

Identifying key players in a network of child exploitation websites using Principal Component Analysis

Fateme Movahedi, Richard Frank

Abstract

One of the main objectives of this study is to help prioritize targets for law enforcement by analyzing online websites hosting child exploitation material and finding key players within. Key players are defined as websites that display a combination of high connectivity and a lot of hardcore material and would provide the most disruption in a network if they were to be removed. In this study, various strategies based on Principal Component Analysis are presented to identify those nodes that act as the key players in an online child exploitation network. For evaluating the results of these strategies, we consider the results of various attack strategies. The measures for evaluation are the density, clustering coefficient, average path length, diameter, and the number of connected components in the resulting network. The results show that the strategies proposed are more successful at reducing all of the outcome measures than existing strategies.

Keywords: Social network analysis, Child exploitation, Network disruption, Attack strategy, Web content.

MSC 2020: 91D30, 94C15, 68R05.

1 Introduction

The Internet has provided the social, individual, and technological conditions needed for child exploitation to flourish online. In 2009, the United Nations estimated that there were over four million websites

containing such content [1] while the UK alone processed 105,047 web-pages containing Child Exploitation Material (CEM) in 2018 alone, more than three times the number from 2014 [2]. This is focusing only on websites and does not include other forms of internet-based media-exchange, such as chatrooms, newsgroups, or peer-to-peer (P2P) networks [3] (in particular, it was found that on peer-to-peer networks 0.25% of all queries were pedophilic [4]).

As this is an international problem, government agencies such as INTERPOL have created the International Child Sexual Exploitation image database to track and combat this problem. Similar technologies have also come out from private organizations. Google, working with the National Center for Missing and Exploited Children (NCMEC), has adapted its copyright-centric pattern recognition program used on YouTube, to detect child pornography [5]. Microsoft, also working with NCMEC, has created an algorithm called PhotoDNA for detecting modified versions of images, which they have made available for free to law enforcement who deal with child exploitation online and offline [6].

Research has also focused onto the problem of combating child exploitation material (CEM). In one such study, P2P-based child exploitation was examined for the purpose of developing a filter that can be used to detect queries as pedophilic [7]. In another study, the eDonkey P2P network was studied for the purpose of profile construction based on users' search terms, after which users were classified into those who prefer pedophilia (prepubescent, generally under age 11) vs. those who prefer hebephilia (pubescent, generally age 11 to 14) content [8]. Websites have also been studied through the retrieval and mapping of large networks of websites that host CEM for the purpose of determining structure [9], key players within the networks [10], and the disruption of those networks through the removal of certain nodes [11].

P2P tends to share files directly and can be queried with keywords and hash values [7] resulting in CEM that is relatively easy to find while web-based repositories require digging and exploration to find such content. Thus, mapping P2P can actually be easier, while analyzing websites requires exhaustive mapping of both the size and content on it. This paper focuses on website-based content.

Although a lot of money and time has been invested into various

forms of combating online CEM, the problem is nowhere near under control, and there is a dearth of statistics on how this expenditure translates into victims being rescued and offenders getting prosecuted. This is not a comment against law enforcement but rather speaks to the extent of the problem. With so many websites containing child sexual abuse images and videos, and the limited resources available to various organizations to combat the problem, there needs to be continued efforts to automate and simplify the process of selecting and prioritizing targets for the purpose of the criminal investigation. While the cessation of online child exploitation and the distribution of CEM are unlikely, to prioritize, investigations need to take into account the severity and exposure of the content rather than simply their presence.

One of the studied issues in analyzing the networks is the removal of some important nodes or the hubs, the nodes with the highest connectivity, which separate a complex network into some disconnected components [12-15].

Finding an optimal strategy for disrupting online networks that deal with CEM depends on the specific goals but is a major task regardless. A good attack strategy will cause the largest disruption of the network by selecting the most important nodes termed key players [11]. In the context of CEM, key players could be measured by a combination of factors, such as the site's influence (possibly measured by the number of other sites linking to them), whether the site is a hub (contains a lot of links to other sites), or the amount of content on it (measured in terms of the number of images or videos it contains). Targeting and removing the sites that score high along these factors would allow law enforcement to make the best use of their limited resources. In that sense, key players represent nodes that are among the goals when it comes to the disruption of an online network [11].

In identifying appropriate attack strategies, it is important to consider the topology of the networks. Online networks have two important structural features: Power-law distribution and Small-world properties [10]. The complexity of online networks resides in the small average path lengths among any two nodes (i.e., the small-world property), along with a large degree of local clustering (i.e., the power law distribution). In other words, some special nodes of the structure de-

velop a larger probability to establish connections pointing to other nodes. This introduces problems when finding a node to remove in these networks. Scale-free networks are dominated by a relatively few, highly connected nodes, with the vast majority of nodes being poorly connected [16]. The simplest attack strategy one can consider consists of random nodes from the network. However, scale-free networks can be regarded as a bounding case of heterogeneous networks, thus, for efficiently attacking a heterogeneous network using a random attack a large number of nodes need to be selected for removal. Therefore, scale-free networks are extremely resistant to disruption by random deletion of nodes, and targeted attacks must be used on these networks to effectively disrupt them [17].

Methods to disrupt online networks have been proposed in the past. For example, hub attacks, bridge attacks, fragmentation attacks, and random attacks have been applied to a large online network of websites hosting child exploitation content [10]. The removal of websites identified by these attack strategies followed a greedy sequential process which i) considered a measure m , ii) identified the website that scored highest according to m , iii) removed the identified website from the network, then iv) re-analyzed the network to identify the next top website according to m . This process was repeated until the top websites were eliminated. The impact of the attack strategies was assessed on four measures of disruption: density, clustering coefficient, average path distance, and distance-based cohesion [10].

Identifying the best attack strategy can help improve the efficiency of law enforcement resources when it comes to combating online child exploitation. This paper proposes a new Principal Component Analysis (PCA)-based method to improve the process by which nodes are identified based on their importance and influence within an online network of child exploitation websites. PCA is a useful statistical technique with a common technique for finding patterns in data of high dimensions. It has multiple applications in chemistry, biology, epidemiology, finance, Medical [18]. For example, in [19], the PCA method is investigated to find the most relevant topological and disease parameters in an epidemiologic model. One of the important problems in the analysis of complex networks is to find the key nodes in the network. In [20], an

approach based on non-linear principal component analysis is proposed to identify the top important nodes. Zhang et al. proposed a statistical method based on the PCA algorithm to evaluate the importance of a node in complex networks [21]. Some studies that propose algorithms based on the PCA method for identifying the important nodes in complex networks in the different fields can be found in [21-25].

In this paper, first, network data is collected from the Internet using a custom-written web-crawler (Section 2.1), after which PCA (Section 2.2) is used to identify key players (Section 2.3), which are then removed (Section 2.4). Results indicate that the proposed PCA method outperforms existing network-attack strategies (Section 3) which would have important implications for law enforcement (Section 4).

2 Methods

To introduce the novel PCA method to disrupt child exploitation networks, first, a sub-network centered on online child exploitation material is extracted from the Web using a custom-written web-crawler called CENE (Section 2.1). Using an adjacency matrix and a Laplacian matrix representation of the network, Principal Component Analysis (Section 2.2) is used to identify key players within the network (Section 2.3). Finally, the proposed method is used to formulate an attack strategy (Section 2.4).

2.1 Web-Crawler

Information on the scale and scope of online child exploitation material can be discovered by studying the websites that contain this type of content. One strategy for doing this is to manually visit the websites under study, read the webpages, establish the content of each webpage, then search for links leading to other webpages, and finally map out the hyper-links between those pages. Analyzing the website manually might lead to accurate conclusions about the content, but studying a large website with thousands of pages in this fashion is infeasible. Similarly infeasible is the manual creation of a map of the inter-linkages between the pages for the purposes of social network analysis. Due to

the large scale of the problem, the data collection and analysis must be performed by computers.

Web-crawlers are the tools used by all search engines to automatically navigate the Internet and collect information about each website and webpage. They, given a starting webpage, will recursively follow the links out of that webpage until some user-specified termination conditions apply. During this process, the web-crawler will keep track of all the links between other websites and (optionally) eventually follow them and retrieve those as well. There is much standalone web-crawler software available on the Internet, such as Win Web Crawler, WebSPHINX, or Black Widow, and some could be used to capture the content of a website onto the machine of the investigator for evidentiary purposes.

However, there are several problems with off-the-shelf web-crawlers. First, most such web-crawlers will save all the content onto the hard-drive, which might work for law-enforcement, who are allowed to do this, but for researchers studying this problem, it is against Canadian law to store such content on a hard-drive even temporarily. All analysis must be done in-memory. Second, when looking for targeted content, off-the-shelf web-crawlers do not perform an adequate job as the search process must be guided by conditions, such as the presence of a child exploitation image on a website, or multiple child exploitation-specific keywords within the body of the webpage. The presence of a child exploitation image can be detected using hash values such as MD5, SHA1, or PhotoDNA, which translate an image into a series of numbers (a hash-value) that are then be compared to a database of previously identified child exploitation images.

To bypass these problems, a custom-written crawler, called Child Exploitation Network Extractor (CENE) was used to collect information on these online child exploitation networks. As CENE visits each page, it captures the contents of the webpage for later analysis, while simultaneously collecting information about the webpage and making decisions on whether it contains child exploitation material, or not. All processing such as hash value and keyword frequency calculations are done in-memory, with the resulting information stored in a central database. For each network extracted, features are collected about

the contents of each webpage and the links between them. This information is stored at the webpage level and then aggregated up to the website level. For example, all pages on `www.website.com` are visited, analyzed, and statistics are calculated for each page. After this process is done for all webpages of interest, then all statistics are aggregated up to a single set of statistics for the website `www.website.com` itself [9].

The data collection was started with seed points collected through popular search engines using previously identified CEM-specific keywords. The URLs presented as the search results were then inputted into CENE, for it to recursively follow links out of the seeds and scan the entire website. These websites were not visited by humans before being analyzed by CENE, thus the size and structure of the webpage were not a factor in whether that website was used as a seed, or not. For each page, CENE decided whether the page should be considered as CEM, and if so, all links on the page were added to a queue for later retrieval. The criterion for this decision was set at 1) the presence of one known child exploitation hash value (i.e., an image already encountered by law-enforcement and known to be an illegal child exploitation image) and/or 2) the presence of seven child exploitation keywords (specified below).

The web-crawler continues to examine websites until it does not find any more such sites, meaning, none of the sites analyzed link to any new sites containing CEM. A criterion used in the crawling process was the exclusion of websites known to not contain child exploitation content. These websites were based on a list of the most popular websites (e.g., Google) and a list of websites collected during previous data collections that were verified to not contain child exploitation content. The resulting network contains information about the number of images (overall and known child exploitation), videos, keywords, and linkages.

2.2 Principal Component Analysis

The main objective of this study is the disruption of an online child exploitation network by removing appropriate websites from the network. To do this, two methods are introduced to select some nodes from the network based on Principal Component Analysis (PCA). PCA is a

common statistical technique for finding information in data with high dimensions [26]. Advanced topics and technological methods in PCA are given in the book by Jolliffe [27]. PCA compresses data without much loss of information [26, Chapter 2], and mathematically defines a new coordinate system by determining the eigenvectors and eigenvalues of a matrix that optimally describe the variance in a single dataset. Correlation between variables in the dataset corresponds to the degree of variance such that the greatest variance by any projection of the data comes to lie on the first coordinate which is called the first principal component [26, Chapter 3]. PCA involves a calculation of a covariance matrix of a dataset to minimize the redundancy and maximize the variance [26, Chapter 1] in the process of finding a linear mapping of a dataset to a dataset of lower dimensionality.

In computational terms, the principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. This process is equivalent to finding the axis system in which the covariance matrix is diagonal. Matlab software is used for computing of PCA algorithm.

In this study, the input data set of PCA is an $n \times n$ matrix M , where n is the number of network nodes and M represents the structure of the network. Then the mean of the data is subtracted from each data value in order to obtain a data set with zero means. The resulting matrix is named $M - \bar{\mu}$. The covariance matrix of $M - \bar{\mu}$ is then calculated, and is given by

$$C^{n \times n} = (c_{ij}, c_{ij} = cov(X_i, X_j)),$$

where $C^{n \times n}$ is a matrix with n rows and n columns, X_i is the i^{th} dimension, and $cov(X_i, X_j) = \frac{\sum_{k=1}^n (X_{ik} - \bar{X}_i)(X_{kj} - \bar{X}_j)}{n-1}$, that is the covariance between i^{th} and j^{th} dimensions.

The covariance is a measure to find out how much the dimensions differ from the mean with respect to each other. Then the eigenvalues and eigenvectors of the covariance matrix are calculated. These eigenvalues and eigenvectors show the useful and important information of the data. Noticeably, the eigenvectors are perpendicular to each other. It should be noted that eigenvalues have quite different values. In general, the eigenvectors from the covariance matrix are ordered by

eigenvalue, highest to lowest. This gives the components in order of significance. At this time, one could decide to ignore the components of lesser significance. By taking the eigenvectors of the covariance matrix, one can extract vectors that characterize the data. The final step in PCA is to choose the first principal component (eigenvector) and to form a feature vector.

The eigenvector with the largest eigenvalue is the direction of greatest variation with the maximal variance of the data set [26], and the second largest eigenvalue is the (orthogonal) direction with the next highest variation, and so on [26]. So, in this study, only the eigenvector corresponding to the highest eigenvalue is considered a feature vector. Let α_1 be this eigenvector with the norm of 1. Define $B_{1 \times n} = \alpha_1^T \times (M - \mu)^T$. The vector B approximately consists of all information pertaining to principal data [26].

2.3 PCA Metrics for Identifying Key Players

In mathematical modeling of complex networks, the representation of the networks can be denoted by the adjacency matrix and Laplacian matrix of the network. For analyzing and studying the networks, it is sufficient to compute some measures in these networks. According to the definition of the adjacency matrix and its properties, some measures are usually used for analyzing the network such as Clustering coefficient, Average path length, and Degree distribution. And in the network modeling with the Laplacian matrix, the measures such as centrality and connected components are used.

We introduce two strategies using PCA, named PCA_A and PCA_L , to identify the most important key players in an online exploitation network. For this purpose, the adjacency matrix and the Laplacian matrix of a network are considered input data sets to the PCA algorithm.

The adjacency matrix, named A , of a network G consisting of n nodes is the $n \times n$ matrix, where $A(i, j) = 0$ if nodes i and j are not connected, otherwise $A(i, j) = 1$. The adjacency matrix is represented to show nodes and connectivity between them.

The Laplacian matrix of a directed network $L = (l_{ij})_{n \times n}$ is defined

as $L = D - A$, where $D = \text{diag}(d_1, d_2, \dots, d_n)$ is the matrix, the diagonal matrix which in d_i denotes the out-degrees or in-degrees of the nodes in the network [27]. From the definition it follows that

$$l_{ij} = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The Laplacian matrix represents the nodes and connectivity between the nodes of the network and shows the network structure. It should be noted at once that loops have no influence on L [28, 29]. The Laplacian matrix of a graph and its eigenvalues can be used in several areas of mathematical research and have a physical interpretation in various physical and chemical theories. The Laplacian spectrum is more important than the adjacency matrix spectrum [28].

Two methods are proposed above to distinguish the key players in a network. In the PCA_A strategy, the adjacency matrix (A) of the network is considered as the input into the PCA algorithm. In the PCA_L strategy, the Laplacian matrix of the network is considered as M . Each column of M yields the information relevant to the connectivity of each website in the network. So, the one node of the network corresponding to the maximum entry of B can be considered a key player.

To clarify the issue, we provide a simple example. We consider a sub-network with 10 nodes from the initial extracted network by CENE (Figure 1(a)). Assume the nodes of the network are labeled with v_i , with $1 \leq i \leq 10$, and consider vector $V = (v_1, v_2, \dots, v_{10})$ as the nodes vector. The adjacency and Laplacian matrices of this network are denoted by A and L , where the Laplacian matrix is calculated by equation (1) (see Figure 1 (b and c), respectively).

For the PCA_A strategy (see Section 2.2), the matrix $(A - \mu)$ is calculated, in which μ is the mean of the data in each column. Let α_1 be the eigenvector associated with the largest eigenvalue from the covariance matrix of $(A - \mu)$. So, the vector $B_{1 \times 10}$, the output of the PCA algorithm, is as follows

$$B = [0.382, 0.383, -1.1601, 0.1559, 0.381, -0.531, \\ 0.156, 0.382, 0.382, -0.521].$$

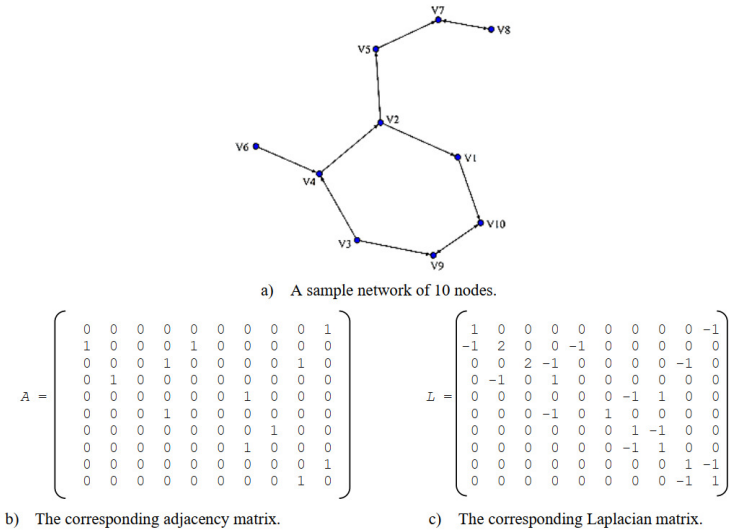


Figure 1. A sample network and its corresponding adjacency matrix, and Laplacian matrix

The maximum entry of this vector is 0.383 which corresponds to the 2nd entry of the nodes vector. So, we select the node v_2 as a key player. According to matrix A , it is also worth mentioning that node v_2 has the maximum out-degrees in this network (based on the hub strategy). It means that this website may provide abundant access to materials in the network. So, it is an appropriate node in the network that removing it can help for disrupting this network. For the PCA_L strategy, the Laplacian matrix (L) of the network is considered as the input data set M into PCA, resulting in vector $B_{1 \times 10}$

$$B = [-0.736, 1.892, 0.949, -1.116, -0.617, 0.192, 0.043, -0.326, -0.645, 0.363].$$

The maximum entry of vector B is 1.892 which corresponds to the 2nd entry of the nodes vector. So, the PCA Algorithm selects the node v_2 from the network.

2.4 Key Player Removal

Key players can be defined as nodes with large volumes of content. However, we focus on the phase before this, when the offender is seeking out this content and is exploring the network of CEM websites, hopping from one website to another seeking new material. As a result, for the purposes of this paper, we define a key player as a website that is important not in terms of content, but in terms of network position. Thus, we consider some measures for the evaluation of the network. Since both the adjacency and Laplacian matrices are used to represent some properties of networks, thus, we consider them for investigating the resulting network in any steps.

In each iteration, one node is identified by PCA as being the most important in the network. The attack scenario presented in this paper is greedy, a node is removed from the network, after which the remaining network is reanalyzed to identify the next best node. This process is repeated until the network is disrupted by removing key players using the PCA approach. Since our aim is to disrupt the initial network while removing the minimum number of nodes, our algorithm selects one node at any step. Finally, the resulting network is examined, and the obtained results are compared to the outcome measures of other attack strategies.

For comparing the results of the presented strategies to other strategies, we consider various attack strategies. These attack strategies involve hub attacks (using the measure of degree centrality), bridge attacks (using the measure of Betweenness), network capital [11] (where the node that contributes the most content is selected), and random attacks (where each node has an equal chance of being targeted).

In networks, centrality is the measure of how important a node is in the network. Betweenness centrality is a measure of centrality in the network. In the bridge attack method, an important node based on Betweenness centrality is selected. Therefore, we use the PCA_L method to compare the proposed approach based on the PCA method by considering the Laplacian matrix as the input of the algorithm to study strategies, especially the bridge attack.

Continuing the example from Section 2.3, after removing the node

v_2 from the network, PCA_A identified node v_9 as the next key player. PCA_L identified nodes $\{v_2, v_3\}$. We consider other attack strategies for this network to identify two nodes in each step. For this purpose, these strategies include Hub Attack, Bridge Attack, and Random Attack. The original and resulting networks using different attack strategies are shown in Figure 2. It can be seen from Figure 2(b) and Figure 2(f) that the network became fragmented into three separate components following both the PCA_A and PCA_L attack strategies. So, in each of the components, it would be harder to discover the other small connected components, as no link leads from one connected component to another. While the graphs of Hub Attack and Bridge Attack have one component with two isolated nodes (see Figures 2(b) and c)). The set of selected nodes by Hub Attack and Bridge Attack from initial network are $\{v_4, v_7\}$ and $\{v_2, v_7\}$, respectively. Also, as seen in Figure 2(d) the network has only one component after removing two nodes v_8 and v_9 by Random Attack.

The impact of removing these websites by the proposed strategy is then examined on several outcome measures: density, clustering coefficient, average path length, diameter, and the number of connected components in the resulted network. Density is calculated by dividing the number of existing links in the network by the maximal number of links [29]. So, the changes in density correspond to the changes in the number of links. The other outcome measure included is the network clustering coefficient which is the average density of the neighborhoods of the websites in a network [29]. In other words, it is defined as the probability that two randomly selected neighbors are connected to each other. The average path length, defined as the average number of links along the shortest paths between two nodes [30], is examined for all pairs of nodes in the network.

In this paper, we use Matlab for computing and coding the attack simulation. For obtaining the average path length of the resulting networks, we compute the shortest distance between pairs of nodes in the network using Matlab's *graphallshortestpaths(G)* function, based on Johnson's algorithm [31]. In this study, for two nodes v_i and v_j that have no other connections, we considered the shortest distance $d(v_i, v_j) = 0$. So, if the size of the network is n , then the average path

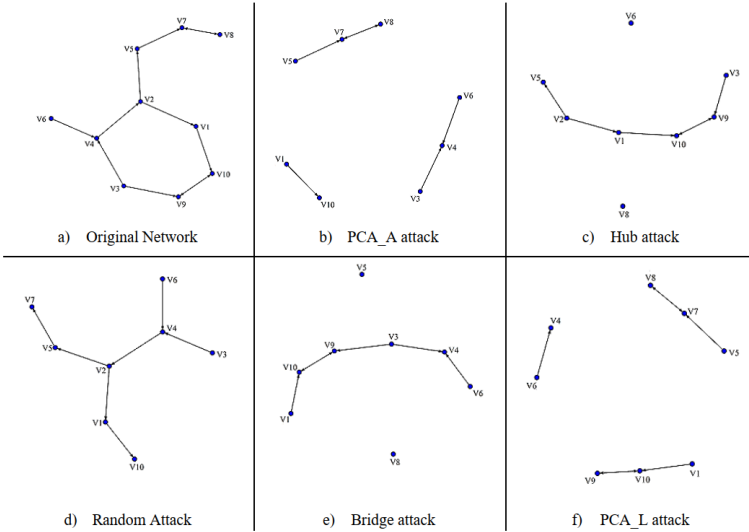


Figure 2. Resulting network after various attacks

length (avg) of the network is obtained as follows

$$avg = \frac{\sum_{i \neq j} d(v_i, v_j)}{n(n-1)}.$$

The distance between the two most distant vertices in a network is called the diameter of the network. The impact on network connectivity of selecting and removing targeted nodes as measured by changes to network diameter is evaluated in [32].

For instance, if the nodes, which are most highly connected, are removed through the Hub attack, then the network diameter increases rapidly, and the resulting network fragments into smaller components or subgraphs [32].

3 Results and Discussion

The main goal of this study is to determine a method to find the websites that should be prioritized by law enforcement agencies involved

in combating child exploitation. In this study, an online network is extracted using CENE, a web-crawler tailored to follow the links out of and into child exploitation websites when given a specific set of starting websites [11]. CENE started to explore a network from a single child exploitation website that was found through extensive searches on Google. CENE would not follow links out of a single webpage if that webpage did not meet certain criteria associated with child exploitation (at least one known image, or at least seven pre-identified keywords related to child exploitation). These were words that were provided by the RCMP as well as those used in previous research [3]. These words are included *qwerty*, *qqaazz*, *ptsc* and *pthc*. This category of keywords consisted of twenty-seven words, which, to the best knowledge of the authors, were still valid during data collection.

With these starting parameters, CENE identified a network of connected websites that contained at least one page matching the requirements and stopped expanding the network when none of the webpages linked to child exploitation material anymore (i.e., the links led to webpages off topic). This resulting network contained 177 nodes and 915 edges, where nodes are defined to be entire websites (web-domains), and the directed edges represent the links pointing from one node (website) to another. A node in the network is labeled with i , where $1 \leq i \leq 177$. The initial network density, clustering coefficient, average path length and network diameter are 0.0294, 0.5095, 0.0030, and 6, respectively.

First, matrix A (the adjacency matrix) and matrix L (the Laplacian matrix) are obtained from the network. The PCA algorithm selects a node from the network using two matrices A and L in each step. After removing the selected node, the algorithm calculates outcome measures of the resulting network and updates the new network to identify the next appropriate node. This process was repeated until the network breaks up into components for both the strategies.

