [29] is an open source software inspired by the design of Google BigTable. It is written in C++ and runs on the top of distributed file system such as HDFS, GFS and CloudStore. It provides good support to consistency of stored data in terms of tables, and divides these tables to acquire scalability and distribution. The importance of Hypertable is that when the master becomes fail to respond for a brief time period and it has no effect on client data transfer. It is designed to provide parallel, scalable databases and better query performance for large size data. [S26, S30]

Cassandra. Initially, Cassandra was developed at Facebook to power the inbox search feature. Now it has become an Apache incubator project. Cassandra [23, 32] is a distributed, wide columnar store NoSQL database management system designed to handle large amount of data across many data centers or commodity servers. It has multiple features like scalability, instants storage, improved frequent read and write operation requests, achieve data consistency through periodic updates on replicating sites, and reliability achieves over large-scale systems [2]. However, the most prominent is high availability with no single point of failure, and fault tolerance and reduce latency which are achieved through clustering, partitioning and replication. [S6, S25, S26, S30]

3.2.3 Document Stores

Document oriented stores are one of the main categories of NoSQL database storage systems designed for storing, retrieving and managing document oriented information which is also called semi-structured data. Document stores are schema less and support secondary indexes. They are inherently a sub-class of key-value store but they support more complex data then key-value stores. In contrast with relational database, they store all information for a given object in a single instance in the database while relational databases store information in separate tables define by the programmer. MongoDB, SimpleDB, CouchDB and others are the examples of document oriented databases. [S3, S6, S16, S17, S25, S26, S27, S29, S30]

MongoDB. MongoDB [37] is an open-source, cross-platform document oriented database developed by Mongo Inc. MongoDB uses JSON like documents with the characteristics of MySQL. MongoDB stores documents in the form of binary representation known as BSON. When the primary server is failed, multiple replicas are considered to achieve the availability of data. Main features provided by MongoDB are adhoc queries (support field, range query and regular expression searches), indexing (primary and secondary indices are used to indexing the document), through the replica sets. Some of the MySQL properties acquired by MongoDB with slight modifications are high availability, load balancing, file storage, aggregation, dynamic updates etc. Aim of the MongoDB is

to provide relational data model facilities to document-oriented databases. [S3, S6, S16, S17, S25, S26, S27, S30, S32]

SimpleDB. SimpleDB is an open-source, distributed, document oriented database which is written in Erlang programming language. It is developed by Amazon Inc [47]. High availability, durability, data model flexibility and automatic indexing are most noticeable features of SimpleDB. Along with features, SimpleDB also have some limitations as compared to the consistency of other database management systems. SimpleDB provide eventual consistency also known as optimistic replication. To overcome the problem of eventual consistency, two new operations are introduced in 2010 that are conditional put and delete, and consistent read. Developers goal is to offer geographic replication for data availability and durability. It is used for complex queries, logs and online games. [S27, S29, S30]

CouchDB. CouchDB is an open source, NoSQL database software having a scalable architecture [4]. It is implemented in a concurrent oriented language Erlang. The main goals of developing CouchDB is to perform data operations and management on the web. CouchDB store any kind of data as documents, and each document has its own self-contained schema. Bi-directional replication and off-line operation were the two goals in the developer's mind at the time of designing the CouchDB. ACID properties of database, built for offline, document storage, eventual consistency, map/reduce views and indexes are the key features of CouchDB. [S17, S25, S26, S30]

3.2.4 Graph-Oriented Stores

Graph database stores are the part of the NoSQL databases, created to overcome the limitations of relational databases and are superlative choice to store data along with relations. The key concept behind the system is graph, that narrates the data items to a collection of nodes and edges. However, nodes represent the data items and edges represent the relationship between the nodes. Relationships between the data items allow stored data to linked together directly and most of the time data is retrieved with one operation. Graph search specific portion according to the execution of query, it does not search irrelevant data. Therefore, it improves the performance of the graph databases systems. Neo4j, InfiniteGraph, HyperGraphDB [31] and many more are the example of graph oriented stores. [S18, S25, S26, S27, S30]

Neo4j. Neo4j is an open source [9], graph database management system introduced by Neo4j, Inc. Neo4j is the effective replacement of relational databases. Scalability, concurrency, transaction load and read request loads are the highlighted properties of Neo4j system. It not only performs improvement on its older version but also competes other graph databases. With the help of buffering and without blocking, Neo4j supports to write-intensive transactions. [S18, S25, S26, S30, S33]

InfiniteGraph. Infinite [24] is a commercial, distributed graph database which is implemented in Java. InfiniteGraph is useful to find hidden relationships in highly connected big data sets. It can store growing data with some schema to further perform normalization and other presentation operations. To achieve scalability, InfiniteGraph implements graph model (Labeled directed multi-graph) technique. While, other key features of InfiniteGraph are concurrency, distribution, multi-threaded, cloud enabled, parallel query support, fully ACID, and having some schema. Easy traversal of complex relationships and provision of support for complex queries over high value data are the main goals for the development of InfiniteGraph. [S27, S30]

HyperGraphDB. HyperGraphDB [25] is an open source data storage mechanism designed for the knowledge management, artificial intelligence and semantic web projects. This graph database provides storage mechanism for random data and also support data mapping between host language and storage. By providing the customizable indexing feature, efficient graph traversal and data retrieval are achieved. However, for storing the graph information Key-value mechanism is used like nodes and edges of a graph are used as a key. In distinction to master-slave storage systems like HBase, Hypertable, Redis HyperGraphDB implements a peer-to-peer data distribution mechanism. [S26, S27, S30]

Summary of Storage Tools. The summary of storage tools is described in Table 6. The table highlights summary with respect to the preferred and non preferred areas, systems, vendors, licence, goals and applications of NoSQL databases for big data storage technologies.

The preferred area of key-value storage is user profile maintenance having no specific schema. It is also suitable for managing a large amount of small-sized data records of web applications like managing session information for online shopping and online games, etc. Moreover, searching for more attributes rather than one from records is the appropriate use case for key-value storage. However, frequent updates, query specific data values, and establish relationships of data values with each other are not suitable areas for key-value data models. Perform analysis to aggregate homogeneous data items is the most common application area for columnar stores. Furthermore, e-library, patient record management, customer data analysis, online attractive applications, write-intensive processing applications, and others are the use cases for column-oriented databases. Despite that, we should avoid column-store systems where the applications need complex queries. The most common applications of the document-oriented data model are maintaining social data, analyzing websites, content management, and e-commerce systems. However, this NoSQL model is not preferred where transactions with multiple operations are required. Recommendation systems, social networks, bioinformatics, pattern mining, and semantic web projects are the applications of the graph-oriented data model. Moreover, it is also preferred for location-based networks and real-time search. However, the use of such a store must be avoided where data cannot be modeled as a graph.

Data Model	Preferred Areas	Not Preferred Areas	Systems	Vendor	License	Goals	Applications
Key-value store	User profile maintenance having no specific schema. Section data for users. Shopping cart's data storage.	Need to be queried specific data value. Frequent updates. Establish relationships of data values with each other.	Aerospike	Aerospike, Inc.	Open Source	Designing a scalable and flexible framework for web applications. Support reliability and consistency like a traditional database.	Web applications
			Riak	Basho Technologies	Open Source	Objective of Riak is to provide high availability to applications.	Diverse data
			Redis	Salvatore Sanfilippo	Open Source	Redis is designed for master-slave environment to efficiently support query operations and replication.	Used for small structured data.
Wide-column	Blogging platforms are the use case of wide column. Counter-based and content management systems. Write intensive processing applications	Applications needed complex querying and has varying patterns queries. Avoid column stores systems where the database requirement is not established	HBase	Apache	Open Source	Motivation behind the development of HBase is to provide random, consistent and real-time access to scalable BigTable with read and write operations.	Latency tolerant applications, sparse and versioned data are the main areas of HBase. LinkedIn also use HBase.
			Cassandra	Apache	Open Source	Aim behind the development of Cassandra is to provide distributed, fault tolerance and highly available storage for data and improved access performance through replication and row distribution of data.	Online interactive applications like Facebook, twitter etc.
			Hypertable	Zvents	Open Source	Designed to provide parallel, high-performance, scalable databases, and better querying performance for large size data.	Store and maintain both structured and un-structured data
			BigTable	Google	Proprietary	Design for the distribution of highly scalable, structured data.	Used to store structure large volume data at google, web pages, and many google products.
Document oriented	Content management and e-commerce systems. Blogging and analytics platforms.	Applications needed complex search queries and transactions with multiples operations	MongoDB	MongoDB, Inc.	Open Source	Developed to provide fast and consistent data access from different applications across multiple interfaces. Another goal is to provide relational data models facilities to document oriented databases.	Real-time applications.
			SimpleDB	Amazon	Open Source	Offer geographic replication for data availability and durability.	Used for complex queries, logs and online games.
			CouchDB	Apache	Open Source	For web documents, make available a dynamic and self-contained schema.	Web applications and social data are the focusing areas.

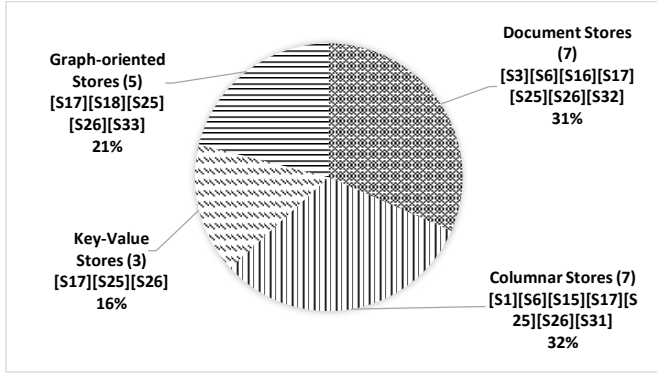
Graph stores	Graph based searches and IT operations are the use cases of graph oriented stores. Fraud detection. Social networks.	Use of such a store must be avoided where data cannot be modeled as a graph	Neo4j	Neo Technology	Open Source	To provide relation-like graph, data relationship manipulation and decision making.	Social networks and recommendations systems.
			Hyper GraphDB	Kobrix Software, Inc.	Open Source	Relational and object oriented data management and memory model for artificial intelligence and semantic web projects are the reasons behind HyperGraphDB.	Bioinformatics, pattern mining and semantic web projects are the applications of HyperGraphDB.
			Infinite Graph	Objectivity, Inc.	Commercial	Easy traversal of complex relationships and provide complex queries over high value data are the main goals.	Preference domains are Social and location-based networks, and real-time search.

Table 6. Summary of storage tools

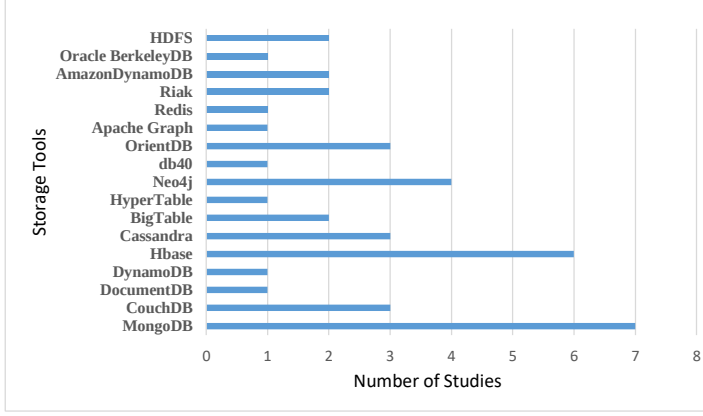
According to the selected studies, we conclude that document and columnar stores are the most frequently used NoSQL databases having the percentage of 31 % and 32 %, respectively, as shown in Figure 7 a). Whereas the graph-oriented and key-value stores are less used database in terms of percentage, i.e. 21 % and 16 %, as compared to document and columnar stores. Many of the researchers used MongoDB and CouchDB document stores with the number of 7 and 3, respectively, in their studies. In columnar stores HBase, Cassandra and BigTable storage tools gain the attention of the researchers for storing big data, as shown in Figure 7 b).

Most Repeated Features of NoSQL Models. Above mentioned big data storage technologies have storage structures to assist scalable resource configuration of big data. Most of the storage systems are developed to ensure availability, consistency, fault tolerance, flexibility, reliability and the durability in general. It can be deduced from Figure 8 that scalability, schema less, calculated performance, low cost, partitioning, data replication, accessibility and sharding are the most repeated features according to the selected studies.

Specific Features of NoSQL Models. NoSQL storage models have specific properties such as scalability, shared-nothing architecture, persistence, partitioning, in-memory, on disk or both memory and disk storage, and rigorous read and write. Figure 9 highlights the specific features of above mentioned NoSQL storage technologies. We can observe that all of the Key-value data models are in-memory, shared-nothing architecture, and scalable rather than the Redis. The Redis provides automatically data partitioning through horizontal scaling whereas, vertical scaling is difficult in it. The Aerospike does not provide a persistence feature while data is stored in memory, however we can persist the data by using persistence memory like disk or device storage. Databases included in the category of wide-column and document-oriented stores support maximum features presented in the



a) Number of studies mentioned NoSQL databases



b) Usage of storage tools in selected publications

Figure 7. NoSQL storage technologies

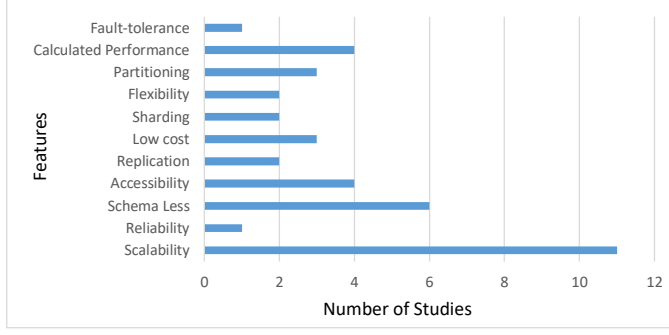


Figure 8. Most repeated features of storage tools

Figure 9. However, SimpleDB only allows persistence data and scalable systems. While, graph-oriented databases like Neo4j, HyperGraphDB, and InfiniteGraph do not back the partitioning and data persistence features.

Storage Systems	Memory Storage	Disk Storage	Intensive Read/Write	Persistence	Partitioning	Shared nothing Architecture	Scalability
Aerospike	✓	x	✓	x	✓	✓	✓
Riak	✓	✓	x	✓	x	✓	✓
Redis	✓	x	✓	✓	✓	✓	x
HBase	x	✓	x	✓	✓	x	✓
Cassandra.	✓	x	✓	x	✓	✓	✓
Hypertable	✓	✓	✓	✓	✓	✓	✓
BigTable	✓	✓	✓	✓	✓	✓	✓
MongoDB	✓	✓	✓	x	✓	✓	✓
SimpleDB	x	x	x	✓	x	x	✓
CouchDB	x	✓	✓	✓	✓	✓	✓
Neo4j	✓	✓	✓	✓	x	✓	✓
HyperGraphDB	✓	✓	-	x	✓	-	✓
InfiniteGraph	-	-	✓	x	✓	✓	✓

Figure 9. Specific features of storage tools

3.3 What Are the Applications of NoSQL Big Data Storage Models in Various Domains? (RQ3)

Big data storage technologies have very significant applications in various domains like real-time big data, time-series data, content management, customer 360 view, mobile applications, fraud detection and others. Many industries are now adopting big data storage technologies like NoSQL databases technologies for critical business applications. These technologies are gradually taking place of relational database technologies to acquire better features like scalability, flexibility, replication, partitioning etc. Some of the applications of big data storage technologies mentioned in under studied articles are discussed below.

Internet of Things (IoT). IoT (the internet of things) states that the control of automated intelligent and inter linked devices command over wide regions through sensors and other computing capabilities. The data produced by IoT is characterized by its continuous growth, unstructured and huge amount. Traditional database technologies are not capable enough to handle such a huge amount of IoT generated data and if you cannot store this heterogeneous data streaming in every second, you would not be able to accomplish any tasks on it. Thus big data storage technologies such as HBase, MongoDB, Cassandra etc. are normally based on distributed file system, database management and data processing technologies, have emerged as a fundamental technology to implement IoT generated data. Selected study [S4] uses MongoDB for storing the

multi-source IoT data sources such as RFID (Radio frequency identification), sensor and GPS (Global positioning systems). In addition, they also devise an effective shared key to maximizing the query speed and horizontally distribute data over data servers.

Healthcare. There is a huge amount of data associated with health sector and it has to be processed and stored. With the progress in health systems, they are continuously moving towards the effective digital solutions. The main objective behind this is to efficiently manage data resources and information associated with health processes. The study [S5] implements OAIS healthcare architecture based on NoSQL column-oriented Cassandra database management system and provides a way to handle such a big amount of HL7 clinical documents in a scalable manner. Moreover authors conduct case study for finding the blood glucose level and assembled results are stored in OAIS system to monitor health condition of patients and to halt deaths. In study [S7], the aim of the authors is to devise a method for the short files storage of genomic and clinical data that will help researchers to execute analytics in healthcare. The given method incorporates the small files of data block and after merging stores these big files so that to reduce the data blocks of HDFS.

Decision Making. We have experienced an immense amount of data on the web. This is because of speedy technological advances with the accessibility of smart devices and social networks like Instagram, Twitter, Facebook, etc. These social sites enable us to make effective decisions. The authors in study [S15] perform ETL (Extract-Transform-Load) operations with HBase to store tweets by using join algorithms. Results highlight the ETL operation execute well with join operation to make effective business decisions. Similarly, the authors in study [S20] perform decision queries on star schema benchmark (SSB) data warehouse and considered HBase columnar NoSQL database for storing purpose.

Electric Power Data. One of the big data storage application is managing enormous electric power data. Electric power systems consist of billions of devices nowadays. These devices generate hundred and thousand of records in a single day. For ensuring the security and maintenance of these power systems, huge amount of data from large number of data sources required to be properly processed and inspected so that a rapid decision could be made in real time. According to study [S2], authors proposed a system through which electric power data can be stored effectively by using HBase. Proposed system is used to monitors a status, and also to perform data migration and fragmentation. The proposed system of study [S10] is capable for storing, quering, visualizing and analyzing large scale smart grid power data sets. Results obtained from this are compared with IBM, MongoDB, Google Cloud and AmpLab provides comparatively easy platform to handle such a big electric power data, with ability of decision making. In studies [S21, S22, S23] researchers devised a unique method to store enormous amount of data by gaining the advantage of HBase. The proposed [S21, S22] systems not only increase the query process but also prevent

space for storage. While authors in study [S23] proposed a data model in which join operation is integrated by using virtual column family.

The number and percentage of selected studies with respect for various domains using NoSQL big data storage models is shown in Figure 10.

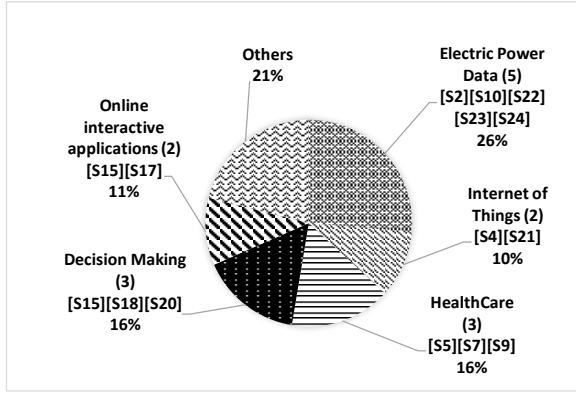


Figure 10. Number of studies contain big data storage applications

3.4 What Are Existing Research Issues and What Should Be the Future Research Agenda in Big Data Storage? (RQ4)

There are many advanced data storage technologies which help us a lot in storing big data but they are not yet perfect platforms. Undoubtedly, the new data storage technologies offer a lot of benefits over conventional data storage technologies, but these technologies are not better enough. There is no ideal platform to be used as a better storage solution. Information extracted from the selected articles help us to answer our research question four (RQ4). Here is a brief look at the existing research issues and distinctions in NoSQL big data storage technologies.

3.4.1 Future Research Challenges

Security. Top level existing research issues include providing security to a stored data. Security of a data is considering as a big challenge on any platform. Big data storage technologies are facing a fairly handsome list of issues regarding security. Many security issues will likely be solved as data storage technologies continue to grow. But for now, security is a distinction in big data storage technologies. Authors in study [S8] provide security suit which contains various algorithms. Providing security to unstructured data, authors take data from Wikipedia and Google search API by taking various data types into account and their sensitivity level. Providing security to Hadoop complex distributed

file system is a challenging task. The main focus of study [S12] is to provide a security model to ensure the variety of secure data operations like insertion, deletion and replication of data over clusters. Likewise, for storing files of small size in HDFS, authors in study [S14] implement the encryption technique known as Twofish to ensure the security of content present in files.

Advantages of MongoDB are that it provides replication, schema-less, supports indexing and many more but it also has some limitations related to security of a data stored in it. So authors in study [S16] played a part in resolving the problem by introducing a middleware encryption before storing the data into database.

Issues relating to security in big data storage systems need to be further investigated into with respect to different tools.

Read Performance. Authors in study [S22, S23] highlighted the issues (join operation, effective indexing and random read) related to HBase. To overcome the limitation of indexing schema authors in study [S22] implement secondary indexing technique to speed up query processing and save the huge storage space. A virtual column family is introduced to resolve the problems of random read and join operation in HBase [S23].

Similar issues relating to performance during fetching data from big data storage systems with respect to different bid data storage tools need to be evaluated.

Data Management. With the escalation of big data, the related data storage industries emphasize more on data management instead of computational management. In recent time, managing a data is a big task. Many data storage technologies have been proposed which help us a lot in storing a growing data and processing resources. However, still more efficient technologies are required for data acquisition, processing, pre-processing, storage and management of big data. The continuing development on big data management focuses mainly on bringing effective solutions that support big data efficiently. Management of growing volume of data is also very significant in this regard beside processing, pre-processing and storage. The main concerns of ongoing development include methods of data clustering, replication and indexing for effective storage exploitation and data retrieving.

Data Consistency. Data consistency is considered as a design goal for big data storage technologies. In distributed systems, consistency and availability have greater impact on each other and one of them is compromised. Data consistency remains a basic task for big data storage technologies. Like, NoSQL databases do not perform ACID transaction, a technique used for ensuring data consistency. In general, data consistency is a major issue in big data storage that needs to be addressed.

Scalability. The term scalability refers to handle and support increasing volume of data in such a way that a prominent optimization in the storage resources

is possible. Scalability is considered as one of the significant design goal for data storage technologies. Existing technologies have better scalability standards over traditional data storage technologies but in many respects, scalability is still a challenge. For instance, some NoSQL databases are not better enough at automating the process of sharding (spreading a database across multiple nodes). Other databases like SQL are also facing the same type of problem.

Single Server Storing of Data. Storing a big data under a single server is not a better decision while considering a nature of data. It is wise to configure a cluster of multiple hardware elements as the distributed storage system.

Frequent Data Update and Schema Change. With the rapidly growing volume of data, the need for increasing the update rate for data is also very high. Changes made in schema is also very communal in case of unstructured data. However, existing storage technologies are better in scalability but requirement to be efficient in data updates and schema is still under consideration.

Partitioning Method. Maintaining acceptable performance in growing size of the database become more complex. So the partitioning is the method to manage busy and large amount of data. Two types of partitioning are offered by the data models that are horizontal and vertical partitioning applied on data based on access patterns. However, during the execution access patterns might be wrong. Thus existing data models identified for big data storage solutions show that partitioning is a critical research challenge.

4 CONCLUSION

Big data is an abstract concept. Experts categorized big data by 5Vs referred to as volume, variety, velocity, veracity and value. As the data is growing continuously and rapidly, so this increased quantity, speed and diverse nature of data require more reliable and logical tools for its storage. The main objective of this survey is to refocus, probe and analyze the futuristic NoSQL big data storage models. We conducted SLR by selecting 33 publications from year 2015 to 2020 on the basis of our defined criteria. The primary and major objective of this SLR is to reconcentrate on the storage tools, mentioned the applications and spot the challenges of storage systems. We categorize our selected publications on the basis of four major questions.

The main concern of our first research question is to highlight the factors that are scalability, availability, schema less, data replication etc. involved in the migration of traditional tools to big data storage systems. According to the selected studies, most of the researchers frequently used document and columnar store 31 % and 32 %, respectively, NoSQL databases. The results clearly show that 14 out of 33

publications mostly used MongoDB, HBase, CouchDB, Cassandra and Neo4J storage tools. Scalability, schema less, calculated performance, partitioning, low cost and accessibility are among the most repeated features of storage tools. Big data storage tools play a consequential role in many fields. But the results gather from our selected publications tells us that smart power grid, healthcare, decision-making, online interactive applications and internet of things are the major domains where their applications are extensively used.

We have recognized that there has been an extraordinary research work done over the years by researchers. However, there are many flaws that still need to be fixed in terms of security, privacy, read performance, data management, data consistency, scalability, single server data storage, frequent update and data partitioning.

REFERENCES

[1] Abouzeid, A.—Bajda-Pawlikowski, K.—Abadi, D.—Silberschatz, A.—Rasin, A.: HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. Proceedings of the VLDB Endowment, Vol. 2, 2009, No. 1, pp. 922–933, doi: 10.14778/1687627.1687731.

[2] Abramova, V.—Bernardino, J.: NoSQL Databases: MongoDB vs Cassandra. Proceedings of the International C* Conference on Computer Science and Software Engineering (C3S2E '13), ACM, 2013, pp. 14–22, doi: 10.1145/2494444.2494447.

[3] Ahad, M. A.—Biswas, R.: Dynamic Merging Based Small File Storage (DM-SFS) Architecture for Efficiently Storing Small Size Files in Hadoop. Procedia Computer Science, Vol. 132, 2018, pp. 1626–1635, doi: 10.1016/j.procs.2018.05.128.

[4] Anderson, J. C.—Lehnardt, J.—Slater, N.: CouchDB: The Definitive Guide: Time to Relax. O'Reilly Media, Inc., 2010.

[5] Ishwarappa—Anuradha, J.: A Brief Introduction on Big Data 5VS Characteristics and Hadoop Technology. Procedia Computer Science, Vol. 48, 2015, pp. 319–324, doi: 10.1016/j.procs.2015.04.188.

[6] Bhogal, J.—Choksi, I.: Handling Big Data Using NoSQL. 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, 2015, pp. 393–398, doi: 10.1109/waina.2015.19.

[7] Biookaghazadeh, S.—Xu, Y.—Zhou, S.—Zhao, M.: Enabling Scientific Data Storage and Processing on Big-Data Systems. 2015 IEEE International Conference on Big Data (Big Data), 2015, pp. 1978–1984, doi: 10.1109/BigData.2015.7363978.

[8] Brereton, P.—Kitchenham, B. A.—Budgen, D.—Turner, M.—Khalil, M.: Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain. Journal of Systems and Software, Vol. 80, 2007, No. 4, pp. 571–583, doi: 10.1016/j.jss.2006.07.009.

[9] Buerli, M.: The Current State of Graph Databases. Department of Computer Science, California Polytechnic State University, San Luis Obispo, December 2012.

[10] Cattell, R.: Scalable SQL and NoSQL Data Stores. ACM SIGMOD Record, Vol. 39, 2011, No. 4, pp. 12–27, doi: 10.1145/1978915.1978919.

[11] Celesti, A.—Fazio, M.—Romano, A.—Bramanti, A.—Bramanti, P.—Villari, M.: An OAIS-Based Hospital Information System on the Cloud: Analysis of a NoSQL Column-Oriented Approach. IEEE Journal of Biomedical and Health Informatics, Vol. 22, 2018, No. 3, pp. 912–918, doi: 10.1109/jbhi.2017.2681126.

[12] Chang, F.—Dean, J.—Ghemawat, S.—Hsieh, W. C.—Wallach, D. A.—Burrows, M.—Chandra, T.—Fikes, A.—Gruber, R. E.: Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems, Vol. 26, 2008, No. 2, Art. No. 4, 26 pp., doi: 10.1145/1365815.1365816.

[13] Chen, C. L. P.—Zhang, C.-Y.: Data-Intensive Applications, Challenges, Techniques and Technologies: A Survey on Big Data. Information Sciences, Vol. 275, 2014, pp. 314–347, doi: 10.1016/j.ins.2014.01.015.

[14] Dehdouh, K.—Bentayeb, F.—Boussaid, O.—Kabachi, N.: Using the Column Oriented NoSQL Model for Implementing Big Data Warehouses. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2015, pp. 469–475.

[15] Dharavath, R.—Kumar, C.: A Scalable Generic Transaction Model Scenario for Distributed NoSQL Databases. Journal of Systems and Software, Vol. 101, 2015, pp. 43–58, doi: 10.1016/j.jss.2014.11.037.

[16] Ding, L.—Liu, Y.—Han, B.—Zhang, S.—Song, B.: HB-File: An Efficient and Effective High-Dimensional Big Data Storage Structure Based on US-ELM. Neurocomputing, Vol. 261, 2017, pp. 184–192, doi: 10.1016/j.neucom.2016.06.080.

[17] Excoffier, L.—Lischer, H. E. L.: Arlequin Suite Ver 3.5: A New Series of Programs to Perform Population Genetics Analyses under Linux and Windows. Molecular Ecology Resources, Vol. 10, 2010, No. 3, pp. 564–567, doi: 10.1111/j.1755-0998.2010.02847.x.

[18] Gantz, J.—Reinsel, D.: The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. IDC iView: IDC Analyze the Future, December 2012, pp. 1–16.

[19] George, L.: HBase: The Definitive Guide: Random Access to Your Planet-Size Data. O'Reilly Media, Inc., 2011.

[20] Gómez, L.—Kuijpers, B.—Vaisman, A.: Performing OLAP over Graph Data: Query Language, Implementation, and a Case Study. Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '17), ACM, 2017, Art. No. 6, doi: 10.1145/3129292.3129293.

[21] Hasija, H.—Kumar, D.: Compression and Security in MongoDB Without Affecting Efficiency. Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (ICTCS '16), ACM, 2016, Art. No. 96, doi: 10.1145/2905055.2905155.

[22] He, H.—Du, Z.—Zhang, W.—Chen, A.: Optimization Strategy of Hadoop Small File Storage for Big Data in Healthcare. The Journal of Supercomputing, Vol. 72, 2016, No. 10, pp. 3696–3707, doi: 10.1007/s11227-015-1462-4.

[23] Hewitt, E.: Cassandra: The Definitive Guide. O'Reilly Media, Inc., 2010.

[24] InfiniteGraph. Infinitegraph – Distributed Graph Database. 2014.

[25] Iordanov, B.: HyperGraphDB: A Generalized Graph Database. In: Shen, H. T. et al. (Eds.): Web-Age Information Management (WAIM 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6185, 2010, pp. 25–36, doi: 10.1007/978-3-642-16720-1_3.

[26] Islam, M. E.—Islam, M. R.—Ali, A. B. M. S.: An Approach to Security for Unstructured Big Data. The Review of Socionetwork Strategies, Vol. 10, 2016, No. 2, pp. 105–123, doi: 10.1007/s12626-016-0067-6.

[27] Jin, J.—Song, A.—Gong, H.—Xue, Y.—Du, M.—Dong, F.—Luo, J.: Distributed Storage System for Electric Power Data Based on HBase. Big Data Mining and Analytics, Vol. 1, 2018, No. 4, pp. 324–334, doi: 10.26599/BDMA.2018.9020026.

[28] Kang, Y.-S.—Park, I.-H.—Rhee, J.—Lee, Y.-H.: MongoDB-Based Repository Design for IoT-Generated RFID/Sensor Big Data. IEEE Sensors Journal, Vol. 16, 2016, No. 2, pp. 485–497, doi: 10.1109/jsen.2015.2483499.

[29] Khetrapal, A.—Ganesh, V.: HBase and Hypertable for Large Scale Distributed Storage Systems. Department of Computer Science, Purdue University, 2006.

[30] Küçükkeçeci, C.—Yazici, A.: Multilevel Object Tracking in Wireless Multimedia Sensor Networks for Surveillance Applications Using Graph-Based Big Data. IEEE Access, Vol. 7, 2019, pp. 67818–67832, doi: 10.1109/access.2019.2918765.

[31] Kaliyar, R. K.: Graph Databases: A Survey. International Conference on Computing, Communication and Automation, IEEE, 2015, pp. 785–790, doi: 10.1109/ccaa.2015.7148480.

[32] Lakshman, A.—Malik, P.: Cassandra: A Decentralized Structured Storage System. ACM SIGOPS Operating Systems Review, Vol. 44, 2010, No. 2, pp. 35–40, doi: 10.1145/1773912.1773922.

[33] Li, D.—Deng, L.—Cai, Z.: Statistical Analysis of Tourist Flow in Tourist Spots Based on Big Data Platform and DA-HKRVM Algorithms. Personal and Ubiquitous Computing, Vol. 24, 2020, No. 1, pp. 87–101, doi: 10.1007/s00779-019-01341-x.

[34] Liu, B.—Zhu, Y.—Wang, C.—Chen, Y.—Huang, T.—Shi, W.—Li, M.—Mao, Y.: A Versatile Event-Driven Data Model in HBase Database for Multi-Source Data of Power Grid. 2016 IEEE International Conference on Smart Cloud (SmartCloud), IEEE, 2016, pp. 208–213, doi: 10.1109/smartcloud.2016.28.

[35] Mallek, H.—Ghozzi, F.—Teste, O.—Gargouri, F.: BigDimETL with NoSQL Database. Procedia Computer Science, Vol. 126, 2018, pp. 798–807, doi: 10.1016/j.procs.2018.08.014.

[36] Manyika, J. et al.: Big Data: The Next Frontier for Innovation, Competition, and Productivity. Report, McKinsey Global Institute, 2011.

[37] MongoDB. MongoDB Architecture Guide (White Paper), 2015.

[38] Petticrew, M.—Roberts, H.: Systematic Reviews in the Social Sciences: A Practical Guide. John Wiley and Sons, 2008, doi: 10.1002/9780470754887.

[39] Pevec, D.—Vdovic, H.—Gace, I.—Sabolic, M.—Babic, J.—Podobnik, V.: Distributed Data Platform for Automotive Industry: A Robust Solution for Tackling Big Challenges of Big Data in Transportation Science. 2019 15th International Conference on Telecommunications (ConTEL), IEEE, 2019, pp. 1–8, doi: 10.1109/contel.2019.8848542.

[40] Pokorny, J.: NoSQL Databases: A Step to Database Scalability in Web Environment. International Journal of Web Information Systems, Vol. 9, 2013, No. 1, pp. 69–82, doi: 10.1108/17440081311316398.

[41] Pore, S. S.—Pawar, S. B.: Comparative Study of SQL and NoSQL Databases. International Journal of Advanced Research in Computer Engineering and Technology, Vol. 4, 2015, No. 5, pp. 1747–1753.

[42] Putnik, G.—Sluga, A.—ElMaraghy, H.—Teti, R.—Koren, Y.—Tolio, T.—Hon, B.: Scalability in Manufacturing Systems Design and Operation: State-of-the-Art and Future Developments Roadmap. CIRP Annals, Vol. 62, 2013, No. 2, pp. 751–774, doi: 10.1016/j.cirp.2013.05.002.

[43] Rao, P. S.—Satyanarayana, S.: Privacy Preserving Data Publishing Based on Sensitivity in Context of Big Data Using Hive. Journal of Big Data, Vol. 5, 2018, No. 1, Art. No. 20, doi: 10.1186/s40537-018-0130-y.

[44] Rocha, L.—Vale, F.—Cirilo, E.—Barbosa, D.—Mourão, F.: A Framework for Migrating Relational Datasets to NoSQL. Procedia Computer Science, Vol. 51, 2015, pp. 2593–2602, doi: 10.1016/j.procs.2015.05.367.

[45] Russom, P.: Big Data Analytics. TDWI Best Practices Report, Fourth Quarter, Vol. 19, 2011, No. 4, pp. 1–34.

[46] Saranya, S.—Sarumathi, M.—Swathi, B.—Victer Paul, P.—Sampath Kumar, S.—Vengattaraman, T.: Dynamic Preclusion of Encroachment in Hadoop Distributed File System. Procedia Computer Science, Vol. 50, 2015, pp. 531–536, doi: 10.1016/j.procs.2015.04.027.

[47] Sciore, E.: SimpleDB: A Simple Java-Based Multiuser Syst for Teaching Database Internals. ACM SIGCSE Bulletin, Vol. 39, 2007, No. 1, pp. 561–565, doi: 10.1145/1227504.1227498.

[48] Sheehy, J.: Riak 0.10 Is Full of Great Stuff. 2010.

[49] Venkatraman, S.—Fahd, K.—Kaspi, S.—Venkatraman, R.: SQL Versus NoSQL Movement with Big Data Analytics. International Journal of Information Technology and Computer Science, Vol. 8, 2016, No. 12, pp. 59–66, doi: 10.5815/ijitcs.2016.12.07.

[50] Skoulis, I.—Vassiliadis, P.—Zarras, A. V.: Growing Up with Stability: How Open-Source Relational Databases Evolve. Information Systems, Vol. 53, 2015, pp. 363–385, doi: 10.1016/j.is.2015.03.009.

[51] Srinivasan, V.—Bulkowski, B.—Chu, W.-L.—Sayyaparaju, S.—Gooding, A.—Iyer, R.—Shinde, A.—Lopatic, T.: Aerospike: Architecture of a Real-Time Operational DBMS. Proceedings of the VLDB Endowment, Vol. 9, 2016, No. 13, pp. 1389–1400, doi: 10.14778/3007263.3007276.

[52] Srivastava, P. P.—Goyal, S.—Kumar, A.: Analysis of Various NoSQL Database. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), IEEE, 2015, pp. 539–544, doi: 10.1109/icgciot.2015.7380523.

[53] Subramaniyaswamy, V.—Vijayakumar, V.—Logesh, R.—Indragandhi, V.: Unstructured Data Analysis on Big Data Using Map Reduce. Procedia Computer Science, Vol. 50, 2015, pp. 456–465, doi: 10.1016/j.procs.2015.04.015.

[54] Tahmassebpour, M.: A New Method for Time-Series Big Data Effective Storage. IEEE Access, Vol. 5, 2017, pp. 10694–10699, doi: 10.1109/access.2017.2708080.

[55] Tulchinsky, I.: The Age of Prediction. WorldQuant Journal, 2017.

[56] Wang, C.—Zhu, Y.—Ma, Y.—Qiu, M.—Liu, B.—Hou, J.—Shen, Y.—Shi, W.: A Query-Oriented Adaptive Indexing Technique for Smart Grid Big Data Analytics. Journal of Signal Processing Systems, Vol. 90, 2018, No. 8-9, pp. 1091–1103, doi: 10.1007/s11265-017-1269-z.

[57] Wang, J.—Wu, H.—Wang, R.: A New Reliability Model in Replication-Based Big Data Storage Systems. Journal of Parallel and Distributed Computing, Vol. 108, 2017, pp. 14–27, doi: 10.1016/j.jpdc.2017.02.001.

[58] White, M.: Digital Workplaces: Vision and Reality. Business Information Review, Vol. 29, 2012, No. 4, pp. 205–214, doi: 10.1177/0266382112470412.

[59] Wilcox, T.—Jin, N.—Flach, P.—Thumim, J.: A Big Data Platform for Smart Meter Data Analytics. Computers in Industry, Vol. 105, 2019, pp. 250–259, doi: 10.1016/j.compind.2018.12.010.

[60] Wu, H.—Zhu, Y.—Wang, C.—Hou, J.—Li, M.—Xue, Q.—Mao, K.: A Performance-Improved and Storage-Efficient Secondary Index for Big Data Processing. 2017 IEEE International Conference on Smart Cloud (SmartCloud), 2017, pp. 161–167, doi: 10.1109/smartcloud.2017.32.

[61] Yassien, A. W.—Desouky, A. F.: RDBMS, NoSQL, Hadoop: A Performance-Based Empirical Analysis. Proceedings of the 2nd Africa and Middle East Conference on Software Engineering (AMECSE'16), ACM, 2016, pp. 52–59, doi: 10.1145/2944165.2944174.

[62] Zheng, X.—Fu, M.—Chugh, M.: Big Data Storage and Management in SaaS Applications. Journal of Communications and Information Networks, Vol. 2, 2017, No. 3, pp. 18–29, doi: 10.1007/s41650-017-0031-9.

[63] Pan, Z.—Zhao, L.: Application and Research of Massive Big Data Storage System Based on HBase. 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 2018, pp. 219–223, doi: 10.1109/icccbda.2018.8386515.

Amen Faridoon is currently Ph.D. student in computer science at the University College Dublin in Ireland. She received her Bachelor's degree in computer science from the Govt. Girls Post Graduate College No. 1, Abbottabad, Pakistan in 2018. She then worked as Research Assistant at National Centre for Physics, Islamabad, Pakistan in 2019. Her research interests include machine learning, big data, data mining and cloud computing.

Muhammad Imran received his Ph.D. degree in electronic engineering from the Dublin City University, Ireland, in 2017. He is currently working in CMS Offline Computing Group at CERN, Geneva, Switzerland since October 2019. In addition, he holds a permanent position as Senior Scientific Officer in the National Centre for Physics, Pakistan since July 2008. His research interests include cloud computing, cluster computing, big data, data science, software engineering, SDN, and optical networks.

Computing and Informatics, Vol. 40, 2021, 522–542, doi: 10.31577/cai_2021_3_522

A DECENTRALIZED AUTHORITATIVE MULTIPLAYER ARCHITECTURE FOR GAMES ON THE EDGE

Aleksandar Tošić

University of Primorska, The Andrej Marušič Institute
Muzejski Trg 2, 6000 Koper, Slovenia
&
Innorenew CoE
e-mail: aleksandar.tosic@upr.si

Jernej Vičič

University of Primorska, The Andrej Marušič Institute
Muzejski Trg 2, 6000 Koper, Slovenia
&
Research Centre of the Slovenian Academy of Sciences and Arts
The Fran Ramovš Institute
e-mail: jernej.vicic@upr.si

Abstract. With the ever growing number of edge devices, the idea of resource sharing systems is becoming more appealing. Multiplayer games are a growing area of interest due to the scalability issues of current client-server architectures. A paradigm shift from centralized to decentralized architectures that would allow greater scalability has gained a lot of interest within the industry and academic community. Research on peer to peer network protocols for multiplayer games was mainly focused on cheat detection. Previously proposed solutions address the cheat detection issues on a protocol level but do not provide a holistic solution for the architecture. Additionally, existing solutions introduce some level of centralization, which inherently introduces single point of failures. We propose a blockchain-based, completely decentralized architecture for edge devices with no single point of failure. Our solution relies on an innovative consensus mechanism based on verifiable delay functions that additionally allows the network to derive verifiable randomness.

We present simulation results that show the assignment of players and referees to instances is pseudo-random, which inherently prevents collusion-based cheats and vulnerabilities.

Keywords: Edge computing, consensus, peer to peer, network protocol, multiplayer games, blockchain

Mathematics Subject Classification 2010: 68T50

1 INTRODUCTION

The gaming industry is worth almost 135 billion at the time of writing [7]. The same source predicts a steady 10 % growth in the next 2 years, reaching 180 billion by the end of 2021. The recent trends toward multiplayer games have been very successful with games like Fortnight earning more than 2.4 billion in revenue in 2018 alone [26]. Steam, the biggest game distribution platform reported it serves as much as 18.5 million clients concurrently. Cloud computing enabled servers need to be migrated real time in order to meet the demand of clients. Additionally, network latency was reduced due to localisation approaches where servers are spawned geographically close to clients if possible. However, maintaining a player base of thousands or even millions together with the hardware and software infrastructure is both very expensive and difficult to maintain [29]. The recent idea of a "sharing economy" can be applied in tandem with the paradigm shift to edge computing. More specifically, clients on the edge of the system can profit from sharing resources, such as bandwidth and computing power, thereby releasing the burden on centralized servers.

This can be achieved by using a peer to peer (P2P) architecture. P2P gaming architectures have been studied extensively but have not been widely adopted [29]. The main issues are closely related to the lack of authority and trust. Centralized architectures solve these issues with authoritative servers. The server's tasks are to simulate game play, validate and resolve conflict in the simulation, and store the game state. P2P multiplayer architectures were previously able to address some of the cheating vectors but required some level of centralization.

More recently, blockchain technology has gained a significantly large interest. Research in cryptography and fault tolerant consensus mechanisms has been driving the evolution of decentralized P2P systems.

The already available schemes that, at least theoretically, address most of the identified cheats in distributed gaming architectures RACS [28] and Goodman [12] still retain a central authority either to store the game state or as a refereeing authority and, thus, still retain the Single Point Of Failure – (SPOF) [8] property. Our research presented in this paper mostly focuses in the elimination of the SPOF but still being able to successfully address the same set of cheats. We were able to

achieve the set goals and were even able to partially address the Collusion cheat, as described in Section 5.7.

2 STATE OF THE ART

Baughman et al. [3] propose an improvement of their lock-step protocol [2] Asynchronous Synchronization (AS), the first protocol for providing cheat-proof and fair play-out of centralized and distributed network games. The protocol also provides implicit robustness in the face of packet loss. At proving the correctness of their approach, they make a number of assumptions: there exists a reliable channel between all players; all players know of all other players; players are able to authenticate messages from each other player; and all players wait only a finite time before making decisions and revealing commitments. Their approach can be implemented in a true peer-to-peer fashion, thus eliminating the SPOF, but, as it can be seen from Table 1, the approach is not immune to Replay/Spoof cheat [28].

GauthierDickey et al. [11] present a protocol designed to improve on lock-step protocol [2] by reducing latency while continuing to prevent cheating. They achieve this by adding a voting mechanism to compensate for packet loss in the environment. They call this protocol New Event Ordering (NEO).

Corman et al. [6] present SEA protocol and argue that it outperforms NEO algorithm in all cheat prevention properties; further, they present three possible cheats that the NEO protocol fails to address: Attacker can replay updates for another player. Attacker can construct messages with any previously seen votes attached. Since the votes are signed, the messages will appear to come from another player. Attacker can send different updates to different opponents.

Cronin et al. [14] present SP protocol which addresses the late-commit cheat and presents a performance improvement on the existing protocols (lock-step).

Goodman [12] proposes IRS hybrid C/S – P2P design; it operates by routing request messages through a centralized server and relaying them to proxy clients, a secure method by which it is certain that the requesting and proxy clients received the same message. The proxy clients perform calculations for others, relieving the server of the calculation burden. The code of the IRS approach relies on identifying malicious clients. This can be done with a certain probability and can still lead to cheat exploits.

Pellegrino and Dovrolis [20] propose a change from Client-Server architecture to Peer-to-Peer with Central Arbiter architecture (PP-CA) that contains server bandwidth requirements when increasing number of players, effectively solving the biggest scalability problem. The system still retains the SPOF in the form of the centralized arbiter. The paper focuses entirely on the elimination of the bandwidth problem and does not deal with any cheats; actually, it introduces a new form of cheating (e.g., blind opponent – BO, discussed in Section 3).

Webb et al. [28] compare all algorithms and show that all the previous distributed protocols and schemes are vulnerable to several cheats. Their proposed

scheme (RACS), which extends PP-CA [20], solves most of the problems but still has the SPOF in the form of the Identity server and Referee: a process running on a trusted host that has authority over the game state.

Most of the presented protocols are SPOF-free as they address only the P2P communication protocol but are vulnerable to cheats, as can be seen on Table 1. The RACS and IRS address most of the cheats but reintroduce the SPOF. Our scheme eliminates the SPOF problem and still retains all the properties described in RACS [28] and at least partially deals with the Collusion cheat that, at least in our opinion, cannot be eliminated by means of protocols and technology.

3 CHEAT TAXONOMY

In this article, we use the definition of Yan and Randel [30] for online game cheating: "Any behaviour that a player uses to gain an advantage over his peer players or achieve a target in an online game is cheating if, according to the game rules or at the discretion of the game operator (i.e., the game service provider, who is not necessarily the developer of the game), the advantage or the target is one that he is not supposed to have achieved." Cheaters try to gain unfair advantage over other players. This can totally destroy the in-game economics of an online game or simply ruin the gaming experience. Grievers, as the name implies, are players with the sole intention of hurting other players' experience as much as possible. When this behaviour adheres to game rules, it is technically not a form of cheating and is out of scope of this paper. Both groups exploit the same set of cheats.

The taxonomy presented in Table 1 and accompanying text and later used in this paper follows the taxonomy presented in Web et al. [28] and Yahyavi et al. [29], we added 3 additional entities, 2 were not addressed by the previous research (Robustness and User data privacy), the last one (Lack of Devices Situation) is a consequence of our approach and not applicable to other architectures. It is addressed later in this section. Multiple authors addressed the issue of systematic classification of cheating in online games, such as Yan and Randel [30] who present a taxonomy of 15 types of cheats and just by comparing the number of entries we could assume that the later introduces additional forms. We argue that the set of cheats presented in our paper fully covers the whole set presented in Yan and Randel [30] with the addition of two new entries that are discussed separately. The translations are presented in Table 1 in the second column. The "Cheating by compromising passwords" can be classified as "Social engineering" class and as such omitted. We argue that these two entries cannot be successfully addressed by the game architecture, they must be addressed mostly by informing the players.

1. *Bug* – bugs in games can lead to potential misuse by the players. No scheme directly addresses this problem, it is assumed that the bugs will be fixed by software developers.
2. *IE, IC* – the goal of IE (Information Exposure) is to obtain secret information to which the cheater is not entitled, thus gaining an unfair advantage in selecting

	Yan	Problem/Cheat	RACS	IRS	C/S	AS	NEO	Damage
1	L	Bug	√	√	√	√	√	√
2	A, F, H	IE, IC	√	√	√	×	×	√
3		Bots	×	×	×	×	×	×
4	A, C, F	Supp. update, TS, FD	√	√	√	√	√	√
5	A, F, H	Replay, Spoofing	√	√	√	×	√	√
6	A, D, F, H	Undo	n/a	n/a	n/a	√	×	n/a
7	A, C, F, H	BO	√	√	n/a	n/a	n/a	√
8	G	DDoS	√	√	√	√	√	√
9	B	Collusion	×	×	×	×	×	$\surd^{1}$
10	M,	Robustness	×	×	×	×	×	√
11	J	User data privacy	√	√	√	×	×	√
12		Lack of Devices Situation	n/a	n/a	n/a	n/a	n/a	√
13	E	Exploiting AI	n/a	n/a	n/a	n/a	n/a	n/a
14	I, O	Social Engineering	n/a	n/a	n/a	n/a	n/a	n/a^{*}

Table 1. Chart presenting all identified cheats and schemes for detection and removal. The n/a is given to a scheme that does not have to deal with the observed cheat for implicit reasons (mostly architectural).

the optimal action. The IC (Invalid Command) cheat occurs when an application or data files are modified to issue commands or command parameters that originally could not be generated.

3. *Bots* – programs that act as players can be introduced to the game. The programs can exercise number of cheats including Collusion.
4. *Supp. update, TS, FD* – A cheater can suppress sending the state update, send the updates at a slower rate (FD) to gain advantage or incorrectly timestamp messages to gain advantage.
5. *Replay, Spoofing* – a player can obtain advantage by replicating messages by means of local software instead of using the tools provided by the game (example: sending impossibly fast series of missiles).
6. *Undo* – a player succeeds to undo a previously sent message after already receiving the opponents message and realizing that the original message was not optimal.
7. *BO* – in distributed schemes that use a Central Arbiter (CA), such as PP-CA [20], cheater may purposely withhold updates to his peers (but not to the CA), effectively covering own actions.
8. *DDoS* – a (cheating) player may use DDoS [17] attack to temporally disable the opponent to send messages and thus get advantage.
9. *Collusion* – unfavorable situation may occur whereby certain clients cooperate with one another in order to gain unfair advantage over others. Collusion via the use of external communication is difficult to eliminate due to the use of non-monitored means of communication [12].

10. *Robustness* – The robustness metric shows how much is the system or scheme or protocol fault tolerant (how much it is tolerant to node failure, at the worst to server node failure). The basic Client-Server architecture is the least robust and totally decentralized system is the most robust.
11. *User Data Privacy* – user profiles with scoring and possibly in-game funds and purchases are stored for future use. Client Server architectures use the server as a means for reliable storage, distributed systems have to deal with security risks as the data is spread on the network or stored locally at clients an thus easily available for tempering.
12. *Lack of Devices Situation* – all schemes that rely on an outer referee (an external entity that is not part of the game) rely on the availability of the referee. In decentralised architectures where refereeing is done by other players, the scheme relies on availability of adequate number of players.
13. *Exploiting AI* – Exploiting artificial intelligence (AI) cannot be handled within the protocol or the architecture. The idea that a player can use an AI to improve its decision making in a game is not possible to detect on the protocol level.
14. *Social Engineering** – social engineering is a very broad term. Generally, it involves using social information about a player to trick the person into revealing sensitive information pertaining to a game, i.e. passwords. Our protocol uses ECDSA public cryptography, and does not require passwords. The authentication is not necessary as messages exchanged between players are all signed and verified.

4 DECENTRALIZED ARCHITECTURE FOR THE EDGE

Previously proposed P2P architectures rely on some level of centralization. We propose a completely decentralized architecture for edge devices that would inherently circumvent the single point of failure (SPOF). In contrast to client-server (C/S) architectures, where the server is authoritative, P2P networks are arguably more exposed to cheats and vulnerabilities. To address the issues Web et al. propose RACS [28], a referee node that takes the authoritative role in case of conflicting or inconsistent states between players.

However, RACS does not address issues of node selection. In completely decentralized networks, deriving secure randomness is an open question. From a security point of view, a decentralized random generator must not be known in advance to avoid attacks and vulnerabilities based on information exposure (IE). At the time of writing, most networks rely on oracle networks secured with game theoretical incentive schemes [21, 10]. However, the security models for such systems require strong incentives, which are mostly based on staking mechanisms that introduce penalties for bad actors and rewards for good actors [13]. A recent paper proposed a mathematical construction for verifiable random functions (VDF) [4], an extension of time lock puzzles [22] that produce verifiable proofs of computation. More specifically,

VDFs are similar to time lock puzzles but require a trusted setup where the verifier prepares each puzzle using its private key. Additionally, a difficulty parameter can be adjusted to increase the amount of sequential work, thereby increasing the delay. The proof can be used as an entropy pool for a seeded random to derive randomness within a decentralized system while preventing attacks based in IE. We solve the requirement for a trusted set up (private key of the VDF) by using a blockchain structure. Blocks have a configurable block time parameter, which is used to adjust the difficulty parameter of the VDF, to target the block time. The consensus algorithm is a novel lottery draw scheme, where nodes draw lottery tickets in order to be voted as block producers.

Suppose the current block is H at height (canonical id) h. The block hash of block H is used to compute the VDF and obtaining proof H_p. Each node $n \in N$ then draws its own lottery ticket H_t, which is defined as the distance between the node's public key, and H_p. Since all nodes share the same H_p, and all nodes (asymptotically) computed H_p at the "same" time, the lottery draw is not predictable. A node is elected to be part of the validator set if H_t is within the v closest tickets, where v is a configurable parameter usually set to $\frac{P}{PPI}$, where P is the total number of players, and PPI is the number of players per instance. Nodes that belong to the validator set are considered referees, and block producers for block $H + 1$. The structure of the block is shown in Figure 1.

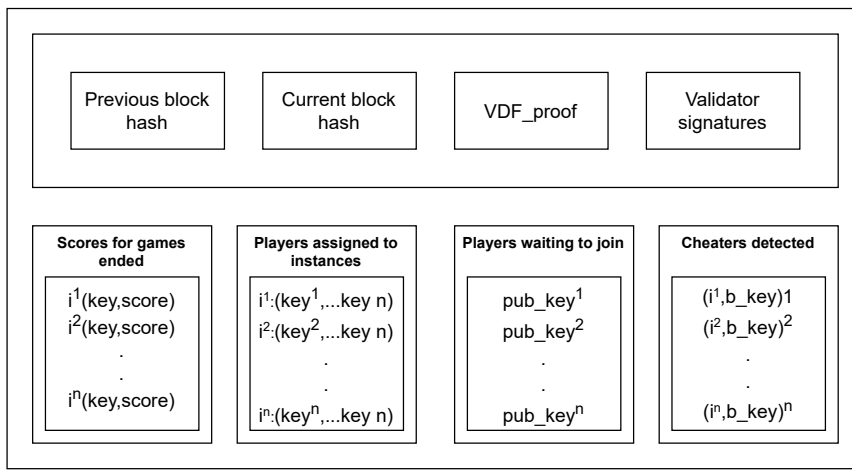


Figure 1. The Header of a block contains the previous block hash, current block hash, VDF proof, and signatures of all validators that signed the block. The body of the block contains a list of game instances that completed (combination of players public addresses), and their scores, a list of players waiting to join the next round (block), and a list of instances with assigned players that were in the waiting queue of the previous block.

Players are uniquely identified by their public key. Each player generates a public-private key pair before connecting to the P2P network. Upon joining the network,

players synchronize the last block to find the validator set. Querying the Distributed Hash Table – DHT [25], a node connects to the one or more validators to broadcast their intent to join a game. Once consensus is reached amongst the validators, a new block H will be forged that will include the player's public key in the players awaiting list. Players will receive block H, learn about their inclusion to the awaiting list, and wait for block $H + 1$. Note that the target block time is configurable with the VDF difficulty and should be set by the game operator. Upon receiving block $H + 1$, the players' public key will be assigned to an instance. Each instance has a unique ID, which is obtained by concatenating the public keys of players assigned to the instance. Each player parses the instance ID to obtain the public keys of opponents and connects to them by querying the DHT (with the public key) for their address. The last address of the instance ID is the assigned validator that will assume the role of the referee. Once the instance is resolved (game finished), the referee (also a validator) will inform all other validators and propose the inclusion of the decision/score in the next block. Validators will vote on the proposed block by signing it with their public key. Clients can verify their signature using their public key. In case a referee of an instance detected a cheater, a proof can be sent to the set of validators for confirmation. The details of how this is achieved are explained in detail in Section 5.

A candidate block $H+1$ is then transmitted (using gossip protocol) [15] through the P2P network. Each client accepts the block if and only if the block references the local block hash at height h, the provided proof H_p is valid, and the candidate block contains signatures of all validators whose public keys can be computed by each node using H_p. The nodes that are part of the validator set execute a matchmaking algorithm that must be deterministic (but can rely on randomness derived from H_p) and can use the previous block as input (list of players wanting to join). The matchmaking algorithm assigns player to game instances, and referees from the validator are set to act as authoritative nodes. The deterministic nature of the matchmaking algorithm is used to reach consensus amongst the validator nodes. The consensus is reached as all honest nodes will construct the same candidate block and will sign all candidate blocks equal to theirs. The result will be a candidate block signed by the majority of validators.

The construction assumes validators, referees, and players to be players. However, a set T of trusted nodes is required and assumed to be maintained by the game maintainers. These nodes are called full-nodes and are necessary to guarantee liveliness of the system even in extreme cases where there are no players in the network. Full-nodes are also responsible for permanently storing the blockchain and maintaining a DHT-based structure other nodes can query to discover other peers. Players are assumed to be lite clients that do not need to store the entire blockchain history in order to participate in the consensus [27]. Additionally, referees are assumed to be players as well. The matchmaking algorithm should avoid assigning players to be referees to their own game instance.

Each game can have one referee, which arguably decreases the robustness. All decisions about conflicts proposed by a referee must be presented to the validator

set in order to reach consensus and gather enough signatures to make the block valid. However, the referee can unexpectedly disconnect or even worse, be attacked by a player during the game. To circumvent this issue, any number of validators can be assigned as backups in case the referee is unresponsive.

Referees in DAMAGE are running the same protocol as RACS. However, in case a referee detects a cheating player, the proof (usually a set of states that allow validators to recreate/simulate the game) must be presented to the set of validators (also RACS referees). Consensus is reached if and only if $\frac{2}{3}$ validators agree [5]. Decisions about the proposed cheat detected is done by voting for the block. Each block contains a list of (*public key*, *instance key*) pairs and the type of cheat detected. Assuming the validator is honest, and the referee proposing the detected cheat is as well, both validators will reach the same conclusion and thus sign the block. In any other case, the block will only be signed by the malicious validator. Proposals that do not reach consensus are considered invalid blocks and will be rejected by the client protocol. A subset of nodes (without the majority vote) running modified clients may choose to accept the invalid block, thereby forking the chain [1]. In such cases, the next block would either resolve the fork if it is accidental or disconnect (network level) the subset of nodes with the modified protocol due to an invalid VDF proof on the forked chain.

5 SECURITY MODEL

All communication between peers provides the same level of security to that of C/S architectures by using public key (ECDSA) cryptography. DAMAGE provides a secure and completely decentralized protocol for selecting referees, and match players to game instances. However, the player and referee protocols are based on RACS [28] and therefore DAMAGE inherits cheat detection properties of RACS, and extends them with efficient and secure peer selection, peer synchronization, robustness, and some aspects of collusion.

5.1 Referee Selection

We address the issue of Referee selection by using VDF to derive randomness with which a lottery-based consensus is reached. The sequential nature of VDFs prevents IE attacks where a player would compute the VDF and, using the proof, obtain information about which nodes are part of the validator set and which node is assigned as the main referee to each game. Game operators should set the difficulty parameter according to their desired performance/security ratio. A more difficult VDF will result in players waiting to be matched to an instance longer (i.e., a few seconds), while a lower difficulty will potentially allow malicious players to discover the nodes that will be within the set of validators before others. We argue that knowing the set of validators and, consequently, the referee node for a game does not give the player a competitive advantage. This is further explained in the case of collusion.

5.2 Referee Trust

In RACS scheme referee nodes are assumed trustworthy (the authors acknowledge this to be an open issue). We solve this issue with the validator set. Even if a RACS-based referee is compromised, any player can dispute the referee and seek a decision by consensus within the set of validators. The player would then have to compromise $\frac{2}{3}$ nodes in the validator set, which is not known in advance and changed every block.

Instead of using only one referee per game, we propose to establish a Referee set (validator set of referees) that. Additionally, the cardinality of the validator set is a configurable parameter analogous to the trust level required by the game (higher trust requires bigger cardinality).

5.3 Synchronization

On the data layer, nodes synchronize through the blockchain. Blocks store the current state of the system on lite clients and the entire history on full-nodes maintained by the game operators. Blocks are gossiped across the network efficiently by maintaining a DHT that maps nodes (public keys) to their network addresses. Additionally, referees in the validator set must synchronize and reach consensus about the detected cheats and results of the games played. Due to the deterministic nature of the cheat detection algorithms, honest nodes will reach the same decision as the referee that reported the cheat. Consensus is reached if the majority of the validators sign the proposed block (which includes the decision about reported cheats).

5.4 Robustness

DAMAGE uses redundancy to increase fault tolerance. There are two main types of faults that can occur. A peer can fail (disconnect or violate protocol) before, after or during playing the game, and a peer acting as a referee (and also as a player in a different game) fails at the same time.

Player faults after entering matchmaking: A peer that faults after it announced inclusion to the validator set will cause the validators to match its public address to an instance. Other peers attempting to connect will fail and/or result in protocol violation. It is up to the client protocol to decide if the game instance can continue to run without the faulty peer or not. In case the instance must be destroyed, this can be trivially solved by extending the referee's protocol to label this as a "cheat". The referee will announce the instance destruction to the validator set.

Player faults during the game: If possible, the game instance should keep running. If not possible, the referee should notify the validator set about the destruction.

Player faults after the game: No effect on the system.

Faulty validators (referees) are arguably a bigger security issue. Even without the ability for players to know which referee will be assigned to their instance there is still a possibility of DDoS attacks on referees during the game. To combat this issue, validators form a randomly shuffled priority queue using the VDF proof. The priority queue is a backup queue of referees that will take over an instance in case the referee assigned faults. The fault tolerance can be increased on demand by increasing the size of the validator set. However, detecting a faulty referee must be done by peers playing in the instance. If messages from peer to referee are either latent or connection is dropped, client protocol will take the following steps:

1. Set up a seeded random with the latest block (local) of the VDF proof.
2. Compute the lottery draw results to find the public keys of the validator set.
3. Shuffle the validator set list with the same seed.
4. Contact the next validator (backup referee for its game).

5.5 User Profile Management

Previous research relied on a central authority for authenticating users and managing their profiles such as avatars, variables, and metadata. Our architecture can be extended with a completely decentralized storage and authentication service, a centralized authoritative server as well as inter-operability between both. A blockchain-based authentication service can be built in by extending the block structure [18]. Additionally, blocks can be used for persistent immutable storage. However, storing data in blocks raises scalability issues [31, 24] as the blockchain becomes hard to maintain even for full nodes. Hybrid approaches have been proposed where data is stored centrally whereas the signatures are stored on-chain [31]. This creates a tamper-proof system where data can be verified and trusted as any attempt to tamper with the data would invalidate the signature (hash) [24].

5.6 Lack of Devices Situation

The refereeing process relies on a set of validators that are randomly chosen for each block time-cycle. The randomness of selection ensures that a player cannot know who is refereeing the next game. Player's devices are used to act as validators. The pool of validators cannot be constructed if there are not enough players. It is developers' or game operator's task to supply enough (a fixed number that does not grow with player-base) resident secure services (servers) that act as starting validators. These actors also maintain the blockchain (full nodes).

5.7 Collusion

Assuming the game runs multiple instances, we argue collusion between players is not possible. We assume colluding players know and, hence, trust each other.

Figure 5 shows a graph of a simulation of 200 players as nodes. The edges represent the number of games (weight) a player was assigned another player as the referee for the instance the player was matched to. Simulating 1 000 blocks, the average degree was 200, and the graph density was 1. We observe that the assignment of a referee and opponents derived using the VDF proof are thereby random. Hence, players cannot know in advance which instance the set of validators will assign them to nor the referee that will observe the game. Section 6.2 and Figure 5 present an empirical evaluation of the "fairness" of the selection method based on VDF. Suppose the colluding players are able to compute the VDF proof faster then other players, and, hence, learn about the game instance assignment in advance. However, since the seed for the next VDF is the block hash, the colluding players can see at most one block time into the future. Every player must announce the desire to be matched to a game in the current block (players awaiting list). Matching awaiting players will be executed in the next block. Despite the ability to see one block in the future, colluding players seek assignment to the same instance since they must announce their willingness to play before they learn about the instance assignment even in the worst case scenario.

6 EVALUATION

The paper presents a scheme to eliminate all known cheats in a fully distributed game setting. The scheme eliminates the SPOF problem in previously presented hybrid P2P – Referee settings for solving game cheats. We base our solution on already presented solutions, mostly RACS [28]. We present simulation results leading to the following conclusions:

- The VDF based selection of referees and players is fair.
- The block propagation scales well.
- Block propagation times are acceptable for fast match making, and conflict resolution.
- Dynamic block size does not impact performance of the system.
- Players do not need to maintain a large number of outgoing connections.
- Latency and bandwidth do not substantially slow down information propagation through the network.

The scalability of the solution is addressed in two ways. The first is the scalability of the consensus mechanism and the ability to propagate state and state transition depending on the block size. In this test we show that the solution can scale to hundreds of thousands of nodes and achieve consensus.

The second scalability test is performed by introducing variance in latency and bandwidth to mimic the instability of home internet connections under standard TCP/IP parameters.

Since every player is also a node, and potentially a referee, we argue that scalability in terms of number of players can be derived by the aforementioned tests. Additionally, due to the nature of the P2P architecture, once players are matched into a game instance, the entire communication is done solely between them, and the referee of that instance, which is completely independent of the rest of the network, and hence does not impact the scalability.