In the PCA_A strategy, the adjacency matrix A , which in this context contains the number of links one website has pointing to another, was the input into the PCA algorithm. Results (see Table 1) show the following changes after removing five selected nodes such that the first disruption happens. Density fell by 26.5% from 0.0294 to 0.0216, while the number of links dropped from 915 to 687 indicating that by re-

moving only five (2.8%) nodes, 228 (24.9%) of the links were removed. Although this was expected, as this attack focuses on the removal of the nodes with the most links, the amount of damage caused was more severe than expected. The average path length decreased by 26.7% from 0.0030 to 0.0022, meaning that with removing the links between some websites, there is no path between some nodes in the network. Therefore, according to the definition of *avg*, this value is reduced. Finally, the clustering coefficient in this network decreased from 0.5095 to 0.4520 (-11%), each node was now less embedded in the network than before. Thus, through the removal of just five nodes, we were able to break 25% of the active links between them, resulting in a much smaller and partially disconnected graph (see Figure 3(a) vs. 3(b)) making it much more difficult for individuals to reach other websites.

Table 1. The results of outcome measures of obtained network after removing one node by PCA_A strategy in each step

| Steps of running PCA | Density of network | Clustering coefficient | Average path length |
|----------------------|--------------------|------------------------|---------------------|
| Step 0 | 0.0294 | 0.5095 | 0.0030 |
| Step 1 | 0.0276 | 0.4990 | 0.0023 |
| Step 2 | 0.0260 | 0.2817 | 0.0023 |
| Step 3 | 0.0244 | 0.4612 | 0.0023 |
| Step 4 | 0.0229 | 0.4608 | 0.0023 |
| Step 5 | 0.0216(↓ 26.5%) | 0.4520(↓ 9%) | 0.0022(↓ 26.7%) |

As a second experiment, with the PCA_L strategy, the Laplacian matrix L is given as the input into the PCA algorithm. Since websites with many in-degree ties may be considered more important, a website can easily link to others, but it may not be relevant or interesting enough to receive links from other websites, thus in-degree for any of the nodes in the Laplacian matrix are considered. After selecting and removing five nodes, the following changes to the network structure were observed. First, density fell by 25.8% from 0.0294 to 0.0216, indicating a similar amount of damage to the network as PCA_A . The clustering coefficient decreased from 0.5095 to 0.4588 (-10%), and each node was now less embedded in the network than before, even when compared to PCA_A . Finally, the average path length fell by 16.8% from

0.0030 to 0.0025 (see Table 2). Overall, PCA_L seems to have severed almost the same number of links (228 in PCA_A vs. 236 in PCA_L), but it did so in such a way that the average path length dropped by only 16.8% (as opposed to 26.7% in PCA_A) meaning that, in the context of a network of child exploitation websites, the average user moving from one website to another would need to click through more websites to reach the destination site while having fewer links available to them to do so. Thus, PCA_A significantly increased the difficulty of finding websites within the network, even compared to PCA_L .

Table 2. The results of outcome measures of obtained network after removing one node by PCA_L strategy in each step

| Steps of running PCA | Density of network | Clustering coefficient | Average path length |
|----------------------|--------------------|------------------------|---------------------|
| Step 0 | 0.0294 | 0.5095 | 0.0030 |
| Step 1 | 0.0277 | 0.4945 | 0.0030 |
| Step 2 | 0.0261 | 0.4813 | 0.0029 |
| Step 3 | 0.0246 | 0.4749 | 0.0028 |
| Step 4 | 0.0232 | 0.4669 | 0.0026 |
| Step 5 | 0.0218(↓ 25.8%) | 0.4588(↓ 10%) | 0.0025(↓ 16.8%) |

Indeed, the aim of this study is to find the strategy which selects the minimum nodes (websites) and will cause the largest disruption to the network. The removal of websites identified by the proposed attack strategies followed a sequential process, in which one node is identified by the PCA algorithm as being the most important in the network. Then a node is removed from the network, after which the remaining network is reanalyzed to identify the next best node (i.e., a greedy approach). Table 1 and Table 2 show the changes in the resulting network after removing the node, where the PCA algorithm is selected in each step (n). Also, n is the number of selected nodes by PCA and it is the number of removed nodes in the resulting network. According to these tables, for $n = 0, 1, 2, 3, 4$, the variations were reduced in the network after each step, but the removal of the nodes with the most links occurs in step $n = 5$ disruptions of this network. According to Figure 3, the active links between these nodes (websites) were removed and resulting in a smaller disconnected graph.

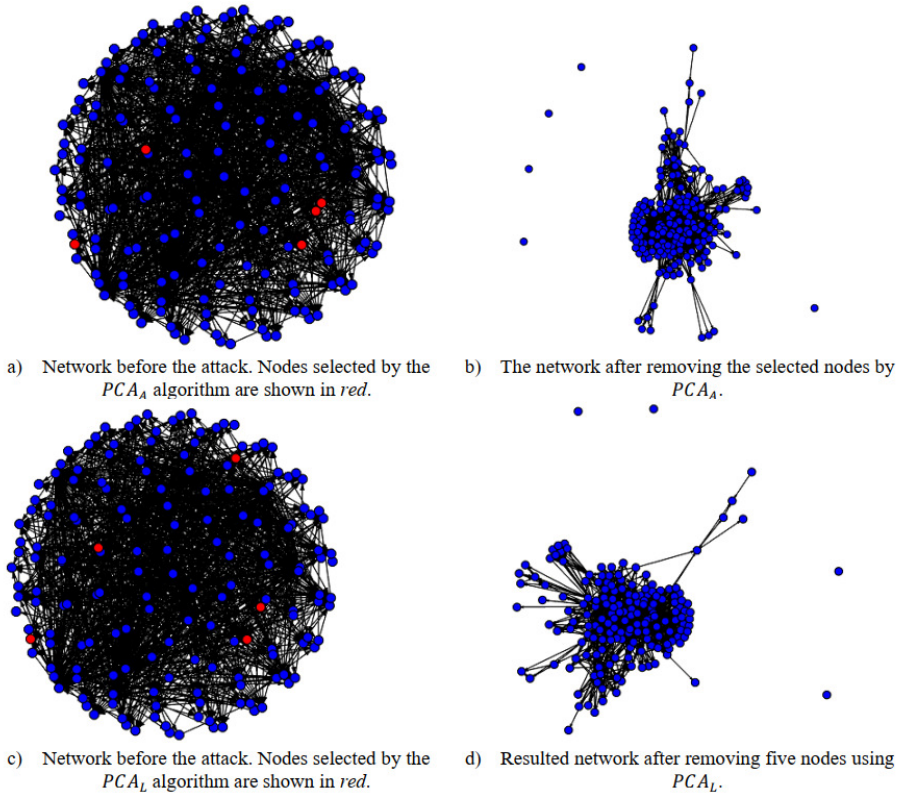


Figure 3. Network before and after attack by the PCA_A and PCA_L strategies

Table 3 shows the results of calculating the outcome measures after removing five nodes using any of the other existing strategies (Hub or Bridge attacks, for example). Results show that for different outcome measures, some attack strategies are less effective in disrupting the network. Furthermore, the effectiveness of these various attacks varied with the different goals, such as reducing density, clustering, and reachability. For example, if the goal is to delete as many links in a network as possible (i.e., reduce density), then the Attack Bridge is the most effective strategy, since it led to a reduction of 18% in the density of our test network. However, using the strategy proposed in

this paper the density decreased by 26.5% and 25.8% using PCA_A and PCA_L , respectively. It means that the proposed strategies are even more effective in decreasing the number of links. To reduce a nodes embeddedness in a tight-knit component of the network (clustering), the results (Table 3) show that the current attack strategies, except hub attack, increase the value of this measure, while PCA_A and PCA_L decrease it by 11% and 10%, respectively. Also, this measure is decreased in the hub attack in a similar amount to PCA strategies.

Table 3. The results of outcome measures of obtained network after removing five nodes by various attack strategies

| Steps of running PCA | Density of network | Clustering coefficient | Average path length | Diameter | Changes of the number of CC |
|-------------------------------|--------------------|------------------------|---------------------|--------------|-----------------------------|
| Hub Attack | 0.0285(↓ 3%) | 0.4520(↓ 11%) | 0.0021(↓ 30%) | 7(↑ 16%) | 2 ↑ |
| Bridge Attack | 0.0240(↓ 18%) | 0.5347(↑ 4.9%) | 0.0016(↓ 46.6%) | 7(↑ 16%) | 4 ↑ |
| Network Capital method | 0.0305(↑ 3%) | 0.5122(↑ 0.52%) | 0.0032(↑ 6.6%) | 6(No-change) | 4 ↓ |
| Random Attack | 0.0299(↑ 1%) | 0.5158(↑ 1.2%) | 0.0034(↑ 13.3%) | 6(No-change) | 5 ↓ |
| PCA_A | 0.0216(↓ 26.5%) | 0.4520(↓ 11%) | 0.0022(↓ 26.7%) | 7(↑ 16%) | 2 ↑ |
| PCA_L | 0.0218(↓ 25.8%) | 0.4588(↓ 10%) | 0.0025(↓ 16.8%) | 7(↑ 16%) | 2 ↑ |

Results show that the bridge attack decreased the average path length the most, and, thus, was the most successful attack among all of the attack strategies, even better than PCA_A or PCA_L (note, however, that in the other measures the bridge attack was significantly inferior to both PCA_A and PCA_L). On the other hand, this method increases the clustering coefficient measure by 4.9%. So, overall, it can't be the most effective strategy for disrupting this network.

There are some isolated nodes in the network after running the proposed attacks (PCA_A and PCA_L). Since the shortest path length for any isolated node with others is equal to zero in our algorithm, so one can expect to decrease this measure using both PCA strategies. However, the PCA_L method has less reduction than the PCA_A method.

We measure the network diameter and the changes in the number of connected components (CC) to ensure the above discussion's integrity.

As for the network diameter, when the targeted nodes are removed, the diameter of the network increases, and the network breaks into isolated, connected components. This occurs because when deleting these nodes, the heart of the network is disturbed, whereas a random attack is most likely does not. The results show that all of the attack strategies increase the network diameter after removing five targeted nodes by 16%, except for the random and network capital strategies. According to Table 3, after the removal of targeted nodes in any strategy, four connected components are added to the resulting network by Bridge attack. By using PCA methods and Hub attack, two connected components are added to the network.

In this study, decreasing the average path length is significant if the diameter and the changes in connected components increase, because after removing the nodes, the resulting network contains some connected components of small size.

The results show that the number of connected components in the resulting network after two random and network capital attacks decreased. According to the obtained results of other measures for these two strategies, it is clear to see that the reduction of the number of connected components is due to selecting isolated nodes.

Figure 3 shows the before and after the process by which the original network is changed when the websites are removed by PCA_A and PCA_L . Most of the targeted websites are located inside the original network where they have the most influence on the transmission of information to other websites (see the red nodes in Figure 3 (a and c)). In addition, as seen in Figure 3(b), the network is now fragmented into one component with five isolates while density decreased by removing 228 edges from the initial network. Also, in Figure 3(d), the resulting network has one component and four isolated nodes after selecting and removing five nodes by the PCA_L strategy. These five nodes also are located inside the initial network. It is also worth mentioning that there are four common nodes between those selected by PCA_A and PCA_L .

It is clear that one may select more nodes and repeat the proposed algorithms for selecting and analyzing the obtained networks in steps $n > 5$ for this dataset. Table 4 shows the results after removing 20

Table 4. The results of outcome measures of obtained network after removing 20 nodes by various attack strategies

| Steps of running PCA | Density of network | Clustering coefficient | Average path length | Diameter | Changes of the number of CC |
|------------------------|--------------------|------------------------|---------------------|--------------|-----------------------------|
| Hub Attack | 0.0082(↓ 72%) | 0.2456(↓ 51%) | 0.0006(↓ 80%) | 5(↓ 16%) | 8 ↑ |
| Bridge Attack | 0.0144(↓ 51%) | 0.5251(↑ 1%) | 0.0004(↓ 86%) | 7(↑ 16%) | 9 ↑ |
| Network Capital method | 0.0178(↑ 39%) | 0.5123(↑ 0.5%) | 0.0019(↑ 36%) | 6(No-change) | 7 ↓ |
| Random Attack | 0.0319(↑ 8%) | 0.5049(↑ 9%) | 0.0024(↑ 20%) | 6(No-change) | 8 ↓ |
| PCA _A | 0.0153(↓ 47%) | 0.2396(↓ 53%) | 0.0008(↓ 73%) | 7(↑ 16%) | 9 ↑ |
| PCA _L | 0.0080(↓ 73%) | 0.2429(↓ 52%) | 0.0005(↓ 83%) | 7(↑ 16%) | 9 ↑ |

nodes by various attack strategies.

The results show that both strategies (PCA_A and PCA_L) have better performance than the four existing strategies after selecting and removing more nodes. According to Table 4, the PCA_L strategy yielded significant improvements over the existing methods versus others. Although the hub attack did yield results close to both PCA strategies in some measures, one strategy is more successful than existing strategies for disrupting the network that significantly changes all outcomes.

According to the obtained results, we can use the PCA_L strategy for disrupting this dataset (see Table 4). This strategy is based on the Laplacian matrix that represents the network and its properties.

4 Conclusion

This experiment attempted to identify the most important nodes in an online network containing child exploitation images, extracted from the Internet. The network was extracted using CENE, a custom-written web-crawler that can identify known child exploitation images through their hash values and focuses on them for the purposes of network extraction. Agencies that combat this problem have limited resources, thus, for the purposes of time savings and cost reduction, it is important to select the most efficient attack strategies. There are multiple

existing strategies for disrupting online child exploitation networks that are dependent on some properties of networks, such as a hub attack strategy which depends on using the measure of degree centrality. The aim of these studies is to select the strategy that will cause the largest disruption to the networks themselves.

In this paper, six attack strategies were used to attack the structure of a single online network of child exploitation websites. The goal was to determine whether the two proposed Principal Component Analysis techniques presented an advantage over four previously established attack methods (hub, bridge, network capital and random). Considering the adjacency matrix (PCA_A) and Laplacian matrix (PCA_L) in the PCA algorithm, each column of these matrices yields the information on connectivity relevant to each website in the network. The nodes which were selected by PCA correspond to websites yielding significant information to the network. The results show both the two strategies (PCA_A and PCA_L) have better performance than the existing four strategies when it comes to disrupting the network, although the Hub attack yielded the results close to both PCA strategies. Overall, however, results shown after selecting and removing more nodes of the network using various strategies, the PCA_L strategy yielded significant improvements over the existing methods.

Although the data collection aimed at finding as many of these websites as possible, there are two limitations to the data collection, which might impact the generalizability of this study. First, the data collection was started with seed points which were identified through keyword searches through various search engines, and data collection proceeded from one website to another through the linkages available within the websites. While websites containing similar material do tend to link together, any website which was not linked would have been missed using this strategy. This limitation was mitigated by having multiple seed starting points, but regardless, if a website is not linked to, then it was not visited and thus not included in the data collection. Second, this research is focused on open internet networks only, thus no other types of networks were included, such as dark-web sites, file-sharing networks, or private password-protected websites.

Future work should look at the effectiveness of these attacks in

other types of networks, not just different networks extracted from the Internet, but also networks that do not share the Internet's Power-law distribution and Small-world properties. The resiliency of nodes of various importances could also be investigated with the help of law enforcement. If law enforcement were able to incorporate such attack strategies into their selection methodologies, and then actually remove them from the Internet, CENE could be used to monitor the resulting network for a possible redistribution of the content and/or importance. Do more important nodes get more prominent, or do the smaller websites take up the opportunity and become more prominent within the network structure? Or perhaps, like a hydra, many other websites are created to fill the void?

Acknowledgments. Fateme Movahedi would like to thank Dr. Vahid Dabbaghian and Dr. Piper Jackson for their constructive suggestions and helps.

References

- [1] E. Engeler, "UN expert: Child porn on internet increases," The Associated Press, 2009. [Online]. Available: <https://www.nbc-news.com/id/wbna32880508>.
- [2] "Once upon a year," Annual Report, Internet Watch Foundation, 2018. [Online]. Available: <https://www.iwf.org.uk/media/tthh3woi/once-upon-a-year-iwf-annual-report-2018.pdf>.
- [3] M. Latapy, C. Magnien, and R. Fournier, "Quantification of paedophile activity in a large p2p system", Project MAPAP SIP-2006-PP-221003 (<http://antipaedo.lip6.fr>), Technical report, 2009. [Online]. Available: <http://antipedo.lip6.fr/T24/TR/quantification.pdf>
- [4] M. Latapy, C. Magnien, and R. Fournier, "Quantifying paedophile activity in a large p2p system," *Information Processing & Management*, vol. 49, pp. 248–263, 2013.
- [5] M. Shiels, "Google tackles child pornography," BBC News, 2008. [Online]. Available: <http://news.bbc.co.uk/2/hi/7347476.stm>.

- [6] “New technology fights child porn by tracking its PhotoDNA,” Microsoft, 2009. [Online]. Available: <https://news.microsoft.com/2009/12/15/new-technology-fights-child-porn-by-tracking-its-photodna/>.
- [7] M. Latapy, C. Magnien, and R. Fournier, “Report on Automatic Detection of Paedophile Queries,” Measurement and Analysis of P2P Activity Against Paedophile Content project (<http://antipaedo.lip6.fr>), Technical report, 2006. [Online]. Available: <http://antipaedo.lip6.fr/T24/TR/query-detection.pdf>.
- [8] S. Hammond, E. Quayle, J. Kirakowski, E. O’Halloran, and F. Wynne, “An Examination of Problematic Paraphilic use of Peer to Peer Facilities,” in *International Conference on Advances in the Analysis of Online Paedophile Activity*, pp. 65–73, 2009.
- [9] R. Frank, B. G. Westlake, and M. Bouchard, “The structure and content of online child exploitation,” in *Proceedings of the 16th ACM SIGKDD Workshop on Intelligence and Security Informatics (ISI-KDD 2010)*, July 2010, Article No. 3, Pages 1–9. DOI: <https://doi.org/10.1145/1938606.1938609>.
- [10] K. Joffres, M. Bouchard, R. Frank, and B. Westlake, “Strategies to disrupt online child pornography networks,” in *European Intelligence and Security Informatics Conference 2011*, (Athens, Greece), pp. 163–170, 2011. DOI: 10.1109/EISIC.2011.32.
- [11] B. G. Westlake, M. Bouchard, and R. Frank, “Finding the key players in online child exploitation networks,” *Policy and Internet*, vol. 3, no. 2, 2011, Article 6. DOI: 10.2202/1944-2866.1126.
- [12] I. A. Kovacs and A. L. Barabási, “Network science: Destruction perfected,” *Nature*, vol. 524, pp. 38–39, 2015.
- [13] R. Albert, H. Jeong, and A. L. Barabási, “Error and attack tolerance of complex networks,” *Nature*, vol. 406, no. 6794, pp. 378–382, 2000.
- [14] F. Morone, B. Min, L. Bo, R. Mari, and H. A. Makse, “Collective influence algorithm to find influencers via optimal percolation in massively large social media,” *Sci Rep*, Article No. 30062, 2016. DOI: <https://doi.org/10.1038/srep30062>.

- [15] B. Amiri, M. Fathian, and E. Asaadi, "Influence maximization in complex social networks based on community structure," *Journal of Industrial and Systems Engineering*, vol. 13, no. 3, pp. 16–40, 2021.
- [16] A. L. Barabási, "The physics of the Web," *Physics World*, vol. 14, pp. 33–38, 2001.
- [17] R. Pastor-Satorras and A. Vespignani, "Immunization of complex networks," *Physical Review E*, vol. 65, Article ID. 036104, 2002.
- [18] Y. Mori, M. Kuroda, and N. Makino, *Nonlinear Principal Component Analysis and Its Applications*, Springer, 2016.
- [19] P. H. T. Schimit and F.H. Pereira, "Disease spreading in complex networks: A numerical study with Principal Component Analysis," *Expert Systems with Applications*, vol. 97, pp. 41–50, 2018.
- [20] S. Basu and U. Maulik, "Mining important nodes in complex networks using nonlinear PCA," in *2017 IEEE Calcutta Conference (CALCON)*, (Kolkata, India), IEEE, 2017, pp. 469–473. DOI: 10.1109/CALCON.2017.8280778.
- [21] K. Zhang, H. Zhang, Y. dong Wu, and F. Bao, "Evaluating the importance of nodes in complex networks based on principal component analysis and grey relational analysis," in *2011 17th IEEE International Conference on Networks*, (Singapore), IEEE, 2011, pp. 231–235. DOI: 10.1109/ICON.2011.6168480.
- [22] Y. J. Jin, K. Xu, N. Xiong, Y. Liu, and G. Li, "Multi-index evaluation algorithm based on principal component analysis for node importance in complex networks," *Networks, IET*, vol. 1, no. 3, pp. 108–115, 2012.
- [23] P. Wang, J. Lu, and X. Yu, "Identification of Important Nodes in Directed Biological Networks: A Network Motif Approach," *PLOS ONE*, vol. 9, Article ID. e106132, 2014.
- [24] F. Hu and Y. Liu, "Multi-index algorithm of identifying important nodes in complex networks based on linear discriminant analysis," *Modern Physics Letters B*, vol. 29, no. 03, Article No. 1450268, 2015.

- [25] F. Hu, Y. Liu, and J. Jin, “Multi-index Evaluation Algorithm Based on Locally Linear Embedding for the Node Importance in Complex Networks,” *DCABES*, pp. 138–142, 2014.
- [26] L. I. Smith, *A tutorial on Principal Components Analysis*, Maintained by Cornell University, 2002.
- [27] I. T. Jolliffe, *Principal component Analysis*, Springer Science & Business Media, 2013.
- [28] C. Godsil and G. Royle, *Algebraic Graph Theory*. New York: Springer-Verlag, 2001.
- [29] B. Mohar, “The Laplacian spectrum of graphs,” in *Graph Theory, Combinatorics, and Application*, Y. Alavi, G. Chartrand, O. R. Oellermann, A. J. Schwenk, Eds., vol. 2, 1991, pp. 871–898.
- [30] D. Knoke and S. Yang, *Social Network Analysis*, Quantitative Applications in the Social Sciences, SAGE Publications, 2019.
- [31] D. B. Johnson, “Efficient algorithms for shortest paths in sparse networks,” *Journal of the ACM*, vol. 24, no. 1, pp. 1–13, 1977. DOI: 10.1145/321992.321993.
- [32] R. Albert, H. Jeong, and A. L. Barabási, “Error and attack tolerance of complex networks,” *Nature*, vol. 406, pp. 378–382, 2000.

Fateme Movahedi, Richard Frank

Received August 10, 2022

Revised April 20, 2023

Accepted April 25, 2023

Fateme Movahedi

ORCID: <https://orcid.org/nnnn-nnnn-nnnn-nnnn>

Department of Mathematics,

Faculty of Sciences, Golestan University, Gorgan, Iran.

MoCSSy Program, The IRMACS Centre, Simon Fraser University, Burnaby, British Columbia, Canada.

E-mail: f.movahedi@gmail.com

Richard Frank

ORCID: <https://orcid.org/nnnn-nnnn-nnnn-nnnn>

School of Criminology International CyberCrime Research Centre,

Simon Fraser University, 8888 University Drive, Burnaby, B.C., Canada. V5A 1S6.

E-mail: rfrank@sfu.ca

Total Italian domatic number of graphs

Seyed Mahmoud Sheikholeslami*, Lutz Volkmann

Abstract

Let G be a graph with vertex set $V(G)$. An *Italian dominating function* (IDF) on a graph G is a function $f : V(G) \rightarrow \{0, 1, 2\}$ such that every vertex v with $f(v) = 0$ is adjacent to a vertex u with $f(u) = 2$ or to two vertices w and z with $f(w) = f(z) = 1$. An IDF f is called a *total Italian dominating function* if every vertex v with $f(v) \geq 1$ is adjacent to a vertex u with $f(u) \geq 1$. A set $\{f_1, f_2, \dots, f_d\}$ of distinct total Italian dominating functions on G with the property that $\sum_{i=1}^d f_i(v) \leq 2$ for each vertex $v \in V(G)$, is called a *total Italian dominating family* (of functions) on G . The maximum number of functions in a total Italian dominating family on G is the *total Italian domatic number* of G , denoted by $d_{tI}(G)$. In this paper, we initiate the study of the total Italian domatic number and present different sharp bounds on $d_{tI}(G)$. In addition, we determine this parameter for some classes of graphs.

Keywords: Total Italian domination number, Total Italian domatic number.

MSC 2010: 05C69.

1 Introduction

For definitions and notations not given here we refer to [10]. We consider simple graphs G with vertex set $V = V(G)$ and edge set $E = E(G)$. The *order* of G is $n = n(G) = |V(G)|$. The *open neighborhood* of a vertex v is the set $N(v) = N_G(v) = \{u \in V(G) \mid uv \in E(G)\}$ and its *closed neighborhood* is the set $N[v] = N_G[v] = N(v) \cup \{v\}$. The *degree* of vertex $v \in V(G)$ is $d(v) = d_G(v) = |N(v)|$. The *maximum degree* and *minimum degree* of G are denoted by $\Delta = \Delta(G)$ and

* Corresponding author: s.m.sheikholeslami@azaruniv.ac.ir

$\delta = \delta(G)$, respectively. The *complement* of a graph G is denoted by \overline{G} . A *leaf* is a vertex of degree one, and its neighbor is called a *support vertex*. An edge incident with a leaf is called a *pendant edge*. We write P_n for the path of order n , C_n for the cycle of length n , and K_n for the complete graph of order n . The *corona* $H \circ K_1$ of a graph H is that graph obtained from H by adding a pendant edge to each vertex of H .

A set $S \subseteq V(G)$ is a (*total*) *dominating set* of G if every vertex of $(V(G) \setminus S)$ is adjacent to a vertex in S . The (*total*) *domination number* of a graph G is the cardinality of a smallest (*total*) dominating set of G and is denoted ($\gamma_t(G)$) $\gamma(G)$. The (*total*) *domatic number* of G , ($d_t(G)$) $d(G)$ is the maximum number of classes of a partition of $V(G)$ such that each class is a (*total*) dominating set of G .