In order to evaluate the solution a simulation environment was developed. We simulate a P2P network where each peer (player) has the following constraints:

- Local bandwidth constraints. Bandwidth constraints are assigned to peers joining the network based on the distribution obtained from the European report on network bandwidth [9].
- Maximal number of outgoing connections (out edge degree) is assigned to nodes (MAE), we ran tests with different values of this parameter, they are color-coded in Figure 3.
- Each node's connection is single-directional taking into account upload and download bandwidth constraints of sender and receiver.
- Each new connection is assigned a round trip time (RTT) to represent variance in latency. RTT values are assigned randomly fitting a Gaussian distribution on an interval $[30, 250]$ ms, the values were taken from the European report on network bandwidth [9].
- Actual throughput of each connection is estimated using the Mathis metric [16] with following parameters that were taken from real-life situations: maximum segment size (MSS) of 1460 bytes (most used in today's communications as shown in papers such as [23]), the connection's RTT, and a TCP packet loss probability of $p = 1.0 * 10^{-5}$ [19].

Nodes (players) join the network by connecting to one of the trusted nodes. Trusted nodes are those operated by the game maintainer and serve only as the entry point for new peers to discover other peers or if needed to persistent storage for player accounts. A node proceeds to run the peer discovery protocol building the DHT. When new nodes are discovered, the peer attempts to make new connections until the MAE limit is reached and the node is considered to be well connected. Examples of different architectures (presented by connected directed graph) for 20 nodes are presented in Figure 2.

Once a new block is forged the origin nodes propagates the block using a basic flooding algorithm simulating the bandwidth, and TCP constraints. In each simulation, multiple directed graphs are constructed following the above protocol. Simulations were carried out with different number of nodes to observe the scalability of the solution. We measure propagation time as the total time it takes for all nodes to receive a newly forged block.

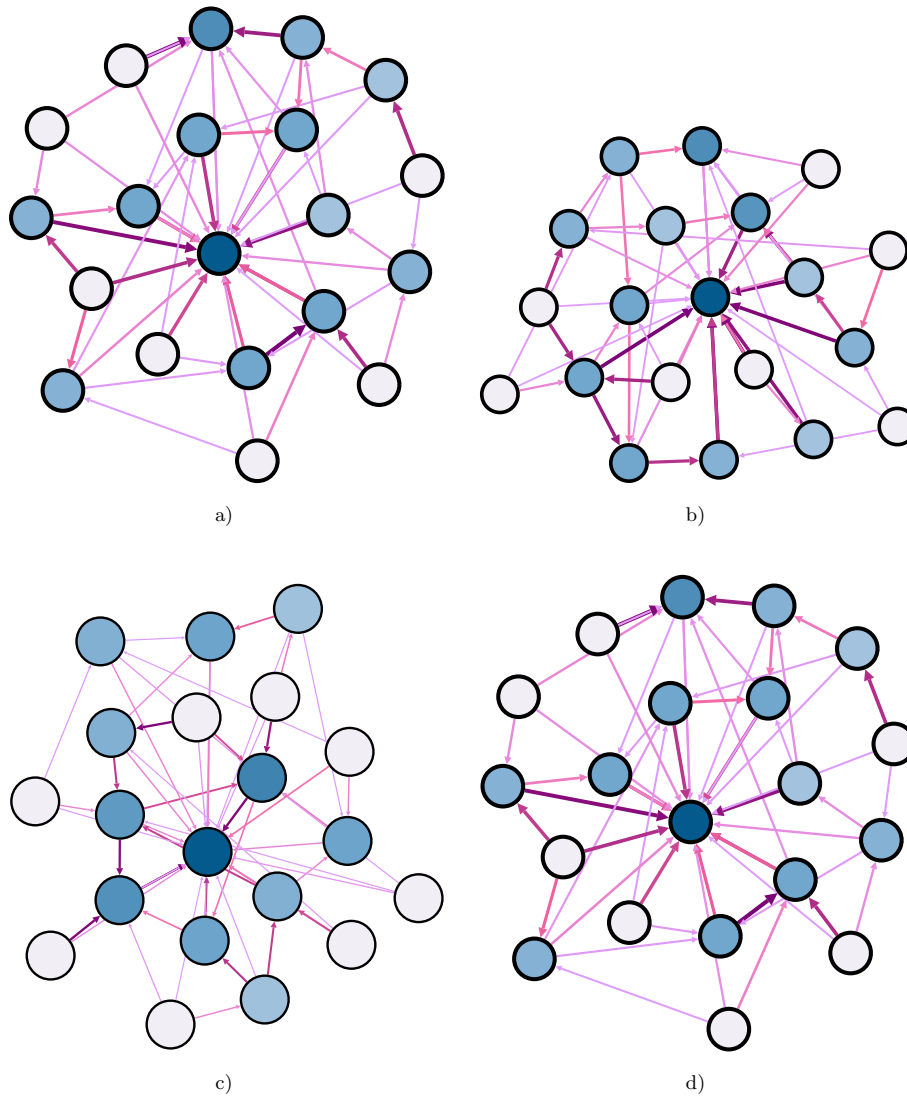


Figure 2. Examples of different architectures obtained by simulation. Number of nodes in all examples is $n = 20$.

6.1 Block Propagation Times

Blocks hold the state of the match making and games being played. Lowering the block time (VDF difficulty) would result into a more responsive experience. However, lower block times reduce security, and can cause network congestion. To avoid possible client synchronization issues the network must be able to reliably propagate blocks before new ones are forged. Additionally, the propagation times vary depending on the network topology, block sizes, and average node degree. We evaluate the scalability of propagating blocks in order to estimate viable block times, and show the scalability of the solution. From Figure 3 we observe that propagation times scale logarithmic as we increase the number of clients. Additionally, increasing the number of outgoing connection a node maintains reduces the average propagation times as it reduces the risks of unfavorable graph typologies.

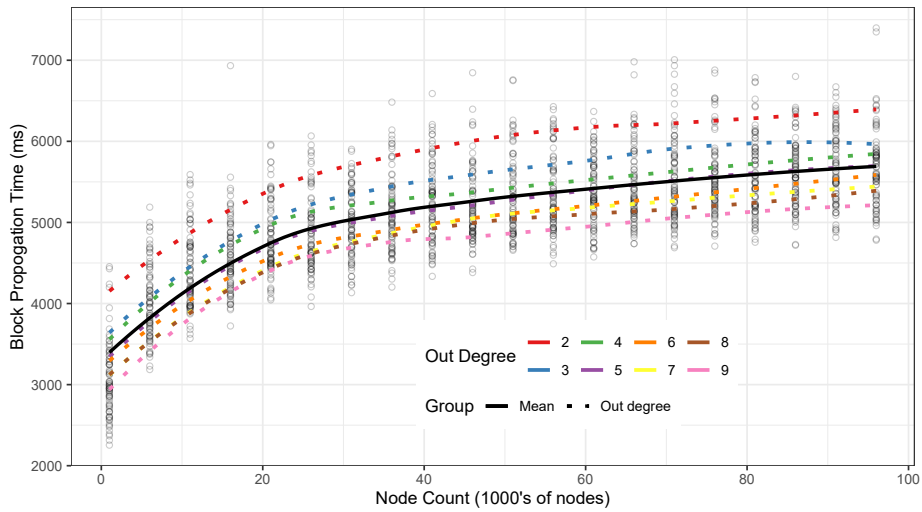


Figure 3. Simulation of block propagation on different graph configurations. Configurations are obtained by increasing the node count (number of players), and out degree using a block size of 1 MB.

Block size scales lineally with the number of clients. Every block has a constant size for the header, which is 64 bytes for previous and current block hash, 100 bytes for the VDF proof, and at least one validator signature of 64 bytes. As more players join the network, more games need to matched, assigned to instances, and scores saved. Figure 4 shows how the network scales with different block sizes. We observe that latency and bandwidth speeds of some nodes can cause considerable propagation slowdowns indicated by some outliers. However, this can be mitigated by having nodes maintain a dynamic number of outgoing connections increasing the

limit as blocks become larger (more players), and lowering the limit as blocks are smaller.

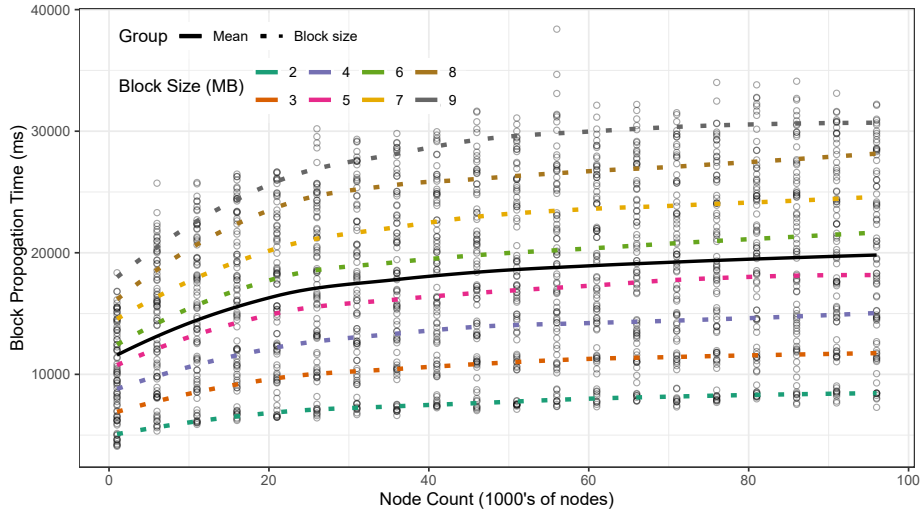


Figure 4. Simulation of block propagation on different graph configurations. Configurations are obtained by increasing the block size. Nodes were limited to 3 outgoing connections.

6.2 The VDF Based Selection of Referees and Players is Fair

Figure 5 shows a graph of a simulation of 200 players as nodes. The edges represent the number of games (weight) a player was assigned another player as the referee for the instance the player was matched to. Simulating 1 000 blocks, the average degree was 200, and the graph density was 1. We observe that the assignment of a referee and opponents derived using the VDF proof are thereby random. Hence, players cannot know in advance which instance the set of validators will assign them to nor the referee that will observe the game.

6.3 The VDF Method Scales Well

The setting presented in Section 6.2 and Figure 5 shows that the VDF based selection method is fair, the graph shows 200 players and 1 000 blocks, as this was the maximum feasible number combination that was still manageable to visualize. The setting was further evaluated with different parameters for the number of players, number of players per game and number of blocks. The number n of players per game: means a game where n players participate, Number of blocks: how much time the matchmaking process was observed. We evaluated the setting with 200, 1 000

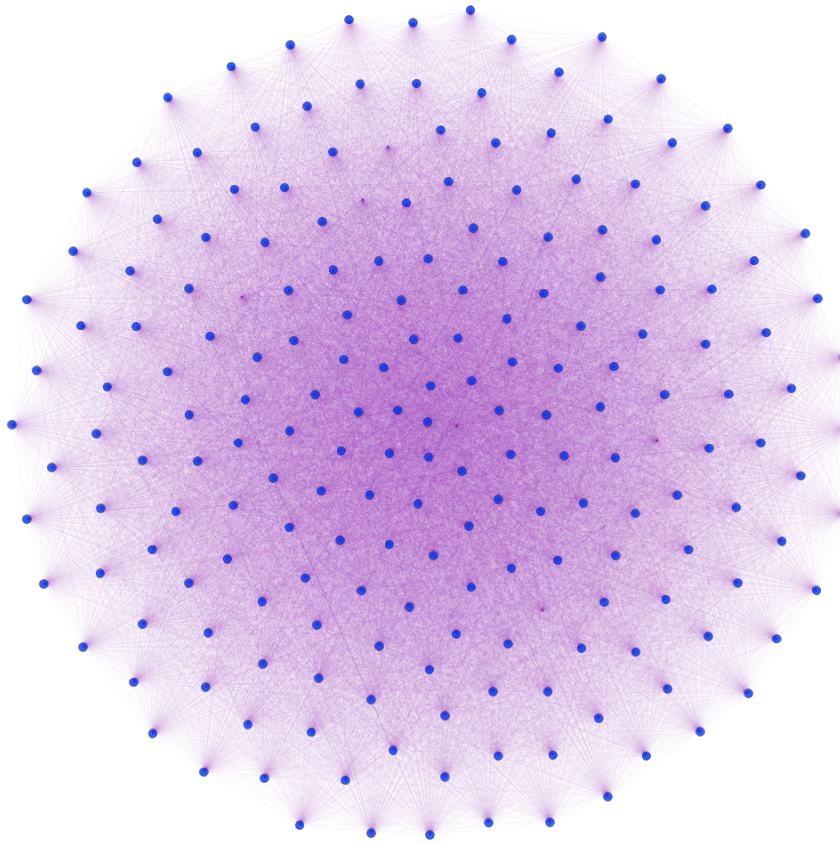


Figure 5. Simulation of 200 players in a 2-player game ($PPI = 2$) that played a total of 1 000 games. Nodes are players and edges represent instances where the destination node was referee for the instance the player was matched to.

and 10 000 players, 1 000, 2 000 and 10 000 blocks, we also changed the number of players per game. All results were consistent with the first test, the degree of all players was near the number of players and the density of the graph was near 1.

7 CONCLUSION AND FURTHER WORK

The paper proposes a blockchain-based, completely decentralized architecture for edge devices with no single point of failure that successfully addresses cheat problems. The presented solution is based on two hybrid approaches to P2P network games anti-cheat schemes that were based of server acting as referees. We propose

a completely decentralised approach while still retaining the same cheat resistance, actually in the case of Collusion we were able to partially address the issue. The proposed solution has not been fully implemented, we implemented the newly proposed building stones and executed empirical testing on a pilot setting. As the solution addresses the cheating problem in all aspects, a fully functional implementation is possible. DAMAGE is applicable to most game types. However, it is most suitable for turn based games, where potential latency does not impact user experience dramatically. Additionally, it reduces the complexity of the referee implementation due to the simple ordering of actions in the discrete time. Our results show, that the architecture scales automatically with the number of players thereby drastically reducing operation costs of running a multiplayer game. Every player added to the system also becomes a node, sharing its resources and contributing to verification as a potential referee.

Acknowledgment

The authors gratefully acknowledge the European Commission for funding the InnoRenew CoE project (Grant Agreement No. 739574) under the Horizon2020 Widespread-Teaming program and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Union of the European Regional Development Fund).

REFERENCES

[1] Baliga, A.: Understanding Blockchain Consensus Models. Technical Report, Persistent Systems Ltd., 2017, pp. 1–14.

[2] Baughman, N. E.—Levine, B. N.: Cheat-Proof Playout for Centralized and Distributed Online Games. Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (INFOCOM 2001), Vol. 1, 2001, pp. 104–113, doi: 10.1109/INFCOM.2001.916692.

[3] Baughman, N. E.—Liberatore, M.—Levine, B. N.: Cheat-Proof Playout for Centralized and Peer-to-Peer Gaming. IEEE/ACM Transactions on Networking (ToN), Vol. 15, 2007, No. 1, pp. 1–13, doi: 10.1109/TNET.2006.886289.

[4] Boneh, D.—Bonneau, J.—Bünz, B.—Fisch, B.: Verifiable Delay Functions. In: Shacham, H., Boldyreva, A. (Eds.): Advances in Cryptology – CRYPTO 2018. Springer, Cham, Lecture Notes in Computer Science, Vol. 10991, 2018, pp. 757–788, doi: 10.1007/978-3-319-96884-1_25.

[5] Castro, M.—Liskov, B.: Practical Byzantine Fault Tolerance. Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, 1999, pp. 173–186.

[6] Corman, A. B.—Douglas, S.—Schachte, P.—Teague, V.: A Secure Event Agreement (SEA) Protocol for Peer-to-Peer Games. First International Confer-

ence on Availability, Reliability and Security (ARES '06), 2006, 8 pp., doi: 10.1109/ARES.2006.15.

[7] Dobrilova, T.: How Much is the Gaming Industry Worth? Techjury, 2019.

[8] Dooley, K.: Designing Large Scale LANs: Help for Network Designers. O'Reilly Media, Inc., 2001, 404 pp.

[9] European Court of Auditors: Broadband in the EU Member States: Despite Progress, Not All the Europe 2020 Targets Will Be Met. Technical Report, European Court of Auditors, 2018.

[10] Gatteschi, V.—Lamberti, F.—Demartini, C.—Pranteda, C.—Santamaría, V.: Blockchain and Smart Contracts for Insurance: Is the Technology Mature Enough? Future Internet, Vol. 10, 2018, No. 2, Art. No. 20, 16 pp., doi: 10.3390/fi10020020.

[11] GauthierDickey, C.—Zappala, D.—Lo, V.—Marr, J.: Low Latency and Cheat-Proof Event Ordering for Peer-to-Peer Games. Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '04), 2004, pp. 134–139, doi: 10.1145/1005847.1005877.

[12] Goodman, J.: A Hybrid Design for Cheat Detection in Massively Multiplayer Online Games. M.Sc. Thesis, McGill University, Montréal, 2008.

[13] Heiss, J.—Eberhardt, J.—Tai, S.: From Oracles to Trustworthy Data On-Chaining Systems. Proceedings of IEEE International Conference on Blockchain (Blockchain 2019), 2019, pp. 496–503, doi: 10.1109/Blockchain.2019.00075.

[14] Jamin, S.—Cronin, E.—Filstrup, B.: Cheat-Proofing Dead Reckoned Multiplayer Games. Proceedings of 2nd International Conference on Application and Development of Computer Games, Hong Kong, 2003, pp. 1–7.

[15] Kwiatkowska, M.—Norman, G.—Parker, D.: Analysis of a Gossip Protocol in PRISM. ACM SIGMETRICS Performance Evaluation Review, Vol. 36, 2008, No. 3, pp. 17–22, doi: 10.1145/1481506.1481511.

[16] Mathis, M.—Semke, J.—Mahdavi, J.—Ott, T.: The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. ACM SIGCOMM Computer Communication Review, Vol. 27, 1997, No. 3, pp. 67–82, doi: 10.1145/263932.264023.

[17] Mirkovic, J.—Reiher, P.: A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. ACM SIGCOMM Computer Communication Review, Vol. 34, 2004, No. 2, pp. 39–53, doi: 10.1145/997150.997156.

[18] Moinet, A.—Darties, B.—Baril, J.-L.: Blockchain Based Trust and Authentication for Decentralized Sensor Networks. 2017, pp. 1–6, arXiv: 1706.01730, doi: 10.1109/wimob.2017.8115791.

[19] Moltchanov, D.: A Study of TCP Performance in Wireless Environment Using Fixed-Point Approximation. Computer Networks, Vol. 56, 2012, No. 4, pp. 1263–1285, doi: 10.1016/j.comnet.2011.11.012.

[20] Pellegrino, J. D.—Dovrolis, C.: Bandwidth Requirement and State Consistency in Three Multiplayer Game Architectures. Proceedings of the 2nd Workshop on Network and System Support for Games, 2003, pp. 52–59, doi: 10.1145/963900.963905.

[21] Peterson, J.—Krug, J.—Zoltu, M.—Williams, A. K.—Alexander, S.: Augur: a Decentralized Oracle and Prediction Market Platform. 2015, pp. 1–16, arXiv: 1501.01042, doi: 10.13140/2.1.1431.4563.

[22] Rivest, R. L.—Shamir, A.—Wagner, D. A.: Time-Lock Puzzles and Timed-Release Crypto. Technical Report, Massachusetts Institute of Technology, 1996, pp. 1–9.

[23] Deering, S. R. H.: Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, RFC Editor, 1998.

[24] Shafagh, H.—Burkhalter, L.—Hithnawi, A.—Duquennoy, S.: Towards Blockchain-Based Auditable Storage and Sharing of IoT Data. Proceedings of the 2017 on Cloud Computing Security Workshop (CCSW '17), Dallas, Texas, USA, 2017, pp. 45–50, doi: 10.1145/3140649.3140656.

[25] Stoica, I.—Morris, R.—Karger, D.—Kaashoek, M. F.—Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM Computer Communication Review, Vol. 31, 2001, No. 4, pp. 149–160, doi: 10.1145/964723.383071.

[26] Superdata: Market Brief – 2018 Digital Games and Interactive Entertainment Industry Year in Review, 2019.

[27] Vorick, D.—Champine, L.: Sia: Simple Decentralized Storage. Nebulous, 2014, pp. 1–8.

[28] Webb, S.—Soh, S.—Lau, W.: RACS: A Referee Anti-Cheat Scheme for P2P Gaming. Proceedings of the 17th International Workshop on Network and Operating Systems Support for Digital Audio and Video, 2007, pp. 34–42, doi: 10.1145/1542245.1542251.

[29] Yahyavi, A.—Kemme, B.: Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey. ACM Computing Surveys (CSUR), Vol. 46, 2013, No. 1, Art. No. 9, 51 pp., doi: 10.1145/2522968.2522977.

[30] Yan, J.—Randell, B.: A Systematic Classification of Cheating in Online Games. Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '05), 2005, pp. 1–9, doi: 10.1145/1103599.1103606.

[31] Zyskind, G.—Nathan, O.—Pentland, A. S.: Decentralizing Privacy: Using Blockchain to Protect Personal Data. 2015 IEEE Security and Privacy Workshops, 2015, pp. 180–184, doi: 10.1109/SPW.2015.27.

Aleksandar Tošić is Teaching Assistant at the University of Primorska, and Research Assitant at InnoRenew CoE. His main research interests are distributed systems and distributed ledger technologies.

Jernej Vičič is Associate Professor at the University of Primorska, Primorska Institute for Natural Sciences and Technology in Koper, Slovenia. His main research interests are distributed systems and natural language processing.

Computing and Informatics, Vol. 40, 2021, 543–574, doi: 10.31577/cai_2021_3_543

REDUCER: ELIMINATION OF REPETITIVE CODES FOR ACCELERATED ITERATIVE COMPILATION

Hameeza Ahmed, Muhammad Ali Ismail

Department of Computer and Information Systems Engineering
NED University of Engineering and Technology
Karachi, Pakistan
e-mail: {hameeza, maismail}@neduet.edu.pk

Abstract. Low Level Virtual Machine (LLVM) is a widely adopted open source compiler providing numerous optimization opportunities. The discovery of the best optimization sequence in this large space is done via iterative compilation, which incurs substantial overheads, especially for big data applications operating on high volume and variety datasets. The large search space is mostly comprised of identical codes generated via different optimizations. However, no mechanism is implemented inside the LLVM compiler to suppress the redundant testings. In this regard, this paper proposes REDUCER for eliminating the identical code executions by performing Intermediate Representation (IR) level comparisons. REDUCER has been tested using the well-accepted MiCOMP technique in LLVM 3.8 and 9.0 compiler, with embedded (cBench) and big data workloads. In comparison to MiCOMP 19.5 k experiments, REDUCER lowers the experiment count up to 327, i.e. 98 %, and on average to 4 375, i.e. 77 %, for cBench (LLVM-3.8). Similarly, for LLVM-9.0 the reductions are up to 1 931, i.e. 90 %, and on average 5 863, i.e. 69.9 %. Due to the significant experiment reduction, for embedded workloads, the iterative compilation is up to 58.6× and on average 4.1× faster with REDUCER (LLVM-3.8) than MiCOMP, whereas, with REDUCER (LLVM-9.0) the compilation is up to 8.5× and on average 2.9× faster. Moreover, REDUCER is found to be scalable and efficient for big data workloads where the iterative compilation is reduced to few days, as code is compared one time only for a single application tested on multiple datasets.

Keywords: Iterative compilation, code redundancy, LLVM, IR, big data

1 INTRODUCTION

Low Level Virtual Machine (LLVM) [1] is an open source compiler infrastructure, which is widely adopted due to high *ease of use*, *flexibility*, *portability*, and *modularity* [1, 24, 23]. It features *source* and *target* independent Intermediate Representation (IR) code, which allows numerous optimizations to be easily applied. Among the millions of available optimizations inside LLVM, the suitable optimizations for the given *application*, *environment* (architecture, OS, compiler), and *dataset* combination can be found by repeated execution of the program with each optimization, commonly known as *iterative compilation*. The testing of this huge search space for varying *application*, *dataset*, and *environment* incurs significant time and resource overheads. These costs are especially exaggerated if the optimizations are tested for *big data* applications [7], involving high *volume* and *variety* datasets [12]. In the case of *volume*, each optimization sequence is executed with a large data size which is more time-consuming than routine sizes. Similarly, in the case of *variety*, each optimization is iteratively executed with multiple datasets, increasing the number of runs.

Several techniques [14, 15, 3, 10, 11] have been proposed for reducing the search space in LLVM. However, these techniques consider the identical codes as separate optimization, hence increasing the overall search space due to redundant testings. Although [20, 22, 19], emphasizes on reducing the search overhead by detecting identical codes. But, no such method has been practically adopted by LLVM compilation techniques, for detecting identical optimization sequences. Instead, iterative compilation is treated as a black box, and complex techniques are proposed for reducing the search space.

The repeated code execution makes the process of iterative compilation infeasible by unnecessarily increasing the number of experiments which ultimately leads to heavy resource and time wastage. The overheads are especially inflated for big data applications processing high volume and variety datasets [7, 12]. For instance, an application has total 19.5 k optimized codes, with only 500 unique codes. If the application is tested with large size data taking approx. 3 h, then $19.5\,\text{k} * 3 = 58\,500$ h approx. is required to find the best optimization sequence, having 19 k redundant experiments. Whereas, if the unique 500 codes are detected at the start, then only $500 * 3 = 1\,500$ h + detection time is needed to search the best optimization. Similarly, for testing the considered application with 5 varying datasets each taking 2 h, approx. $19.5\,\text{k} * 2 * 5 = 195\,000$ h is required with 19 k redundant tests. However, if the redundant tests are suppressed, then only $500 * 2 * 5 = 5\,000$ h + detection time is enough.

For reducing the LLVM optimization space, this paper proposes **REDUCER**, which lowers the search space by detecting *identical* codes. It has been tested using well-accepted Mitigates the Compiler Phase-ordering (MiCOMP) [3] technique in Low Level Virtual Machine (LLVM) compiler [1]. LLVM makes the REDUCER portable enough to work on any host platform as the reductions are made on the basis of machine-independent code. REDUCER selects the executable candidates

after comparing the generated IR with the already existing IR codes. In case, only if the IR does not match with existing IRs, it is selected as an executable candidate. REDUCER testing has been performed using embedded, i.e. cBench benchmark suite (LLVM-3.8 & 9.0), and big data workloads (LLVM-9.0). MiCOMP [3] was tested in 2017 on LLVM-3.8[1], using *agglomerative* clustering, thus for LLVM-9.0 we have extended MiCOMP approach by finding the optimization clusters, but using k-means algorithm. It has been observed that our derived k-means based sub-sequences for LLVM-9.0 exploit greater speedup than MiCOMP's *agglomerative* based [3] sub-sequences for LLVM-3.8.

For both LLVM 3.8 and 9.0, REDUCER shows substantial reduction in experiment count of embedded workloads. Also, it is discovered that the increase in optimization sequence length, increases the redundancy fraction, which encourages the testing of longer sequence lengths. In this manner, the larger optimization space can be exploited, which has been uncovered till now. Moreover, Dynamic Programming (DP) has been applied to estimate the unique code sequences. DP is found to be less accurate than REDUCER, with increased experiment count. Despite a significant number of code comparisons, REDUCER is observed to be faster than MiCOMP, and this speed is expected to significantly increase for longer sequences and big data applications. As evident via experiments, REDUCER cuts down the iterative compilation of big data benchmarks to a few days in comparison to months and years taken by MiCOMP. This way, the possibilities of finding the best optimization sequence for *big data* applications are enlarged due to REDUCER. Hence, REDUCER is a simple yet effective solution to be adopted by any iterative compilation technique. Following are the main contributions of this paper:

1. To the best of our knowledge, the first work which practically accelerates the iterative compilation process of LLVM by reducing the search space via simple IR code comparisons.
2. Higher portability, i.e., REDUCER can work on any host platform due to machine-independent LLVM IR code.
3. Exploitation of greater speedup by extending MiCOMP for LLVM-9.0 using k-means clustering.
4. Facilitating the iterative compilation process for *big data* applications, i.e., repeated tests are suppressed by comparing code one time for a single application tested on multiple datasets.
5. Facilitating the exploitation of large search space via longer sequence length, because the analysis shows redundancy fraction is increased with sequence length.

Rest of the paper is organized as follows: Section 2 discusses background and motivation, REDUCER is presented in Section 3, the experimental setup is discussed

[1] LLVM-3.8 was the newer version in 2017, but at the time of experimentation of this paper, i.e. 2020, LLVM-9.0 is the newer version.

in Section 4, Section 5 analyzes the results. Section 6 discusses the related work, followed by a conclusion in Section 7.

2 BACKGROUND AND MOTIVATION

This section discusses various terminologies and concepts. Also, the motivation behind this work is presented.

2.1 Compiler

Compiler is a program that translates the high-level language code into architecture specific assembly and enables the optimizations for exploiting the hardware resources. This implies that despite the presence of powerful hardware design, the performance goals are not met due to a lack of competent software solutions. In the present era, the hardware resources (processors, caches, DRAMs, and hard disks) show reliance on the compiler for extracting the higher performance, energy efficiency, and reduced development time. The compilation life cycle proceeds by passing the source code through *front end*, *middle end* (optimizer), and *back end*. The front end emits Intermediate Representation (IR) code, which is passed through middle end to perform specific optimizations like *inlining*, *unrolling*, etc. In the end, the back end generates the machine code [25, 18, 8, 9, 2, 17].

2.2 LLVM

Low Level Virtual Machine (LLVM) is an open source compiler infrastructure, containing reusable and modular compiler technologies. It provides wide optimization opportunities due to library based optimizer's design. Besides, it allows flexibility as the optimizations passes can be ordered to be executed in a specific order. This way, the design enables the selection of individual optimization passes to execute. LLVM allows working with anyone optimizer separately, without considering other modules attached to it. Whereas, the traditional compilers are designed as tightly interconnected code, which is tougher to break into small parts for better understanding and use. The LLVM code is represented by Static Single Assignment (SSA)-based Intermediate Representation (IR), which provides *low level operations*, *type safety*, *portability*, *flexibility*, etc. The LLVM IR appears to be a universal IR, as all the phases of LLVM compilation use this IR [1, 24, 23].

2.3 Iterative Compilation

For an *application*, *dataset*, and *architecture*, the optimal set of optimizations is found via *iterative compilation*, where the best optimization combination is detected by running a program multiple times, each time with different optimizations combinations. This iterative testing involves billions of different optimizations. To avoid

this huge space exploration, standard optimizations, i.e., -O1, -O2, -O3, -Os, have been provided in commercial compilers, which on average bring good performance on a set of applications. However, there exist optimizations combinations that outperform the standard optimization levels for many programs by a considerable margin. Finding the best optimizations ordering can significantly improve the performance metrics like *execution time*, *energy*, *power consumption*, and *code size*.

Despite the great potential offered by iterative optimization, it is not widely used in compilers, because it requires numerous recompilations and training runs to detect the best optimization combination for a given program. This way, the costly overhead of recompilation and training runs can eradicate the benefits of iterative optimization, hence it is not a feasible option due to excessive compilation time [3, 4, 12, 6].

2.4 Phase Ordering Problem

In multi-phase optimizing compilers, there exist no ideal ordering of phases which results into *phase ordering* problem. For instance, a transformation pass X, optimizes the code such that the effect of some optimizations to be performed by the following pass Y is hindered. Similarly, by switching phases order, pass Y can deprive pass X optimizations. On the contrary, a phase can bring new optimization opportunities for the other. In this situation, it is the responsibility of compiler writers to carefully consider the order in which each optimization phase is performed [4, 3].

Consider an application that is passed through the front end by disabling all optimizations to emit an Intermediate Representation (IR) a. Consider a set of optimizations $o_1, o_2, \ldots, o_n$. For finding the suitable optimization for application, a is required to be passed through the optimization set. The optimizations space due to the phase-ordering problem is in the factorial as permutations are involved, which is represented by Equation (1). Where n is the number of optimizations under study [4, 3].

$$|\Omega_{Phases}| = n!. \tag{1}$$

Considering the optimizations to be applied repeatedly with a variable-length sequence of optimizations. The problem space will be expanded as per Equation (2). Where, m is the maximum desired length for the optimization sequence [4, 3].

$$|\Omega_{Phases\_Repetition\_variableLength}| = \sum_{i=0}^{m} n^i. \tag{2}$$

Even with reasonable n and m, the optimization search space is huge. For instance, with n and m 10, an optimization search space consisting of more than 11 billion different optimization sequences is formed [4, 3].

The phase-order search problem finds an optimal optimization sequence for a program from an infinitely huge space of optimization sequence. The problem is combinatorial in nature having no convexity or linearity properties, thus a given

sequence cannot be called optimal. Only the performance of a sequence can be compared relative to a default compiler optimization sequence (like -O3). The sequence is said to be good for a program, if it is showing a noticeable improvement in performance over default optimization sequences. Therefore, it is not necessary that a good sequence is also an optimal one [33]. Similar to previous works [3, 15, 11], this paper reports the performance speedup relative to LLVM's highest optimization level of -O3.

2.5 MiCOMP

Several techniques [14, 15, 11, 33] have been proposed for search space reduction in the given scenario, but the Mitigates the Compiler Phase-ordering (MiCOMP) has been selected in this paper, due to its *systematic* and *reproducible* approach for reducing the optimization space of those compilers, which exhibit the phase ordering problem. It works by clustering the LLVM's -O3 optimization passes into different groups (sub-sequences). The optimizations order within a group is internally fixed, but the group ordering can be altered.

The phase-ordering is exploited by using the sub-sequences instead of individual optimizations, which reduces the search space significantly. These sub-sequences can be found using any automated clustering technique. MiCOMP is effective as it exploits greater speedup by testing smaller search space as compared to other techniques. MiCOMP reduces the search space (Equation (2)) by fixing n to be 5 and m ranges from 3 to 7 [3]. For the experimentation of MiCOMP, n and m are fixed to 5 and 6, respectively, in [3], as shown by Equation (3):

$$\Omega = \sum_{i=0}^{6} 5^i = 19.5\,\text{k}. \tag{3}$$

Assuming $m = 6$, $n = 63$ (LLVM-3.8), Equation (2) becomes

$$\Omega = \sum_{i=0}^{6} 63^i = 62.5\,\text{b}. \tag{4}$$

Hence, MiCOMP achieves a speedup of $62.5\,\text{b}/19.5\,\text{k} = 3\,201.2\,\text{k}$ approx. over LLVM-3.8[2]. Amongst the given 19.5 k tests, there exists a strong likelihood of *identical* codes, hence such codes are not required to be executed repeatedly. However, identical code testing has not been performed by MiCOMP. Assuming α be the fraction of identical codes out of 1, for the given optimization space, Equation (2) becomes

$$\Omega = \sum_{i=0}^{m} n^i * (1 - \alpha). \tag{5}$$

[2] LLVM-3.8 -O3 has 63 internal passes.

Assuming $\alpha = 20\,\% = 0.2$, MiCOMP Equation (3) becomes

$$\Omega = \sum_{i=0}^{6} 5^i = 19.5\,\mathrm{k} * (1 - 0.2) = 15.6\,\mathrm{k}. \tag{6}$$

In this manner, for $\alpha = 0.2$, MiCOMP search space is reduced by 20 % by ignoring redundant optimizations. Let $P(\alpha)$ be the probability of finding α percent repeated codes, where $0 \leq P(\alpha) \leq 1$. $P(\alpha)$ depends on input application characteristics and the effect of optimizations on that. It is independent of platform features for generic codes.

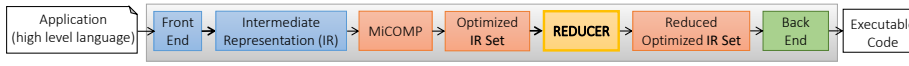


Figure 1. Compilation flow

3 REDUCER

For suppressing the identical codes in LLVM, this paper proposes REDUCER as shown in Figure 1. As it can be observed, an application is compiled by front end, which emits Intermediate Representation (IR), then the IR is passed through MiCOMP's given optimization sub-sequences and REDUCER, which emits unique optimized IR by comparing each new IR with existing old IR. As represented via Equation (7), the un-optimized IR (x_1) is passed through optimization set (opt_n) to get the new IR (y_n). The new IR (y_n) is only retained if it is not identical to old IRs (y_o), otherwise, the IR is discarded. The execution of REDUCER is continued until redundancy checking is not done for all the optimized IRs. Once the REDUCER is stopped, the found unique code IRs are added to the reduced optimized IR set. In the end, the reduced set is passed through the backend for converting the IR codes into target-specific executables.

$$y_n = opt_n(x_1), \text{ if } y_n \neq y_o. \tag{7}$$

The overall compiler optimization space reduction is shown in Algorithm 1. The algorithm receives unoptimized IR, -O3[3] optimized IR, -O3's optimization sequence, the desired number of clusters, and maximum sequence length as input. Firstly, MiCOMP procedure is invoked for constructing the required clusters using a clustering algorithm. Then, a *sequence set* is constructed by inserting the appropriate optimization permutations of length 1 to maximum sequence, which are generated from the set of derived optimization clusters.

After this REDUCER is invoked for generating the optimized IR codes by suppressing redundancies. Firstly, an -O3 optimized IR is added to the optimized set,

[3] -O3 is a baseline to compare optimization sub-sequences performance [3].

Algorithm 1: Compiler optimization space reduction

Input: Un-optimized IR (x_1), O3 optimized IR (x_2), Set of LLVM -O3 optimization sequence $o = \{o_1, \ldots, o_N\}$, Desired number of clusters (*NumClust*), Maximum sequence length (*MaxSeqLen*)

Output: Reduced Executables Set (y)

```
   /* Finding Optimal -O3 Sub-Sequences using MiCOMP                    */
 1 Construct an optimization dependency graph G = (V, E) using o;
 2 Construct a weighted adjacency matrix M from G;
 3 clusters ← M. ApplyClusteringTechnique (NumClust);
 4 for SeqLen in 1 to MaxSeqLen do
 5 |   SeqSet += GeneratePermutations(clusters, SeqLen);
 6 end
   /* Reducing search space by evicting repeated codes using proposed
      REDUCER                                                          */
 7 IRSet.Add(x2);
 8 for flag in SeqSet do
 9 |   temp ← x1.ApplyOptimization(flag);
10 |   if !temp.IsEquivalent(IRSet) then
11 |   |   IRSet.Add(temp);
12 |   end
13 end
   /* Compile IR codes to generate executables                         */
14 for k in IRSet do
15 |   y ← CompileIRtoExecutable(k);
16 end
```

and then a new optimized IR is generated by applying the individual optimization subsequence obtained from the *sequence set*. It is followed by comparing the generated subsequence IR with existing -O3 optimized IR. In the case of different codes, the generated IR is added to the optimized set, otherwise, it is discarded. This process is repeated for all optimization sub-sequences. Each new IR is compared with the ones generated in previous iterations. Finally, the reduced set of IR codes is compiled to generate executables.

The proposed REDUCER algorithm is based on the sequential comparison, which is time-consuming process. However, this timing overhead is justified because it eliminates redundant testing, which is highly beneficial for *big data* applications processing *volume* and *variety* datasets. For a single application, the IR comparison is done one time only, irrespective of the size and format of datasets. Consider an application, operating on 5 datasets with average execution time as d_1 (5 min), d_2 (10 min), d_3 (25 min), d_4 (30 min), d_5 (65 min). With MiCOMP overall time is roughly $(5 + 10 + 25 + 30 + 65\,\text{min} = 135\,\text{min} * 19\,531 = 2\,636\,685\,\text{min})$. However, with the inclusion of REDUCER having $\alpha = 0.8$, and comparison time = 2 880 min, the overall time is roughly $2\,880 + (135 * 3\,907)\,\text{min} = 530\,325\,\text{min}$, which is around 4.9× faster than MiCOMP.

Parameters	**Embedded Workloads**	**Big Data Workloads**
Total RAM	8 GB	64 GB
Total Swap	2 GB	2 GB
Disk Cache	1 GB	1 GB
Model Name	Intel(R) Core(TM) i7-8550U CPU @ 1.80 GHz	Intel(R) Xeon(R) Silver 4216 CPU @ 2.10 GHz
Page Size	4 kB	4 kB
Hard Disk	1 TB SATA Harddisk	1 TB SATA Harddisk
Operating System	Linux Ubuntu 18.04.4 LTS	Linux Ubuntu 18.04.4 LTS
L3 Cache	8 192 KiB Associativity: 16-way Set-associative	8 192 KiB Associativity: 16-way Set-associative
L2 Cache	256 KiB Associativity: 4-way Set-associative	256 KiB Associativity: 4-way Set-associative
L1I, D cache	32 KiB Associativity: 8-way Set-associative	32 KiB Associativity: 8-way Set-associative
Compiler	LLVM-3.8, LLVM-9.0	LLVM-9.0
Benchmark	Ctuning cBench suite v1.1 [13, 5, 16] dataset one	Rodinia [28], Phoenix [31], Cortex Suite [30], Genann [32], Grep [29]
LLVM-9.0	k-means (5 clusters), Python-3.8.1 scikit-learn	Same

Table 1. Experimental setup

4 EXPERIMENTAL SETUP

This section discusses the details of the setup which has been established to test the proposed technique. Firstly, the steps behind finding the optimization clusters for LLVM-9.0 are discussed. Then, the performance benchmarks and metrics are mentioned. Finally, the implementation steps of REDUCER are discussed.

4.1 LLVM 9.0 Clusters

k-means clustering is a partitioning method that tries to discover the k number of clusters. The algorithm specifies the cluster centroid as the mean of the points. Firstly, k is selected randomly of the objects in the data set, each of which represents a cluster mean. For each of the remaining objects, an object is allocated to the cluster, on the basis of shortest Euclidean distance between the cluster mean and the object. Then, the algorithm iteratively improves the within-cluster variation. For each cluster, the new mean is computed using the objects allocated to the cluster in the previous iteration. Finally, all the objects are reassigned using the updated means as the new cluster centers. The iterations continue until the clusters built in the current turn are the same as the previous turn [36, 37].

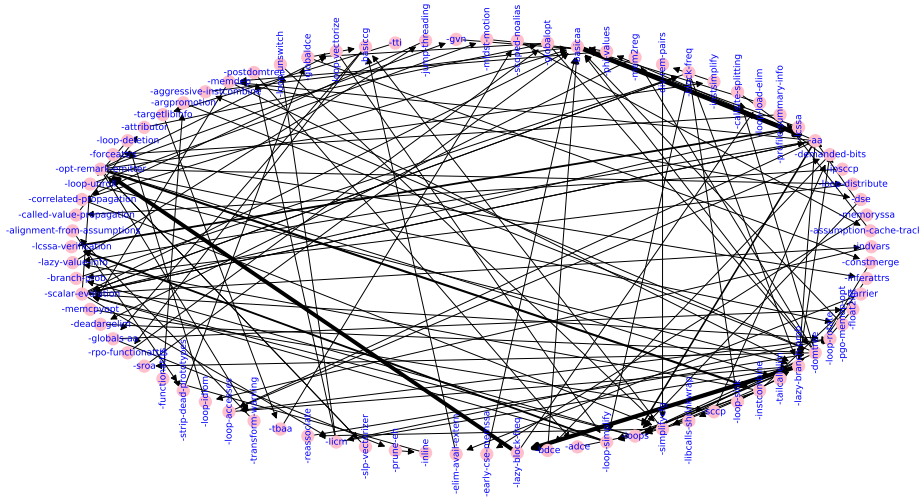


Figure 2. Directed graph for LLVM's 9.0 -O3. Each node represents an optimization pass, edge thickness depicts the strength in the connection between two nodes.

Using MiCOMP [3] approach (Algorithm 1), clusters of size 5 have been found using well-accepted *elbow* method[4] [36] for LLVM-9.0 -O3 optimization sequence via k-means technique in Python, as mentioned in Table 1. The directed graph is shown in Figure 2. The obtained clusters for LLVM-9.0 are presented in Table 2. In comparison to MiCOMP implementation in [3], we believe our implementation is easier, adaptable, and reproducible as it is done using basic k-means technique via Python based library. Whereas, in [3] MATLAB is used with a complex Graph Agglomerative Clustering (GAC) toolbox [26], which is not easily adaptable for producing the results. Our k-means based clusters exploit better performance than GAC as evident by Section 5.4.1. It is possibly because all merges are final in agglomerative clustering that is once a decision is made to combine two clusters it cannot be undone afterward, which prevents a local optimization criterion from becoming a global optimization criterion. This creates difficulty for high-dimensional, noisy, and complex graph data with multiple edges like Figure 2. This issue is tackled by partitioned based k-means clustering technique. Hence, k-means appears to be the suitable choice in a given situation [37].

4.2 Benchmark and Performance Metrics

For evaluating the proposed technique, embedded workloads belonging to automotive, security, office, and telecommunication categories from Collective Benchmark

[4] It chooses the optimal number of clusters by fitting the model for a range of a number of clusters k values [36].

Sub-seq	Compiler Passes (LLVM-3.8) [3]	Our derived Compiler Passes (LLVM-9.0)
A	-ipsccp -globalopt -deadargelim -simplifycfg -functionattrs -argpromotion -sroa -jump-threading -reassociate -indvars -mldst-motion -lcssa -rpo-functionattrs -bdce -dse -inferattrs -prune-eh -alignment-from-assumptions -barrier -block-freq -loop-unswitch -branch-prob -demanded-bits -float2int -forceattrs -loop-idiom -globals-aa -gvn -loop-accesses -loop-deletion -loop-unroll -loop-vectorize -sccp -strip-dead-prototypes -inline -globaldce -constmerge	-forceattrs -inferattrs -callsite-splitting -ipsccp -called-value-propagation -attributor -globalopt -mem2reg -deadargelim -lazy-block-freq -prune-eh -inline -functionattrs -argpromotion -memoryssa -jump-threading -libcalls-shrinkwrap -branch-prob -reassociate -loop-simplify -lcssa-verification -loop-rotate -indvars -loop-idiom -loop-deletion -mldst-motion -gvn -memcpyopt -sccp -dse -barrier -float2int -loop-distribute -loop-vectorize -slp-vectorizer -alignment-from-assumptions -strip-dead-prototypes -constmerge -instsimplify
B	-licm -mem2reg	-lazy-branch-prob -block-freq -licm -loop-unroll -demanded-bits -loop-accesses -loop-sink
C	-loop-rotate -instcombine -loop-simplify	-instcombine -simplifycfg -tailcallelim -loop-unswitch -adce -div-rem-pairs
D	-memcpyopt	-sroa -early-cse-memssa -correlated-propagation -aggressive-instcombine -pgo-memop-opt -lcssa -scalar-evolution -phi-values -bdce -loop-load-elim
E	-loop-unswitch -adce -slp-vectorizer -tailcallelim	-globals-aa -elim-avail-extern -rpo-functionattrs -globaldce

Table 2. Compiler optimizations clusters using MiCOMP for LLVM-3.8 -O3 [3] and our derived for LLVM-9.0 -O3

(cBench) programs [13, 5, 16] are used, as described in Table 3. The evaluation is done in terms of *percentage experiment reduction*, *percentage redundancy fraction*, *speedup*, and *percentage time improvement* metrics represented by Equations (8), (9), (10), and (11).

$$\text{Percentage Experiment Reduction} = \frac{\text{old count} - \text{new count}}{\text{old count}} * 100, \tag{8}$$

$$\text{Percentage Redundancy Fraction} = \frac{\text{Redundant Codes Count}}{\text{Total Codes Count}} * 100, \tag{9}$$

$$\text{Speedup} = \frac{\text{Execution Time}_{base}}{\text{Execution Time}_{new}}, \tag{10}$$

$$\text{Percentage Time Improvement} = \frac{\text{Execution Time}_{base} - \text{Execution Time}_{new}}{\text{Execution Time}_{base}} * 100. \tag{11}$$

4.3 REDUCER Implementation Details

REDUCER has been implemented using *bash script* in Linux with 5 optimization clusters and a maximum sequence length of 6. The implementation is inspired from [22] by comparing the checksum of each IR code with the ones stored in a file. In case, if checksums are not matched, the new code checksum is stored in the file, otherwise, it is discarded. The checksum has been computed using Linux *md5sum*

cBench Programs	Description
automotive_bitcount	Bit counter
automotive_qsort1	Quick sort
automotive_susan_c	Smallest Univalue Segment Assimilating Nucleus Corner
automotive_susan_e	Smallest Univalue Segment Assimilating Nucleus Edge
automotive_susan_s	Smallest Univalue Segment Assimilating Nucleus S
bzip2d	Burrows Wheeler compression algorithm
bzip2e	Burrows Wheeler compression algorithm
consumer_jpeg_c	JPEG compression kernel
consumer_jpeg_d	JPEG decompression kernel
consumer_lame	MP3 encoder
consumer_mad	MPEG audio decoder
consumer_tiff2bw	convert a color TIFF image to gray scale
consumer_tiff2rgba	Convert a TIFF image to RGBA space
consumer_tiffdither	Convert a TIFF image to dither noisespace
consumer_tiffmedian	Convert a color TIFF image to create a TIFF palette file
network_dijkstra	Dijkstra's algorithm
network_patricia	Patricia Trie data structure
office_ispell	Spelling checker
	Text to speech synthesis program
office_stringsearch1	Boyer-Moore-Horspool pattern match
security_blowfish_d	Symmetric-key block cipher Decoder
security_blowfish_e	Symmetric-key block cipher Encoder
security_pgp_d	Pretty Good Privacy decryption algorithm
security_pgp_e	Pretty Good Privacy encryption algorithm
security_rijndael_d	AES algorithm Rijndael Decoder
security_rijndael_e	AES algorithm Rijndael Encoder
security_sha	NIST Secure Hash Algorithm
telecom_adpcm_c	Intel/dvi adpcm coder/decoder Coder
telecom_adpcm_d	Intel/dvi adpcm coder/decoder Decoder
telecom_CRC32	32 BIT ANSI X3.66 crc checksum files
telecom_gsm	GSM for voice encoding/decoding

Table 3. cBench benchmark suite details [5, 16]

command[5]. REDUCER source code has been released on *Github*[6]. The embedded workloads have been run on Intel Core i7 laptop machine with 8 GB RAM, while big data workloads have been run on Intel Xeon Server machine with 64 GB RAM. Both machines have used the same Linux Ubuntu operating system. Further, experimental setup details are shown in Table 1.

[5] *md5sum* uses the MD5 algorithm for printing a 32-character checksum of the given file. A checksum is a string of letters and numbers used to uniquely identify a file.

[6] `https://github.com/hameeza/REDUCER/`

5 RESULTS ANALYSIS

This section analyzes the results in four parts. Firstly, a reduction in experiment count is reported, which is followed by studying the longer sequences exploitation and dynamic programming (DP) analysis. Finally, REDUCER performance is analyzed for embedded and big data workloads.

5.1 Experiment Count Reduction

REDUCER experiment count has been compared with MiCOMP's static 19.5 k[7] via Figures 3 and 4. For all applications and both versions of the compiler, the experiment count has been reduced by a significant amount.

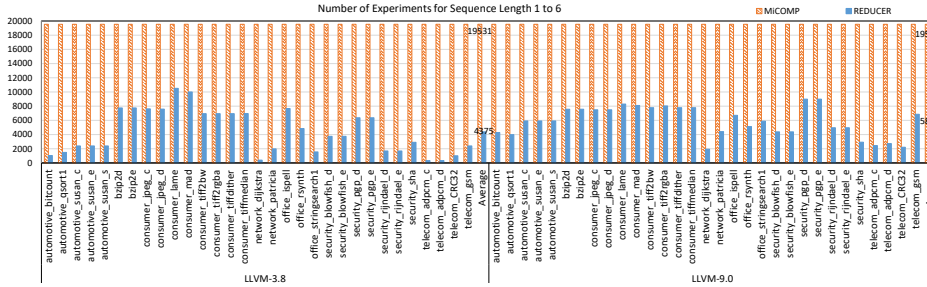


Figure 3. Number of experiments in REDUCER vs. MiCOMP for LLVM-3.8 & 9.0

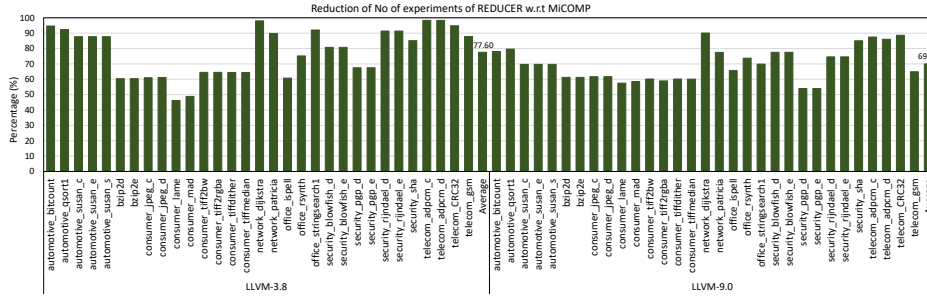


Figure 4. Percentage reduction of number of experiments in REDUCER vs MiCOMP for LLVM-3.8 & 9.0

For embedded workloads (cBench) compilation in LLVM-3.8, it can be observed that REDUCER narrows down the experiment count to 327 from MiCOMP's 19.5 k for *telecom_adpcm_c* and *telecom_adpcm_d*. This large improvement of 98 %

[7] Computed in Equation (3), keeping optimization clusters $= n = 5$ and maximum sequence length $= m = 6$.

is achieved because REDUCER detects 98 % of repeated codes produced by MiCOMP's optimization sub-sequences. Despite varying optimizations, the identical codes are generated because the majority of optimizations produce nil effect on the considered applications.

Whereas, for *consumer_lame* (LLVM-3.8), the experiment count is reduced to 10 503, bringing only 46 % improvement, which are applied on the considered application, thus generating the greater proportion of unique codes. On average, REDUCER brings a decent reduction of 77 % for LLVM-3.8. However, for the same applications and datasets, REDUCER average experiment reduction is lowered to 69.9 % for LLVM-9.0, as the newer optimization sub-sequences produce lesser identical codes. The highest reduction of 90 % is observed for *network_dijkstra* as the experiment count is lowered to 1931. This way, REDUCER discovers the emergence of highly redundant codes via the interaction of LLVM optimization passes.

The derived optimization sub-sequences for different compiler versions show varying code redundancy behavior despite keeping the uniform test environment. For *telecom_adpcm_c*, LLVM-3.8 optimization sub-sequences produce 98 % redundant codes, but LLVM-9.0 sub-sequences produce only 87 % repeated codes. Overall, LLVM-3.8 sub-sequences produce more redundant codes as compared to LLVM-9.0, which is possibly due to increased transformation opportunities in LLVM-9.0 as optimization passes in LLVM-9.0 are greater than LLVM-3.8. By enabling a greater number of transformations, the optimized codes are likely to differ from each other. With the higher experiment reduction in LLVM-9.0, it is expected that REDUCER will bring promising outcomes for future compilers as well.

Furthermore, Table 4 shows the equivalent optimal sub-sequences reported in [3] by MiCOMP for LLVM-3.8. For most applications, it has been found that MiCOMP optimal sub-sequences contain the identical code of the ones already generated in previous generations. For example, for *telecom_adpcm_c*, MiCOMP sub-sequence is ECDDCC (length 6), which is equivalent to EC (length 2), found at the second generation. This way, EC is enough and ECDDCC is not needed.

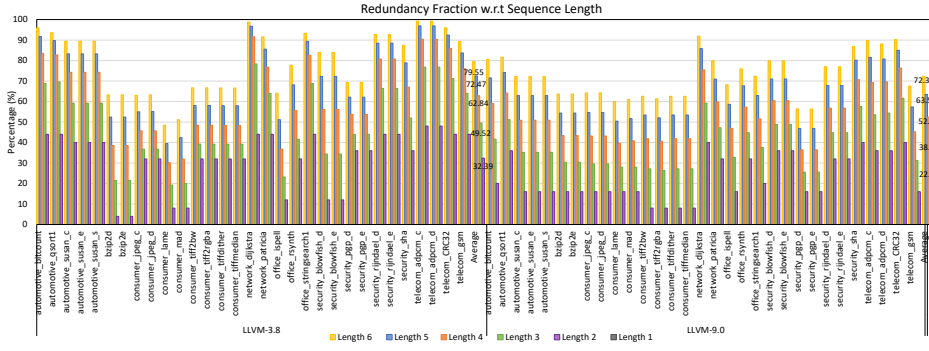


Figure 5. Application redundancy behavior w.r.t. sequence length for LLVM-3.8 & LLVM-9.0

Applications	MiCOMP Optimal Sub-Sequence	Equivalent Optimal Sub-Sequence
automotive_bitcount	BEACCA	BEACA
automotive_qsort1	CBAAAC	CBAAAC
automotive_susan_c	BDBCCB	BBCCB
automotive_susan_e	AABACA	AABACA
automotive_susan_s	ECCCDE	ECE
bzip2d	CBDACA	CBDACA
bzip2e	CBADCA	CBADCA
consumer_jpeg_c	DDC	C
consumer_jpeg_d	CCED	CED
consumer_lame	BCBACB	BCBACB
consumer_mad	DCEDCD	DCEDCD
consumer_tiff2bw	DDCAB	CAB
consumer_tiff2rgba	DDCA	CA
consumer_tiffdither	CCDCD	CDC
consumer_tiffmedian	DEDDC	EC
network_dijkstra	EECBBE	CBE
network_patricia	CECBAA	CECBAA
office_ispell	ABCBAC	ABCBAC
office_rsynth	ABCBA	ABCBA
office_stringsearch1	ABCBAC	ABCBAC
security_blowfish_d	ECEACD	CECAC
security_blowfish_e	BCCEEA	BCCEA
security_pgp_d	DCAACA	CAACA
security_pgp_e	DCA	CA
security_rijndael_d	ACCACE	ACCACE
security_rijndael_e	CAEEC	CAEC
security_sha	DACECA	ACECA
telecom_adpcm_c	ECDDCC	EC
telecom_adpcm_d	DCAACA	CAACA
telecom_CRC32	DCAACA	CAACA
telecom_gsm	DCAAC	CAAC

Table 4. MiCOMP equivalent optimal sub-sequence in LLVM-3.8

5.2 Exploitation of Longer Sequences

The effectiveness of REDUCER in facilitating the exploitation of longer sequences is shown for embedded workloads (cBench), by studying the redundancy behavior w.r.t. sequence length in Figure 5. For all applications (LLVM-3.8 & 9.0) sequence length 1, the redundancy is null. The redundancy is increased as the sequence length is increased. For *automotive_bitcount* (LLVM-3.8), redundancies are 96 %, 91 %, 83 %, 68 %, 44 %, and 0 % for sequence lengths 6, 5, 4, 3, 2, and 1, respectively. The average redundancies are 79 %, 72 %, 62 %, 49 %, 32 %, and 0 %,

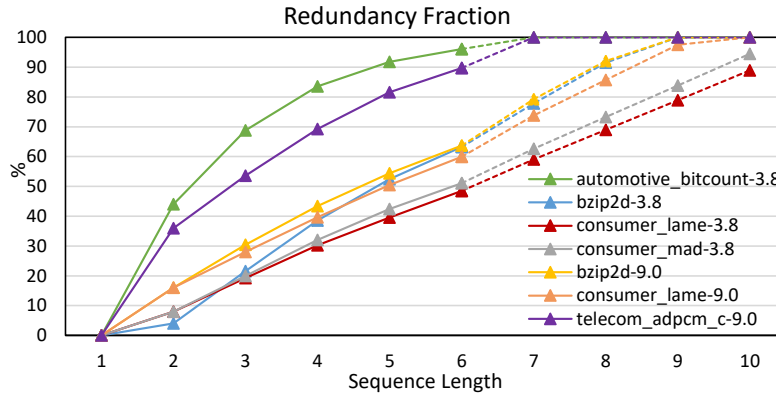


Figure 6. Redundancy fraction w.r.t. sequence length for LLVM-3.8 & LLVM-9.0, dotted lines indicate extrapolated values

respectively, for LLVM-3.8. Whereas, for LLVM-9.0, these values are 72 %, 63 %, 52 %, 38 %, 22 %, and 0 %, respectively. This study discovers how the redundant codes are increased substantially when sequence length is increased. It encourages the testing of longer sequence lengths containing optimal solutions, which are usually not exploited due to a large number of executions. With REDUCER, these longer sequences can be exploited, by the elimination of high proportion redundant codes.

The redundancy fraction has been extrapolated[8] for sequence lengths 7, 8, 9, and 10, which is shown in Figure 6 for few applications. It can be seen for *automotive_bitcount-3.8* and *telecom_adpcm_c-9.0* the expected redundancy is 100 % for length 7 and above. This way, the programmer can safely skip the longer sequences for these applications, without feeling the guilt of missing the optimal by not testing the higher space, as the fraction of unique codes is expected to be minimal in that region. On the contrary, for *consumer_lame-3.8* the predicted redundancies are 59 %, 68 %, 78 %, and 88 %, which are increased but less than 100 %. A similar trend has been observed for *consumer_mad-3.8*, *bzip2d-3.8*, and *bzip2d-9.0*. This way, longer sequence testing is needed for such applications. In this regard, REDUCER can significantly speed the testing process by suppressing the increased proportion of repeated codes in these longer sequences.

5.3 Dynamic Programming Analysis

Dynamic Programming (DP) is a recursive optimization approach that transforms a complex problem into a sequence of simpler sub-problems, and stores the solution to each sub-problem such that it is solved only once. Each time the same sub-

[8] Extrapolation is done using the Excel TREND function.

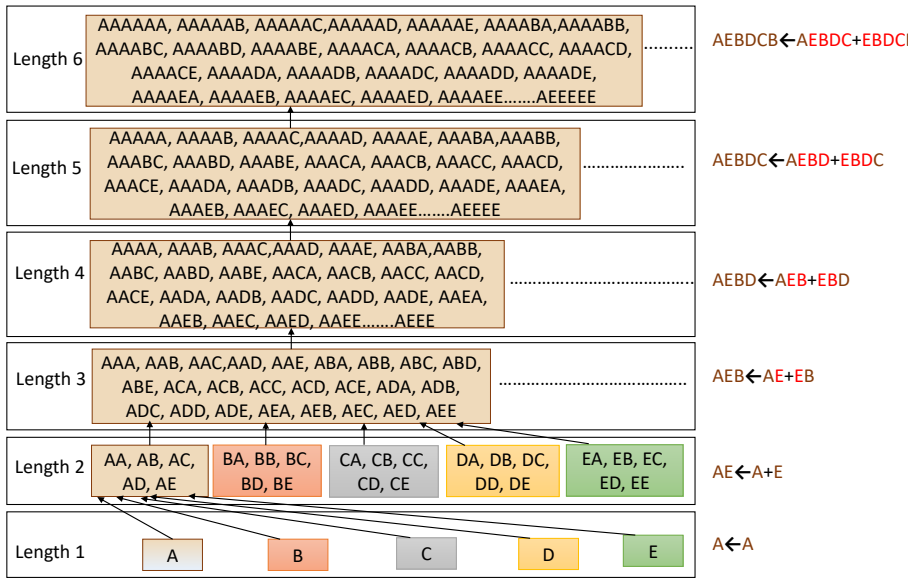


Figure 7. Composition of optimization sequences. Length 2 sequence (AE) is formed by concatenating two length 1 (A + E) sequences. Length 3 (AEB) sequence is formed by merging two sequences of length 2 (AE + EB), only if the first one ends and the second one begins with the same character. Sequences of lengths 4, 5, and 6 are formed similarly.

problem occurs, the previously calculated solutions are used instead of recomputing it, thus computation time is saved [34, 35]. REDUCER retains the unique code sequences by comparing codes with each other. To *estimate the unique code sequences of length 2 to 6*, we have applied Dynamic Programming (DP) technique and compared it with REDUCER. The main motivation behind using DP has been taken from Table 4, where equivalent sequences are a subset of MiCOMP sequences. It implies that large sequences can be constructed by merging two smaller ones. This recursive composition is represented in Figure 7. It can be observed from bottom to top that the length 2 sequence is derived from two length 1 sequences, length 3 from two length 2, and so on. DP initially stores the unique length 1 sequences which derive length 2, then length 2 sequences are stored which derive length 3, and so on.

However, two sequences can be merged only if they have common characters, which means that the second to last characters of the first sequence match with the first to second last characters of the second sequence, as depicted in Figure 7.

The performance of DP is analyzed by means of the sequence estimation accuracy and experiment count increase in Figures 8 and 9. REDUCER shows 100 % accuracy of finding unique sequences for all applications, as it compares each code with the existing ones. Additionally, REDUCER narrows down the experiment

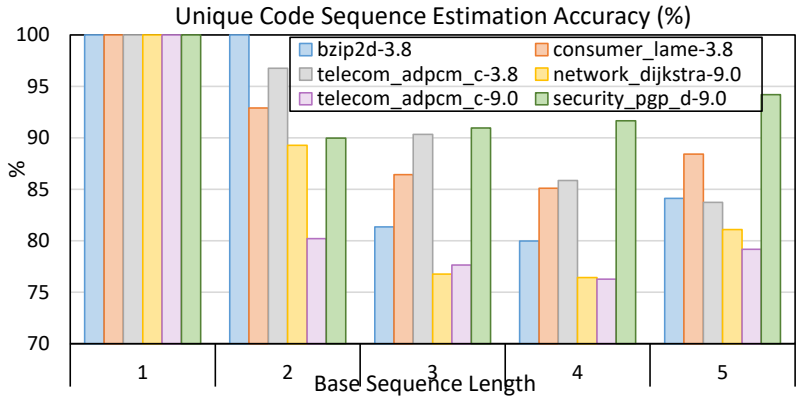


Figure 8. Accuracy of estimating unique optimization sequences via Dynamic Programming (DP) for LLVM-3.8 & LLVM-9.0 (the higher the better). Accuracy computed by dividing correctly estimated DP sequence count with total unique sequence count.

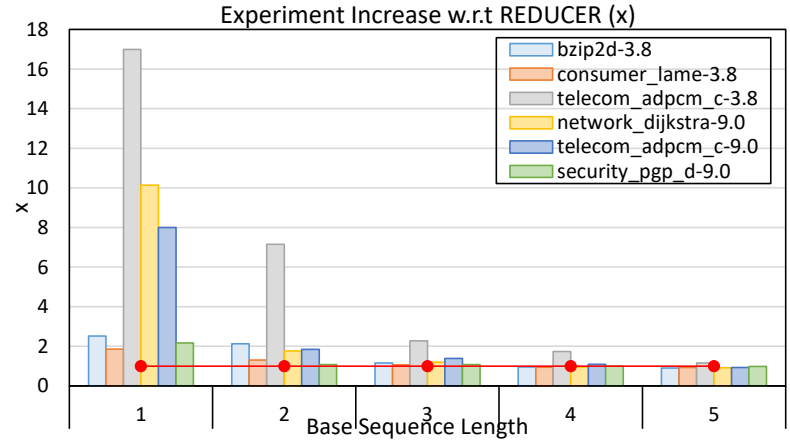


Figure 9. Dynamic Programming (DP) increase in experiment count w.r.t. REDUCER for LLVM-3.8 & LLVM-9.0 (the lower the better). Increase computed by dividing DP estimated experiment count with REDUCER experiment count.

count as it retains only the unique sequences removing all the redundant ones. Unlike REDUCER, DP does not check the actual uniqueness by comparing with existing codes, instead it estimates the unique codes by combining the smaller sequences. This way, DP is likely to be faster than REDUCER. However, with DP there is a higher chance of skipping of unique sequences and inclusion of redundant codes, which results in accuracy loss and experiment count increase.