Cockayne, Dreyer, S.M. Hedetniemi, and S.T. Hedetniemi [8] introduced the concept of *Roman domination* in graphs, and since then a lot of related variations and generalizations have been studied (see [4]–[7]). In this paper, we continue the study of Roman and Italian dominating functions in graphs G . If $f : V(G) \rightarrow \{0, 1, 2\}$ is a function, then let (V_0, V_1, V_2) be the ordered partition of $V(G)$ induced by f , where $V_i = \{v \in V(G) \mid f(v) = i\}$ for $i \in \{0, 1, 2\}$. There is 1-1 correspondence between the function f and the ordered partition (V_0, V_1, V_2) . So, we also write $f = (V_0, V_1, V_2)$.

A function $f : V(G) \rightarrow \{0, 1, 2\}$ is a *Roman dominating function* (RDF) on G , if every vertex v with $f(v) = 0$ is adjacent to a vertex u with $f(u) = 2$. The *Roman domination number* $\gamma_R(G)$ is the minimum weight of an RDF on G .

A *total Roman dominating function* (TRDF) on a graph G without isolated vertices is defined in [11] as a Roman dominating function f on G with the property that the subgraph induced by $V_1 \cup V_2$ has no isolated vertex. The *total Roman domination number* $\gamma_{tR}(G)$ is the minimum weight of a TRDF on G .

An *Italian dominating function* (IDF) on a graph G is defined in [3] as a function $f : V(G) \rightarrow \{0, 1, 2\}$ such that $f(N(v)) \geq 2$ for every vertex v with $f(v) = 0$. The weight of an IDF f is the value $\omega(f) = \sum_{u \in V(G)} f(u)$. The *Italian domination number* $\gamma_I(G)$ is the minimum weight of an IDF on G . In [3], the authors called the Italian domination number the Roman $\{2\}$ -domination number.

A *total Italian dominating function* (TIDF) on a graph G without isolated vertices is defined in [1] as an Italian dominating function f on G with the property that the subgraph induced by $V_1 \cup V_2$ has no isolated vertex. The *total Italian domination number* $\gamma_{tI}(G)$ is the minimum weight of a TIDF on G . A TIDF on G with weight $\gamma_{tI}(G)$ is called a $\gamma_{tI}(G)$ -function.

A set $\{f_1, f_2, \dots, f_d\}$ of distinct total Roman dominating functions on a graph G without isolated vertices with the property that $\sum_{i=1}^d f_i(v) \leq 2$ for each vertex $v \in V(G)$, is called in [2] a *total Roman dominating family* (of functions) on G . The maximum number of functions in a total Roman dominating family on G is the *total Roman domatic number* $d_{tR}(G)$ of G .

A set $\{f_1, f_2, \dots, f_d\}$ of distinct total Italian dominating functions on a graph G without isolated vertices with the property that $\sum_{i=1}^d f_i(v) \leq 2$ for each vertex $v \in V(G)$, is called a *total Italian dominating family* (of functions) on G . The maximum number of functions in a total Italian dominating family on G is the *total Italian domatic number* $d_{tI}(G)$ of G . Italian domatic number has been studied in [12], [14].

If G is a graph without isolated vertices, then $\gamma_{tI}(G) \leq \gamma_{tR}(G)$ and $d_{tR}(G) \leq d_{tI}(G)$. On the other hand, if $S_1 \cup S_2 \cup \dots \cup S_{d_t}$ is a partition of $V(G)$ such that each class is a total dominating set of G , then the family $\{f_1, f_2, \dots, f_{d_t}\}$ of functions, where f_i is defined on G by $f_i(x) = 2$ for $x \in S_i$ and $f_i(x) = 0$ otherwise, is a total Italian dominating family (of functions) on G and so $d_t(G) \leq d_{tI}(G)$.

In this paper, we initiate the study of the total Italian domatic number, and we present different sharp bounds on $d_{tI}(G)$. In particular, we prove the Nordhaus-Gaddum type result $d_{tI}(G) + d_{tI}(\overline{G}) \leq n$ for graphs G of order $n \geq 4$ with $\delta(G) \geq 1$ and $\delta(\overline{G}) \geq 1$. In addition, we determine the total Italian domatic number for some classes of graphs.

We make use of the following known results.

Proposition 1 ([1]). *If $n \geq 3$, then $\gamma_{tI}(P_n) = \lceil \frac{2n+2}{3} \rceil$ and $\gamma_{tI}(C_n) = \lceil \frac{2n}{3} \rceil$.*

Proposition 2 ([1]). *Let G be a connected graph of order $n \geq 2$. Then $\gamma_{tI}(G) \leq n$, with equality if and only if G is the corona $F \circ K_1$ of some*

connected graph F or $G = P_3$.

Proposition 3 ([1]). *If G is a graph without isolated vertices of order n , then $\gamma_{tI}(G) \geq 2$. We have $\gamma_{tI}(G) = 2$ if and only if there exist two vertices u and v with $d(u) = d(v) = n - 1$.*

Proposition 4 ([1]). *If G is a graph without isolated vertices of order n , then*

$$\gamma_{tI}(G) \geq \left\lceil \frac{2n}{\Delta(G) + 1} \right\rceil.$$

Proposition 5 ([2]). *If $t \geq s \geq 1$ are integers, then $d_{tR}(K_{t,s}) = s$.*

2 Bounds and Properties

In this section, we present sharp bounds on the total Italian domatic number and investigate its basic properties. In addition, we determine this parameter for some classes of graphs.

Theorem 1. *Let G be a graph of order $n \geq 3$ without isolated vertices. If G has $2 \leq p \leq n$ vertices of degree $n - 1$, then $d_{tI}(G) \geq p$.*

Proof. Let $\{v_1, v_2, \dots, v_n\}$ be the vertex set of G , and let, without loss of generality, v_1, v_2, \dots, v_p be the vertices of degree $n - 1$. If $p \geq 3$, then define the functions f_i by $f_i(v_i) = f_i(v_{i+1}) = 1$ and $f_i(x) = 0$ for $x \neq v_i, v_{i+1}$ for $1 \leq i \leq p$, where $v_{p+1} = v_1$. Then f_1, f_2, \dots, f_p are distinct TIDF on G such that $\sum_{i=1}^p f_i(x) \leq 2$ for each $x \in V(G)$. Therefore, $\{f_1, f_2, \dots, f_p\}$ is a total Italian dominating family on G and, thus, $d_{tI}(G) \geq p$. If $p = 2$, then define f_1 by $f_1(v_1) = f_1(v_2) = 1$ and $f_1(x) = 0$ for $x \neq v_1, v_2$. Moreover, define f_2 by $f_2(v_i) = 1$ for all $1 \leq i \leq n$. Since $n \geq 3$, it follows that $\{f_1, f_2\}$ is total dominating family on G , and so $d_{tI}(G) \geq 2 = p$ also in this case. \square

Theorem 2. *If G is a graph of order n without isolated vertices, then*

$$\gamma_{tI}(G) \cdot d_{tI}(G) \leq 2n.$$

Moreover, if we have the equality $\gamma_{tI}(G) \cdot d_{tI}(G) = 2n$, then for each total Italian dominating family $\{f_1, f_2, \dots, f_d\}$ with $d = d_{tI}(G)$, each f_i is a $\gamma_{tI}(G)$ -function and $\sum_{i=1}^d f_i(v) = 2$ for all $v \in V(G)$.

Proof. Let $\{f_1, f_2, \dots, f_d\}$ be a total Italian dominating family on G with $d = d_{tI}(G)$. Then

$$\begin{aligned} d \cdot \gamma_{tI}(G) &= \sum_{i=1}^d \gamma_{tI}(G) \leq \sum_{i=1}^d \sum_{v \in V(G)} f_i(v) = \\ &= \sum_{v \in V(G)} \sum_{i=1}^d f_i(v) \leq \sum_{v \in V(G)} 2 = 2n. \end{aligned}$$

If $\gamma_{tI}(G) \cdot d_{tI}(G) = 2n$, then the two inequalities occurring in the proof become equalities. Hence, for the total Italian dominating family $\{f_1, f_2, \dots, f_d\}$ on G and for each i , $\sum_{v \in V(G)} f_i(v) = \gamma_{tI}(G)$. Thus, each f_i is a $\gamma_{tI}(G)$ -function and $\sum_{i=1}^d f_i(v) = 2$ for all $v \in V(G)$. \square

Proposition 3 and Theorem 2 imply the next result immediately.

Corollary 1. *If G is a graph of order n without isolated vertices, then $d_{tI}(G) \leq n$.*

Theorem 3. *If G is a graph of order n without isolated vertices, then*

$$d_{tI}(G) \leq \delta(G) + 1.$$

Moreover, if $F = \{f_1, f_2, \dots, f_{d_{tI}(G)}\}$ is a total Italian dominating family with $d_{tI}(G) = \delta(G) + 1$, then for any minimum degree vertex v , the following statements must be held:

- (a) $\sum_{u \in N[v]} f_i(u) = 2$ for each $f_i \in F$ and $\sum_{i=1}^d f_i(u) = 2$ for each $u \in N[v]$.
- (b) There are exactly $\delta(G) - 1$ Italian dominating functions such that $f_i(v) = 0$, and exactly two TIDFs such that $f_i(v) = 1$.
- (c) If $f_i(v) = 1$, then $f_i(u) = 0$ for each neighbor of v but exactly one which is assigned 1 under f_i .

Proof. Let $\{f_1, f_2, \dots, f_d\}$ be a total Italian dominating family on G with $d = d_{tI}(G)$. Assume that v is a vertex of minimum degree. It follows from the definitions that $\sum_{x \in N[v]} f_i(x) \geq 2$ for each $i \in \{1, 2, \dots, d\}$. Therefore, we deduce that

$$2d \leq \sum_{i=1}^d \sum_{x \in N[v]} f_i(x) = \sum_{x \in N[v]} \sum_{i=1}^d f_i(x) \leq \sum_{x \in N[v]} 2 = 2(\delta(G) + 1) \quad (1)$$

and so, $d_{tI}(G) = d \leq \delta(G) + 1$.

Assume that the equality holds, that is $d_{tI}(G) = \delta(G) + 1$. Then the inequalities occurring in (1) become equalities which gives the properties given in the statement (a).

Without loss of generality, assume that $f_1, f_2, \dots, f_{d'}$ are the TIDFs such that $f_i(v) = 0$ (for some d'). For each i such that $f_i(v) = 0$, we must have $\sum_{x \in N(v)} f_i(x) \geq 2$. Therefore,

$$2d' \leq \sum_{i=1}^{d'} \sum_{x \in N(v)} f_i(x) = \sum_{x \in N(v)} \sum_{i=1}^{d'} f_i(x) \leq \sum_{x \in N(v)} 2 = 2\delta(G). \quad (2)$$

If the equality holds in (2), that is $d' = \delta(G)$, then we must have $\sum_{i=1}^{d'} f_i(x) = 2$ for each $x \in N(v)$. It follows from $2 = \sum_{i=1}^{d'} f_i(x) \leq \sum_{i=1}^d f_i(x) \leq 2$ that $f_d(x) = 0$ for each $x \in N(v)$, which contradicts the totality of f_d . Thus, there are at most $\delta(G) - 1$ total Italian dominating functions such that $f_i(v) = 0$. Since there are at most two Italian dominating functions such that $f_i(v) \geq 1$, we deduce that there are exactly $\delta(G) - 1$ Italian dominating functions such that $f_i(v) = 0$, and exactly two TIDFs such that $f_i(v) = 1$. Thus, the statement (b) holds.

(c) immediately comes from $\sum_{u \in N[v]} f_i(u) = 2$ (see (a)). This completes the proof. \square

For regular graphs, we can use the statements about vertices of minimum degree at equality to every vertex, so that if $d = d_{tI}(G) = \delta(G) + 1$, and $F = \{f_1, f_2, \dots, f_d\}$ is a family of Italian dominating functions, then this implies each Italian dominating function is a function $f_i : V(G) \rightarrow \{0, 1\}$. So we can consider the Italian dominating

functions as indicator functions, and in what follows, it will be convenient to restate the property that $d_{tI}(G) = \delta(G) + 1$ for a regular graph G in terms of a family of sets. The proof of next result is essentially similar to the proof of Lemma 1 in [12].

Corollary 2. *Let G be a δ -regular graph, where $\delta \geq 1$. Then $d_{tI}(G) = \delta + 1$ if and only if there are distinct sets $S_1, S_2, \dots, S_{\delta+1}$, $S_i \subseteq V(G)$, that satisfy the following:*

- (a) *Every vertex of G appears in exactly two sets S_i .*
- (b) *Each set S_i induces a perfect matching, i.e., the induced subgraph $G[S_i]$ is 1-regular.*
- (c) *For any vertex $v \notin S_i$, $|N(v) \cap S_i| = 2$.*
- (d) *For each i , $|S_i| = \frac{2n}{\delta+1} = \gamma_{tI}(G)$.*

Proof. Suppose that there exist sets $S_1, S_2, \dots, S_{\delta+1} \subseteq V(G)$ satisfying (a),(b),(c) and (d). Let f_i be the characteristic function of S_i for each i . By Conditions (b) and (c), each f_i is a total Italian dominating function, and by Condition (a), these functions form a total Italian dominating family with $\delta + 1$ total Italian dominating functions. Since $d_{tI}(G) \leq \delta + 1$, we get $d_{tI}(G) = \delta + 1$.

Conversely, assume that $d_{tI}(G) = \delta + 1$ and let $F = \{f_1, f_2, \dots, f_{\delta+1}\}$ be a total Italian dominating family. Since G is δ -regular, we deduce from Theorem 3-(b) that $f_i(v) \leq 1$ for each i and each $v \in V(G)$. For each f_i , define $S_i = \{v \in V(G) \mid f_i(v) = 1\}$. Note that $\omega(f) = |S_i|$. Clearly, (a) and (c) come from Theorem 3-(b). Also (b) follows from Theorem 3-(c). Now we prove (d). Using Proposition 4 and noting that G is δ -regular, we obtain $\lceil \frac{2n}{\delta+1} \rceil (\delta+1) \leq \sum_{i=1}^{\delta+1} |S_i| = 2n \leq \lceil \frac{2n}{\delta+1} \rceil (\delta+1)$. Equality is possible only if $2n$ is divisible by $\delta + 1$, and $|S_i| = \frac{2n}{\delta+1}$ for each i . \square

Corollary 3. *Let G be a graph of order $n \geq 3$ without isolated vertices. Then $d_{tI}(G) = n$ if and only if $G = K_n$.*

Proof. If $G = K_n$, then Theorem 1 and Corollary 1 imply $d_{tI}(G) = n$.

Conversely, assume that $d_{tI}(G) = n$. If $\delta(G) \leq n - 2$, then Theorem 3 yields the contradiction $d_{tI}(G) \leq \delta(G) + 1 \leq n - 1$. Therefore, $\delta(G) = n - 1$ and, thus, $G = K_n$. \square

The Cartesian product of two graphs G and H , denoted $G \square H$, is a graph whose vertex set is $V(G) \times V(H) = \{(x, y) \mid x \in V(G) \text{ and } y \in V(H)\}$ and two vertices (x_1, y_1) and (x_2, y_2) of $G \square H$ are adjacent if and only if either $x_1 = x_2$ and $y_1 y_2 \in E(H)$ or $y_1 = y_2$ and $x_1 x_2 \in E(G)$. It is shown that, for any two graphs G and H without isolated vertices, $d_t(G \square H) \geq \max\{d(G), d(H)\}$ [9].

Corollary 4. *If $n \geq 2$, then $d_{tI}(K_n \square K_2) = n$.*

Proof. Since $d(K_n) = n$, we have $d_{tI}(K_n \square K_2) \geq d_t(K_n \square K_2) \geq \max\{d(K_n), d(K_2)\} = n$. On the other hand, one can easily see that $\gamma_{tI}(K_n \square K_2) = 4$ and so by Theorem 2 we have $d_{tI}(K_n \square K_2) \leq \frac{4n}{4} = n$. Thus, $d_{tI}(K_n \square K_2) = n$. \square

Theorem 4. *Let C_n be a cycle of length $n \geq 3$. Then $d_{tI}(C_n) = 3$ when $n \equiv 0 \pmod{3}$ and $d_{tI}(C_n) = 2$ when $n \equiv 1, 2 \pmod{3}$.*

Proof. Let $n \equiv 0 \pmod{3}$, and let $C_n = v_1 v_2 \dots v_n v_1$ with $n = 3p$ for an integer $p \geq 1$. Define the functions f, g , and h by $f(v_{3i-2}) = f(v_{3i-1}) = 1$ and $f(v_{3i}) = 0$, $g(v_{3i-1}) = g(v_{3i}) = 1$ and $g(v_{3i-2}) = 0$, and $h(v_{3i}) = h(v_{3i-2}) = 1$ and $h(v_{3i-1}) = 0$ for $1 \leq i \leq p$. Then f, g , and h are total Italian dominating functions on C_n such that $f(x) + g(x) + h(x) = 2$ for each vertex $x \in V(C_n)$. Therefore, $\{f, g, h\}$ is a total Italian dominating family on C_n and, thus, $d_{tI}(C_n) \geq 3$. Theorem 3 yields to $d_{tI}(C_n) \leq 3$ and so $d_{tI}(C_n) = 3$ in this case.

Let now $n \equiv 1, 2 \pmod{3}$ and $C_n = v_1 v_2 \dots v_n v_1$. Theorem 2 and Proposition 1 imply

$$d_{tI}(C_n) \leq \frac{2n}{\gamma_{tI}(C_n)} = \frac{2n}{\lceil \frac{2n}{3} \rceil} < 3$$

and, hence, $d_{tI}(C_n) \leq 2$. Define the functions f and g by $f(v_i) = 1$ for $1 \leq i \leq n$ and $g(v_1) = 0$ and $g(v_i) = 1$ for $2 \leq i \leq n$. Then f and g are total Italian dominating functions on C_n such that $f(x) + g(x) \leq 2$ for each vertex $x \in V(C_n)$. Therefore, $\{f, g\}$ is a total Italian dominating

family on C_n and, thus, $d_{tI}(C_n) \geq 2$. This leads to $d_{tI}(C_n) = 2$ in this case. \square

Proposition 6. *If P_n is a path of order $n \geq 5$, then $d_{tI}(P_n) = 2$.*

Proof. Let $P_n = v_1v_2 \dots v_n$. Define the functions f and g by $f(v_i) = 1$ for $1 \leq i \leq n$ and $g(v_3) = 0$ and $g(v_i) = 1$ for $1 \leq i \leq n$ with $i \neq 3$. Then f and g are total Italian dominating functions on P_n such that $f(x) + g(x) \leq 2$ for each vertex $x \in V(P_n)$. Therefore, $\{f, g\}$ is a total Italian dominating family on P_n and, thus, $d_{tI}(P_n) \geq 2$. Theorem 3 implies $d_{tI}(P_n) \leq 2$ and so we obtain $d_{tI}(P_n) = 2$. \square

The proof of the next proposition is identical to the proof of Proposition 5 and is, therefore, omitted.

Proposition 7. *If $t \geq s \geq 1$ are integers, then $d_{tI}(K_{t,s}) = s$.*

Theorem 5. *Let G be a connected graph of order $n \geq 2$. Then $d_{tI}(G) = 1$ if and only if every vertex of G is a leaf or a support vertex.*

Proof. Let G contain a vertex w which is neither a leaf nor a support vertex. Since w is not a leaf, w has at least two neighbors, and since w is not a support vertex, $G - w$ has no isolated vertex. Therefore, the function f with $f(w) = 0$ and $f(x) = 1$ for $x \in V(G) \setminus \{w\}$ is a TIDF on G . In addition, the function g with $f(x) = 1$ for all $x \in V(G)$ is also a TIDF on G with the property that $f(x) + g(x) \leq 2$ for all $x \in V(G)$. Therefore, $\{f, g\}$ is a total Italian dominating family on G and, thus, $d_{tI}(G) \geq 2$.

Conversely, assume that each vertex of G is a leaf or a support vertex. Theorem 3 implies $d_{tI}(G) \leq 2$. Suppose that $\{f, g\}$ is a total Italian dominating family on G . If v is a support vertex, then the definitions lead to $f(v), g(v) \geq 1$. If $f(v) = 2$, then the condition $f(v) + g(v) \leq 2$ yields the contradiction $g(v) = 0$. Thus, $f(v) = g(v) = 1$ for all support vertices v . It follows that $f(u) = g(u) = 1$ for all leaves u , a contradiction to the condition that f and g are distinct. Consequently, $d_{tI}(G) = 1$, and the proof is complete. \square

Theorem 6. *Let G be a connected graph of order $n \geq 3$. Then*

$$\gamma_{tI}(G) + d_{tI}(G) \leq n + 2,$$

with equality if and only if $G = K_n$.

Proof. If $d_{tI}(G) = 1$, then Proposition 2 implies $\gamma_{tI}(G) + d_{tI}(G) \leq n + 1$. Let next $d_{tI}(G) \geq 2$. It follows from Theorem 2 that

$$\gamma_{tI}(G) + d_{tI}(G) \leq \frac{2n}{d_{tI}(G)} + d_{tI}(G).$$

Using the bounds $2 \leq d_{tI}(G) \leq n$ (see Corollary 1), and the fact that the function $g(x) = \frac{2n}{x} + x$ is decreasing for $2 \leq x \leq \sqrt{2n}$ and increasing for $\sqrt{2n} \leq x \leq n$, we obtain

$$\gamma_{tI}(G) + d_{tI}(G) \leq \frac{2n}{d_{tI}(G)} + d_{tI}(G) \leq \max\{n + 2, 2 + n\} = n + 2, \quad (3)$$

and the bound is proved.

If $G = K_n$, then we deduce from Proposition 3 and Corollary 3 that $\gamma_{tI}(G) + d_{tI}(G) = n + 2$.

Conversely, assume that $\gamma_{tI}(G) + d_{tI}(G) = n + 2$. It follows from (3) that

$$n + 2 = \gamma_{tI}(G) + d_{tI}(G) \leq \frac{2n}{d_{tI}(G)} + d_{tI}(G) \leq n + 2$$

and, therefore, $d_{tI}(G) = 2$ and $\gamma_{tI}(G) = n$ or $d_{tI}(G) = n$ and $\gamma_{tI}(G) = 2$. If $d_{tI}(G) = n$ and $\gamma_{tI}(G) = 2$, then Corollary 3 yields $G = K_n$. If $d_{tI}(G) = 2$ and $\gamma_{tI}(G) = n$, then Proposition 2 implies $G = F \circ K_1$ for a connected graph F or $G = P_3$. But now Theorem 5 leads to the contradiction $d_{tI}(G) = 1$. \square

3 Nordhaus-Gaddum type results

Results of Nordhaus-Gaddum type study the extreme values of the sum or product of a parameter on a graph and its complement. In their classical paper [13], Nordhaus and Gaddum discussed this problem for the chromatic number. We establish such inequalities for the total Italian domatic number.

Theorem 7. *If G is a graph of order $n \geq 4$ with $\delta(G) \geq 1$ and $\delta(\overline{G}) \geq 1$, then*

$$d_{tI}(G) + d_{tI}(\overline{G}) \leq n.$$

Proof. Theorem 3 implies

$$d_{tI}(G) + d_{tI}(\overline{G}) \leq (\delta(G) + 1) + (\delta(\overline{G}) + 1) = \delta(G) + 1 + (n - \Delta(G) - 1) + 1.$$

If G is not regular, then $\Delta(G) - \delta(G) \geq 1$, and the inequality chain above leads to the desired bound.

Let now G be δ -regular. Then \overline{G} is $\overline{\delta}$ -regular with $\overline{\delta} = n - \delta - 1$. Assume, without loss of generality, that $\delta \leq \overline{\delta}$.

If $\delta = 1$, then $G = \frac{n}{2}K_2$ and, thus, $d_{tI}(G) = 1$. According to Corollaries 1 and 3, we observe that $d_{tI}(\overline{G}) \leq n - 1$ and, thus, $d_{tI}(G) + d_{tI}(\overline{G}) \leq n$.

Thus, let now $\delta \geq 2$ and $n = p(\delta + 1) + r$ with integers $p \geq 1$ and $0 \leq r \leq \delta$. If $r \neq 0, \frac{\delta+1}{2}$, then Corollary 2 implies $d_{tI}(G) \leq \delta$ and, as above, we obtain $d_{tI}(G) + d_{tI}(\overline{G}) \leq n$. Next we discuss the case $r = 0$ or $r = \frac{\delta+1}{2}$.

Case 1: Let $r = 0$ and, therefore, $n = p(\delta + 1)$. We also have $n = (\overline{\delta} + 1) + \delta$ with $2 \leq \delta \leq \overline{\delta}$. If $\delta \neq \frac{\overline{\delta}+1}{2}$, then Corollary 2 yields $d_{tI}(\overline{G}) \leq \overline{\delta}$, and we obtain $d_{tI}(G) + d_{tI}(\overline{G}) \leq n$ as above. Let now $\delta = \frac{\overline{\delta}+1}{2}$. Then

$$n = \overline{\delta} + 1 + \frac{\overline{\delta} + 1}{2} = \frac{3}{2}(\overline{\delta} + 1) = \frac{3}{2}(n - \delta)$$

and so $n = 3\delta$. Hence, $n = p(\delta + 1) = 3\delta$ and, thus, $p = 2$. We deduce that $\delta = 2$ and $n = 6$. Consequently, G is a cycle of length 6 or the union of two cycles of length 3. Using Theorem 4 and Proposition 7, it is easy to verify that $d_{tI}(G) + d_{tI}(\overline{G}) = 6 = n$ in both cases.