This work estimates the larger sequences by considering base sequence lengths of 1 to 5. In the case of base length 1, lengths 2 to 6 are estimated by keeping actual

single length unique sequences. For *telecom_adpcm_c_3.8*, *A*, *B*, *C*, and *E* are unique codes. Hence, the larger sequences are formed from these four codes ignoring *D*. In the case of base length 2, length 3 to 6 sequences are estimated by keeping actual double length unique sequences. For *telecom_adpcm_c_3.8*, *AA*, *AB*, *AC*, *AE*, *BA*, *BB*, *BC*, *BE*, *CA*, *CB*, *CE*, *EB*, and *EC* are unique codes. The same criterion is used for base lengths 3, 4, and 5. From Figure 8, 100 % estimation accuracy can be seen for base length 1 in all cases. However, the higher accuracy is at the cost of increased redundant codes. For all applications except *telecom_adpcm_c-3.8*, the single length unique sequences are 5 (A, B, C, D, E), forming 5^2 double length sequences, which results in 5^3, 5^4, 5^5, and 5^6 length 3 to 6 sequences, leading to original 19.5 k experiments. It implies that due to the bottom-up approach, the lower layer behavior is propagated to the upper layers too. As there is no redundancy in the single length sequences, so the upper layers also keep all the sequences. On the contrary, for *telecom_adpcm_c-3.8*, the experiment count is reduced to 5 460, due to four single length unique codes. However, as depicted in Figure 9, still the experiment count is 17× more than REDUCER which is the highest. Similarly, for *network_dijkstra-9.0* experiment count is 10× more.

Furthermore, as per Figure 8, the accuracy starts dropping by increasing the base length, due to missed unique sequences. These misses occur because estimation in base 2, 3, 4, and 5 is done using correct unique sequences. It implies that the problem of estimating larger sequences from smaller ones is not absolute, as the interaction between multiple optimizations is indeterministic. For instance, the sequence *ACCE* might be a unique one, despite its subsequences *ACC* and *CCE* are identical. This behavior can be seen via *bzip2d-3.8*, where accuracy is 100 % for base length 2, but with an increase in length, the accuracy is dropped. It implies that the unique double length combinations are able to derive all the unique higher length sequences, however, the greater length sequences miss several unique combinations. For certain application, a base length shows good accuracy, but for other, the accuracy is not good with the same base. This depends on the interaction effect of optimizations on a certain application, for one application a base covers such sequences whose interaction produces greater unique combinations, thus increasing the accuracy. For other, the effect can be reverse.

The base length increase also reduces the experiment count as the combinations are derived by fixing the unique sequences, which are lesser in quantity. As per Figure 9, for base length 5, all the applications except *telecom_adpcm_c-3.8* show a reduction in experiment count over REDUCER at the cost of lower accuracy. For *security_pgp_d-9.0*, the experiments are lesser than REDUCER at 91.6 % and 94.2 % accuracy for base lengths 4 and 5, respectively. This way, DP is reasonable for *security_pgp_d-9.0* as the experiment increase is not much high for base lengths ≥ 2 and accuracy is decent as well. This behavior is due to the presence of a large number of unique codes in *security_pgp_d-9.0* and also because the base length ≥ 2 sequences are able to derive a higher number of unique sequences of greater lengths. However, the accuracy of less than 100 % makes DP unsuitable because the skipped unique code might be the optimal one with the highest speedup over -O3.

5.4 REDUCER Performance

REDUCER achieves the above experiments reduction (Section 4.3), at the cost of code comparison time which is not present in conventional iterative compilation techniques. In this regard, this section compares the overall time taken by REDUCER and MiCOMP for embedded (Section 5.4.1) and big data (Section 5.4.2) workloads.

5.4.1 Embedded Workloads

A set of embedded applications have been executed from *cBench* suite with dataset *one* on Intel Core i7-8550U laptop machine. The details of the experimental setup and cBench suite are mentioned in Table 1 and 3, respectively. The *cBench* applications have been executed using both REDUCER and MiCOMP for LLVM 3.8, and 9.0. For each application, the execution time is measured by averaging three executions of a loop-wrap. The total time taken by REDUCER is the sum of code *comparison*, *compilation*, and *execution* time. Whereas, MiCOMP total time is the sum of code *compilation* and *execution* time. The individual timings have been reported in Table 5. REDUCER performance is depicted via Figure 10 in terms of total time speedup w.r.t. MiCOMP. Additionally, each application *speedup w.r.t. -O3*, and *optimal sub-sequence* for both LLVM-3.8 and 9.0 have been reported in Table 6. The speedups and optimal sub-sequences are different than reported in [3], due to different test environments.

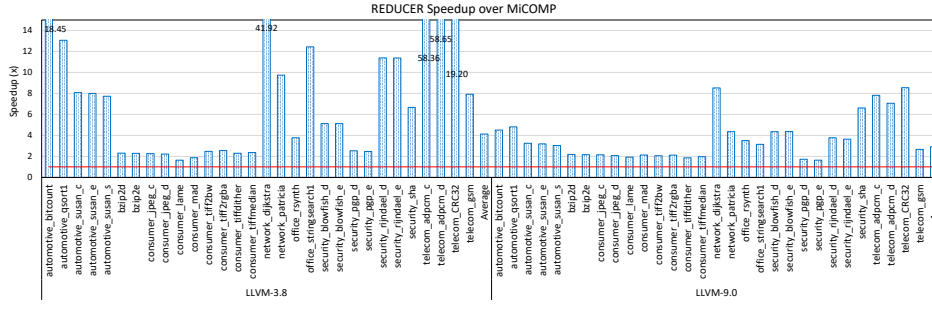


Figure 10. REDUCER time speedup w.r.t. MiCOMP for LLVM-3.8 & LLVM-9.0

As per Table 5, for all the applications the IR comparison time is significantly smaller than the compilation and execution time. This way, REDUCER's prior IR comparison cuts down the experiment count without increasing the time and resource overheads. Hence, the compilation and execution times are shorter for all applications, because these have been measured only for the unique codes which are lesser than the original 19.5 k. Further, a larger comparison time can be seen for applications possessing a greater number of unique codes, due to an increased number of comparisons. Also, the comparison time is increased for larger code

Applications	LLVM-3.8		LLVM-9.0	
	REDUCER Comparison + Compilation & Execution Time	**MiCOMP Execution Time**	**REDUCER Comparison + Compilation & Execution Time**	**MiCOMP Execution Time**
automotive_bitcount	2 m 7.437 s + 1 h 45 m 45.072 s	33 h 9 m 48.835 s	8 m 12.073 s + 12 h 32 m 36.607 s	57 h 18 m 24.959 s
hline automotive_qsort1	2 m 3.219 s + 4 h 6 m 26.163 s	54 h 5 m 32.677 s	13 m 17.055 s + 11 h 48 m 30.306 s	57 h 55 m 5.473 s
hline automotive_susan_c	19 m 15.745 s + 37 h 32 m 1.081 s	305 h 42 m 1.198 s	1 h 27 m 54.887 s + 97 h 31 m 36.153 s	321 h 48 m 36.468 s
hline automotive_susan_e	19 m 0.523 s + 17 h 22 m 1.422 s	141 h 26 m 58.470 s	1 h 28 m 18.939 s + 45 h 51 m 51.6 s	151 h 20 m 20.859 s
hline automotive_susan_s	19 m 0.354 s + 5 h 58 m 43.483 s	48 h 41 m 42.396 s	1 h 27 m 39.483 s + 17 h 11 m 42.905 s	56 h 44 m 21.810 s
hline bzip2d	2 h 25 m 29.997 s + 26 h 36 m 1.811 s	67 h 6 m 52.703 s	4 h 18 m 40.583 s + 24 h 16 m 6.252 s	62 h 38 m 18.913 s
hline bzip2e	2 h 27 m 48.848 s + 24 h 25 m 43.140 s	61 h 38 m 5.735 s	4 h 18 m 19.182 s + 22 h 34 m 24.166 s	58 h 15 m 48.966 s
hline consumer_jpeg_c	4 h 28 m 5.858 s + 34 h 58 m 47.546 s	89 h 40 m 10.199 s	6 h 52 m 26.987 s + 32 h 34 m 40.058 s	85 h 5 m 52.390 s
hline consumer_jpeg_d	4 h 6 m 43.359 s + 25 h 51 m 7.058 s	66 h 31 m 56.764 s	6 h 42 m 40.772 s + 25 h 48 m 32.593 s	67 h 24 m 28.608 s
hline consumer_lame	4 h 40 m 17.279 s + 35 h 50 m 39.655 s	66 h 39 m 17.488 s	6 h 5 m 45.341 s + 28 h 3 m 20.393 s	66 h 3 m 30.571 s
hline consumer_mad	3 h 16 m 23.984 s + 73 h 30 m 15.024 s	143 h 41 m 25.161 s	6 h 57 m 39.909 s + 54 h 12 m 10.991 s	130 h 42 m 44.686 s
hline consumer_tiff2bw	3 h 54 m 23.318 s + 28 h 32 m 20.286 s	80 h 23 m 50.338 s	6 h 49 m 58.180 s + 32 h 12 m 45.765 s	80 h 47 m 40.070 s
hline consumer_tiff2rgba	3 h 53 m 25.429 s + 39 h 4 m 29.516 s	110 h 4 m 41.021 s	6 h 48 m 40.286 s + 47 h 25 m 19.920 s	115 h 24 m 52.931 s
hline consumer_tiffdither	3 h 51 m 34.296 s + 17 h 29 m 27.2 s	49 h 11 m 18.818 s	6 h 46 m 21.959 s + 19 h 37 m 32.768 s	49 h 13 m 28.066 s
hline consumer_tiffmedian	4 h 19 m 45.241 s + 22 h 53 m 27.874 s	64 h 19 m 43.911 s	7 h 17 m 12.163 s + 27 h 2 m 49.544 s	67 h 49 m 46.514 s
hline network_dijkstra	1 m 18.064 s + 6 m 53.783 s	5 h 43 m 36.315 s	7 m 7.499 s + 38 m 15.240 s	6 h 26 m 55.092 s
hline network_patricia	1 m 54.758 s + 3 h 9 m 13.651 s	31 h 2 m 48.228 s	9 m 22.514 s + 8 h 20 m 48.717 s	37 h 2 m 31.854 s
hline office_rsynth	43 m 53.006 s + 10 h 52 m 36.417 s	43 h 50 m 13.451 s	1 h 10 m 37.906 s + 13 h 47 m 16.173 s	52 h 32 m 39.824 s
hline office_stringsearch1	3 m 17.225 s + 3 h 9 m 22.507 s	39 h 57 m 4.458 s	15 m 19.732 s + 4 h 53 m 5.920 s	16 h 12 m 43.740 s
hline security_blowfish_d	14 m 17.719 s + 16 h 6 m 48.013 s	83 h 55 m 21.205 s	22 m 50.546 s + 17 h 1 m 40.431 s	75 h 52 m 39.645 s
hline security_blowfish_e	14 m 10.497 s + 16 h 23 m 46.093 s	85 h 23 m 43.636 s	22 m 43.436 s + 18 h 4 m 13.958 s	80 h 31 m 25.664 s
hline security_pgp_d	4 h 11 m 42.643 s + 19 h 33 m 58.447 s	60 h 8 m 0.460 s	7 h 2 m 54.529 s + 27 h 14 m 40.322 s	59 h 11 m 45.659 s
hline security_pgp_e	4 h 13 m 47.843 s + 17 h 6 m 0.036 s	52 h 33 m 14.188 s	7 h 1 m 54.334 s + 22 h 1 m 1.559 s	47 h 50 m 16.878 s
hline security_rijndael_d	7 m 40.857 s + 5 h 13 m 48.272 s	60 h 59 m 3.278 s	37 m 33.950 s + 14 h 28 m 10.230 s	56 h 59 m 58.524 s
hline security_rijndael_e	7 m 39.601 s + 5 h 9 m 9.975 s	60 h 4 m 58.253 s	37 m 31.819 s + 7 h 44 m 50.671 s	30 h 31 m 9.446 s
hline security_sha	6 m 38.600 s + 16 h 3 m 59.443 s	107 h 50 m 0.057 s	10 m 8.242 s + 13 h 45 m 6.575 s	92 h 4 m 35.121 s
hline telecom_adpcm_c	1 m 16.927 s + 54 m 50.166 s	54 h 35 m 14.502 s	7 m 16.430 s + 6 h 7 m 19.345 s	48 h 50 m 37.601 s
hline telecom_adpcm_d	1 m 16.785 s + 1 h 9 m 40.124 s	69 h 21 m 9.758 s	7 m 9.597 s + 7 h 58 m 36.192 s	57 h 9 m 3.196 s
hline telecom_CRC32	1 m 6.076 s + 1 h 23 m 20.919 s	27 h 1 m 23.813 s	6 m 21.450 s + 3 h 10 m 10.373 s	27 h 59 m 8.641 s
hline telecom_gsm	16 m 42.200 s + 8 h 22 m 53.616 s	68 h 36 m 31.154 s	1 h 22 m 6.525 s + 20 h 30 m 31.041 s	58 h 26 m 57.930 s
hline **Mean**	**1 h 37 m 44.256 s + 17 h 21 m 27.260 s**	**74 h 26 m 51.616 s**	**2 h 54 m 48.210 s + 22 h 48 m 59.559 s**	**72 h 32 m 31.670 s**

Table 5. Comparison of REDUCER with MiCOMP

sizes. For instance, *telecom_adpcm_c* (LLVM-3.8) and *network_dijkstra* (LLVM-9.0) take lesser time because the identical codes are already generated by some previous sub-sequence, stopping the comparison for a code the moment its identical is found. It can be observed that the comparison time of *telecom_CRC32* (LLVM-3.8 & LLVM 9.0) is slightly lesser than *telecom_adpcm_c* (LLVM-3.8) and *network_dijkstra* (LLVM-9.0) due to the smaller IR code size of *telecom_CRC32*. The redundancy proportion of *telecom_CRC32* is higher but lesser than the maximum identical codes count of *telecom_adpcm_c* (LLVM-3.8) and *network_dijkstra* (LLVM-9.0). On the contrary, due to comparing each code with a large number of unique codes, the *consumer_lame* (LLVM-3.8) and *security_pgp_d* (LLVM-9.0) exhibit the maximum comparison time.

Applications	LLVM-3.8		LLVM-9.0		Speedup
	Speedup w.r.t. -O3	Optimal Sub-Sequence	Speedup w.r.t. -O3	Optimal Sub-Sequence	Optimal LLVM 9.0 w.r.t. 3.8
automotive_bitcount	1.08×	ACACA	1.21×	BDADB	1.0×
automotive_qsort1	1.08×	AABCAB	1.04×	CAAADB	0.90×
automotive_susan_c	1.37×	AAAAAE	1.08×	AAAAAA	1.03×
automotive_susan_e	1.23×	BCCEEB	1.06×	AAABCB	1.26×
automotive_susan_s	1.07×	AAAACA	1.24×	AAAAAD	1.008×
bzip2d	1.40×	BDECED	1.14×	DADAC	1.03×
bzip2e	1.46×	BBDE	1.08×	CDBAAB	1.03×
consumer_jpeg_c	1.17×	BAD	1.56×	ACDCDA	1.35×
consumer_jpeg_d	1.45×	BADAAE	1.02×	AAAAAD	1.06×
consumer_lame	1.01×	DBECAA	1.25×	AADDBC	1.28×
consumer_mad	1.23×	ABAC	1.11×	AACBCD	1.01×
consumer_tiff2bw	1.16×	BCEABC	1.05×	BECCCD	1.01×
consumer_tiff2rgba	1.01×	ABAEDA	1.24×	AAAABD	1.43×
consumer_tiffdither	1.08×	EABECB	1.06×	BCDAA	1.01×
consumer_tiffmedian	1.27×	CAEAEB	1.28×	CBCABA	1.02×
network_dijkstra	1.19×	AAAA	1.09×	AAAAB	0.87×
network_patricia	1.13×	AAAABC	1.06×	AAAABC	0.81×
office_rsynth	1.05×	AAAABC	1.26×	BCDCCB	0.98×
office_stringsearch1	1.08×	CABAAE	1.12×	BACCE	1.06×
security_blowfish_d	1.11×	ABADEE	1.006×	BCACAD	1.05×
security_blowfish_e	1.08×	EBAE	1.006×	BBCACA	1.01×
security_pgp_d	1.17×	ADAACC	1.05×	DDABC	1.23×
security_pgp_e	1.05×	BCACDE	1.07×	BDDCBA	1.07×
security_rijndael_d	1.11×	BECACA	1.18×	AACAAC	1.19×
security_rijndael_e	1.12×	ABECAB	1.24×	BBDCBC	1.19×
security_sha	1.13×	CBACAC	1.06×	BABDCC	1.01×
telecom_adpcm_c	1.48×	BB	1.79×	B	1.33×
telecom_adpcm_d	1.15×	CAE	1.24×	CCACDB	1.24×
telecom_CRC32	1.08×	AAAABC	1.06×	BCABD	1.004×
telecom_gsm	1.31×	EBBAAB	1.03×	DCCAAB	1.02×
Mean	**1.16**×	–	**1.14**×	–	**1.06**×

Table 6. Optimal speedup for LLVM 3.8 and 9.0

Despite increased comparison time, REDUCER clearly outperforms MiCOMP for LLVM-3.8 and 9.0, which can be observed via Table 5. For all the applications, REDUCER takes lesser time, as compared to MiCOMP. As depicted via Figure 10, for LLVM-3.8, the maximum speedup of 58.6× is observed for *telecom_adpcm_d*, because a large number of repeated executions have been suppressed. The lowest speedup observed is 1.64× for *consumer_lame*, due to less number of repeated codes.

On average, for LLVM-3.8, a decent speedup of 4.13× is seen. Similarly, LLVM-9.0 shows a maximum speedup of 8.54× for *telecom_CRC32*, lowest speedup of 1.64× for *security_pgp_e*, and average speedup of 2.92×.

The redundancy count is not the only parameter to affect REDUCER performance, instead it is equally affected by code size. For LLVM-9.0, *network_dijkstra* depicts the highest redundancy count, but its speedup is not dominating because its comparison time is greater due to code size. Conversely, *telecom_CRC32* shows the highest speedup due to both smaller code size than *network_dijkstra* and higher redundancy count than the majority of other applications. In this manner, for longer sequence lengths (> 6) and smaller code sizes, the performance of REDUCER is expected to increase exponentially w.r.t. MiCOMP, as Figure 6 depicts the substantial increase in redundancy proportion for longer sequence lengths.

Despite same workloads, the MiCOMP and REDUCER speed is different for both LLVM-3.8 and 9.0, which is possibly due to wide differences in the compiler versions resulting in varying -O3 internal passes. This way, the constructed subsequences widely vary for both versions. It can be observed via Table 5, despite a same number of experiments, the MiCOMP (LLVM-9.0) average speed is higher than LLVM-3.8 because, for the majority of applications, LLVM-9.0 generated codes are executed in lesser time than LLVM-3.8 due to enhanced optimizations. On the contrary, REDUCER (LLVM-3.8) is on the average 1.35× faster than REDUCER (LLVM-9.0). Primarily, two factors are affecting the speed of REDUCER, i.e. experiment count and code execution time. For instance, REDUCER (LLVM-9.0) is slower for *telecom_adpcm_c*, because it is required to process 2.4 k codes which are only 327 with LLVM-3.8. Conversely, for a few applications REDUCER (LLVM-9.0) dominates REDUCER (LLVM-3.8) but with a minor margin, for instance in *bzip2d*, the processing is reduced to 7.5 k, which is 7.7 k with LLVM-3.8. The redundancy count is varied by the impact caused by optimization sequences on a given application, which is indeterministic.

Overall, the sub-sequences are able to exploit reasonable speedup for majority applications as evident by Table 6. On average speedup[9] of 1.16× and 1.14×, are achieved for LLVM-3.8 and LLVM-9.0, respectively. Besides, the maximum speedup is 1.48× and 1.79× for LLVM-3.8 and LLVM-9.0 *telecom_adpcm_c*, respectively. With LLVM-9.0, -O3 has become even more powerful due to increased optimization passes, thus it gets tougher to beat -O3 performance. This way, the speedup w.r.t. -O3 is observed to be lesser for LLVM-9.0.

As per Table 6, LLVM-9.0 optimal sub-sequence is showing greater speedup w.r.t. LLVM-3.8 optimal sub-sequence. The maximum speedup of 1.43× is seen for *consumer_tiff2rgba*, and the average speedup is 1.06×. The speedup is possibly due to LLVM-9.0 being faster than LLVM-3.8 with an enhanced set of optimization passes. Besides, the speedup is greater due to our efficient implementation of MiCOMP for LLVM-9.0, which means that we have derived better LLVM-9.0 -O3 sub-sequences using k-means than the ones reported in [3].

[9] Harmonic mean is used to average the speedup gains [3].

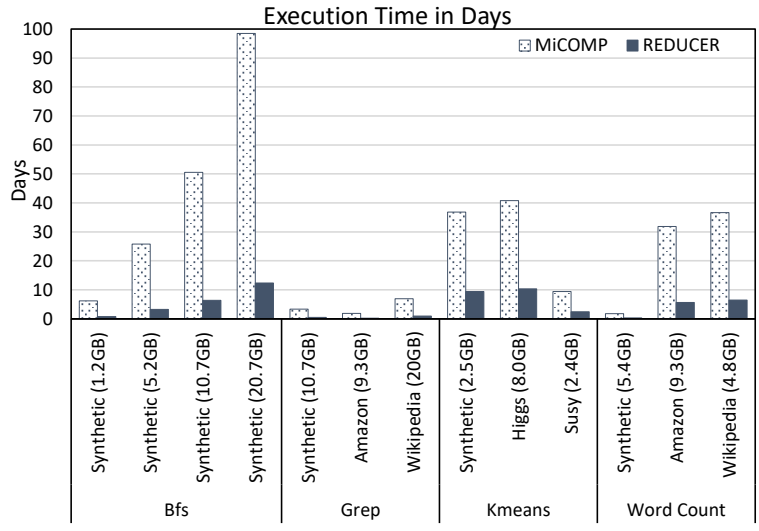
5.4.2 Big Data Workloads

Several well known C/C++ based applications from Rodinia [28], Phoenix [31], CortexSuite [30], genann [32], and grep-bench [29] benchmarks have been tested. Only those applications have been selected which are part of standard big data benchmarks, representing graph mining, classification, clustering, and statistics categories. These include bfs, grep, k-means, word count, etc., as discussed in Table 7. These benchmarks have been run on an Intel Xeon Server machine whose details are listed in Table 1. Each application has been run with 3 to 4 datasets of varying sizes and formats.

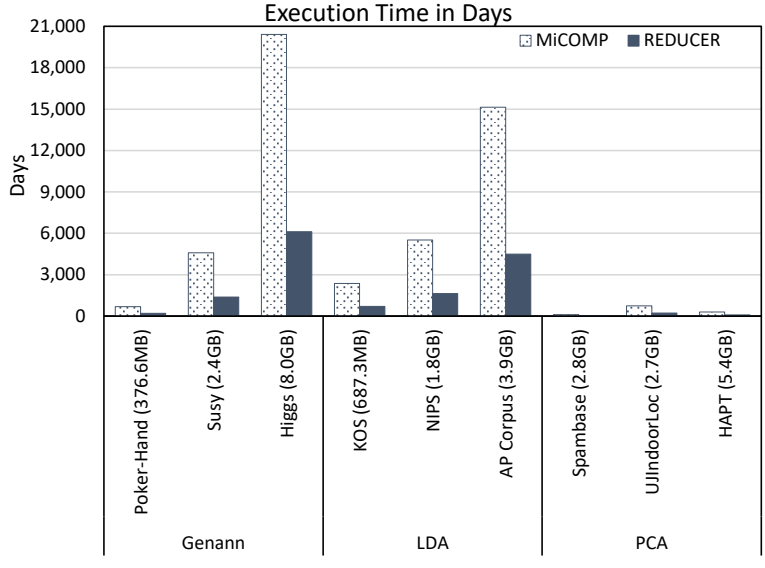
Application	Description	Input Dataset Format
Breadth-First Search (BFS) [28]	Traverses a graph in a breadthward motion.	Graph generated by specifying the number of nodes.
Grep [29]	Searches a file for a particular pattern of characters, and displays the lines containing that pattern.	Text file containing words.
k-means [28]	Represents the data objects by the centroids of the sub-clusters by dividing a cluster of data objects into k sub-clusters.	Dataset consisted of a set of numeric features.
Word Count (WC) [31]	Counts the frequency of occurrence of each unique word in a text document.	Text files containing words.
Gennan [32]	Neural network library for using and training feedforward artificial neural networks (ANN).	Numeric predictive attributes and the class.
Latent Dirichlet Allocation (LDA) [30]	Topic modeling algorithm that is used in natural language processing for discovering topics from unordered documents.	Document is represented as a sparse vector of word counts, in the form: [M][term_1]:[count]... [term_N]:[count]
Principle Component Analysis (PCA) [30]	It is a statistical technique for feature extraction in multivariate datasets.	Data numeric attributes and the class.

Table 7. Benchmark details

The impact of REDUCER is more prominent with big data workloads which consumes larger execution time than conventional cases. As can be observed via Figure 11, for all the applications and datasets REDUCER makes the experimentation feasible by bringing a substantial reduction in execution time. For each application, the comparison and compilation are done only one time. This way, in the Figure 11, only the bar corresponding to the first dataset includes the comparison and compilation times along with execution time, the other bars only involve



a)



b)

Figure 11. REDUCER performance for big data applications

Technique	Environment	Benchmark Suite	Experiment Count per Application	Avg Exe Time Imp w.r.t. Baseline	Collection Time
Less is More [14]	Cortex-M0, LLVM-3.8 Cortex-M3, LLVM-5.0	BEEBS	50, 64 optimizations + (O2 baseline)	2.4 % 5.3 %	–
Lost in translation [15]	Intel i5-6300U, LLVM-6.0 Arm Cortex-A53, LLVM-6.0	CK Milepost-GCC-Codelet	66, 64 optimizations +(O3 baseline)	11.5 % 5.1 %	–
IODC [12]	Intel Xeon, AMD, & Loongson clusters, GCC-4.4	MapReduce & Server Applications	300 random optimizations +(O3 baseline)	32.43 % 10.71 %	110 days 740 days
FFD [27]	Cortex-M0, Cortex-M3, Cortex-A8, Epiphany, XMOS L1, GCC-4.7	MiBench & WCET Applications	2 048 optimizations +(O1, O2 baseline)	–	–
Sensitivity Analysis [10]	Intel Core i7 LLVM-3.8.1	BEEBS	1 728 000	–	–
Hybrid Approach [11]	Intel Core i7-3779, LLVM 3.5	Polybench & cBench	– +(O3 baseline)	8.01 % 6.07 %	–
MiCOMP [3]	Intel Xeon LLVM-3.8 Intel Core i7-8550U, LLVM-3.8, LLVM-9.0	Ctuning cBench	19 530 optimizations +(O3 baseline)	16.66 % 14.41 % 12.50 %	 93 days 91 days
REDUCER	Intel Core i7-8550U, LLVM-3.8, LLVM-9.0	Ctuning cBench	Avg reduction w.r.t. MiCOMP **77.60 %**, **69.98 %**	14.41 % 12.50 %	**24 days** **33 days**

Table 8. REDUCER comparison with existing works

execution time. Hence, the comparison time is spent only for the first dataset execution, the rest datasets execution is comparison free. REDUCER removes the redundant codes at the start, hence executes all the datasets with a reduced number of codes. Whereas, in MiCOMP, each dataset is executed with all the codes including both the unique and identical. It can be observed via Figure 11, REDUCER greatly facilitates the iterative compilation of *bfs (Synthetic-20.7 GB)* dataset, by cutting down execution time to only 12 days from 98 days of MiCOMP. Similarly, for iterative compilation of *PCA (Spambase-2.8 GB)* dataset, only 32 days are required with REDUCER in comparison to 111 days of MiCOMP. Similarly, for other workloads, it can be seen how REDUCER makes the iterative compilation feasible for big data workloads comprising of high volume and variety datasets.

6 RELATED WORK

Several works [13, 22, 21, 20, 19] have emphasized on code comparisons for removing redundant executions. [22, 21, 20, 19] were based on VPO (Very Portable Optimizer) compiler back end, which performed all the analyses and optimizations on a single low-level representation called Register Transfer Lists (RTLs). It detected the iden-

tical function instances by performing three checks including instructions count, instructions byte-sum, and the CRC (Cyclic Redundancy Code) checksum on the bytes of the RTLs. Similarly, in [13] MD5 checksum of assembler code was obtained to verify that no two optimizations combinations generate the same binary. The work selected GCC 200 optimization combinations using a random search strategy. In comparison to these, REDUCER to the best of our knowledge is the first work that detects identical codes in LLVM by comparing the complete IR codes. The granularity of comparison is complete IR code, not just a function or basic block instance.

Conversely, other works [14, 15, 3, 12, 10, 27, 11] did not make the code level comparisons and executed the same code repeatedly. The comparison of these works with REDUCER is depicted via Table 8. In [14, 15], initially, the required performance metrics were tested using standard optimization levels (O2, O3, etc). Then, the metrics were measured by excluding one pass at a time from the standard optimization level, till all the passes were eliminated. In this way, the tested configuration count was lesser, because only the passes present in the standard LLVM optimization level were considered. With this approach, the search space is reduced, but the performance improvement is significantly lesser than the other approaches.

In [12], iterative optimization for the data center (IODC) was proposed which found the optimal compiler configurations for Map Reduce and server applications involving a large collection of massive size datasets. IODC showed greater speedup but at the cost of collection time of 850 days. Despite testing only 300 randomly chosen combinations of compiler optimizations, the collection time was higher due to the execution of a single application with multiple large datasets. However, if the redundancy fraction is $f\,\%$ in the derived optimizations, then all the datasets are required to be executed with these redundant $f\,\%$ codes, increasing the time significantly. In this situation, the integration of REDUCER with IODC can achieve the reported speedup in lesser runs, with a substantial reduction in data collection time, because for an application the redundant codes are checked only one time irrespective of the number of datasets.

The authors in [27] proposed fractional factorial design which reduced the search space for finding optimal optimization combination in GCC. [10] performed sensitivity test for analyzing the impact of 54 LLVM code optimizations on the execution time of applications. Similarly, a design-space exploration was proposed in [11] for searching compiler optimization sequence. The given hybrid approach found optimizations and their order of application, through previously generated sequences for training programs set. Initially, a clustering algorithm selected optimizations, followed by a metaheuristic algorithm for discovering the sequence of optimizations. As per results, the discovered optimized code sequences on average brought the only improvement of 8.01 % and 6.07 % w.r.t. -O3, which is less in comparison to other works. In [3] MiCOMP was proposed, which reduced the search space from billions to few thousand. It did so by clustering LLVM -O3 optimizations into five sub-sequences by using agglomerative clustering. The search space was reduced

because phase ordering was exploited using sub-sequences, instead of individual optimizations. Overall MiCOMP showed significant performance improvement relative to -O3, with 5 clusters and a maximum sequence length of 6 (total 19.5 k experiments). By comparing the proposed works listed in Table 8 with MiCOMP, it can be observed that MiCOMP searches the optimal optimization sequence (better than -O3) in lesser runs for an application. Also, it is evident that MiCOMP sub-sequences exploit greater speedup (w.r.t. -O3) than others. However, MiCOMP did not exclude the identical codes present in 19.5 k sub-sequences, instead, all the permutations were executed to find optimal sub-sequences, increasing the data collection time.

This paper reduces MiCOMP search space by proposing REDUCER which is responsible for eliminating identical codes. In this manner, the repeated execution of the same code is prevented, saving the testing time without affecting the performance accuracy. As per Table 8, the performance improvements and data collection time of MiCOMP reported in [3] and MiCOMP implemented in our work are not comparable because each technique has been tested on different test environments. It can be observed that REDUCER shortens the data collection time to 24 and 33 days (LLVM-3.8 & LLVM-9.0) from MiCOMP's 93 and 91 days without sacrificing the performance improvement w.r.t. baseline. Further, we have extended MiCOMP for LLVM-9.0 by constructing 5 clusters using k-means clustering. Our derived optimization sub-sequences shows average speedup of 1.06× w.r.t. MiCOMP (LLVM-3.8) sub-sequences given in [3].

7 CONCLUSION

The compiler search space reduction technique REDUCER has been presented in this paper. REDUCER relies on straightforward code comparisons to inhibit *identical* code executions. REDUCER has been tested using well-accepted MiCOMP iterative compilation technique with LLVM-3.8 and 9.0. As per reported results, REDUCER substantially accelerates the iterative compilation process in comparison to MiCOMP by eliminating a large number of redundant experiments. In this regard, REDUCER completes the overall iterative compilation of embedded workloads within 24 and 33 days (LLVM-3.8 & LLVM-9.0), respectively, whereas MiCOMP takes 93 and 91 days for the same task.

The promising results of REDUCER (LLVM-9.0) anticipate the high significance of REDUCER for forthcoming compilers as well. Furthermore, REDUCER is proved to be significantly faster for *big data* workloads. Besides, it is found to be simple, generic, and easily adaptable in any iterative compilation technique. In the future, we intend to reduce comparison time by implementing a parallel version of REDUCER. Presently, REDUCER can only detect identical codes, in the future REDUCER will be extended to detect equivalent codes as well, which is expected to further reduce the search space.

REFERENCES

[1] LLVM 2019 (Accessed October 20, 2019). The LLVM Compiler Infrastructure. `https://llvm.org/`.

[2] Aho, A.: Compilers: Principles, Techniques, and Tools (for Anna University). 2/e, 2003.

[3] Ashouri, A. H.—Bignoli, A.—Palermo, G.—Silvano, C.—Kulkarni, S.—Cavazos, J.: MiCOMP: Mitigating the Compiler Phase-Ordering Problem Using Optimization Sub-Sequences and Machine Learning. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 14, 2017, No. 3, Art. No. 29, pp. 1–28, doi: 10.1145/3124452.

[4] Ashouri, A. H.—Killian, W.—Cavazos, J.—Palermo, G.—Silvano, C.: A Survey on Compiler Autotuning Using Machine Learning. ACM Computing Surveys (CSUR), Vol. 51, 2019, No. 5, Art. No. 96, pp. 1–42, doi: 10.1145/3197978.

[5] Ashouri, A. H.—Mariani, G.—Palermo, G.—Park, E.—Cavazos, J.—Silvano, C.: COBAYN: Compiler Autotuning Framework Using Bayesian Networks. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 13, 2016, No. 2, Art. No. 21, pp. 1–25, doi: 10.1145/2928270.

[6] Bodin, F.—Kisuki, T.—Knijnenburg, P.—O'Boyle, M.—Rohou, E.: Iterative Compilation in a Non-Linear Optimisation Space. Workshop on Profile and Feedback-Directed Compilation, 1998.

[7] Chen, M.—Mao, S.—Liu, Y.: Big Data: A Survey. Mobile Networks and Applications, Vol. 19, 2014, No. 2, pp. 171–209, doi: 10.1007/s11036-013-0489-0.

[8] Chong, F. T.—Franklin, D.—Martonosi, M.: Programming Languages and Compiler Design for Realistic Quantum Hardware. Nature, Vol. 549, 2017, No. 7671, pp. 180–187, doi: 10.1038/nature23459.

[9] Cooper, K.—Torczon, L.: Engineering a Compiler. Elsevier, 2011.

[10] de la Torre, J. C.—Ruiz, P.—Dorronsoro, B.—Galindo, P. L.: Analyzing the Influence of LLVM Code Optimization Passes on Software Performance. In: Medina, J., Ojeda-Aciego, M., Verdegay, J., Perfilieva, I., Bouchon-Meunier, B., Yager, R. (Eds.): Information Processing and Management of Uncertainty in Knowledge-Based Systems. Applications (IPMU 2018). Springer, Cham, Communications in Computer and Information Science, Vol. 855, 2018, pp. 272–283, doi: 10.1007/978-3-319-91479-4_23.

[11] de Souza Xavier, T. C.—da Silva, A. F.: Exploration of Compiler Optimization Sequences Using a Hybrid Approach. Computing and Informatics, Vol. 37, 2018, No. 1, pp. 165–185, doi: 10.4149/cai_2018_1_165.

[12] Fang, S.—Xu, W.—Chen, Y.—Eeckhout, L.—Temam, O.—Chen, Y.—Wu, C.—Feng, X.: Practical Iterative Optimization for the Data Center. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 12, 2015, No. 2, Art. No. 15, pp. 1–26, doi: 10.1145/2739048.

[13] Fursin, G.—Temam, O.: Collective Optimization: A Practical Collaborative Approach. ACM Transactions on Architecture and Code Optimization (TACO) Vol. 7, 2010, No. 4, Art. No. 20, pp. 1–29, doi: 10.1145/1880043.1880047.

[14] Georgiou, K.—Blackmore, C.—Xavier-de-Souza, S.—Eder, K.: Less Is More: Exploiting the Standard Compiler Optimization Levels for Better Performance and Energy Consumption. 21st International Workshop on Software and Compilers for Embedded Systems (SCOPES '18), 2018, pp. 35–42, doi: 10.1145/3207719.3207727.

[15] Georgiou, K.—Chamski, Z.—Amaya Garcia, A.—May, D.—Eder, K.: Lost in Translation: Exposing Hidden Compiler Optimization Opportunities. The Computer Journal, 2020, doi: 10.1093/comjnl/bxaa103.

[16] Guthaus, M. R.—Ringenberg, J. S.—Ernst, D.—Austin, T. M.—Mudge, T.—Brown, R. B.: MiBench: A Free, Commercially Representative Embedded Benchmark Suite. Fourth Annual IEEE International Workshop on Workload Characterization (WWC-4), 2001, pp. 3–14, doi: 10.1109/WWC.2001.990739.

[17] Hall, M.—Padua, D.—Pingali, K.: Compiler Research: The Next 50 Years. Communications of the ACM, Vol. 52, 2009, No. 2, pp. 60–67, doi: 10.1145/1461928.1461946.

[18] Hoste, K.—Eeckhout, L.: Cole: Compiler Optimization Level Exploration. 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO '08), 2008, pp. 165–174, doi: 10.1145/1356058.1356080.

[19] Jantz, M. R.—Kulkarni, P. A.: Analyzing and Addressing False Interactions During Compiler Optimization Phase Ordering. Software: Practice and Experience, Vol. 44, 2014, No. 6, pp. 643–679, doi: 10.1002/spe.2176.

[20] Kulkarni, P.—Hines, S.—Hiser, J.—Whalley, D.—Davidson, J.—Jones, D.: Fast Searches for Effective Optimization Phase Sequences. ACM SIGPLAN Notices, Vol. 39, 2004, No. 6, pp. 171–182, doi: 10.1145/996893.996863.

[21] Kulkarni, P. A.—Whalley, D. B.—Tyson, G. S.—Davidson, J. W.: Exhaustive Optimization Phase Order Space Exploration. International Symposium on Code Generation and Optimization (CGO '06), 2006, pp. 1–13, doi: 10.1109/CGO.2006.15.

[22] Kulkarni, P. A.—Whalley, D. B.—Tyson, G. S.—Davidson, J. W.: Practical Exhaustive Optimization Phase Order Exploration and Evaluation. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 6, 2009, No. 1, Art. No. 1, pp. 1–36, doi: 10.1145/1509864.1509865.

[23] Lopes, B. C.—Auler, R.: Getting Started with LLVM Core Libraries. Packt Publishing Ltd, 2014.

[24] Sarda, S.—Pandey, M.: LLVM Cookbook. Packt Publishing Ltd, 2015.

[25] Triantafyllis, S.—Vachharajani, M.—Vachharajani, N.—August, D. I.: Compiler Optimization-Space Exploration. International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization (CGO 2003), 2003, pp. 204–215, doi: 10.1109/CGO.2003.1191546.

[26] Zhang, W.—Zhao, D.—Wang, X.: Agglomerative Clustering via Maximum Incremental Path Integral. Pattern Recognition, Vol. 46, 2013, No. 11, pp. 3056–3065, doi: 10.1016/j.patcog.2013.04.013.

[27] Pallister, J.—Hollis, S. J.—Bennett, J.: Identifying Compiler Options to Minimize Energy Consumption for Embedded Platforms. The Computer Journal, Vol. 58, 2015, No. 1, pp. 95–109, doi: 10.1093/comjnl/bxt129.

[28] Che, S.—Boyer, M.—Meng, J.—Tarjan, D.—Sheaffer, J. W.—Lee, S. H.—Skadron, K.: Rodinia: A Benchmark Suite for Heterogeneous Computing. IEEE International Symposium on Workload Characterization (IISWC), 2009, pp. 44–54, doi: 10.1109/IISWC.2009.5306797.

[29] Grep-Bench, 2019 (Accessed March 18, 2019). `https://github.com/pokle/grep-bench`.

[30] Thomas, S.—Gohkale, C.—Tanuwidjaja, E.—Chong, T.—Lau, D.—Garcia, S.—Taylor, M. B.: CortexSuite: A Synthetic Brain Benchmark Suite. IEEE International Symposium on Workload Characterization (IISWC), 2014, pp. 76–79, doi: 10.1109/IISWC.2014.6983043.

[31] Yoo, R. M.—Romano, A.—Kozyrakis, C.: Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System. IEEE International Symposium on Workload Characterization (IISWC), 2009, pp. 198–207, doi: 10.1109/IISWC.2009.5306783.

[32] C Neural Network Library: Genann, 2019 (Accessed March 01, 2019). `https://codeplea.com/genann`.

[33] Purini, S.—Jain, L.: Finding Good Optimization Sequences Covering Program Space. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 9, 2013, No. 4, Art. No. 56, pp. 1–23, doi: 10.1145/2400682.2400715.

[34] Nalbantoğlu, Ö. U.: Dynamic Programming. In: Russell, D. (Ed.): Multiple Sequence Alignment Methods. Humana Press, Totowa, NJ, Methods in Molecular Biology (Methods and Protocols), Vol. 1079, 2014, pp. 3–27, doi: 10.1007/978-1-62703-646-7_1.

[35] Dasgupta, S.—Papadimitriou, C. H.—Vazirani, U. V.: Dynamic Programming. In: Dasgupta, S., Papadimitriou, C. H., Vazirani, U. V. (Eds.): Algorithms. Chapter 6. Vol. 1, 2006, pp. 169–199.

[36] Han, J.—Pei, J.—Kamber, M.: Data Mining Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems, 2011, pp. 83–124.

[37] Steinbach, M.—Kumar, V.—Tan, P. N.: Cluster Analysis: Basic Concepts and Algorithms. Introduction to Data Mining, Pearson Addison Wesley, 2005.

Hameeza Ahmed received her M.Eng. and B.Eng. degrees in computer and information systems from the NED University of Engineering and Technology, Pakistan in 2015 and 2012, respectively. She is currently pursuing her Ph.D. from the same university. Her research interests include big data computing, compiler optimizations, and computer architecture.

Muhammad Ali Ismail is Professor and Chair at the Department of Computer and Information Systems Engineering, NED University of Engineering and Technology. He is also serving as Director of the High Performance Computing Center and Scientific Director of the Exascale Open Data Analytics Lab, National Center in Big Data and Cloud Computing at the same university. He has more than 16 years experience of research, teaching and administration in both national and international universities. He received his Ph.D. in high performance computing in 2011. Afterwards he pursued his post doctorate in automatic design space exploration from ULBS Romania and become a HiPEAC member. He has published over 65 scientific papers in international journals and conferences along with a U.S. patent. He has won many of the national and international grants of worth above Rs. 200 Million. He is also the recipient of Research Productivity Award by Pakistan Council for Science and Technology, Ministry of Science and Technology, Government of Pakistan. His current research interests include computational HPC, big data mining, cluster and cloud computing, multicore processor architecture and programming, machine learning, heuristics and automatic design space exploration. He is also serving IET Karachi Network as its Vice Chairman.

Computing and Informatics, Vol. 40, 2021, 575–605, doi: 10.31577/cai_2021_3_575

AUTOMATING TEST CASE IDENTIFICATION IN JAVA OPEN SOURCE PROJECTS ON GITHUB

Matej Madeja, Jaroslav Porubän, Michaela Bačíková
Matúš Sulír, Ján Juhár, Sergej Chodarev, Filip Gurbáľ

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
e-mail: {matej.madeja, jaroslav.poruban, michaela.bacikova,
matus.sulir, jan.juhar, sergej.chodarev, filip.gurbal}@tuke.sk

Abstract. Software testing is one of the very important Quality Assurance (QA) components. A lot of researchers deal with the testing process in terms of tester motivation and how tests should or should not be written. However, it is not known from the recommendations how the tests are written in real projects. In this paper, the following was investigated: (i) the denotation of the word "test" in different natural languages; (ii) whether the number of occurrences of the word "test" correlates with the number of test cases; and (iii) what testing frameworks are mostly used. The analysis was performed on 38 GitHub open source repositories thoroughly selected from the set of 4.3 M GitHub projects. We analyzed 20 340 test cases in 803 classes manually and 170 k classes using an automated approach. The results show that: (i) there exists a weak correlation ($r = 0.655$) between the number of occurrences of the word "test" and the number of test cases in a class; (ii) the proposed algorithm using static file analysis correctly detected 97 % of test cases; (iii) 15 % of the analyzed classes used `main()` function whose represent regular Java programs that test the production code without using any third-party framework. The identification of such tests is very complex due to implementation diversity. The results may be leveraged to more quickly identify and locate test cases in a repository, to understand practices in customized testing solutions, and to mine tests to improve program comprehension in the future.

Keywords: Program comprehension, Java testing, testing practices, test smells, open-source projects, GitHub

Mathematics Subject Classification 2010: 68-04

1 INTRODUCTION

The development of automated tests in a software project is a time-consuming and costly process, as it represents more than half of the entire development process [1]. The main aim of testing is to maintain the quality of the product and, in addition to that, tests describe the expected behavior of the production code being tested. Years ago, Demeyer et al. [2] suggested that if the tests are maintained together with the production code, their implementation is the most accurate mirror of the product specification and can be considered as up-to-date documentation. Tests can contain many useful production code metadata that can support program comprehension.

Understanding the code is one of the very first tasks a developer should cope with before the implementation of a particular feature. When the product specification changes (e.g., the requirements for new features are added), the developer must first understand them, then create his/her mental model [3] and finally, the created mental model is expressed in a specific artifact – code implementation. The problem is that two developers are likely to create two different mental models for the same issue because according to Mayer [4] mental model may vary with respect to its completeness and veridicality. A comprehension gap could arise when one developer needs to adapt another programmer's mental model from the code.

An assumption can be made that by using the knowledge about the structure and semantics of tests and their connection to the production code, it is possible to increase the effectiveness of program comprehension and reduce the comprehension gap. This would be possible, for example, by enriching the source code with metadata from the tests directly into the production code, e.g. data used for testing, test scenarios, objects relations, comments, etc. To achieve this goal, it is necessary to know in detail how the tests are actually written and what data they use.

There exist many guidelines on how tests should be created. First, naming conventions may aid the readability and comprehension of the code. According to the empirical study by Butler et al. [5], developers largely follow naming conventions. Our previous research [6] shows that there is a relation between the naming of identifiers in the test code and the production code being tested. This indicates that the relationship between the test and production code is not only at the level of method calls, object instances, or identifier references, but also at the vocabulary level, depending on the domain knowledge and mental model of a tester/developer.

Furthermore, many authors [7, 8, 9] define best practices to simplify the test with the benefit of a faster understanding of the testing code and the identification of test failure. Some guidelines lead to avoiding test smells [10] because as reported by recent studies [11, 12], their presence might not only negatively affect the comprehension of test suites but can also lead to test cases being less effective in finding bugs in the production code. All mentioned approaches are only recommendations but do not really express how the tests are written in real projects. That means we know how tests should be written, but we do not know how they

are written in practice. Many researchers have tried to clarify the motivation of writing tests [13, 14, 15], the impact of test-driven development (TDD) on code quality [16, 17] or the popularity of testing frameworks [18].

To reveal testing practices in real and independent projects it is necessary to find a way to identify test cases in a project, without the time-consuming code analysis. Much more important than the number of test cases is the information where they are located. When a testing framework is used, the test identification is mostly straightforward, e.g. by the presence of the framework imports. On the other hand, to obtain a general overview of testing practices regardless of the used framework, it is advisable to consider tests that do not use any third-party framework and can be regarded as customized testing solutions. In most of the related works, tests are identified by searching specific file and folder names, or some specific keywords. Considering that these keywords usually included the word "test" and based on the authors' experience of Java test cases development, it can be assumed that there is a relation between the word "test" and the number of test cases in a file. That means searching for the "test" string could be beneficial for faster test case identification. Based on the previous reasoning, this paper defines the following hypothesis and research question:

H 1. There is a strong correlation ($r \notin (-0.8, 0.8)$) between the number of occurrences of the word *"test"* in the file content and the number of test cases.

RQ 1. How many testing classes are implemented as customized testing solutions without using any third party framework?

This paper is focused exclusively on unit testing and analyzes 38 projects that have been carefully selected (see Section 3.4.2) from all GitHub projects with Java as a primary language (most of the code written in Java). Section 2 presents the current state and found gaps in the research. In Section 3, the research method is described, containing an examination of whether it is appropriate to search for tests using the word "test" due to different natural languages of developers, an overview of known testing frameworks, and a proposed algorithm for static code analysis to automate the identification of test cases. Section 4 summarizes the results, threats to validity are mentioned in Section 5, and conclusions can be found in Section 6.

2 STATE OF THE ART

Many researchers examine software testing but we still know little about the structure and semantics of test code. This chapter summarizes the related work of software testing from various perspectives.

Learning about real testing practices is a constant research challenge. The goal of such research is mostly to find imperfections and risks, learn, and make recommendations on how to prevent them and how to streamline their development. Leitner and Bezemer [19] studied 111 Java-based projects from GitHub that contain

performance tests. Authors identify tests by searching for one or more terms in the test file name or for the presence of popular framework import, solely in the `src/test` project directory. Selected projects were subjected to manual analysis, in which they monitored several metrics. The most important result for this paper was the fact that 103 projects also included unit tests, usually following standardized best practices. On the other hand, the performance testing approach of the same projects often appears less extensive and less standardized. Another finding was that 58 projects (52 %) mix performance tests freely with their functional test suite, i.e., performance tests are in the same package, or even the same test file, as functional tests. Six projects implemented tests as the usage examples. Using a similar approach [19], in our case by searching for the word "test" and searching for imports of testing frameworks in all project's Java files, we would like to analyze unit tests, but with a careful selection from all GitHub projects at a specific time, resulting in more relevant projects used for analysis.

Code coverage, also known as test coverage, is a very popular method for evaluating project quality. Ellims et al. [20] investigated the usage of unit testing in practice in three projects that authors evaluated as well-tested. Statement coverage was found to be indeed a poor measure of test adequacy. According to the findings of Hemmati [21], basic criteria such as statement coverage are a very weak metric, detecting only 10 % of the faults. A test case may cover a piece of code but miss its faults. According to Hilton et al. [22], coverage can be beneficial in the code review process if a smaller part of the project is evaluated. By reducing coverage to a single ratio of the whole project, much valuable information could be lost. Kochhar et al. [23] performed an analysis of 100 large open-source Java projects showing that 31 % of the projects have coverage greater than 50 % and only 8 % are greater than 75 %.

Many experiments try to express the quality of tests by testing "mutants" [24], i.e., by modifying a program in small ways to create artificial defects. According to Gopinath et al. [25] mutants do not necessarily represent real bugs, therefore, they are not able to relevantly evaluate the quality of the test suite nor to find relations between the coverage and mutants' reveal. However, there is a statistically significant correlation between code coverage and bug kill effectiveness of real software errors (non-mutants) [26]. The quality of the test suite is influenced by the way the mental model is expressed in the code, so examining real tests is more beneficial instead of using mutants.

The fact that unit tests are the most common test type in a project is confirmed by Cruz et al. [27]: 39 % of 1 000 analyzed Android projects used unit tests. Another finding was that frequently updated projects were more aware of the importance of using automated tests than those updated several years ago. The adoption of tests has increased over the last few years, so focusing on information mining from the tests makes sense.

Another type of research was done by Munaiah et al. [28], who focused on the assessment of GitHub projects. They proposed a tool that can be used to identify repositories containing real engineered software projects. The aim was to eliminate

the repository noise such as example projects, homework assignments, etc. One of the metrics they use for assessment is unit test occurrence in the project using test ratio (number of source lines of code in test files to the number of source lines of code in all source files) to quantify the extent of the unit testing effort. Package imports of *JUnit* and *TestNG* frameworks were searched to identify tests in the project. This method could be useful when looking for the occurrence of specific testing frameworks in the code.

3 METHOD

First of all, it is necessary to find suitable projects containing test cases. Thus, metadata of all GitHub open-source projects was obtained via GHTorrent [29] (Section 3.1) due to their high availability. GHTorrent collects projects' metadata from GitHub, one of the biggest project-sharing platform in the world. The experiment was limited to projects with Java as the primary language. Searching for testing frameworks' imports [30] or files containing the word "test" in the filename [19] are common test class identification techniques.

Because our main goal for the future is to improve production code comprehension from a particular test case, we go deeper in this study and try to identify specific test cases (not only test classes), therefore, it is necessary to consider whether the searching for the word "test" is appropriate. Keep in mind, that the aim is not to count the number of test cases in a project. Otherwise, we could run tests via an automated build tool (e.g. ant, maven, or gradle) and collect the number of tests. In that case, the issue is that building such open-source projects often fails [31] and we need to build every single project and run tests what is a time-consuming task. In this paper, we try to count and especially find the location of such test cases.

Since the testing process can also be denoted by other keywords (e.g. verify[1], examine, etc.), an in-depth analysis (Section 3.2) of testing process denotation in various foreign languages was performed, which showed that searching for the word "test" is suitable. Due to the limitations of the GitHub Search API, it was possible to search only one word across all Github Java projects.

As the framework is assumed to influence developer thinking and test case implementation, a list of 50 unit testing frameworks for Java (Section 3.3) has been created. Because the goal is to detect customized testing practices compared with framework-based ones in existing projects, it is not possible to use an automated method, and since it is not possible to manually analyze all GitHub projects, we need to select the most suitable ones. Based on the meaning of the word "test" we assume that there will be a correlation between the number of occurrences of the word "test" (in file content or filename) and the number of test cases.

[1] See Mockito `verify()` method used for soft assertions: `https://javadoc.io/static/org.mockito/mockito-core/3.11.2/org/mockito/verification/VerificationMode.html`

Therefore, three datasets were created using the searching GitHub API for (Section 3.4):

1. the word "test" in filename,
2. the word "test" in file content,
3. frameworks' imports in file content (38 frameworks).

Every single project was searched as mentioned above, 4.3 million projects in total. It is possible to expect that the more occurrences of the word "test" in the project, the more test cases will be present in it and the more we will learn from it in the future. Therefore, projects with the highest occurrence of the word "test" (in file content or filename) or with the highest occurrence of a specific framework's import were selected for manual analysis. By searching for "test" regardless of the framework, we were also able to analyze testing practices without using any third-party framework. Because GitHub contains many projects that are not relevant, e.g. testing, homework, or cloned projects, rules for searching relevant projects have been defined (Section 3.4.2), resulting in a set of projects used for manual and automated analysis. A script for automated analysis was created to partially automate the identification of test cases (see Section 3.5). All methodology details are described in the following sections.

3.1 Data Source

To provide conclusions that are as general as possible, it would be ideal to analyze all types of projects, i.e. proprietary and open source. Because of limited access to proprietary projects, this experiment is focused exclusively on open source projects. GitHub[2] has become one of the most popular web-based services to host both proprietary and mostly open-source projects, therefore, we can consider it a suitable source of projects. It provides an open Application Programming Interface (API)[3] allowing one to work with all public projects (with small exceptions).

To avoid the latency of the official API, the GitHub Archive project[4] stores public events from the GitHub timeline and publishes them via Google BigQuery. Downloading via Google BigQuery is charged, therefore, *GHTorrent* [29] was used instead, which provides a mirror of GitHub projects' metadata. It monitors the GitHub public event timeline, retrieves contents and dependencies of every event, and requests GitHub API to store project data into the database. That includes general info about projects, commits, comments, users, etc. The study data mining started in May 2019, therefore, the last MySQL dump[5] `mysql-2019-05-01` has been used.

[2] `https://github.com/`
[3] `https://docs.github.com/en/rest`
[4] `https://www.gharchive.org/`
[5] `https://ghtorrent.org/downloads.html`

3.2 Denotation of the Word "test"

Leitner et al. [19] searched for tests only in `src/test` directory and test classes identified manually. However, the tests can be placed in any project's directory (e.g. Android[6] uses `src/androidTest`). Another approach is to search for *"test"* string in filenames as executed by Kochhar et al. [15] because they assumed that the tests would be exclusively in files containing the case-insensitive *"test"* string. As in the previous case, best practices lead the developer to use "test" in the file name, but it is not mandatory. For this reason, the most accurate should be searching for the word "test" in the file content. Of course, firstly it is necessary to consider whether the word "test" is the right one for searching. Therefore, the meaning of the word "test" using Google Translate[7] was verified in 109 different languages (all available by Google) as follows:

1. **From English to foreign language and back to English**
   Using this method the most frequent[8] meanings of the word "test" in a foreign language were obtained. By translating them back to English we found out which foreign language translations correspond to the original word "test".
2. **From foreign language to English and back to foreign language**
   The opposite approach was used to find whether the string "test" has a meaning in a particular foreign language. The word was translated into English and all its meanings were verified against the available translation alternatives in the given language.

Multiple translations ensured that the correct meaning of the word in a particular language was understood. Using the first method it was found out that word sets related to the testing process of different foreign languages are mostly translated as "test" in English, see Figure 1. This means that when a foreign developer would like to express something related to testing (e.g. to write a test case), he/she will use mostly the word "test". In this meaning, it is the first choice when searching test cases by a string. Occasionally occurred meaning outside of testing area, e.g., *essay*, *audition* or *flier*. Because such meanings occurred only infrequently, they can be omitted. There were also 14 languages which did not include the word "test" in their reverse translation at all, but its meaning was rather denoting *examination*, *check* or *quiz*.

A total of 44 languages used non-Latin charset. For these languages, the second approach did not make sense to use. For the remaining languages, the meaning was completely identical in 43 languages and the same or similar meaning in 20 cases.

[6] `https://developer.android.com/`

[7] `https://translate.google.com/`

[8] Frequency determined by Google Translate service, indicates how often a translation appears in public documents: 3 – high; 2 – middle; 1 – low frequency.

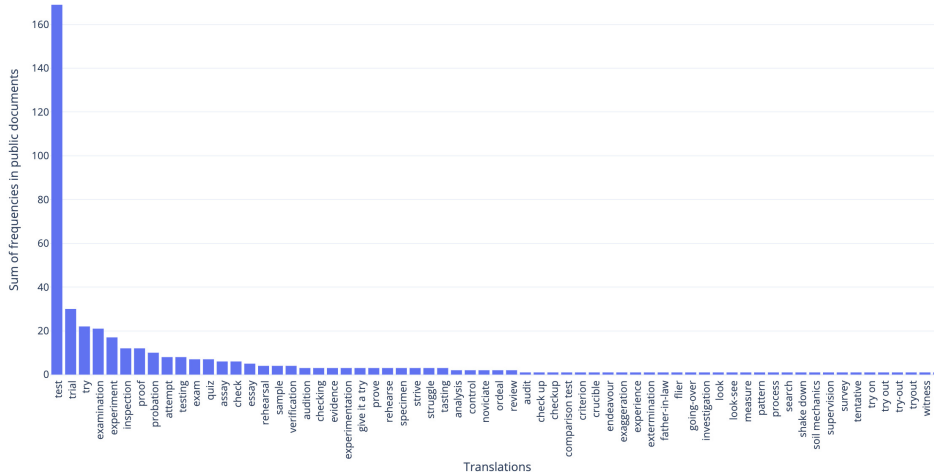


Figure 1. Sum of reverse translation frequency of the word "test" in public documents of different languages

We found only 2 languages (Hungarian[9] and Latvian[10]), in which the word "test" has a completely different meaning, such as *body*, *hew*, or *tool* (nothing related to testing). The analysis shows that the word "test" will refer to the testing process in the code and the meaning can vary in very rare cases. Only the word "test" will be searched for in this study because of the rate limitations of the GitHub API (explained in Section 3.4).

3.3 Java Testing Frameworks

The crucial question is whether developers are motivated to use the word "test" in their code. The developer is often influenced by a testing framework, which leads him or her to different habits. As a part of this study, we analyzed 50 Java unit testing frameworks, extensions, and support libraries (see Table 1) to determine whether the use of the word "test" during test implementation is optional, recommended, or mandatory. The list was created from different sources, such as blogs, technical reports, research papers, etc.

Because it is sometimes difficult to find the boundary between unit and integration testing, the table lists frameworks supporting integration testing under the *unit testing* category. Information about the first version and the last commit may be interesting in terms of the framework lifetime and its occurrence in projects. Projects marked as *archived* or *test generators* in Table 1 were excluded from further analysis for the following reasons:

[9] `https://translate.google.com/?sl=hu&tl=en&text=test`
[10] `https://translate.google.com/?sl=lv&tl=en&text=test`

1. archived projects usually had unavailable documentation or were never released;
2. test generators produce tests that are not based on the programmer's mental model but are generated automatically (semi-randomly), which is not interesting from the code comprehension point of view.

Name	**Package for Import**	**Framework Type**	**First Version**	**Last Commit**	**Must Include "test"**
SpryTest	N/A	U	N/A	N/A (archived)	N/A
Instinct	N/A	B	24.01.2007	07.03.2010 (archived)	N/A
Java Server-Side Testing framework (JSST)	N/A	U	17.11.2010	17.11.2010 (archived)	■
NUTester	N/A	U	05.02.2009	27.03.2012 (archived)	N/A
SureAssert	N/A	A	29.05.2011	04.02.2019 (archived)	N/A
Tacinga	N/A	U	14.02.2018	22.02.2018 (archived)	N/A
Unitils	N/A	U	29.09.2011 (v3.2)	08.10.2015 (archived)	N/A
Cactus	org.apache.cactus	U	11.2008	05.08.2011 (archived)	■
Concutest	N/A	U	30.04.2009	12.01.2010 (archived)	■
Jtest	N/A	G	1997	21.05.2019 (last release)	■
Randoop	N/A	G	23.08.2010	05.05.2020	■
EvoSuite	N/A	G	25.12.2015 (v1.0.2)	30.04.2020	■
JWalk	N/A	G	19.05.2006	14.06.2017	■
TestNG	org.testng	U	31.07.2010 (v5.13)	11.04.2020	■
Artos	com.artos	U	22.09.2018	19.04.2020	■
JUnit 5	org.junit	U	10.09.2017	02.05.2020	■
JUnit 4	org.junit	U	16.02.2006	10.04.2020	■
JUnit 3	junit.framework	U	N/A	N/A	■
BeanTest	info.novatec.bean-test	U	23.04.2014	02.05.2015	■
GrandTestAuto	org.GrandTestAuto	U	21.11.2009	22.01.2014	■
Arquillian	org.jboss.arquillian	U	10.04.2012	21.04.2020	■
EtlUnit	org.bitbucket. bradleysmithllc.etlunit	U	02.12.2013 (v2.0.25)	04.04.2014	■
HavaRunner	com.github.havarunner	U	16.12.2013	08.06.2017	■
JExample	ch.unibe.jexample	U	2008	N/A	■
Cuppa	org.forgerock.cuppa	U	22.03.2016	01.10.2019	■
DbUnit	org.dbunit	U	27.02.2002	24.02.2020	■
GroboUtils	net.sourceforge.groboutils	U	20.12.2002	05.11.2004	■
JUnitEE	org.junitee	U	23.07.2001 (v1.2)	11.12.2004	■
Needle	de.akquinet.jbosscc.needle	U	N/A	16.11.2016	■
OpenPojo	com.openpojo	U	13.10.2010	20.03.2020	■
Jukito	org.jukito	U/M	25.01.2011	17.04.2017	■
Spring testing	org.springframework.test	M/U	01.10.2002	06.05.2020	■
Concordion	org.concordion	U/SbE	23.11.2014 (v1.4.4)	27.04.2020	□
Jnario	org.jnario	B	23.07.2014		□
Cucumber-JVM	io.cucumber	B	27.03.2012	04.05.2020	□
Spock	spock.lang	B	05.03.2009	01.05.2020	□
JBehave	org.jbehave	B	2003	23.04.2020	□
JGiven	com.tngtech.jgiven	B	05.04.2014	10.04.2020	■

JDave	org.jdave	B	18.02.2008	17.01.2013	□
beanSpec	org.beanSpec	B	15.09.2007	27.06.2014 (alpha)	□
EasyMock	org.easymock.EasyMock	M	2001	10.04.2020	■
JMock	org.jmock	M	10.04.2007	23.04.2020	■
JMockit	org.jmockit	M	20.12.2012	13.04.2020	■
Mockito	org.mockito	M	2008	30.04.2020	■
Mockrunner	com.mockrunner	M	2003	16.03.2020	■
PowerMock	org.powermock	M	28.05.2014 (v1.5.5)	30.03.2020	■
AssertJ	org.assertj	A	26.03.2013	05.05.2020	■
Hamcrest	org.hamcrest	A	01.03.2012	06.05.2020	■
XMLUnit	org.xmlunit	A	03.2003	04.05.2020	■

Legend: U – unit; B – behavioural; A – assert; M – mock; G – generator; SbE – specification by example

Table 1: Analyzed unit testing frameworks and extensions for Java

It can be seen that 37 of 50 frameworks require the word "test" as method/class annotation (`@Test`) or part of its name (`testMethod`, `methodTest`). The listed frameworks are mostly extensions that depend on one of the base frameworks, such as *JUnit* or *TestNG*. Different versions of *JUnit* are listed separately because test labeling differs between them (annotations vs. method name format). A deeper analysis of frameworks' JavaDocs revealed that many frameworks include other classes, methods, or annotations that include the word "test" in their names. Although the use of these methods is not mandatory, it may support the search.

3.4 Searching Projects and Data Gathering

The whole process of data gathering can be seen in Figure 2. GHTorrent provided 140 million GitHub projects. From this set all deleted, non-Java, or duplicated projects have been removed. After cleaning the initial data, a total of 6.7 million projects were kept for further analysis.

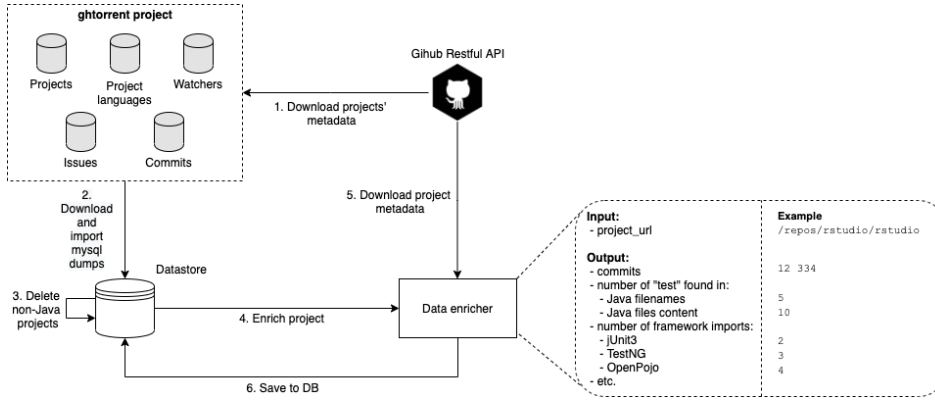


Figure 2. The GitHub data mining process for the study

GHTorrent contained only basic metadata about the projects, which was not sufficient for our needs. Given the meaning of the word "test" (see Section 3.2) we searched for it across all projects. The GitHub API provides a code search[11] endpoint, which index only original repositories. Repository forks are not searchable unless the fork has more stars than the parent repository. If the project has been detected as deleted, private, or blocked by GitHub during querying code search, it has been not considered. Finally, a total of 4.3 million projects were included. For each project, two requests to the GitHub code search API were called, as presented in Table 2. The GitHub code search API had the following limitations:

- up to 1 000 results for each search;
- up to 30 requests per minute (authenticated user);
- global requests rate limited at 5 000 requests per hour;
- only files smaller than 384 KB and repositories with fewer than 500 000 files are searchable.

Search *"test"* in	Example request at `https://api.github.com/search/code`
Java files content	`?q=test+in:file+language:java+repo:apache/camel`
Java filenames	`?q=filename:test+language:java+repo:apache/camel`

Table 2. The GitHub API requests used to search the string *"test"* in a project

3.4.1 Code Search Strategy

GitHub indexes only the default branch code (usually `master`), so the whole analysis was performed only using the default branch. The string "test" can also be a part of other words, e.g. *fas**test***, *las**test***, *thisis**test**framework*. There exist 532 such words containing "test"[12] in total. To avoid inaccuracies when searching for a word of the selected string, false positives must be excluded from the search. When using regular GitHub search, the search term will appear in the results when driven by the following rules:

- string uses camel case convention without numbers[13], e.g., `myTest`,
- string uses snake case convention, e.g., `my_test`, `test_123`;
- string includes a delimiter or special character (space, ., #, $, @, etc.), e.g., `test.delimiter`, `@Test`;
- search is case insensitive, e.g. `Test sentence`, `test sentence`.

[11] `https://docs.github.com/en/rest/reference/search`

[12] `https://www.thefreedictionary.com/words-containing-test`

[13] Numbers can be used, but they are not considered as individual words, e.g. `123Test` or `test123` will not be found.

GitHub considers as Java language file any file with `.java` or `.properties` extensions. The same search rules apply to both search types: file content and filename search. Obviously, according to the above rules, GitHub search automatically filters the results, therefore, unwanted words containing the string "test" do not appear in the results, but neither the words `testing` or `testsAllMethods` will be matched.

3.4.2 Selection of Relevant Projects

When searching for different testing types, the effort is to go through as many projects as possible. Because GitHub contains millions of repositories, it is a challenge to choose the projects that can be the most instructive and filter out irrelevant ones. To make the selection as objective as possible, we planned to use *reaper* tool [28], which can assess a GitHub repository in collaboration with *GHTorrent* using project metadata and code: architecture, community, continuous integration, documentation, history, issues, license, and unit testing. By evaluating all these metrics (see [28] for details), reaper tags a particular repository as a real software project and thus exclude example projects, forks, irrelevant ones, etc.

Many assessment attributes of the *reaper* tool[14] require project files to be available, so each project needs to be cloned or downloaded as an archive. For large projects, it can be gigabytes of data and the size of the project subsequently affects the length of the analysis. To find out whether *reaper* will be beneficial for our study, a manual analysis of 50 projects was performed and the results were compared with the evaluation by *reaper*. All available evaluation attributes were selected except for unit tests assessment because it was limited to *JUnit* and *TestNG* frameworks. The thresholds and weights of particular attributes defined by the developers of the tool were preserved because these values were considered empirically confirmed.

Because we want to select a sample of projects from which we would learn the most, projects with the highest number of files containing the word "test" in its body and filename were selected for the comparison. The same attributes as used by the *reaper* were taken into account in the manual evaluation, but the relevance of the project for this study was assessed by an observer. Evaluation of 50 projects using the *reaper* tool took 10 days, with the most time being spent on evaluating the project architecture. Many repositories with the highest "test" presence in file content or filename were actually identified as *Subversion* (SVN) mirrors[15] by manual analysis and because there were multiple copies of the same code (caused by the SVN's branching style), the projects were not relevant, but the *reaper* assessed such projects as suitable. According to this significant issue, important projects could be lost by assessing project in an automated manner, so it was concluded that

[14] `https://github.com/RepoReapers/reaper`

[15] e.g. `https://github.com/zg/jdk`, `https://github.com/dmatej/Glassfish`, `https://github.com/svn2github/cytoscape`

it is more efficient to select projects manually driven by the following rules, inspired by existing research:

- **Priority** was given to projects with the highest number of the word "test" in the project (in file content and filename). According to [32] we can expect the presence of tests in popular projects. If it is assumed that the word "test" will be correlated with the number of test cases in the project, large and long maintained projects are expected, which authors consider the best sample for the study.
- **History**, as evidence of sustained evolution. Projects under 50 commits were excluded (inspired by the *reaper*) because they represented small or irrelevant projects. Those projects that contained a large number of commits (more than 1 000 per day), considered committed by a robot, were also excluded.
- **Originality** was evaluated by comparing the `readme` file for similarities in other repositories. By such comparison, it is possible to detect clones and similar repositories [33]. Jiang et al. [34] found that developers clone repositories to submit pull requests, fix bugs, add new features, etc. The problem is that developers often do not create forks but project clones (a manual copy of a project), but `readme` file is often unchanged.
- **Community**, as evidence of collaboration, was assessed by the number of contributors in the project. The more developers participate in the project, the more likely it is that the (testing) code will be written in a different style.

3.4.3 Searching Java Testing Frameworks

We were inspired by the work of Stefan et al. [30], who searched for Java performance testing frameworks imports to assess performance testing practices. In our work we are interested in the impact of testing frameworks on test writing, so we also searched for imports of all testing frameworks in Table 1 (excluding generators and archived projects).

Using the search for imports we obtained projects with different testing frameworks. Only projects that contained the word "test" in the Java file body at least once were queried. Because there was a large number of requests (37 per single project), the project set was limited to 500 000, ordered by the number of Java files containing the word "test" in its body, using the following request:

```
https://api.github.com/search/code?q="org.testng"+in:
file+language:java+repo:apache/camel
```

For each testing framework, we created a separate list of projects, sorted by the occurrence of the word "test" in the project, to find projects with a high number of test cases if possible. Original repositories of the searched framework were removed from the analysis (e.g. when searching for JUnit, the original JUnit framework repository was excluded). Subsequently, the selection of relevant projects was performed

according to the steps mentioned in the Section 3.4.2. For some frameworks, e.g. *JExample*[16], which were created as a part of the research [35], no software repositories with business focus were found and as a consequence, it was necessary to include also example, homework, or cloned/forked ones, if the original one was not publicly available.

3.5 Repository Analysis

Three different data sets were received by searching via GitHub API:

1. the word "test" in filename,
2. the word "test" in file content,
3. frameworks' imports in file content.

The first four relevant and top projects (highest "test" or framework's import string occurrence) were manually investigated from each set to find out the test writing practices. The projects were cloned[17] and to keep the consistency between the "test" search and the manual analysis, the project was reverted to the timestamp of GitHub API download using the following command:

```
git checkout ‘git rev-list -n 1
--before="<DOWNLOADED_AT>" "<DEFAULT_BRANCH>"‘
```

For each project, all files with the word "test" in content or filename, or framework's import in file content has been selected as possible option for manual analysis. The project files that contained the largest occurrence of the word "test" and framework's import in their content (expected a higher number of tests) were analyzed first. During the investigation of tests from different authors and projects, we created an automated supportive method for detecting the number of test cases in a file. It does not require compiling the code, such as for computing code coverage, or building abstract syntax tree (AST), e.g. indexing in an IDE.

Regardless of the framework, it is advisable to investigate the count of the following attributes of a source file containing the word "test":

Annotations `@Test`: very popular mostly thanks to *JUnit* and *TestNG*.

Methods containing `test` in the beginning of the name: best practices lead developers to use this convention (also for historical purposes).

Methods containing `Test` in the end of the name: an alternative of previous one.

Public methods: possibly all public methods of a test class can be considered as tests.

[16] `https://github.com/akuhn/jexample`
[17] `git clone`

Occurrence of `main`: customized testing solutions are executed via `main()`.

File path containing `test`: should relate to testing.

Classes containing `$` in the name: the character `$` in a class name mostly denotes a generated code[18] that should not be analyzed.

Total number of `test` occurrence in file content: to reveal the relation between executable test cases and the word "test" presence in the content.

All listed metrics (counts of occurrence in a file) were saved for each analyzed file. The pseudocode for collecting mentioned metrics can be seen in Listing 1 (implementation available at GitHub[19]). The presented algorithm is partly the result of the study because it was created in parallel with the manual analysis. The manual analysis complements the algorithm implementation and vice versa. This algorithm was used to evaluate the test identification for each Java file containing the word "test". Subsequently, the automated identification was checked during the manual analysis to determine the correct number of test cases and the metric used for the calculation (e.g., the number of annotations and public methods can be the same, but the relevant number of tests can only come from one of them). It is necessary to identify the number of particular test cases to link a specific test case with the unit under test (UUT) and its specific method. Each test case is likely to represent a unique use case and thus unique information to enrich the production code.

```
Algorithm predictTests(filePath)
    Input: File path to analyze.
    Output: List of statistical data

    content := load filePath content and remove comments
    nonClassContent := remove all class content, keep only content outside
        of it
        such as imports or class annotations
    classContent := remove all content outside of the class block and keep
        only
        first-level methods without body using /\{([^\{\}]++|(?R))*\}/

    annotations := matches count of regex /@Test/ in classContent
    startsWithTest := matches count of regex
        /public +.*void *.* +[Tt]est[a-zA-Z\\d$\_]* *\(/
        in classContent
    endsWithTest := matches count of regex
        /public +.*void *.* +[a-zA-Z$\_]{1}[a-zA-Z\\d$\_]*Test *\(/
        in classContent
    publicMethods := matches count of regex /public +.*void +.*\(/
        in classContent
    includesMain := matches count of /public +static +void +main.*\(/
        in classContent

    hasDollar := if $ in filename, then true, else false
    testInPath := if "/test" in filePath, then true, else false
```

[18] https://docs.oracle.com/javase/specs/jls/se11/html/jls-3.html#jls-3.8

[19] https://github.com/madeja/unit-testing-practices-in-java/blob/master/AnalyzeProjectCommand.php

```
if TestNG import found in content, then
    if @Test found in nonClassContent, then
        testCaseCount := publicMethods
    else
        testCaseCount := annotations
else if JUnit4 import found in content, then
    testCaseCount := annotations
else if JUnit3 import found in content, then
    testCaseCount := startsWithTest
else if startsWithTest > 0, then
    testCaseCount := startsWithTest
else if annotations > 0, then
    testCaseCount := annotations
else
    testCaseCount := 0

return annotations, startsWithTest, endsWithTest, publicMethods
        includesMain, hasDollar, testInPath, testCaseCount
```

Listing 1. Pseudocode of the algorithm for gathering metadata and identified number of tests in a Java source file

Gathered metadata about test case identification were analyzed from different perspectives. Test classes with the highest number of the following attributes were analyzed:

1. `@Test` annotations,
2. public methods with names starting with `test`,
3. public methods with names ending with `Test`,
4. `main` method,
5. word "test" occurrence.

For framework-dependent searches there was an additional analysis of files with the highest framework import occurrence in the content.

3.6 Hypothesis and Research Question Evaluation

Our null and alternative hypotheses are:

$\mathbf{H}_{\text{null}}$ **1** (H 1)**.** There is **not** a strong correlation ($r \in (-0.8, 0.8)$) between the number of occurrences of the word *"test"* in the file content and the number of test cases in projects with high number of "test" occurrence.

$\mathbf{H}_{\text{alt}}$ **1** (H 1)**.** There is a strong correlation ($r \notin (-0.8, 0.8)$) between the number of occurrences of the word *"test"* in the file content and the number of test cases.

The method of calculating standard Pearson's correlation coefficient [36] was used to confirm or reject H 1. The correlation coefficient was calculated as follows:

$$r = \frac{\sum(x - m_x)(y - m_y)}{\sqrt{\sum(x - m_x)^2 \sum(y - m_y)^2}} \tag{1}$$

where m_x is the mean of the vector x (number of "test" occurrences in file) and m_y is the mean of the vector y (number of identified test cases in file). We will consider the H_{null} 1 as accepted when $r \in (-0.8, 0.8)$, as only absolute correlation higher than 0.8 is commonly considered significant.

To address RQ 1, a class/file will be considered a customized testing solution if the following conditions are met:

- Must include actual tests of production code.
- There is at least one occurrence of the word "test".
- There is no framework import from Table 1.
- File contains `main()` function.

The conditions are based on Section 4.4.2 which shows that customized testing solutions were mostly implemented as common java programs using `main()` function without using any third party framework import.

4 RESULTS

Using the automated script all repositories' files from Table 3 were processed, 38 repositories and 170 076 classes altogether, from which 803 classes and 20 340 test methods were manually investigated. Some special practices in terms of the structure of the testing code or the developer's reasoning were observed. The first 4 projects from Table 3 represent repositories with the largest occurrences of the word "test" in the filename, another 4 in file content and other repositories represent the top import occurrence of a particular framework. The whole dataset of searching "test" via GitHub API can be found at Zenodo[20].

4.1 Accuracy of Automated Test Case Identification

To evaluate the precision of the algorithm from Listing 1, results were compared to manual test identification of 20 340 test cases across all three datasets. Accuracy of 95.72 % for test cases detection was achieved by automated identification considering only test methods, i.e., 95.72 % of all test cases were correctly identified. Considering all 28 975 methods of manually analyzed files (with non-testing ones) a total accuracy of 96.97 % was achieved with the sensitivity of

$$Sensitivity = \frac{true\ positives}{true\ positives + false\ negatives} = \frac{19\,600}{19\,600 + 62} = 0.9968 \qquad (2)$$

and specificity of

$$Specificity = \frac{true\ negatives}{true\ negatives + false\ positives} = \frac{8\,498}{8\,498 + 815} = 0.9125. \qquad (3)$$

[20] `https://doi.org/10.5281/zenodo.4566198`

Repository	Framework	Analyzed Classes		Analyzed Tests		Java KLOC	T_A
		A	M	A	M		
openjdk/client	testng, junit	30 410	130	30 410	1 661	5 149	20 798
SpoonLabs/astor	junit	30 331	36	30 331	1 548	2 338	13 324
apache/camel	junit	10 438	81	10 438	625	1 240	6 847
apache/netbeans	testng, junit	13 056	78	13 056	1 627	5 009	11 908
JetBrains/intellij-community	testng, junit	20 375	49	20 375	4 805	3 842	13 630
SpoonLabs/astor	testng, junit	30 331	44	30 331	5 883	2 338	13 324
corretto/corretto-8	testng, junit	13 688	10	13 688	1 659	3 638	10 792
aws/aws-sdk-java	junit	28 574	18	28 574	302	3 680	20 528
wildfly/wildfly	arquillian	5 109	24	5 109	123	548	3 553
eclipse-ee4j/cdi-tck	arquillian	4 758	30	4 758	139	97	2 748
resteasy/Resteasy	arquillian	2 821	13	2 821	144	220	1 675
keycloak/keycloak	arquillian	1 681	16	1 681	104	396	1 286
jsfunit/jsfunit	cactus	222	13	222	125	21	142
bleathem/mojarra	cactus	737	16	737	250	171	556
topcoder-platform /tc-website-master	cactus	1 635	8	1 635	42	366	1 199
apache/hadoop-hdfs	cactus	325	4	325	20	101	282
zanata/zanata-platform	dbunit	770	21	770	171	197	554
B3Partners/brmo	dbunit	145	18	145	37	47	106
gilbertoca/construtor	dbunit	145	18	145	64	24	53
sculptor/sculptor	dbunit	153	11	153	101	26	103
geotools/geotools	groboutils	3 424	5	3 424	5	1 272	3 659
notoriousre-i-d/ce-packager	groboutils	107	11	107	75	46	91
tliron/prudence	groboutils	16	2	16	3	13	11
MichaelKohler/P2	jexample	36	12	36	53	4	24
akuhn/codemap	jexample	132	15	132	286	41	112
wprogLK/TowerDefenceANTS	jexample	17	3	17	50	9	12
rbhamra/Jboss-Files	needle	44	21	44	30	5	30
akquinet/mobile-blog	needle	19	10	19	33	2	10
s-case/s-case	needle	46	15	46	13	39	33
dbarton-uk/population-pie	needle	7	6	7	16	1	4
abarhub/rss	openpojo	26	2	26	3	6	20
BRUCELLA2 /Prescriptions-Scolaires	openpojo	25	19	25	40	10	18
jpmorganchase/tessera	openpojo	382	8	382	12	45	234
tensorics/tensorics-core	openpojo	161	3	161	1	24	85
orange-cloudfoundry /static-creds-broker	jgiven	21	11	21	33	2	16
eclipse/sw360	jgiven	175	4	175	51	56	161
Orchaldir /FantasyWorldSimulation	jgiven	54	13	54	198	7	37
kodokojo/docker-image-manager	jgiven	11	5	11	8	3	8
Sum		**170 076**	**803**	**363 730**	**20 340**	**31 033**	**127 973**

Legend: A – processed automated; M – investigated manually; KLOC – kilo of lines of code;
T_A – average time of automated test case detection in ms.

Table 3. Statistics of the investigated repositories

Most false positives and false negatives occurrences were caused by customized testing solutions, e.g., when tests were performed directly from the `main()` function by calling methods of the class. If the naming conventions of the called (testing) methods were not governed by the principles of frameworks (e.g., prepending method name with *"test"* or using public methods), not all test cases were detected in an automated way.

4.2 Correlation Between the Number of the Word "test" and the Number of Test Cases in a Class

The proposed algorithm was used to identify all tests in all Java classes of projects from Table 3. The script was used for all Java files that contained string *"test"* in the file content or the filename (in total 170 076 files). Figure 3 shows the correlation with the linear regression line of the word "test" and the number of test cases in a particular class. A standard Pearson's correlation coefficient of $r = 0.655$ was reached (statistical significance $p = 0.0$, rounded on 5 decimal places), that means there is a weak correlation when considering absolute threshold $\alpha = 0.2$ defined in Section 3.6. Nevertheless, from the perspective of finding projects containing tests, this technique is beneficial and can help future experimenters to filter projects containing tests much faster. Because projects have different numbers of test classes and use different frameworks, the detailed ratio of the word "test" occurrence and test case presence per project can be found at GitHub[21].

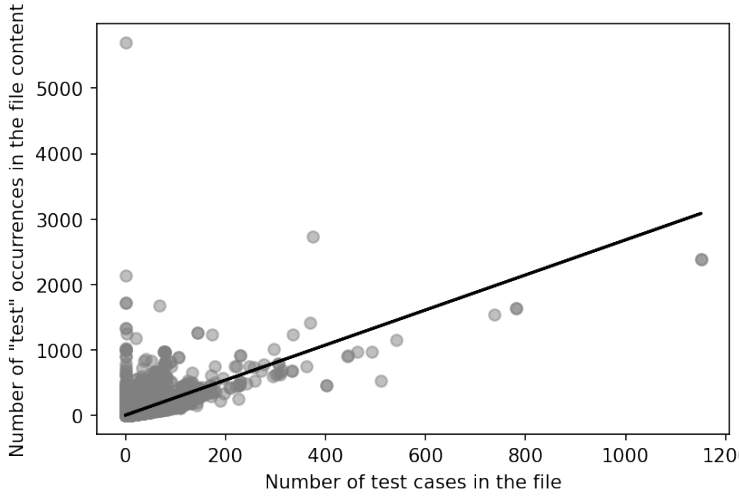


Figure 3. Correlation of the word "test" presence and number of test cases for analyzed classes by automated script

Due to existing research [19] that identified test files using searching "test" in the file path, when limiting our results to files containing "test" in the path (120 907 files) the correlation coefficient of $r = 0.6649$ was reached. On the other hand, 49 169 classes with 3 855 test cases were discarded. Limiting results to files containing "test" in filename (74 530 files), we reached correlation coefficient $r = 0.7004$ with

[21] `https://github.com/madeja/unit-testing-practices-in-java/blob/master/correlation-boxplot.png`

loss of 95 546 classes and 17 440 test cases. By any limitation the correlation did not significantly change, therefore, to find as many test cases as possible it is convenient to search for the word "test" in the file content.

Occurrence of the function `main` without the third party testing framework (more explained in Section 4.4.2) was detected in 26 205 (15.41 %) classes containing the word "test" in their content. The proposed algorithm in Section 3.5 successfully identified test cases in only 6 % classes of this set. Because `main` tests make up a fairly large proportion and the identification of test cases is not clear, it is necessary to investigate this testing style deeper in the future.

> H 1: There is a strong correlation ($r \notin (-0.8, 0.8)$) between the number of occurrences of the word *"test"* in the file content and the number of test cases.
>
> We accept H_{null} 1 and reject H_{alt} 1 because only weak Pearson's correlation coefficient $r = 0.655$ was achieved in general. In some projects, when the correlation was calculated for each project separately, a significant correlation was achieved but so far no relationship has been found concerning the framework, the number of the word "test" presence in the content, or other dependencies.

4.3 Efficiency of the Proposed Automated Test Case Identification

Executing a full code analysis, e.g. in an IDE, of a large project with thousands of kilo of lines of code (KLOC), is a time-consuming task. Such example is the project `openjdk/client` from Table 3. To get faster feedback about tests in a project, the proposed algorithm was used for static source code analysis. Because the proposed automated algorithm should run as a part of an integrated development environment (IDE) extension in the future it should be fast enough. To emulate a similar environment that a developer can use, a laptop with 2.3 GHz Dual-Core Intel Core i5 CPU and 8 GB 2 133 MHz LPDDR3 RAM was used. In Table 3, the average time (T_A) of automated analysis executed 10 times can be seen. The average time of execution was 158ms per KLOC, which authors consider as a satisfactory response time in terms of user experience for use in an IDE extension.

4.4 Revealed Testing Practices

In related work (Section 2) there are best practices that developers can follow and therefore can be expected in the code. During the manual investigation of multiple repositories containing tests, we identified special testing practices used by developers, which are described in the following paragraphs. The listings that are given as examples come from the analyzed repositories, but the code was simpli-

fied for presentation purposes. Code listings refer to GitHub[22] repository of this paper.

4.4.1 Testing Using Third Party Frameworks

Regular test. Tests that follow best practices and avoid test smells fall into this category. They represent the most of occurrences in the projects and since these approaches are already described in the available literature [7, 8, 9], this group will not be given detailed attention. However, the basic aspect of such tests is that information about context and evaluation are available directly in the particular test method (considering also test setup, teardown, and fixtures), thanks to which the test comprehension is straightforward.

Master test. This testing code style represents test classes which contain only one executable test method (see GitHub[23]). *JUnit* will consider only the `all()` method as a test case because it is annotated with `@Test` annotation. Other methods are considered auxiliary ones. The problem with such a notation is the complexity of test comprehension. If the test fails, the developer only has information that the test case titled `all` failed but does not know what the test should have verified, what data was used, etc.

According to the best practices, it should be clear from the test name what the test verifies. In this context, from a semantic point of view, it is possible to consider methods as test cases on lines 1–8 (here from L1–8). The mentioned methods are crucial in terms of failure and understanding of the test, and from the method name, it is also clear what the test verifies. Another disadvantage of these test types is the *assertion roulette* test smell [10] because iterations of the test over the input data make it difficult to determine which data caused the test failure and whether the input data do not interfere with each other between the tests.

Reverse proxy test. If a separate test is written for each use case, the recommendations are met, but this does not mean that it will be easy to comprehend. Some tests call one auxiliary method in multiple tests and the result is evaluated in the auxiliary method. According to the test evaluation manner, they can be divided into:

1. Result evaluation via *method name* (see GitHub[24]).
2. Result evaluation via *internal object state* (see GitHub[25]).

[22] `https://github.com/madeja/unit-testing-practices-in-java`

[23] `https://github.com/madeja/unit-testing-practices-in-java/blob/master/examples/c_masterTest.java`

[24] `https://github.com/madeja/unit-testing-practices-in-java/blob/master/examples/c_reverseProxyMethod.java`

[25] `https://github.com/madeja/unit-testing-practices-in-java/blob/master/examples/c_reverseProxyObject.java`

The first approach is much more difficult to comprehend due to the high degree of abstraction. It is not clear directly from the test method code (L6-8) what is compared during the test because the input data are loaded from a file determined by the test method name (L3). In the `JetBrains/intellij-community` project, from which the example is given, the `doTest()` method is the general one and it was necessary to investigate multiple classes to comprehend how tests are evaluated. At the same time, too generic auxiliary method can result in the *general fixture* test smell.

The second approach is similar to the previous one but uses the internal state of an object (that is initialized before a particular test during test setup) or the `enum` type with different method implementations. The problem may arise when object attribute or method input parameter change the control flow. If the same test is called with different object state or input data, the test logic does not change and therefore it is the same test. However, if the control flow changes in the test, e.g. by some variable value, it can be considered as a separate test (different flow, different test). If the same help method is called more than once, it may behave like two different test cases, which contradicts best practices and makes the comprehension difficult.

Multiple test execution. Server-side applications test different use cases, which require an action after the execution of base functionality, e.g. whether the right content is shown after main test execution (see GitHub[26]). Because of using *JUnit3* in the example, every public method prepended by "test" is considered as test case, so `testEcho()` is executed twice; as a single test case and as a part of `testA4JRedirect()`.

4.4.2 Customized Testing Solutions

Custom testing practices are classic Java programs executable via `main()` function, whose task is to verify the functionality of the production code. Such tests are often written due to the possibility of configuring the execution via command line parameters, which allows variability of test execution. On the other hand, tests should not be so environmentally dependent that they need to be configured to such an extent. The second reason for writing such tests is that they make the code with a large number of test cases more readable. Test methods are called directly from `main()` and, if necessary, also the environment setup is performed in this function. The following ways of calling test methods and objects were observed (examples can be found at GitHub[27]):

Calling methods one by one: all testing methods are manually called from `main()` together with parameters.

[26] `https://github.com/madeja/unit-testing-practices-in-java/blob/master/examples/c_multipleExecution.java`

[27] `https://github.com/madeja/unit-testing-practices-in-java`

Calling methods according to input data: by iterating the test data, specific tests are called based on the current data.

Helper function that returns an array of test cases: the helper method returns an array of instances created from abstract classes, whereas the abstract methods (which represent test cases) are implemented during the instance creation. The `main()` contains an iteration over the array of object instances.

Iterating values of `enum`: similar to the previous one, but it iterates over `enum` values. When creating the `enum`, the method of test class is implemented and the data is set. The test class has its own implementation of a method and state in each iteration.

Calling constructor: in the `main` function the testing class instance is created and the tests are called from the constructor.

There is a problem of how to identify such tests using an automated way and how to determine the number of tests in such a class. The `main()` function also occurs in classic tests (e.g. to run test outside of IDE or without a build automation tool[28]), e.g. based on *JUnit* or *TestNG*. The function can also be found in modified runners of testing frameworks. To clearly distinguish the presence of a customized solution without any framework, it is possible to check the presence of the framework import – if a class contains the `main()` function and an import together, it is a runner or regular test based on the framework, not a customized solution.

Other interesting ways of writing customized tests were also observed. For example, in the `openjdk/client` repository, there were tests for trichotomous relations for which a custom `@Test` annotation was implemented (see GitHub[29]). The annotation is used to indicate the test and, at the same time, to define the type of comparison in the method (L1, L4). Thanks to the word "test" usage, it is possible to detect the correct number of tests, in a similar way as for *JUnit*. In this example, the impact of third party framework on the developer's customized solution is visible. There are many tests in the repository using standardized frameworks, therefore the usage of `@Test` annotation is a logical way of defining a test case. Writing tests manually using a framework would not be as effective and would be difficult to comprehend. On the other hand, such tests in large iterations can easily give rise to the *assertion roulette* test smell, which makes it difficult to identify a test failure.

While in the previous case the test was evaluated using asserts, some approaches have their own error handling. e.g. in the same repository for all *ResourceBundle* classes, a helper test class `RBTestFmwk` has been implemented, which represents a custom framework and test classes inherit from it. The framework provides the processing of the `main()` function parameters, performing tests, and processing results. The test methods to be performed are defined as input parameters. The

[28] `https://junit.org/junit4/faq.html`

[29] `https://github.com/madeja/unit-testing-practices-in-java/blob/master/examples/c_main1.java`

disadvantage is that when performing such tests, it is necessary to know the internal structure of the class, at least method names that need to be performed.

In general, the following risks were observed by analyzing other `main` testing methods:

Execution interruption: If a test fails, execution may be completely interrupted and no further tests will be performed (e.g. raised exception).

Failure identification: Because testing is often performed repeatedly over different data, it can be difficult to identify the exact cause of test failure and in some cases may require debugging the test code.

Dependence: Tests often use the same sources or data for testing and may affect the results of other tests. Also, the tests are often order-dependent and the test order randomness was not found in any repository.

Occurrence of the `main()` function without any third party testing framework was detected in 26 205 (15.41 %) classes containing the word "test" in their content. The proposed algorithm in Section 3.5 successfully identified test cases in only 6 % classes of this set. The set can contain not only testing code, but also a production one. Because such classes make up a fairly large proportion and the identification of test cases is not clear due to the high diversity of writing such tests, it is necessary to carry out an extensive study dealing solely with this issue, to find a way to precisely identify such test cases.

> RQ 1: How many testing classes are implemented as customized testing solutions without using any third party framework?
>
> A total of 15 % of classes were identified as customized testing solutions. The diversity of such tests is very high, therefore, future investigation is needed. This high incidence is probably caused by the nature of big projects with a high occurrence of the word "test" in file content and it is assumed the use of third party frameworks should be more common in smaller projects.

5 THREATS TO VALIDITY

Internal validity: The study relied on GHTorrent databank and GitHub API search algorithm to identify relevant projects. Because only projects with Java as a primary language were selected, testing practices in projects, where Java was not a major language could have been lost. Test classes that did not use the word "test" to indicate a test case were also lost. Searching for test cases was based on best practices and rules of the identified frameworks, but there may still exist other ways of how to identify them. The manual classification was based on observers' experiences and identification of practices out of the generally known recommendations (best practices, test smells, etc.).

Test case detection results were compared to manual ones with an accuracy of 96.97 %. As stated, it is necessary to further investigate customized testing solutions that use regular Java programs to test the production code. The implementation of such programs is often diametrically different and it is difficult to identify test cases. Real test cases were identified by the script in 6 % of classes containing `main()` function.

External validity: To provide generalizable results, 20 k of test cases were analyzed manually and 170 k by an automated way. Also, the meaning and occurrence of the word "test" was analyzed for different natural languages and test frameworks. The results can be used to identify test cases in Java-based projects or projects with a different programming language with the usage of similar testing conventions. Despite the presented observations, our findings, as is usual in empirical software engineering, may not be directly generalized to other systems, particularly to commercial or to the ones implemented in other programming languages.

6 CONCLUSION AND FUTURE WORK

This paper presented an empirical study of Java open source GitHub projects to better understand how to identify test cases and their exact location in the project without the need for deep and time-consuming dynamic code analysis. An algorithm based on searching the word "test" in the repository files content or filenames was proposed and, at the same time, the unusual testing practices were investigated. In total 20 340 test cases in 803 classes were investigated manually and 170 k classes in an automated way. We summarise the most interesting findings from our study:

- There is not a strong correlation between the number of occurrences of the word "test" and the number of test cases in a class.
- Searching for the word "test" in the file content can be used to identify projects containing tests.
- Using static file analysis, the proposed algorithm can correctly detect 97 % of test cases.
- Approximately 15 % of the analyzed files contains "test" in the content together with `main()` function whose represent regular Java programs that test the production code without using any third-party framework. The success rate of identification of such test cases is very low because of implementation diversity.

Several test writing styles were found and classified, along with code samples of the analyzed repositories. Possible code comprehension defects were also mentioned. Based on these findings the following main contributions of this paper are concluded:

- Possibility of fast and automated test case identification together with the exact location in the project.

- Finding of correlation coefficient $r = 0.655$ between the number of occurrences of the word "test" and the number of test cases in a file, which allows to threshold projects or files for similar analysis.
- Overview of observed testing practices concerning the existence of customized testing solutions with an emphasis on places in testing code usable for mining information about the production code.

As future work, we plan to find a solution for accurate identification of test cases in customized solutions, probably based on training a machine learning model with manually labeled test cases of such testing solutions. We believe that studying testing practices can help comprehend the production code more easily. Gathered data could be used for training a machine learning model to automatically recognize tests by the structure and nature of the code. At the same time, we would like to focus on mining information from tests that could support the production source code comprehension and streamline the development process.

Acknowledgement

This work was supported by project VEGA No. 1/0762/19: Interactive Pattern-Driven Language Development.

REFERENCES

[1] Scalabrino, S.—Linares-Vásquez, M.—Poshyvanyk, D.—Oliveto, R.: Improving Code Readability Models with Textual Features. 2016 IEEE 24th International Conference on Program Comprehension (ICPC), 2016, pp. 1–10, doi: 10.1109/ICPC.2016.7503707.

[2] Demeyer, S.—Ducasse, S.—Nierstrasz, O.: Object-Oriented Reengineering Patterns. Elsevier, 2002, doi: 10.1016/B978-1-55860-639-5.X5000-7.

[3] Corritore, C. L.—Wiedenbeck, S.: Mental Representations of Expert Procedural and Object-Oriented Programmers in a Software Maintenance Task. International Journal of Human-Computer Studies, Vol. 50, 1999, No. 1, pp. 61–83, doi: 10.1006/ijhc.1998.0236.

[4] Mayer, R. E.: The Psychology of How Novices Learn Computer Programming. ACM Computing Surveys, Vol. 13, 1981, No. 1, pp. 121–141, doi: 10.1145/356835.356841.

[5] Butler, S.—Wermelinger, M.—Yu, Y.: Investigating Naming Convention Adherence in Java References. 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2015, pp. 41–50, doi: 10.1109/ICSM.2015.7332450.

[6] Madeja, M.—Porubän, J.: Tracing Naming Semantics in Unit Tests of Popular Github Android Projects. In: Rodrigues, R., Janousek, J., Ferreira, L., Coheur, L., Batista, F., Oliveira, H. G. (Eds.): 8th Symposium on Languages, Applications and Technologies (SLATE 2019). OpenAccess Series in Informatics (OASIcs), Vol. 74, 2019, pp. 3:1–3:13, doi: 10.4230/OASIcs.SLATE.2019.3.

[7] Nayyar, A.: Instant Approach to Software Testing: Principles, Applications, Techniques, and Practices. BPB Publications, 2019.

[8] Lewis, W. E.: Software Testing and Continuous Quality Improvement. Second Edition. Auerbach Publications, 2004.

[9] Garcia, B.: Mastering Software Testing with JUnit 5: Comprehensive Guide to Develop High Quality Java Applications. Packt Publishing, 2017.

[10] Van Deursen, A.—Moonen, L. M. F.—Van Den Bergh, A.—Kok, G.: Refactoring Test Code. Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001), 2001, pp. 92–95.

[11] Peruma, A.—Almalki, K.—Newman, C. D.—Mkaouer, M. W.—Ouni, A.—Palomba, F.: On the Distribution of Test Smells in Open Source Android Applications: An Exploratory Study. Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering (CASCON '19), 2019, pp. 193–202.

[12] Spadini, D.—Palomba, F.—Zaidman, A.—Bruntink, M.—Bacchelli, A.: On the Relation of Test Smells to Software Code Quality. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018, pp. 1–12, doi: 10.1109/ICSME.2018.00010.

[13] Linares-Vásquez, M.—Bernal-Cardenas, C.—Moran, K.—Poshyvanyk, D.: How Do Developers Test Android Applications? 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 613–622, doi: 10.1109/ICSME.2017.47.

[14] Beller, M.—Gousios, G.—Panichella, A.—Zaidman, A.: When, How, and Why Developers (Do Not) Test in Their IDEs. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015), ACM, 2015, pp. 179–190, doi: 10.1145/2786805.2786843.

[15] Kochhar, P. S.—Thung, F.—Nagappan, N.—Zimmermann, T.—Lo, D.: Understanding the Test Automation Culture of App Developers. 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015, pp. 1–10, doi: 10.1109/ICST.2015.7102609.

[16] Fucci, D.—Erdogmus, H.—Turhan, H.—Oivo, M.—Juristo, N.: A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last? IEEE Transactions on Software Engineering, Vol. 43, 2017, No. 7, pp. 597–614, doi: 10.1109/TSE.2016.2616877.

[17] Bissi, W.—Serra Seca Neto, A. G.—Pereira Emer, M. C. F.: The Effects of Test Driven Development on Internal Quality, External Quality and Productivity: A Systematic Review. Information and Software Technology, Vol. 74, 2016, pp. 45–54, doi: 10.1016/j.infsof.2016.02.004.

[18] Zerouali, A.—Mens, T.: Analyzing the Evolution of Testing Library Usage in Open Source Java Projects. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2017, pp. 417–421, doi: 10.1109/SANER.2017.7884645.

[19] Leitner, P.—Bezemer, C.-P.: An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Projects. Proceedings of the 8th

ACM/SPEC International Conference on Performance Engineering (ICPE '17), ACM, 2017, pp. 373–384, doi: 10.1145/3030207.3030213.

[20] Ellims, M.—Bridges, J.—Ince, D. C.: Unit Testing in Practice. 15th International Symposium on Software Reliability Engineering, 2004, pp. 3–13, doi: 10.1109/ISSRE.2004.44.

[21] Hemmati, H.: How Effective Are Code Coverage Criteria? 2015 IEEE International Conference on Software Quality, Reliability and Security, 2015, pp. 151–156, doi: 10.1109/QRS.2015.30.

[22] Hilton, M.—Bell, J.—Marinov, D.: A Large-Scale Study of Test Coverage Evolution. Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018), ACM, 2018, pp. 53–63, doi: 10.1145/3238147.3238183.

[23] Kochhar, P. S.—Lo, D.—Lawall, J.—Nagappan, N.: Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects. IEEE Transactions on Reliability, Vol. 66, 2017, No. 4, pp. 1213–1228, doi: 10.1109/TR.2017.2727062.

[24] Just, R.—Jalali, D.—Inozemtseva, L.—Ernst, M. D.—Holmes, R.—Fraser, G.: Are Mutants a Valid Substitute for Real Faults in Software Testing? Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014), ACM, 2014, pp. 654–665, doi: 10.1145/2635868.2635929.

[25] Gopinath, R.—Jensen, C.—Groce, A.: Mutations: How Close Are They to Real Faults? 2014 IEEE 25th International Symposium on Software Reliability Engineering, 2014, pp. 189–200, doi: 10.1109/ISSRE.2014.40.

[26] Kochhar, P. S.—Thung, F.—Lo, D.: Code Coverage and Test Suite Effectiveness: Empirical Study with Real Bugs in Large Systems. 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015, pp. 560–564, doi: 10.1109/SANER.2015.7081877.

[27] Cruz, L.—Abreu, R.—Lo, D.: To the Attention of Mobile Software Developers: Guess What, Test Your App! Empirical Software Engineering, Vol. 24, 2019, No. 4, pp. 2438–2468, doi: 10.1007/s10664-019-09701-0.

[28] Munaiah, N.—Kroh, S.—Cabrey, C.—Nagappan, M.: Curating GitHub for Engineered Software Projects. Empirical Software Engineering, Vol. 22, 2017, pp. 3219–3253, doi: 10.1007/s10664-017-9512-6.

[29] Gousios, G.: The GHTorent Dataset and Tool Suite. 2013 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 233–236, doi: 10.1109/MSR.2013.6624034.

[30] Stefan, P.—Horky, V.—Bulej, L.—Tuma, P.: Unit Testing Performance in Java Projects: Are We There Yet? Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE '17), ACM, 2017, pp. 401–412, doi: 10.1145/3030207.3030226.

[31] Sulír, M.—Bačíková, M.—Madeja, M.—Chodarev, S.—Juhár, J.: Large-Scale Dataset of Local Java Software Build Results. Data, Vol. 5, 2020, No. 3, Art. No. 86, doi: 10.3390/data5030086.

[32] Pham, R.—Singer, L.—Liskin, O.—Filho, F. F.—Schneider, K.: Creating a Shared Understanding of Testing Culture on a Social Coding Site. 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 112–121, doi: 10.1109/ICSE.2013.6606557.

[33] Zhang, Y.—Lo, D.—Kochhar, P. S.—Xia, X.—Li, Q.—Sun, J.: Detecting Similar Repositories on GitHub. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2017, pp. 13–23, doi: 10.1109/SANER.2017.7884605.

[34] Jiang, J.—Lo, D.—He, J.—Xia, X.—Kochhar, P. S.—Zhang, L.: Why and How Developers Fork What from Whom in GitHub. Empirical Software Engineering, Vol. 22, 2017, No. 1, pp. 547–578, doi: 10.1007/s10664-016-9436-6.

[35] Kuhn, A.—Van Rompaey, B.—Haensenberger, L.—Nierstrasz, O.—Demeyer, S.—Gaelli, M.—Van Leemput, K.: JExample: Exploiting Dependencies Between Tests to Improve Defect Localization. In: Abrahamsson, P., Baskerville, R., Conboy, K., Fitzgerald, B., Morgan, L., Wang, X. (Eds.): Agile Processes in Software Engineering and Extreme Programming (XP 2008). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 9, 2008, pp. 73–82, doi: 10.1007/978-3-540-68255-4_8.

[36] Pearson's Correlation Coefficient. In: Kirch, W. (Eds.): Encyclopedia of Public Health. Springer, Dordrecht, 2008, p. 1090, doi: 10.1007/978-1-4020-5614-7_2569.

Matej Madeja graduated (M.Sc.) at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice in 2017. He defended his Master's thesis in the field of informatics. Currently, he is Ph.D. student in the same department. His research is focused on the improvement of program comprehension efficiency, source code testing techniques, and teaching of programming.

Jaroslav Porubän is Professor and the Head of Department of Computers and Informatics, Technical University of Košice, Slovakia. He received his M.Sc. degree in 2000 and his Ph.D. in 2004, both in computer science. Since 2003 he is a member of the Department of Computers and Informatics at the Technical University of Košice. Currently, the main subject of his research is computer language engineering focused on the design and implementation of domain-specific languages and computer language composition and evolution.

Michaela Bačíková is Assistant Professor and the Head of the Information Systems Laboratory at the Department of Computers and Informatics, Technical University of Košice, Slovakia. She received her Ph.D. in computer science in 2014. Since 2014 she is a member of the Department of Computers and Informatics at the Technical University of Košice. Currently, the main subject of her research is UX, HCI and usability while focusing on the domain-related terminology in user interfaces (domain usability). Her interest is also in software languages and innovations in the teaching process.

Matúš Sulír is Assistant Professor at the Department of Computers and Informatics, Technical University of Košice, Slovakia. At the same university, he graduated with his Master's degree in computer science in 2014 and obtained his Ph.D. in 2018. His research is focused on program comprehension, particularly on the integration of run-time information with source code, attribute-oriented programming, and debugging. He is also interested in empirical studies in software engineering.

Jan Juhár is Researcher at the Department of Computers and Informatics, Technical University of Košice. He received his Ph.D. in computer science in 2018. Since 2018 he is a member of the Department of Computers and Informatics at the Technical University of Košice. His research focuses on program comprehension, programming tools, source code metadata, and program projections.

Sergej Chodarev is Assistant Professor at the Department of Computers and Informatics, Technical University of Košice, Slovakia. He received his Master's degree in 2009 and his Ph.D. in 2012, both in computer science. The subject of his research includes domain-specific languages, metaprogramming, and software engineering.

Filip Gurbáľ is Ph.D. student at the Department of Computers and Informatics, Technical University of Košice, Slovakia. He graduated at the university in computer science in 2020. He is a member of the Computer Network Laboratory at the Technical University of Košice. The subject of his research is improving program comprehension using methods and tools. He also focuses on software testing methods and tools.

Computing and Informatics, Vol. 40, 2021, 606–627, doi: 10.31577/cai_2021_3_606

REAL TIME MOBILE AD INVESTIGATOR: AN EFFECTIVE AND NOVEL APPROACH FOR MOBILE CLICK FRAUD DETECTION

Iroshan Aberathne

Department of Information and Communication Technology
Faculty of Technology, University of Sri Jayewardenepura
Sri Lanka
e-mail: iroshan@sjp.ac.lk

Chamila Walgampaya

Department of Engineering Mathematics
Faculty of Engineering, University of Peradeniya
Sri Lanka
e-mail: ckw@pdn.ac.lk

Abstract. Today, mobile advertising is considered as the most effective medium to convey promotional messages to customers because of the excessive usage of mobile phones and tablets all around the world. However, this ecosystem has severely been affected by fraudulent activities due to a large sum of money circulated in the advertising industry. The term ad fraud is referred to as any kind of fraudulent activities that are executed by fraudulent users either a human or an automated script. The combat between researchers and fraudulent users never ends because more smarter strategies are being used by the fraudsters to bypass the significant number of detection and prevention solutions. The Real Time Mobile Ad Investigator-RTMAI is proposed as a software solution to address this problem where a novel supervised learning algorithm based on the hidden Markov model along with a rule engine have been proposed to classify fraudulent impressions in real time. Furthermore, RTMAI proposed a solution to address the class imbalance problem which is generic to most of the classification datasets. The experimental results show the significance of the proposed approach to classify the fraud or non-fraud clicks/events, impressions and even user sessions more confidently in real time.

Keywords: Mobile advertising, click fraud, classification, supervised learning, class imbalance, hidden Markov model

1 INTRODUCTION

The inception of online advertising goes to the early nineties where the very first online advertisement, which was a banner advertisement of a web magazine named HotWired, had been published in 1994 on a web page owned by AT & T [1]. The Mobile Marketing Association has defined the term Mobile Advertising as *a form of advertising that transmits advertisement messages to users via mobile phones or other wireless communication devices* [2]. The recent statistics show that 33 % of the world population has been using smartphones, which is nearly more than 2.5 billion users [3]. Significant increase of mobile Internet browsing in recent years has led to an increase in the popularity of advertising in mobile devices. The analysis states that currently 51 % of digital advertising market share will be dominated by mobile advertising and predicted to be 70 % by 2019 with the worth of US$ 200 billions [4].

The key contributors of this market are *User, Advertisers, Publishers,* and *Advertising Networks known* as *Ad networks*. Users are individuals who surf websites or use mobile apps where they see the ads shown on web pages or within the apps that they may click on the ads. Advertiser is the one who designs the ads and makes a contract with an ad network to publish advertisements on behalf of himself or a company [5]. Publisher can be a web site or mobile application which displays the advertisements to the site/app visitors [5]. Generally, Large publishers often sell around 60 % of their ad space known as ad inventories though ad networks and smaller ones sell their entire inventories [6]. Ad network are online companies which play a broker role between advertiser and publisher. Advertisers are charged by the ad networks for publishing their advertisements. Ad networks find suitable publishers to display ads [6].

The most popular revenue models in this industry are *Pay-Per-Impression (PPI)* and *Pay-Per-Click (PPC)* models where internet content providers are paid by the advertiser per each impression or click [7, 8, 9] An impression is defined as the displaying or loading event of advertisement into an advertisement frame while click would be any kind of user interaction/event on the displayed advertisement.

In this study, we propose a software solution called real Time Mobile Ad Investigator (RTMAI). The RTMAI has been implemented by encapsulating the Rule Engine, a Behavioural Pattern Recognition module and a novel supervised machine learning model. These modules interact with each other simultaneously to achieve the final goal of classifying clicks/events, impressions and subsequently user sessions in real time. The remainder of this paper is organised as follows. In Section 2 discuss existing detection and prevention systems. Proposed approach of this study is discussed in detail under Section 3. The experimental results are available in Section 4. Conclusion is given in Section 5.

2 LITERATURE SURVEY

The researchers have proposed and implemented a number of click or impression fraud detection tools using distinct methodologies. Xu et al. [10] have developed a detection system where they used a stepwise evaluation process including proactive functionality test at front end (i.e. user interface) backed by JavaScript and passive examination of browsing behaviour to differentiate a clickbots from a human clicker. DECAF [11] proposed an offline click fraud detection approach using rule-based methods to detect placement fraud by analysing the advertisement user interface status in mobile apps/pages. NAB (Not-A-Bot) [12] is a system that enables a range of verifier policies for applications that would like to separate human-generated requests from bot traffic. NAB approximately identifies and certifies human-generated activities. Client machines of the NAB system attest the legitimacy of individual requests to remote parties with a trusted component that monitors keyboard and mouse input. FCFraud is an operating system anti-malware service proposed by Iqbal et al. [13] inspects HTTP packets from all user processes and analyzes the ad-related traffic from captured HTTP packets. The FCFraud detects malware that is a part of a botnet and launches attacks using technologies similar to the desktop malware.

MAdFraud [14] studies mobile ad fraud perpetrated by Android apps and identifies two kinds of fraudulent behavior. First one is requesting ads in the background and the second one is clicking on ads without user interaction. They further developed an analysis tool to automatically trigger and expose ad fraud in Android emulators. However, the intrinsic limitation of offline testing on coverage and the lack of a reliable way to distinguish benign from fraud ads make it hard for such approaches to detecting sophisticated means of doing frauds, especially bot-driven frauds. AdAttester [15] tries to detect and prevent well-known ad fraud by identifying incoming click or impression is actually delivered by a real user with two primitives called verifiable display and unforgeable clicks. AdAttester cannot detect mobile ads that violate the ad policy of the ad provider. Walgampaya et al. [16] has used a multi-level data fusion process to detect and prevent click fraud in real time and decision. Agarwal [17] also suggested a real time click fraud detection technique which mainly focuses on the advisor side. Either approach has used the same mathematical theory called the Dempster-Shafer evidence theory to tackle fraudulent clicks in desktop environments.

Several machine learning (ML) approaches have also been experimented by researchers to improve the accuracy, performance and reliability of fraudulent clicks and impressions detection mechanisms in mobile advertising. Perera et al. [18] evaluated a number of ML algorithms such as decision trees, regression trees, artificial neural networks and support vector machines on real data produced by Buzzcity Ltd. The researchers were able to identify a number of different fraudulent patterns in the data set but did not focus on detecting each individual event called impression. Haider et al. [4] has discussed another ML based approach which is similar to previous authors' approach but these authors were able to achieve better improve-

ment with identifying each individual fraud impression than the common pattern of fraudulent events. Botnets detection approach was proposed by Gobel [19] using the hidden Markov model in their study. The proposed approach has used network traffic generated by computers to model the HMMs so that bots can be identified through measuring the distances between these HMMs.

3 REAL TIME MOBILE AD INVESTIGATOR-RTMAI

Real Time Mobile Ad Investigator – RTMAI is an extended version of Real Time Mobile Bot Miner – RTMBM [5] to address click and impression fraud problems in the pay-per-click internet advertising model. The proposed solution is capable enough to integrate not only in mobile but also in desktop environments. The RTMAI is a composition of rule engine, behavioural pattern recognition techniques [5], advanced algorithms, novel machine learning and resampling algorithms to identify fake user events/clicks, impressions and subsequently user sessions in real time.

3.1 System Architecture

The abstract view of RTMAI architecture is graphically represented in Figure 1. There are four modules named Data Collection, Data Processing, Decision Making and Admin modules that incorporate each other to build the overall system. The RTMAI has neither functionality nor implementation change made to Data Collection or Data Processing modules compared to RTMBM [5]. However, a new Admin Module has been implemented in RTMAI while enhancing the capability of Decision Making module with novel supervised learning algorithm.

RTMAI uses two approaches to make a decision on whether the use event or session is valid or fraud. Rule engine is implemented with two subsystems called Horizontal and Vertical Analysis Sub System as the first approach. Secondly, a novel supervised learning algorithm has been implemented based on the hidden Markov model to classify each individual click or impression into either fraud or non fraud category in real time. The proposed model is enhanced with another new sequence data resampling technique to solve the class imbalanced problem in the dataset.

The admin module is the place where all the admin tasks are performed. Session handler is responsible to manage session validation through heart-beat and scheduler algorithms [5]. Report handler is the admin dashboard to visualize all the analytics results along with user session and event details.

3.2 Rule Engine

The RTMAI is capable of identifying user events via a data filtering process which is implemented with novel advanced algorithms. The front end java script consists of all the business logics to identify advanced user events such as Touch Zoom Event (TZE) over Swap Touch Zoom Event (STZE) where TZE event is triggered when

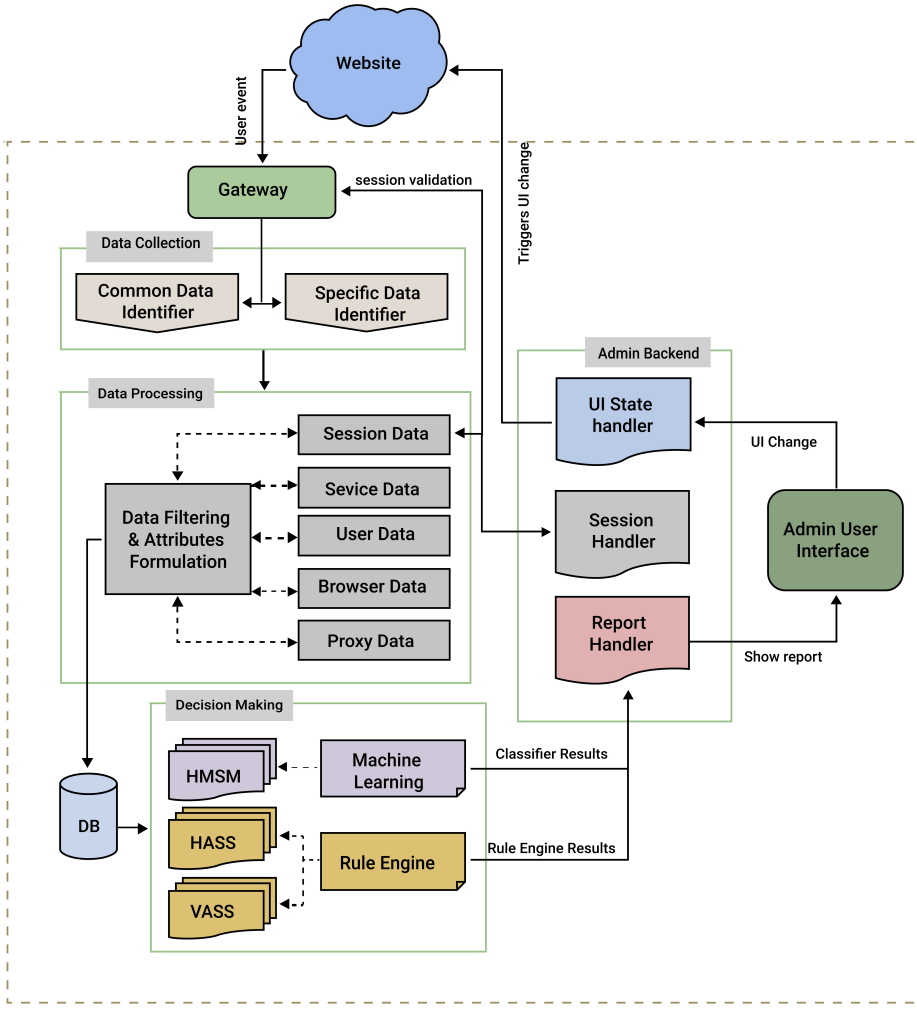


Figure 1. System architecture

user clicks on mobile screen twice in order to zoom HTML component. The STZE is identified when a particular user does the HTML component zoom using his/her finger tips rather perform two clicks during a tiny time period. Such tricky user events are captured by the RTMAI very accurately. The decision making module has a rule engine where pre-processed data are being analysed to label as fraud or not. A set of static rules have been defined under Horizontal and Vertical analysis sub systems [5].

3.3 Hidden Markov Scoring Model-HMSM

The Hidden Markov Scoring Model – HMSM is a novel supervised learning classification algorithm which is implemented as a part of the decision making module. The HMSM is based on the scoring approach of HMM rather than conventional probabilistic models of HMM. The proposed methodology discusses an N to N process from feature selection to target state classification via fully automated process. The initial version of the HMSM algorithm was evaluated based on a labeled dataset borrowed from Haider et al. [4] before production implementation.

3.3.1 Dataset

The dataset contains a number of attributes such as deliveryId, timestamp, clientIp, marketId, adSpaceId, accoundId, siteId, unknownDeviceId, clientVersionId, ipMarketId, ipCountyCode, ipIsp, adRelType, forcedAd, eventType, eventId, eventTimeStamp and status, etc. Details of the attributes can be found in [4].

3.3.2 Derived Attributes

Derived attributes were introduced to the dataset out of existing variables with respect to individual impression so that dimension of the feature vector will be reduced. eventCount, distEventTypes, surfTimeSec are some of the derived attributes:

- *eventCount* is the number of triggered events per impression,
- *distEventTypes* is the number of distinct event types of a given impression,
- *surfTimeSec* is the number of seconds users engage with an impression,
- *maxEventCount* is the maximum event count out of *distEventTypes*,
- *distEventFreqGroups* is the number of distinct event frequency groups,
- *dayOfWeek* is an impression triggered date.

Finally, the dataset arranged in ascending order of the timestamp variable to guarantee the state transition from previous state to the next state which is a fundamental nature of HMM.

3.3.3 Feature Vector

HMM basically interacts with state transition. All the numerical variables were transformed into categorical variables through entropy-based binning technique based on the target variable. Once numerical variables transformations are completed, all the categorical variables were evaluated with chi-square test of independence against the target variable with a significance level of 0.05. The variables which have higher chi-square test statistics than critical value were identified as the observe or emission variables.

Altogether nine variables were identified by the chi-square test which have higher test statistics than critical value. Out of these nine variables, five were derived attributes and the rest of them were raw attributes. The selected feature vector is fed into the proposed hidden Markov scoring model as emission variables.

3.3.4 HMSM Algorithm Implementation

The HMSM algorithm calculates scores for each target class based on observe variables to classify the data point in supervised learning approach. In a supervised learning approach, a model should be trained first and then makes the predictions with trained parameters. Calculation of A, B and π is referred to as training the model in HMSM. Equations (1), (2) and (3) were used to calculate the A, B and π with the training data set. The mathematical representation of the hidden Markov model is defined by A, B, π and can be denoted by λ [20] where $\lambda = (A, B, \pi)$.

$$A_{(i,j)} = p\left(\frac{S_t = j}{S_{t-1} = i}\right), \quad \forall_i = 1, \ldots, M, \sum_{i=1}^{M} A_{(i,j)} = 1, \tag{1}$$

$$B_{(j,k)} = p\left(\frac{O_t = k}{S_t = j}\right), \tag{2}$$

$$\pi_i = p(S_1 = i), \quad \forall_i = 1, \ldots, M, \sum_{i=1}^{M} \pi_i = 1, \tag{3}$$

where,

- M = total number of hidden states,
- $i, j = 1, \ldots, M$ index the state,
- k = number of possible discrete observations,
- s = hidden state,
- s_t = hidden state at time t,
- π = initial state distribution, a vector of size M,
- A = state transition probability matrix, a matrix of size $M * M$,
- B = emission probability matrix, a matrix of size $M * K$,
- x_t = observation at time t.

HMSM algorithm classifies the test data with a scoring model based on HMM. Target variable of the experimental data set has two states called OK and Fraud, where OK being the click is genuine and the Fraud being that the click is not genuine. The HMSM classifies each individual record in test data into either state.

Feature Score ($\mathbf{fscore_{f,s}}$): Calculates scores towards each hidden state (i.e. Fraud or OK) of the target variable called $fscore_{Fraud}$ and $fscore_{OK}$ for each individual

record in test dataset with respect to each emission feature. Equation (4) is defined as the mathematical representation of fscore:

$$fscore_{f,s=i} = -\left\{\log \pi_i + \log A_{i,s} + \log B_{s,k}\right\}. \tag{4}$$

Mean Deviation Feature Score ($\mathbf{mdfscore_{f,s}}$): Equation (5) calculates the mean scores for each subset (i.e. $fscore_{Fraud}$ and $fscore_{OK}$) in order to calculate the deviation of the feature score from its subset mean represented in Equation (6):

$$\mu_{s=i} = mean\left(fscore_{s=i}\right), \tag{5}$$

$$mdfscore_{f,s=i} = fscore_{f,s=i} - \mu_{s=i}, \tag{6}$$

$$\forall_i = \left\{Fraud, OK\right\}, \forall_f = 1, \ldots, F.$$

Minimum Mean Deviation Score ($\mathbf{mdscore_{min,s=i}}$): Select minimum mean deviation score out of *mdfscore* for each hidden status as in Equation (7):

$$mdscore_{min,s=i} = \min\left(mdfscore_{s=i}\right). \tag{7}$$

HMSM identifies the most probable hidden state of a given record as the state of the maximum $mdscore_{min,s=i}$, as shown in Equation (8):

$$hiddenState = \max\left(mdscore_{min,OK}, mdscore_{min,Fraud}\right) \tag{8}$$

where s = hidden states and F = number of observe features.

3.4 Step-Factor Resampling and Smoothing Technique

Any kind of dataset that is related to fraud detection including the click fraud suffers with Class Imbalance problem [21, 22]. The class imbalance occurs when the dataset carries only a small number of data points representing the minor class compared to the major class in a particular dataset [23]. A novel methodology is introduced to solve this class imbalance problem in a sequence data via a resampling technique and smoothing approach.

3.4.1 Smoothing Technique

One of the major smoothing techniques called Additive smoothing is used in this study to upturn the zero probabilities for unseen data using ε as a tuning parameter since it provides better estimates compared to the other methods. Here, ε is referred to as the smoothing factor in Additive smoothing. Estimated probabilities p_ε based on the actual counts can be calculated as in Equation (9) for each value x

of a variable X in a sample of N observations [24]:

$$p_\varepsilon = \frac{(x + \varepsilon)}{(N + \varepsilon * N_x)} \tag{9}$$

where N_x is the number of possible values contained in the sample space.

3.4.2 Resampling Technique

The proposed resampling technique identifies the major and minor classes from a dataset and then calculates the optimum number of records to be converted so called *Conversion Factor* (Z) from major class into minor class in order to balance the dataset. The derivation of the proposed resampling technique is illustrated as follows:

- C^i_{major} – number of major class instances before resample,
- C^i_{minor} – number of minor class instances before resample,
- C^f_{major} – number of major class instances after resample,
- C^f_{minor} – number of minor class instances after resample,
- Z – number of instances to be converted from major to minor class,

$$R_i = \frac{C^i_{major}}{C^i_{minor}}, \tag{10}$$

$$R_f = \frac{C^f_{major}}{C^f_{minor}}. \tag{11}$$

Since $C^f_{major} < C^i_{major}$ and $C^f_{minor} > C^i_{minor}$, the relationship among the major and minor classes before and after the resampling can be expressed with Z, as shown in Equations (12) and (13):

$$C^f_{major} = C^i_{major} - Z, \tag{12}$$

$$C^f_{minor} = C^i_{minor} + Z. \tag{13}$$

The Equation (11) is rearranged according to the Equations (12) and (13) so that Equation (14) is derived:

$$R_f = \frac{C^i_{major} - Z}{C^i_{minor} + Z}. \tag{14}$$

The formula to calculate conversion factor – Z is derived with Equations (10) and (14):

$$Z = \left(\frac{R_i - R_f}{R_f + 1}\right) C^i_{minor}. \tag{15}$$

The R_f is considered as a tuning parameter to the algorithm where different values from 1 to $R_i - R_f = 1$ are assigned to R_f so that a set of Z values can be calculated according to the Equation (15).

Once the conversion factors are calculated, the respective number of records should be converted from major to minor class in order to overcome the class imbalance problem. There should be a consistent procedure to perform this conversion. The proposed method calculates the index distance of adjacent minor class instances and arranges them in three different ways called ASC, DESC and MID to perform the conversion so that the consistency is guaranteed. The term Step-Factor is used to refer to these minor class instances arrangement methods. The target class of the dataset contains two states called OK and Fraud where OK is genuine and Fraud means not genuine instances. The target class sequence is arranged as a list so that the index distance of adjacent minor class instances can be calculated. The graphical representation of this process is shown in Figure 2.

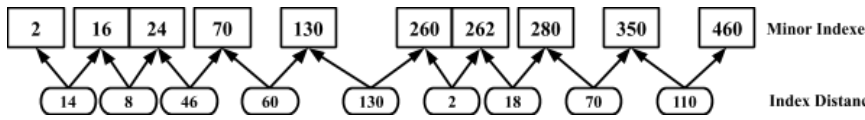


Figure 2. Index distance of minor class instances

The implementation of the first two step-factors are straightforward. The index distances are arranged in ascending order in ASC and descending order in DESC method. Merge sort algorithm is used to perform these said sorting because the worst case time complexity of the merge sort is $O(n \log n)$ which enhances the efficiency of step-factor algorithm. The index distances are arranged by the MID step-factor following a shuffling mechanism. The MID step-factor algorithm first calculates the number of index distances available in the list. If the number of indexes is odd, the last index value of the list is omitted and then the index distance list so called initial index distance list is divided into two sublists named left and right index distance sub lists. Then the shuffling starts at the last index value of the left index distance sub list and then first index value of the right index distance sub list and so on. The algorithm performs conversion of major to minor class instance with respect to selected step-factor. The shuffling process of the indexes based on the MID step-factor is illustrated in Figure 3.

4 EXPERIMENTAL RESULTS

The Rule engine was evaluated with real world data which have been gathered from a web based application. The HMSM and its enhancement with resampling and smoothing technique was tested with labeled dataset [4] before integrating with the current version of RTMAI in production.

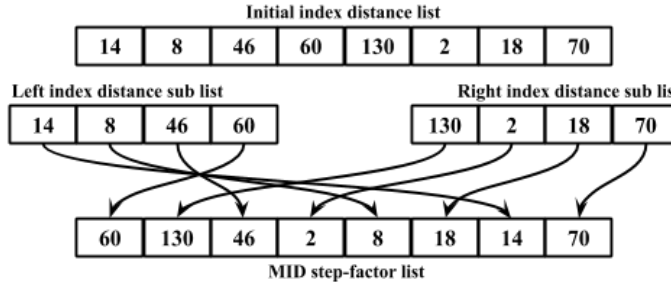


Figure 3. MID step-factor index shuffling process

4.1 Rule Engine

We have tested RTMAI with automatically generated scripts which are implemented with selenium web driver to imitate real user behaviuor and mobile emulators such as android and web browser emulators.

4.1.1 Horizontal Analysis Sub System – HASS

Table 1 shows the experimental results of a user session with respect to the horizontal analysis subsystem in the decision making module where each individual user event is analysed. The user has triggered 6 events from a desktop device and out of those 6 events three were identified as mobile events. The rule engine applies static rules on top of this information and then identifies that this user session has been created by an automated script and flagged respective events as fraudulent events. The particular user tried to simulate mobile user behaviour in a desktop but RTMAI smartly identified its fraudulent activities.

4.1.2 Vertical Analysis Sub System – VASS

Experimental results of vertical analysis or user session analysis categorised into three segments called device, user and event data analysis. Figure 4 is a real screen-shot of VASS analysis dashboard view. *Device Analysis* graph shows that both mobile and desktop events have been triggered in this particular user session. Then the device behaviour can be fraud to some extent.

The attribute values in user analysis must be constant and cannot be changed for a given session. If there is a variance of an attribute, the system identifies it as a negative behaviour. The experimental results in the *User Analysis* graph shows that there are deviated behaviours in time zone, event sequence and user location. This user is trying to hide his or her real geolocation by altering time zone offset and location details. Meanwhile, there is a mismatch in the event sequence as well.

Final analysing aspect of the vertical analysis sub system is event behaviour analysis where the experimental results of a particular user events are shown in the

Event	Tag	EventTime	FingerTips	Event (X, Y)	TimeZone	ZoneTime	Device	Device (W, H)	View (W, H)	View	Browser	Proxies	Status
TS	DIV	20:17:50.208	1	1 152, 289	+05:30	20:17:52.926	Desktop	1 366, 768	1 366, 447	−1	Chrome	1	Fraud
TZE	H3	20:17:49.83	1	982, 216	+05:30	20:17:51.797	Desktop	1 366, 768	1 366, 447	−1	Chrome	1	Fraud
TS	H3	20:17:48.708	1	540, 205	+05:30	20:17:51.441	Desktop	1 366, 768	1 366, 447	−1	Chrome	1	Fraud
SE	−1	20:16:59.744	−1	0, 0	+05:30	20:17:02.493	Desktop	1 366, 768	1 366, 447	−1	Chrome	1	OK
LC	DIV	20:16:59.447	1	438, 345	+05:30	20:17:02.16	Desktop	1 366, 768	1 366, 447	−1	Chrome	1	OK
SE	−1	20:16:33.931	−1	0, 0	+05:30	20:16:37.336	Desktop	1 366, 768	1 366, 447	−1	Chrome	1	OK

Table 1. Horizontal analysis

Figure 4. Dashboard screenshot in VASS

Event Behaviour Analysis graph. The user tries to pretend as a mobile user but a small number of desktop events have been captured by the RTMAI. There are 6 touch start events (40 % out of total events) but there is less number of complex mobile events such as touch zoom event (only 1), swap zoom event (zero event). The real reason is, it is difficult to automate more advanced mobile events by scripts. Comparing the probability of user events RTMAI can make appropriate decisions.