Case 2: Let $r = \frac{\delta+1}{2}$ and, therefore, $n = p(\delta + 1) + \frac{\delta+1}{2}$. As in Case 1, there remains the case that $n = 3\delta$. Hence, $n = 3\delta = (p + \frac{1}{2})(\delta + 1)$ and so $p \leq 2$. If $p = 1$, then we obtain the contradiction $\delta = 1$. If $p = 2$, then $\delta = 5$ and $n = 15$, a contradiction to the fact that the number of vertices of odd degree is even. \square

Since $d_{tR}(G) \leq d_{tI}(G)$, Theorem 7 leads to the next known Nordhaus-Gaddum bound.

Theorem 8 ([2]). *If G is a graph of order $n \geq 4$ with $\delta(G) \geq 1$ and $\delta(\overline{G}) \geq 1$, then*

$$d_{tR}(G) + d_{tR}(\overline{G}) \leq n.$$

Theorem 9. *If G is a graph of order $n \geq 5$ with $\delta(G) \geq 1$ and $\delta(\overline{G}) \geq 1$, then*

$$d_{tI}(G) + d_{tI}(\overline{G}) \geq 3.$$

Proof. Assume, without loss of generality, that $d_{tI}(G) \leq d_{tI}(\overline{G})$. If $d_{tI}(G) \geq 2$, then we even see that $d_{tI}(G) + d_{tI}(\overline{G}) \geq 4$. So let now $d_{tI}(G) = 1$.

If G is not connected, then the condition $\delta(G) \geq 1$ shows that \overline{G} is connected such that $\delta(\overline{G}) \geq 2$. Therefore, Theorem 5 leads to $d_{tI}(\overline{G}) \geq 2$, and we obtain $d_{tI}(G) + d_{tI}(\overline{G}) \geq 3$.

Let now G be connected. Then it follows from Theorem 5 that each vertex of G is a leaf or a support vertex. Let $S(G) = \{v_1, v_2, \dots, v_s\}$ be the set of support vertices. Since $\delta(\overline{G}) \geq 1$, we observe that $s \geq 2$. If $s = 2$, then the condition $n \geq 5$ shows that v_1 or v_2 , say v_1 , is adjacent to more than one leaf. We deduce that v_2 is neither a leaf nor a support vertex of \overline{G} . Since \overline{G} is connected, it follows from Theorem 5 that $d_{tI}(\overline{G}) \geq 2$ and, thus, $d_{tI}(G) + d_{tI}(\overline{G}) \geq 3$. If $s \geq 3$, then $\delta(\overline{G}) \geq 2$, and Theorem 5 leads to $d_{tI}(G) + d_{tI}(\overline{G}) \geq 3$ again. \square

Since $d_{tI}(P_4) + d_{tI}(\overline{P_4}) = 2$, we observe that the condition $n \geq 5$ in Theorem 9 is necessary.

References

- [1] H. Abdollahzadeh Ahangar, M. Chellali, S.M. Sheikholeslami, and J.C. Valenzuela-Tripodoro, “Total Roman $\{2\}$ -dominating functions in graphs,” *Discuss. Math. Graph Theory*, vol. 42, pp. 937–958, 2022.
- [2] J. Amjadi, S. Nazari-Moghaddam, and S.M. Sheikholeslami, “Total Roman domatic number of a graph,” *Asian-Eur. J. Math.*, vol. 13, no. 06, Article No. 2050110, 12 p., 2020.

- [3] M. Chellali, T. Haynes, S.T. Hedetniemi, and A. McRae, “Roman $\{2\}$ -domination,” *Discrete Appl. Math.*, vol. 204, pp. 22–28, 2016.
- [4] M. Chellali, N. Jafari Rad, S. M. Sheikholeslami, and L. Volkmann, “Roman domination in graphs,” in *Topics in Domination in Graphs*, T. W. Haynes, S. T. Hedetniemi, and M. A. Henning, Eds. Springer, 2020, pp. 365–409.
- [5] M. Chellali, N. Jafari Rad, S. M. Sheikholeslami, and L. Volkmann, “Varieties of Roman domination,” in *Structures of Domination in Graphs*, T. W. Haynes, S. T. Hedetniemi, and M. A. Henning, Eds. Springer, 2021, pp. 273–307.
- [6] M. Chellali, N. Jafari Rad, S. M. Sheikholeslami, and L. Volkmann, “Varieties of Roman domination II,” *AKCE Int. J. Graphs Comb.*, vol. 17, pp. 966–984, 2020.
- [7] M. Chellali, N. Jafari Rad, S.M. Sheikholeslami, and L. Volkmann, “The Roman domatic problem in graphs and digraphs: A survey,” *Discuss. Math. Graph Theory*, vol. 42, pp. 861–891, 2022.
- [8] E. J. Cockayne, P. A. Dreyer, S. M. Hedetniemi, and S. T. Hedetniemi, “Roman domination in graphs,” *Discrete Math.*, vol. 278, pp. 11–22, 2004.
- [9] P. Francis, D. Rajendraprasad, *On domatic and total domatic numbers of product graphs*, arXiv:2103.10713.
- [10] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, *Fundamentals of Domination in Graphs*, New York: Marcel Dekker, Inc., 1998.
- [11] C.-H. Liu and G. J. Chang, “Roman domination on strongly chordal graphs,” *J. Comb. Optim.*, vol. 26, pp. 608–619, 2013.
- [12] J. Lyle, “Regular graphs with large Italian domatic number,” *Commun. Comb. Optim.*, vol. 7, pp. 257–271, 2022.
- [13] E. A. Nordhaus and J. W. Gaddum, “On complementary graphs,” *Amer. Math. Monthly*, vol. 63, pp. 175–177, 1956.

- [14] L. Volkmann, “The Italian domatic number of a digraph,” *Commun. Comb. Optim.*, vol. 4, pp. 61–70, 2019.

Seyed Mahmoud Sheikholeslami, Lutz Volkmann

Received June 09, 2022

Accepted March 10, 2023

Seyed Mahmoud Sheikholeslami

ORCID: <https://orcid.org/0000-0003-2298-4744>

Department of Mathematics

Azarbaijan Shahid Madani University

Tabriz, I. R. Iran

E-mail: s.m.sheikholeslami@azaruniv.ac.ir

Lutz Volkmann

ORCID: <https://orcid.org/0000-0003-3496-277X>

Lehrstuhl C für Mathematik

RWTH Aachen University

52062 Aachen, Germany

E-mail: volkm@math2.rwth-aachen.de

Comprehensive Performance Study of Hashing Functions

G. M. Sridevi, M. V. Ramakrishna, and D. V. Ashoka

Abstract

Most literature on hashing functions speaks in terms of hashing functions being either ‘good’ or ‘bad’. In this paper, we demonstrate how a hashing function that gives good results for one key set, performs badly for another. We also demonstrate that, for a single key set, we can find hashing functions that hash the keys with varying performances ranging from perfect to worst distributions. We present a study on the effect of changing the prime number ‘ p ’ on the performance of a hashing function from H_1 Class of Universal Hashing Functions. This paper then explores a way to characterize hashing functions by studying their performance over all subsets of a chosen Universe. We compare the performance of some popular hashing functions based on the average search performance and the number of perfect and worst-case distributions over different key sets chosen from a Universe. The experimental results show that the division-remainder method provides the best distribution for most key sets of the Universe when compared to other hashing functions including functions from H_1 Class of Universal Hashing Functions.

Keywords: H_1 Universal Class of Hashing Functions, Radix transformation, Mid-Square method, Multiplicative Hashing, Division-remainder method.

MSC 2020: 68P05, 68P10, 68P20.

ACM CCS 2020: Information systems—Database management system engines, Information systems—Information Storage Systems

1 Introduction

Hashing techniques have been used extensively in various applications involving storage and retrieval, cryptography, networking, cloud, etc. Hashing functions map keys to table indexes and provide fast retrieval of data on different types of storage. The terms ‘good hashing function’ and ‘bad hashing function’ are usually used while talking about how a hashing function distributes the data in the hash table. While some hashing functions might distribute keys perfectly without any collisions, a few others may result in all collisions. The term ‘collision’ refers to the event where more than one key gets mapped to the same location in memory. In general, a hashing function causing fewer collisions is said to be ‘good’ and is said to be ‘bad’ otherwise. Different textbooks on database and data structures talk about characteristics of a ‘good’ hashing function.

The performance of a hashing function is dependent on the input key set and cannot be considered to be ‘good’ or ‘bad’ independent of the input. It is reasonable to evaluate the performance of a hashing function over all possible subsets of a Universe and not just over one key set. This idea was proposed by Lum [1]. Lum recommends considering all the subsets of a Universe to study the performance of hashing functions. To our surprise, there is no reported literature which studies the performance based on this strategy so far. Carter and Wegman’s H_1 class of Universal Hashing functions was proved to provide practical performance on real files and is expected to provide better results [2], [3]. A comparison of the hashing functions with H_1 class of Universal Hashing functions has not been done so far as per our knowledge.

In this paper, we demonstrate the effect of the prime number on the performance of functions from H_1 class of Universal Hashing functions. We then present an exhaustive study on the performance of some of the known hashing functions in comparison with functions from H_1 class of hashing functions based on Lum’s model. While functions from Universal Class were expected to give the best results, experimental results show that the division-remainder method performs better than the functions from H_1 class of Universal Hashing functions for the Universe chosen.

The rest of the paper is organized as follows: Section II presents the related work; Section III demonstrates that the performance of hashing functions is dependent on the input, followed by a study on H_1 class of hashing functions and presents the model used for the study of hashing functions; Section IV presents the results and the paper is concluded in Section V.

2 Related Work

Most of the studies on hashing functions was presented in the 60s. Surprisingly, there has been no study on the performance of hashing functions since 70s. Peterson presented an early study on hashing functions based on the number of probes required to find a record [4]. Buchholz provided an analysis of hashing functions covering different aspects of hashing functions including an analysis of overflow handling methods but with minimal experimental results [5]. According to the author, a hashing function that involves division of the key by a prime value provides efficient distribution of keys. Few other studies on hashing techniques were presented by Mc Ilroy [6], Roberts [7] and Schay et al. [8], [9]. The performance of different hashing functions over different sets of keys of varying datatypes was presented by Lum et al. with a conclusion that the division method provides the best search results with fewer collisions [10].

Later, Lum proposed a way to characterize the hashing functions by selecting key sets from a key space and hashing the keys using different hashing techniques [11]. Sorenson et al.'s survey on different hashing techniques along with collision-resolution techniques presents an analysis from a practitioner's point of view [12]. Deutscher et al. provide an excellent characterization and comparison of distribution-dependent hashing functions with that of division method [3].

Ramakrishna's study on H_1 class of Universal hashing functions defined by Carter and Wegman concludes that functions from this class can provide practical performance on real files [10]. In this paper, we present a comparison of the performance of some of the hashing techniques with functions from H_1 class. In the next section, we demonstrate that any hashing function cannot be declared as 'good' or 'bad' without considering the input given to the hashing function.

3 Hashing Functions: ‘good’ or ‘bad’

In this section, we consider a few examples to illustrate that the same hash function can perform differently for different key sets. We chose functions from H_1 class of Universal hashing functions defined by Carter and Wegman which have been claimed to be good for providing practical performance on real files [2]. A separate chaining method is applied to link the keys that result in collisions for our examples. For the demonstration, we consider 2 different scenarios:

Case 1: Same hashing function, different key sets.

We hashed 4 different key sets with the hashing function $h(x) = ((519x + 703) \bmod 36373) \bmod 13$ to a hash table of size 13. The key sets considered for the demonstration of the first case are as follows:

$$\begin{aligned}
 K_1 &= \{936, 748, 996, 815, 864, 867, 730, 971\} \\
 K_2 &= \{772, 841, 738, 907, 876, 932, 713, 816\} \\
 K_3 &= \{757, 861, 929, 744, 755, 797, 808, 836\} \\
 K_4 &= \{715, 952, 899, 754, 860, 913, 992, 781\}
 \end{aligned}$$

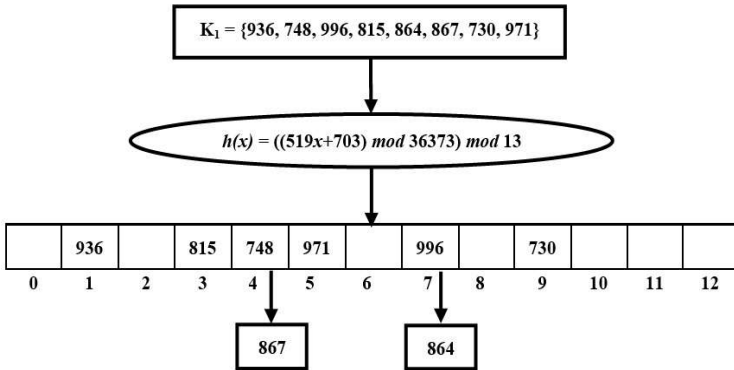


Figure 1. Good/Normal Distribution of K_1 by $h(x)$

Fig. 1 shows the distribution of the keys in key set K_1 to the hash table using the hash function $h(x)$. The hash function $h(x)$ causes only 2 collisions for key set K_1 . The function $h(x)$ distributes the keys from K_2 without any collisions making it a perfect hashing function for key

set K_2 . On the other hand, it results in a larger number of collisions for K_3 , with all keys hashing to either table index 8 or to index 10, and in all collisions for K_4 . The average search performance of $h(x)$ for different key sets is shown in Table 1. The hash function's performance over the key sets was found to be good for K_1 ; perfect for K_2 ; bad for K_3 and worst for K_4 with all keys hashing to table index 11. As we can see, a hashing function's performance depends on the key set chosen.

Table 1. Average Search Performance of $h(x)$ over different key sets

| Key Set | Average Successful Search Length | Performance |
|---------|----------------------------------|----------------|
| K_1 | 1.25 | Good |
| K_2 | 1 | Perfect |
| K_3 | 2.625 | Average/Normal |
| K_4 | 4.5 | Worst |

Case 2: Same key set, different functions.

In the previous scenario, we showed the performance of a single hashing function on different key sets. In another scenario, we show that, for a single key set, we can find functions $h_1, h_2...$ such that h_1 is perfect, h_2 is average and so on.

For our demonstration of the second case, we consider a key set $K_5 = \{763, 789, 841, 867, 893, 919, 945, 997\}$ of size 8 chosen randomly from a Universe U ranging from 700 to 1000. For the demonstration, the key set K_5 is hashed using 4 different hashing functions listed below:

$$\begin{aligned}
 h_1(x) &= ((412x + 371) \bmod 5443) \bmod 13 \\
 h_2(x) &= ((321x + 576) \bmod 27283) \bmod 13 \\
 h_3(x) &= ((513x + 413) \bmod 106759) \bmod 13 \\
 h_4(x) &= ((417x + 294) \bmod 158647) \bmod 13
 \end{aligned}$$

Fig. 2 shows the distribution of the key set K_5 by the hashing function $h_1(x)$ which is found to be perfect with no collisions. The same key set K_5 results in a few collisions when hashed with the function $h_2(x)$. A larger number of collisions is seen when K_5 is hashed with

$h_3(x)$ with all keys getting hashed to either index 0 or index 3 of the hash table. The hashing function $h_4(x)$ hashes the key set with all collisions by mapping all the keys to the address 0. The average successful search performance of the hash functions over the key set K_5 is shown in Table 2. This shows that, for the same key set, different hashing functions may perform differently. Choosing a hashing function that provides the best results is vital for any application.

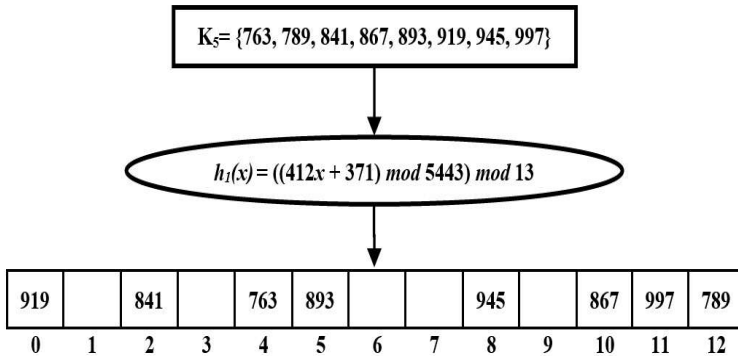


Figure 2. Perfect distribution for K_5 by $h_1(x)$

Table 2. Average Search Performance of different hashing functions over K_5

| Hashing Function | Average Successful Search Length | Performance |
|------------------|----------------------------------|-------------|
| $h_1(x)$ | 1 | Perfect |
| $h_2(x)$ | 1.625 | Good |
| $h_3(x)$ | 3 | Bad |
| $h_4(x)$ | 4.5 | Worst |

We have illustrated with examples that the performance of any hashing function is dependent on the key set and, for any given key set, we can find functions that vary in performance. Also, given any hashing function, we can find key sets for which the performance of the hashing function varies from perfect to good, bad, or even worse.

This can be done for any other class of hashing functions such as radix transformation, division-remainder method, and so on.

3.1 The effect of the prime ‘ p ’ on the performance of H_1 Class of Hashing Functions

We demonstrated that a hashing function cannot be considered ‘good’ or ‘bad’ without taking the input into consideration. For the demonstration, we used hashing functions from H_1 class of Universal hashing functions defined by Carter and Wegman of the form

$$h(x) = ((c * x + d) \bmod p) \bmod m, \quad (1)$$

where c and d are chosen at random, p is a large prime, and m is the table size. Functions from H_1 class are said to be Universal indicating that the maximum number of collisions expected is nearly n/m , where n is the size of the key set and m is the hash table size. We can observe that the performance of the hashing functions varied depending on the values chosen for ‘ c ’, ‘ d ’, and ‘ p ’. This motivated us to study the effect of varying the values of ‘ p ’ on the performance of H_1 class of Universal hashing functions.

In particular, we chose a key set with keys that generate the same remainder on dividing by the hash table size m taken as 13. We studied the performance of functions from H_1 class over a key set K of size 10 ranging from 700 to 1000. The key set considered for the study is

$$K = \{737, 945, 763, 815, 867, 841, 893, 789, 919, 997\}.$$

We can see that the keys chosen would result in all collisions if hashed using the modulo method. The keys from the key set K were hashed to the hash table using hashing functions of the form in Eq. (1). The values for ‘ c ’ and ‘ d ’ were kept constant at $c = 453$ and $d = 657$. The performance of the hashing function was evaluated for different values for ‘ p ’ to study the effect of the prime number on the performance of the hashing function and is shown in Table 3. The results indicate that as the ratio of ‘ $(cx+d)$ ’ to ‘ p ’ decreases, the number of collisions increases. Larger values of ‘ p ’ result in a higher number of collisions. If the value

of ‘ p ’ is too large, it results in all collisions. Whereas, smaller values result in fewer collisions and can also provide perfect distribution. In the next subsection, we present a comprehensive study of some of the popular hashing functions.

Table 3. Performance variation of a hashing function $h(x)$ on varying ‘ p ’

| p value | No. of overflow records | Average Search length | $\frac{(c*x+d)}{p}$ | Search Performance |
|-----------|-------------------------|-----------------------|---------------------|--------------------|
| 226637 | 9 | 5.5 | 1.47 | Worse |
| 170689 | 8 | 4.6 | 1.95 | Bad |
| 156967 | 9 | 5.5 | 1.1 | Worse |
| 128761 | 8 | 3.0 | 2.59 | Average |
| 70571 | 7 | 2.7 | 4.74 | Average |
| 36373 | 7 | 2.3 | 9.19 | Average |
| 32203 | 5 | 1.7 | 10.38 | Good |
| 30241 | 6 | 1.8 | 11.06 | Good |
| 19739 | 3 | 1.3 | 16.94 | Good |
| 13219 | 1 | 1.1 | 25.30 | Good |
| 10687 | 0 | 1 | 31.30 | Perfect |
| 7477 | 2 | 1.2 | 44.73 | Good |
| 5879 | 0 | 1 | 56.9 | Perfect |
| 2357 | 0 | 1 | 141.9 | Perfect |

3.2 Exhaustive Performance Study of Hashing Functions

We demonstrated that the performance of a hashing function is highly dependent on the input. Based on this fact, the best strategy to study the performance of a hashing function over any given Universe of keys would be to consider the performance of the hashing function over all the possible subsets of the Universe rather than on just a few key sets. As already demonstrated, different hashing functions may provide varied performance for same key set while they behave differently for other key sets. Hence, we consider exhaustive subsets of the Universe, according to which all possible subsets of a Universe must be considered for

hashing to properly understand how a hashing function distributes the data. This was suggested by Lum in his earlier studies. A study on Universal hashing functions based on this idea is presented here in comparison with other existing methods.

Consider a Universe of keys $U = \{k_1, k_2, k_3, \dots, k_n\}$ of size n . There are ${}^n C_r$ possible subsets KS (Key Set) of size r . Each of these subsets is hashed to the Hash Table individually, and the average search performance of the hashing function over all the subsets is taken into consideration to study the performance. Some key sets may get distributed perfectly without any collisions. Some may be worst cases (all keys colliding) and others will be in between. We evaluate the performance of the hashing functions by taking an average of all the extreme cases. The algorithm for the comprehensive study is presented in Algorithm 1.

Algorithm 1.

Input:
 Universe $U = \{10, 11, \dots, 29\}$
 Universe Size $n = 20$
 Table Size $m = 10$
 Subset Sizes $r = \{3, 4, \dots, 10\}$

Begin:
 1: For each r
 Generate all subsets/key sets of U (KS) of size r
 // $KS = \{KS_1, KS_2, \dots, KS_{{}^n C_r}\}$
 2: For each subset KS_x // $KS_x = \{k_1, k_2, \dots, k_r\}$
 Initialize Total Search Length for KS_x to 0
 3: For each key k_x in a subset KS_x
 hash(k_x)
 Compute Search length for k_x // ($SL(k_x)$)
 Total Search Length for $KS_x =$ Total Search Length for
 $KS_x + SL(k_x)$
 End For3
 Calculate Average Search Length over KS_x ($ASL(KS)$)
 End For2
 $ASL(KS) =$ Average search Length over ${}^n C_r$ subsets
 End For1

4 Results and Discussion

For our experiment, we considered mid-square method, radix transformation, division-remainder method, multiplicative method [2],[13], and H_1 class of hashing functions. A brief explanation of the hashing functions considered for the study is given below.

Mid-square method extracts the middle portion of square value of the key as the hash address for the key.

For example, for the key $x = 2518$, squaring the key gives 63,40,324. The middle of the squared value, i.e., '0' is taken as the hash address to store the key x .

Radix transformation transforms the key from decimal to a different base and the resulting value is used to generate the hash address. The key can be converted to any base of our choice. For demonstration, we convert the key to base 9.

Consider a key $x = 7698$. Converting the decimal 7698 to base 9 gives the value 11503. The converted value is then used to generate the hash address. For a table size of $m = 10$, we use a modulo operation on the transformed value with the table size to generate a hash address within the address range 0 to 9. For $m = 10$, we get 3 ($11503 \bmod 10$) as the hash address.

Division-remainder method or modulo method is of the form $h(x) = x \bmod m$, where x is the key and m is the hash table size. The remainder obtained on dividing the key by table size is taken as the hash address.

For $x = 1893$ and $m = 10$, we get 3 ($1893 \bmod 10$) as the hash address.

Multiplication method is of the form $h(x) = \text{floor}(m * ((x * c) \bmod 1))$, where c is a floating-point value ranging between 0 to 1 and m is the table size. The key x is multiplied by c , and the fractional part is extracted by applying a modulo operation with 1 on the product obtained. The remainder is then multiplied by the table size and the real part is taken as the hash address.

For example, for a key $x = 1788$, if $c = 0.572593$, $x * c$ is 1023.796284. Modulo operation on the product with 1 gives 0.796284 as the remainder which is multiplied with the table size m . If $m = 10$, we get 7.96284 as a result, from which the floor value is taken as the hash address, i.e., 7.

H_1 **class of hashing functions** defined by Carter and Wegman are of the form $h(x) = ((cx + d) \bmod p) \bmod m$, where c and d are integers chosen at random, p is a large prime, and m is the table size. For $x = 18$, if $c = 58$, $d = 67$, $p = 137$, and $m = 10$, we get 5 as the hash address.

For any given set of size n , the number of possible subsets of size r is exponential and is equal to ${}^n C_r$. Thus, we have to keep the size of the Universe n to be small so that the total time taken to conduct the experiments is within manageable limits. For $n = 100$, the number of possible subsets of size $r = 7$ is ${}^{100} C_7 = 16007560800$ and the time taken to generate all the subsets is nearly 42 hours. For $r = 8$, we get 186087894300 possible subsets which is estimated to take nearly 21 days to generate. For larger values of $r = 9$ and 10 , we get 1902231808400 and 17310309456440 possible subsets. It would take longer to generate and hash all the subsets to the hash table; hence, we chose a smaller Universe U of size $n = 20$, for which the number of subsets is 184756 for $r = 10$, which takes lesser time to generate. The Universe U chosen ranges from 10 to 29. Subsets are constructed for different sizes ' r ' ranging from 3 to 10. Each of these subsets is hashed to a Hash Table of size $m = 10$, and the average search performance is calculated over all the subsets.

Each of the hashing functions was used to hash all the possible subsets of the Universe for subset sizes $r = 3$ to 10 . For Universal hashing functions, we generated 100 different functions by varying the values of ' c ', ' d ' at random and keeping the prime ' p ' as constant. The average of Average search length for the 100 hashing functions was used to compute the search performance of Universal hashing functions. For example, if $r = 3$, for each hashing function, we get 1140 subsets; we take the average search length for all the subsets for each hashing function. This is repeated for 100 hashing functions. Later we take the

average of the 100 average search lengths to compute the performance of Universal hashing functions.

Table 4. Comparison based on Average Successful Search Length

| Subset Size (r) | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------------------------|------|------|-------|-------|-------|--------|--------|--------|
| No. of Subsets (${}^n C_r$) | 1140 | 4845 | 15504 | 38760 | 77520 | 125970 | 167960 | 184756 |
| Mid-Square method | 1.12 | 1.19 | 1.25 | 1.31 | 1.38 | 1.44 | 1.5 | 1.57 |
| Modulo method | 1.05 | 1.08 | 1.1 | 1.13 | 1.16 | 1.18 | 1.21 | 1.23 |
| Multipli-cation method | 1.13 | 1.19 | 1.25 | 1.31 | 1.38 | 1.44 | 1.5 | 1.57 |
| Decimal to base 11 | 1.06 | 1.09 | 1.12 | 1.16 | 1.19 | 1.22 | 1.25 | 1.28 |
| Decimal to base 15 | 1.16 | 1.23 | 1.31 | 1.39 | 1.47 | 1.55 | 1.63 | 1.71 |
| Uni-versal Hashing | 1.07 | 1.11 | 1.15 | 1.18 | 1.22 | 1.26 | 1.29 | 1.33 |

Table 4 shows the results of experiments based on successful search length. The subset size r (shown in the first row) varies from 3 to 10. For each size, we generated all possible subsets of that size taking the elements from our small Universe. The total number of subsets generated in each case is shown in the second row (${}^n C_r$). After hashing all the keys in a subset, the successful search length is computed. The average of this search length over all the subsets is computed and shown in the next six rows corresponding to each method of the hashing

function used.

Table 5. Comparison based on Percentage of Perfect distributions

| Subset Size (r) | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------------------------|-------|-------|-------|-------|-------|--------|--------|--------|
| No. of Subsets (nC_r) | 1140 | 4845 | 15504 | 38760 | 77520 | 125970 | 167960 | 184756 |
| Mid-Square method | 65.96 | 42.13 | 21.96 | 8.79 | 2.41 | 0.34 | 0 | 0 |
| Modulo method | 84.21 | 69.35 | 52.01 | 34.67 | 19.81 | 9.15 | 3.05 | 0.55 |
| Multipli- cation method | 67.63 | 50.09 | 33.69 | 20.52 | 10.79 | 4.65 | 1.46 | 0.25 |
| Decimal to base 11 | 81.40 | 64.75 | 46.31 | 29.05 | 15.40 | 6.49 | 1.94 | 0.31 |
| Decimal to base 15 | 59.65 | 36.12 | 18.78 | 8.31 | 3.04 | 0.88 | 0.18 | 0.02 |
| Uni- versal Hashing | 78.77 | 60.56 | 41.49 | 24.75 | 12.43 | 4.96 | 1.41 | 0.22 |

Tables 5 and 6 show the percentage of distributions that are perfect(P) and worst(W) respectively for different hashing functions. For example, with $r = 3$, we have 1140 possible subsets. A distribution is perfect if the resulting successful search length is precisely 1.0 or if there were no collisions. With the mid-square method, out of 1140 subsets, 752 distributions gave a search length of 1.0. Thus, we get $(752/1140) * 100 = 65.96\%$ of perfect distributions for mid-square method depicted in Table 5. Conversely, out of 1140 subsets, 22 subsets result in all collisions which gives us $(22/1140) * 100 = 1.93\%$ of the worst

Table 6. Comparison based on Percentage of worst-case distributions

| Subset Size (r) | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|------------------------------|------|------|-------|-------|-------|--------|--------|--------|
| No. of Subsets (nC_r) | 1140 | 4845 | 15504 | 38760 | 77520 | 125970 | 167960 | 184756 |
| Mid-Square method | 1.93 | 0.31 | 0.04 | 0 | 0 | 0 | 0 | 0 |
| Modulo method | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Multiplication method | 2.68 | 0.91 | 0.33 | 0.11 | 0.03 | 0.01 | 0.001 | 0.0001 |
| Decimal to base 11 | 0.18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Decimal to base 15 | 3.51 | 0.62 | 0.08 | 0.01 | 0 | 0 | 0 | 0 |
| Uni-versal Hashing | 0.37 | 0.02 | 0 | 0 | 0 | 0 | 0 | 0 |

distribution for the mid-square method depicted in Table 6. Similarly, the division-remainder method provides 960 perfect distributions and 0 worst-case distributions for $r = 3$. Our experiments show that the division-remainder method gives the highest percentage of perfect distributions and the least percentage of the worst distributions for all subset sizes when compared to other methods.

While we were inclined to believe that functions from H_1 class would provide the best results when compared to other hashing techniques, it was surprising to see that, contrary to our belief, the average search performance of functions from H_1 class was not as good as division-remainder method. The performance of functions from H_1 was closely behind that of the remainder method and radix transformation to base 11. Even the number of subsets with perfect distribution was highest with the remainder method.

5 Conclusions

In this paper, we have presented the performance of some popular hashing techniques over all the possible subsets of a Universe. For the Universe chosen, the remainder method distributed the subsets of different sizes with fewer collisions resulting in a smaller average successful search. It was closely followed by the radix transformation method to base 11. The division-remainder method also gave the highest percentage of perfect distributions followed by radix transformation to base 11 for all subset sizes r . A smaller value for the prime ‘ p ’ improved the performance of the Universal Hashing function making it better than the radix transformation but it did not do better than the division-remainder method which was contrary to our belief.

Acknowledgments

This research was supported by Visvesvaraya Technological University, Jnana Sangama, Belagavi.

Conflict of interest

The authors declare no potential conflict of interests.

References

- [1] V. Y. Lum, P. S. Yuen, and M. Dodd, “Key-to-address transform techniques: A fundamental performance study on large existing formatted files,” *Communications of the ACM*, vol. 14, no. 4, pp. 228–239, 1971.
- [2] J. L. Carter and M. N. Wegman, “Universal classes of hash functions,” in *Proceedings of the ninth annual ACM symposium on Theory of computing*, 1977, pp. 106–112.
- [3] R. Deutscher, P. G. Sorenson, and J. P. Tremblay, “Distribution-dependent hashing functions and their characteristics,” in *Proceedings of the 1975 ACM SIGMOD international conference on Management of data*, 1975, pp. 224–236.
- [4] W. W. Peterson, “Addressing for random-access storage,” *IBM journal of Research and Development*, vol. 1, no. 2, pp. 130–146, 1957.
- [5] W. Buchholz, “File organization and addressing,” *IBM Systems Journal*, vol. 2, no. 2, pp. 86–111, 1963.
- [6] M. D. Mc Ilroy, “A variant method of file searching,” *Communications of the ACM*, vol. 6, no. 3, p. 101, 1963.
- [7] D. C. Roberts, “File organization techniques,” in *Advances in Computers*. Elsevier, 1972, vol. 12, pp. 115–174.
- [8] G. Schay and N. Raver, “A method for key-to-address transformation,” *IBM Journal of research and development*, vol. 7, no. 2, pp. 121–126, 1963.
- [9] G. Schay Jr and W. G. Spruth, “Analysis of a file addressing method,” *Communications of the ACM*, vol. 5, no. 8, pp. 459–462, 1962.
- [10] M. V. Ramakrishna, “Hashing practice: analysis of hashing and universal hashing,” *ACM SIGMOD Record*, vol. 17, no. 3, pp. 191–199, 1988.
- [11] V. Y. Lum, “General performance analysis of key-to-address trans-

- formation methods using an abstract file concept,” *Communications of the ACM*, vol. 16, no. 10, pp. 603–612, 1973.
- [12] P. Sorenson, J. Tremblay, and R. Deutscher, “Key-to-address transformation techniques,” *INFOR: Information Systems and Operational Research*, vol. 16, no. 1, pp. 1–34, 1978.
- [13] J. Von Neumann, “13. various techniques used in connection with random digits,” *Appl. Math Ser*, vol. 12, no. 36-38, p. 3, 1951.

G. M. Sridevi, M. V. Ramakrishna,
D. V. Ashoka

Received October 26, 2022
Accepted January 25, 2023

G. M. Sridevi

ORCID: <https://orcid.org/0000-0003-3864-9983>

Research Scholar - Visvesvaraya Technological University (VTU),
Dayananda Sagar Academy of Technology and Management,
Department of Information Science and Engineering,
Udayapura, Kanakapura Road, Bengaluru-560082, India.
E-mail: sridevi.gereen87@gmail.com

M. V. Ramakrishna

ORCID: <https://orcid.org/0000-0001-7058-7562>

SJB Institute of Technology, Department of Information Science and Engineering,
BGS Health and Education City, Dr. Vishnuvardhan Road,
Bengaluru-560060, India.
E-mail: mvrma@yahoo.com

D. V. Ashoka

ORCID: <https://orcid.org/0000-0003-1326-2387>

JSS Academy of Technical Education
Department of Information Science and Engineering,
Dr. Vishnuvardhan Road,
Bengaluru-560060, India.
E-mail: dr.dvashoka@gmail.com

Deep learning-based software for detecting population density of Antarctic birds

Sinan Uğuz

Abstract

Monitoring populations of bird species living in Antarctica with current technologies is critical to the future of habitats on the continent. Studies of bird species living in Antarctica are limited due to climate, challenging geographic conditions, and transportation and logistical constraints. The goal of this study is to develop Deep Learning-based software to determine the population densities of Antarctic penguins and endangered albatrosses. Images of penguins and albatrosses obtained from internet sources were labeled using the segmentation technique. For this purpose, 4144 labeled data were trained with five different convolutional neural network architectures TOOD, YOLOv3, YOLOF, Mask R-CNN, and Sparse R-CNN. The performance of the obtained models was measured using the average precision (AP) metric. The experimental results show that the TOOD-ResNet50 model with 0.73 AP^{50} detects the Antarctic birds adequately compared to the other models. At the end of the study, a software was developed to detect penguins and albatrosses in real time.

Keywords: Deep Learning, Antarctic birds, Remote sensing, Population estimation, Convolutional neural network.

MSC 2020: 68T07, 68T45, 62P12.

1 Introduction

As a result of global warming in the world, the melting of sea ice has negatively affected the feeding ecosystems of populations of bird species such as penguins, albatrosses, skuas, cormorants, petrels, and Arctic

terns living in Antarctica. Polar regions have difficult conditions for scientific studies due to climatic difficulties, geographic barriers, transportation and logistical constraints, and ecological limitations. Despite these difficult conditions, knowledge of bird species populations is very important for the region's habitat. Because many seabird species are closely spaced in colonies or breed in large numbers in inaccessible areas, population estimates of these birds are difficult [1]. Penguins are among the most important bird species living in Antarctica. It is estimated that the total number of penguin pairs breeding in Antarctica, where there are 18 different species of penguins, is about 20 million. Although penguins cover a large geographic area, they are concentrated in coastal areas with less harsh climates [2]. Another important bird species that lives in Antarctica is the albatross. All but seven of the world's 22 albatross species are threatened with extinction. Every year, tens of thousands of albatrosses die because they get caught in large nets while fishing behind fishing boats [3].

Determining the population densities of both bird species will provide information on the breeding behavior of these birds and make an important contribution to the protection of the Antarctic habitat. To this end, researchers have used various remote sensing technologies such as satellite imagery and computer vision techniques. In [4], satellite imagery with a resolution of 10 m was used to survey the penguin population and determine their reproductive behavior. The same authors used commercial satellite imagery at 30 cm resolution for albatross colonies in their 2017 study [5]. In [6] were identified breeding colonies of the bird species *Thalassoica Antarctica* using images in six spectral bands from Landsat-8. In another study using satellite imagery, in [7] was used a convolutional neural network (CNN) called U-Net for albatross colony detection. According to the researchers, the main limitations of the research are noise in satellite images, cloud cover, and complex background images.

The main disadvantage of studies based on satellite imagery is that the spatial resolution of satellite imagery is not high enough to clearly detect birds. While it is possible with military satellites to detect objects in an area of 10 cm^2 , these satellites cannot be used by researchers because they are not open to the public [8]. Another problem that arises

when using satellite imagery is that the number of birds detected in the colony cannot be accurately estimated. In particular, birds approaching each other for warmth in very cold time periods cause the population to be miscalculated. Satellite imagery can be of great use in determining the location of new colonies. However, various solutions need to be developed to determine the population density at that location.

The most successful methods for bird colony population detection are Deep Learning, which has recently achieved great success in all fields, and image processing techniques [9]. In image processing, researchers must manually extract features from images and apply various machine-learning techniques to solve each problem. This is a very long process. Also, in order to define an object, all the features must be defined by the researchers; for example, to create a penguin's feature map using image processing, researchers must define features such as its beak, arm, and leg. However, in the Deep Learning-based approach, feature extraction is done automatically using Deep Learning architectures [10]. In [9], images of birds near lakes and on agricultural land were collected by unmanned aerial vehicles (UAVs). This dataset includes popular CNN models, Faster Region-based CNN (Faster R-CNN), Region-based Fully Convolutional Network (R-FCN), Single Shot MultiBox Detector (SSD), RetinaNet, and You Only Look Once (YOLO). The best accuracy was achieved with Faster R-CNN and the fastest result production with YOLO.

In the other study [11], a penguin dataset was created. The unique feature of this dataset is the use of the dot annotations technique. That is, each penguin in the dataset is represented by dots. The researchers proposed a CNN model based on the VGG16 classification architecture. As a result of the study, the population density of penguins was determined by creating kernel density maps. In [12], it was aimed to detect Black-browed albatross and Southern Rockhopper penguin colonies using drone imagery. Approximately 37.000 data labels were made. Training was conducted using the RetinaNet architecture. The mAP for the albatross model was 97.66% and the mAP for the penguin model was 87.16%. In another study, the Penguin Counting model de-

veloped by¹ is still in the project phase. The main objective of the project is to count the penguins in the images captured by the camera traps deployed in Antarctica. For this purpose, the Microsoft Azure platform and the PyTorch framework were used.

Though several approaches were presented for population prediction of Antarctica birds, there exist some significant challenges in it. Some Antarctic seabirds gather at the sea surface, while albatrosses and penguins are not seen in groups in the sea. For this reason, it is necessary to estimate the population of albatrosses and penguins from land rather than from the surface of the sea. For land imaging, the resolution of commercial satellite images is insufficient. Another problem is the difficult geographic conditions when taking images with a drone. On the other hand, population estimates can be made with deep learning techniques using camera images placed in specific regions.

The aim of this study is to develop a Deep Learning-based remote sensing software for penguin and albatross colony prediction. The specific contributions of this study are as follows: (1) A new dataset consisting of 4144 penguin and albatross images is collected to train deep learning models. (2) In contrast to similar studies, a segmentation-based annotation technique was used here. (3) By using various state-of-the-art object detection models, the best AP value of 73% in prediction success was achieved.

2 Material and Methods

The general processes of the project are shown in the diagram in Fig.1. According to this diagram, the first phase of the project is the data preprocessing process. The images of penguins and albatrosses were obtained from the Internet using open-source images and videos. The penguins and albatrosses in the images are individually labeled based on segmentation. In the next step, model trainings were conducted using innovative CNN models. As a result of all trainings, a performance evaluation of the model was performed. The model with the best performance was used in the developed graphical user interface

¹<https://penguin-counting-app.azurewebsites.net>

(GUI) software.

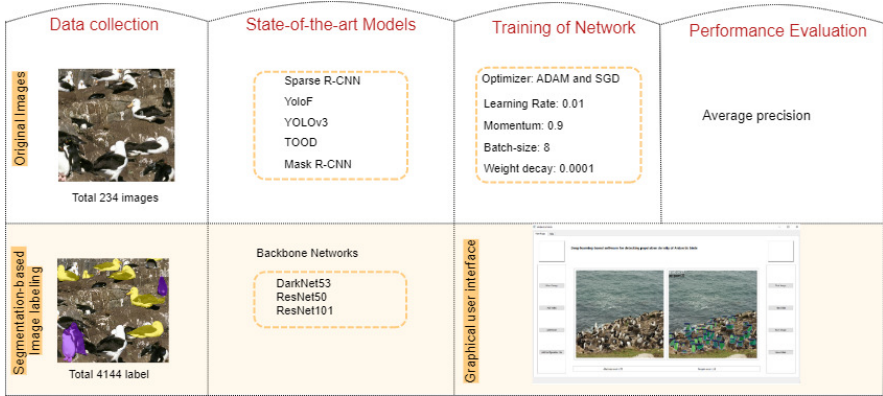


Figure 1. General operating diagram for experimental studies

2.1 Data collection and preparation

To create the dataset, images of penguins and albatrosses were obtained from the Internet via open-source images and videos. The VLC player program was used for the images obtained from the videos. With the help of this program, one image per 20 frames was obtained from each of the video images of albatross and penguin colonies. The common feature of the obtained images is that they are images taken from a high angle and not in the form of a drone or satellite image. This is because the software we developed is designed to recognize camera images taken from a specific height and angle. In addition, birds in colonies were preferred for all images. Penguin and albatross images in the dataset were labeled based on segmentation via the Supervisely platform².

An example of labeling is shown in Fig.2. It can be seen that the labeled penguins in the image are colored purple. On the Supervisely platform, segmentation by rectangles and polygons can be done using the tools on the left. Each record stores the coordinate information of the bird in that image. In this study, a total of 4.144 labels were

²<https://supervise.ly>

created. The fact that the labeling process is based on segmentation means that more time is required. It took the project team about 50 hours to label 4.144 images. The distribution of the number of images and labels in the dataset is shown in Table 1.

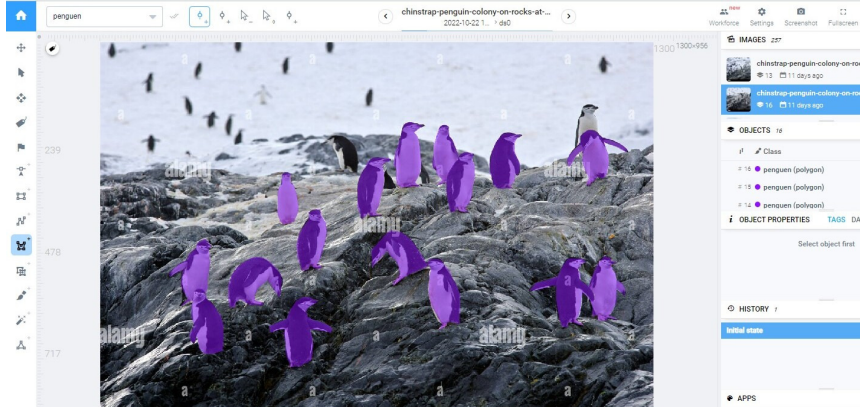


Figure 2. Segmentation-based labeling

Table 1. Numerical distribution of classes in the dataset

| Classes | Image Count | Label Count | Size |
|-----------|-------------|-------------|------|
| Penguin | 128 | 2072 | |
| Albatross | 106 | 2072 | |
| Total | 234 | 4144 | 55MB |

2.2 Implementation details of CNN models

CNN architectures first appeared in 1998 with LeNET-5 [13], then came AlexNet in 2012 [14], and later several other CNN architectures were developed that provided successful solutions to artificial intelligence problems. With these architectures, numerous convolutional operations, pooling operations, and different types of activation functions were tried.

In this study, experiments were conducted using state-of-the-art CNN architectures to detect population densities of Antarctic birds.

The experiments in this study were conducted using the MMDetection³, a state-of-the-art PyTorch-based modular object detection library released by the OpenMMLab project⁴. The toolbox directly supports state-of-the-arts deep learning frameworks. Sparse R-CNN, YOLOF, TOOD, YOLOv3, and Mask R-CNN frameworks were preferred in this study. For these frameworks, ResNet-50, ResNet-101, and DarkNet-53 architectures were chosen as backbones.

In this study, 90% of the enlarged data was randomly selected as the training set and the remaining 10% as the test set. Of these, 10% of the training set was selected as the validation set. All models were configured to use the base models, and the training hyperparameters were predefined and remained static for all experiments. For training the state-of-the-art models, the number of iterations was started with 100 and gradually increased. In the selection of parameters, the preferred values in some literature studies [15]–[18] were used. Accordingly, Adam and SGD were used as learning algorithms. The learning rate was initially set to 0.01 and the mini-batch size was set to 8. The weight decay and momentum coefficients were set to 0.0001 and 0.9, respectively.

The Supervisely platform provides the ability to run MMDetection architectures. The training and testing processes on the Supervisely platform are performed on a workstation with the configurations of an Intel Xeon CPU, 16GB Nvidia Quadro RTX5000 GPU with 16GB RAM.

2.3 Performance evaluation of CNN models

Object detection problems are challenging tasks that involve both classifying the objects on the image and determining the coordinates of the images. In these problems, the labeled regions are expressed as ground truth (G), and the regions detected by the trained model are expressed as default boxes (D). As can be seen in Fig.3, the ratio of the intersection of the G and D regions to the union is defined as the intersection over the union (IoU).

³<https://github.com/open-mmlab/mmdetection>

⁴<https://github.com/open-mmlab>

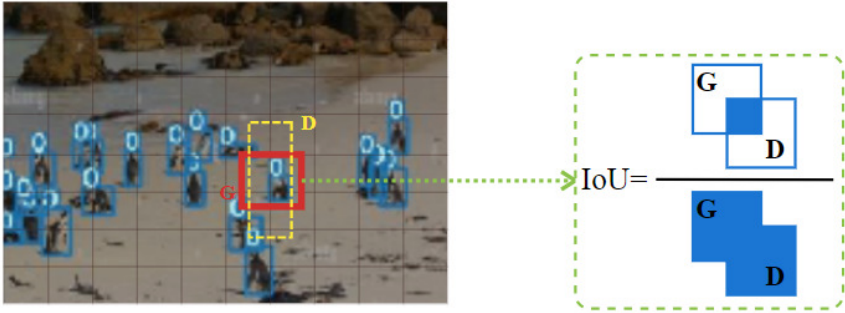


Figure 3. Ground-truth and default-box match

IoU takes values between 0 and 1, and $IoU = 1$ means that the boxes overlap [19]. The greater the overlap, the better the detection success. Average precision (AP) was used as a performance evaluation metric for the Sparse R-CNN, YOLOF, TOOD, YOLOv3, and Mask R-CNN architectures used in this study. AP given in Eq.(1) is a popular metric used for the performance of the Microsoft Common Objects in Context (COCO) dataset [20].

$$AP = \frac{1}{10}(AP^{50} + AP^{55} + AP^{60} + \dots + AP^{95}). \quad (1)$$

For the performance evaluation of the system in this study, the AP was preferred, which is commonly used in the literature [21], [22] for object detection problems. Accordingly, the AP^{50} metric expresses the calculated AP values for $IoU > 0.5$. As stated in Section 3 of this study, the best results were obtained with the AP^{50} .

3 Results and Discussion

The performance results obtained as a result of training for the state-of-the-art CNN architectures used in this study are shown in Table 2. The experiments for five different models lasted approximately 48.5 hours. In this study, the ResNet50, ResNet101, and DarkNet-53 architectures were preferred as backbone networks. The main task of the backbone networks is to classify the objects detected in the image as albatross or penguin. As can be seen in Table 2, the best results were obtained

with the AP^{50} metric. Accordingly, the TOOD_ResNet50 model was the best performing model with 73.0%. The lowest performance was achieved with the YOLOv3 architecture with 49.2%.

Table 2. Performance comparison of state-of-the-art models

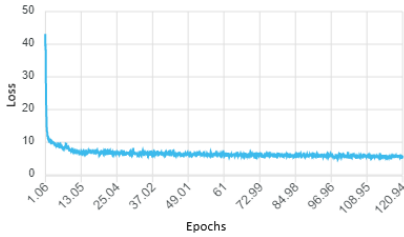
| Model | Backbone | $AP^{50:0.5:9}$ | AP^{50} | AP^{75} | AP^S | AP^M | AP^L |
|--------------|------------|-----------------|-----------|-----------|--------|--------|--------|
| Sparse R-CNN | ResNet50 | 33.3 | 54.1 | 37.0 | 16.0 | 33.2 | 50.7 |
| YOLOF | ResNet50 | 31.0 | 59.8 | 27.4 | 09.0 | 34.3 | 50.9 |
| TOOD | ResNet101 | 47.4 | 68.7 | 55.2 | 23.7 | 53.0 | 63.9 |
| TOOD | ResNet50 | 51.3 | 73.0 | 61.6 | 35.1 | 54.0 | 56.5 |
| YOLOv3 | DarkNet-53 | 22.8 | 49.2 | 17.1 | 2.8 | 24.5 | 32.6 |
| Mask R-CNN | ResNet50 | 43.0 | 60.1 | 54.0 | 29.3 | 47.1 | 54.5 |

The training loss curves are shown in Figure 4, where the coordinates represent epoch and loss values.

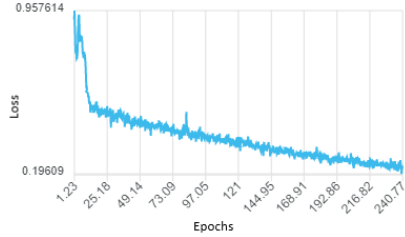
As the number of training iterations increased continuously, the loss values of all models decreased abruptly at first and then slowly. The epoch number is limited to 500 in this study. Increasing the number of epochs will help to further reduce the oscillations in the loss curve. However, increasing the iteration number would significantly increase the duration of the experiments [23]. When considering the loss curves, the Sparse R-CNN and YOLOv3 models are the models with the least oscillations. The loss curve of TOOD-ResNet50, the best prediction model, gradually converged toward 0.162 after 250 iterations, while that of YOLOv3, the lowest prediction model, converged toward 306.75. The loss values for the YOLOF, Sparse R-CNN, and Mask R-CNN models decreased to 0.215, 5.25, and 0.217, respectively. It is shown that the TOOD-ResNet50 model has a lower loss value among all models and learns the attributes of penguin and albatross images effectively.

It can be seen that the loss is fixed at a certain value in all models except the YOLOF model. In the YOLOv3 model, the loss decreased to a certain level in the first iterations, but no significant decrease was observed in the next iterations.