4.2 Hidden Markov Scoring Model – HMSM

The HMSM was evaluated with 20 different test samples in order to verify the performance of the model across the dataset. An individual train/test sample is represented by S-xx notation as shown in Figure 5 where the samples are equally distributed throughout the dataset so that consistency of the classifier can be evaluated by calculating accuracy, precision, recall, specificity and F-score for each individual sample as the performance measures.

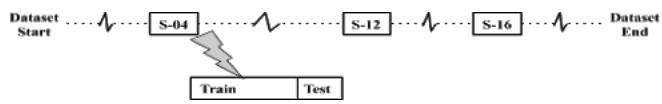


Figure 5. Train/test samples distribution

The fundamental problem of any machine learning algorithm is finding optimum training set size to perform regression or classification tasks accurately. To solve this issue HMSM trains with 10 distinct training samples where sample size starts from 3 000 records to 25 000 records to find the optimum training sample size. Each training set evaluated with 20 different test samples and calculated the mean value and mean of the standard deviation of performance measures to identify the optimum training set record size.

The HMSM model performs well in classifying test data when training sample size is 5 000. Figure 6 illustrates the model performance with all training data sets where mean values for accuracy, recall, precision and F-score have been reached to above 80 % and specificity reached to 79 % at the sample size containing 5 000 records. The stability of the model can be evaluated with the mean values of standard deviations of each training sample. The results show that the model is more stable when the training data set has 5 000 records where standard deviations of all the performance measures are less than 0.25 including specificity.

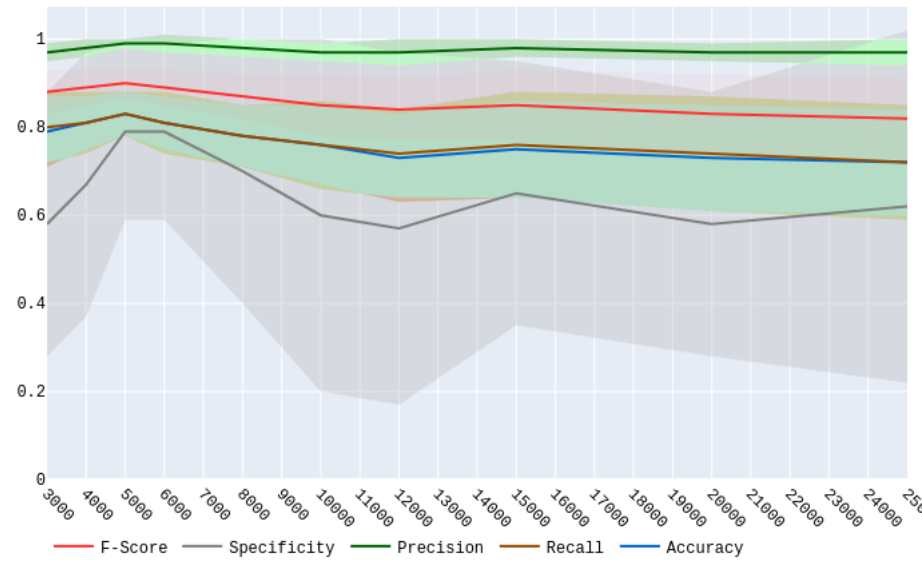


Figure 6. Training set evaluation by mean and standard deviation

The observe variables which have initially been identified with the chi-square test of independence were categorised into three groups.

Derived features: all derived features,

Raw features: all initial features without any prepossessing,

Combined features: all derived and raw features.

The proposed HMSM classifier tested with a random test sample containing 50 data points based on each feature group so that the best performing feature vector can be identified. The experimental results are visualised in Figure 7 where the x axis represents the data point and the y axis represents the *Minimum Mean Deviation Score Difference: (mmdsd)* of each data point, as shown in Equation (16).

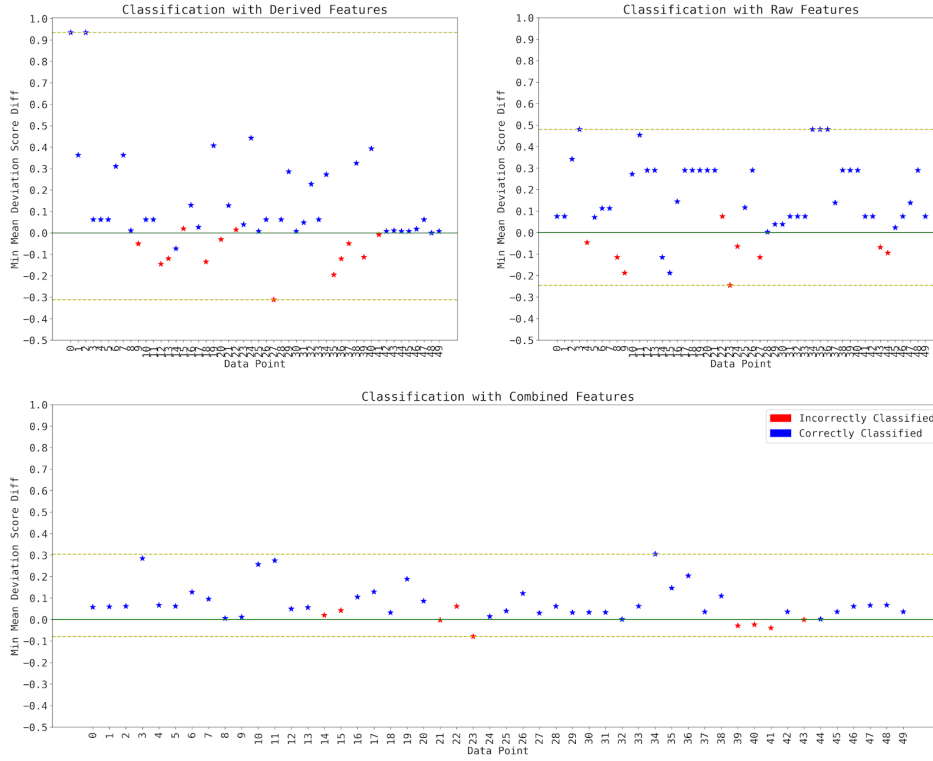


Figure 7. Classifier behaviour with types of features

$$mmdsd = mdscore_{min,OK} - mdscore_{min,Fraud}. \tag{16}$$

4.3 Step-Factor Resampling and Smoothing Technique

The HMSM was enhanced with smoothing factor – ε as a tuning parameter to the model in order to scale the performance of the model. The major advantage of smoothing factor is that it ensures the first priority objective of HMSM by increasing specificity while keeping almost constant values for all other performance measures. The experimental results in Figure 8 show that there is a significant effect on the model performance and stability for all training sets with the smoothing factor.

Moreover, it is clear that specificity is much more sensitive to the smoothing factor than all the other performance measures where a slight change in the smoothing factor produces significant performance change in specificity.

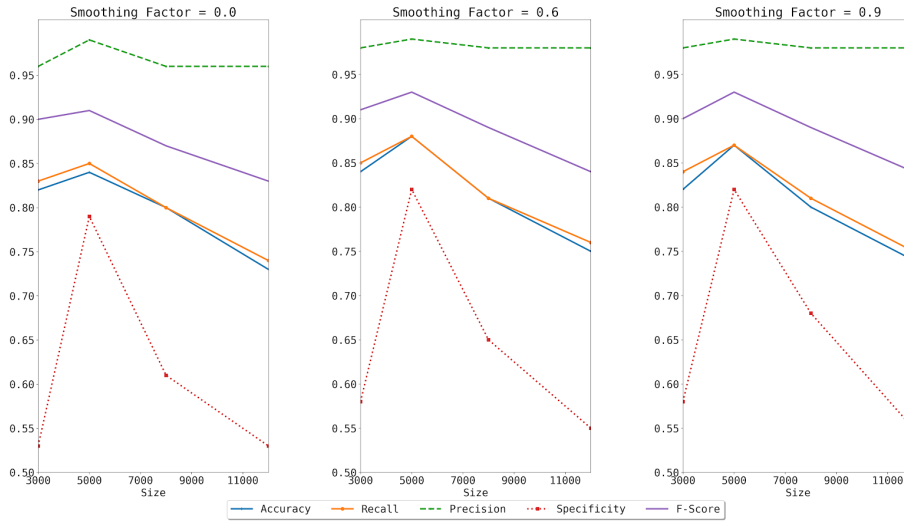


Figure 8. Model performance with smoothing factor

The proposed resampling technique is applied to sample datasets in order to experiment the performance enhancement of HMSM. Different datasets are validated against each step-factor to identify the effectiveness of each individual step-factor in this resampling approach. The model trains with the same training data set sizes that are used in smoothing factor to compare the performance variation between the two approaches. The experimental results in Table 2 shows the association of step-factor resampling approach towards a sequence data set. The model performs better when training data with 5 000 records as in smoothing factor. The experimental results illustrate that all step-factors are almost equally contributed to the model performance along with each training data set size.

The step-factor resampling technique has achieved comparatively greater statistics for each performance measure than smoothing with each training record size. Moreover, drastic changes in performance measures can not be experienced with respect to training record size in this approach like in smoothing approach. For instance, the statistics in specificity varies from 55 % to 82 % with the range of 17 in smoothing but that of resampling is from 75 % to 84 % where the range has been reduced to 9 % by step-factor resampling.

The resampling technique was applied to all 20 sample training datasets and then evaluated the HMSM model with the smoothing factor as 0.06. The experimental

Records	Step Factor	Acurracy		Recall		Precision		Specificity		F-Score	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
3 000	ASE	0.86	0.06	0.87	0.06	0.99	0.02	0.77	0.29	0.92	0.04
	MID	0.86	0.06	0.87	0.06	0.98	0.02	0.65	0.36	0.92	0.03
	DESC	0.85	0.08	0.86	0.08	0.99	0.02	0.68	0.36	0.92	0.05
5 000	ASE	0.88	0.04	0.89	0.05	0.99	0.02	0.82	0.24	0.94	0.03
	MID	0.89	0.03	0.89	0.04	0.99	0.02	0.8	0.27	0.94	0.02
	DESC	0.89	0.03	0.90	0.03	0.99	0.02	0.84	0.26	0.94	0.02
8 000	ASE	0.84	0.10	0.84	0.10	0.99	0.02	0.79	0.30	0.90	0.07
	MID	0.84	0.07	0.84	0.07	0.99	0.02	0.74	0.29	0.91	0.05
	DESC	0.84	0.05	0.84	0.06	0.99	0.02	0.73	0.31	0.91	0.03
12 000	ASE	0.79	0.14	0.79	0.14	0.99	0.02	0.84	0.26	0.87	0.10
	MID	0.80	0.11	0.80	0.11	0.99	0.02	0.86	0.23	0.88	0.07
	DESC	0.81	0.11	0.81	0.11	0.98	0.02	0.75	0.29	0.89	0.08

Table 2. Model performance with step-factor resampling

results show that step-factor plays a vital role in classifying test data. Each step-factor equally contributes to enhance the performance of the model, as shown in Figure 9, where 7 samples with DESC, 8 samples with ASC and 5 samples with MID step-factor has enhanced the performance of the model. Moreover, 17 out of 20, as a percentage of 85 %, samples show highest performance when the sample size equals or is less than 6 000 records. This approach including smoothing and resampling has been ensured the most valuable capability of HMSM to classify sequence data with small training dataset. The maximum percentage of conversion factor -Z out of the training size is 25 % and the rest of the samples are below 20 %. Furthermore, 16 samples have been reached to the optimum model performance when the conversion factor is below 10 %. There are only three samples which have been reached to 1 000 records to train the model in order to absorb better classification accuracy but out of these three, only one sample has a higher conversion rate.

There are four models for HMSM. The first model is identified as the initial model where neither smoothing factor nor resampling technique is applied. The second and third models are incorporated with smoothing factor and step-factor resampling technique respectively. The final model enhanced with both smoothing and step-factor resampling approaches to evaluate the HMSM model. The individual performance of both smoothing and resampling models are almost equal but have gained better classification optimisation compared with the initial model. A significant improvement of all performance measures can be identified after applying smoothing factor and resampling technique together as a hybrid approach on HMSM. The key objective of any fraud detection approach is to increase the specificity while keeping the statistics of all other performance measures in satisfactory level. The HMSM has achieved not only said objective with a dramatic increase of the specificity from 79 % to 91 %, but also other performance measures with almost

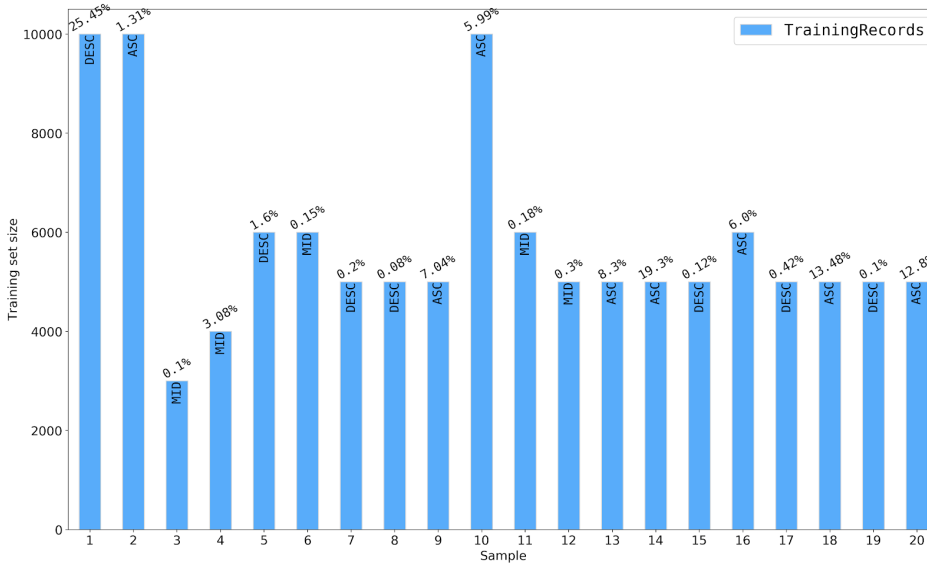


Figure 9. Best performing training set size along with step-factor

the same quantity. The accuracy and recall have been improved by 10 %, F-score by 6 % and precision has remained the same throughout the initial to final model with a tiny standard deviation change since it has already reached a higher statistics of 99 %. The consistency and stability of the model has also been guaranteed by the diminishing rate of standard deviation of each performance measure from initial model to final model. The experimental results for each individual model (Inital: Model-1, Smoothing: Model-2, Resampling: Model-3, Resampling and Smoothing: Model-4) have been summarised in Table 3.

	Model-1		Model-2		Model-3		Model-4	
	μ	σ	μ	σ	μ	σ	μ	σ
Accuracy	0.84	0.05	0.88	0.04	0.89	0.03	0.94	0.03
Recall	0.85	0.05	0.88	0.04	0.90	0.03	0.95	0.03
Precision	0.99	0.01	0.99	0.01	0.99	0.02	0.99	0.02
Specificity	0.79	0.26	0.82	0.23	0.84	0.26	0.91	0.17
F-Score	0.91	0.03	0.93	0.02	0.94	0.02	0.97	0.01

Table 3. Performance evaluation of the models

Experimental results show that HMSM model performs better after enhancing the model with smoothing factor and resampling technique. Figure 10 illustrates the HMSM performance across 20 different samples after applying the resampling and smoothing factor. There are only four instances with specificity less than 70 % and the lowest is 50 %. All other performance measures are above 90 % for all sam-

ples. Moreover, Figure 11 illustrates the performance of the all four models with the Receiver operating characteristic (ROC) curve to get a much better understanding.

Figure 10. Best performing model

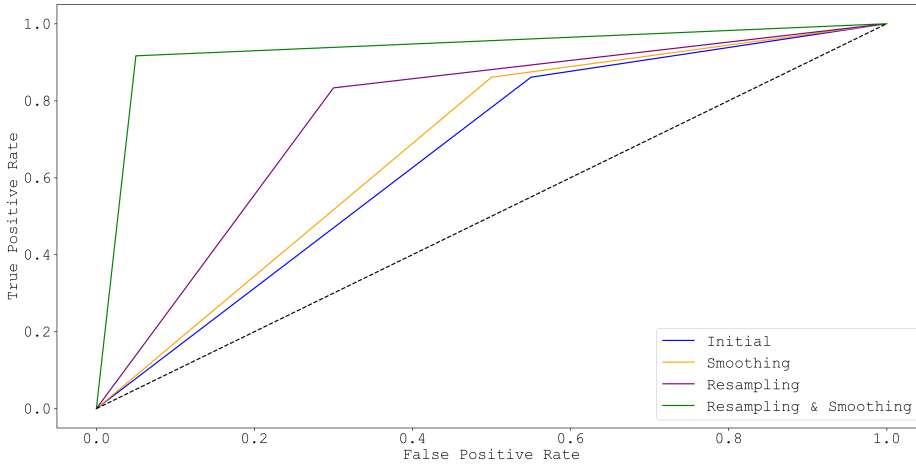


Figure 11. Models performance

5 CONCLUSION

The RTMAI is smart enough to identify fraud user events/clicks and sessions with a rule engine where HASS and VASS analyse each individual user event and session accordingly. Since RTMAI is a extended version of RTMBM, it encapsulated behavioural pattern recognition capability that facilitates more accurately second level verification on top of the VASS for user session identification. The Hidden Markov Scoring Model (HMSM), a novel supervised learning algorithm was introduced to the RTMAI to classify fraudulent impressions in addition to the classification of fraudulent clicks and subsequently user sessions with minimal computational power. The HMSM guarantees high performance with minimum training data in a smaller amount of time. A new resampling technique is proposed to address the class imbalance problem in sequence data. The findings of this study prove that proposed resampling and smoothing techniques perform well with the model in sequence data. There are a number of interesting features in this resampling technique. No synthetic data are introduced to the dataset. Natural patterns and the characteristics of the initial dataset are narrowly affected because the number of conversions from major to minor instances is very low. Altogether, the HMSM is a good alternative algorithm in supervised learning which reduces the computation time in training along with higher performance and higher learning efficiency on unseen data. The proposed model can be identified as a stable classification algorithm because it performs really well by identifying both positive and negative classes in higher performance measures. The optimum model shows significant capability as a fraudulent impression classifier with an average accuracy of 94 %, average precision of 99 %, average recall of 95 %, average specificity of 91 % and average F-score of 97 % across 20 different test samples with the standard deviation of 0.03, 0.02, 0.03, 0.17 and 0.01, respectively. Thus, this proposed RTMAI is a composition of several advanced techniques to solve almost all the dimensions of this click fraud detection domain.

REFERENCES

[1] Evans, D. S.: The Online Advertising Industry: Economics, Evolution, and Privacy. Journal of Economic Perspectives, Vol. 23, 2009, No. 3, pp. 37–60, doi: 10.1257/jep.23.3.37.

[2] Martins, J.—Costa, C.—Oliveira, T.—Gonçalves, R.—Branco, F.: How Smartphone Advertising Influences Consumers' Purchase Intention. Journal of Business Research, Vol. 94, 2019, pp. 378–387, doi: 10.1016/j.jbusres.2017.12.047.

[3] Tao, K.—Edmunds, P.: Mobile APPs and Global Markets. Theoretical Economics Letters, Vol. 8, 2018, No. 8, pp. 1510–1524, doi: 10.4236/tel.2018.88097.

[4] Haider, C. M. R.—Iqbal, A.—Rahman, A. H.—Rahman, M. S.: An Ensemble Learning Based Approach for Impression Fraud Detection in Mobile Advertising. Journal of Network and Computer Applications, Vol. 112, 2018, No. 15, pp. 126–141, doi: 10.1016/j.jnca.2018.02.021.

[5] Aberathne, I.—Walgampaya, C.: Smart Mobile Bot Detection Through Behavioral Analysis. In: Kolhe, M., Trivedi, M., Tiwari, S., Singh, V. (Eds.): Advances in Data and Information Sciences. Springer, Singapore, Lecture Notes in Networks and Systems, Vol. 38, 2018, pp. 241–252, doi: 10.1007/978-981-10-8360-0_23.

[6] Fridgeirsdottir, K.—Najafi-Asadolahi, S.: Cost-Per-Impression Pricing for Display Advertising. Operations Research, Vol. 66, 2018, No. 3, pp. 653–672, doi: 10.1287/opre.2017.1697.

[7] Alrwais, S. A.—Gerber, A.—Dunn, C. W.—Spatscheck, O.—Gupta, M.—Osterweil, E.: Dissecting Ghost Clicks: Ad Fraud via Misdirected Human Clicks. Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12), 2012, pp. 21–30, doi: 10.1145/2420950.2420954.

[8] Hu, Y.—Shin, J.—Tang, Z.: Performance-Based Pricing Models in Online Advertising: Cost Per Click Versus Cost Per Action. 2012, Georgia Institute.

[9] Mahdian, M.—Tomak, K.: Pay-Per-Action Model for Online Advertising. In: Deng, X., Graham, F. C. (Eds.): Internet and Network Economics (WINE 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4858, 2007, pp. 549–557, doi: 10.1007/978-3-540-77105-0_59.

[10] Xu, H.—Liu, D.—Koehl, A.—Wang, H.—Stavrou, A.: Click Fraud Detection on the Advertiser Side. In: Kutyłowski, M., Vaidya, J. (Eds.): Computer Security – ESORICS 2014. Springer, Cham, Lecture Notes in Computer Science, Vol. 8713, 2014, pp. 419–438, doi: 10.1007/978-3-319-11212-1_24.

[11] Liu, B.—Nath, S.—Govindan, R.—Liu, J.: DECAF: Detecting and Characterizing Ad Fraud in Mobile Apps. 11th USENIX Symposium on Networked Systems Design and Implementation, 2014, pp. 57–70.

[12] Gummadi, R.—Balakrishnan, H.—Maniatis, P.—Ratnasamy, S.: Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks. Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09), 2009, pp. 307–320.

[13] Iqbal, M. S.—Zulkernine, M.—Jaafar, F.—Gu, Y.: Protecting Internet Users From Becoming Victimized Attackers of Click-Fraud. Journal of Software: Evolution and Process, Vol. 30, 2017, No. 3, Art. No. e1871, doi: 10.1002/smr.1871.

[14] Crussell, J.—Stevens, R.—Chen, H.: MAdFraud: Investigating Ad Fraud in Android Applications. Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '14), 2014, pp. 123–134, doi: 10.1145/2594368.2594391.

[15] Li, W.—Li, H.—Chen, H.—Xia, Y.: AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone. The 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '15), Vol. 1, 2015, pp. 75–88, doi: 10.1145/2742647.2742676.

[16] Walgampaya, C.—Kantardzic, M.—Yampolskiy, R.: Real Time Click Fraud Prevention Using Multi-Level Data Fusion. Proceedings of the World Congress on Engineering and Computer Science (WCECS 2010), Vol. 1, 2010, 6 pp.

[17] Agarwal, A.: Automatic Detection of Click Fraud in Online Advertisements. M.Sc. Thesis, Texas Tech University, 2012.

[18] Perera, K. S.—Neupane, B.—Faisal, M. A.—Aung, Z.—Woon, W. L.: A Novel Ensemble Learning-Based Approach for Click Fraud Detection in Mobile Advertising. In: Prasath, R., Kathirvalavakumar, T. (Eds.): Mining Intelligence and Knowledge Exploration. Springer, Cham, Lecture Notes in Computer Science, Vol. 8284, 2013, pp. 370–382, doi: 10.1007/978-3-319-03844-5_38.

[19] Gobel, W.: Detecting Botnets Using Hidden Markov Models on Network Traces. 2008.

[20] Stamp, M.: A Revealing Introduction to Hidden Markov Models. 2004, pp. 26–56.

[21] Berrar, D.: Random Forests for the Detection of Click Fraud in Online Mobile Advertising. Proceedings of the 1st International Workshop on Fraud Detection in Mobile Advertising, 2012, pp. 1–10.

[22] Blundo, C.—Cimato, S.: SAWM: A Tool for Secure and Authenticated Web Metering. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE '02), 2002, pp. 641–648, doi: 10.1145/568867.568871.

[23] Somasundaram, A.—Reddy, U. S.: Data Imbalance: Effects and Solutions for Classification of Large and Highly Imbalanced Data. 1st International Conference on Research in Engineering, Computers and Technology (ICRECT 2016), 2016, pp. 1–16.

[24] Nivre, J.: Sparse Data and Smoothing in Statistical Part-of-Speech Tagging. Journal of Quantitative Linguistics, Vol. 7, 2000, No. 1, pp. 1–17, doi: 10.1076/0929-6174(200004)07:01;1-3;ft001.

Iroshan Aberathne is Senior Lecturer at the Department of Information and Communication Technology at Faculty of Technology, University of Sri Jayewardenepura, Sri Lanka. He completed his M.Phil. degree in engineering mathematics from Faculty of Engineering, University of Peradeniya, Sri Lanka in 2020 and received his B.Sc. degree with honours in computation and management from the Faculty of Science at the same university in 2015. He has industry and research experience in software engineering and data science in addition to his teaching experience. His research focuses on data science, machine learning, software engineering and computational modelling.

Chamila Walgampaya is Senior Lecturer at the Department of Engineering Mathematics at Faculty of Engineering, University of Peradeniya, Sri Lanka. He earned his B.Sc. in computer engineering with honours in 2001 from the Faculty of Engineering, University of Peradeniya, Sri Lanka and completed his M.Sc. and Ph.D. degrees from the School of Engineering at the University of Louisville, Kentucky, U.S.A. in 2006 and 2011, respectively. He has almost 20 years of extensive and diverse experience as an administrator, computer programmer, researcher and teacher. His research focus lies on click fraud mining, automatic web robots and agents, data and evidence fusion, ensemble methods and machine learning.

Computing and Informatics, Vol. 40, 2021, 628–647, doi: 10.31577/cai_2021_3_628

LEARNING TO TRANSLATE KANNADA AND ENGLISH QUERIES FOR MIXED SCRIPT INFORMATION RETRIEVAL

B. S. Sowmya Lakshmi

Department of Machine Learning
B. M. S. College of Engineering
Bangalore, Karnataka
e-mail: sowmyalakshmibs.mel@bmsce.ac.in

B. R. Shambhavi

Department of Information Science and Engineering
B. M. S. College of Engineering
Bangalore, Karnataka
e-mail: shambhavibr.ise@bmsce.ac.in

Abstract. Due to increase in the availability of numerous languages in the Web, cross language information retrieval is one of the happening issues in the field of natural language processing and information retrieval. Nowadays, people are habituated to combine two or more language words during oral or written discourse. Speakers have also employed intermixing of different languages and scripts in digital media while querying, blogging and on social media platforms. The way of representing two different language words of an utterance in their native scripts is known as mixed scripting. In the present work, we attempted to translate mixed script queries of Kannada and English languages into monolingual queries. We proposed three approaches for translation by constructing bilingual dictionary, word embeddings and Google translate. The proposed method outperforms the conventional dictionary based approach, when word embeddings were combined with the translations learnt from Google Translate and Dictionary.

Keywords: Code mixing, mixed script queries, cross language information retrieval, machine translation

1 INTRODUCTION

1.1 Information Retrieval (IR)

The term "Information Retrieval" was first devised by Calvin Mooers in 1950s. Later on many researchers focused on IR in the mid of 1990s.

According to Manning et al., "Information Retrieval" refers to the technology of "finding material (usually documents) of an unstructured nature (usually text) that satisfies information need from within large collections (usually stored on computers)". The term "material" can be understood in many folds as tweets, videos, music, books, images, documents etc. In this study, we restrict ourselves to text data. Basic terminology behind IR are:

Corpus: A large repository of documents stored on single or multiple computers.

Information need: A topic about which user wants the information, often referred as query.

Relevance: Few of the documents in the corpus might contain topics relevant to information need.

There exist three flavours of IR based of the degree of retrieval as follows.

Web search: WWW is a huge repository of contents which can be searched with the aid of search engines like Google, Bing etc.

Enterprise search: It can also be called Intranet search where search for documents is confined inside a particular organization or company.

Personal search: This search is restricted to one's personal computer where the user search required file stored in his computer. The collection is typically a set of files on a personal computer of the user.

1.2 Cross Language Information Retrieval (CLIR)

As more digital information is made available, the Web continues to be the foremost channel for communication and the largest data repository. Besides the large number of English speaking users, dominance of English on the Web is caused also by the fact that several organizations create English versions of their websites (besides those in their native languages) and of their broad business needs, probably to be widely accessible. Governments around the world also imposed English as a formal language, to some extent, in their educational and governmental spheres. As a result, English was, and still is, the most dominant language for scientific articles, lexicons, dissemination of information and different types of knowledge. However, there exist growth of non-English languages on the Web, as some governments enforce that national corporations and organizations publish some material like people's heritage, geographical data and educational technical material in native languages.

Accordingly, more and more pages on the Web are written in different languages. This resulted in globalized information and a large number of resources that are very much diverse and in a multitude of languages. This feature makes the Web essentially cross-lingual and/or multilingual. But, this linguistic multiplicity and moving towards an international community should no longer be a barrier for accessing information, regardless of its language, on the Web. When users need to search in any language for a particular topic, the search results should no longer be restricted to the native languages of those users. For such users, CLIR provides a solution. In CLIR, users are able to obtain relevant information (document sets) in a language that is different from the language they used in their information need requests (queries). For example, a user may type his/her query in Kannada, a South Indian language, but relevant document sets retrieved are in English or any other language. CLIR system is more complex than traditional monolingual IR system as CLIR also includes a Translation phase.

In query translation approach, query in the source language is translated into the language in which documents are to be retrieved (target language). Machine Translation (MT) is the task of automatically converting the sentences in one natural language into another, preserving the meaning of the input text and producing fluent text in output language. The main objective is to fill up the language gap between two different languages speaking people, communities or countries. The goals of proposed MT system are as follows:

1. In the proposed approach, as input is a Mixed Script query adoption of POS tagging would make the translation process fruitful. Unlike, conventional MT systems, we followed word by word translations by ignoring syntax structure of the respective language. So, if POS of each word in the Mixed Script query is known then translation could be performed in accordance with the POS tags.
2. To develop a bilingual dictionary of Kannada and English Languages.
3. To develop a MT system to translate Kannada – English mixed script queries into monolingual Kannada and English queries.

Handling Indian languages is a challenging task as they differ in morphology and semantic features from English. Even though, the worldwide web is a host to numerous languages, statistics shows that English holds the major share of documents and usage. Which results in creation of Mixed Script space, having documents and queries in single or multiple languages in one or more scripts. IR in Mixed Script space can be called Mixed Script IR (MSIR). MSIR is more challenging than IR as it involves understanding and matching of queries written in two or more scripts with the documents in either of the scripts.

There have been several studies on CLIR including Indian languages. In a CLIR setup, language of the query and retrieved documents are different. MSIR deals with querying in more than one language to retrieve documents in one or more languages. In either case, the documents and the query are written in their native scripts. This

article intends to familiarize the issue of MSIR for Kannada-English mixed query terms. Present state of the art systems are unable to process Mixed Script Queries due to the lack of resources such as transliterated dictionaries and MT systems. Semantic search for Mixed-Script query is still an unsolved problem and it increases in many folds when applied on web search. Adequate tools are not available to process queries having Mixed-Script terms.

The major contributions of this paper are:

- To present the concept, formal definition of MSIR from web for Indian languages, particularly Kannada-English bilingual texts.
- To create a POS tagger for Kannada words.
- To demonstrate how difficult the MSIR problem is and where existing IR techniques fail when applied on such data.

The remaining sections of this paper are arranged as follows. Section 2 describes prior research in this area. Proposed method for translating Mixed Script query is described in Section 3. The results obtained are briefed in Section 4. Section 5 communicates conclusion of the proposed method.

2 LITERATURE SURVEY

Though MSIR has achieved very modest consideration, many laterally correlated tasks like CLIR and transliteration reveals few problems of MSIR. Whereas languages like Chinese and Japanese follow more than one script [1], they might not come across the actual difficulty of the MSIR as they abide by benchmark rules for script writing and spelling. However, this is not true in the case of Indian languages. For instance, in Romanization of Kannada words, there exists no such rules resulting in great number of discrepancies. Furthermore, these Romanized words are combined with English words making difficult to identify transliterated text.

In CLIR queries are translated to the language of the document set. However, out of vocabulary words like Named Entities need to be transliterated rather than translated. There exist no standard rules for mapping alphabets of Indian languages to English or vice versa. This has led to lot of discrepancies in developing a transliteration model [2]. Most of the researchers have highlighted the difficulties in developing transliterated language models for Indian languages in web search [3]. Researchers emphasize on this issue in Hindi Song Search system in Latin script [4, 5]. They focused handling transliterated word pairs matching while crawling song lyrics from various websites from the web. Edit-distance is one among most familiar methods for matching word pairs. Authors in [6] and [7] have followed this method for English-Telugu and Tamil-English language pairs, respectively. Authors in [8] proposed a method to normalize transliterated text using combination-based approach in which a statistical stemmer is used to delete commonly used suffixes along with rules to map spelling variants. An equivalent system that handles both stemming and conversion of grapheme to phoneme was used in [9] to build a standalone search

engine for ten Indian languages. Even though, there are few substantial works present in the field of handling variants and normalization of transliterated text, in practice the process of MSIR is largely ignored.

Gupta et al. [10], analysed query log data of most familiar Bing search engine, to evaluate the significance of their MSIR system. They projected a deep learning paradigm to match mixed- script terms and handle variations in spelling. Method significantly achieved better results when judged with Naive Bayes model by 12 % and 29 % increase in Mean Recall and Mean average precision value.

Pathak et al. [11] attempted to create Automatic Parallel Corpus Creation for Hindi English News Translation Task. Authors developed parallel corpus from comparable corpus crawled from the web from various sources. Quality of the parallel corpus created was analysed by Gestalt Pattern Matching, Hamming Distance and Levenshtein Distance algorithm to calculate sentence matching between Hindi – English sentences. Li et al. [12] developed a Neural Machine Translation (NMT) system, which learns a general network as usual, and then fine-tunes the network for each test sentence. The fine-tune work was done on a small set of the bilingual training data that was obtained through similarity search according to the test sentence. Similarity among sentences were calculated using Levenshtein distance, average word embedding and hidden states of the encoder in NMT measures. Authors observed that performance of Levenshtein distance based similarity was better than other two measures.

Another well known metric used for evaluating machine translation is Evaluation of Translation with Explicit Ordering (METEOR) [13]. Dunđer et al. [14] proposed a machine translation for poetry and a low resource language pair, such as Croatian-German. The authors collected data set that contained the works of a contemporary poet of the Croatian language and the translations of his poems in German. Results were evaluated through BLEU, METEOR, RIBES and Character metrics. An English to Urdu and Hindi translation system was developed using Neural network and translation rules by Khan and Usman [15]. System was evaluated using n-gram, BLEU, METEOR, presion and F-measure scores. METEOR score achieved was 0.7956 for Urdu and 0.8083 for Hindi.

The necessity to recognize and process Indian language scripts is in demand as nearly 50 % of the Indian population use internet daily (according to statistics). Indian Language Technology Proliferation and Deployment Centre (TDIL-DC) has provided phonetic keyboard input is support for all Indian languages. However, POS tagging on Indian languages and especially on Dravidian Languages is quite a difficult task due to the unavailability of annotated data for these languages. Various techniques have been applied for POS tagging in Indian languages. Gadde and Yeleti [16] used morphological features with Hidden Markov Model (HMM) tagger and obtained 92.36 % for Hindi and 91.23 % for Telugu. The Hindi POS tagger used Hindi Treebank 3 of size 450 K. Ekbal and Bandyopadhyay [17] used Support Vector Machine (SVM) for POS tagging in Bengali obtaining 86 % accuracy. The POS tagging in morphologically richer Dravidian Indian languages has always posed a great challenge for researchers. Malayalam is a highly agglutinative language

in the Dravidian family. Sandhi splitting or word segmentation between conjoined words should precede the POS tagging to find word boundaries.

Devadath and Sharma [18] explored the significance of Sandhi splitting on shallow parser and built a POS tagger using Conditional Random Field (CRF). Their POS tagger performed well with 90.45 % accuracy. Antony et al. [19] used SVM with lexicon to obtain 94 % accuracy. A semi-supervised pattern-based bootstrapping technique was implemented by Ganesh et al. [20] to build a Tamil POS Tagger. Their system scored 87.74 % accuracy on 20 000 documents containing 271 K unique words.

Due to the scarcity of quality annotated data very little work has been done on Kannada language. Kannada language has a free form of word arrangement in a sentence which makes POS tagging task for Kannada rigid. Most of the recent works in POS tagging on Kannada have been experimented only with traditional ML techniques like HMM, CRF or SVM. One of such noticeable works was proposed by Shambhavi and Kumar [21]. Authors focused on assignment of POS tags for every word belonging to input Kannada language sentences using machine learning algorithms like second-order HMM and CRF. They have used EMILLE corpus which has 51 269 words as train data, and 2 932 words as test data. Authors were able to achieve accuracies of 79.9 % and 84.58 % for HMM and CRF methods, respectively.

Graves and Schmidhuber [22] proposed a POS tagger for Kannada language by applying CRF with corpus consisting 80 000 words. They followed TDIL tags for training and testing the system. They obtained an accuracy of 92.4 % for POS tagging.

3 CONTRIBUTION

3.1 Corpora Extraction

MT is one of the well known NLP applications. In the recent years, MT systems are built based on Neural network approach [23], parallel data or with the aid of bilingual dictionaries. It is hard to find a machine readable dictionary for resource scarce language like Kannada. Bilingual dictionaries are usually built using sentence aligned parallel text corpus. But, latest advances in developing a bilingual dictionary is using comparable corpora [24]. Wikipedia is a well- known comparable corpora, we used Wikipedia for the construction of bilingual dictionary of Kannada and English language pair.

Wikipedia contains wide range of articles in different languages and several link statistics amongst articles. It is being utilized as corpora in various NLP tasks fruitfully. Pages on Wikipedia connect to equivalent pages in other languages on similar topic via interlanguage links. For instance, there exist an interlanguage link between English article “Telephone” to the corresponding Kannada article, as depicted in Figure 1.

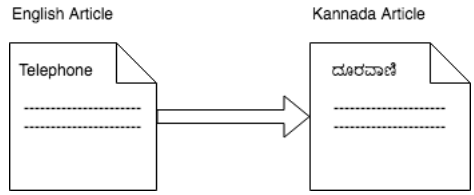


Figure 1. Interlanguage link example

Titles of the majority of the articles which are associated by an interlanguage links are translations of each other. Even though interlanguage links are accurate, there exists some extent of discrepancies as links are generally created manually. It is observed that, in addition to article titles, text inside the articles also share parallel contents to the great extent. Grounded on the above observation, we used interlanguage link to collect Kannada-English comparable corpus from Wikipedia. Steps adapted to develop comparable corpora of 19 263 articles of English and Kannada from Wikipedia are as follows:

Step 1: Kannada and English latest Wikipedia database dump was downloaded from `http://download.wikimedia.org` using a python script.

Step 2: Articles in English which have Kannada interlanguage link were downloaded, followed by the extraction of linked Kannada articles.

Step 3: Paragraphs under each heading are assumed to be related and those which contained general information are retained to ensure comparability.

Step 4: Extracted articles are cleaned by removing unrelated words and superlinks.

3.2 Bilingual Dictionary Creation

Proposed method to create bilingual dictionary, assumes that there exists a correlation amongst the patterns of word-co-occurrence across languages. However, it only requires a medium set of comparable documents which are pre-aligned documents with similar topics.

1. Generating Named Entity Dictionary

   Named Entities (NEs) are the names of persons, organizations, companies etc., i.e., during translation NEs should be transliterated rather than translation. Most of the conventional dictionaries do not have NEs. We took advantage of these NEs to locate comparable sentences in both Kannada and English documents. Also, these NE mapping helped us to find similar sentences across sections. So, to begin with, we tried to map NE in similar articles of both languages. A list of every NE in each English article in the downloaded corpus was created. NE recognition of English words was performed using built

in Stanford NE tagger in Python. Using the combination based transliteration algorithm [25] identified NEs were transliterated to Kannada script. The resulted transliterated NEs in Kannada script were searched and extracted in corresponding Kannada article to match similar sentences. Levenshtein distance algorithm was implemented to perform string matching of transliterated NE and the corresponding NE in Kannada article. Thus, a list of NEs in English articles and its corresponding mapping in Kannada articles was built and appended to our bilingual dictionary. The sentences which contained NE in English and its corresponding Kannada article were short-listed to find to obtain word level association (mappings).

2. Generate Title Dictionary

   Comparable corpus consisted of text related to similar topics but in distinct languages and authored by different authors. Therefore, the article contents may not be precise translations, but they convey information on similar topics. However, titles of such documents are perfect candidates of dictionary entries. To begin with, document title pairs of source and target languages were aligned and preprocessed to remove special characters and numerals. Title pairs were appended to the dictionary, forming a seed dictionary of title pairs. As observed, sub headings of source and target documents may not be same as they are written by different authors. Based on the initial dictionary constructed, related sections of articles in both English and Kannada were found. Sentences which were parallel to some extent are mined from these related sections. Most frequent words in these sentences were appended to the existing dictionary list by calculating word level similarity. Word level similarity was calculated using Pearson correlation coefficient which provided score, where every word in Kannada language gets a score for words in English language. These words were sorted based on their scores to get the best related words in Kannada language for each English word. Algorithm 1 describes generation of title mappings, sub heading mappings and word pair mappings.

   Finally, NE dictionary, title mappings, sub heading mappings and word pair mappings are combined to form a bilingual English–Kannada dictionary.

3. Results

   Dictionary created by proposed method has been evaluated using precision metrics. Precision (P) is the fraction of sum of appropriately (N) translated word pairs to total (T) number of translations in the dictionary which is used to moderate accuracy.

   Bilingual dictionary generated was evaluated manually and their respective precision scores are shown in Table 1.

Algorithm 1 Generate title mappings, sub heading mappings and word pair mappings

```
for all En-document in English-corpus do
    En-title ← Title of En-document
    Ka-document ← corresponding Kannada document in Kannada-corpus
    Ka-title ← Title of Ka_document
    if (En-title, Ka-title) not present in Dict then
        Dict ← Dict U (En-title, Ka-title)
    end if
    for all (En-subheading, Ka-subheading) do
        score-map ← Pearson correlation coefficient(En-subheading,
            Ka-subheading)
        while score-map is not empty do
            (En-subheading, Ka-subheading) ← max-ScoreEntry(score-map)
        if (En-subheading, Ka-subheading) not present in dictionary then
            Dict ← Dict U (En_subheading, Ka_subheading)
            remove all other entries from score-map
        end if
        end while
    end for
    for all partial parallel sentences do
        remove stop words
        Add co-occuring word pairs to score map
        while score-map is not empty do
            (En-word, Ka-word) ← max-ScoreEntry(score-map)
        if (En-word, Ka-word) not present in dictionary then
            Dict ← Dict U (En_word, Ka_word)
            remove all other entries from score_map
        end if
        end while
    end for
end for
```

Phase	Tokens	Precision
Gathering NEs	33 000	0.76
Gathering Title heading	23 398	0.89
Gathering Subheading mapping	1 362	0.86
Co-occuring words	16 785	0.65
Overall	77 545	0.79

Table 1. Precision scores

3.3 Query Translation Process

The wholeness of an IR paradigm stays in its capability to figure out the proper meaning of the input queries before search. In contrast to regular IR, MSIR need a translation system either by human or machine. The proposed Query translation approach ensures to convert user query into document language before retrieval. The proposed approaches to translate mixed script Kannada English queries to monolingual queries are following.

1. Dictionary based translation with POS tagging

   (a) POS Tagging
   In the proposed approach, as input is a Mixed Script query adoption of POS tagging was identified as a fruitful step in the translation process. Unlike, conventional MT systems, we followed word by word translations by ignoring syntax structure of the respective language.
   Input Mixed Script query contains both Kannada and English words in their respective scripts. POS tagging of English words was performed by in built Stanford POS tagger. Kannada words were tagged with BiLSTM-CRF neural network approach, which yielded accuracy of 92 %.

   (b) Translation
   Input query was translated to monolingual Kannada and English queries with the help of bilingual dictionary constructed. Each POS tagged English word in input query was translated to Kannada using dictionary and NEs were transliterated to Kannada script. Thus, forming input query in Kannada language. Query in Kannada language was translated to English using bilingual dictionary forming an English query. All translations were word to word without considering the syntactic structure of the respective languages.

2. Word Embedding (WE) + dictionary

   We found that dictionary-based method fail to translate words which do not have translations. Word Embeddings were adopted to handle such query terms. We trained the word2vec package for both the Kannada and English monolingual documents of comparable corpus obtained from Wikipedia dumps. We used the Continuous Bag of Words (CBOW) model with a window size of 5 and output vector of 300 dimensions with other default parameters set.

   Given an mixed script query as input, each English word in the query translations were taken from the bilingual dictionary, if a translation exists. If not, it is transformed into vector to find similar vector embeddings from corpus, and then translation of a English word of input query to Kannada is performed. Thus, input mixed script query is translated into a monolingual Kannada query. The above technique is followed to translate Kannada words in the input query to English to form monolingual English query.

3. Dictionary +WE+ Google translate

   In this technique a hybrid method is followed by combining dictionary-based method, WE and Google translation. If the translations for input query word does not exist either in the dictionary or in the WEs, then the words were translated using Google translation.

4 RESULTS

We used Anaconda with Python 3 version to build all translation models. We used NLTKs Bilingual Evaluation Understudy (BLEU) Score and Metric for METEOR to evaluate translation performance. BLEU is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another based on n-gram precision. Whereas, METEOR metric is based on the harmonic mean of unigram precision and recall, with recall weighted higher than precision. We tested translation paradigm using mixed script queries on current trending topics from Google trends, newspaper headlines and YouTube search queries.

1. Dictionary Based Translation

   (a) English to Kannada translation to form monolingual Kannada Queries
   Table 2 and Figure 2 portrays sample dictionary-based English to Kannada translation and BLEU scores for sample queries, respectively.

Kannada English Input Mixed Script Query	**Translation in Kannada**	**Translations (dictionary)**	**BLEU**	**METEOR score**
ವಿರುಷ್ಕಾ grand reception	ವಿರುಷ್ಕಾ ಅದ್ದೂರಿ ಆರತಕ್ಷತೆ	ವಿರುಷ್ಕಾ grand ಆರತಕ್ಷತೆ	0.81	0.63
where ಕಬ್ಬಿಣದ ಕಂಬ located in india	ಕಬ್ಬಿಣದ ಕಂಬ ಭಾರತದಲ್ಲಿ ಎಲ್ಲಿದೆ	ಎಲ್ಲಿ ಕಬ್ಬಿಣದ ಕಂಬ located in india	0.45	0.45
ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ live telecast	ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ ನೇರ ಪ್ರಸಾರ	ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ live telecast	0.57	0.59

Table 2. Sample dictionary-based English to Kannada translation

   (b) Kannada to English translation to form monolingual English Queries
   Table 3 and Figure 3 portrays sample dictionary-based Kannada to English translation and BLEU scores for mixed script queries, respectively.

2. Dictionary based + WE translation

   (a) English to Kannada translation to form monolingual Kannada Queries

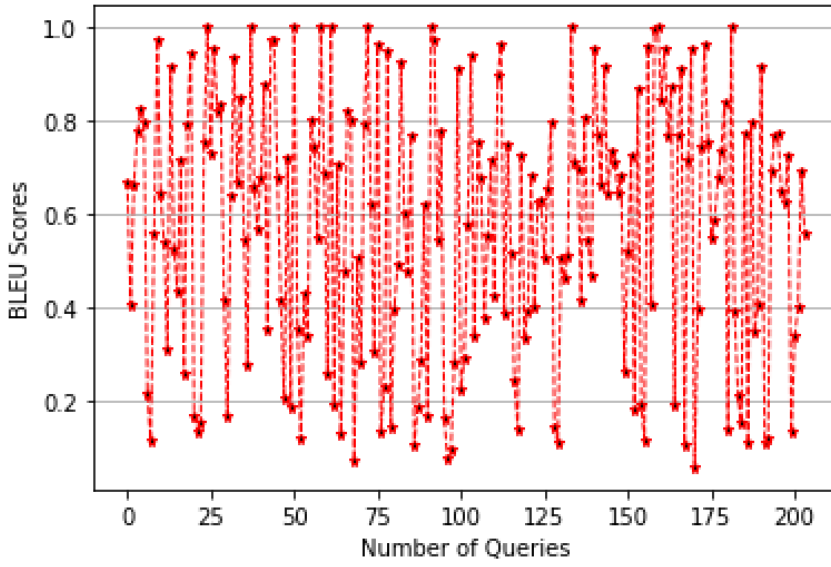


Figure 2. BLEU scores for English to Kannada dictionary translation

Kannada English Input Mixed Script Query	**Translation in English**	**Translations (dictionary)**	**BLEU**	**METEOR score**
ವಿರುಷ್ಕಾ grand reception	Virushka grand reception	ವಿರುಷ್ಕಾ grand reception	0.57	0.62
where ಕಬ್ಬಿಣದ ಕಂಬ located in india	where is iron pillar located in India	where is iron plated pillared located in India	0.65	0.97
earth ಒಳ ಪದರ how much	earth inner layers how much	earth ಒಳ layers how much	0.63	0.75

Table 3. Sample dictionary-based Kannada to English translation

An illustration of results obtained for dictionary-based + WE English to Kannada translation and BLEU scores for mixed script queries are shown in Table 4 and Figure 4.

(b) Kannada to English translation to form monolingual English Queries

An illustration of results obtained for dictionary-based + WE English to Kannada translation and BLEU scores for mixed script queries are shown in Table 5 and Figure 5.

3. Dictionary based + WE + Google translate

(a) English to Kannada translation to form monolingual Kannada Queries

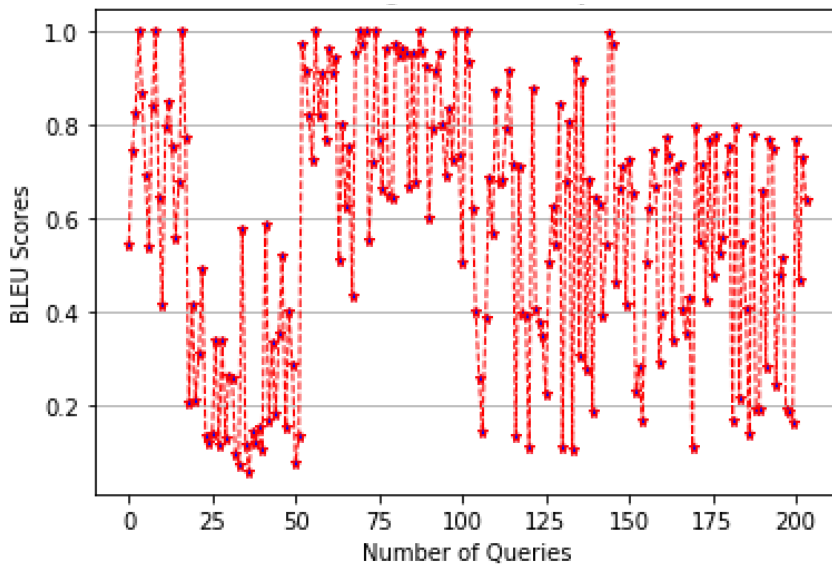


Figure 3. BLEU scores for Kannada to English dictionary translation

Kannada English Input Mixed Script Query	**Translation in Kannada**	**Translations**	**BLEU**	**METEOR score**
ವಿರುಷ್ಕಾ grand reception	ವಿರುಷ್ಕಾ ಅದ್ದೂರಿ ಆರತಕ್ಷತೆ	ವಿರುಷ್ಕಾ ಮಹೋನ್ನತ ಆರತಕ್ಷತೆ	0.81	0.63
where ಕಬ್ಬಿಣದ ಕಂಬ located in india	ಕಬ್ಬಿಣದ ಕಂಬ ಭಾರತದಲ್ಲಿ ಎಲ್ಲಿದೆ	ಎಲ್ಲಿ ಕಬ್ಬಿಣದ ಕಂಬ ಭಾರತದಲ್ಲಿದೆ	0.94	0.67
ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ live telecast	ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ ನೇರ ಪ್ರಸಾರ	ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ live telecast	0.57	0.59

Table 4. Sample Dictionary based + WE English to Kannada translation

Kannada English Input Mixed Script Query	**Translation in English**	**Translations**	**BLEU**	**METEOR score**
ವಿರುಷ್ಕಾ grand reception	Virushka grand reception	ವಿರುಷ್ಕಾ grand reception	0.57	0.62
where ಕಬ್ಬಿಣದ ಕಂಬ located in india	where is iron pillar located in India	where is iron plated pillared located in India	0.75	0.97
earth ಒಳ ಪದರ how much	earth inner layers how much	earth ಒಳ layers how much	0.63	0.75

Table 5. Sample Dictionary based + WE Kannada to English translation

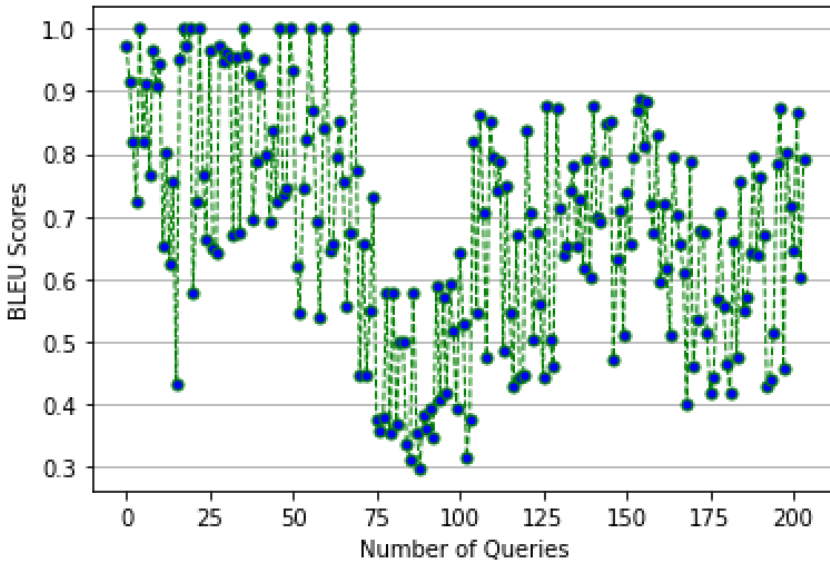


Figure 4. BLEU scores for dictionary based + WE English to Kannada translation

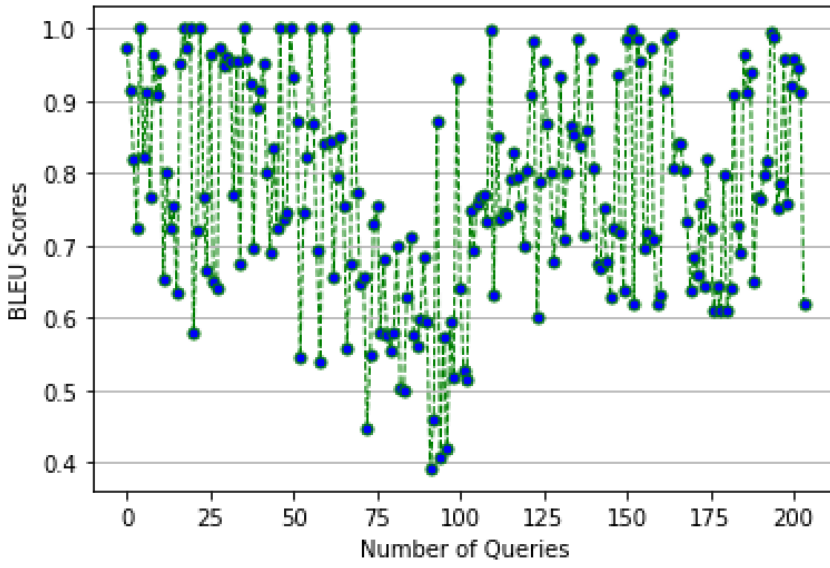


Figure 5. BLEU scores for dictionary based + WE Kannada to English translation

It was observed that translation results were improved by appending google search along with dictionary and WE which is presented in Table 6 and Figure 6.

Kannada English Input Mixed Script Query	**Translation in Kannada**	**Translations**	**BLEU**	**METEOR score**
ವಿರುಷ್ಕಾ grand reception	ವಿರುಷ್ಕಾ ಅದ್ದೂರಿ ಆರತಕ್ಷತೆ	ವಿರುಷ್ಕಾ ಮಹೋನ್ನತ ಆರತಕ್ಷತೆ	0.81	0.63
where ಕಬ್ಬಿಣದ ಕಂಬ located in india	ಕಬ್ಬಿಣದ ಕಂಬ ಭಾರತದಲ್ಲಿ ಎಲ್ಲಿದೆ	ಎಲ್ಲಿ ಕಬ್ಬಿಣದ ಕಂಬ ಭಾರತದಲ್ಲಿದೆ	0.94	0.67
ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ live telecast	ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ ನೇರ ಪ್ರಸಾರ	ರಗ್ಬಿ ವಿಶ್ವ ಕಪ್ ನೇರ ಪ್ರಸಾರ	1.0	1.0

Table 6. Dictionary based + WE + Google translate English to Kannada translation

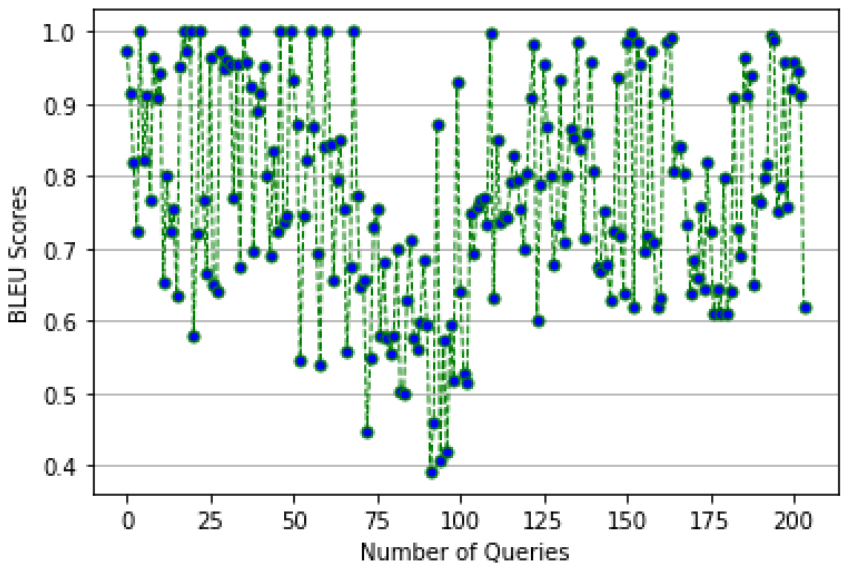


Figure 6. BLEU scores for dictionary based + WE + Google translate English to Kannada translation

(b) Kannada to English translation to form monolingual English Queries
Table 7 and Figure 7 portrays sample dictionary-based + WE + Google translate English to Kannada translation and BLEU scores for mixed script queries respectively.

Kannada English Input Mixed Script Query	**Translation in English**	**Translations**	**BLEU**	**METEOR score**
ವಿರುಷ್ಕಾ grand reception	Virushka grand reception	Virushka grand reception	1.0	1.0
where ಕಬ್ಬಿಣದ ಕಂಬ located in india	where is iron pillar located in India	where is iron plated pillared located in India	0.65	0.97
earth ಒಳ ಪದರ how much	earth inner layers how much	earth inner layers how much	1.0	1.0

Table 7. Dictionary based + WE + Google translate Kannada to English translation

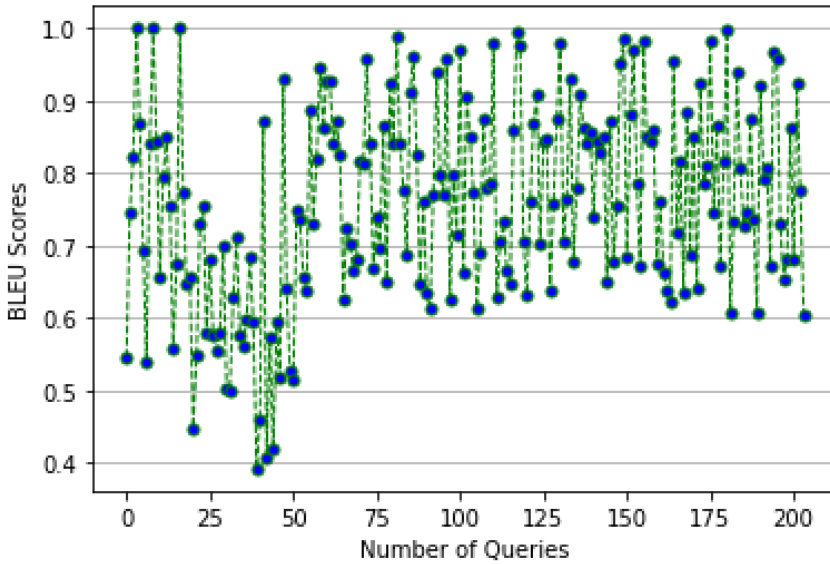


Figure 7. BLEU scores for dictionary based + WE + Google translate Kannada to English translation

It was observed that combination of all three methods, i.e. Dictionary based + word embedding+Google translate, yielded good performance in English to Kannada translation and vice versa. Hence, the method was followed to achieve translations. Words which were not translated by Dictionary based + word embedding + Google translate method were assumed to be NEs and they were transliterated.

5 CONCLUSION

Even though MSIR is a very notable and pervasive problem, it has gained very little attention. In this study, the problem of MSIR is handled for Queries of Kannada

English language pair. A promising solution to address the principal issue of MSIR, i.e., script variations in query was proposed. The MSIR model understands POS of the query terms using BiLSTM-CRF algorithms such that input query words were translated to other language words appropriately. Bilingual dictionary of Kannada and English language was built using Wikipedia dumps to aid translation. An attempt to translate mixed script queries of Kannada and English languages into monolingual queries was done. Three approaches for translation was proposed by constructing bilingual dictionary, word embeddings and Google translate. Proposed approaches were evaluated using BLEU and METEOR metrics. Experimental results shows that proposed Dictionary based + WE + Google translate model achieve better translations than other two models.

Future work includes refinement of the machine translation approach by exploring alternative techniques. One of the refinements could be to make the choice of NMT. As for alternative evaluation techniques, it would be interesting to experiment with other metrics like Translation Error Rate (TER), NIST. Future effort in evaluation would be directed toward character-based metrics which might show the highest correlation with human judgement.

Acknowledgement

The authors of this article gratefully thank the Visvesvaraya Technological University, Jnana Sangama, Belagavi for financial support extended to this research work.

REFERENCES

[1] Yan, Q.—Grefenstette, G.—Evans, D. A.: Automatic Transliteration for Japanese-to-English Text Retrieval. Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003, pp. 353–360, doi: 10.1145/860435.860499.

[2] Ahmed, U. Z.—Bali, K.—Choudhury, M.—Sowmya, V. B.: Challenges in Designing Input Method Editors for Indian Languages: The Role of Word-Origin and Context. Proceedings of the Workshop on Advances in Text Input Methods (WTIM 2011), Chiang Mai, Thailand, 2011, pp. 1–9.

[3] Pal, D.—Majumder, P.—Mitra, M.—Mitra, S.—Sen, A.: Issues in Searching for Indian Language Web Content. Proceedings of the 2nd ACM Workshop on Improving Non English Web Searching (iNEWS '08), 2008, pp. 93–96, doi: 10.1145/1460027.1460044.

[4] Dua, N.—Gupta, K.—Choudhury, M.—Bali, K.: Query Completion Without Query Logs for Song Search. Proceedings of the 20th International Conference Companion on World Wide Web (WWW '11), 2011, pp. 31–32, doi: 10.1145/1963192.1963209.

[5] Gupta, K.—Choudhury, M.—Bali, K.: Mining Hindi-English Transliteration Pairs from Online Hindi Lyrics. Proceedings of the Eighth International Conference

on Language Resources and Evaluation (LREC '12), Istanbul, Turkey, 2012, pp. 2459–2465.

[6] Sowmya, V. B.—Varma, V.: Transliteration Based Text Input Methods for Telugu. In: Li, W., Mollá-Aliod, D. (Eds.): Computer Processing of Oriental Languages. Language Technology for the Knowledge-Based Economy (ICCPOL 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5459, 2009, pp. 122–132, doi: 10.1007/978-3-642-00831-3_12.

[7] Janarthanam, S. C.—Subramaniam, S.—Nallasamy, U.: Named Entity Transliteration for Cross-Language Information Retrieval Using Compressed Word Format Mapping Algorithm. Proceedings of the 2nd ACM Workshop on Improving Non English Web Searching (iNEWS '08), 2008, pp. 33–38, doi: 10.1145/1460027.1460033.

[8] Oard, D. W.—Levow, G.-A.—Cabezas, C. I.: CLEF Experiments at Maryland: Statistical Stemming and Backoff Translation. In: Peters, C. (Ed.): Cross-Language Information Retrieval and Evaluation (CLEF 2000). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2069, 2001, pp. 176–187, doi: 10.1007/3-540-44645-1_17.

[9] Srivastava, R.—Bhat, R. A.: Transliteration Systems Across Indian Languages Using Parallel Corpora. Proceedings of the 27th Pacific Asia Conference on Language, Information, and Computation (PACLIC 27), Taipei, Taiwan, 2013, pp. 390–398.

[10] Gupta, P.—Bali, K.—Banchs, R. E.—Choudhury, M.—Rosso, P.: Query Expansion for Mixed-Script Information Retrieval. Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '14), 2014, pp. 677–686, doi: 10.1145/2600428.2609622.

[11] Pathak, A. K.—Acharya, P.—Kaur, D.—Balabantaray, R. C.: Automatic Parallel Corpus Creation for Hindi-English News Translation Task. 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, IEEE, 2018, pp. 1069–1075, doi: 10.1109/ICACCI.2018.8554461.

[12] Li, X.—Zhang, J.—Zong, C.: One Sentence One Model for Neural Machine Translation. 2016, arXiv: 1609.06490.

[13] Sepesy Maučec, M.—Donaj, G.: Machine Translation and the Evaluation of Its Quality. In: Sadollah, A., Sinha, T. S. (Eds.): Recent Trends in Computational Intelligence. IntechOpen, 2019, doi: 10.5772/intechopen.89063.

[14] Dunder, I.—Seljan, S.—Pavlovski, M.: Automatic Machine Translation of Poetry and a Low-Resource Language Pair. 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), Opatia, Croatia, IEEE, 2020, pp. 1034–1039, doi: 10.23919/MIPRO48935.2020.9245342.

[15] Khan, S.—Usman, I.: A Model for English to Urdu and Hindi Machine Translation System Using Translation Rules and Artificial Neural Network. The International Arab Journal of Information Technology, Vol. 16, 2019, No. 1, pp. 125–131.

[16] Gadde, P.—Yeleti, M. V.: Improving Statistical POS Tagging Using Linguistic Feature for Hindi and Telugu. International Conference on Natural Language Processing (ICON-2008), 2008.