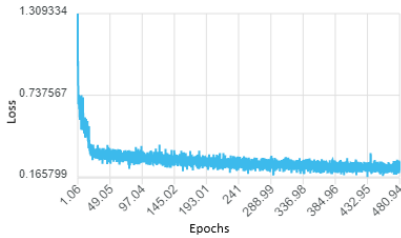
In this study, a graphical user interface (GUI) was developed for the



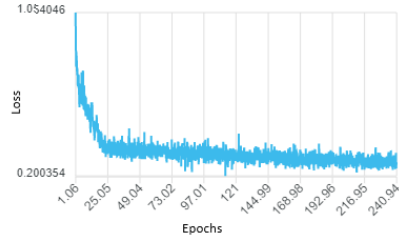
(a)



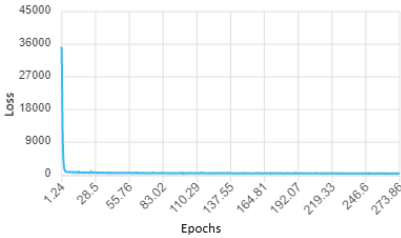
(b)



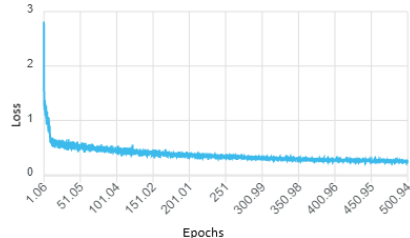
(c)



(d)



(e)



(f)

Figure 4. The loss curve of the models on the test set. (a) The loss curve of Sparse R-CNN model with ResNet50 backbone (b) The loss curve of YOLOF model with ResNet50 backbone (c) The loss curve of TOOD model with ResNet101 backbone (d) The loss curve of TOOD model with ResNet50 backbone (e) The loss curve of YOLOv3 model with DarkNet-53 backbone (f) The loss curve of Mask R-CNN model with ResNet50 backbone

state-of-the-art application using PyQt5, one of Python’s libraries. In the GUI screen shown in Figure 5, the image is first selected from the file system using the "Select Image" button. The "Add Model" button allows the selection of the file with extension .pth, which contains the weights of the best state-of-the-art model. The "Add Configuration File" button is used to load the configuration file created by the Supervisely platform. The program also detects penguins and albatrosses on video. For this purpose, the "Add Video" button must be selected. After all the selection processes are completed, the results can be viewed in real time on the right side of the screen. The obtained results can be saved in the database.

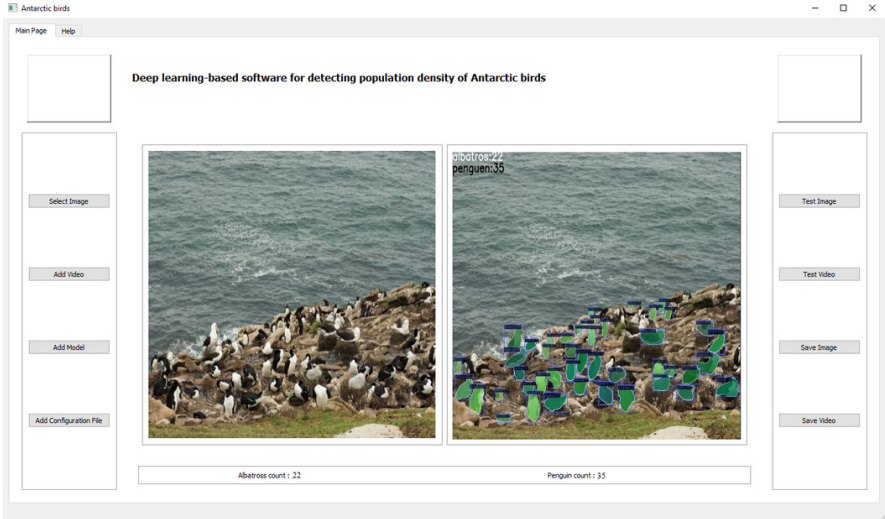


Figure 5. GUI developed for detection of Antarctic birds

The sample results obtained for the best model are shown in Figure 6.

Table 3 shows the comparison between this study and some other studies. Other than this study, there is only one study [12] that detects both penguins and albatrosses. This is one of the two studies using data from the Internet as the data source. It can be seen that the studies are divided into two areas: Image processing and CNN as the method

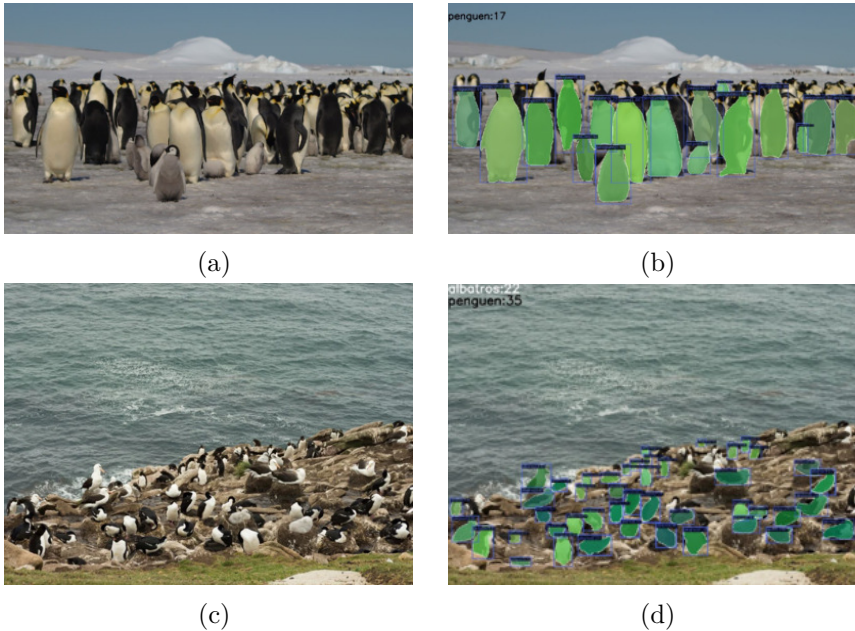


Figure 6. Results of the best model. (a) Original image of penguin colony (b) The result produced by the model (c) Original image with penguins and albatrosses (d) The result produced by the model

used. The five different CNN models used in this study have not been used in any previous study.

Apart from this study, there is only one study [11] that uses segmentation labeling. In CNN-based studies, [12] achieved a prediction success of 97.6% for black-browed albatrosses and 90% for southern rockhopper penguins using the RetinaNet architecture. [9], on the other hand, achieved a prediction success between 85% and 95.4% in his experiments. In our study, a prediction rate of up to 73% was achieved.

Detection of colony populations can be said to be a difficult problem for several reasons. One of these reasons is that birds in Antarctica stay close together to protect themselves from the cold. This makes the angle of the camera very important. For example, in the image in Figure 6a, the camera took an image from the horizontal position. For

Table 3. Results from previous studies on detection of birds in Antarctica

| Ref. | Birds | Data Source | Method | Labeling | GUI |
|------------|----------------------|-------------|---|--------------|-----|
| [4] | Penguin | Satellite | Image Pro. | - | No |
| [5] | Albatross | Satellite | Image Pro. | - | No |
| [6] | Penguin | Satellite | Image Pro. | - | No |
| [7] | Albatross | Satellite | U-NET | Rectangle | No |
| [9] | Land birds | Drone | R-CNN,SSD | Rectangle | No |
| [11] | Penguin | Internet | VGG-16 | Point | Yes |
| [12] | Penguin Albatross | Drone | RetinaNet | Rectangle | No |
| This Paper | Penguin Albatross | Internet | TOOD YOLOv3 YOLOF Faster R-CNN Sparse R-CNN | Segmentation | Yes |

this reason, some penguins left behind are not detected by the software. It is recommended that researchers wishing to conduct similar studies work with images taken from the top angle.

Complex background images are the type of problem that researchers find difficult in deep learning applications. Since the data was not collected under controlled conditions, background trees, rocks, etc. affect the prediction success. However, the results obtained in this study are encouraging for studies with other Antarctic birds such as skuas, cormorants, petrels, and arctic terns.

In this study, 4,144 labeling operations took approximately 50 hours. The fact that the labeling process is based on segmentation means that more time is spent. As the number of tags increases, the models are more successful, so datasets with more tags can be created. However, limited image resources on the Internet are a major problem. In particular, data collection with cameras in Antarctica will provide much more successful results.

4 Conclusion

The protection of Antarctic habitat is considered important everywhere in the world. However, reasons such as climate, geographic conditions, and an insufficient research budget require the use of new technological solutions in Antarctic research. In this study, Deep Learning-based software was developed for real-time detection of penguin and albatross colonies in Antarctica. The dataset obtained from internet sources was trained with five different convolutional neural network architectures: YOLOv3, YOLOF, Faster R-CNN, and Sparse R-CNN. In addition to the close proximity of albatrosses and penguins in the colony images, the complex background structure complicates the problem. However, the experimental results show that the TOOD-ResNet50 model with 0.73 AP^{50} adequately detects the birds compared to the other models.

References

- [1] G. P. Rush, L. E. Clarke, M. Stone, and M. J. Wood, “Can drones count gulls? minimal disturbance and semiautomated image processing with an unmanned aerial vehicle for colony-nesting seabirds,” *Ecology and evolution*, vol. 8, no. 24, pp. 12 322–12 334, 2018.
- [2] B. A. Surveys, “Penguins,” <https://www.bas.ac.uk/about/antarctica/wildlife/penguins/>, 2022, accessed: 2022-10-10.
- [3] B. A. Surveys, “Albatross,” <https://www.bas.ac.uk/about/antarctica/wildlife/albatross/>, 2022, accessed: 2022-10-10.
- [4] P. T. Fretwell, M. A. LaRue, P. Morin, G. L. Kooyman, B. Wiencke, N. Ratcliffe, A. J. Fox, A. H. Fleming, C. Porter, and P. N. Trathan, “An emperor penguin population estimate: the first global, synoptic survey of a species from space,” *PloS one*, vol. 7, no. 4, p. e33751, 2012.
- [5] P. T. Fretwell, P. Scofield, and R. A. Phillips, “Using super-high resolution satellite imagery to census threatened albatrosses,” *Ibis*, vol. 159, no. 3, pp. 481–490, 2017.

- [6] M. R. Schwaller, H. J. Lynch, A. Tarroux, and B. Prehn, “A continent-wide search for antarctic petrel breeding sites with satellite remote sensing,” *Remote Sensing of Environment*, vol. 210, pp. 444–451, 2018.
- [7] E. Bowler, P. T. Fretwell, G. French, and M. Mackiewicz, “Using deep learning to count albatrosses from space: Assessing results in light of ground truth uncertainty,” *Remote Sensing*, vol. 12, no. 12, p. 2026, 2020.
- [8] L. Goddijn-Murphy, N. J. O’Hanlon, N. A. James, E. A. Masden, and A. L. Bond, “Earth observation data for seabirds and their habitats: an introduction,” *Remote Sensing Applications: Society and Environment*, vol. 24, p. 100619, 2021.
- [9] S.-J. Hong, Y. Han, S.-Y. Kim, A.-Y. Lee, and G. Kim, “Application of deep-learning methods to bird detection using unmanned aerial vehicle imagery,” *Sensors*, vol. 19, no. 7, p. 1651, 2019.
- [10] S. Uğuz and N. Uysal, “Classification of olive leaf diseases using deep convolutional neural networks,” *Neural Computing and Applications*, vol. 33, no. 9, pp. 4133–4149, 2021.
- [11] C. Arteta, V. Lempitsky, and A. Zisserman, “Counting in the wild,” in *European conference on computer vision*. Springer, 2016, pp. 483–498.
- [12] M. C. Hayes, P. C. Gray, G. Harris, W. C. Sedgwick, V. D. Crawford, N. Chazal, S. Crofts, and D. W. Johnston, “Drones and deep learning produce accurate and efficient monitoring of large-scale seabird colonies,” *The Condor*, vol. 123, no. 3, p. duab022, 2021.
- [13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

- [15] J. Peng, D. Wang, X. Liao, Q. Shao, Z. Sun, H. Yue, and H. Ye, “Wild animal survey using uas imagery and deep learning: modified faster r-cnn for kiang detection in tibetan plateau,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 169, pp. 364–376, 2020.
- [16] M. Favorskaya and A. Pakhirka, “Animal species recognition in the wildlife based on muzzle and shape features using joint cnn,” *Procedia Computer Science*, vol. 159, pp. 933–942, 2019.
- [17] M. Fennell, C. Beirne, and A. C. Burton, “Use of object detection in camera trap image identification: Assessing a method to rapidly and accurately classify human and animal detections for research and application in recreation ecology,” *Global Ecology and Conservation*, vol. 35, p. e02104, 2022.
- [18] J. Brown, Y. Qiao, C. Clark, S. Lomax, K. Rafique, and S. Sukkarieh, “Automated aerial animal detection when spatial resolution conditions are varied,” *Computers and Electronics in Agriculture*, vol. 193, p. 106689, 2022.
- [19] W. Li, P. Chen, B. Wang, and C. Xie, “Automatic localization and count of agricultural crop pests based on an improved deep learning pipeline,” *Scientific reports*, vol. 9, no. 1, pp. 1–11, 2019.
- [20] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European conference on computer vision*. Springer, 2014, pp. 740–755.
- [21] K. Tong, Y. Wu, and F. Zhou, “Recent advances in small object detection based on deep learning: A review,” *Image and Vision Computing*, vol. 97, p. 103910, 2020.
- [22] Q.-J. Wang, S.-Y. Zhang, S.-F. Dong, G.-C. Zhang, J. Yang, R. Li, and H.-Q. Wang, “Pest24: A large-scale very small object data set of agricultural pests for multi-target detection,” *Computers and Electronics in Agriculture*, vol. 175, p. 105585, 2020.

- [23] S. Uğuz, G. Şikaroğlu, and A. Yağız, “Disease detection and physical disorders classification for citrus fruit images using convolutional neural network,” *Journal of Food Measurement and Characterization*, pp. 1–10, 2022.

Sinan UĞUZ

Received February 3, 2023

Revised April 14, 2023

Accepted April 25, 2023

Sinan UĞUZ

ORCID: <https://orcid.org/0000-0003-4397-6196>

Department of Computer Engineering

Faculty of Technology

Isparta University of Applied Sciences

E14 Block (3. Floor) West Campus 32260 Isparta/TURKEY

E-mail: sinanuguz@isparta.edu.tr

Correcting Instruction Expression Logic Errors with GenExp: A Genetic Programming Solution

Mohammed Bekkouche

Abstract

Correcting logical errors in a program is not simple even with the availability of an error locating tool. In this article, we introduce GenExp, a genetic programming approach to automate the task of repairing instruction expressions from logical errors. Starting from an error location specified by the programmer, we search for a replacement instruction that passes all test cases. Specifically, we generate expressions that will substitute the selected instruction expression until we obtain one that corrects the input program. The search space is exponentially large, making exhaustive methods inefficient. Therefore, we utilize a genetic programming meta-heuristic that organizes the search process into stages, with each stage producing a group of individuals. The results showed that our approach can find at least one plausible patch for almost all cases considered in experiments and outperforms a notable state-of-the-art error repair approach like ASTOR. Although our tool is slower than ASTOR, it provides greater precision in detecting plausible repairs, making it a suitable option for users who prioritize accuracy over speed.

Keywords: error correction, instruction expression, plausible patch, crossover, mutation.

MSC 2020: 68W50, 68T05, 68T20, 68N30.

1 Introduction

Developers spend a significant amount of their engineering time and effort in finding and fixing bugs in their code [1]. Even after locating errors, program debugging remains a challenging task. Logical errors

are among the most common bugs in programming. The programmer, by correcting these errors, in a totally manual way, can insert others which make matters worse. For this, we need to use an automatic or at least a semi-automatic approach for error correction. Many researchers have dealt with this recent subject in view of its importance in the process of software development. Among the early proposed approaches, we can cite [2] and [3], which were the pioneers in utilizing the genetic programming (GP) technique to evolve the erroneous program until finding an individual that resolves the errors. The problem which can oppose the use of this technique in the context of error correction is that the number of individuals of generated programs can become important before arriving at a solution. To address this, we propose an approach called GenExp in this paper. GenExp leverages knowledge about the program being repaired to find plausible patches for faulty instruction expressions.

The automatic phase of this approach follows a series of steps that form the structure of the genetic algorithm:

1. create the initial population,
2. evaluate each candidate using a fitness function,
3. evolve new generation,
4. evaluate candidates,
5. if a candidate that corrects the program, according to the test cases considered as input, is found, the process ends. Otherwise, we need to repeat steps 3 and 4.

The goal is to select the best individuals and form other individuals that combine elements from the previous stage. The process starts with a set of individuals randomly generated and enriched with expressions from the program under repair. To create the initial population, our basic algorithm proposes randomly generating expressions (candidates or patches) from a set of variables, constants, and operators chosen by the user. We propose that these sets be defined automatically based on the program to be repaired and the erroneous instruction. The process of GP from this initial population can lead to an explosion in the

number of generated individuals before reaching a plausible patch. To reduce the effect of this problem, we adopt the idea presented by the GenProg approach [3] which suggests that most defects can be repaired by adopting existing code from elsewhere in the program. This is referred to as "repair ingredients" or "patch ingredients". In GenProg, a candidate patch is synthesized from instructions taken as is elsewhere in the application. Our idea which is close to that of CARDUMEN [4] is to reuse code instruction expressions as ingredients rather than reusing raw, unmodified code elements (such as raw instructions in GenProg). For this, we add to the initial population the expressions of instructions of the program which allow it, by replacing the erroneous expression, to compile correctly. We calculate the degree of correctness of the candidates according to the input test cases and this will be our fitness function. To evolve the current generation into a new generation, we first select the best individuals in terms of their fitness function value to survive in the new population. Then, we apply a set of mutations and crossovers to individuals in the current population to generate the other individuals of the new generation. The goal is to improve the expressions. We continue evolving new generations until we obtain a plausible patch, that is to say a patch that produces correct outputs for all inputs in the test suite [5].

We evaluated the capacity of GenExp to discover plausible patches (test-suite adequate patches). Our tool was implemented in Java to repair Java programs. To do the experiments, we constructed a set of erroneous academic programs. Each of these programs contains a single error to be corrected. We compared our implementation with three approaches from the ASTOR tool [6], an open-source framework for repairing buggy Java program. The results showed that our tool successfully found a plausible patch for nearly all the programs under repair and outperformed the ASTOR approaches, even when united¹. However, our tool is slower than other tools. Nonetheless, thanks to its higher precision in detecting plausible repairs, GenExp can be a

¹To harness the collective power of ASTOR's approaches, it is sufficient for at least one of them to discover a plausible patch in order to claim that ASTOR has fixed the program.

suitable option for users who prioritize accuracy² over speed.

This article is structured as follows. We illustrate our approach on an example in Section 2. Section 3 deals with the presentation of relevant works that have a strong interest in the context of error correction. The details of the approach proposed and results are exposed respectively in Sections 4 and 5. The conclusion is presented in Section 6.

2 Illustrative example

We consider an erroneous version of SquareRoot program (Listing 1). The SquareRoot correct program finds the square root integer part of an integer value greater than or equal to 0. If we assume that the locating errors phase is complete and so on we get a set of suspicious instructions. We obtain this set thanks to an error localization tool. Now comes the role of the programmer to analyse the produced set and choose the instruction that should be corrected using GP. In fact, we aim to modify the expression of this instruction until the program becomes correct. The programmer after checking realizes that the instruction in the line 10 ($res = i + 1$) should be corrected. The error correction tool then evolves the program by just changing the expression of this instruction until the correct program is obtained. The tool changes the expression of the selected instruction, whether it is an assignment or a branching. A timeout is specified to indicate that the tool has not found solutions. In such cases, the programmer will review the selected instruction and choose another one.

Listing 1. The SquareRoot program with an error

```
1 public class SquareRoot {
2     public static int squareRoot (int val) {
3         int i = 1;
4         int v = 0;
5         int res;
6         while (v < val){
7             v = v + 2*i + 1;
```

²One program repair tool can be considered more accurate than another when it consistently achieves a higher success rate in finding plausible patches when benchmarking erroneous programs.

```

8             i = i + 1;
9         }
10        res = i+1; //error: should be res = i-1
11        return res;
12    }
13 }
```

Here we will explain how to automatically search for a plausible patch for a specific suspicious instruction. First, the programmer determines which operators, variables, and constants to take, these sets allow to build expressions that replace "i+1" in "res = i+1". This stage is very important, wrong selection may result in making the correct expression inaccessible. We suppose that these sets are $\{+, -, *\}$ (for operators) and $\{val, i, v\}$ (for variables) and integers in the interval $[0, 10]$ (for constants). Here, these sets are defined manually. However, in the subsequent Subsection 4.1, we will explore improvements that enable the automatic construction of these sets from the program under repair. So the algorithm continues by applying the following genetic processes from an initial population of individuals: selection, crossover, and mutation. To establish a preference order among individuals, a fitness function is utilized, which calculates the number of successful executions in the modified program (of course test cases should be used with at least one counterexample). An expression that passes all test cases is considered a potential correction (plausible patch). Table 1 prints the output expected for each test case input used.

To illustrate how the selection of individuals works, we display the computed outputs for the same inputs while considering the expression "(v-val)" instead of "i+1" in the statement at line 10 (Table 1, column 3). The executed program is depicted in Listing 2.

Listing 2. SquareRoot program, considering the expression "(v-val)" in the statement at line 10.

```

1  public class SquareRootv2 {
2      public static int squareRoot (int val) {
3          int i = 1;
4          int v = 0;
5          int res;
6          while (v < val){
7              v = v + 2*i + 1;
8              i = i + 1;
9          }
}
```

```

10         res = (v-val);
11     }
12 }
```

Table 1. Computed and expected outputs for SquareRoot and SquareRootv2

| Input (val) | Output for SquareRoot | Output for SquareRootv2 | Expected output |
|-------------|-----------------------|-------------------------|-----------------|
| 16 | 6 | 8 | 4 |
| 20 | 6 | 4 | 4 |
| 25 | 7 | 10 | 5 |
| 30 | 7 | 5 | 5 |
| 36 | 8 | 12 | 6 |
| 40 | 8 | 8 | 6 |
| 49 | 9 | 14 | 7 |
| 60 | 9 | 3 | 7 |
| 64 | 10 | 16 | 8 |
| 80 | 10 | 0 | 8 |
| 81 | 11 | 18 | 9 |
| 90 | 11 | 9 | 9 |
| 100 | 12 | 20 | 10 |

This result shows that there are only two successful test cases (input = {20,90}), while the others do not produce the expected output. Based on this, we can conclude that "(v-val)" is not a plausible patch.

The initial population consists of a set of candidates (expressions) that were randomly generated (6, 8, 1, v, v + 2, 2 * val, i * v, i * val, ...). Each candidate in this set will be evaluated using the same method as illustrated previously. In order to generate a new generation, we apply genetic operators to candidates. For example:

- i*val and 1 crossover to i*1
- i*1 mutates to i-1

The iterative process of generating new generations continues until a termination condition is met:

1. An expression that successfully executes all test cases (a plausible patch) has been found;
2. The specified timeout has elapsed.

In our case, the first condition is checked because the expression "i-1" permits obtaining all expected outputs. The programmer analyses this result to decide if it should be considered definitive (a patch is considered definitive when the programmer finds it to be correct). This patch is a correction because it achieves not only the test cases used but also all the functionalities of SquareRoot program. The erroneous instruction will become "res = i - 1".

3 State of the art

3.1 Categories of patch generation techniques

Patch generation techniques can be categorized into four main classes [7]: heuristic-based, template-based, constraint-based, and learning-based repair techniques.

3.1.1 Heuristic-based repair techniques

Heuristic search methods use a generation-and-test methodology, building and iterating over a search space of syntactic program modifications [8]. Among these methods, GenProg [9] is regarded as a seminal work in this field. It utilizes GP to evolve variants of the program until one is found that both retains the required functionality and also avoids the defect in question. The technique takes as input a program, a set of successful positive test cases that encode the required behaviour of the program, and a failed negative test case that demonstrates a defect. GenProg defines a fitness function that measures the quality of each program variant based on the number of passing and failing test cases. The search is restricted to only produce changes based on structures in other parts of the program. Mutation and crossover genetic operations only operate on the region of the program that is relevant to the error, i.e., the parts of the program that were on the path of execution that produced the error. Arcuri and Yao [2] are the ones who proposed the idea of using GP to co-evolve programs and unit tests in order to automate the task of fixing bugs. Subsequently, Arcuri [10] developed a research prototype called JAFF, which models bug fixing as a research problem. RSRepair (Random-Search-based Repair) [11] is a tool that

repairs program defects using the same mutation operations as GenProg, but it employs random search instead of GP. Unlike GP, which requires an evaluation of the fitness of a candidate patch even if GenProg has been aware that the patch is invalid, random search has no such constraint. Furthermore, RSRepair can speed up the process of early identification of invalid patches using traditional test case prioritization techniques. SCRepair (Similar Code-based Repair) [12] uses the same mutation operators as RSRepair to modify the faulty program. Note that the insertion operator needs code from other places, which is the main difference between SCRepair and RSRepair. To select a new code to replace the existing code during the mutation process, SCRepair introduces a metric that calculates the similarity between two code fragments based on their Abstract Syntax Tree (AST). The most suitable instruction is chosen to replace the faulty location, and test cases are used to verify the elimination of the bug. In order to improve the efficiency of research via GP for program repair, Yuan and Banzhaf [13] present a new repair system based on this technique for automated repair of Java programs, called ARJA. ARJA is mainly characterized by a new patch representation, a multi-objective search, a test filtering procedure, and several strategies to reduce the search space.