[17] Ekbal, A.—Bandyopadhyay, S.: Part of Speech Tagging in Bengali Using Support Vector Machine. International Conference on Information Technology (ICIT '08), Bhubaneswar, India, IEEE, 2008, pp. 106–111, doi: 10.1109/ICIT.2008.12.

[18] Devadath, V. V.—Sharma, D. M.: Significance of an Accurate Sandhi-Splitter in Shallow Parsing of Dravidian Languages. Proceedings of the ACL 2016 Student Research Workshop, Berlin, Germany, ACL, 2016, pp. 37–42, doi: 10.18653/v1/p16-3006.

[19] Antony, P. J.—Mohan, S. P.—Soman, K. P.: SVM Based Part of Speech Tagger for Malayalam. 2010 International Conference on Recent Trends in Information, Telecommunication and Computing (ITC), Kerala, India, IEEE, 2010, pp. 339–341, doi: 10.1109/itc.2010.86.

[20] Ganesh, J.—Parthasarathi, R.—Geetha, T. V.—Balaji, J.: Pattern Based Bootstrapping Technique for Tamil POS Tagging. In: Prasath, R., O'Reilly, P., Kathirvalavakumar, T. (Eds.): Mining Intelligence and Knowledge Exploration. Springer, Cham, Lecture Notes in Computer Science, Vol. 8891, 2014, pp. 256–267, doi: 10.1007/978-3-319-13817-6_25.

[21] Shambhavi, B. R.—Kumar, P. R.: Kannada Part-of-Speech Tagging with Probabilistic Classifiers. International Journal of Computer Applications, Vol. 48, 2012, No. 17, pp. 26–30, doi: 10.5120/7442-0452.

[22] Graves, A.—Schmidhuber, J.: Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. Neural Networks, Vol. 18, 2005, No. 5-6, pp. 602–610, doi: 10.1016/j.neunet.2005.06.042.

[23] Liu, X.—Zhao, J.—Sun, S.—Liu, H.—Yang, H.: Variational Multimodal Machine Translation with Underlying Semantic Alignment. Information Fusion, Vol. 69, 2021, pp. 73–80, doi: 10.1016/j.inffus.2020.11.011.

[24] Laville, M.—Hazem, A.—Morin, E.: TALN/LS2N Participation at the BUCC Shared Task: Bilingual Dictionary Induction from Comparable Corpora. Proceedings of the 13th Workshop on Building and Using Comparable Corpora, Marseille, France, 2020, pp. 56–60.

[25] Sowmya Lakshmi, B. S.—Shambhavi, B. R.: Automatic English to Kannada Back-Transliteration Using Combination-Based Approach. In: Sridhar, V., Padma, M., Rao, K. (Eds.): Emerging Research in Electronics, Computer Science and Technology. Springer, Singapore, Lecture Notes in Electrical Engineering, Vol. 545, 2019, pp. 159–170, doi: 10.1007/978-981-13-5802-9_15.

B. S. Sowmya Lakshmi received B.E. degree from the Visvesvaraya Technological University (VTU) in 2011 and M.Tech. in 2013. In 2021 she completed her Ph.D. from VTU in the field of natural language processing and information retrieval. She has academic experience of about 2 years and published more than 10 research papers in international journals and conferences. Currently she is Assistant Professor in the Department of Machine Learning, BMSCE, Bangalore.

B. R. Shambhavi completed her Ph.D. from the Visvesvaraya Technological University in the area of natural language processing. She has academic and industry field experience of about 13 years. Her areas of interest are natural language processing and information retrieval. She has published more than 20 research papers in international journals and conferences. Currently she is Associate Professor in the Department of ISE, BMSCE, Bangalore. She is a life member of Indian Society for Technical Education (ISTE).

Computing and Informatics, Vol. 40, 2021, 648–689, doi: 10.31577/cai_2021_3_648

MODELLING AND CONTROL OF RESOURCE ALLOCATION SYSTEMS WITHIN DISCRETE EVENT SYSTEMS BY MEANS OF PETRI NETS – PART 1: INVARIANTS, SIPHONS AND TRAPS IN DEADLOCK AVOIDANCE

František Čapkovič

Institute of Informatics
Slovak Academy of Sciences
Dúbravská cesta 9
845 07 Bratislava, Slovakia
e-mail: Frantisek.Capkovic@savba.sk

Abstract. Solving the deadlocks avoidance problem in Resource Allocation Systems (RAS) in Discrete-Event Systems (DES) is a rife problem, especially in Flexible Manufacturing Systems (FMS), alias Automated Manufacturing Systems (AMS). Petri Nets (PN) are an effectual tool often used at this procedure. In principle, there are two basic approaches how to deal with deadlocks in RAS based on PN. They are listed and illustrated here. First of the approaches is realized by means of the supervisor based on P-invariants of PN, while the second one is realized by means of the supervisor based on PN siphons. While the first approach needs to know the reachability graph/tree (RG/RT) expressing the causality of the development of the PN model of RAS, in order to find (after its thorough analysis) the deadlocks, the second approach needs the thorough analysis of the PN model structure by means of finding siphons and traps. Next, both approaches will be applied on the same PN model of RAS and the effectiveness of the achievement of their results will be compared and evaluated. Several simple illustrative examples will be introduced. For the in-depth analysis of the problem of deadlock avoiding, next Part 2 of this paper is prepared, where the newest research will be introduced and illustrated on more complicated examples. If necessary (because of the limited length of particular papers), also the third part – Part 3, will be prepared.

Keywords: Automated manufacturing systems, control synthesis, deadlocks, deadlock avoidance, discrete-event systems, flexible manufacturing systems, modelling,

Petri nets, place/transition Petri nets, P-invariants, resource allocation systems, siphons, supervisor, T-invariants, traps

Mathematics Subject Classification 2010: 93-C65, 93-C30

1 INTRODUCTION

This paper has the character of an overview paper. It is conceived as the first part of a two-part paper (maybe also a three-part, if it will be needed). Its main aim is to introduce and describe principled terms and two basic approaches to the deadlock avoidance in Resource Allocation Systems (RAS) in Discrete-Event Systems (DES) as well as to present simple illustrative examples. The intended Part 2 of this paper (possibly also the Part 3, if necessary), to solve more complicated cases of RAS using newer methods, as well as comparing the mentioned two approaches, will be submitted later.

Flexible Manufacturing Systems (FMS), lately also called Automated Manufacturing Systems (AMS), represent a class of DES. They consists of various resources like machine tools, robots, buffers, transport belts, automatically guided vehicles (AGV) and so on. The resources are usually shared by two or more subsystems of AMS/FMS. Because of a limited number of resources different kinds of problems arise during the system operation, especially deadlocks [46]. Deadlocks are undesirable and unfavorable because they disrupt the course of the technological process. Due to deadlocks, either the entire plant or some of its parts remain stagnate. In such a way the primal intention of the production cannot be achieved. Such a situation can be, especially from the practical view, understood as a very unsafe form of non-determinism. Consequently, the approaches how to deal with this are sought. The approaches employing Petri Nets (PN)-based models of AMS/FMS are often used [52]. Besides the approaches based on the analysis of PN reachability trees/graphs (RT/RG), the approaches based on utilizing PN siphons are more frequently used. Moreover, in the recent years siphon-based approaches even started to prevail.

DES are systems discrete in nature. Such a system remains in a real intact state until it is forced to change this state as a consequence of the occurrence of a discrete event. In this document, PN-based models of DES will be exclusively used.

1.1 Petri Net Structure

As to the structure, PN are bipartite directed graphs $\langle P, T, F, G\rangle$ with two kinds of nodes – places p_i, $i = 1, \ldots, n$, and transitions t_j, $j = 1, \ldots, m$, and also two kinds of edges being directed arcs – $f_{ij} \in \mathbb{Z}_{\geq 0}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$, from places to transitions (where $\mathbb{Z}_{\geq 0}$ is the set of non-negative integers), and $g_{ji} \in \mathbb{Z}_{\geq 0}$,

$j = 1, \ldots, m$, $i = 1, \ldots, n$, from transitions to places. In other words, f_{ij} and g_{ji} represent weights of the directed arcs (i.e. their multiplicity).

Thus, $P = \{p_1, \ldots, p_n\}$ is a set of places; $T = \{t_1, \ldots, t_m\}$ is a set of transition; $P \cap T = \varnothing$, $P \cup T \neq \varnothing$, with $\varnothing$ being the empty set; $F = \{f_{ij}\}_{i=1,n;\, j=1,m}$ is the set of the directed arcs from places to transitions (i.e. $p_i \to t_j$), $F \subseteq P \times T$; $G = \{g_{ji}\}_{j=1,m;\, i=1,n}$ is the set of the directed arcs from transitions to places (i.e. $t_j \to p_i$), $G \subseteq T \times P$; the set $B \subseteq (P \times T) \cup (T \times P)$, $B \in \mathbb{Z}$, were $\mathbb{Z}$ is the set of integers.

When nonzero elements of F, G are solely of value 1 the arcs are called ordinary, when some of nonzero elements of F, G or all of them are greater than 1 the arcs are called weighted. The sets F, G, B can be represented, respectively, by the incidence matrices $\mathbf{F} \in \mathbb{Z}_{\geq 0}^{(n \times m)}$, $\mathbf{G} \in \mathbb{Z}_{\geq 0}^{(m \times n)}$, $\mathbf{B} \in \mathbb{Z}^{(n \times m)}$ of the directed arcs. In general, $\mathbb{Z}_{\geq 0}^{(a \times b)}$ represents the $(a \times b)$ matrix of non-negative integers and $\mathbb{Z}^{(a \times b)}$ is the $(a \times b)$ matrix of integers.

PN defined in such a way are called place/transition PN(P/T PN). PN are called *pure* when they do not contain self-loops. PN are called *ordinary* when all weights of their arcs are equal to one.

In general, PN places can be of three kinds:

1. operation places representing a progress in AMS/FMS;
2. fixed resources representing shared devices or elements (e.g. working tools);
3. variable resources representing e.g. availability of semi-products, parts, etc.

A transition $t \in T$ is enabled in marking M, denoted by $M[t\rangle$, if and only if (verbally expresed by *iff* or symbolically by $\Leftrightarrow$) $\forall p \in P$: $M(p) \geq F(p,t)$. Consequently, when t is *enabled* in M, then t may yield (after its firing) another marking M' where $\forall p \in P : M'(p) = M(p) - F(p,t) + G(t,p)$. This is denoted as $M[t\rangle M'$. It means, that the *enabled* transition may be fired and partake of the PN marking evolution. When t does not meet the above introduced condition, i.e. when $\forall p \in P$: $M(p) < F(p,t)$, it is *disabled*. Such t cannot be fired, i.e., it cannot share in the marking development.

There exist specific transitions:

1. the *source* transition is a transition without any input place – it is unconditionally enabled;
2. the *sink* transition is a transition without any output place – it consumes but does not create any tokens.

1.2 Petri Net Dynamics

Besides the graph structure, PN have also dynamics (the PN marking evolution). In an effort to make an analogy with the classical control theory, consider the PN marking to be the state vector of the system being the PN model. Thus, the model

can be expressed by the following constrained linear discrete integer system:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{B}.\mathbf{u}_k, \quad k = 0, 1, \ldots \tag{1}$$

$$\mathbf{F}.\mathbf{u}_k \leq \mathbf{x}_k \tag{2}$$

where $\mathbf{x}_k = (\sigma^k_{p_1}, \ldots, \sigma^k_{p_n})^T$ with $\sigma^k_{p_i} \in \mathbb{Z}_{\geq 0}$ is the state vector (marking) expressing the states of particular places (the number of tokens in p_i) in the step k; $\mathbf{u}_k = (\gamma^k_{t_1}, \ldots, \gamma^k_{t_m})^T$ with $\gamma^k_{t_j} \in \{0, 1\}$ (where 0 means the disabled t_j, while 1 means the enabled t_j) is the control vector in the step k; $\mathbf{F}$ (frequently being named as **Pre**), $\mathbf{G}^T$ (frequently being named as **Post**) are, respectively, the incidence matrices corresponding to sets F, G; $\mathbf{B} = (\mathbf{G}^T - \mathbf{F})$ is the incidence matrix being the structural matrix of the system (1)–(2); $\mathbf{x}_0$ is the initial state vector (initial marking). Thus, the state vector $\mathbf{x}_k$ in (1) corresponds to $M(p)$ mentioned above.

In the following, we will use the symbol N for the PN introduced above, and the term marking of PN places as an alternative to the state of the places (i.e. the number of tokens placed in them). In other words, under PN we will understand $(N, \mathbf{x}_0)$. Under the symbol $\mathcal{R}$ we will mean the set of reachable states including the initial state $\mathbf{x}_0$. Sometimes $\mathcal{R}$ will be expressed by a matrix $\mathbf{X}_r \in \mathbb{Z}^{(n \times N_v)}_{\geq 0}$, whose columns are particular reachable state vectors with N_v being the number of the reachable state vectors (including the initial state $\mathbf{x}_0$). The columns of $\mathbf{X}_r$ represent the particular nodes of RT corresponding to the PN in question.

2 PRELIMINARIES

Let us introduce here the basic terms representing important terminology and properties of the PN models of AMS/FMS, which will be used in this paper.

2.1 Siphons, Traps, Deadlocks, Invariants, Repetitive and Characteristic Vectors

Definition of Siphons and Traps. There exist many papers where siphons and traps are defined – see e.g. [72, 84, 58, 60, 61, 92, 93, 54] and many newer ones mostly written by Chinese authors [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]. Siphons and traps are basic net structures of PN [84] which allow important views on the behaviour of the modelled system as well as on some implications on this behaviour.

For ordinary PN the definitions of siphons and traps are defined in many papers – see e.g [72, 17, 78, 84, 1] and many other new ones mentioned above – as follows. A nonempty subset $S \subset P$ in N is called a siphon if ${}^\bullet S \subseteq S^\bullet$, i.e., if every transition having an output place in S has an input place in S. In [72] and several other older works the siphon is even identified with deadlock. However, at present it is an obsolete understanding. A nonempty subset $Q \subset P$ in an ordinary PN is called a trap if $Q^\bullet \subseteq {}^\bullet Q$ i.e., if every transition having an input place in Q has an output place in Q.

The illustration of the simple siphon and the simple trap is introduced in Figure 1.

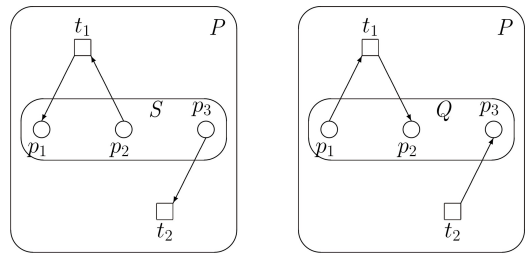


Figure 1. The a) siphon S and b) the trap Q inside a net N with the set of places P

As we can see, here ${}^\bullet S = \{t_1\}$, $S^\bullet = \{t_1, t_2\}$, ${}^\bullet S \subseteq S^\bullet$ while $Q^\bullet = \{t_1\}$, ${}^\bullet Q = \{t_1, t_2\}$, $Q^\bullet \subseteq {}^\bullet Q$. The number of tokens in the siphon S remains the same by firing t_1 and decreases by firing t_2. The number of tokens in the trap Q remains the same by firing t_1, but increases by firing t_2.

Consequences of Siphons and Traps. In general,

1. the siphon behaviour is such that if it has no token in a state (marking) of N, then it remains without any token in each successor state. Siphons represent a very important structural concept of PN. When all places in a siphon have no token, all transitions connecting with the siphon cannot be firable any more. Siphons are widely used to analyze PN liveness, and also to prevent deadlocks in PN models of DES. The terms liveness and deadlocks are introduced in the next paragraph;
2. the trap behaviour is such that if it has at least one token in a state (marking) of N, then it remains marked under each successor state. It was proved in [72] that the union of two siphons (traps) is again a siphon (trap).

The evident resume is that siphons are sets of places which, if become empty of tokens, they will always remain empty for all reachable markings of the net, while traps are sets of places which, if become marked, will always remain marked for all reachable markings of the net.

Traps can also be useful in combination with place invariants (see Subsection 2.2) to recapture information lost in the incidence matrix due to the cancellation of self-loop arcs.

Deadlocks and Their Relation with Siphons and Traps. In some literary sources – see e.g. [76] – is shown that each reachable marking of PN enables at least one transition. In doing so it means that each siphon S of PN contains as a subset

an initially marked trap. It was proved by the same author in [77] that a totally deadlocked ordinary Petri net contains at least one empty siphon.

If every non-empty siphon of PN includes a (sufficiently) marked trap then (see e.g. [84], but also many other authors) no dead marking is reachable. This is very important finding.

A siphon (trap) is named to be *minimal* if it does not contain any other siphon (trap). Minimal siphons provide a sufficient condition for the non-existence of deadlocks.

A *strict minimal* siphon [27] is a siphon containing neither other siphon nor a trap except itself. The sum of token numbers in S is denoted by $M(S)$, where $M(S) = \sum_{p \in S} M(p)$. A subset $S \subseteq P$ is marked by M if $M(S) > 0$. A siphon is under-marked if $\nexists t \in S^{\bullet}$ which can fire.

The *proper* siphon is the siphon when the set of its predecessors is strictly included in the set of its successors. It was shown in [11, 1] that in a deadlocked PN model all unmarked places form a siphon. Thus, the siphon-based approach for deadlocks detection checks if the net contains a proper siphon that can become unmarked by some firing sequence. A proper siphon does not become unmarked if it contains an initially marked trap.

Deadlocks and Liveness. The problem of deadlocks and their effective resolution was studied for the first time in the 60's [15, 16], in context of the multi-threaded computation or multi-programming (emerging at that time). Of course, since that time the deadlock theory, and especially the deadlock avoidance one, has been intensively developed – see e.g. [46, 78, 79, 80, 81, 75, 74, 73, 82, 86, 87] and especially [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53].

Simply said, the deadlock is the state of N if all transitions $t \in T$ are disabled. N is named to be deadlock-free when none reachable state $\mathbf{x} \in \mathcal{R}$ is a deadlock. Thus, the deadlock is a state of N when DES modelled by PN comes to a state in which no further changes are possible. In other words a deadlock is [70] a subset of places which, if none of them is marked at the beginning of the Petri net activity, will remain unmarked in all subsequent evolution. It can also be said that a subset of places $I \subset P$ is a deadlock *iff* (if and only if) each transition which is input transition of a place in I is also output transition of a place in I.

It was said in [26] that if M_0 is initial marking of N then in (N, M_0) is a deadlock only on the condition that $t \in T : M_0[t\rangle$ is never found.

In general, deadlocks occur in DES (especially AMS/FMS) when processes, which want to run (and should run), hold insufficient resources, as a result of which the system comes to a standstill. This is the acute problem which should be solved by means of PN based RAS.

A N is said to be live, more precisely M_0 is said to be a live marking for N, if (no matter what state (marking) has been reached from M_0) it is possible to ultimately fire any transition of N by progressing through further firing sequence. A live PN

guarantees [72] deadlock-free operation, no matter what firing sequence is chosen. More details about five kinds of liveness can be found in [72]. Namely, a transition t in (N, M_0) is said to be:

1. *dead* (lived on the level 0 – $L0$ live) – if it never be fired in any firing sequence in $L(M_0)$;
2. *potentially fired* ($L1$ live) – if it can be fired at least once in some firing sequence in $L(M_0)$;
3. $L2$ live – when for $k > 0$ it is fired at least k-times in $L(M_0)$;
4. $L3$ live – if it appears infinitely often in some firing sequence in $L(M_0)$;
5. $L4$ live or live if it is $L1$ live for every marking M in $\mathcal{R}(M_0)$.

The simple example of the live PN together with its reachability tree (RT) is given in Figure 2 while the simple example of the nonlive PN together with its reachability tree (RT) is given in Figure 3.

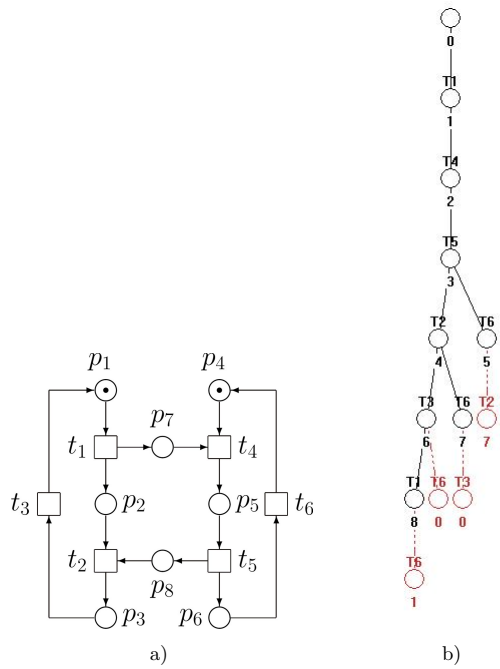


Figure 2. a) The live PN and b) its RT

As we can see in Figure 2 b), no state discontinues the course of the modelled process. On the other hand, in Figure 3 b) we can see that the states $\mathbf{x}_1 = (0\,1\,0\,1\,0\,0)^T$, $\mathbf{x}_6 = (0\,0\,1\,0\,1\,0)^T$ and $\mathbf{x}_7 = (0\,0\,0\,1\,0\,0)^T$ do this. These states are deadlocks.

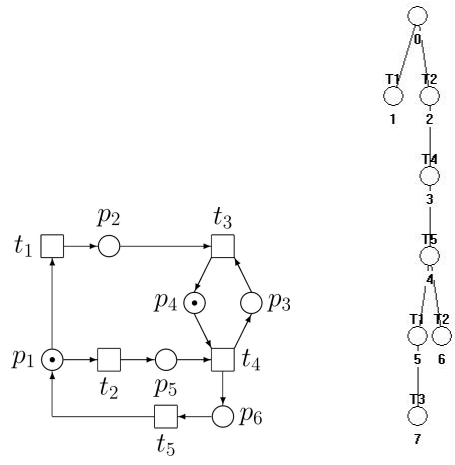


Figure 3. a) The nonlive PN and b) its RT

Simply said, a transition t of N is said to be live *iff* for all reachable states (markings) $\mathbf{x}_r \in \mathcal{R}$ there exists a sequence of transition firings which results in a marking in which t is enabled. The N is said to be live if all its transitions are live. Liveness of PN implies absence of deadlocks in the modelled DES. When RT has a node (vertex) without a successor, then PN is not live. As to RT, the Koenig lemma [14] is also useful. It says: "Let RT be a tree of finite degree (i.e., every vertex has a finite number of successors) and with an infinite number of vertices. Then RT has an infinite branch."

In PN theory siphons and traps have been introduced [11] to characterize deadlocks of PN. Simultaneously, they can help us at finding deadlock avoidance methods.

Other details about behavioral properties of liveness are summarized in [18].

How to deal with the problem caused by the deadlock, that is a cardinal question. There exist three basic approaches for RAS (see e.g. [19, 74, 73]) how to deal with deadlocks and problems pertinent to them:

1. deadlock detection and recovery – i.e., to detect deadlock occurrences and restore the systems operations with recovery procedures;
2. deadlock prevention – i.e., to prevent circular wait conditions using offline strategies, mutual exclusion,
3. deadlock avoidance – i.e., to prevent deadlock situation applying online policy control of resource allocation.

In doing so there are two principles how to create related methods, namely by means of digraphs or by means of PN. Their comparison can be found in [20].

In this paper, solely the PN-based approaches will be used.

2.2 Invariants, Repetitive and Characteristic Vectors

Invariants. There are two kinds of PN invariants: T-invariants and P-invariants. It is well known – see e.g. [72] – that the T-invariant of PN is defined as the $(m \times 1)$ vector $\mathbf{w}$ for which $\mathbf{B}.\mathbf{w} = \mathbf{0}$, $\mathbf{w} \neq \mathbf{0}$, where $\mathbf{0}$ is the $(n \times 1)$ vector of zeros, while the P-invariant is defined as the $(n \times 1)$ vector $\mathbf{y}$ for which $\mathbf{B}^T.\mathbf{y} = \mathbf{0}$, $\mathbf{y} \neq \mathbf{0}$, where $\mathbf{0}$ is the $(m \times 1)$ vector of zeros. When we want to compute so called *proper* invariants we have to ask $\mathbf{w} > \mathbf{0}$ and $\mathbf{y} > \mathbf{0}$, respectively. What is important is that:

1. The T-invariant, if it exists at all, will give the number of times different transitions should be fired in order that a particular marking may be reproducible.
2. From the definition of P-invariants ($\mathbf{y}^T.\mathbf{B} = \mathbf{0}$, i.e. $\mathbf{y}^T.\mathbf{x}_k \stackrel{!}{=} \mathbf{y}^T.\mathbf{x}_0$) it follows that for all reachable markings $\mathbf{x}_k \in \mathcal{R}$, the weighted sum of tokens is a constant. Let $\mathbf{I}$ is a P-invariant. The set $P_I \subset P$ is called the *support* of $\mathbf{I}$ [45] *iff* $P_I = \{p \in P \mid \mathbf{I}(p) \neq \mathbf{0}\}$. $\mathbf{I}$ is called *non-negative iff* $\mathbf{I} \geq \mathbf{0}$. $\mathbf{I} \gneqq \mathbf{0}$ is called minimal *iff* there exists no P-invariant $\mathbf{I}' \gneqq \mathbf{0}$ with $\mathbf{I}' \lneqq \mathbf{I}$. Here, the symbol $\gneqq$ means *greater-than but not equal* and the symbol $\lneqq$ means *less-than but not equal*.

   For the example of PN given in Figure 2 invariants (Figure 2 a)) and proper invariants (Figure 2 b)) are as follows:

$$\begin{aligned}
\mathbf{I}_1 &= (0,0,0,1,1,1,0,0)^T,\\
{}^1\mathbf{I}_{prop} &= (0,0,0,1,1,1,0,0)^T,\\
\mathbf{I}_2 &= (1,1,1,0,0,0,0,0)^T,\\
{}^2\mathbf{I}_{prop} &= (1,0,1,0,1,0,1,1)^T,\\
\mathbf{I}_3 &= (0,-1,0,0,1,0,1,1)^T,\\
{}^3\mathbf{I}_{prop} &= (1,1,1,0,0,0,0,0)^T.
\end{aligned}$$

   Consequently, when we use the proper invariants being considered to be invariants, the supports are the following:

$$\begin{aligned}
P_{I_1} &= \{p_4, p_5, p_6\},\\
P_{I_2} &= \{p_1, p_3, p_5, p_7, p_8\},\\
P_{I_3} &= \{p_1, p_2, p_3\},
\end{aligned}$$

   while for PN displayed in Figure 3 there exists only one invariant, one proper invariant equal to this invariant, and one support invariant as follows:

$$\begin{aligned}
\mathbf{I}_1 &= (0,0,1,1,0,0)^T,\\
{}^1\mathbf{I}_{prop} &= (0,0,1,1,0,0)^T,\\
P_{I_1} &= \{p_3, p_4\}.
\end{aligned}$$

Simply said, P-invariants are the sets of places whose weighted token sum remains constant for all possible markings, while T-invariants are the sets of firings that will cause a cycle in the state space, meaning the comeback to the original state (markings). The set of nodes corresponding to non-zero entries of an invariant is called the support of this invariant $\mathbf{I}$, written as supp($\mathbf{I}$). An invariant $\mathbf{I}$ is called minimal if $\nexists\, \mathbf{I}' : \text{supp}(\mathbf{I}') \subset \text{supp}(\mathbf{I})$, i.e., its support does not contain the support of any other invariant $\mathbf{I}'$, and the greatest common divisor of all non-zero entries of $\mathbf{I}$ is 1.

Characteristic Vectors. Let, in general, the P-vector means a vector expressing states of places (number of tokens inside them) – in (1)–(2) it is the state vector $\mathbf{x}$ – and the T-vector means a vector expressing states of transitions (enabled, disabled) – in (1)–(2) it is the control vector $\mathbf{u}$.

Let $S \subseteq P$ be a subset of places of N. The $(n \times 1)$ vector ${}^S\boldsymbol{\sigma}$ is called [58, 51, 91] the *characteristic P-vector* of S if $\forall p \in S : {}^S\sigma_p = 1$; otherwise ${}^S\sigma_p = 0$. The $(m \times 1)$ vector ${}^S\boldsymbol{\gamma} = \mathbf{B}^T.{}^S\boldsymbol{\sigma}$ is called the *characteristic T-vector* of S.

The physical interpretation of the T-vector of a subset of places is the following:

1. ${}^S\gamma(t) > 0$ means that ${}^S\gamma(t)$ tokens are put into S when the transition t fires;
2. ${}^S\gamma(t) = 0$ means that the number of tokens in S does not change after t fires;
   (iii) ${}^S\gamma(t) < 0$ implies that $|{}^S\gamma(t)|$ tokens are removed from S when t fires.

Repetitive Vectors. In [72] the term *repetitiveness* was also introduced. Namely, N is said to be (partially) *repetitive* if there exists a marking $\mathbf{x}_0$ and a firing sequence $\mathcal{U} = \{t_a, t_b, t_c \dots\}$ from $\mathbf{x}_0$, i.e. $\mathbf{x}_0[t_a\rangle\mathbf{x}_1[t_b\rangle\mathbf{x}_2[t_c\rangle \dots$, such that every (some) transition occurs infinitely often in $\mathcal{U}$. It was proved there that N is (partially) repetitive *iff* there exists an $(m \times 1)$ vector $\mathbf{q}$ of positive (non-negative) integers such that $\mathbf{B}.\mathbf{q} \geq \mathbf{0}$, $\mathbf{q} \neq \mathbf{0}$. Such vector $\mathbf{q}$ is named as the *repetitive vector*.

A repetitive vector $\mathbf{q}$ is reachable [23] *iff* there exists a reachable state (marking) $\mathbf{x} \in \mathcal{R}$ that allows firing a sequence $\mathcal{U}$ whose corresponding characteristic T-vector is $\mathbf{q}$.

The System Evolution. The (1)–(2) represent the discrete event system – DES. For completeness' sake it is necessary to introduce also the procedure of the marking development in PN (i.e. the system evolution).

Let us develop the system (1) from $\mathbf{x}_0$ to $\mathbf{x}_q$

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{B}.\mathbf{u}_0 \tag{3}$$

$$\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{B}.\mathbf{u}_1 = \mathbf{x}_0 + \mathbf{B}.\mathbf{u}_0 + \mathbf{B}.\mathbf{u}_1 = \mathbf{x}_0 + \mathbf{B}.(\mathbf{u}_0 + \mathbf{u}_1) \tag{4}$$

$$\dots \tag{5}$$

$$\mathbf{x}_q = \mathbf{x}_0 + \mathbf{B}.(\mathbf{u}_0 + \mathbf{u}_1 + \dots + \mathbf{u}_{q-1}) = \mathbf{x}_0 + \mathbf{B}.\sum_{i=0}^{q-1}\mathbf{u}_i \tag{6}$$

By the way, the vector represented the sum in (6) is called (in general) the *Parikh's vector*. Its entries show how many times the particular transitions are fired during the system evolution. Denote this vector in our case as $\mathcal{P}_q$. Now, develop analogically the system from $\mathbf{x}_q$ to $\mathbf{x}_r$

$$\mathbf{x}_{q+1} = \mathbf{x}_q + \mathbf{B}.\mathbf{u}_q \tag{7}$$

$$\mathbf{x}_{q+2} = \mathbf{x}_q + \mathbf{B}.\mathbf{u}_q + \mathbf{B}.\mathbf{u}_{q+1} = \mathbf{x}_q + \mathbf{B}.(\mathbf{u}_q + \mathbf{u}_{q+1}) \tag{8}$$

$$\dots \tag{9}$$

$$\mathbf{x}_r = \mathbf{x}_q + \mathbf{B}.(\mathbf{u}_q + \mathbf{u}_{q+1} + \dots + \mathbf{u}_{r-1}) = \mathbf{x}_q + \mathbf{B}.\sum_{i=q}^{r-1}\mathbf{u}_i \tag{10}$$

Denote the vector represented the sum in (10) as $\mathcal{P}_r$. Hence,

$$\mathbf{x}_r = \mathbf{x}_0 + \mathbf{B}.(\mathcal{P}_q + \mathcal{P}_r). \tag{11}$$

The Parikh's vectors $\mathcal{P}_q$, $\mathcal{P}_r$ represent, respectively, not only the firing sequences $\mathcal{U}_1$ (from $\mathbf{x}_0$ to $\mathbf{x}_q$) and $\mathcal{U}_2$ (from $\mathbf{x}_q$ to $\mathbf{x}_r$) but also how many times particular transitions are fired during the developments (6) and (10). From the point of view of the state $\mathbf{x}_q$ we can speak about the input firing sequence $\mathcal{U}_1$ and output firing sequence $\mathcal{U}_2$ or about $\mathcal{P}_q$ and $\mathcal{P}_r$, respectively.

A live PN guarantees deadlock-free operation, no matter what firing sequence is chosen. Moreover, equations introduced above represent the analytical expression of the principle of causality in PN.

2.3 Controllability Conditions for PN vs. Invariants and Siphons

An ordinary net N is said to be completely controllable if any marking is reachable from any other marking. Details how invariants and siphons make possible to control DES represented by PN models are introduced in following sections.

In case of control synthesis based on P-invariants, the permissive controller (supervisor) have to fulfill some conditions imposed on mutual relations among states of particular PN markings (states) in order to avoid deadlock identified by means of the thorough analysis of RT (RG).

In case of siphon-based approach the thorough structural analysis of PN is performed. Then, the properties of found siphons are utilized at the control synthesis. Although this topic was opened long ago [72, 1, 2] it is still very live – see recent contributions [7, 24, 25, 51, 9, 10, 63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]. The basis of such an approach to control consists in avoiding of emptying (unmarking) of places creating the siphons. A siphon in ordinary PN is said to be controlled [24] if it cannot be empted (unmarked) at any reachable marking.

If PN is generalized, owing to the weights of arcs, the non-emptyability of a siphon is not sufficient for the absence of dead transitions, and the controllability of a siphon is much more complex.

Elementary and Dependent Siphons. The overview and definitions of different kind of siphons in PN are given in [58]. The motivation to propose the concept of *elementary* siphons is to control *dependent* siphons by explicitly controlling their elementary siphons only.

Elementary siphons play an important role in the development of deadlock prevention approaches, that lead to structurally simple supervisors enforcing liveness, based on monitors.

The set $\Pi_E = \{S_\alpha, S_\beta, \ldots, S_\gamma\}$, $\{\alpha, \beta, \gamma\} \subseteq \mathbb{Z}$, is called the set of elementary siphons if $\{\boldsymbol{\gamma}_\alpha, \boldsymbol{\gamma}_\beta, \ldots, \boldsymbol{\gamma}_\gamma\}$ is a linearly independent maximal set of the matrix ${}^S\boldsymbol{\Gamma}$ consisting of T-vectors ${}^S\boldsymbol{\gamma}$ – i.e. ${}^S\boldsymbol{\Gamma} = \mathbf{B}^T.{}^S\boldsymbol{\Sigma}$, where ${}^S\boldsymbol{\Sigma}$ is the matrix consisting of P-vectors ${}^S\boldsymbol{\sigma}$.

The T-vector ${}^S\boldsymbol{\gamma}$ is associated [91] with each siphon S such that ${}^S\gamma(i)$ is the number of tokens gained in or lost from S by firing the transition t_i once. A dependent siphon S_0 strongly depends on elementary siphons S_1, S_2, …, S_k if

$$ {}^S\gamma(0) = a_1.{}^S\gamma(1) + a_2.{}^S\gamma(2) + \cdots + a_k.{}^S\gamma(k) \tag{12} $$

with $a_i \in \mathbb{Z}_{\geq 0}$, $i = 1, 2, \ldots, k$, being positive integers and ${}^S\gamma(k)$ being nonzero entries of ${}^S\boldsymbol{\gamma}$. Such dependent siphons are named as the strongly dependent siphons (SDS). S_0 is a weakly dependent siphon (WDS) if some a_i are negative. The T-vectors for elementary siphons are mutually independent. More details can be found e.g. in [47, 48, 49, 50, 51] as well as in [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40].

More details can be seen e.g. in [47, 48, 49, 50, 51] as well as in a great deal of applications in [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40], where also newest approaches to the siphon-based control of AMS/FMS and RAS are presented as follows.

In [56] the new controllability condition for siphons is presented. In [59] the necessary and sufficient condition of a kind of PN (namely GS^{3}PR) was proved. In [60, 62, 64] important findings in the area of robust deadlock control of AMS with unreliable resources are published. In [53, 50, 49, 48, 47, 91] the application of elementary siphons, being topical at present, is broadly investigated. In [55, 28, 29, 30, 31, 32, 33] the very useful iterative solution how to avoid the need of enumerating all the states or siphons using mathematical programming techniques were published. In [35, 36] also distributed resolution approaches to solving the deadlock avoiding were published. In [37] the very useful approach to simplification of the supervisor structures was published. In [37] the supervisor synthesis and performance improvement in an integrated way are also presented. In the works [38, 39, 40] the direct application of assembly AMS in the practice was shown.

2.4 Resource Allocation Systems vs. Petri Nets

RAS represent [86] a special class of concurrent systems, especially AMS/FMS, where the attention is focused on resources. RAS consist of a finite set of processes that share (in a competitive way) a finite set of resources. Such a competition can bring (i.e., is conducive to) existence of deadlocks. The deadlock causes an unacceptable state when some processes in AMS/FMS are waiting for the evolution of other processes that are also waiting for the evaluation of former ones in order to evolve.

PN models of RAS are especially useful at synthesizing deadlock prevention policies as well as deadlocks avoidance ones. Although many papers about RAS were published in the last three decades, it may be said that principle papers about RAS are [46, 78, 79, 80, 81, 75, 74, 73, 82, 86, 87]. Newer papers with very important contributions in this area are especially [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53].

There exist several standard kinds (paradigms) of RAS [41, 51, 22, 21, 94]. Specific nomenclatures have been established, e.g. Simple Sequential Process (S^2P), Simple Sequential Process with Resources (S^2PR), Systems of Simple Linear Sequential Processes with Resources (S^2LSPR), Systems of Simple Sequential Processes with Multiple Resources (S^3PMR), the subclass of System of Simple Sequential Processes with General Resource Requirements (S^3PGR2) [75], Generalised Systems of Simple Sequential Processes with Resources (GS^3PR), Systems of Simple Sequential Processes with Resources (S^3PR), Linear S^3PR (LS^3PR), Extension of S^3PR (ES^3PR), and already mentioned S^3PGR2 modelling manufacturing systems in general, Weighted System of Simple Sequential Processes with Several Resources (WS^3PSR), System of Sequential Systems with Shared Resources (S^4R), System of Sequential Systems with Shared Process Resources (S^4PR), etc.

The S^3PR are frequently used in AMS and they are modelled by means of PN. They represent a class of AMS with flexible routing and single-unit resource acquisition. In such systems the part being produced using only one copy of one resource at each processing step. Such systems create a subclass of a higher (upper) class S^*PR [94, 21] where more copies of one resource are allowed. The asterix does not represent exactly an integer expressing the number of copies, but a level of complexity. Multiple-unit systems with routing flexibility are much less investigated. S^4PR are explored a lot less than S^3PR. They are adequate [86, 87] for the modeling of a wide variety of RAS. The special syntactic characteristics of this class allow to study the modelled systems from a structural perspective. The newer publications [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53] investigate also such kinds of AMS/FMS.

The relation among some of PN-based models of RAS is illustrated in Figure 4.

From this point of view two kinds of PN places (being added to the PN model because of the siphon control) can be distinguished as to the synthesis of AMS/FMS

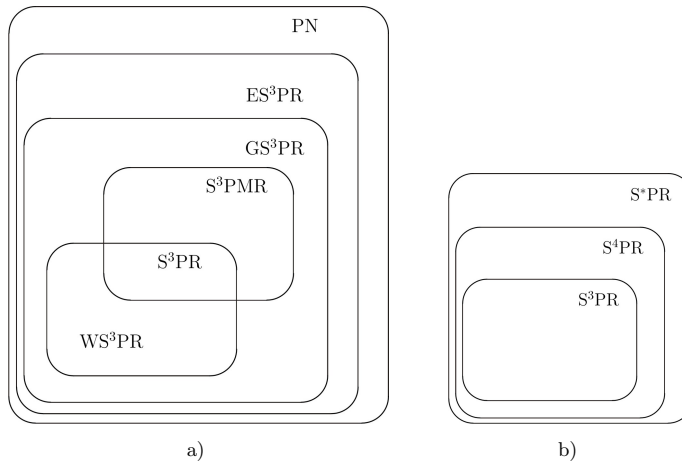


Figure 4. Relations (based on sets) among some of more important PN models of RAS

control, namely:

1. ordinary places;
2. weighted places.

Ordinary places have ordinary arcs and are added to the original PN in order to prevent related siphon from becoming unmarked whenever it is possible. Weighted places adopt a conservative policy controlling the release of component or parts in AMS/FMS, modelled by PN, into the system. It means that they are added to the original/modified PN, namely to the source transitions of the resultant PN, by means of their output arcs.

2.5 Literature Survey

In case of the approach based on P-invariants most important contributions can be seen in [42, 43, 71, 88, 89], but also in many other works.

However, in about the last two decades the siphon-based approach dominates among the methodologies that deal with the deadlock analysis and control of resource allocation systems. This research has uncovered many useful results. Very important share on the development of the siphon-based approach to deadlock avoiding have the works [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53], but also some others. The new controllability condition for siphons is presented in [56]. The necessary and sufficient condition of a kind of PN (namely GS^{3}PR) was proved in [59]. Very important are findings in the area of robust deadlock control of AMS with unreliable resources [60, 62, 64]. The application of elementary siphons, being topical at present, is broadly investigated in [53, 50, 49, 48, 47, 91]. Very useful

is the iterative solution using mathematical programming techniques to avoid the need of enumerating all the states or siphons published in [55, 28, 29, 30, 31, 32, 33] having direct impact on practice. Also distributed resolution approaches to solving the deadlock avoiding were published [35, 36]. Very useful is the approach to simplification of the supervisor structures [37]. The supervisor synthesis and performance improvement in an integrated way are also presented in [37]. For the direct application in the practice of assembly AMS are important the works [38, 39, 40].

A suitable combination of the approaches in the form of invariant-controlled elementary siphons is presented in [50].

Of course, in this paper it is impossible to devote to the complete problem of the deadlock avoidance in AMS/FMS. Here, in the Part 1, only a broader introductory part to the problem will be inducted and illustrated by simple explanatory examples. The permissible scope of this article does not allow more. In the second part – Part 2, being in preparation, newer methods of deadlock avoidance as well as their application to more complicated cases of AMS/FMS will be analyzed. May be that also a third part – Part 3 will be necessary because of the limited space for one paper in this journal.

2.6 The Paper Organization

After the detailed introduction in Section 1 and preliminaries in Section 2, which were necessary for initiation into the problem of the widely developed subject inside PN, the paper includes the next parts.

In Section 3, solving problems in RAS is introduced – namely, the description how to remove deadlocks and how to control RAS. This section is the core of the paper. It consists of three subsections. Subsection 3.1 details the proposal and application of the P-invariant method for synthesizing the supervisor removing deadlocks in RAS while Subsection 3.2 presents the proposal and application of siphon-based method for synthesizing the supervisor removing deadlocks in RAS. Subsection 3.3 presents a short comparison and evaluation of both approaches. Section 4 introduces the research plan for the future. Section 5 concludes the paper.

3 SOLVING PROBLEMS WITH DEADLOCKS IN RAS

Two possible approaches to remove deadlocks by means of the supervisory control are presented here.

The first one starts from the thorough analysis of RT of the PN model of RAS or equivalently from RG. It is necessary to say that RG arises from RT by means of joining all RT leaves with the same name into one node of RG. The adjacency matrix is the same for both RT and RG. From RT/RG (expressed either in graphical form or in the form of the adjacency matrix), information about deadlocks can be obtained. After finding deadlocks, the supervisor based on P-invariants is synthesized in order to remove these deadlocks.

The second approach performs the thorough structural analysis of the PN model of RAS. It finds and uses PN siphons and traps to synthesize the supervisor removing deadlocks. No RT/RG is necessary in this case.

Each of the mentioned approaches has its advantages and disadvantages. Therefore, we will test both approaches on simple practical examples and compare them. In Part 2 (potentially also in Part 3) more complicated cases of RAS will be tested using the newest findings in that field.

3.1 An Approach Based on P-Invariants with Current Knowledge of RG

After a detailed analysis of RG the deadlocks can be identified. Then, the conditions for supervisor synthesis based on P-invariants may be established.

Let us try to design a controller based on P-invariants – see e.g. [4]. Let the matrix $\mathbf{Y}$ denote the $(s \times n)$ matrix of P-invariants, which are not known till now. Start from the definition of P-invariants

$$\mathbf{Y}^T.\mathbf{B} \stackrel{!}{=} \mathbf{0}. \tag{13}$$

Consider the restrictive condition (following from the detail analysis of RG) on the state vector in the form as follows:

$$\mathbf{L}.\mathbf{x} \leq \mathbf{b} \tag{14}$$

where $\mathbf{L}$ is the $(s \times n)$ matrix of integers expressing the expected relations among states to eliminate deadlocks, and $\mathbf{b}$ is the $(s \times 1)$ vector of positive integers determining some restrictions on linear combinations of corresponding entries of the state vector $\mathbf{x}$. In order to remove the inequality (14), add a slack vector $\mathbf{x}_s$ and put

$$\mathbf{L}.\mathbf{x} + \mathbf{x}_s = \mathbf{L}.\mathbf{x} + \mathbf{I}_s.\mathbf{x}_s = (\mathbf{L}\mathbf{I}_s).(\mathbf{x}^T\mathbf{x}_s^T)^T = \mathbf{b} \tag{15}$$

where the $(n_s \times 1)$ vector $\mathbf{x}_s$ consists of *slack variables* and $\mathbf{I}_s$ is the $(s \times s)$ identity matrix. Now, when we force $(\mathbf{L}\mathbf{I}_s)$ instead of $\mathbf{Y}^T$, we obtain $(\mathbf{L}\mathbf{I}_s).(\mathbf{B}^T\mathbf{B}_s^T)^T \stackrel{!}{=} \mathbf{0}$. Hence, the structural matrix and the initial state of the supervisor are

$$\begin{aligned} \mathbf{B}_s &= -\mathbf{L}.\mathbf{B}, \\ \mathbf{x}_s^0 &= \mathbf{b} - \mathbf{L}.\mathbf{x}_0 \end{aligned} \tag{16}$$

where $\mathbf{B}_s = \mathbf{G}_s^T - \mathbf{F}_s$. Then the extended PN model (the original uncontrolled PN model together with the supervisor) has the following structural matrix and the initial state.

$$\begin{aligned} \mathbf{B}_{ex} &= \begin{pmatrix} \mathbf{B} \\ \mathbf{B}_s \end{pmatrix}, \\ \mathbf{x}_{ex}^0 &= \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_s^0 \end{pmatrix}. \end{aligned} \tag{17}$$

The approach can be explained in details by means of the following simple example illustrating the primary problem of RAS – removing deadlocks. There it will be presented what difficulties can deadlocks cause in RAS as well as ways how to deal with them.

3.1.1 Example 1

Consider the very simple PN model in Figure 5 a). The corresponding RT is shown in Figure 5 b). There it can be seen that the state No. 9 (i.e. $\mathbf{x}_9$), being the 10^{th} column of the matrix $\mathbf{X}_r$ in (20) (because the numbering of reachable states starts from 0), represents the deadlock.

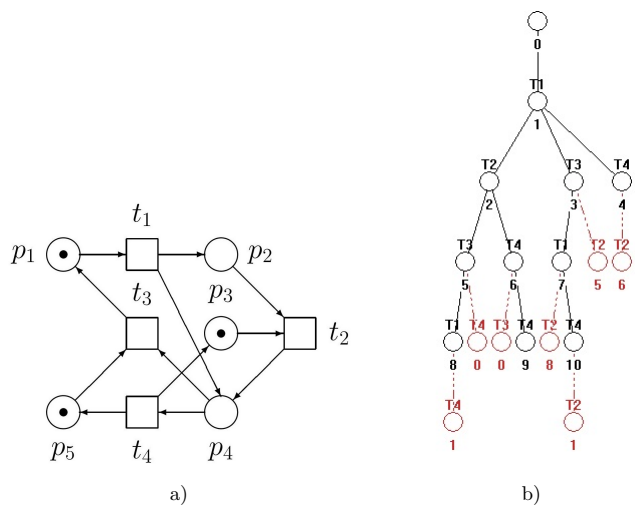
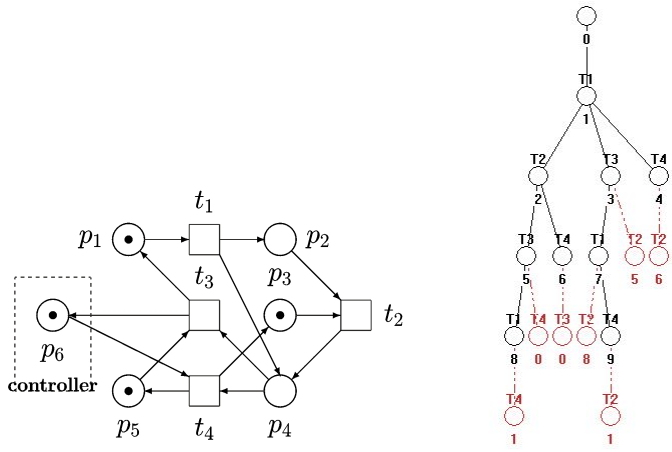


Figure 5. The simple example of a) the deadlocked PN and b) its RT

Therefore, it is necessary to avoid the deadlock. Let us demonstrate removing the deadlock by a supervisor synthesis based on P-invariants.

The P-invariant based approach starts by the thorough analysis of the RT. Doing so we can see that in order to eliminate the deadlock it is necessary to ensure the priority $t_3 \succ t_4$. Namely, it follows from the "fork" emerging from the state No. 2 (i.e. $\mathbf{x}_2$) being the 3^{rd} column in (20), as well as from the "fork" emerging from the state No. 6 (i.e. $\mathbf{x}_6$) being the 7^{th} column in (20).

In this *specific case* it is possible to do this elimination very simply – by adding p_6 to the original PN model and interconnect it with the PN model by the arcs from t_3 to p_6 and from p_6 to t_4. This ensures the priority $t_3 \succ t_4$ in RT given in Figure 5. It is clear from Figure 6 a) and from RT (right). However, the states of the controlled model displayed in Figure 6 as well as the numbers of RT nodes are different from the states and numbers of RT nodes of the uncontrolled model given in Figure 5.



a) b)

Figure 6. The a) supervised PN model and b) its RT

However, the P-invariant based approach analytically described by (13)–(16) is *general one*. The described procedure of the supervisor synthesis presented in [4] can impose the restriction on the state vector $\mathbf{x}$ in the form (14). In our case

$$\mathbf{L} = (0, 0, 1, -1, 0) \qquad \text{and} \qquad \mathbf{b} = (1) \tag{18}$$

where the matrix $\mathbf{L}$ is represented by the row vector and the restriction $\mathbf{b} = (1)$ is the scalar, because only one of two possibilities (firing either the transition t_3 or transition t_4) is possible.

To perform the analytical expression of the approach let us introduce the following. The structural matrix of the original PN model, the initial state $\mathbf{x}_0$ and RT nodes in (20) are as follows:

$$\mathbf{B} = \mathbf{G}^T - \mathbf{F} = \begin{pmatrix} -1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 1 & 1 & -1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}, \quad \mathbf{x}_0 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \tag{19}$$

$$\mathbf{X}_r = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 2 & 1 & 0 & 2 \\ 1 & 1 & 0 & 1 & 2 & 0 & 1 & 1 & 0 & 2 & 2 \\ 0 & 1 & 2 & 0 & 0 & 1 & 1 & 1 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 2 & 0 & 2 & 0 & 0 & 3 & 1 \end{pmatrix}. \tag{20}$$

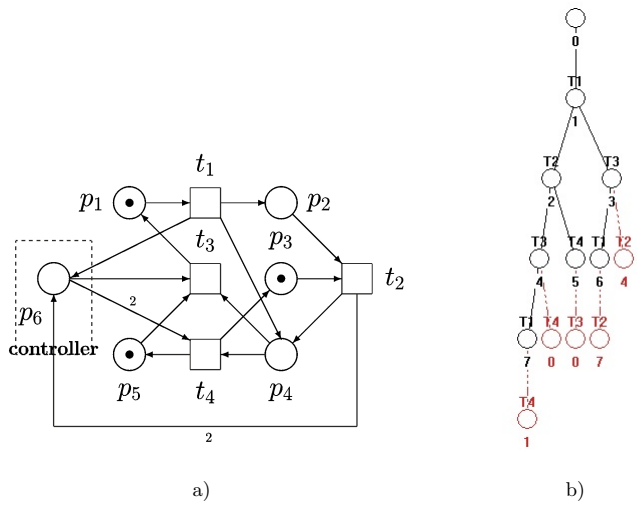
Because of (16), the structural matrix $\mathbf{B}_s$ of the supervisor and the initial state $\mathbf{x}_0^s$ of the supervisor are

$$\mathbf{B}_s = (1, 2, -1, -2) \text{ and } \mathbf{x}_0^s = (0). \tag{21}$$

The structure of the supervisor is

$$\mathbf{B}_s = \mathbf{G}_s^T - \mathbf{F}_s \text{ where } \mathbf{G}_s^T = (1, 2, 0, 0) \text{ and } \mathbf{F}_s = (0, 0, 1, 2). \tag{22}$$

The supervisor has very simple structure – it is represented by the place p_6. Hence, the supervised PN model is given in Figure 7 a). The corresponding RT is shown in Figure 7 b). As we can see in RT, no deadlock is detected. It is necessary to add that the states of the controlled PN in such a way as well as the RT nodes in this case are different from the states and RT nodes in previous two cases.



a) b)

Figure 7. The a) supervised PN by means of P-invariant and b) its RT

3.1.2 Example 2

Let us introduce now a practical example. A cell of AMS/FMS consists of three workstations, W1 with a robot R1, W2 with a robot R2, and W3 with a robot R3, and a single AGV (being served by the robots) that transports parts among the workstations and the input and output ports I/O of the cell. The simple scheme of the cell is displayed in Figure 8.

There is exercised the concurrent production of the two process types with planes as follows – P1: W1 → W2 → W3 and P2: W3 → W2 → W1. P1 produces parts of a kind A while P2 produces parts of a kind B. Each workstation has a working table

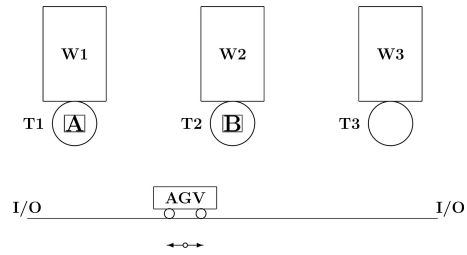


Figure 8. The example of the real RAS with the deadlock

on which only one workpiece at a time can be held. It is evident from the routing information about processes P1, P2 that none of the currently loaded parts is able to go forward to the next workstation, because the corresponding working table is occupied by the other part. This situation illustrate the deadlock that can be met in this AMS/FMS. Simultaneously, the Figure 8 can be understood to be a kind of RAS.

The PN model of the cell and its RT are given in Figure 9. Its parameters are

$$\mathbf{B} = \mathbf{G}^T - \mathbf{F} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 & -1 & 1 & 0 & 0 \end{pmatrix}, \ \mathbf{x}_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad (23)$$

The places p_1, p_2, p_3 express the presence of parts A on the working tables, while p_4, p_5, p_6 express the presence of parts B on the working tables. The resource availability is monitored by the marking of the resource places p_7, p_8, p_9.

As we can also see on the PN model of the RAS in Figure 9 a), and especially on RT in Figure 9 b), there is the deadlock in the state $\mathbf{x}_1 = (1\,0\,0\,1\,1\,0\,0\,0\,0)^T$. It occurs immediately after firing t_5 at the initial state $\mathbf{x}_0 = (1\,0\,0\,0\,1\,0\,0\,0\,1)^T$.

To control the system consider the natural initial state $\mathbf{x}_0 = (0\,0\,0\,0\,0\,0\,1\,1\,1)^T$ when all resources are available at the beginning. The corresponding RT is given in Figure 10.

As we can see there, the states $\mathbf{x}_{13} = (1\,1\,0\,1\,0\,0\,0\,0\,0)^T$ and $\mathbf{x}_{14} = (1\,0\,0\,1\,1\,0\,0\,0\,0)^T$, i.e., the 14$^{\text{th}}$ and 15$^{\text{th}}$ column of the following matrix of reachable states $\mathbf{X}_r$ are deadlocks. All nodes of the RT, where the first column represents the initial

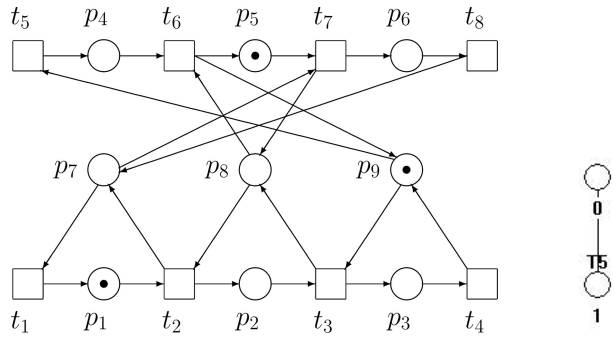


Figure 9. The a) PN model and b) corresponding RT of the RAS

state $\mathbf{x}_0$, are expressed by particular columns as follows:

$$\mathbf{X}_r = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & |1| & |1| & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & |1| & |0| & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & |0| & |0| & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & |1| & |1| & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & |0| & |1| & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & |0| & |0| & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & |0| & |0| & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & |0| & |0| & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & |0| & |0| & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \quad (24)$$

3.1.3 Detail Analysis of RT

The detail analysis of RT yields the following results:

1. the sequences of firing $t_2 \succ t_5$ as well as $t_5 \succ t_2$ have to be forbidden, i.e., p_2 and p_4 cannot be active simultaneously;
2. the sequences of firing $t_1 \succ t_6$ and $t_6 \succ t_1$ have to be forbidden, i.e., p_1 and p_5 cannot be active simultaneously;
3. the sequences of firing $t_1 \succ t_5$ as well as $t_5 \succ t_1$ have to be forbidden, i.e., p_1 and p_4 cannot be active simultaneously.

Thus, putting inequalities $\sigma_{p_2} + \sigma_{p_4} \leq 1$, $\sigma_{p_1} + \sigma_{p_5} \leq 1$, and $\sigma_{p_1} + \sigma_{p_4} \leq 1$ we can synthesize the supervisor. As we can see below, these places create the nonzero entries of the matrix $\mathbf{L}$, and right sides of the inequalities create the vector $\mathbf{b}$. Putting the initial state of the PN model of RAS as $\mathbf{x}_0 = (0\,0\,0\,0\,0\,0\,1\,1\,1)^T$ (i.e., when all three resources are available), the process of the supervisor synthesis is the following.

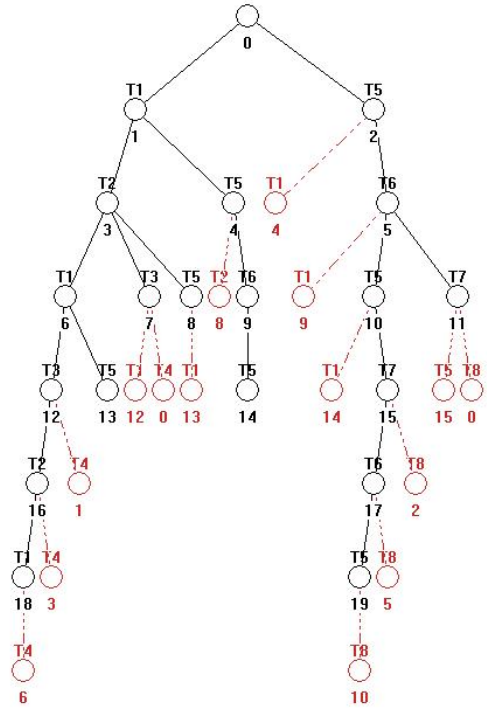


Figure 10. The corresponding RT of the RAS model

3.1.4 Supervisor Synthesis

The anterior inequalities create the restrictive condition with $\mathbf{L}$ and $\mathbf{b}$ in the form

$$\mathbf{L} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \tag{25}$$

$$\mathbf{B}_s = -\mathbf{L}.\mathbf{B} = \begin{pmatrix} 0 & -1 & 1 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & -1 & 1 & 0 \\ -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{x}_s^0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{26}$$

Here, the positive nonzero entries of $\mathbf{B}_s = \mathbf{G}_s^T - \mathbf{F}_s$ create nonzero entries of $\mathbf{G}_s^T$, while negative nonzero entries of $\mathbf{B}_s$ create nonzero entries of $(-\mathbf{F}_s)$.

The PN model of the supervised system is displayed in Figure 11 where the controller is represented by the triplet $\{p_{10}, p_{11}, p_{12}\}$. RT of such system is given in Figure 12.

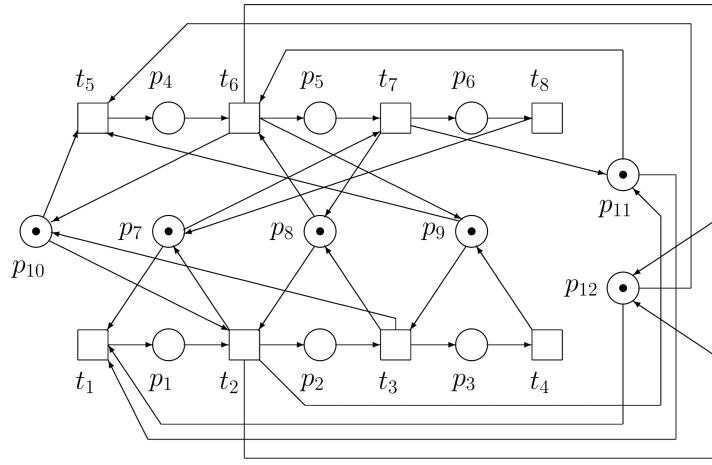


Figure 11. The controlled PN model

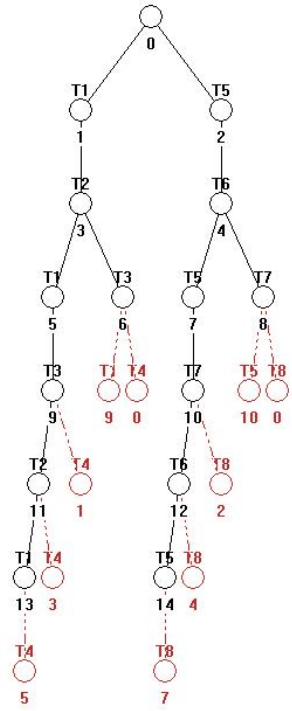


Figure 12. The RT of the controlled PN model

Of course, the incidence matrices and the initial state of the controlled system are

$$\mathbf{F}_{cs} = \begin{pmatrix} \mathbf{F} \\ \mathbf{F}_s \end{pmatrix}, \quad \mathbf{G}_{cs}^T = \begin{pmatrix} \mathbf{G}^T \\ \mathbf{G}_s^T \end{pmatrix}, \quad \mathbf{x}_{cs}^0 = \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_s^0 \end{pmatrix} \tag{27}$$

where $\mathbf{F}$, $\mathbf{G}^T$ are incidence matrices of the uncontrolled RAS and $\mathbf{x}_0$ is its initial state.

The RT of the controlled RAS is given in Figure 12. No deadlock can be seen in Figure 12. The particular nodes of this RT are expressed by the columns of the matrix

$$\mathbf{X}_r = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \tag{28}$$

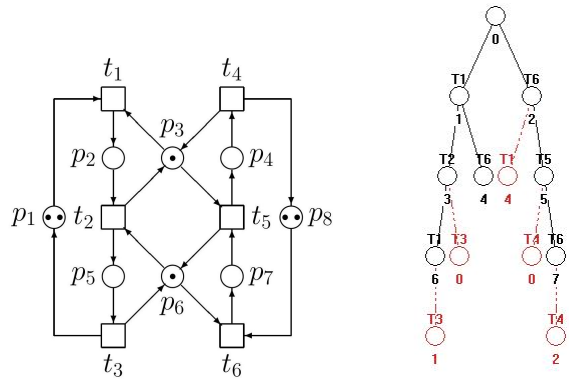
where the first column represents the initial state $\mathbf{x}_{cs}^0$ of the controlled RAS.

3.1.5 Example 3

Let us add a simple example relating the deadlocked S^{3}PR PN model. The problem of a deadlock in S^{3}PR can be resolved also by means of adding a transition to the original deadlocked PN model of RAS. Namely, imbedding such a transition into the deadlocked PN model it is possible to achieve initial state from the deadlocked state. Then, the development of the model can proceed from the initial state in some other way. It was proved in [83] for S^{3}PR kind of deadlocked PN model. Applying this on (6), the single additional transition brings the structural matrix of the supervisor $\mathbf{B}_s = \mathbf{x}_0 - \mathbf{x}_q$. Consider the deadlocked PN model given in Figure 13 a) having RT displayed in Figure 13 b) where the RT node No. 4 (i.e. $\mathbf{x}_5$) is the deadlock in question.

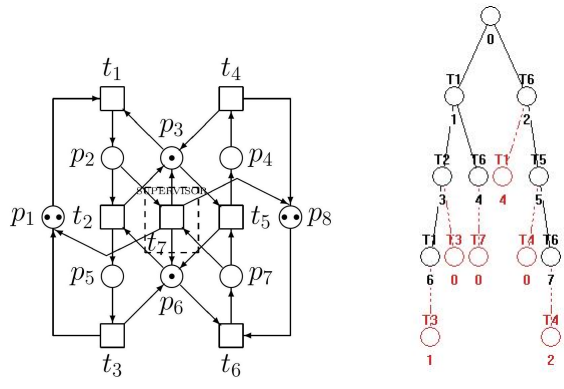
Because $\mathbf{x}_0 = (2,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 2)^T$ and $\mathbf{x}_5 = (1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1)^T$, after calculation $\mathbf{B}_s = (1,\ -1,\ 1,\ 0,\ 0,\ 1,\ -1,\ 1)^T$. Thus, $\mathbf{F}_s = (0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0)^T$ (arcs from places to the transition) and $\mathbf{G}_s^T = (1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1)^T$ (arcs from the transition to places). Hence, the supervised PN model without the deadlock is given in Figure 14 a) and its RT is displayed in Figure 14 b).

Sometimes such a procedure can be used in more complicated RT having so called *diamond(s)* between two different nodes. Even, the paths (left and right)



a) b)

Figure 13. The a) PN model of RAS with the deadloch $\mathbf{x}_5$ and b) its RT



a) b)

Figure 14. The a) supervised PN model by means of the added transition t_7 and b) its RT

creating the diamond(s) may be longer than that in RT displayed in Figure 14 a) and Figure 14 b). More about this will be said in Part 2 of this paper.

3.1.6 Local Conclusion

The approach based on P-invariants has an exact analytically expressed procedure. After thorough analysis of RG and finding conditions how to mutually eliminate states of relevant places (in order to eliminate deadlocks), the procedure of the supervisor synthesis is very clear and simple. Introduced examples illustrate that the deadlock problem can be resolved relatively simply. However, in a more complicated

structure of the PN model of RAS with RT having too patulous branching, just the choice of the restrictions in the form of inequalities may be complicated or even impossible. Therefore, it is also necessary to look for another approaches or combinations of them.

3.2 Approach Based on Siphons Without the Need to Know RG

Based on the previous findings, especially those being introduced in Section 2 (Preliminaries), we have to unconditionally control siphons. Of course, first of all we have to know them, i.e., we have to compute them. There are several algorithms how to do this. Although the calculation of siphons is not a subject of this paper, let us mention at least several approaches to it by means of:

1. solving logical equations, namely a siphon S has to satisfy a set of conditions: $\forall t_i \in T : t_i^{\bullet} \cap S \neq \varnothing \Rightarrow {}^{\bullet}t_i \cap S \neq \varnothing$ – see e.g. [44];
2. linear algebraic calculation – see e.g. [45];
3. Thelen's prime implicant method – see [90];

and some others. However, for working in Matlab the most useful seems to be the GPenSIM tool developed by Davidrajuh [12, 13] for PN, which is able to calculate (among other things) also siphons and traps as well as minimal siphons and minimal traps.

The problem of deadlock avoidance in DES is equivalent to the problem of the avoidance of empty siphons in the original ordinary PN model. The siphon based control of a deadlocked PN has to guarantee that none of its siphons ever becomes empty. Unfortunately, this approach does not have as much analytical support as in the first approach based on P-invariants. Therefore, it is necessary to work with graphical tools for the PN modelling and analyzing, more than in the first approach. The siphon behaviour is such that if it has no token in a state (marking) of PN, then it remains without any token in each successor state. The trap behaviour is such that if it has at least one token in a state (marking) of PN, then it remains marked under each successor state.

It can be said that a siphon can only *lose* tokens whereas a trap can only *gain* tokens. Therefore, arising out of these properties, we want to utilize siphons and traps for analyzing of PN liveness as well as for synthesizing supervisors in order to avoid deadlocks in PN models.

Siphons are tied with deadlocks especially in PN models of RAS. As it was already mentioned, once a siphon loses all its tokens, it remains unmarked at any subsequent markings that are reachable from the current marking – see e.g. [22]. If a siphon is emptied at a certain marking, some of its output transitions would never be enabled. This leads to a deadlock. There exist many papers about deadlock prevention which have been based on siphons. They especially add monitors (additional places) to the PN model for strict minimal siphons in order to achieve

deadlock prevention. In this paper using the minimal siphons will be illustrated on simple examples.

On the other hand, there exist newer papers – e.g. [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53]. They bring very interesting contributions in the form of new approaches how to deal with deadlocks by means of siphons, as it already was mentioned in Section 2.5. Some of them utilize elementary siphons. These newer approaches will be analyzed in Part 2 of this paper, which is being prepared.

A siphon S is said to be controlled in a net (N, M_0) *iff* $\forall M \in R(N, M_0)$, $M(S) > 0$. Hence, any siphon that contains a marked trap is controlled, since the marked trap can never be emptied. In an ordinary Petri net, a siphon that is controlled does not cause any deadlock.

Siphon S in an ordinary net system (N, M_0) is [51] invariant-controlled by P-invariant I under M_0 *iff* $L^T.M_0 > 0$ and $\forall p \in P \backslash S, I(p) \leq 0$, or equivalently, $l^T.M_0 > 0$ and $||I||^+ \subseteq S$. Here $||I||^+ = p \in P \mid I(p) > 0$ is the positive support of P-vector I.

Briefly, a siphon S is said to be controlled if it can never be emptied, and it is said to be invariant-controlled by P-invariant I if $l^T.M_0 > 0$ and $||I||^+ \subseteq S$.

The problem of deadlock avoidance in DES is equivalent [88, 89] to the problem of avoidance of empty siphons in ordinary PN model of DES. This is very important especially in real AMS/FMS. Siphons that do not contain other siphons are named as minimal siphons. It is sufficient to consider only minimal siphons at the supervisor synthesis. Hence, it is necessary to ensure that the sum of the number of tokens in each minimal siphon S is never less than one in any reachable marking. Thus, the general condition for i^{th} siphon $\mathbf{S}_i.\mathbf{x} \geq b$ proceeds into the form $\mathbf{S}_i.\mathbf{x} \geq 1$.

Generally, the main purpose of control of DES by means of PN is to avoid *undesirable* or *illegal* markings. In [42, 43, 88] an appropriate formal specification

$$\mathbf{l}^T.\mathbf{x} \geq b \quad \text{i.e.} \quad b - \mathbf{l}.\mathbf{x} \leq 0 \tag{29}$$

is proposed where $\mathbf{l}$ is a $(s \times 1)$-dimensional weight row vector; $\mathbf{x}$ is a state vector (i.e. marking); and b is a scalar. Verbally it means that the weighted sum of the number of tokens in each place should be greater than or equal to a constant. The theorem was proved there that if a PN with incidence matrix $\mathbf{B}$ satisfies $b - \mathbf{l}^T.\mathbf{x}_0 \leq 0$, then a control place p_c can be added which enforces the previous inequality (29). When $\mathbf{b}_c : T \to \mathbb{Z}$ denotes the weight vector of arcs connecting p_c with the transitions in the PN, the $\mathbf{b}_c$ can be obtained by

$$\mathbf{b}_c = \mathbf{l}^T.\mathbf{B} \tag{30}$$

and the initial number of tokens in p_c is

$$\mathbf{x}_0(p_c) = \mathbf{l}^T.\mathbf{x}_0 - b \geq 0. \tag{31}$$

The control place p_c enforces maximally permissive control strategy (or logic). It means that the only reachable markings of the original net N, that p_c avoids, are those violating [42, 43]. Here, the $\mathbf{b}_c$ is the row extending the matrix $\mathbf{B}$ with respect to p_c.

In general (for more additive places $p_1, \ldots, p_s$) it can be written the following

$$\mathbf{L}.\mathbf{x} \geq \mathbf{b} \quad \text{or} \quad \mathbf{L}.\mathbf{x} - \mathbf{x}_c = \mathbf{b} \tag{32}$$

where $\mathbf{L} \in \mathbb{Z}_{\geq 0}^{s \times n}$ having the weighted vectors $\mathbf{l}_i^T$, $i = 1, \ldots, s$ as its rows; $\mathbf{b} \in \mathbb{Z}_{\geq 0}^{s \times 1}$ is the vector of restrictions and $\mathbf{x}_c \in \mathbb{Z}_{\geq 0}^{s \times 1}$ is the vectors of slacks. We can found the extended incidence matrix $\mathbf{B}_{ex}$ and the initial state of extended state vector $\mathbf{x}_{ex}^0$ as follows:

$$\mathbf{B}_c = \mathbf{L}.\mathbf{B}, \tag{33}$$

$$\mathbf{x}_c^0 = \mathbf{L}.\mathbf{x}_0 - \mathbf{b} \stackrel{!}{\geq} \mathbf{0}, \tag{34}$$

$$\mathbf{B}_{ex} = \begin{pmatrix} \mathbf{B} \\ \mathbf{B}_c \end{pmatrix}, \quad \mathbf{x}_{ex}^0 = \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_c^0 \end{pmatrix} \tag{35}$$

where $\mathbf{B}_c = \mathbf{G}_c^T - \mathbf{F}_c$ is the matrix corresponding to s additive places (monitors) $p_1, \ldots, p_s$.

Putting $\mathbf{l}_i^T \stackrel{!}{=} S_i$, where S_i is the i^{th} PN siphon, or in general $\mathbf{L} \stackrel{!}{=} \mathbf{S}_m$ where $\mathbf{S}_m$ is the matrix of all PN siphons (being its rows), we have the structure of the supervisor as follows $\mathbf{B}_c = \mathbf{S}_m.\mathbf{B}$.

3.2.1 Example 4

Consider the same PN model given in Figure 5. In the PN siphon-based approach we have to find siphons and traps.

Minimal siphons in this net are $\{p_1, p_3, p_4\}$, $\{p_1, p_2, p_4\}$ and in the matrix form:

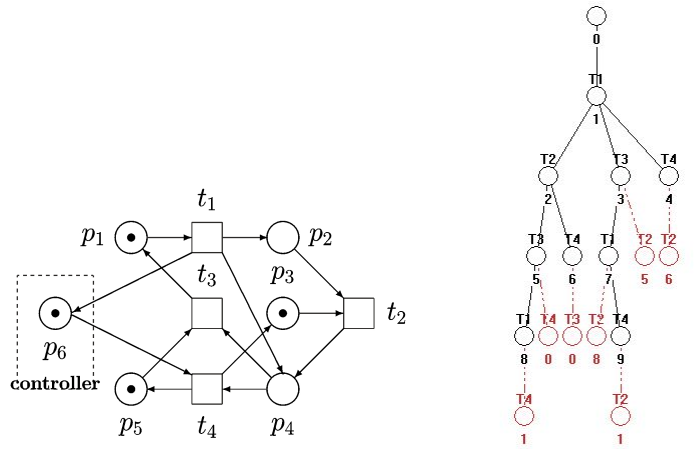
$$\mathbf{S}_m = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}. \tag{36}$$

Minimal traps in this net are $\{p_1, p_3, p_4\}$, $\{p_1, p_4, p_5\}$ and in the matrix form:

$$\mathbf{T}_m = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{37}$$

We can see that the first siphon is equal to the first trap. This is also clear from comparing first rows of the matrices $\mathbf{S}_m$ and $\mathbf{T}_m$. Such a siphon is out of our interest because it cannot be emptied once it is initially marked. Namely, this siphon contains the marked trap, i.e., there is no deadlock threat.

Now, we have to consider the second siphon $\{p_1, p_2, p_4\}$. We can see from $\mathbf{X}_r$ in Example 1 (see (20)) that for $S = (1\,1\,0\,1\,0)$ no marking contains $M(S) > 0$.



a) b)

Figure 15. The a) supervised PN by means of siphons and b) its RT

The siphon contains neither the marked trap, because the trap is $T = (1\,0\,0\,1\,1)$. Consequently, the siphon is not controlled. Hence, it is necessary to control it in order to avoid the deadlock. Putting $l \stackrel{!}{=} S$, or in general $\mathbf{L}^T \stackrel{!}{=} \mathbf{S}_m$, we have the structure of the supervisor as follows

$$\mathbf{B}_c = \mathbf{S}_m.\mathbf{B} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 1 & 1 & -1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}. \tag{38}$$

Because the first row of $\mathbf{B}_c$ is the zero vector, the equation implies that there is only one place p_c, namely p_6 in Figure 15, that will represent the supervisor/controller. Thus, incidence matrices of the supervisor are as follows:

$$\mathbf{F}_c = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{G}_c^T = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}. \tag{39}$$

The reachable states of the controlled (supervised) PN model are the following

$$\mathbf{X}_c = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 2 & 1 & 0 & 2 \\ 1 & 1 & 0 & 1 & 2 & 0 & 1 & 1 & 0 & 2 & 2 \\ 0 & 1 & 2 & 0 & 0 & 1 & 1 & 1 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 2 & 0 & 2 & 0 & 0 & 3 & 1 \\ 1 & 2 & 2 & 2 & 1 & 2 & 1 & 3 & 3 & 0 & 2 \end{pmatrix} \tag{40}$$

where the columns represent state vectors $\mathbf{x}_0 \ldots \mathbf{x}_{10}$ being nodes of the RT.

The controllability of siphon S is ensured by adding a monitor. Namely, the number of tokens leaving S is limited by a marking invariant law being implemented by a P-invariant whose *support* contains the monitor.

The supervised PN model is given in Figure 15 a). The corresponding RT is shown in Figure 15 b) in order to see that no deadlock occurs in the supervised system.

By the way, the controlled system in Figure 15 a) is the same as that in Figure 6 a) obtained by the prejudged relation $t_3 \succ t_4$.

3.2.2 Example 6

In Subsection 3.1.2 (Example 2) the supervisor eliminating deadlocks was proposed. Consider here the same deadlocked PN model and let us use the siphon-based approach to solve the problem. The PN model contains the following minimal siphons and traps:

$$
\begin{aligned}
S_1 &= \{p_3,\, p_4,\, p_9\},\\
Tr_1 &= \{p_3,\, p_4,\, p_9\},\\
S_2 &= \{p_2, p_5, p_8\},\\
Tr_2 &= \{p_2,\, p_5,\, p_8\},\\
S_3 &= \{p_1,\, p_6,\, p_7\},\\
Tr_3 &= \{p_1,\, p_6,\, p_7\},\\
S_4 &= \{p_3,\, p_5,\, p_8,\, p_9\},\\
Tr_4 &= \{p_2,\, p_4,\, p_8,\, p_9\},\\
S_5 &= \{p_2,\, p_6,\, p_7,\, p_8\},\\
Tr_5 &= \{p_1,\, p_5,\, p_7,\, p_8\},\\
S_6 &= \{p_3, p_6, p_7, p_8, p_9\},\\
Tr_6 &= \{p_1, p_4, p_7, p_8, p_9\}.
\end{aligned}
$$

or in the matrix form

$$
\mathbf{S}_m = \begin{pmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1\\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0\\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0\\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1\\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0\\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1
\end{pmatrix}, \quad
\mathbf{T}_m = \begin{pmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1\\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0\\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0\\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1\\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0\\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1
\end{pmatrix}. \quad (41)
$$

The first three siphons are equal to the first three traps.

As it can be seen from (41), the places p_2, p_6, p_7, p_8 of the siphon S_5 (the 5^{th} row of $\mathbf{S}_m$) are active in several state vectors of the PN model – see the corresponding rows 2, 6, 7, 8 of this matrix (24).

As we can see in Figure 9, places p_2, p_6, p_7, p_8 included in the siphon S_5, create even the empty siphon (as to marking). All the output transitions of S_5 are $S_5^\bullet = \{t_2, t_3, t_6, t_7, t_8\}$. The transition t_1 does not belong in $S_5^\bullet$ because it is the *source* transition (generating siphons) and it does not fall with the siphon definition. By the way, t_5 is also the *source* transition. All actual output transitions of $S_5^\bullet$ are disabled, since they require at least one token from some place in S_5.

All of input transitions of S are ${}^\bullet S_5 = \{t_2, t_3, t_6, t_7, t_8\}$.

Even, any transition which could bring tokens in S_5 is a part of ${}^\bullet S_5$, and consequently is disabled. Consequently, S_5 will remain empty during entire system dynamics evolution as well as the transitions in $S_5^\bullet$ will be dead during this evolution. Alike, we could analyze also other nonzero siphons S_4, S_6. Such a work is too toilsome.

It means that it is necessary to find a way how to deliver tokens into the siphon places. Therefore, let us compute the monitors that will make this for us.

Put $\mathbf{L} \stackrel{!}{=} \mathbf{S}_m$. Then,

$$\mathbf{B}_c = \mathbf{S}_m.\mathbf{B} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & -1 & 1 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \end{pmatrix}. \tag{42}$$

Excluding the upper zero sub-matrix we obtain the controller with the incidence matrices

$$\mathbf{F}_c = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \quad \mathbf{G}_c^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{43}$$

The supervised PN model of RAS and its RT are given in Figure 16 a) and 16 b), respectively. The supervisor consists of the imbedded places (monitors) A, B, C. RT is not necessary in this approach but it is included in order to see that no deadlock occurs in the supervised RAS.

The nodes of RT are the rows of the following matrix where the first row is the initial state vector $\mathbf{x}_0$. No state vector represents a deadlock. The nodes of the RT are the state vectors being rows of the following matrix

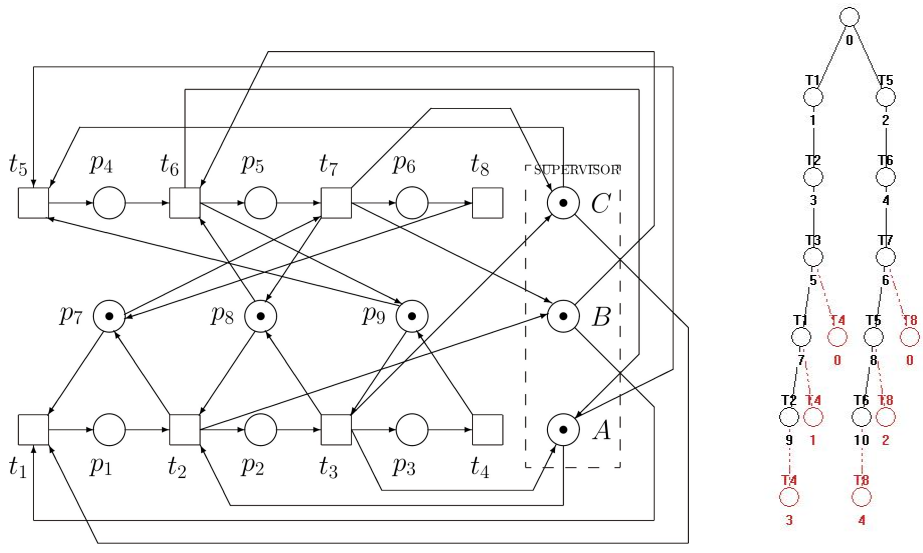


Figure 16. The PN model of the supervised RAS

$$\mathbf{X}_r^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}. \tag{44}$$

3.2.3 Local Conclusion

The approach based on siphons has not so exact analytically expressed procedure (especially in case of siphons and traps computation) like the previous approach based on P-invariants and analysis of RT. Firstly, it is necessary to analyze the PN model of RAS and find its siphons and traps. There are Matlab tools for calculation of siphons and traps – see e.g. [12, 13]. After thorough analysis of the set of siphons and the set of traps, the procedure of the supervisor synthesis can start on. The analytical approach to computation of monitors is utilized. Then, the procedure is

also clear and simple as well as in case of the previous approach. The analysis of RT is not necessary in this case. However, it is possible to generate RT in order to be sure that the controlled PN model of RAS is deadlock-free.

3.3 A Short Comparison of Both Approaches

When we compare both approaches applied on the relatively simple cases of deadlocked RAS, we can say the following:

1. the weak point (shortcomming, weakness) of the first approach consists in the computational demands at finding RT/RG and the labor consumption at finding the conditions for elimination deadlocks. In case of largely branched RT computing takes a very long time. The dependance on the initial state is also a weakness;
2. the weak point of the second approach consists in computational demands at finding siphons. This may also takes a very long time.

The first approach yields the supervisor which restricts the development of the system a little less, than the supervisor synthesized by the second approach. On the other hand, there is a perspective of finding new and new methods in the second approach.

However, because from several simple DES and RAS it is impossible to draw serious conclusions, we will do this in Part 2 (may be also in Part 3) of this paper, which is being prepared. There, not only the more complicated examples will be tested by both approaches, and then compared each other, but also newer approaches will be analyzed, especially some of those published in [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53].

4 FURTHER RESEARCH IN THE FUTURE

In this paper, the problem of deadlock avoiding in RAS was presented by means of two different approaches. Relatively simple examples, but one of them being practical, were introduced to illustrate the applicability of these approaches. In the further research, we will continue in solving the problem with deadlocks in RAS, especially of S^3PR and S^4PR kinds, investigated in [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53]. Both approaches (first based on P-invariants and second based on siphons) will be tested on the same more complicated practical examples. The effectiveness of finding their results will be mutually compared and evaluated.

Moreover, a deeper view on admissibility of deadlock-free control of RAS will be performed. Namely, there exist following kinds of PN states (markings) [65]:

1. legal;

2. illegal, or else forbidden;

3. admissible,

i.e., such from which the system cannot uncontrollably reach illegal state. In PN which are controllable only partially, an illegal state may be reachable from a legal one by firing of uncontrollable transitions (see e.g. [3, 4, 5, 6, 65, 66, 67, 68, 69, 53, 71, 85]). Consequently, it is necessary to find more restrictive control policy which will enforce a subset of legal states, i.e. admissible states. From such states the system cannot uncontrollably reach an illegal state. However, this topic will be analyzed later, probably only in Part 3 of this paper (because of the limited space for one paper in this journal).

5 CONCLUSION

PN are a formal modelling tool. They are a popular mathematical formalism to investigate and analyze modelling and control of DES. PN theory has been one of the most interesting topics in computer science. PN find wide application in contemporary technical systems, especially in AMS/FMS where there is also a mathematical framework to investigate the deadlock control problems in a variety of RAS. In such a way PN become the effective tool for the design and management of modern AMS/FMS. In the last years the siphon-based approach has dominated among the methodologies dealing with the deadlock analysis and control of RAS – see the new research especially in [63, 8, 25, 55, 56, 57, 59, 62, 64, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 92, 93, 94, 91, 88, 89, 65, 66, 67, 68, 69, 58, 60, 61, 47, 48, 49, 50, 51, 52, 53]. Although the approach to the same problem based on P-invariants is a little older, it does not lag behind as to the quality of its results (sometimes more to the contrary).

In this paper, both approaches were presented and illustrated by examples. First on simple examples and then on more complicated ones related to real RAS. The total number of introduced examples was 6.

Each approach was evaluated in the particular local conclusion (see Subsection 3.1.6 and Subsection 3.2.3). It was demonstrated that both approaches are suitable for the deadlock elimination very well. The detailed comparison of both approaches on the same more complicated examples will be performed in the planned Part 2 (may be only Part 3) of this paper to be published later.

Acknowledgement

The author thanks for the partial support of the VEGA Agency (under Grant No. 2/0020/21).

REFERENCES

[1] Barkaoui, K.—Couvreur, J. M.—Dutheillet, C.: On the Liveness in Extended Non Self-Controlling Nets. In: De Michelis, G., Diaz, M. (Eds.): Application and Theory of Petri Nets 1995 (ICATPN 1995). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 935, 1995, pp. 25–44, doi: 10.1007/3-540-60029-9_32.

[2] Barkaoui, K.—Pradat-Peyre, J. F.: On Liveness and Controlled Siphons in Petri Nets. In: Billington, J., Reisig, W. (Eds.): Application and Theory of Petri Nets 1996 (ICATPN 1996). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1091, 1996, pp. 57–72, doi: 10.1007/3-540-61363-3_4.

[3] Čapkovič, F.: Petri Nets at Modelling and Control of Discrete-Event Systems Containing Nondeterminism – Part 1. Computing and Informatics, Vol. 37, 2018, No. 5, pp. 1258–1292, doi: 10.4149/cai_2018_5_1258.

[4] Čapkovič, F.: Petri Nets at Modelling and Control of Discrete-Event Systems Containing Nondeterminism – Part 2. Computing and Informatics, Vol. 38, 2019, No. 3, pp. 728–764, doi: 10.31577/cai_2019_3_728.

[5] Čapkovič, F.: Modeling and Control of Discrete-Event Systems with Partial Non-Determinism Using Petri Nets. Acta Polytechnica Hungarica, Vol. 17, 2020, No. 4, pp. 47–66, doi: 10.12700/APH.17.4.2020.4.3.

[6] Čapkovič, F.: Timed and Hybrid Petri Nets at Solving Problems of Computational Intelligence. Computing and Informatics, Vol. 34, 2015, No. 4, pp. 746–778.

[7] Chao, D. Y.—Pan, Y. L.: Uniform Formulas for Compound Siphons, Complementary Siphons and Characteristic Vectors in Deadlock Prevention of Flexible Manufacturing Systems. Journal of Intelligent Manufacturing, Vol. 26, 2015, No. 1, pp. 13–23, doi: 10.1007/s10845-013-0757-7.

[8] Chao, D. Y.: Max'-Controlled Siphons for Liveness of S^3PGR^2. IET Control Theory and Applications, Vol. 1, 2007, No. 4, pp. 933–936, doi: 10.1049/iet-cta:20060275.

[9] Chen, Y. F.—Li, Z. W.: Design of a Maximally Permissive Liveness-Enforcing Supervisor with a Compressed Supervisory Structure for Flexible Manufacturing Systems. Automatica, Vol. 47, 2011, No. 5, pp. 1028–1034, doi: 10.1016/j.automatica.2011.01.070.

[10] Chen, Y. F.—Li, Z. W.: Optimal Supervisory Control of Automated Manufacturing Systems. 1st Edition. CRC Press, 2012, doi: 10.1201/b14588.

[11] Chu, F.—Xie, X. L.: Deadlock Analysis of Petri Nets Using Siphons and Mathematical Programming. IEEE Transactions on Robotics and Automation, Vol. 13, 1997, No. 6, pp. 793–804, doi: 10.1109/70.650158.

[12] Davidrajuh, R.: GPenSIM, General Purpose Petri Net Simulator for MATLAB Platform. Available at: `http://www.davidrajuh.net/gpensim/`.

[13] Davidrajuh, R.: General Purpose Petri Net Simulator GPenSIM. Version 9.0. University of Stavanger, Norway, 2014. Available at: `http://www.davidrajuh.net/gpensim/v9/GPenSIM_v9_User_Manual.pdf`.

[14] Diaz, M. (Ed.): Petri Nets: Fundamental Models, Verification and Applications. John Wiley and Sons, 2009, doi: 10.1002/9780470611647.

[15] Dijkstra, E. W.: Cooperating Sequential Processes. Technical Report, Technological University, Eindhoven, Netherlands, 1965.

[16] Dijkstra, E. W.: Cooperating Sequential Processes. In: Genuys, F. (Ed.): Programming Languages: NATO Advanced Study Institute. Lectures at the Summer School, Villard-le-Lans, 1966, pp. 43–112, Academic Press Inc., London, 1968.

[17] Desel, J.—Reisig, W.: Place/Transition Petri Nets. In: Reisig, W., Rozenberg, G. (Eds.): Lectures on Petri Nets I: Basic Models (ACPN 1996). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1491, 1998, pp. 122–173, doi: 10.1007/3-540-65306-6_15.

[18] Ezpeleta, J.—Colom, J. M.—Martinez, J.: A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems. IEEE Transaction on Robotics and Automation, Vol. 11, 1995, No. 2, pp. 173–184, doi: 10.1109/70.370500.

[19] Fanti, M. P.—Maione, B.—Mascolo, S.—Turchiano, B.: Event-Based Feedback Control for Deadlock Avoidance in Flexible Production Systems. IEEE Transaction on Robotics and Automation, Vol. 13, 1997, No. 3, pp. 347–363, doi: 10.1109/70.585898.

[20] Fanti, M. P.—Maione, B.—Turchiano, B.: Comparing Digraph and Petri Net Approaches to Deadlock Avoidance in FMS. IEEE Transaction on Systems, Man, and Cybernetics, Part B (Cybernetics), Vol. 30, 2000, No. 5, pp. 783–798, doi: 10.1109/3477.875452.

[21] Farooq, A.—Huang, H.—Wang, X. L.: Petri Net Modeling and Deadlock Analysis of Parallel Manufacturing Processes with Shared-Resources. Journal of Systems and Software, Vol. 83, 2010, No. 4, pp. 675–688, doi: 10.1016/j.jss.2009.11.705.

[22] Guan, X.—Li, Y.—Xu, J.—Wang, C.—Wang, S.: A Literature Review of Deadlock Prevention Policy Based on Petri Nets for Automated Manufacturing Systems. International Journal of Digital Content Technology and Its Applications (JDCTA), Vol. 6, 2012, No. 21, pp. 426–433, doi: 10.4156/jdcta.vol6.issue21.48.

[23] Hernández-Flores, E.—López-Mellado, E.—Ramírez-Treviño, A.: Diagnosability Analysis of Partially Observable Deadlock-Free Petri Nets. Proceedings of the 3rd International Workshop on Dependable Control of Discrete Systems (DCDS'11), Saarbrücken, Germany, 2011, pp. 174–179, doi: 10.1109/dcds.2011.5970337.

[24] Hou, Y. F.—Barkaoui, K.: Deadlock Analysis and Control Based on Petri Nets: A Siphon Approach Review. Advances in Mechanical Engineering, Vol. 9, 2017, No. 5, pp. 1–30, doi: 10.1177/1687814017693542.

[25] Hou, Y. F.—Zhao, M.—Liu, D.—Hong, L.: An Efficient Siphon-Based Deadlock Prevention Policy for a Class of Generalized Petri Nets. Discrete Dynamics in Nature and Society, Vol. 2016, Art. No. 8219424, 12 pp., doi: 10.1155/2016/8219424.

[26] Hu, W. S.—Zhu Y. Y.—Lei, J.: The Detection and Prevention of Deadlock in Petri Nets. Physics Procedia, Vol. 22, 2011, pp. 656–659, doi: 10.1016/j.phpro.2011.11.102.

[27] Hu, H.—Liu, Y.—Yuan, L.: Supervisor Simplification in FMSs: Comparative Studies and New Results Using Petri Nets. IEEE Transactions on Control Systems Technology, Vol. 24, 2016, No. 1, pp. 81–95, doi: 10.1109/TCST.2015.2420619.

[28] Hu, H. S.—Zhou, M. C.—Li, Z. W.: Liveness Enforcing Supervision of Video Streaming Systems Using Non-Sequential Petri Nets. IEEE Transactions on Multimedia, Vol. 11, 2009, No. 8, pp. 1457–1465, doi: 10.1109/TMM.2009.2032678.

[29] Hu, H. S.—Zhou, M. C.—Li, Z. W.: Algebraic Synthesis of Timed Supervisor for Automated Manufacturing Systems Using Petri Nets. IEEE Transactions on Automation Science and Engineering, Vol. 7, 2010, No. 3, pp. 549–557, doi: 10.1109/TASE.2009.2037825.

[30] Hu, H. S.—Zhou, M. C.—Li, Z. W.: Low-Cost and High-Performance Supervision in Ratio-Enforced Automated Manufacturing Systems Using Timed Petri Nets. IEEE Transactions on Automation Science and Engineering, Vol. 7, 2010, No. 4, pp. 933–944, doi: 10.1109/TASE.2010.2046412.

[31] Hu, H. S.—Zhou, M. C.—Li, Z. W.: Supervisor Design to Enforce Production Ratio and Absence of Deadlock in Automated Manufacturing Systems. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, Vol. 41, 2011, No. 2, pp. 201–212, doi: 10.1109/TSMCA.2010.2058101.

[32] Hu, H. S.—Zhou, M. C.—Li, Z. W.: Supervisor Optimization for Deadlock Resolution in Automated Manufacturing Systems with Petri Nets. IEEE Transactions on Automation Science and Engineering, Vol. 8, 2011, No. 4, pp. 794–804, doi: 10.1109/TASE.2011.2156783.

[33] Hu, H. S.—Zhou, M. C.—Li, Z. W.: Liveness and Ratio-Enforcing Supervision of Automated Manufacturing Systems Using Petri Nets. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, Vol. 42, 2012, No. 2, pp. 392–403, doi: 10.1109/TSMCA.2011.2162502.

[34] Hu, H. S.—Zhou, M. C.—Li, Z. W.—Tang, Y.: An Optimization Approach to Improved Petri Net Controller Design for Automated Manufacturing Systems. IEEE Transactions on Automation Science and Engineering, Vol. 10, 2013, No. 3, pp. 772–782, doi: 10.1109/TASE.2012.2201714.

[35] Hu, H. S.—Liu, Y.—Zhou, M. C.: Maximally Permissive Distributed Control of Large Scale Automated Manufacturing Systems Modeled with Petri Nets. IEEE Transactions on Control Systems Technology, Vol. 23, 2015, No. 5, pp. 2026–2034, doi: 10.1109/TCST.2015.2391014.

[36] Hu, H. S.—Su, R.—Zhou, M. C.—Liu, Y.: Polynomially Complex Synthesis of Distributed Supervisors for Large-scale AMSs Using Petri Nets. IEEE Transactions on Control Systems Technology, Vol. 24, 2016, No. 5, pp. 1610–1622, doi: 10.1109/TCST.2015.2504046.

[37] Hu, H. S.—Liu, Y.: Supervisor Simplification for AMS Based on Petri Nets and Inequality Analysis. IEEE Transactions on Automation Science and Engineering, Vol. 11, 2014, No. 1, pp. 66–77, doi: 10.1109/TASE.2013.2288645.

[38] Hu, H. S.—Liu, Y.: Supervisor Synthesis and Performance Improvement for Automated Manufacturing Systems by Using Petri Nets. IEEE Transactions on Industrial Informatics, Vol. 11, 2015, No. 2, pp. 450–458, doi: 10.1109/TII.2015.2402619.

[39] Hu, H. S.—Zhou, M. C.: A Petri Net-Based Discrete-Event Control of Automated Manufacturing Systems with Assembly Operations. IEEE Transactions on Control Systems Technology, Vol. 23, 2015, No. 2, pp. 513–524, doi: 10.1109/TCST.2014.2342664.

[40] Hu, H. S.—Zhou, M. C.—Li, Z. W.—Tang, Y.: Deadlock-Free Control of Automated Manufacturing Systems with Flexible Routes and Assembly Operations Using Petri Nets. IEEE Transactions on Industrial Informatics, Vol. 9, 2013, No. 1, pp. 109–121, doi: 10.1109/TII.2012.2198661.

[41] Huang, Y. S.—Jeng, M. D.—Xie, X. L.—Chung, D. H.: Siphon-Based Deadlock Prevention Policy for Flexible Manufacturing Systems. IEEE Transactions on Systems, Man, and Cybernetics – Part A: System and Humans, Vol. 36, 2006, No. 6, pp. 1248–1256, doi: 10.1109/TSMCA.2006.878953.

[42] Iordache, M. V.—Antsaklis, P. J.: Supervision Based on Place Invariants: A Survey. Discrete Event Dynamic Systems, Vol. 16, 2006, No. 4, pp. 451–492, doi: 10.1007/s10626-006-0021-9.

[43] Iordache, M. V.—Antsaklis, P. J.: Supervisory Control of Concurrent Systems: A Petri Net Structural Approach. Birkhäuser, 2006, doi: 10.1007/0-8176-4488-1.

[44] Karatkevich, A.: Dynamic Analysis of Petri Net-Based Discrete Systems. Springer, Heidelberg, Lecture Notes in Control and Information Sciences, Vol. 356, 2007, doi: 10.1007/978-3-540-71560-3.

[45] Lautenbach, K.: Linear Algebraic Calculation of Deadlocks and Traps. In: Voss, K., Genrich, H. J., Rozenberg, G. (Eds.): Concurrency and Nets. Springer, Berlin, Heidelberg, 1987, pp. 315–336, doi: 10.1007/978-3-642-72822-8_21.

[46] Lawley, M. A.—Reveliotis, S. A.: Deadlock Avoidance for Sequential Resource Allocation Systems: Hard and Easy Cases. International Journal of Flexible Manufacturing Systems, Vol. 13, 2001, No. 4, pp. 385–404, doi: 10.1023/A:1012203214611.

[47] Li, Z. W.—Zhou, M. C.: Elementary Siphons of Petri Nets and Their Application to Deadlock Prevention in Flexible Manufacturing Systems. IEEE Transactions on Systems, Man, and Cybernetics, Part A: System and Humans, Vol. 34, 2004, No. 1, pp. 38–51, doi: 10.1109/TSMCA.2003.820576.

[48] Li, Z. W.—Zhou, M. C.: Control of Elementary and Dependent Siphons in Petri Nets and Their Application. IEEE Transactions on Systems, Man, Cybernetics, Part A, System, Humans, Vol. 38, 2008, No. 1, pp. 133–148, doi: 10.1109/TSMCA.2007.909548.

[49] Li, Z. W.—Zhou, M. C.: Elementary Siphons of Petri Nets for Efficient Deadlock Control. In: Zhou, M. C., Fanti, M. P. (Eds.): Deadlock Resolution in Computer-Integrated Systems. CRC Press, 2005, pp. 309–348, doi: 10.1201/9781315214665.

[50] Li, Z. W.—Wei, N.: Deadlock Control of Flexible Manufacturing Systems via Invariant-Controlled Elementary Siphons of Petri Nets. The International Journal of Advanced Manufacturing Technology, Vol. 33, 2007, pp. 24–35, doi: 10.1007/s00170-006-0452-3.

[51] Li, Z. W.—Zhou, M. C.: Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach. Springer, London, Advances in Industrial Control Series, 2009, doi: 10.1007/978-1-84882-244-3.

[52] Li, Z. W.—Wu, N. Q.—Zhou, M. C.: Deadlock Control of Automated Manufacturing Systems Based on Petri Nets – A Literature Review. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), Vol. 42, 2012, No. 4, pp. 437–462, doi: 10.1109/TSMCC.2011.2160626.

[53] Li, X. Y.—Liu, G. Y.—Li, Z. W.—Wu, N. Q.—Barkaoui, K.: Elementary Siphon-Based Robust Control for Automated Manufacturing Systems with Multiple Unreliable Resources. IEEE Access, Vol. 7, 2019, pp. 21006–21019, doi: 10.1109/access.2019.2897753.

[54] Liu, G. J.—Jiang, C. J.: Incidence Matrix Based Methods for Computing Repetitive Vectors and Siphons of Petri Net. Journal of Information Science and Engineering, Vol. 25, 2009, No. 1, pp. 121–136.

[55] Liu, G. Y.—Li, Z.: General Mixed Integer Programming-Based Liveness Test for System of Sequential Systems with Shared Resources Nets. IET Control Theory and Applications, Vol. 4, 2010, No. 12, pp. 2867–2878, doi: 10.1049/iet-cta.2009.0557.

[56] Liu, G.—Li, Z.—Zhong, C.: New Controllability Condition for Siphons in a Class of Generalised Petri Nets. IET Control Theory and Applications, Vol. 4, 2010, No. 5, pp. 854–864, doi: 10.1049/iet-cta.2009.0264.

[57] Liu, G.—Li, Z.—Zhong, C.: Correction to 'New Controllability Condition for Siphons in a Class of Generalised Petri Nets'. IET Control Theory and Applications, Vol. 7, 2013, No. 4, pp. 632–633, doi: 10.1049/iet-cta.2012.0357.

[58] Liu, G. Y.—Barkaoui, K.: A Survey of Siphons in Petri Nets. Information Sciences, Vol. 363, 2016, pp. 198–220, doi: 10.1016/j.ins.2015.08.037.

[59] Liu, G. Y.—Barkaoui, K.: Necessary and Sufficient Liveness Condition of GS^3PR Petri Nets. International Journal of Systems Science, Vol. 46, 2015, No. 7, pp. 1147–1160, doi: 10.1080/00207721.2013.827257.

[60] Liu, G. Y—Li, Z. W.—Al-Ahmari, A. M.: Liveness Analysis of Petri Nets Using Siphons and Mathematical Programming. IFAC Proceedings Volumes, Vol. 47, 2014, No. 2, pp. 383–387, doi: 10.3182/20140514-3-FR-4046.00078.

[61] Liu, D.—Barkaoui, K.—Zhou, M. C.: On Intrinsically Live Structure of a Class of Generalized Petri Nets Modeling FMS. IFAC Proceedings Volumes, Vol. 45, 2012, No. 29, pp. 187–192, doi: 10.3182/20121003-3-MX-4033.00032.

[62] Liu, G. Y.—Li, Z. W.—Barkaoui, K.—Al-Ahmari, A. M.: Robustness of Deadlock Control for a Class of Petri Nets with Unreliable Resources. Information Sciences, Vol. 235, 2013, pp. 259–279, doi: 10.1016/j.ins.2013.01.003.

[63] Liu, G. Y.—Barkaoui, K.: A Survey of Siphons in Petri Nets. Information Sciences, Vol. 363, 2016, pp. 198–220, doi: 10.1016/j.ins.2015.08.037.

[64] Liu, G.—Li, P.—Li, Z.—Wu, N.: Robust Deadlock Control for Automated Manufacturing Systems with Unreliable Resources Based on Petri Net Reachability Graphs. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 49, 2019, No. 7, pp. 1371–1385, doi: 10.1109/TSMC.2018.2815618.

[65] Ma, Z. Y.—Li, Z. W.—Giua, A.: Computation of Admissible Marking Sets in Weighted State Machines by Dynamic Programming. Proceedings of the 2017 IEEE 56th Annual Conference on Decision and Control (CDC), Melbourne, VIC, Australia, 2017, pp. 4847–4852, doi: 10.1109/CDC.2017.8264375.

[66] Ma, Z. Y.—Li, Z. W.—Giua, A.: Design of Optimal Petri Net Controllers for Disjunctive Generalized Mutual Exclusion Constraints. IEEE Transactions on Automatic Control, Vol. 60, 2015, No. 7, pp. 1774–1785, doi: 10.1109/TAC.2015.2389313.

[67] Ma, Z. Y.—Li, Z. W.—Giua, A.: A Constraint Transformation Technique for Petri Nets with Certain Uncontrollable Structures. IFAC Proceedings Volumes, Vol. 47, 2014, No. 2, pp. 66–72, doi: 10.3182/20140514-3-fr-4046.00085.

[68] Ma, Z. Y.—Li, Z. W.—Giua, A.: Petri Net Controllers for Disjunctive Generalized Mutual Exclusion Constraints. Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies and Factory Automation (ETFA), Cagliari, Italy, 2013, pp. 1–8, doi: 10.1109/ETFA.2013.6648003.

[69] Ma, Z. Y.—Li, Z. W.—Giua, A.: Petri Net Controllers for Generalized Mutual Exclusion Constraints with Floor Operators. Automatica, Vol. 74, 2016, pp. 238–246, doi: 10.1016/j.automatica.2016.07.042.

[70] Minoux, M.—Barkaoui, K.: Deadlocks and Traps in Petri Nets as Horn-Satisfiability Solutions and Some Related Polynomially Solvable Problems. Discrete Applied Mathematics, Vol. 29, 1990, No. 2-3, pp. 195–210, doi: 10.1016/0166-218X(90)90144-2.

[71] Moody, J. O.—Antsaklis, P. J.: Petri Net Supervisors for DES with Uncontrollable and Unobservable Transitions. IEEE Transactions on Automatic Control, Vol. 45, 2000, No. 3, pp. 462–476, doi: 10.1109/9.847725.

[72] Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, Vol. 77, 1989, No. 4, pp. 541–580, doi: 10.1109/5.24143.

[73] Nakamoto, F. Y.—Miyagi, P. E.—dos Santos Filho, D. J.: Automatic Generation of Control Solution for Resource Allocation Using Petri Net Model. Produçaõ (International Journal of Advanced Manufacturing Technology), Vol. 19, 2009, No. 1, pp. 8–26, doi: 10.1590/S0103-65132009000100002.

[74] Nakamoto, F. Y.—Miyagi, P. E.—dos Santos Filho, D. J.: Resources Allocation Control in Flexible Manufacturing Systems Using the Deadlock Avoidance Method. ABCM (Brazilian Society of Mechanical Science and Engineering), ABCM Symposium Series in Mechatronics, Vol. 3, 2008, pp. 454–460.

[75] Park, J.—Reveliotis, S. A.: Deadlock Avoidance in Sequential Resource Allocation Systems with Multiple Resource Acquisitions and Flexible Routings. IEEE Transactions on Automatic Control, Vol. 46, 2001, No. 10, 2001, pp. 1572–1583, doi: 10.1109/9.956052.

[76] Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer, 2013, doi: 10.1007/978-3-642-33278-4.

[77] Reisig, W.: Petri Nets. Springer, Berlin, Heidelberg, EATCS Monographs on Theoretical Computer Science, Vol. 4, 1985, doi: 10.1007/978-3-642-69968-9.

[78] Reveliotis, S. A.—Ferreira, P. M.: Deadlock Avoidance Policies for Automated Manufacturing Cells. IEEE Transactions on Robotics and Automation, Vol. 12, 1996, No. 6, pp. 845–857, doi: 10.1109/70.544768.

[79] Reveliotis, S. A.—Lawley, M. A.—Ferreira, P. M.: Polynomial-Complexity Deadlock Avoidance Policies for Sequential Resource Allocation Systems. IEEE

Transaction on Automatic Control, Vol. 42, 1997, No. 10, pp. 1344–1357, doi: 10.1109/9.633824.

[80] Reveliotis, S. A.: Logical Control of Complex Resource Allocation Systems. Foundations and Trends in Systems and Control, Vol. 4, 2017, No. 1-2, pp. 1–223, doi: 10.1561/2600000010.

[81] Reveliotis, S. A.: Coordinating Autonomy: Sequential Resource Allocation Systems for Automation. IEEE Robotics and Automation Magazine, Vol. 22, 2015, No. 2, pp. 77–94, doi: 10.1109/MRA.2015.2401295.

[82] Reveliotis, S. A.: On the Siphon-Based Characterization of Liveness in Sequential Resource Allocation Systems. In: van der Aalst, W. M. P., Best, E. (Eds.): Applications and Theory of Petri Nets 2003 (ICATPN 2003). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2679, 2003, pp. 241–255, doi: 10.1007/3-540-44919-1_17.

[83] Row, T. C.—Syu, W. M.—Pan, Y. L.—Wang, C. C.: One Novel and Optimal Deadlock Recovery Policy for Flexible Manufacturing Systems Using Iterative Control Transitions Strategy. Mathematical Problems in Engineering, Vol. 2019, Art. No. 4847072, 12 pp., doi: 10.1155/2019/4847072.

[84] Schmidt, K.: Verification of Siphons and Traps for Algebraic Petri Nets. In: Azéma, P., Balbo, G. (Eds.): Application and Theory of Petri Nets 1997 (ICATPN 1997). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1248, 1997, pp. 427–446, doi: 10.1007/3-540-63139-9_49.

[85] Taoka, S.—Furusato, S.—Watanabe, T.: A Heuristic Algorithm FSDC Based on Avoidance of Deadlock Components in Finding Legal Firing Sequences of Petri Nets. In: van der Aalst, W. M. P., Best, E. (Eds.): Applications and Theory of Petri Nets 2003 (ICATPN 2003). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2679, 2003, pp. 417–439, doi: 10.1007/3-540-44919-1_26.

[86] Tricas, F.—Ezpeleta, J.: Computing Minimal Siphons in Petri Net Models of Resource Allocation Systems: A Parallel Solution. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, Vol. 36, 2006, No. 3, pp. 532–539, doi: 10.1109/TSMCA.2005.855751.

[87] Tricas, F.: Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems. Ph.D. Thesis, University of Zaragoza, Zaragoza, Spain, 2003. Available at: `https://www.researchgate.net/publication/33419595`.

[88] Wang, Y.—Lafortune, S.—Kelly, T.—Kudlur, M.—Mahlke, S.: The Theory of Deadlock Avoidance via Discrete Control. ACM SIGPLAN Notices, Vol. 44, 2009, No. 1, pp. 252–263, doi: 10.1145/1594834.1480913.

[89] Wang, Y.—Kelly, T.—Kudlur, M.—Mahlke, S.—Lafortune, S.: The Application of Supervisory Control to Deadlock Avoidance in Concurrent Software. Proceedings of 9th IEEE/IFAC International Workshop on Discrete Event Systems (WODES'08), Göteborg, Sweden, 2008, pp. 287–292, doi: 10.1109/WODES.2008.4605961.

[90] Wegrzyn, A.—Karatkevich, A.—Bieganowski, J.: Detection of Deadlocks and Traps in Petri Nets by Means of Thelen's Prime Implicant Method. International Journal of Applied Mathematics and Computer Science, Vol. 14, 2004, No. 1, pp. 113–121.

[91] Wu, W. H.—Chao, D. Y.: Controllability of Weakly Dependent Siphons under Elementary-Siphon Control. Transactions of the Institute of Measurement and Control, Vol. 38, 2016, No. 8, pp. 941–955, doi: 10.1177/0142331214568606.

[92] Yan, M. M.— Zhu, R. M.—Li, Z. W.—Wang, A.—Zhou, M. C.: A Siphon-Based Deadlock Prevention Policy for a Class of Petri Nets – S^3PMR. Proceedings of the 17th World Congress of the International Federation of Automatic Control (IFAC), Seoul, Korea, 2008, Vol. 6, 2008, pp. 3352–3357. Available at: `http://toc.proceedings.com/04672webtoc.pdf`. ISBN: 978-1605607580.

[93] Yan, M. M.—Li, Z. W.—Wei, N.—Zhao, M.: A Deadlock Prevention Policy for a Class of Petri Nets S^3PMR. Journal of Information Science and Engineering, Vol. 25, 2009, No. 1, pp. 167–183.

[94] Yue, H.—Xing, K. Y.—Hu, H. S.—Wu, W. M.—Su, H. Y.: Petri-Net-Based Robust Supervisory Control of Automated Manufacturing Systems. Control Engineering Practice, Vol. 54, 2016, pp. 176–189, doi: 10.1016/j.conengprac.2016.05.009.

František Čapkovič received his Master's degree in 1972 from the Faculty of Electrical Engineering of the Slovak Technical University, Bratislava, Slovakia. Since 1972 he has been working with the Slovak Academy of Sciences (SAS), Bratislava, in 1972–1991 at the Institute of Technical Cybernetics, in 1991–2001 at the Institute of Control Theory and Robotics and in 2001 till now at the Institute of Informatics. In 1980 he received his Ph.D. from SAS, since 1998 serving as Associate Professor. He works in the area of modelling, analysing and intelligent control of Discrete-Event Systems (DES) and Hybrid Systems. He is the author of more than 240 publications.

Computing and Informatics, Vol. 40, 2021, 690–728, doi: 10.31577/cai_2021_3_690

WHAT IS YOUR CODE CLONE DETECTION AND EVOLUTION RESEARCH MADE OF?

Chaman Wijesiriwardana

University of Moratuwa, Katubedda
Sri Lanka
e-mail: chaman@uom.lk

Prasad Wimalaratne

University of Colombo School of Computing
Reid Avenue, Colombo 7, Sri Lanka
e-mail: spw@ucsc.cmb.ac.lk

Abstract. Over the past few decades, clone detection and evolution have become a major area of study in software engineering. Clone detection experiments present several challenges to researchers such as accurate data collection, selecting proper code detection algorithms, and understanding clone evolution phenomena. This paper attempts to facilitate clone detection and evolution research by providing a structured and systematic mechanism to conduct experiments. Clone detection experiments usually consist of several tasks such as fetching data from a version control system, performing necessary pre-processing activities, and feeding the data to a clone detection algorithm. Therefore, a particular clone detection experiment can interpret as a meaningful combination of such tasks into a scientific workflow. In this work, the concrete tasks in a code clone detection workflow are referred to as Building Blocks. This paper presents a useful collection of Building Blocks identified based on a systematic literature review, and a conceptual framework of an experimental testbed to facilitate clone detection experiments. The reusability of the Building Blocks was validated using four case studies selected from the literature. The validation results confirm the reusability and the expressiveness of the Building Blocks in new ventures. Besides, the proposed experimental testbed is proven beneficial in conducting and replicating clone detection experiments.

Keywords: Code clone detection, clone evolution, scientific workflows, building blocks, experiemental testbed

1 INTRODUCTION

Code clones are source code fragments that are similar or identical in terms of text, structure, or meaning. During software development, code fragments are copied and pasted with or without major alternations. The pasted portion of the code is said to be a clone, and this practice is known as code cloning. This has been a common practice in the software development process due to several reasons such as limitations of the programming languages, delaying refactoring, and high code reuse [46]. The negative impacts of code cloning has been reported in several studies [34, 3, 8]. The consequences of code cloning, clone evolution, and clone removal have both positives and negatives. Fowler et al. [24] were one of the firsts to argue that code cloning is one of the leading causes of *bad smells* in software systems. On the other hand, some researchers counter-argued by highlighting several positives of cloning such as improved productivity [5, 43, 71]. According to [15], there can be organizational reasons to copy-paste code. Therefore, a systematic analysis is required before the clone removal. As a result, clone removal can sometimes directly associate with a considerable risk factor. More research along this direction should be conducted to get a better understanding, and the uncertainties mentioned above evidently emphasize the need for systematic Code Clone Detection and Evolution (CCDE) experiments.

This paper contributes to this field of study by proposing a structured and organized way to plan and conduct CCDE experiments. Clone detection studies are conducted based on a well-defined logical process with some common steps. For example, studies typically start with mining activity, e.g., by retrieving data from a version control system. Then the mined source code is processed and transformed into an intermediate format, such as tokens, code metrics, Abstract Syntax Trees (ASTs), or Program Dependency Graphs (PDGs). This data is then fed into a code clone detection algorithm to extract the clones. Finally, the detected code clones can be further subjected to clone genealogy analysis (i.e., clone patterns, visualization) to understand clone evolution better. This research insists that the steps mentioned above can adequately arrange and combine into a well-defined scientific workflow [18, 80]. Each step represents a particular clone detection task, such as mining the code base, calculating metrics, or generating an AST. These tasks may be in different representation levels and the effort required to perform the tasks may have significant differences. However, the identification of representation levels and the effort required for the tasks are out of the scope of this paper. In this paper, these concrete tasks are referred to as *Building blocks* for CCDE research. We believe that the concept of *Building blocks* will provide direct solutions to some of the common challenges such as accurate data extraction, data cleaning, and pick the

correct clone detection technique in conducting CCDE studies. Therefore, *Building blocks* provide ways to thoroughly comprehend the clone detection process as well as to replicate previous experiments systematically. This work devised a conceptual framework and a proof-of-concept experimental testbed to conduct CCDE experiments, which could be extended to conduct other types of software engineering experiments as well.

This paper intends to address the following Research Questions (RQs):

RQ1: Is it possible to identify reusable data flow-based *Building Blocks* for code clone detection and evolution from the existing literature and express them in a unified manner?

RQ2: How to interlink such *Building Blocks* categorized into multiple abstraction levels to develop a conceptual framework of an experimental testbed to conduct code clone detection and evolution experiments?

Based on the research questions, this paper presents three main contributions. First, a novel concept and a methodology for identifying re-usable building blocks from various research workflows in the area of code clone detection. Second, a concrete collection of useful formal building blocks was collected via the above methodology. Finally, a conceptual framework of an experimental testbed by interlinking the identified building blocks to conduct and replicate previous CCDE experiments.

The remainder of this paper is organized as follows: Section 2 describes the background of software evolution analysis and code clone detection. Section 3 details our research methodology to identify building blocks from the analysis workflows. In Section 4, we present our catalog of building blocks extracted via the literature survey followed by the conceptual framework of the experimental testbed in Section 5. Then we present validation in Section 6 followed by the discussion and conclusion in Section 7 and Section 8, respectively.

2 BACKGROUND

Analysis of software evolution is known as an enormously dynamic field of research in software engineering. Understanding the evolution of large-scale software systems is a demanding problem for several reasons: huge amounts of information have to be considered, and historical data has to be analyzed. Software evolution analysis mainly focuses on two main aspects; to better understand the reasons for its existing problems and to forecast its future developments [17]. Software evolution analysis experiment such as CCDE is a classic example to address both of these goals. First, we summarize some of the pioneer surveys in CCDE and indicate how our approach conceptually differs from the existing surveys. Then we dig into the CCD techniques and tools followed by a summary of the existing approaches for software engineering data analytics. Finally, we briefly describe the need for a novel mechanism to facilitate the researchers in conducting CCDE experiments.

2.1 Surveys in Code Clone Detection and Evolution

A considerable number of code clone related survey papers are published in the past. Koshke [46] was one of the first to write a survey paper on code clone detection. That paper reports some essential aspects such as different categorizations of clone types, root causes for cloning, current opinions of cloning, empirical studies on the evolution of clones, benchmarks for clone detector evaluations, and presentation issues.

Roy et al. [70] presented a qualitative comparison and evaluation of the existing literature in clone detection techniques and tools. A more detailed description can be found in [68]. The findings of their research could help new potential users of clone detection techniques in understanding the range of available techniques and tools and selecting those most appropriate for their needs. Ratten et al. [67] perform a systematic review of existing code clone approaches based on 213 identified papers. The results are presented in different dimensions like classification of clone research, code clone management as cross-cutting domain, types of clones, clone detection tools, and clone detection approaches. Similar to [68], this approach also intends to facilitate researchers in conducting code clone detection researches. Sheneamer et al., [76] also surveyed code clone detection. The aim of this paper goes beyond comparing the tools and techniques. Instead, it presents several observations in developing hybrid techniques in the future. Pate et al. present a survey paper in code clone evolution., [63]. They have indicated that human-based empirical studies and classification of clone evolution patterns as two significant areas for further work. Ain et al. [3] reviewed 54 journal papers and conference papers, which emphasized the need to introduce novel approaches to detect all four types of clones. Walker et al. [84] presented a systematic mapping study on existing CCD tools with regards to technique, open-source nature, and language coverage. Finally, they propose some possible future directions for code-clone detection tools.

2.2 Code Clone Detection: Techniques and Tools

Code reuse is a frequent activity in software development. Code reusing can be in the form of copying a portion of the code and pasting it with or without modifications. This type of reuse known as code cloning and the pasted code fragment is called a clone of the original code [70]. Nevertheless, during the maintenance stage, identifying the original code fragment and the copied code fragment is a non-trivial task. Several clone detection approaches have been proposed in the literature spanning from textual to semantic approaches. However, this paper does not consider the techniques and tools proposed for cross-language clone detection [57].

Text based clone detection: During this approach, code fragments are compared with each other in the form of texts; strings or lexemes and similar portions are identified as code clones [67]. One of the earliest clone detection approaches was proposed by Johnson [36, 38]. He applied a fingerprinting mechanism for

comparison of source code. Ducasse et al. [21] developed a language independent clone detection tool, duploc, which aims at overcoming the obstacle of having the right parser for the right dialect for every language. However, this approach requires a significant effort in pre-processing and transforming the source code into the required syntax. Duploc cannot detect Type-3 clones or deal with modifications and insertions in copy-pasted code [76]. NICAD [69] is a text-based clone detection tool that is capable of effectively detecting clones up to type 3. It is based on lightweight parsing to implement code normalization and code filtering. Seunghak and Jeong [49] presented a text-based code clone detection technique. They have implemented Similar Data Detection (SDD) tool, which is an Eclipse plug-in. Dou et al. [20] has effectively used text-based clone detection technique in detecting clones in spreadsheets.

Token based clone detection: Token-based clone detection techniques are considered better than text-based detection. In this approach, lexical analysis is used to extract the tokens from the source code by lexical analysis. One of the focal points behind token-based clone detection algorithms is to perform suffix tree or suffix array based token-to-token comparisons. A suffix tree is a data structure that exposes the internal structure of a string in a deeper way [28]. Suffix array is also a conceptually simple data structure which is initially developed for on-line string searches [54]. The central advantage of suffix arrays over suffix trees is that, in practice, suffix arrays use three to five times less space. CCFinder [40] is a well-known tool of this category, which finds identical subsequences by using a suffix tree matching algorithm. The research community for code clone analysis as well as code clone management broadly uses CCFinder. Dup [7, 6] is another token-based clone detection tool, which divides the source files into tokens by a lexical analyzer. CCLEARNER [51] is a token based clone detection tool developed by leveraging deep learning. However, approaches such as [69] and [74] are not exploiting suffix trees or arrays in detecting code clones. For example, Sajanai et al., [74] uses an optimized partial index and filtering heuristics to achieve large-scale clone detection. This technique has used in recent studies as well [73].

Tree-based clone detection: In tree-based clone detection algorithms, the source code transformed into a parse tree or an abstract syntax tree. A parse tree is a data structure for the parsed representation of a statement in a particular code fragment [12]. The usefulness of generating the parse tree is that, by parsing two code fragments and comparing their parse trees, it is possible to determine whether the code fragments are identical or not. Abstract Syntax Tree is also a special kind of parse tree. In parse trees, the roots of subtrees represent nonterminal symbols of the grammar, while leaves represent terminal grammar symbols. In an abstract syntax tree, operators represent root nodes, while leaves symbolize operands [60]. Once the trees are generated, tree-matching algorithms are used to find the similar code fragments. One of the first approaches of this category was presented by Yang [89]. CloneDR [11] is another token-based clone

detection tool, which can detect exact and near-miss clones using hashing and dynamic programming. Wahler et al. [83] presented a technique to find clones at a more abstract level by converting abstract syntax tree to XML format.

Program dependency graph based clone detection: A program dependence graph (PDG) is a graph representation of a code fragment. In PGDs, basic statements such as variable declarations, assignments, and function calls are represented by program vertices. Edges between program vertices in PDGs represent the data and control dependencies between statements [23]. In Program Dependency Graphs (PDG), the source code is abstracted to extract the control flow and data flow graphs. Krinke [48] has presented a methodology for identifying similar code based on finding similar maximal sub-graphs by using the k-limiting technique. Hugo and Kusumoto [29] proposed a methodology to enhance PGD based clone detection based on PDG specializations and detection heuristics. Clone detection tools based on PGDs, as proposed in [44] and [87], can be used to identify type 4 clones.

Metric based clone detection: In a metric-based approach [45, 55] the source code is divided into smaller units (e.g., one line, one method, one class) and metrics are calculated for each unit. The metrics of each unit are compared, and those with the same values are identified as clones. Examples of metrics are the number of function calls within a unit or the cyclomatic complexity of the unit. The type of metrics used by each tool impacts the language dependency of the tool.

Hybrid clone detection: Hybrid clone detection techniques typically employ a combination of clone detection techniques. A hybrid approach aims at overwhelming the problems encountered by specific techniques. Leitao [50] presents a hybrid approach that combines syntactic techniques and semantic techniques with specialized comparison functions. Hummel et al. [33] present ConQat, which is an incremental index-based hybrid technique to detect clones. Agrawal et al. [2] described a hybrid approach by combining token-based and textual approaches to find code cloning. Hu et al. [31, 32] recently proposed BINMATCH, which is a hybrid approach to detect binary clone functions.

2.3 Code Clone Evolution

Software evolves from one version to another when adding new features, getting involved with fixing bugs, improving performance and increasing reliability. As a result, code clones also evolve simultaneously together with software systems. Therefore, analysis of code clone evolution is critical to comprehend the effect of code clones to the entire software system.

There are several noteworthy studies in the literature on code clone evolution with a particular focus on clone genealogies. Kim et al. [43] has pioneered one of the first investigations on code clone evolution. That paper defined clone genealogies as the history of how each element in a group of clones has changed concerning

other elements in the same group. In that research, they emphasized the need for understanding clone genealogies to maintain code clones better. Saha et al. [71] extended the research conducted by Kim et al. [43] by incorporating different dimensions. They have presented an empirical study to investigate clone genealogies using 17 open source software systems. Clone evolution related investigations have been further reported by Göde [26], Barbour et al. [8] and Krinke [47].

2.4 Software Engineering Data Analytics

Existing frameworks for software engineering data analytics based on either generic query languages such as SQL or domain-specific languages. Some of the approaches that directly follow the standard SQL syntax are Gitana [16], AlitheiaCore [27] and MetricMiner [77]. However, such approaches not specifically targeted at facilitating CCDE experiments. Thus, CCDE specific functionalities are not available. Besides, the domain-specific languages such as Boa [22] and QWALKEKO [78] requires a prior understanding of the language itself. Hence, the usability of such frameworks, particularly, for novice researchers is questionable.

2.5 Summary

Our approach is conceptually different from the previous work. Existing survey papers on CCDE mainly focused on identifying the different techniques, tools, compare them, selecting appropriate technique, and present the observations on future CCD tools. This paper presents a different dimension to facilitate researchers in understanding and conducting code clone detection experiments by utilizing a set of Building Blocks that are meant to CCD experiments. For example, conducting comparison studies is hard for the researchers as the clone detection techniques are naturally complex as there are many different pre-processing activities, transformation activities, and algorithms are involved. Therefore, a unified framework to facilitate replication studies is important, and to the best of our knowledge, such frameworks are not adequately presented in the literature. Thus, we believe that our approach would shed light on future directions such as [63].

3 APPROACH

This research aims at providing a systematic way to conduct CCDE experiments by identifying reusable tasks from the literature. Initially, a literature review on papers published on reputed software engineering conferences such as ICSE[1], MSR[2], ICSM(E)[3] and FSE[4] conducted for 11 years (2010-2020). All the papers are ex-

[1] International Conference on Software Engineering
[2] Working Conference on Mining Software Repositories
[3] International Conference on Software Maintenance (and Evolution)
[4] Foundations of Software Engineering

tracted from the main track of each of the conferences. First, the papers published on CCDE has filtered and carefully investigated the methodology section in each article. However, the study is not strictly limited to the baseline papers on the mentioned conferences in the given duration. We further traced back and forth to find related work published outside the selected papers as well. Based on this study, a mechanism has introduced to drill-down the experiments to identify the concrete, reusable tasks. From the clone detection experiments, four main *activities* has identified: data gathering, pre-processing, clone detection and post-processing. Such activities are implemented via smaller tasks or sub-tasks, which refers to as *Building Blocks* in this paper. Finally, we proposed a method to represent the building blocks using a semi-structured textual notation and a graphical notation. Below we provide an overview of our proposed methodology.

This review can be further extended by considering the other reputed software engineering conferences. However, the objective of this study is to find a useful collection of building blocks to conduct CCDE experiments. Thus, the papers published in the selected conferences were rich enough to identify the building blocks for CCDE.

3.1 Data Collection

The process of selecting suitable research publications for a particular review has two major problems: identifying the relevant work and assessing the quality of the selected studies. Therefore, it was decided to minimize the risk of errors by mainly reviewing the papers published in well-reputed software engineering conferences. A literature review is conducted in the proceedings of the ICSE, MSR, ICSM(E) and FSE conferences for eleven years (2010–2020). From that the papers published on *code clone detection* has filtered out. Table 1 presents a summary of the reviewed papers.

Year	**No. of Papers Reviewed**			
	ICSE	**MSR**	**ICSM(E)**	**FSE**
2010	1	2	3	3
2011	2	1	3	0
2012	4	0	3	0
2013	3	2	3	1
2014	3	3	7	1
2015	0	1	2	1
2016	1	0	1	1
2017	2	1	2	0
2018	1	2	2	1
2019	3	1	2	0
2020	3	0	1	0

Table 1. Number of reviewed clone detection papers

3.2 Drilling Down Code Clone Detection Experiments to Building Blocks

Scientific workflows are meant to be data flow oriented, which facilitates streamlining of scientific tasks to make significant scientific discoveries [53]. They widely recognized as a useful mechanism to describe and manage complex scientific analyses. Scientific workflows provide means to specify how a specific experiment can be modeled and carried out. In such workflows, relevant activities need to sequence in a pipeline to create a workflow that can execute a particular analysis experiment. Therefore, a specific CCDE experiment can interpret as a problem of creating a suitable workflow and running it without interruptions. In this research, scientific workflows are considered to be the top level abstraction of CCDE experiments.

Activities: A CCDE workflow is composed of activities. Activities are the tasks and sub-tasks that directly associated with CCDE experiments such as fetching data from VCS, calculating metrics or removing test files. In this paper, tasks that serve a specific analysis or perform a specific operation are grouped under a particular Activity. For example, extracting data from a VCS is an essential task in CCDE experiments. Therefore, fetch data from GIT, fetch data from SVN and fetch data from CVS can be grouped under an activity called *data gathering*. Similarly, tasks such as snapshot generation and token generation can pool under the activity called *pre-processing*. As explained in [43], though it is not necessary to follow all the steps, a typical clone detection process follows the key activities namely pre-processing, transformation, match detection, formatting, filtering, and aggregation. We slightly modified the activities while keeping the core concept unchanged to fit into our context, as shown in Figure 1.

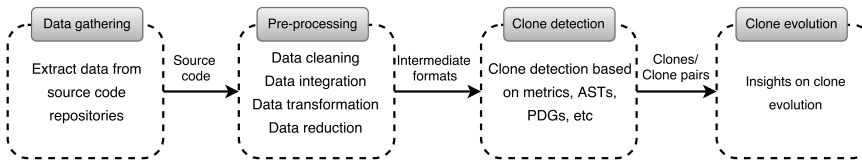


Figure 1. Common activities in a code clone detection and evolution workflow

1. Data gathering: During this phase, historical data about software projects are extracted from version control systems (e.g., CVS, SVN, GIT). Given that one of the leading contributions of this research is to identify reusable BBs in CCDE research, the other variant of data extractions such as gathering data from binaries has not considered.
2. Pre-processing: Data pre-processing can be in different forms such as data cleaning, data conversion or data integration. For example, data originating from the version control repositories need to be converted to different formats to facilitate various kinds of CCDE experiments. Furthermore, data from a VCS has to tokenize before feeding it into a token-based clone detection algorithm. The pre-processing steps can be carried out within the

clone detectors as well. However, separating it out from clone detectors has several notable advantages. For instance, novice researchers can better understand the fine-grained details of the entire CCDE process. Besides, the BBs in the pre-processing stage can be utilized in various software evolution experiments.