3.1.2 Template-based repair Techniques

An automated program repair strategy involves generating concrete patches based on remediation templates, also known as program transformation patterns. This strategy is widely used in the literature and has been implemented in several automated program repair systems [14]. Techniques like GenProg, which rely on heuristics, can generate nonsensical patches due to the randomness of their mutation operations. To address this limitation, a new patch generation approach called Pattern-based Automatic program Repair (PAR) [15] has been proposed. PAR utilizes patch templates learned from existing human-written patches. Durieux, Cornu, Seinturier, and Monperrus [16] proposed NPEfix, a new technique to explore the search space of potential fixes for null pointer exceptions using meta-programming. NPEfix

is based on nine predefined fix templates specifically tailored for null pointer exceptions. Long, Amidon, and Rinard designed Genesis [17] to infer remediation patterns from successful human fixes for three types of defects: null pointer, out-of-bounds, and class cast. Genesis leverages the expertise and patching strategies of developers around the world to automatically fix bugs in new applications. Stack Overflow has millions of posts that could potentially be useful for fixing numerous bugs. This observation motivates Liu and Zhong [18]’s work on extracting repair patterns from Stack Overflow for automatic program repair. To find as many adequate fixes as possible for a test suite for a given bug, Martinez and Monperrus [4] created CARDUMEN, an automated repair approach based on extracted patterns that has ultra-large search space. CARDUMEN extracts code patterns from code being repaired. Liu, Koyuncu, Kim, and Bissyandé [14] implemented TBar, an automated patch generation system that incorporates a superset of patch patterns collected, summarized, organized, and labelled from literature data.

3.1.3 Constraint-based repair techniques

Typically, these approaches infer semantic constraints from the provided test cases and then generate the appropriate test suite fix by solving the resulting constraint satisfaction problem, in particular, the SMT problem [13]. Nguyen, Roychoudhury, and Chandra [19] proposed SemFix, a pioneering tool for constraint-based program repair. Given a program location to be fixed, constraints on the expression to appear in the program location are derived so that the modified program passes all the given tests. The repair constraints are generated by symbolic execution and the expression to be repaired is obtained by program synthesis. Ke, Stolee, Le Goues, and Brun [20] developed a repair method based on semantic code search called SearchRepair. The idea is to utilize semantic code search [21] on existing open-source code to find correct implementations of buggy components and methods and use the results to automatically generate fixes for defective software. This method encodes a database of human-written code fragments as SMT constraints on input/output behaviour and searches the database for

potential fixes with an input/output specification. Mehtaev, Yi, and Roychoudhury [22] present a new semantic-based repair method called DirectFix that generates the simplest fix to maximize the preservation of the program structure of the buggy program. To take into account repair simplicity in an efficient way, their method merges fault localization and repair generation into a single step. They achieve this by leveraging partial MaxSAT constraint solving and component-based program synthesis. The same authors proposed Angelix [23], a new semantic-based repair method that is adaptable to programs of similar size as heuristic-based repair tools like GenProg. They demonstrate that Angelix is more scalable than previously proposed semantic-based repair methods such as SemFix and DirectFix. The scalability of Angelix is attributed to the new lightweight repair constraint called angelic forest, which is independent of the size of the program being repaired. Furthermore, this repair method can repair multiple buggy locations that depend on each other. In [24], the authors investigate automated error repair using a reference implementation. They propose deriving a correct specification from the reference implementation and using it to guide the repair of the program to solve the test overfitting problem.

3.1.4 Learning-based repair techniques

Machine learning techniques can enhance the efficiency of automatic bug-fixing systems. Unlike the techniques in the aforementioned three categories, learning-based techniques typically require additional training data (i.e., the tuples of buggy, context, and fixed lines of code) to capture the intricate relationships between buggy and fixed code [7]. For example, [25] presents an algorithm that learns model parameters through a training set of successful human patches collected from open-source project repositories. It generates a candidate patch space, utilizes the model to rank the candidate patches in order of likely correctness, and validates the ranked patches against a suite of test cases to discover the correct patches. Tufano, Watson, Bavota, Di Penta, White, and Poshyvanik [26] extensively evaluate the ability of adopting neural machine translation (NMT) techniques to learn code fixes from real bug fixes. Furthermore, Lutellier, Pham, Pang, Li, Wei,

and Tan [27] employ ensemble learning on the combination of convolutional neural networks and a new context-aware NMT architecture to automatically fix bugs in multiple programming languages. This architecture separately represents buggy source code and its surrounding context. SequenceR [28] utilizes sequence-to-sequence learning on source code to generate one-line patches. This approach employs the copy mechanism to address the challenge of unlimited vocabulary in the source code. In [29], the authors aim to advance deep learning-based automated program repair by introducing DEAR, a deep learning-based model that facilitates fixing general bugs with changes dependent on one or more buggy statements belonging to one or multiple buggy hunks of code.

3.2 The ASTOR tool

The ASTOR tool, also known as Automatic Software Transformations for Program Repair (ASTOR) [6], automatically repairs Java programs. We utilized this tool in our experiments. It takes a buggy program, its test suite with at least one failed test case as input, and generates a patch, if possible, that fixes the bug (i.e., all test cases pass after the repair). The change point is driven by an existing spectrum-based fault localization technique called Ochiai [30]. ASTOR offers various modes, each corresponding to a different repair algorithm (their original implementations were for other programming languages). The modes we employed in our experiments are the following: JGenProg, MutRepair, and CARDUMEN.

JGenProg JGenProg is a Java implementation of GenProg [3]. It operates at the statement level, meaning it deletes, replaces, and inserts statements. The inserted code fragments through addition or replacement always originate from the same program. The replacement operator replaces one statement with another of the same type (e.g., an assignment is only replaced by another assignment). There is a risk that this Java implementation may not reflect the actual performance of the original GenProg system for C [31].

MutRepair MutRepair implements the approach of Debroy and Wong [32], which is a repair approach that applies operators from mutation testing to repair C code. MutRepair applies mutation operators to suspicious "if" condition statements and performs a single change to the condition. There are three types of mutation operators: relational (there are six interchangeable operators: $>$, $>=$, $<$, $<=$, $==$ and $!=$); logic (there are two: *OR*, *AND*); unary (there are two mutations: *negation* and *positivation*³).

CARDUMEN CARDUMEN is a repair approach based on mined templates that has an ultra-large search space [4]. It extracts code templates from the application being repaired to create a template-based search space. The repair always consists of replacing the suspected code element with an instance of the code template.

Note that MutRepair and CARDUMEN are template-based approaches, while JGenProg is heuristic-based.

4 Approach

The automatic stage of our approach (see Algorithm 1) takes as input the following: an erroneous program (*prog*), error provided from an error location tool (such as LocFaults [33], [34] or BugAssist [35], [36]) and examined by the programmer (*error*), failing and successful test cases (*tests*) with expected output for each one, and a timeout for process execution (*timeOut*). It automatically generates a set of potential corrections for *error* in *prog* that passes all tests in *tests*. We perform, for this, the GP algorithm in five steps: Initial population, Fitness function, Selection, Mutation, and Crossover.

The process begins with a set of individuals called *population* (Alg. 1, line 2). Each individual (expression) represents a possible correction randomly generated from sets that contain *variables*, *constants*, and *operators* provided from the programmer. For example, $i * val$ is an expression that can be generated for this configuration: $variables = \{\dots, i, \dots, val, \dots\}$, $operators = \{\dots, *, \dots\}$, and

³Removal of the negation operator.

$constants = \{\dots\}$.

In the next step, we compute a fitness score for each expression in *population* to measure how these expressions correct the instruction error in *prog* (Alg. 1, line 3). If *population* contains at least one individual that succeeds all test cases, the error correction algorithm ends and returns all individuals with higher fitness function (Alg. 1, lines 10–11). In the opposite case, we need to build new generations of populations using crossover and mutation genetic operators until we find one or more individuals that satisfy the above condition, and then return them to the programmer (Alg. 1, lines 4–9). Algorithm 1 can terminate with no correction found in case that the execution exceeds the specified timeout (Alg. 1, lines 12–14).

Algorithm 1 Algorithm for error correction from a suspected instruction.

Inputs: *prog*: the erroneous program; *error*: the error to correct; *tests*: a map that stores the expected output for each test case input; *timeOut*: the maximum allowed execution time.

Output: a set of potential corrections for error to be checked by the programmer.

```

1: (variables, constants, operators)  $\leftarrow$  read(); {Read variables, constants, and operators
   from the user or programmer.}
2: population  $\leftarrow$  create(variables, constants, operators, size) {Build the initial popula-
   tion.}
3: individualsFitness  $\leftarrow$  fitnessFunction(population, prog, error, tests); {Compute the
   fitness score for each individual in population.}
4: while (! existSolution(individualsFitness) and ! exceed(timeOut)) do
5:   newPopulation  $\leftarrow$  selectBestIndividuals(population, individualsFitness, ratio1);
6:   newPopulation  $\leftarrow$  newPopulation  $\cup$  mutation(population, ratio2);
7:   newPopulation  $\leftarrow$  newPopulation  $\cup$  crossover(population, ratio3);
8:   individualsFitness  $\leftarrow$  fitnessFunction(newPopulation, prog, error, tests);
9: end while
10: if existSolution(individualsFitness) then
11:   return bestIndividuals(population, individualsFitness);
12: else
13:   print("No plausible patch found, the timeout indicated is elapsed.");
14:   return null;
15: end if

```

existSolution(*individualsFitness*): tests if it exists an element in *individualsFitness* equal to the number of test cases in *tests*, this means that the current generation includes at least one potential correction.

exceed(*timeOut*): tests if the timeOut is exceeded.

bestIndividuals(*population*): selects individuals with highest fitness score in *population*.

The idea behind the composition of a new generation is to develop

new expressions that may contain a plausible patch or, at the very least, improve the fitness score of individuals from the previous generation (Alg. 1, lines 4–9). To achieve this, we apply a set of crossovers and mutations to the individuals in the current population. However, before doing so, we select the fittest individuals and include them in the next generation. A mutation modifies an individual to generate a new one. We perform mutations at a specific point, which means that a random element in the expression to be mutated will be replaced with a new element from the same category (variables, operators, or constants). A crossover combines individuals to produce a new expression which may improve their fitness score. To accomplish this, we replace a randomly selected sub-expression in the first individual with a randomly selected sub-expression from the other individual. Figure 1 [37] illustrates an example of how subtree crossover can be used to combine two tree structures to create a new one. To specify how many times these genetic operations (selection, mutation, crossover) are activated, we use ratio values for each one ($ratio_1$, $ratio_2$, and $ratio_3$).

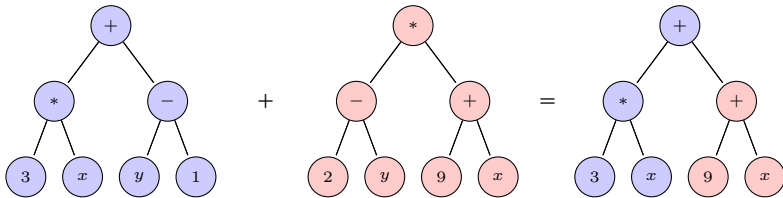


Figure 1. Example of subtree crossover operation

To calculate the fitness score for each expression in the population (*population*), we generate a program for each individual by putting it in the place of the error expression in *prog*, *prog'* is used to denote the modified program (Alg. 2, line 2). *prog'* will be compiled (Alg. 2, line 3) to obtain *compiledProg*. Next, we execute *compiledProg* on each test case input in *tests*, comparing the obtained results with the expected outputs to determine the number of successful executions (Alg. 2, lines 8–11). This count represents the fitness score for the current individual's expression and is stored in a list. Once the fitness scores for all individuals have been computed, this list, denoted by the variable *individualsFitness* in Algorithm1, will be returned (Alg. 2, line 15).

Algorithm 2 fitnessFunction: Function to compute fitness score for individuals.

Inputs: *population* : individuals ; *prog* : program to be corrected; *error*: instruction to be corrected in *prog*; *tests*: test cases.

Output : a list of fitness values.

```
1: for all individual  $\in$  population do
2:   prog'  $\leftarrow$  replaceError(prog, individual, error); {Replace the error expression in prog
   with individual, and let prog be the modified program.}
3:   compiledProg  $\leftarrow$  compile(prog'); {Compile prog'.}
4:   testCases  $\leftarrow$  getTestCases(tests); {Get test cases from tests.}
5:   cpt  $\leftarrow$  0;
6:   for all testCase  $\in$  testCases do
7:     output  $\leftarrow$  run(compiledProg,testCase); {Execute compiledProg by using
     testCase as input.}
8:     expectedOutput  $\leftarrow$  getExpectedOutput(testCase, tests); {Retrieve the expected
     output of testCase from tests.}
9:     if Equal(output,expectedOutput) then
10:       increment(cpt);
11:     end if
12:   end for
13:   add(cpt,result);
14: end for
15: return result;
```

Equal(*output*,*expectedOutput*) : tests if data in *output* is equal to that in *expectedOutput*.
increment(*cpt*) : increments *cpt*.

4.1 Improvements

Our error repair algorithm can be improved. The improvements we propose are related to the calculation of the fitness function and the initial population.

Fitness function To repair an instruction expression of a program that produces a numeric value as output, we can consider the following improvement related to the calculation of the fitness function, aiming to better assess the plausibility of a replacement expression. Specifically, we calculate the difference between the computed value of the modified program and the expected output for each test case, taking the absolute value of this result. The objective is to measure the extent to which the program variant (the replacement expression or patch) deviates from correctness for each test case. By performing the same calculation for all utilized test cases and summing the results, we obtain the fitness function value for the patch. The patch is considered plausible when the

value of its fitness function is zero. This method of fitness calculating is different from that in the basic algorithm. Indeed, we do not count the number of test cases that succeed to evaluate the correction of the program to be repaired, but we calculate how much all the test cases deviate from the expected results. Following this improvement, several functions of our algorithm (Alg. 1) need to be modified. The first one is "fitnessFunction", which should calculate the list of fitness values of individuals in order according to the method explained above. The second one is "selectBestIndividuals", which should return the list of the best individuals in *population*. An individual is considered more fit than another if its fitness value is lower (instead of higher), by flipping the fitness function, lower fitness values correspond to better individuals in terms of their proximity to the expected results. Finally, the "existSolution" and "bestIndividuals" functions should, respectively, test if there is one or more individuals with a fitness value equal to zero and, if so, return this list of individuals.

Initial population This improvement is limited to programs that need to be repaired, where assignment and return statement expressions always yield an integer, and all variables are integers. The only Boolean expressions with arithmetic are those within conditional statements. Currently, the process of defining the variables required to construct the expressions (individuals) of the initial population is manual. We aim to automate this process by collecting them from the program to be repaired, specifically from the instruction path leading to the instruction that requires repair. Variables should be either local to the function (or procedure) containing the expression to be repaired or global. Local variables should be modified after declaration. There is no requirement to modify global variables in order to utilize them. The parameters of the function (or procedure) containing the expression to be repaired are also included in the set of variables used to construct the initial population. Let's consider the example of the AbsMinus program⁴ (Listing 3). The set of variables to be used for correcting the expression in line 9 is $\{i, j, k\}$. Although *result* is declared along the

⁴The AbsMinus correct program returns the absolute value of i minus j (i and j are the inputs).

path leading to the instruction that requires repair, it is only modified at this instruction or in line 12, which is on a different path. If we had global variables, we could include them in the set of variables without any issue.

Listing 3. The AbsMinus program with an error

```

1  public class AbsMinus {
2      public static int absM (int i, int j) {
3          int result;
4          int k = 0;
5          if (i <= j) {
6              k = k+1;
7          }
8          if (k == 1 && i != j) {
9              result = i-j; //error: should be
10                 result = j-i
11          }
12         else {
13             result = i-j;
14         }
15         return result;
16     }

```

After the variables, the process of defining the set of constants should be automated. However, before explaining how we proceed with that, let us discuss an improvement that could accelerate the search for a plausible patch. This improvement involves adding instruction expressions from the program to be repaired to the initial population, provided they compile successfully⁵. We have two scenarios: either the error is in an assignment statement, in which case we only consider the expressions from assignment statements, or it is in a branching instruction, which means that only the expressions from conditional instructions are added. Applying this principle to the AbsMinus program shown in Listing 3, we would add the expressions "0", " $k + 1$ ", and " $i - j$ " to the initial population (the error was in an assignment). If the error was in a branching instruction, we would add the expressions " $(i \leq j)$ " and " $(k == 1 \ \&\& \ i \neq j)$ ". We construct the set of constants from the expressions added to the initial population: we parse each expression and whenever we encounter a constant, we add

⁵This means that if we replace the instruction expression to be repaired with each of these expressions, the program compiles without errors.

it to this set. In the case of the AbsMinus program example, the set of constants would be $\{0, 1\}$ (for an error in an assignment). If the error was in a condition, then the set would be $\{1\}$. The set of operators to be determined depends on the instruction to be repaired: if the error is in an assignment, it will contain the usual arithmetic operators; otherwise, it will consist of arithmetic operators, comparison operators, and Boolean operators.

5 Results

We used Java programming language to develop our approach and to test it in practice. This implementation, called GenExp, is an extension on OakGP, an open-source GP framework written in Java⁶. This initial implementation can only correct integer expressions, and the operators considered are: multiplication (*), division (/), addition (+), subtraction (-), logical OR (||), logical AND (&&), NOT (!), is equal to (==), is not equal to (!=), is less than (<), is greater than (>), is greater than or equal (>=), and is less than or equal (<=). Our implementation includes the improvements explained in the previous section. We have also conducted a series of experiments on a number of toy programs that we built ourselves, which require a correction in a single location. The considered error can be either in an assignment or in a conditional statement.

Table 2. Programs used the experiments

| Programs class | Nbr programs | Nbr tests |
|--------------------|--------------|-----------|
| AbsMinus | 10 | 9 |
| BSearch | 6 | 9 |
| BubbleSort | 7 | 5 |
| Gcd | 7 | 25 |
| Heron | 5 | 9261 |
| Maxmin6var (M6var) | 3 | 4096 |
| Mid | 10 | 36 |
| Minimum (Min) | 2 | 7 |
| SquareRoot | 6 | 15 |
| Tritype | 7 | 125 |
| Total | 63 | |

⁶OakGP is available at this link : <http://www.oakgp.org/>.

Table 2 shows the programs used in the experiments⁷. To explore the capabilities of our implementation on a program (column "Programs class"), we introduce an error in one of its instruction expressions and run the tool to determine if a correction can be found (i.e., a replacement expression that passes all test cases considered). The number of test cases employed is indicated in the "Nbr tests" column. This process is repeated multiple times by injecting a different error each time (see "Nbr programs" column). All developed repair programs meet the prerequisites described in Subsection 4.1.

We conducted a performance comparison between our implementation and ASTOR [6], an Automatic Software Transformations for Program Repair tool (refer to Subsection 3.2). The modes of this tool utilized in our experiments included JGenProg, MutRepair, and CARDUMEN.

Our tool repairs erroneous programs by focusing on a single instruction expression that is responsible for the error. In other words, we assume that we have achieved perfect error localization. JGenProg, MutRepair, and CARDUMEN repair the input program by initially identifying a set of suspicious instructions through their first phase of error localization. To ensure a fair comparison with ASTOR's approaches, we not only start with the same erroneous program and utilize the same test cases, but also narrow down the set of suspicious instructions generated during the error localization step to include only the injected erroneous instruction. The repair process concludes once a plausible patch is found for all tools. Additionally, there is a timeout of 5 minutes, after which the process may terminate without a solution. Testing a variant of the program to be fixed is limited to a maximum 20 milliseconds to address cases involving infinite loops.

All experiments were conducted using an Intel Core i7-3720QM processor, clocked at 2.6 GHz and equipped with 8 GB of memory. The experiments were performed on a 64-bit Linux operating system.

In our experiments, we measure the running times by running the tools multiple times on the same input. We record the time taken for each run and select the best time obtained as our metric for per-

⁷Experimental programs are available at <https://sites.google.com/prod/esi-sba.dz/error-correction-experiments>.

formance comparison. This approach allows us to capture the most efficient performance achieved by the tools.

The results of our experiments are summarized in Table 3. The "B" column lists the benchmark programs used, which correspond to the "Programms class" column in Table 2. The "Versions" column shows the erroneous versions created for each of these programs. The third, fourth, fifth, and sixth columns indicate whether CARDUMEN, JGenProg, MutRepair, and our implementation were able to find a plausible patch (✓) or not (✗) for each erroneous program considered. If a tool finds a plausible patch, we also display the time elapsed during the repair process. The "G" column displays the number of generations reached by our GP-based algorithm.

The results⁸ showed that our approach successfully finds a plausible patch for almost all cases and outperforms ASTOR's tools, even when combined. Figure 2 presents a side-by-side bar chart that compares the CARDUMEN, JGenProg, MutRepair, and GenExp tools in terms of the number of times a plausible patch is found, allowing to compare for each benchmark program.

The low number of generations observed in the majority of cases presented in Table 3 can be attributed to several factors that contribute to the efficiency of our approach. One important factor is the inclusion of interesting expressions from the program being corrected in the initial population. These expressions serve as potential building blocks for constructing plausible patches. Additionally, it is important to note that the program itself may already contain an expression that represents the plausible patch. Moreover, the construction of initial sets of variables and constants specifically derived from the program being corrected helps reduce the search space effectively. This reduction enables the genetic algorithm to focus its exploration on the most relevant and promising areas. The fitness function further enhances the convergence process by favoring the selection of superior expressions. As a result, the convergence process is accelerated, leading to the discovery of plausible patches in a shorter number of generations.

Before each new generation, GenExp calculates the fitness value for

⁸Experimental results are also available at <https://sites.google.com/prod/esi-sba.dz/error-correction-experiments>.

Genetic Programming for Logic Error Correction in Instructions

Table 3: Summary of the results of our experiments

| B | Versions | CARDUMEN | JGenProg | MutRepair | GenExp | G |
|-------------|----------------------|------------|------------|-------------|-------------|-----|
| AbsMinus | 1 | ✓(0,86s) | ✗ | ✗ | ✓(2,435s) | 3 |
| | 2 | ✗ | ✗ | ✗ | ✓(1,435s) | 1 |
| | 3 | ✗ | ✗ | ✗ | ✗ | \ |
| | 4 | ✓(0,739s) | ✗ | ✗ | ✓(1,286s) | 1 |
| | 5 | ✗ | ✗ | ✗ | ✓(1,531s) | 1 |
| | 6 | ✓(0,778s) | ✗ | ✗ | ✓(2,113s) | 2 |
| | 7 | ✓(0,769s) | ✗ | ✗ | ✓(2,356s) | 3 |
| | 8 | ✓(0,751s) | ✗ | ✗ | ✓(1,747s) | 1 |
| | WrongIf1 | ✓(2,395s) | ✓(2,454s) | ✓(1,436s) | ✓(1,289s) | 1 |
| WrongIf2 | ✓(0,793s) | ✗ | ✓(0,713) | ✓(1,129s) | 1 | |
| BSearch | 1 | ✗ | ✗ | ✗ | ✓(2,705s) | 1 |
| | 2 | ✓(2,657s) | ✗ | ✗ | ✓(2,267s) | 1 |
| | 3 | ✗ | ✗ | ✗ | ✓(5,552s) | 2 |
| | WrongIf1 | ✗ | ✗ | ✓(1,166s) | ✓(3,7s) | 3 |
| | WrongIf2 | ✗ | ✗ | ✓(0,795s) | ✗ | \ |
| WrongWhile | ✗ | ✗ | ✗ | ✓(129,586s) | 70 | |
| BubbleSort | 1 | ✗ | ✗ | ✗ | ✓(2,826s) | 1 |
| | 2 | ✗ | ✗ | ✗ | ✓(107,059s) | 112 |
| | 3 | ✓(0,855s) | ✗ | ✗ | ✓(2,338s) | 2 |
| | 4 | ✓(0,877s) | ✓(0,784s) | ✗ | ✓(0,605s) | 1 |
| | WrongIf | ✓(4,035s) | ✗ | ✓(1,43s) | ✓(1,399s) | 2 |
| | WrongWhile1 | ✗ | ✗ | ✗ | ✓(2,084s) | 3 |
| WrongWhile2 | ✓(9,428s) | ✗ | ✗ | ✓(1,015s) | 1 | |
| Gcd | 1 | ✓(1,074s) | ✗ | ✗ | ✓(33,423s) | 6 |
| | 2 | ✓(0,802s) | ✗ | ✗ | ✓(13,07s) | 2 |
| | WrongIf1 | ✓(0,861s) | ✗ | ✓(2,808s) | ✓(3,262s) | 1 |
| | WrongIf2 | ✗ | ✗ | ✗ | ✓(2,929s) | 1 |
| | WrongIf3 | ✓(0,912s) | ✗ | ✓(2,729s) | ✓(36,293s) | 4 |
| | WrongIf4 | ✓(1,137s) | ✗ | ✗ | ✓(3,176s) | 1 |
| | WrongWhile | ✗ | ✗ | ✗ | ✓(8,489s) | 1 |
| Heron | 1 | ✗ | ✓(5,669s) | ✗ | ✓(37,474s) | 1 |
| | 2 | ✗ | ✗ | ✗ | ✗ | \ |
| | WrongIf1 | ✓(23,469s) | ✗ | ✓(0,766s) | ✓(279,383s) | 6 |
| | WrongIf2 | ✗ | ✗ | ✗ | ✗ | \ |
| | WrongIf3 | ✗ | ✗ | ✓(7,67s) | ✗ | \ |
| Mévar | WrongIf1 | ✓(7,33s) | ✓(1,15s) | ✓(1,577s) | ✓(26,138s) | 1 |
| | WrongIf2 | ✗ | ✗ | ✗ | ✗ | \ |
| | WrongIf3 | ✗ | ✓(7,405s) | ✗ | ✓(257,766s) | 12 |
| Mid | 1 | ✗ | ✓(1,184s) | ✗ | ✓(1,699s) | 1 |
| | 2 | ✗ | ✓(1,151s) | ✗ | ✓(1,16s) | 1 |
| | 3 | ✗ | ✗ | ✗ | ✓(0,829s) | 1 |
| | 4 | ✗ | ✓(2,008s) | ✗ | ✓(0,902s) | 1 |
| | 5 | ✗ | ✓(1,191s) | ✗ | ✓(0,805s) | 1 |
| | WrongIf1 | ✓(3,245s) | ✗ | ✓(1,305s) | ✓(1,291s) | 1 |
| | WrongIf2 | ✓(2,613s) | ✗ | ✓(1,402s) | ✓(3,318s) | 3 |
| | WrongIf3 | ✓(2,766s) | ✗ | ✓(1,23s) | ✓(2,347s) | 2 |
| | WrongIf4 | ✓(2,322s) | ✗ | ✓(1,399s) | ✓(1,245s) | 1 |
| WrongIf5 | ✓(1,397s) | ✗ | ✓(1,154s) | ✓(1,175s) | 1 | |
| Min | WrongIf1 | ✗ | ✗ | ✓(1,491s) | ✓(2,369s) | 1 |
| | WrongWhile | ✗ | ✓(2,466s) | ✗ | ✓(2,068s) | 2 |
| SquareRoot | 1 | ✗ | ✗ | ✗ | ✓(2,862s) | 5 |
| | 2 | ✗ | ✗ | ✗ | ✓(0,625s) | 1 |
| | 3 | ✗ | ✗ | ✗ | ✓(1,349s) | 7 |
| | 4 | ✗ | ✗ | ✗ | ✓(2,495s) | 4 |
| | 5 | ✗ | ✗ | ✗ | ✓(30,085s) | 13 |
| | WrongWhile | ✓(1,6s) | ✗ | ✗ | ✓(4,518s) | 1 |
| Tritype | 1 | ✗ | ✓(0,76s) | ✗ | ✓(1,305s) | 1 |
| | WrongIf1 | ✓(6,452s) | ✓(13,711s) | ✗ | ✓(3,569s) | 1 |
| | WrongIf2 | ✗ | ✓(2,711s) | ✗ | ✓(41,755s) | 27 |
| | MultPerimetre | ✓(4,121s) | ✓(1,378s) | ✗ | ✓(3,13s) | 1 |
| | MultPerimetreWrongIf | ✓(4,282s) | ✓(4,386s) | ✗ | ✓(30,621s) | 19 |
| | Perimetre | ✓(12,084s) | ✓(1,333s) | ✗ | ✓(2,962s) | 1 |
| | Perimetre2 | ✓(3,95s) | ✓(3,482s) | ✗ | ✓(2,145s) | 1 |