3. Clone detection: During this step, how the different clone detection approaches function in different settings are described. For example, after tokenizing the source code in the pre-processing stage, it has to be fed into a token-based clone detection algorithm. Similarly, metrics-based clone detectors depend on the metrics that generated in the pre-processing step.
4. Clone evolution: The primary goal of this step is to investigate how the clone detection results can utilize in clone evolution. For example, code clone genealogies provide useful insights to express how code clones change over multiple versions of the software.

Building Blocks: Activities are implemented via *Building Blocks* (BBs). A building block noticeably represents a specific analysis task such as fetch data from GIT, fetch data from SVN, generate snapshots or create ASTs. Figure 2 is a graphical representation of such BBs. Each BB is responsible only for a small fragment of functionality. Dependencies between BBs within a workflow determined by a list of parameters such as input and output parameters, pre-conditions and post-conditions. Input parameters and the pre-conditions are directly associated with Predecessor BBs, whereas output parameters and the post-conditions are directly associated with Follow-up BBs. Predecessor BBs and Follow-up BBs are two properties used to represent a particular BB, which explains in the next paragraph. For example, if an input parameter of an activity B is connected to an output parameter of activity A, it means that activity A must execute before activity B, and the data produced by activity A is consumed by activity B. The connection logic is explained in Section 6.2 under the implementation details of the proposed experimental testbed. Therefore, more comprehensively, we can infer that CCDE experiments consist of activities (e.g., data gathering, pre-processing), which are implemented via building blocks (e.g., SVN miner, AST generator).

Representing Building Blocks: A building block may consist of processes, data sources, operators and relationships. A process is a concrete example of activity, as mentioned earlier (e.g., mine version control repositories, mine bug repositories). The data source can be a source code repository such as Git, a bug repository such as Bugzilla, or any other intermediate location that keeps data. Operators are the basic operations that could perform on data (e.g., filter, sort). Likewise, a set of well-defined building blocks will be created, offering the option to use and combine such building blocks into a workflow to solve or to better understand complex CCD tasks.The usefulness and reusability of BBs mainly depend on the degree of expressiveness of such BBs. This expressiveness allows easily identifying when and why to use specific BBs, as well as when and why not

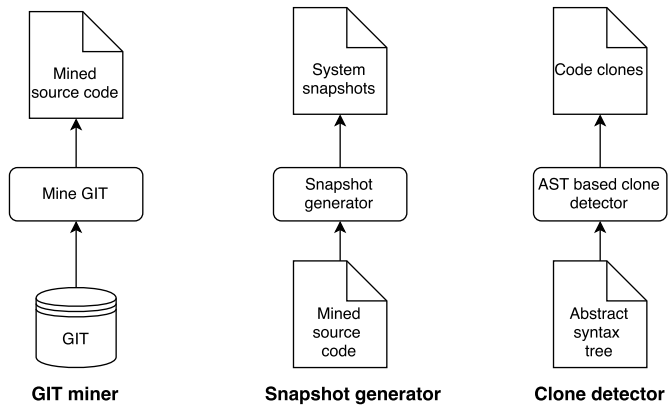


Figure 2. Building blocks in a CCDE workflow

use them. As per our understanding, there is no universally accepted standard for representing any sort of analysis tasks, in our case *Building Blocks*. However, we consider the following properties are rich enough to effectively describe a BB.

- Building block name: name of the BB
- Activity name: high level activity of the BB
- Problem(situation): why and when to apply the BB
- Solution: how to apply the BB and what it exactly does
- Alternatives: what other BBs could be used to replace this BB
- Predecessors: what other BBs should be executed prior to this BB
- Follow-up BBs: what other BBs could be executed after this BB
- References: how other researchers have used this BBs in conducting CCD experiments

4 BUILDING BLOCKS FOR CODE CLONE DETECTION AND EVOLUTION

This section presents a catalog of BBs, which classified into four main activities: data gathering, pre-processing, clone detection and post-processing. Such BBs further described by using the properties mentioned in Section 3.2. Individual BBs are not represented graphically. However, an overall view of BBs and how they can meaningfully interconnect with other BBs provided in Figure 3. This representation goes beyond a simple classification, but provide insights to the researchers on how to utilize and compose BBs to solve a particular analysis task.

Building Blocks for Data Gathering: Software evolution experiments typically require information about software projects that are collected via repository

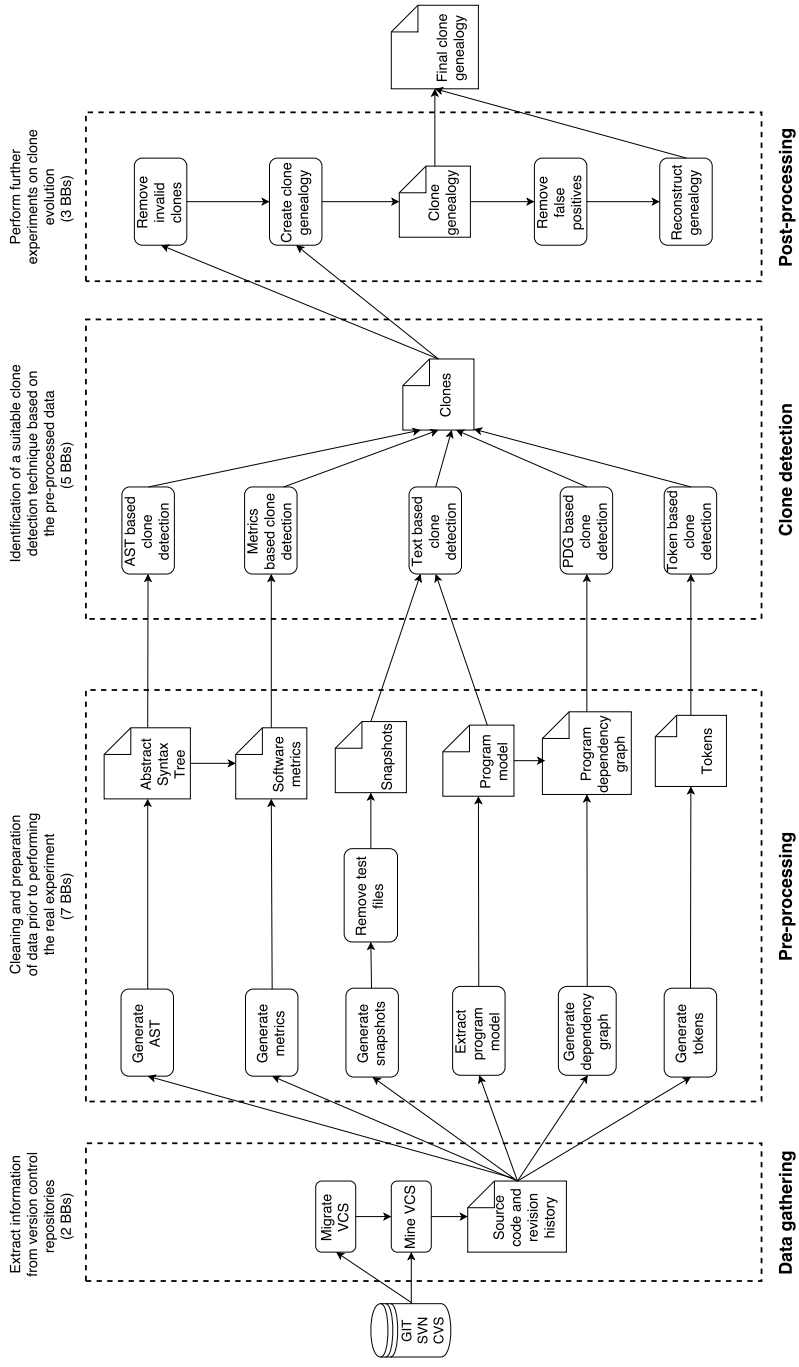


Figure 3. Activities and building blocks in code clone detection and evolution

mining. For data gathering, two important BBs have identified and presented to extract information from version control repositories namely VCS Miner and VC Migrator (See Table 2). More specifically, VCS Miner is a more general term that explicitly represents GIT Miner, SVN Miner, and CVS Miner.

Building Blocks for Pre-Processing: As with any dataset, there is certainly a great deal of cleaning and pre-processing required before any real analysis can perform. The pre-processing stage can often take the majority of the time spent on a data analysis project. Having a proper understanding of the required pre-processing steps allows a researcher to speed up the data preparation process as well as to reduce the complexity of the mining process. In this work, seven main BBs have identified for pre-processing: Snapshot Generator, Test Files Remover, Program Model Generator, Dependency Graph Generator, AST Generator, Metrics Generator and Token Generator (see Table 3).

Building Blocks for Clone Detection: Several code clone detection approaches and tools have proposed in the literature spanning from textual to semantic. Designing a clone detection experiments requires identification of a suitable clone detection technique based on the available data set and the output of the pre-processing step. In this work, five central BBs have identified for clone detection: String Based Clone Detector, AST Based Clone Detector, Metrics Based Clone Detector, Graph-Based Clone Detector and Token-Based Clone Detector (see Table 4).

Building Blocks for Clone Evolution: Conducting a comprehensive analysis of clone evolution can uncover the patterns and characteristics exhibited by clones as they evolve within a system. Software practitioners can use the results of this study to understand and to manage the clones more efficiently. In this work, three BBs have identified for post-processing: Genealogy Generator 1, Genealogy Generator 2 and Genealogy Reconstructor (see Table 5).

5 CONCEPTUAL FRAMEWORK OF THE EXPERIMENTAL TESTBED

This section presents the conceptual foundation of a domain-specific framework to support CCDE experiments. The framework adheres to an extensible multi-layered abstraction mechanism that consists of the collection of BBs identified previously. The BBs are systematically organized on top of a collection of basic operators derived from relational algebra.

5.1 Stack of Building Blocks

As depicted in Figure 4, the Building Blocks stack consists of several layers, that are arranged based on the BBs identified in the previous section along with a newly introduced collection of Operators.

Problem	BB Name and Solution Overview	Alternatives	Predecessors	Follow-ups	References
Source code management systems (SCM) characteristically offer a rich version history of software projects. They give crude information about the origins of the code and it's change history. Such information indirectly provide code cloning information at various stages. Therefore, there is a need to gather data from version control repositories such as SVN, CVS and GIT.	**VCS Miner** It mines the version control repository of software systems.	SVN Miner, CVS Miner, GIT Miner	None	Token generator, Snapshot generator, Program model generator, VC migrator, Metrics generator, AST generator	**Xie et al [86]** : This paper analyzed change history of three software systems by mining SVN repositories. **Saha et al. [72]**: This paper analyzed six open source software systems by mining SVN repositories. **Aversano et al. [5]**: This paper extracted code from CVS repositories of two software systems (i.e., ArgoUML and DNSJava) to conduct and empirical study.
Mostly the original software system are stored in traditional SCMs (e.g., SVN, CVS). Decentralized source code management systems (e.g., GIT) can provide richer content histories than the traditional SCMs. Therefore, clone detection algorithms and tools might perform better with a DSCM repository.	**VC Migrator** It migrates data from traditional SCM repository to DCSM repository (e.g., from SVN to GIT)	None	SVN Miner, CVS Miner	Token generator, Snapshot generator, Program model generator, VC migrator, Metrics generator, AST generator	**Rahman et al. [65, 66]**: They migrated SVN and CVS repositories to GIT in order to speed up the process. This paper analyses relationship between cloning and defect proneness.

Table 2: Building blocks for data gathering

Problem	BB Name and Solution Overview	Alternatives	Predecessors	Follow-ups	References
Software evolution experiments need to capture revisions and releases of software systems for a considered time period in order to conduct various code clone detection experiments.	**Snapshot Generator** It can generate a set of snapshots of the software system for a mentioned time period.	release level snapshot generator, revision level snapshot generator	VCS miner	Clone detector, Test files remover	**Xie et al.** [**86**]: This paper extracts snapshots at each revision. **Saha et al.** [**72**]: This paper captures all minor and major releases. **Rahman et al.** [**65, 66**]: This paper requires snapshots in monthly revisions.
Software contains many test files that are used during the development of the system to test the different functionalities. Since test files are frequently copied and modified to test a different case, they can contain many clones.	**Test Files Remover** It removes test files from the subject systems.	None	Snapshot generator	Clone detector	**Barbour et al.** [**9**]: This paper removes test files from ARGOUML and ANT systems. **Xie et al.** [**86**]: This paper removes test files to avoid the clones that are not involved in normal executions.
Logical clones can reveal many business and programming rules that are often not properly documented during software development.Therefore, there is a need for making these rules explicit in order to improve the efficiency in software maintenance.	**Program Model Generator** It extracts the program model from the source code, which consists of methods, entity classes, etc.	None	VCS miner	Text based clone detector	**Qian et al.** [**64**]: This paper extracts a program model from the source code. That is used to analyze the business and programming rules of software applications.
The nodes of a PDG represent the statements and conditions of a program, while edges represent control and data dependencies. Extracting such information is vital to identify similar code fragments.	**Dependency Graph Generator** It generates the PDG of a given source file.	None	VCS miner	Dependency graph based clone detector	**Krinker** [**48**]: Authors have presented an approach based on PDGs, by symbolizing the basic structure of a program together with the corresponding data flow. **Horwitz** [**30**]: This paper investigates both syntactic and semantic differences of various versions of software by exploiting PDGs.

Continued on next page...

Table 3 – Continued

Problem	BB Name and Solution Overview	Alternatives	Predecessors	Follow-ups	References
In literature, several Abstract Syntax Tree based approaches have been proposed to automate the identification of software clones. During a parsing step, the algorithm should create an AST based representation of the source code.	**AST Generator** It parses the source code and generate AST representation of the source code.	None	VCS miner	AST based clone detector	**Corazza et al. [14]**: This paper proposes a methodology, which investigate ASTs as well as lexical information for discovering software clones up to Type 3. **Tairas and Gray [79]**: This paper proposes a clone detection methodology by utilizing AST representation.
Metrics-based approaches calculate a number of metrics and then compare them rather than directly comparing the source code or ASTs.	**Metrics Generator** It generates code metrics from the source code.	None	VCS miner	Metric-based clone detector	**Anatoniol et al. [4]**: This paper analyzes nineteen releases of Linux kernel to identify code duplication by means of the metrics. **Mayrand et al. [55]**: This paper proposes a metrics-based technique to automatically discover duplicate (or near duplicate) functions in software systems.

Continued on next page...

Table 3 – Continued

Problem	BB Name and Solution Overview	Alternatives	Predecessors	Follow-ups	References
Token-based clone detectors are considered better than simple keyword matches [67]. In these techniques, lexical analysis is primarily used to extract the tokens from the source code.	**Token Generator** It generates tokens from the source code.	None	VCS miner	Token-based clone detector	**Baker [7]**: In research uses a lexical analyzer to divide the lines of source files into tokens. These tokens are then split into parameter tokens and non-parameter tokens. **Li et al. [52]**: In this approach, statements are mapped to numbers by first tokenizing its components, such as variables, operators, constants, functions, keywords, etc. **Wang et al. [85]**: Proposed CCAligner: a token based large-gap clone detector.

Table 3: Building blocks for pre-processing

Problem	BB Name and Solution Overview	Alternatives	Predecessors	Follow-ups	References
Identifying software clones and understanding how software changes between releases are two important issues for maintainers where a text-based approach is likely to be useful.	**String Based Clone Detector** It detects code clones by string matching.	None	Snapshot generator, Program model generator	Clone genealogy generator 1, Clone genealogy generator 2	**Johnson [37]**: This paper considers the source as text and analyze it the way documents are analyzed to detect the code clones. **Ossher et al. [62]**: This paper proposes a file-level clone detection method by combining three simple string matching techniques.
After creating the AST from the source code, similar subtrees need to be identified as code clones. In this case, differences of various coding styles as well as variable names are ignored.	**AST Based Clone Detector** It detects code clones from AST representations.	None	AST generator	Clone genealogy generator 1, Clone genealogy generator 2	**Corazza et al. [14]**: This paper exploits together AST and lexical information to identify software clones. **Baxter et al. [11]**: This paper proposes an AST based methodology to detect near-miss clones.
Metrics-based techniques calculate a number of metrics for code fragments and then compare metrics vectors rather than the source code or ASTs directly.	**Metrics Based Clone Detector** It detects code clones from source code metrics.	None	Metrics generator	Clone genealogy generator 1, Clone genealogy generator 2	**Mayrand et al. [55]**: This paper considers both control flow metrics and data flow metrics to detect code clones.
Semantic approaches on code clone detection vastly rely on the the PDG of the source code. Then subgraph matching techniques are used to identify the code clones in a program.	**Graph Based Clone Detector** It detects code clones from tokens.	None	Program dependency graph generator	Clone genealogy generator 1, Clone genealogy generator 2	**Komondoor and Horwitz [44]**: This paper represents each function in the source code using its PDG, which could then be used to find the code clones. **Krinke [48]**: This paper attempts to identify similar subgraph structures that are stemming from duplicated code.

Continued on next page...

Table 4 – Continued

Problem	BB Name and Solution Overview	Alternatives	Predecessors	Follow-ups	References
Token-based approaches are naturally language-independent and also considered as low-cost. Further it produces faster results because they only need to transform the source code into tokens, without the need to construct ASTs or PDGs.	**Token Based Clone Detector** It detects code clones from tokens.	None	Token generator	Clone genealogy generator 1, Clone genealogy generator 2	**Kamia et al. [40]**: CCFinder is an outcome of this research. **Basit et al. [10]**: This paper proposes a simple and customizable tokenization mechanism for code clone detection. **Jalbert and Bradbury [35]**: This paper proposes a methodology to identify bugs using ConQAT [39], which is a token based clone detector.

Table 4: Building blocks for clone detection

Problem	BB Name and Solution Overview	Alternatives	Predecessors	Follow-ups	References
Code clone genealogies show how clone groups evolve with the evolution of a software system over multiple versions of the program. Therefore, it could provide important insights on the maintenance implications of code clones.	**Genealogy Generator 1** It creates clone genealogies from the detected clones.	Clone genealogy generator 2	Clone detector	None	**Kim and Notkin [42]**: Authors of this paper have presented an approach to discover clone genealogy by exploiting the cloning relationships among all consecutive versions. **Adar and Kim [1]**: They create clone genealogies, allowing visually and programmatically analyzing code clones.
Code clone genealogies show how clone groups evolve with the evolution of a software system over multiple versions of the program. Therefore, it could provide important insights on the maintenance implications of code clones.	**Genealogy Generator 2** It removes invalid clones prior to generating clone genealogies.	Clone genealogy generator 1	Clone detector	Genealogy reconstructor	**Xie et al. [86]**: Prior to building the clone genealogies, this study removes all the unchanged and invalid (i.e., code segments belong to the same method).
Clone genealogy generation is usually done using tools such as gCad. Such tools could sometimes provide erroneous results. Therefore, manual verification is indeed important.	**Genealogy Reconstructor** It manually verifies the correctness of the results and remove the false positives.	None	Clone genealogy generator 1/2	None	**Saha et al. [72]**: They run gCad for the second time to reconstruct the genealogies after removing the false positives manually.

Table 5: Building blocks for clone evolution

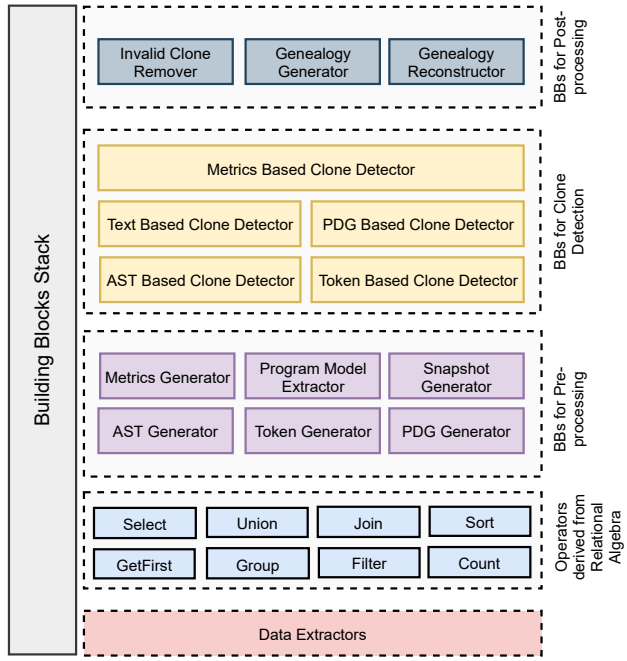


Figure 4. Building blocks stack supported by the operators derived from relational algebra

These operators are directly derived from relational algebra. Relational algebra is a procedural query language, which operates on input relations. It consists with a set of fundamental operators such as `select`, `project`, `union` and `cartesian product`. Though several relational algebra theorems do not strictly hold in query languages such as SQL and LINQ, they are the native implementations of the underline concept of relational algebra. Therefore, we borrowed some ideas from such languages to identify basic operators supported by relational algebra. In this paper, operators such as `filter`, `select`, `join`, `sort`, `count`, etc. has been categorized as basic level operators. Such operators are useful in conducting CCDE experiments.

5.2 Architectural Overview of the Experimental Testbed

Figure 5 presents the architectural overview of the experimental testbed to conduct CCDE experiments. It consists of two main components: BBs repository, and workflow composition and execution engine. BBs repository contains all the BBs (i.e., BBs for data extraction, pre-processing, clone detection, and clone evolution) and a useful collection of *Operators* that are described previously. The purpose of

Operators is to facilitate the basic functionalities such as counting or filtering. Once the BBs and Operators are defined, CCDE experiments can be accomplished by pipelining the required BBs and *operators*. Workflow composition and execution engine is responsible for translating the CCDE workflow defined by a user into an executable process. Finally, the analysis results will be presented to the user.

For the workflow generation, all BBs and Operators are defined directly on an underlying logical representation, a static grammar. Static grammar consists of the production rules to combine BBs and Operators into a meaningful workflow, which strictly follows the connections in Figure 3. For example, *AST based clone detector* can be directly composed with the *AST generator*. However, it cannot be composed with *PDG generator*. Static grammar has to be defined manually and should be evolved with the introduction of new BBs and Operators.

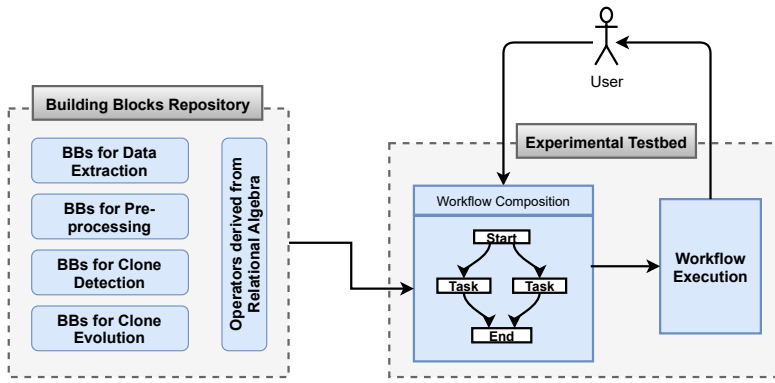


Figure 5. Architectural overview of the experimental testbed

6 VALIDATION

Our vision is to introduce a collection of reusable BBs that are derived from the state-of-the-art code clone detection and evolution research and efficiently utilize them in developing an experimental testbed to conduct CCDE experiments systematically and conveniently. In this section, we sought to validate the two research questions. RQ1 mainly focuses on identifying building blocks from existing CCDE experiments, which could reuse in new ventures. For that, a case study based evaluation is employed to show how a particular CCDE experiment can represent by utilizing the identified BBs. To validate RQ2, a simple prototype was implemented to demonstrate how to develop an experimental testbed to utilize BBs effectively. The prototype was validated with a usage scenarios for three open source projects. Finally, the extensibility of the proposed approach in conducting a diverse range of software analytics experiments is examined.

6.1 Reusability of BBs in CCDE Experiments

As described previously, the BBs have identified by the literature survey conducted on the papers published in ICSE, ICSM, MSR and FSE conferences for the last eight years. Therefore, for the validation purpose, four journal papers on code clone detection have selected as case studies. Then, each experimental procedures were represented as a workflow by utilizing the identified BBs. For the selected case studies, it was evident that one experiment can be fully expressed and three experiments can be partially represented using BBs.

Case Study 1 – Kontogiannis et al. [45]

Summary: Authors of this paper have presented a number of pattern matching techniques by using ASTs as the code representation scheme that could use for both code-to-code as well as concept-to-code matching. Metric-based clone detection technique has used in the study by taking three medium-sized C programs (i.e., tcsh, bash and CLIPs) as the subject systems. First, the source code is parsed to create the Abstract Syntax Tree (AST). Five different metrics have calculated for every statement, block, function, and file stored as annotations in the nodes of the AST. As the next step, a reference table was maintained, which consists of source code entities sorted by their associated metric values.

Representation Using BBs: In this experiment, VCS miner considered the BB for data gathering. Pre-processing step is covered with two BBs namely AST generator and Metrics generator. Metrics based clone detector is the responsible BB in the clone detection phase. Thus, the experimental design of the above paper can be fully represented using four main BBs, as shown in Figure 6.

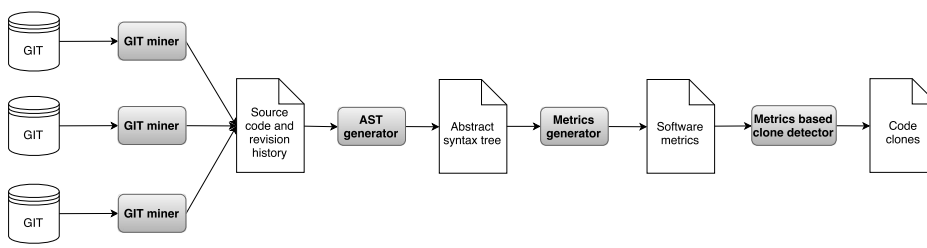


Figure 6. Kontogiannis's [45] approach using BBs

Case Study 2 – Anatoniol et al. [4]

Summary: This paper studies the evolution of code duplications in the Linux kernel. The paper followed a functional level metric-based approach to analyze nineteen releases to identify code duplication among Linux subsystems.

Representation Using BBs: Figure 7 is an example how BBs can be used to partially representing a previously conducted experimental design. Authors of this paper have described mechanisms to handle preprocessor directives as well as to handle the functions in the C code of the Linux kernel. Such tasks come under the above mentioned Pre-processing activity. However, at the moment, the exact BB to perform this task is not available in our BBs catalog. As stated before, our approach will evolve with time and build its BBs catalog. Therefore, one can define a new BB and add to our BBs catalog. However, the above experiment can partially represent by utilizing VCS miner, Metrics generator and Metrics based clone detector.

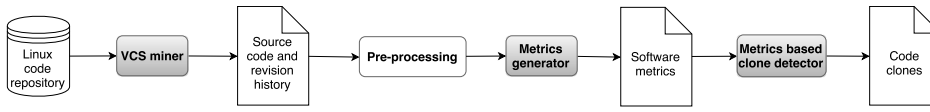


Figure 7. Anatoniol's [4] approach using BBs

Case Study 3 – Geiger et al. [25]

Summary: In this paper, the authors examined whether a correlation exists between code clones and code change. The steps of this research include code clone detection, categorization into clone types, extraction of change couplings, and computing a relation metric. The proposed framework has validated with the Mozilla project. The results show that a reasonable number of cases can found where such a relation exists.

Representation Using BBs: Part of the experiment of this research can represent using BBs, as shown in Figure 8. In this paper, authors have used CCFinder as the clone detection tool. In Figure 8, we further describe the tasks in CCFinder as a BPMN 2.0 subprocess. VCS miner, Token generator, and Token-based clone detector have used in this expanded representation.

Case Study 4 – Kanwal et al. [41]

Summary: This paper investigates the evolution of structural clones by conducting a longitudinal analysis of several versions of Java systems. The authors have defined structural clones and their evolution patterns in a formal notation. The trends in the patterns reveal that evolutionary characteristics of structural clones can facilitate better clone management systems.

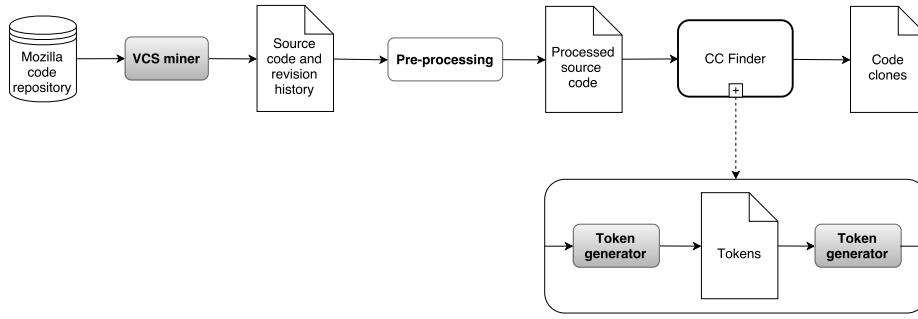


Figure 8. Geiger's [25] approach using BBs

Representation Using BBs: As depicted in Figure 9, the experiment can be effectively represent with the identified BBs. The above experiment can be partially represented using VCS Miner, Token generator, Token based clone detector, and Genealogy generator.

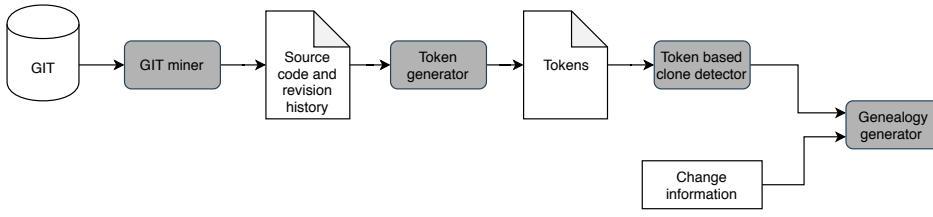


Figure 9. Kanwal's [41] approach using BBs

Summary of the evaluation results is shown in Table 6. Based on the case studies, the RQ1 can be addressed, and we claim that it is a serious first proof of the usefulness of the proposed BBs. The selection of case studies is based on the clone detection and evolution experiments spanning from the year 1996 to 2019 denoting the applicability of the proposed approach in the future clone detection and evolution experiments.

6.2 Usage Scenario in the Experimental Testbed

Clone analysis over multiple versions and releases is a major component in many CCDE experiments [75, 56]. Such studies would reveal the trends over time as well as the relationship between code size and the number of code clones for large-scale software projects [13]. Below we show how to use the experimental testbed to find the code clone percentage over multiple versions of a software project.

In order to find code clones over multiple versions, the following tasks have to perform in the given order. First, project history for a given version/release needs

Title of the journal paper	**Used BBs**	**Graphical Representation**
Pattern matching for clone and concept detection. Kontogiannis et al. [45]	VCS Miner AST Generator Metrics Generator Metrics Based Clone Detector	Figure 4
Analyzing cloning evolution in the Linux kernel. Anatoniol et al. [4]	VCS Miner Metrics Generator Metrics based Clone Detector	Figure 5
Relation of code clones and change couplings Geiger et al. [25]	VCS Miner Token Generator Token Based Clone Detector	Figure 6
Evolutionary Perspective of Structural Clones in Software Kanwal et al. [41]	VCS Miner Token Generator Token Based Clone Detector Genealogy generator	Figure 7

Table 6. Summary of the case study based validation

to be extracted from the version control repository using VCS Miner. Second, it is converted to an intermediate data-model using one of the pre-processing BBs. Then the results are fed to a Clone Detector to detect the duplicates. Finally, the steps mentioned above are repeated for several versions of the software system. In the prototype implementation, the BBs can be pipelined as a workflow and run the analysis. Additional BBs (i.e., filter, loop) can be implemented to facilitate rich analyses based on complex conditions. As such, a user needs to drag the BBs to the canvas and combine them using linkers and run it. Three Apache projects have been selected for the experiment; Apache Commons Lang[5], Apache Tomcat[6] and Apache Wink[7]. Figure 10 presents the cloning behavior for the years 2014–2016.

However, by no means, this is a complex CCDE experiment. But, still, it answers RQ2 by evidently demonstrating how BBs can be used in the proposed experimental testbed to produce useful insights to the researchers.

6.3 Extensibility of Experiment Testbed for Software Engineering Experiments

The core idea behind BBs and the conceptual framework of the experimental testbed is not strictly limited to CCDE research. The proposed architecture of the testbed along with the composition logic of BBs provide versatility for extending the experimental testbed for other types of software engineering experiments. However,

[5] https://commons.apache.org/proper/commons-lang/
[6] http://tomcat.apache.org/
[7] https://wink.apache.org/

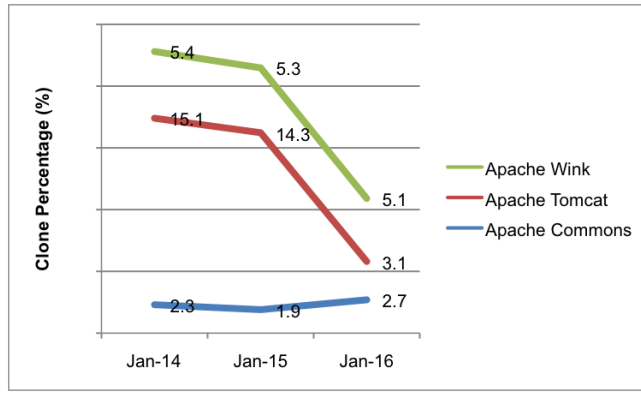


Figure 10. Clone percentage for 2014–2016

it requires a formal arrangement of BBs into several layers. Figure 11 presents the proposed extended stack of BBs that could be used in different software engineering experiments. The extended BBs stack for software analytics has multiple layers: Primary BBs, Secondary BBs, and Advanced BBs.

Below we demonstrate how to build the logic to perform a software analysis task by utilizing the BBs from the BBs Stack.

Analysis Task: Finding critical issues resolved by most frequent committer in a project.

Background: Measuring the performance of the developers who work in a project is a challenging task for the project managers when the team size is large and the nature of the project is complex. However, it is notable that total lines of codes, the number of bugs fixed, the total number of commits, or a combination of them could produce useful insights into performance.

Implementation Using BBs: Figure 12 presents how to utilize the BBs to perform the above analysis task. It demonstrates how the data is integrated from both version control and bug tracking repositories to find how many critical bugs have been fixed by the most frequent committer.

We further tested the above scenario with three open source projects by using the prototype implementation. The prototype allows users to utilize the BBs to perform the tasks directly. Therefore, it provides a great level of convenience to the users. Summary of the experimental results present in Table 7.

As shown in the Figure 12, FindMax, which is a Secondary BB, is formulated by utilizing three Primary BBs. Thus, Secondary BBs, on the ohter hand, can be considered as composite BBs.

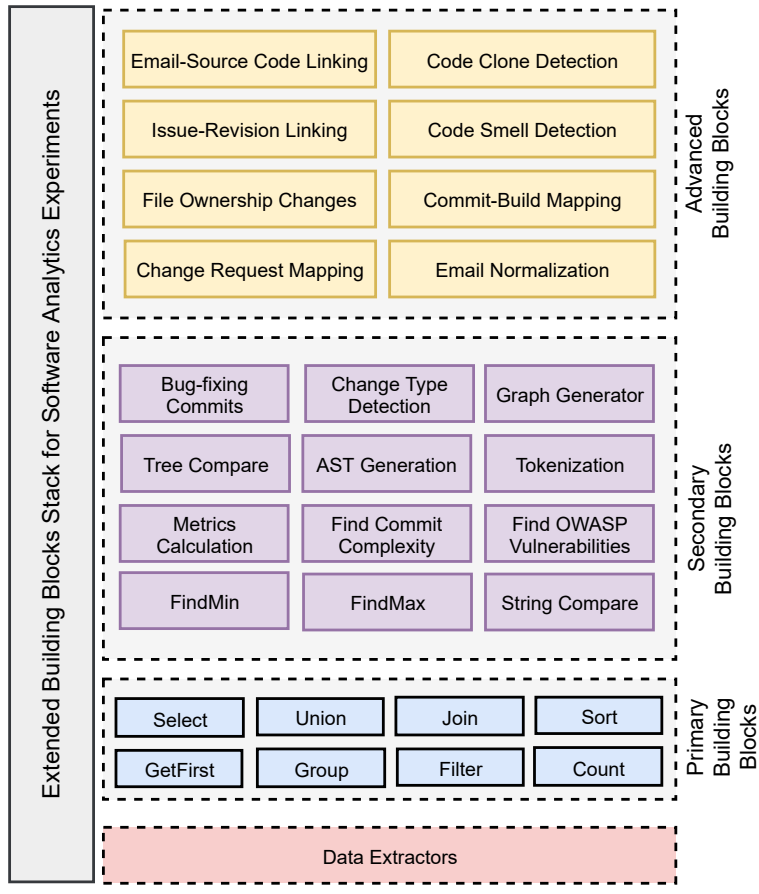


Figure 11. Extended building blocks stack for software analytics

7 DISCUSSION

In this paper, we provide a catalog of Building Blocks on which the clone detection research can be carried out. A particular BB represents a specific analysis task in any CCDE experiment. Based on that, we demonstrated that such distinctive BBs could properly arrange as workflows to perform a wide range of CCDE experiments. From the selected case studies for the validation, it was evident that CCDE experiments can either wholly or partially represent by means of BBs. Therefore, this approach is a step towards standardization of CCDE research by providing a structured way to conduct experiments. Our approach has multifold benefits and is worth further exploration.

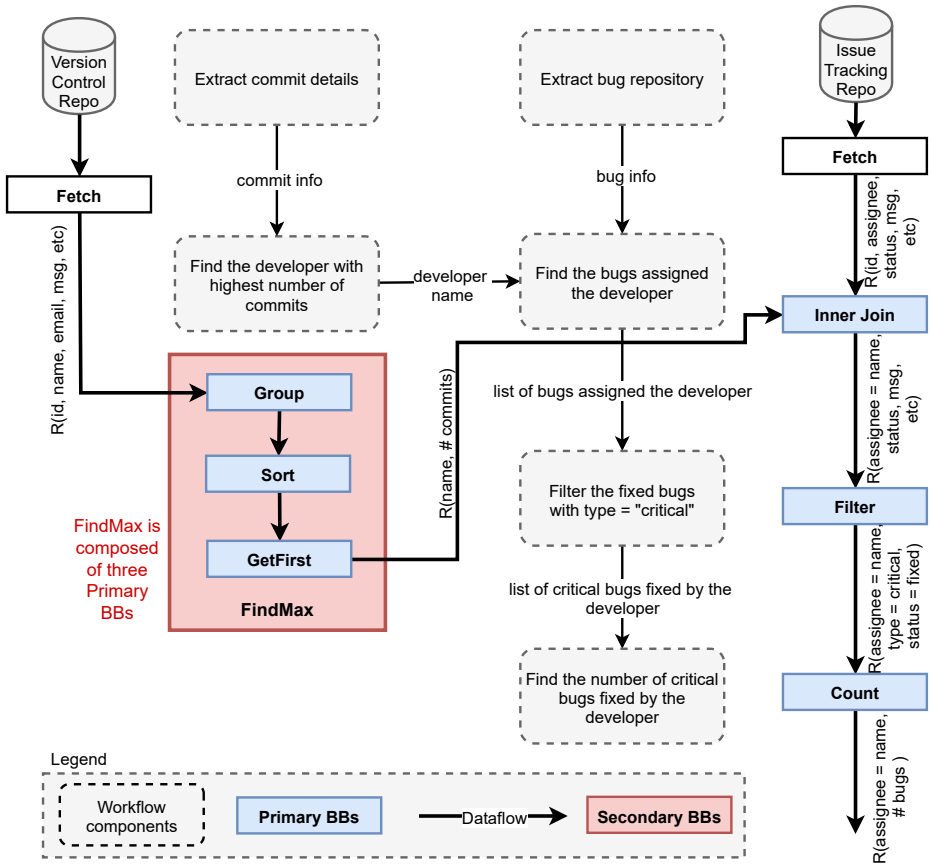


Figure 12. Finding the number of critical issues resolved by the most frequent committer in a project

Guideline for Novice Researchers. This paper does not target providing a comprehensive literature review in the area of CCDE. Most importantly, it presents some useful conceptual and practical insights to novice researchers by allowing them to use BBs as a guide to carrying out CCDE experiments. Novice researchers can make use of BBs to conduct experiments in a quick and community accepted way. Having a prior understanding of the BBs will help them comprehend the published CCDE research approaches and recognize the essential background requirements; hence, can better plan their experiments. Further analyzing the usages of those BBs in different analysis scenarios will help them in running successful experiments.

Helping Overcome Common Problems in CCDE Experiments. Conducting CCDE experiments presents a number of common difficulties and challenges to

Apache Project	No. of Commits	Frequent Developer	No. of Bug Fixes
Gora	1 053	Developer A	52
Commons-lang	5 171	Developer B	4
IO	2 091	Developer C	0
Winx	1 312	Developer D	2

Table 7. Summary of the experimental results

researchers such as:

1. mechanism to locate the repositories to gather accurate and timely data,
2. filtering or converting such data to different formats,
3. exploring various analysis to be performed on such data, and
4. effectively running such analyses.

The concept of Activities and Building Blocks is beneficial to overcome such exertions. For example, BBs for data gathering facilitates a mechanism to locate and extract data from repositories. Similarly, BBs for pre-processing provide ways to convert and filter data. BBs for clone detection solves the difficulty in exploring distinctive analysis on such data. In that way, our approach simplifies the challenges mentioned above and provides a structured way to conduct software analysis experiments.

Facilitating Comparison. Several imperative systematic literature reviews have published on software clones in general and software clone detection in particular. These approaches typically focus on only some traits of categorization, and most of them do not rely on an explicit high level meta-model. Therefore, there is a need of a model, which facilitates the comparison of different clone detection approaches at the experimental level. In this paper, we present a meta-model infrastructure for representing, combining and comparing such experiments in a structured way.

Fostering the Replication of Studies. The replication of such studies is just as fundamental and is one of the main threats to validity that empirical software engineering suffers. Such threats are manifold and range from lack of independent validation of the results, unavailability of the tools and methodologies used, to no impossibility to generalize the gained knowledge. Though this paper does not provide a fully functional framework for replication, still it presents ways to better plan the replication studies and reveal the imprecise descriptions in *Methodology* sections of research publications.

Based on the nature of the BBs, it is important to realize that the proposed approach works only with syntactically similar code clones.For example, as described in the BBs for pre-processing, the entire detection process is facilitated by ASTs, PDGs, tokens, metrics, program models, and snapshots. Thus the BBs for clone detection facilitates only the syntactically similar code clones. However, the detec-

tion of semantically similar code clones requires a new set of BBs that are capable of inferring the associations across functionally similar code clones.

Several recent studies have reported on cross-language code clone detection [58, 88, 59]. For example, LICCA, a tool for cross-language clone detection [82] is based on a tree-based intermediate representation of the source code. Thus, the proposed BBs for pre-processing can be used for this purpose. However, this direction has to further investigate to identify a useful set of BBs for cross-language clone detection.

Besides, visualization of the results produced by software analytics is considered important nowadays [81, 19]. Also, recent studies have highlighted the importance of visualizing the differences between the versions of software models [61]. Thus, the proposed BBs stack for software analytics has provisions to augment with new BBs that could be used to facilitate the visualization aspects of software analytics experiments.

8 CONCLUSIONS

This paper introduced a concrete set of formal constructs, which we refer to as *Building Blocks*, which can be used to conduct various CCDE experiments. These *Building Blocks* provide a structured way to conduct experiments, hence it offers direct solutions to everyday challenges in code clone detection, such as accurate data collection, data cleaning, and selecting proper CCD algorithms. Our goal is not to introduce novel CCD algorithms or report the loopholes in the existing CCDE research, but to provide a systematic understanding of how CCDE experiments are conducted in practice by utilizing the identified *Building Blocks*. *Building Blocks* are represented using both textual and graphical representation, which provide means to software researchers to conduct or replicate CCDE experiments in an unambiguous manner. The conceptual framework of the experimental testbed indicates the usefulness and the replication capabilities of *Building Blocks* and is proven useful in conducting CCDE experiments. Besides, this paper presents how the stack of *Building Blocks* can be extended to facilitate a wide range of software analytics experiments beyond CCDE.

Future work of this research will focus on enhancing the experimental testbed to a point where we can conduct a field study with professional software practitioners in the industry. By doing that it is expected to obtain the future potentials and the limitations of the experimental testbed in practice. In that way, useful insights can be gained to convert our testbed to a full-fledged software evolution analysis testbed.

Acknowledgements

The authors of this paper gratefully acknowledge the financial support provided by the National Research Council of Sri Lanka (Grant No. NRC 15-74). We thankfully

acknowledge the insights and expertise provided by the colleagues at SEAL Lab at the University of Zurich.

REFERENCES

[1] Adar, E.—Kim, M.: SoftGUESS: Visualization and Exploration of Code Clones in Context. 29th International Conference on Software Engineering (ICSE '07), 2007, pp. 762–766, doi: 10.1109/ICSE.2007.76.

[2] Agrawal, A.—Yadav, S. K.: A Hybrid-Token and Textual Based Approach to Find Similar Code Segments. 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), 2013, pp. 1–4, doi: 10.1109/ICCCNT.2013.6726700.

[3] Ain, Q. U.—Butt, W. H.—Anwar, M. W.—Azam, F.—Maqbool, B.: A Systematic Review on Code Clone Detection. IEEE Access, 2019, Vol. 7, pp. 86121–86144, doi: 10.1109/ACCESS.2019.2918202.

[4] Antoniol, G.—Villano, U.—Merlo, E.—Di Penta, M.: Analyzing Cloning Evolution in the Linux Kernel. Information and Software Technology. Vol. 44, 2002, No. 13, pp. 755–765, doi: 10.1016/S0950-5849(02)00123-4.

[5] Aversano, L.—Cerulo, L.—Di Penta, M.: How Clones Are Maintained: An Empirical Study. 11th European Conference on Software Maintenance and Reengineering (CSMR '07), 2007, pp. 81–90, doi: 10.1109/CSMR.2007.26.

[6] Baker, B. S.: A Program for Identifying Duplicated Code. Computing Science and Statistics, 1993, pp. 49–49.

[7] Baker, B. S.: On Finding Duplication and Near-Duplication in Large Software Systems. Proceedings of 2nd Working Conference on Reverse Engineering, 1995, pp. 86–95, doi: 10.1109/WCRE.1995.514697.

[8] Barbour, L.—Khomh, F.—Zou, Y.: An Empirical Study of Faults in Late Propagation Clone Genealogies. Journal of Software: Evolution and Process, Vol. 25, 2013, No. 11, pp. 1139–1165, doi: 10.1002/smr.1597.

[9] Barbour, L.—Khomh, F.—Zou, Y.: Late Propagation in Software Clones. 2011 27th IEEE International Conference on Software Maintenance (ICSM), 2011, pp. 273–282, doi: 10.1109/ICSM.2011.6080794.

[10] Basit, H. A.—Jarzabek, S.: Efficient Token Based Clone Detection with Flexible Tokenization. Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), 2007, pp. 513–516, doi: 10.1145/1287624.1287698.

[11] Baxter, I. D.—Yahin, A.—Moura, L.—Sant'Anna, M.—Bier, L.: Clone Detection Using Abstract Syntax Trees. International Conference on Software Maintenance, 1998, pp. 368–377, doi: 10.1109/ICSM.1998.738528.

[12] Buehrer, G.—Weide, B. W.—Sivilotti, P. A. G.: Using Parse Tree Validation to Prevent SQL Injection Attacks. 5th International Workshop on Software Engineering and Middleware (SEM '05), 2005, pp. 106–113, doi: 10.1145/1108473.1108496.

[13] Chen, X.—Wang, A. Y.—Tempero, E.: A Replication and Reproduction of Code Clone Detection Studies. Proceedings of the Thirty-Seventh Australasian Computer Science Conference (ACSC 2014), Conferences in Research and Practice in Information Technology (CRPIT), Vol. 147, 2014, pp. 105–114.

[14] Corazza, A.—Di Martino, S.—Maggio, V.—Scanniello, G.: A Tree Kernel Based Approach for Clone Detection. IEEE International Conference on Software Maintenance (ICSM), 2010, pp. 1–5, doi: 10.1109/ICSM.2010.5609715.

[15] Cordy, J. R.: Comprehending Reality – Practical Barriers to Industrial Adoption of Software Maintenance Automation. 11th IEEE International Workshop on Program Comprehension, 2003, pp. 196–205, doi: 10.1109/WPC.2003.1199203.

[16] Cosentino, V.—Izquierdo, J. L. C.—Cabot, J.: Gitana: A SQL-Based Git Repository Inspector. In: Johannesson, P., Lee, M., Liddle, S., Opdahl, A., Pastor López, Ó. (Eds.): Conceptual Modeling (ER 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9381, 2015, pp. 329–343, doi: 10.1007/978-3-319-25264-3_24.

[17] D'Ambros, M.: Supporting Software Evolution Analysis with Historical Dependencies and Defect Information. IEEE International Conference on Software Maintenance, 2008, pp. 412–415, doi: 10.1109/ICSM.2008.4658092.

[18] Deelman, E.—Gil, Y.: Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges. 2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science '06), 2006, p. 144, doi: 10.1109/E-SCIENCE.2006.261077.

[19] Dominic, J.—Tubre, B.—Houser, J.—Ritter, C.—Kunkel, D.—Rodeghero, P.: Program Comprehension in Virtual Reality. Proceedings of the 28th International Conference on Program Comprehension (ICPC '20), 2020, pp. 391–395, doi: 10.1145/3387904.3389287.

[20] Dou, W.—Cheung, S. C.—Gao, C.—Xu, C.—Xu, L.—Wei, J.: Detecting Table Clones and Smells in Spreadsheets. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016), 2016, pp. 787–798, doi: 10.1145/2950290.2950359.

[21] Ducasse, S.—Rieger, M.—Demeyer, S.: A Language Independent Approach for Detecting Duplicated Code. IEEE International Conference on Software Maintenance (ICSM '99), 1999, pp. 109–118, doi: 10.1109/ICSM.1999.792593.

[22] Dyer, R.—Nguyen, H. A.—Rajan, H.—Nguyen, T. N.: Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories. Proceedings of the 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 422–431, doi: 10.1109/ICSE.2013.6606588.

[23] Ferrante, J.—Ottenstein, K. J.—Warren, J. D.: The Program Dependence Graph and Its Use in Optimization. ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 9, 1987, No. 3, pp. 319–349, doi: 10.1145/24039.24041.

[24] Fowler, M.: Refactoring: Improving the Design of Existing Code. Pearson Education India, 1999.

[25] Geiger, R.—Fluri, B.—Gall, H. C.—Pinzger, M.: Relation of Code Clones and Change Couplings. In: Baresi, L., Heckel, R. (Eds.): Fundamental Approaches to Software Engineering (FASE 2006). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3922, 2006, pp. 411–425, doi: 10.1007/11693017_31.

[26] Göde, N.—Harder, J.: Clone Stability. 2011 15th European Conference on Software Maintenance and Reengineering (CSMR), 2011, pp. 65–74, doi: 10.1109/CSMR.2011.11.

[27] Gousios, G.—Spinellis, D.: Conducting Quantitative Software Engineering Studies with Alitheia Core. Empirical Software Engineering, Vol. 19, 2014, No. 4, pp. 885–925, doi: 10.1007/s10664-013-9242-3.

[28] Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1997, doi: 10.1017/CBO9780511574931.

[29] Higo, Y.—Kusumoto, S.: Code Clone Detection on Specialized PDGs with Heuristics. 2011 15th European Conference on Software Maintenance and Reengineering (CSMR), 2011, pp. 75–84, doi: 10.1109/CSMR.2011.12.

[30] Horwitz, S.: Identifying the Semantic and Textual Differences Between Two Versions of a Program. ACM SIGPLAN Notices, Vol. 25, 1990, No. 6, pp. 234–245, doi: 10.1145/93542.93574.

[31] Hu, Y.—Wang, H.—Zhang, Y.—Li, B.—Gu, D.: A Semantics-Based Hybrid Approach on Binary Code Similarity Comparison. IEEE Transactions on Software Engineering, Vol. 47, 2021, No. 6, pp. 1241–1258, doi: 10.1109/TSE.2019.2918326.

[32] Hu, Y.—Zhang, Y.—Li, J.—Wang, H.—Li, B.—Gu, D.: BinMatch: A Semantics-Based Hybrid Approach on Binary Code Clone Analysis. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018, pp. 104–114, doi: 10.1109/ICSME.2018.00019.

[33] Hummel, B.—Juergens, E.—Heinemann, L.—Conradt, M.: Index-Based Code Clone Detection: Incremental, Distributed, Scalable. 2010 IEEE International Conference on Software Maintenance, 2010, pp. 1–9, doi: 10.1109/ICSM.2010.5609665.

[34] Islam, J. F.—Mondal, M.—Roy, C. K.—Schneider, K. A.: Comparing Bug Replication in Regular and Micro Code Clones. 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 2019, pp. 81–92, doi: 10.1109/ICPC.2019.00022.

[35] Jalbert, K.—Bradbury, J. S.: Using Clone Detection to Identify Bugs in Concurrent Software. 2010 IEEE International Conference on Software Maintenance (ICSM), 2010, pp. 1–5, doi: 10.1109/ICSM.2010.5609529.

[36] Johnson, J. H.: Identifying Redundancy in Source Code Using Fingerprints. Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering (CASCON '93), Vol. 1, 1993, pp. 171–183.

[37] Johnson, J. H.: Substring Matching for Clone Detection and Change Tracking. 1994 International Conference on Software Maintenance, 1994, pp. 120–126, doi: 10.1109/ICSM.1994.336783.

[38] Johnson, J.: Visualizing Textual Redundancy in Legacy Source. Proceedings of the 1994 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering (CASCON '94), 1994, pp. 32–41.

[39] Juergens, E.—Deissenboeck, F.—Hummel, B.: CloneDetective – A Workbench for Clone Detection Research. Proceedings of the IEEE 31st International Conference on Software Engineering, 2009, pp. 603–606, doi: 10.1109/ICSE.2009.5070566.

[40] Kamiya, T.—Kusumoto, S.—Inoue, K.: CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE Transactions on Software Engineering, Vol. 28, 2002, No. 7, pp. 654–670, doi: 10.1109/TSE.2002.1019480.

[41] Kanwal, J.—Maqbool, O.—Basit, H. A.—Sindhu, M. A.: Evolutionary Perspective of Structural Clones in Software. IEEE Access, Vol. 7, 2019, pp. 58720–58739, doi: 10.1109/ACCESS.2019.2913043.

[42] Kim, M.—Notkin, D.: Using a Clone Genealogy Extractor for Understanding and Supporting Evolution of Code Clone. ACM SIGSOFT Software Engineering Notes, Vol. 30, 2005, No. 4, pp. 1–5, doi: 10.1145/1083142.1083146.

[43] Kim, M.—Sazawal, V.—Notkin, D.—Murphy, G.: An Empirical Study of Code Clone Genealogies. ACM SIGSOFT Software Engineering Notes, Vol. 30, 2005, No. 5, pp. 187–196, doi: 10.1145/1095430.1081737.

[44] Komondoor, R.—Horwitz, S.: Using Slicing to Identify Duplication in Source Code. In: Cousot, P. (Ed.): Static Analysis (SAS 2001). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2126, 2001, pp. 40–56, doi: 10.1007/3-540-47764-0_3.

[45] Kontogiannis, K. A.—DeMori, R.—Merlo, E.—Galler, M.—Bernstein, M.: Pattern Matching for Clone and Concept Detection. Automated Software Engineering, Vol. 3, 1996, No. 1-2, pp. 77–108, doi: 10.1007/BF00126960.

[46] Koschke, R.: Survey of Research on Software Clones. In: Koschke, R., Merlo, E., Walenstein, A. (Eds.): Duplication, Redundancy, and Similarity in Software. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Dagstuhl Seminar Proceedings, 2007, pp. 368–377.

[47] Krinke, J.: A Study of Consistent and Inconsistent Changes to Code Clones. 14th Working Conference on Reverse Engineering (WCRE 2007), 2007, pp. 170–178, doi: 10.1109/WCRE.2007.7.

[48] Krinke, J.: Identifying Similar Code with Program Dependence Graphs. Eighth Working Conference on Reverse Engineering, 2001, pp. 301–309, doi: 10.1109/WCRE.2001.957835.

[49] Lee, S.—Jeong, I.: SDD: High Performance Code Clone Detection System for Large Scale Source Code. Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '05), 2005, pp. 140–141, doi: 10.1145/1094855.1094903.

[50] Leitão, A.: Detection of Redundant Code Using R^2D^2. Software Quality Journal, Vol. 12, 2004, No. 4, pp. 361–382, doi: 10.1023/B:SQJO.0000039793.31052.72.

[51] Li, L.—Feng, H.—Zhuang, W.—Meng, N.—Ryder, B.: CCLearner: A Deep Learning-Based Clone Detection Approach. 2017 IEEE International Conference on

Software Maintenance and Evolution (ICSME), 2017, pp. 249–260, doi: 10.1109/ICSME.2017.46.

[52] Li, Z.—Lu, S.—Myagmar, S.—Zhou, Y.: CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. IEEE Transactions on Software Engineering, Vol. 32, 2006, No. 3, pp. 176–192, doi: 10.1109/TSE.2006.28.

[53] Lu, S.—Zhang, J.: Collaborative Scientific Workflows Supporting Collaborative Science. International Journal of Business Process Integration and Management (IJBPIM), Vol. 5, 2011, No. 2, pp. 185-199, doi: 10.1504/IJBPIM.2011.040209.

[54] Manber, U.—Myers, G.: Suffix Arrays: A New Method for On-Line String Searches. SIAM Journal on Computing, Vol. 22, 1993, No. 5, pp. 935–948, doi: 10.1137/0222058.

[55] Mayrand, J.—Leblanc, C.—Merlo, E. M.: Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. Proceedings of the 1996 International Conference on Software Maintenance (ICSM '96), 1996, pp. 244–253, doi: 10.1109/ICSM.1996.565012.

[56] Mui, H. H.—Zaidman, A.—Pinzger, M.: Studying Late Propagations in Code Clone Evolution Using Software Repository Mining. Electronic Communications of the EASST, Vol. 63, 2014, doi: 10.14279/tuj.eceasst.63.916.

[57] Nafi, K. W.—Kar, T. S.—Roy, B.—Roy, C. K.—Schneider, K. A.: CLCDSA: Cross Language Code Clone Detection Using Syntactical Features and API Documentation. 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1026–1037, doi: 10.1109/ASE.2019.00099.

[58] Nafi, K. W.—Roy, B.—Roy, C. K.—Schneider, K. A.: A Universal Cross Language Software Similarity Detector for Open Source Software Categorization. Journal of Systems and Software, Vol. 162, 2020, Art. No. 110491, doi: 10.1016/j.jss.2019.110491.

[59] Nichols, L.—Emre, M.—Hardekopf, B.: Structural and Nominal Cross-Language Clone Detection. In: Hähnle, R., van der Aalst, W. (Eds.): Fundamental Approaches to Software Engineering (FASE 2019). Springer, Cham, Lecture Notes in Computer Science, Vol. 11424, 2019, pp. 247–263, doi: 10.1007/978-3-030-16722-6_14.

[60] Noonan, R. E.: An Algorithm for Generating Abstract Syntax Trees. Computer Languages, Vol. 10, 1985, No. 3-4, pp. 225–236, doi: 10.1016/0096-0551(85)90018-9.

[61] Ondik, J.—Rástočný, K.: Interactive Visualization of Differences Between Software Model Versions. Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), 2019, pp. 264-271, doi: 10.5220/0007345502640271.

[62] Ossher, J.—Sajnani, H.—Lopes, C.: File Cloning in Open Source Java Projects: The Good, the Bad, and the Ugly. 2011 27th IEEE International Conference on Software Maintenance (ICSM), 2011, pp. 283–292, doi: 10.1109/ICSM.2011.6080795.

[63] Pate, J. R.—Tairas, R.—Kraft, N. A.: Clone Evolution: A Systematic Review. Journal of Software: Evolution and Process, Vol. 25, 2013, No. 3, pp. 261–283, doi: 10.1002/smr.579.

[64] Qian, W.—Peng, X.—Xing, Z.—Jarzabek, S.—Zhao, W.: Mining Logical Clones in Software: Revealing High-Level Business and Programming Rules. 2013

29th IEEE International Conference on Software Maintenance, 2013, pp. 40–49, doi: 10.1109/ICSM.2013.15.

[65] Rahman, F.—Bird, C.—Devanbu, P.: Clones: What is that Smell? 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), 2010, pp. 72–81, doi: 10.1109/MSR.2010.5463343.

[66] Rahman, F.—Bird, C.—Devanbu, P.: Clones: What is that Smell? Empirical Software Engineering, Vol. 17, 2012, No. 4-5, pp. 503–530, doi: 10.1007/s10664-011-9195-3.

[67] Rattan, D.—Bhatia, R.—Singh, M.: Software Clone Detection: A Systematic Review. Information and Software Technology, Vol. 55, 2013, No. 7, pp. 1165–1199, doi: 10.1016/j.infsof.2013.01.008.

[68] Roy, C. K.—Cordy, J. R.: A Survey on Software Clone Detection Research. Technical Report No. 2007-541, School of Computing, Queen's University at Kingston, Ontario, Canada, 2007, pp. 64–68.

[69] Roy, C. K.—Cordy, J. R.: NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. 2008 16th IEEE International Conference on Program Comprehension, 2008, pp. 172–181, doi: 10.1109/ICPC.2008.41.

[70] Roy, C. K.—Cordy, J. R.—Koschke, R.: Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. Science of Computer Programming, Vol. 74, 2009, No. 7, pp. 470–495, doi: 10.1016/j.scico.2009.02.007.

[71] Saha, R. K.—Asaduzzaman, M.—Zibran, M. F.—Roy, C. K.—Schneider, K. A.: Evaluating Code Clone Genealogies at Release Level: An Empirical Study. 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM), 2010, pp. 87–96, doi: 10.1109/SCAM.2010.32.

[72] Saha, R. K.—Roy, C. K.—Schneider, K. A.—Perry, D. E.: Understanding the Evolution of Type-3 Clones: An Exploratory Study. 2013 10th IEEE Working Conference on Mining Software Repositories (MSR), 2013, pp. 139–148, doi: 10.1109/MSR.2013.6624021.

[73] Saini, V.—Farmahinifarahani, F.—Lu, Y.—Baldi, P.—Lopes, C. V.: Oreo: Detection of Clones in the Twilight Zone. Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018), 2018, pp. 354–365, doi: 10.1145/3236024.3236026.

[74] Sajnani, H.—Saini, V.—Svajlenko, J.—Roy, C. K.—Lopes, C. V.: SourcererCC: Scaling Code Clone Detection to Big-Code. Proceedings of the 38th International Conference on Software Engineering (ICSE '16), 2016, pp. 1157–1168, doi: 10.1145/2884781.2884877.

[75] Schwarz, N.—Lungu, M.—Robbes, R.: On How Often Code is Cloned Across Repositories. Proceedings of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 1289–1292, doi: 10.1109/ICSE.2012.6227097.

[76] Sheneamer, A.—Kalita, J.: A Survey of Software Clone Detection Techniques. International Journal of Computer Applications, Vol. 137, 2016, No. 10, pp. 1–21, doi: 10.5120/IJCA2016908896.

[77] Sokol, F. Z.—Aniche, M. F.—Gerosa, M. A.: MetricMiner: Supporting Researchers in Mining Software Repositories. 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2013, pp. 142–146, doi: 10.1109/SCAM.2013.6648195.

[78] Stevens, R.—De Roover, C.: Querying the History of Software Projects Using QWALKEKO. 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2014, pp. 585–588, doi: 10.1109/ICSME.2014.101.

[79] Tairas, R.—Gray, J.: Phoenix-Based Clone Detection Using Suffix Trees. Proceedings of the 44th Annual Southeast Regional Conference (ACM-SE 44), 2006, pp. 679–684, doi: 10.1145/1185448.1185597.

[80] Taylor, I. J.—Deelman, E.—Gannon, D. B.—Shields, M. (Eds.): Workflows for e-Science: Scientific Workflows for Grids. Springer Publishing Company, Incorporated, 2014, doi: 10.1007/978-1-84628-757-2.

[81] Vincur, J.—Navrat, P.—Polasek, I.: VR City: Software Analysis in Virtual Reality Environment. 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2017, pp. 509–516, doi: 10.1109/QRS-C.2017.88.

[82] Vislavski, T.—Rakić, G.—Cardozo, N.—Budimac, Z.: LICCA: A Tool for Cross-Language Clone Detection. 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2018, pp. 512–516, doi: 10.1109/SANER.2018.8330250.

[83] Wahler, V.—Seipel, D.—Wolff, J.—Fischer, G.: Clone Detection in Source Code by Frequent Itemset Techniques. Fourth IEEE International Workshop on Source Code Analysis and Manipulation, 2004, pp. 128–135, doi: 10.1109/SCAM.2004.6.

[84] Walker, A.—Cerny, T.—Song, E.: Open-Source Tools and Benchmarks for Code-Clone Detection: Past, Present, and Future Trends. ACM SIGAPP Applied Computing Review, Vol. 19, 2019, No. 4, pp. 28–39, doi: 10.1145/3381307.3381310.

[85] Wang, P.—Svajlenko, J.—Wu, Y.—Xu, Y.—Roy, C. K.: CCAligner: A Token Based Large-Gap Clone Detector. Proceedings of the 40th International Conference on Software Engineering (ICSE '18), 2018, pp. 1066–1077, doi: 10.1145/3180155.3180179.

[86] Xie, S.—Khomh, F.—Zou, Y.: An Empirical Study of the Fault-Proneness of Clone Mutation and Clone Migration. Proceedings of the Tenth Working Conference on Mining Software Repositories (MSR), 2013, pp. 149–158, doi: 10.1109/MSR.2013.6624022.

[87] Xue, Y.—Xing, Z.—Jarzabek, S.: CloneDiff: Semantic Differencing of Clones. Proceedings of the 5th International Workshop on Software Clones (IWSC '11), 2011, pp. 83–84, doi: 10.1145/1985404.1985428.

[88] Xuyang, Y.—Chiba, S.: Attempts on Applying Graph Neural Network to Cross-Language Code-Clone Detection. Graduate School of Information Science and Technology, University of Tokyo, 2020. `http://jssst.or.jp/files/user/taikai/2020/FOSE/fose1-3.pdf`.

[89] Yang, W.: Identifying Syntactic Differences Between Two Programs. Software: Practice and Experience, Vol. 21, 1991, No. 7, pp. 739–755, doi: 10.1002/spe.4380210706.

Chaman Wijesiriwardana received his B.Sc. (Hons) special degree in computer science from the University of Peradeniya, Sri Lanka and obtained his M.Sc. in information and communications technology from the Asian Institute of Technology, Thailand and his Ph.D. degree in software engineering from the University of Colombo School of Computing. He worked as Research Assistant in the Software Evolution and Architecture Lab at the University of Zurich for 3 years. His research interests include software evolution analysis, mining software repositories and software security.

Prasad Wimalaratne obtained his B.Sc. special degree in computer science from the University of Colombo and his Ph.D. in virtual environments from the University of Salford, United Kingdom, in 2002. He is Senior Member of IEEE and Member of Computer Society of Sri Lanka (CSSL). He has won several awards including the Presidential Award, University of Colombo Vice Chancellor's Award for Research Excellence, University of Colombo Senate (Open) Awards and CSSL's ICT Researcher of the Year award for research excellence. His research interests include interactive 3D interfaces, unmanned aerial vehicles (UAVs), virtual environments, assistive technology and code analysis. He joined the academic staff of the University of Colombo in 1995 and is currently the Head of the Department of Communication and Media Technologies at the University of Colombo School of Computing.