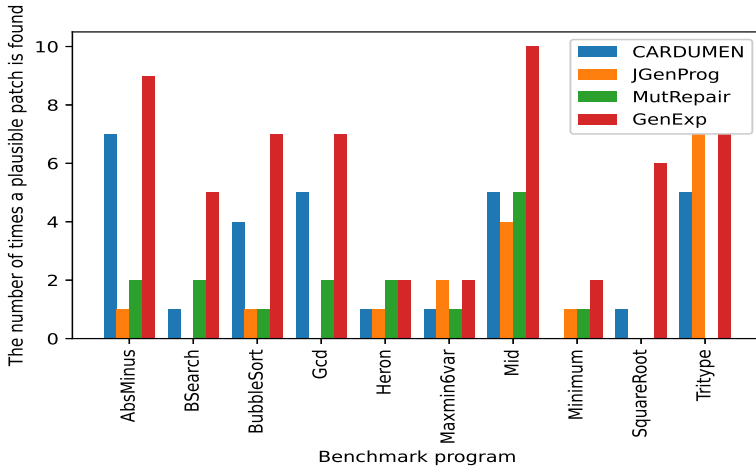


Figure 2. The number of times a plausible patch is found by CARDUMEN, JGenProg, MutRepair, and GenExp for each benchmark program

every replacement expression in the current population. This involves executing each variant of the program under repair, which corresponds to a replacement expression, on all the test cases used as input for that program. It's important to note that the running time in Table 3 does not always correlate directly with the number of generations. For instance, the WrongIf1 version of the Heron program completes 6 generations in 279.383 seconds, while the WrongIf3 version of the Maxmin6var program takes 12 generations in 257.766 seconds. On the other hand, the WrongWhile version of BSearch requires 70 generations in 129.586 seconds, and the version 2 of BubbleSort completes 122 generations in 107.059 seconds. These discrepancies can be attributed to the number of test cases used for each benchmark. The Heron and Maxmin6var benchmarks use 9261 and 4096 test cases, respectively, while the BSearch and BubbleSort benchmarks use only 9 and 5 test cases, respectively (refer to Table 2).

5.1 Limitation

To summarize and analyse the repair times of each tool used in our experiments, we have presented basic statistics such as the mean, median, standard deviation, as well as the minimum and maximum values in Table 4. We only consider the times corresponding to cases where the tools have identified a plausible patch.

Table 4. The basic statistics on the execution times obtained in Table 3

| | CARDUMEN | JGenProg | MutRepair | GenExp |
|--------------------|----------|----------|-----------|----------|
| Mean | 3,602s | 3,131s | 1,818s | 19,728s |
| Median | 2,359s | 2,008s | 1,401s | 2,369s |
| Standard deviation | 4,656 | 3,293s | 1,669s | 53,313s |
| Minimum | 0,739s | 0,76s | 0,731s | 0,605s |
| Maximum | 23,469s | 13,711s | 7,67s | 279,383s |

The results suggest that GenExp takes much longer on average than the other tools to produce repair results, with an average of 19.7 seconds and a large variation in time, ranging from 0.6 to 279.4 seconds. However, it is interesting to note that the median time of GenExp is similar to that of the other tools, indicating that it most often finds a plausible repair within a reasonable time.

Tools CARDUMEN, JGenProg, and MutRepair have much faster average times to produce repair results, with averages ranging from 1.8 to 3.6 seconds and minimum times under 1 second. These tools may be more convenient for users who need quick results, but their accuracy may be lower than that of GenExp.

Figure 3 depicts a line plot with each tool represented by a line. The various statistics (mean, median, standard deviation, minimum, and maximum) are plotted based on the tool. This enables the visualization of the evolution of execution times for each tool.

The significant difference in the range of running times, where the range of GenExp is approximately 10 times greater than that of CARDUMEN and about 30 times greater than that of MutRepair, can be attributed to two factors.

Firstly, the success rate of GenExp is much higher than that of other tools, meaning that there are many more values (times) to consider

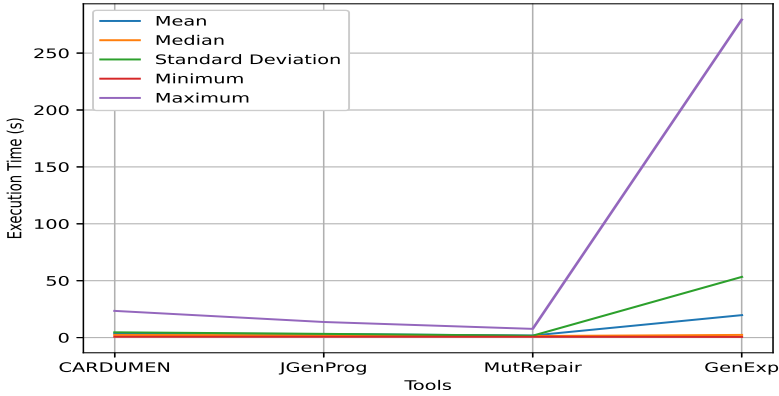


Figure 3. Evolution of execution times by tool

for calculating the standard deviation and other statistics. Specifically, there are 57 values for GenExp, 30 for CARDUMEN, 17 for JGenProg, and 16 for MutRepair. Assuming that GenExp failed to repair the following versions: BSearch (WrongWhile), BubbleSort (version 2), Gcd (version 1 and WrongIf3), Heron (WrongIf1), Maxmin6var (WrongIf3), SquareRoot (version 5), and Tritype (WrongIf2). Consequently, the standard deviation for GenExp decreases from 53,313s to 7,392s, and the success rate is now 49 out of 63. It should be noted that the success rates are 30/63 for CARDUMEN, 17/63 for JGenProg, and 16/63 for MutRepair.

Secondly, the difference can also be attributed to the search space utilized by each tool. JGenProg’s search space is limited to the program instructions to be repaired and cannot correct the error using code that is not part of the program. The MutRepair algorithm focuses on mutating the operators within ”if” conditions (see Subsection 3.2). CARDUMEN, as a template-based approach, has a wide search space but not as extensive as our approach since it is limited to templates extracted from the code to be repaired. GenExp has a broader search space compared to the ASTOR tools. While we do utilize the code under repair to construct the sets of variables, operators, and constants

necessary for building replacement expressions, we also leverage the repair program to enhance the initial population with expressions extracted from it. However, our algorithm is capable of generating new expressions that cannot be derived from the code being repaired. This explains the high success rate achieved by our tool, which in turn accounts for the difference in the range of running times observed in all experiments.

The large variation in GenExp's times may be problematic for users who need consistent, fast results. It may be preferable to use GenExp when increased accuracy is needed, even if it takes longer.

Although GenExp has the capability to repair the expression of a single instruction, it should be noted that programs containing two or more different erroneous instructions cannot be repaired using this tool. This limitation is also present in CARDUMEN, JGenProg, and MutRepair, as they too are designed to repair input programs from a single suspect instruction. Another constraint of GenExp is its inability to repair programs that require the addition or removal of instructions. Similar limitations exist in CARDUMEN and MutRepair, but JGenProg has the potential to address such errors. Additionally, JGenProg can also correct the left-hand side of an assignment, a capability not available in our tool, CARDUMEN, or MutRepair. Another notable limitation of GenExp is its inability to generate expressions where numbers and variables are not integers. In contrast, CARDUMEN, JGenProg, and MutRepair have the capability to handle variables or constants of any kind.

6 Conclusion

To correct a program, our approach is based on using GP to construct a new program without errors. We utilize the suspected instructions generated by an error locating tool. The error correction process is applied to a selected instruction, chosen by the programmer, which is potentially the source of the problem in the input program. Genetic operators are employed to evolve the expression of this instruction, aiming to generate an individual that produces the expected outputs for the given test cases. The results demonstrate the effectiveness of our ap-

proach in program correction, as our implementation successfully fixed nearly all cases used in the experiments. GenExp has outperformed ASTOR's three approaches (JGenProg, MutRepair, and CARDUMEN) in finding plausible fixes for the set of academic repair programs we have developed.

Based on the results, it seems that GenExp is a slower option compared to the other tools used in the experiments, but it is also more accurate in finding plausible repairs. This makes it a good choice for users who prioritize accuracy over speed. However, the large variation in time and the longer average time may be a concern for users who need consistent, fast results. Therefore, it is important to consider the specific needs of the user when deciding whether to use our tool or one of the faster tools.

In most cases, the programmer corrects only one instruction in the set obtained during the error locating step. However, a challenge arises when the program requires corrections in multiple instructions. Our current approach is unable to find a plausible patch in such scenarios since it focuses on a single instruction. As part of future work, we aim to address this limitation by extending our approach to handle multiple suspected instructions. One possible solution could involve running the GP process evolution on multiple suspected instructions.

Currently, GenExp possesses the ability to generate instruction expressions, encompassing both algebraic and boolean expressions with arithmetic operations. Moreover, GenExp generates integer-based expressions where numbers and variables are restricted to the integer domain. Our benchmark is carefully designed to showcase GenExp's capabilities, emphasizing that the correct expressions obtained should exclusively involve integer values for numbers and variables. Moving forward, our objective is to enhance our implementation to accommodate various expression types, enabling comprehensive testing and comparison with ASTOR tools across a wider range of expression classes.

7 Funding

No funding was received to assist with the preparation of this manuscript.

References

- [1] C. Le Goues, M. Pradel, A. Roychoudhury, and S. Chandra, “Automatic program repair,” *IEEE Software*, vol. 38, no. 4, pp. 22–27, 2021.
- [2] A. Arcuri and X. Yao, “A novel co-evolutionary approach to automatic software bug fixing,” in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 162–168.
- [3] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, “Automatically finding patches using genetic programming,” in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 364–374.
- [4] M. Martinez and M. Monperrus, “Ultra-large repair search space with automatically mined templates: The cardumen mode of astor,” in *International Symposium on Search Based Software Engineering*. Springer, 2018, pp. 65–86.
- [5] Z. Qi, F. Long, S. Achour, and M. Rinard, “An analysis of patch plausibility and correctness for generate-and-validate patch generation systems,” in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 24–36.
- [6] M. Martinez and M. Monperrus, “Astor: A program repair library for java,” in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 441–444.
- [7] Q. Zhang, Y. Zhao, W. Sun, C. Fang, Z. Wang, and L. Zhang, “Program repair: Automated vs. manual,” *arXiv preprint arXiv:2203.05166*, 2022.
- [8] C. L. Goues, M. Pradel, and A. Roychoudhury, “Automated program repair,” *Communications of the ACM*, vol. 62, no. 12, pp. 56–65, 2019.

- [9] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, “Genprog: A generic method for automatic software repair,” *Ieee transactions on software engineering*, vol. 38, no. 1, pp. 54–72, 2011.
- [10] A. Arcuri, “Evolutionary repair of faulty software,” *Applied soft computing*, vol. 11, no. 4, pp. 3494–3514, 2011.
- [11] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang, “The strength of random search on automated program repair,” in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 254–265.
- [12] T. Ji, L. Chen, X. Mao, and X. Yi, “Automated program repair by using similar code containing fix ingredients,” in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 197–202.
- [13] Y. Yuan and W. Banzhaf, “Arja: Automated repair of java programs via multi-objective genetic programming,” *IEEE Transactions on software engineering*, vol. 46, no. 10, pp. 1040–1067, 2018.
- [14] K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandé, “Tbar: Revisiting template-based automated program repair,” in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 31–42.
- [15] D. Kim, J. Nam, J. Song, and S. Kim, “Automatic patch generation learned from human-written patches,” in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 802–811.
- [16] T. Durieux, B. Cornu, L. Seinturier, and M. Monperrus, “Dynamic patch generation for null pointer exceptions using metaprogramming,” in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 349–358.
- [17] F. Long, P. Amidon, and M. Rinard, “Automatic inference of code transforms for patch generation,” in *Proceedings of the 2017 11th*

- Joint Meeting on Foundations of Software Engineering*, 2017, pp. 727–739.
- [18] X. Liu and H. Zhong, “Mining stackoverflow for program repair,” in *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2018, pp. 118–129.
- [19] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, “Semfix: Program repair via semantic analysis,” in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 772–781.
- [20] Y. Ke, K. T. Stolee, C. Le Goues, and Y. Brun, “Repairing programs with semantic code search (t),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 295–306.
- [21] K. T. Stolee, S. Elbaum, and D. Dobos, “Solving the search for source code,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 3, pp. 1–45, 2014.
- [22] S. Mechtaev, J. Yi, and A. Roychoudhury, “Directfix: Looking for simple program repairs,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 448–458.
- [23] S. Mechtaev, J. Yi, and A. Roychoudhury, “Angelix: Scalable multiline program patch synthesis via symbolic analysis,” in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 691–701.
- [24] S. Mechtaev, M.-D. Nguyen, Y. Noller, L. Grunske, and A. Roychoudhury, “Semantic program repair using a reference implementation,” in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 129–139.
- [25] F. Long and M. Rinard, “Automatic patch generation by learning correct code,” in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2016, pp. 298–312.

- [26] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, “An empirical investigation into learning bug-fixing patches in the wild via neural machine translation,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 832–837.
- [27] T. Lutellier, H. V. Pham, L. Pang, Y. Li, M. Wei, and L. Tan, “Coconut: combining context-aware neural translation models using ensemble for program repair,” in *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, 2020, pp. 101–114.
- [28] Z. Chen, S. Kommrusch, M. Tufano, L.-N. Pouchet, D. Poshyvanyk, and M. Monperrus, “Sequencer: Sequence-to-sequence learning for end-to-end program repair,” *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1943–1959, 2019.
- [29] Y. Li, S. Wang, and T. N. Nguyen, “Dear: A novel deep learning-based approach for automated program repair,” *arXiv preprint arXiv:2205.01859*, 2022.
- [30] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, “An evaluation of similarity coefficients for software fault localization,” in *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC’06)*. IEEE, 2006, pp. 39–46.
- [31] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, “Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset,” *Empirical Software Engineering*, vol. 22, no. 4, pp. 1936–1964, 2017.
- [32] V. Debroy and W. E. Wong, “Using mutation to automatically suggest fixes for faulty programs,” in *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 65–74.
- [33] M. Bekkouche, H. Collavizza, and M. Rueher, “Locfaults: A new flow-driven and constraint-based error localization approach,” in

Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015, pp. 1773–1780.

- [34] M. Bekkouche, “Combining techniques of bounded model checking and constraint programming to aid for error localization,” *Constraints*, vol. 22, no. 1, pp. 93–94, 2017.
- [35] M. Jose and R. Majumdar, “Bug-assist: assisting fault localization in ansi-c programs,” in *International conference on computer aided verification*. Springer, 2011, pp. 504–509.
- [36] M. Jose and R. Majumdar, “Cause clue clauses: error localization using maximum satisfiability,” *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 437–446, 2011.
- [37] “OakGP Genetic Programming Framework,” <http://www.oakgp.org/index.html>. [Online]. Available: <http://www.oakgp.org/index.html>

Mohammed Bekkouche

Received April 02, 2023
Revised 1 – June 16, 2023
Revised 2 – July 11, 2023
Accepted July 13, 2023

Mohammed Bekkouche

ORCID: <https://orcid.org/0000-0002-8305-0542>

LabRI-SBA Laboratory, Ecole Superieure en Informatique,

Sidi Bel Abbes, Algeria

BP. 73, Bureau de poste EL WIAM, Sidi Bel Abbes, 22016, Algeria

E-mail: m.bekkouche@esi-sba.dz

An approach to Augmented Reality Classification and an example of its usage for application development with VAK learning styles Markers

Inga Titchiev, Olesea Caftanatov, Veronica Iamandi,
Dan Talambuta, Daniela Caganovschi

Abstract

Augmented reality (AR) encompasses both technology and the experience it provides, making it applicable in real-world contexts. The field of education is particularly suited for utilizing AR techniques as a novel means of engaging with students. Various classifications of AR techniques exist, each offering remarkable potential for educational purposes. This paper presents an approach to classifying augmented reality based on the characteristics of different techniques. Additionally, we demonstrate the application of a specific type of AR technology in the development of an educational application. Furthermore, we emphasize the importance of designing augmented learning scenarios that align with the VAK learning styles, aiming to deliver personalized and immersive learning experiences. The integration of AR and VAK learning styles shows the potential for creating educational tools that are both engaging and effective.

Keywords: AR Classification, VAK learning styles markers, personalized learning, marker-based application.

MSC 2020: 97C70, 68T05.

1 Introduction

Augmented reality (AR) goes beyond mere technology; it is an immersive experience with tangible applications in real-world scenarios.

Among these scenarios, e-learning stands out as a context where AR can be leveraged as a novel means of engaging with students. In the coming years, augmented techniques hold immense potential to revolutionize education on a broader scale. By seamlessly integrating digital elements into the learning process, AR has the ability to transform traditional educational methods and provide students with unique and interactive learning experiences.

To maximize the potential of respective technology, it is beneficial to comprehend the various types of AR and their potential to enhance learning experiences. Thus, in the first part of this paper, we analyze the emergence of augmented technologies and propose a classification based on tech characteristics. In addition, we presented some ways of using the specific type in education.

The integration of augmented reality in education has the power to reshape how knowledge is imparted and absorbed, paving the way for a significant shift in the educational landscape. Additionally, the possibilities for incorporating and presenting information in innovative and interactive ways are virtually limitless. Through these technologies, various forms of multimedia content can be introduced, ranging from videos, sounds, and graphics to the inclusion of three-dimensional (3D) models. Developing learning environments with unforgettable experiences will, definitely, enhance students' perception of the real world. Thus, it will increase their engagement in the learning process, because when the physical and digital worlds collide, it changes everything. Literally, augmented reality is changing the way we see the physical world.

In the second part of this paper, we stressed the importance of integrating learning styles in augmented content. When designing augmented reality in education based on learning styles, the goal is to provide personalized and immersive learning experiences. For visual learners, AR can offer visually rich content, such as interactive 3D models or visual simulations, to enhance their comprehension and retention of information. Auditory learners can benefit from AR applications that include audio instructions, narration, or discussions to support their preferred mode of learning. Kinesthetic learners can engage with AR experiences that involve interactive elements, allowing them to physically manipulate virtual objects or participate in virtual simula-

tions. By tailoring augmented reality experiences to different learning styles, educators can create inclusive and engaging learning environments. This approach recognizes the diversity of learners' preferences and optimizes the potential for knowledge acquisition and retention.

2 The emergence of Augmented reality

Over the past five decades, the landscape of human interaction with the physical world has been profoundly transformed by the emergence of AR technology. Augmented reality revolutionizes our perception of reality by overlaying digital information onto our surroundings. It is important to note that augmented reality is often confused with virtual reality (VR), despite some shared developmental origins. However, AR and VR are distinct from one another. Unlike VR, which constructs entirely artificial environments to replace the real world with a virtual one, augmented reality seamlessly integrates technology into our actual surroundings. By directly integrating with our existing environment, augmented techniques enriches our sensory experience by introducing multimedia elements such as videos, sounds, graphics, and more. Augmented reality is a field that leaves some aspects to scientific advancements while sparking the imagination to envision its limitless potential.

As stated in [1], augmented reality is not confined to a singular device or program, but rather represents a form of interaction between humans and computers. This interaction is facilitated by a blend of technologies that superimpose computer-generated content over the real-world environment. The roots of augmented reality can be traced back to 1968 when Ivan Sutherland pioneered the first head-mounted display system. However, it was not until 1990 that the term "augmented reality" was coined by researchers Thomas Caudell and David Mizell [2].

During the early 1990s, augmented reality made a significant transition from laboratory settings to practical applications. In 1992, Louis Rosenberg pioneered the development of the groundbreaking augmented reality system known as "Virtual Fixtures" at the USAF Armstrong Labs. This system enabled military personnel to remotely control and provide guidance to machinery, serving various purposes

such as training US Air Force pilots in safer flying practices. Figure 1 showcases an illustration of this technology in action.



Figure 1. Louis Rosenberg testing Virtual Fixtures [15]

A significant milestone in the popularization of augmented reality occurred in 2000 with the development of an open-source software library called ARToolKit by Hirokazu Kato. This software package proved instrumental in enabling other developers to create AR software programs. As our reliance on mobile devices continues to grow, the demand for AR software has also increased, leading to a proliferation of applications in the field. The convergence of AR technology and mobile devices has opened up new possibilities and opportunities for incorporating augmented reality into various aspects of our lives.

In a surprising move, Esquire Magazine introduced the concept of integrating augmented reality into print media back in 2009. This innovative approach aimed to bring static pages to life by leveraging AR technology. By scanning the magazine cover, readers were able to experience an interactive augmented reality experience. Notably, the AR-enhanced magazine featured Robert Downey Jr. engaging in a virtual conversation with the readers. This bold initiative by Esquire Magazine showcased the early potential of augmented reality to transform traditional print media and create dynamic and engaging experiences for readers.

Another impressive application is Volkswagen's MARTA, short for Mobile Augmented Reality Technical Assistance. It is a remarkable adaptation of AR technology. It offers valuable support to technicians

by guiding them visually through the repair process. The MARTA app utilizes real-time images of the vehicle, superimposing outlines and labels on the parts, and provides contextually relevant information. For each step of the repair, it even indicates the specific tools required, ensuring technicians have the necessary guidance and assistance.

Furthermore, in 2013, Google introduced its Glass devices, a pair of augmented reality glasses that offer users relevant information through visual, audio, and location-based inputs. For example, when entering an airport, a user could automatically receive flight status updates.

Over the past decade, augmented reality has gained significant recognition as one of the most promising areas within computer graphics. During this time, numerous innovative applications have been developed, highlighting the growing importance of augmented reality in our daily lives. Augmented reality has been built from the ground up, and now is the opportune moment for it to truly flourish and reach new heights.

3 Types of augmented reality

Augmented reality initially made its way into public spaces several decades ago, but its true explosion has occurred in recent years, primarily due to the increased processing power available in today's smart devices. One of the greatest advantages of AR is its accessibility to ordinary users. In this section, we will explore different types of AR and discuss some of its implementations in the field of education.

As per the findings from Wilson's team [3], augmented reality can be categorized into five distinct experience types: "video launch," "3D object," "360-degree surround," "interactive game," and "information overlay". Within each of these types, there are numerous possibilities and opportunities for exploration and innovation.

According to Onirix [4], another classification of augmented reality (AR) is based on triggers that initiate the AR experience. These triggers play a role in determining the placement of augmented content. Onirix identifies three types of trigger-based AR experiences:

1. *Targets* – the anchors that connect the digital and physical

worlds; these are images or surfaces.

2. *Space* – it uses SLAM technologies to create a detailed 3D model of a real-life location. For this case, the triggers are scene recognition.
3. *Places* – this type of experience ties AR content to a specific location; for these types, the main triggers identify the user’s geolocation. The used tech can help to guide users through a certain area, etc.

Analyzing many other sources, we found out there are various types of classification, for instance, in [5], the author believes there are four types of augmented reality (marker-based, markerless, projection-based, and superimposition-based AR). As stated in [6], Triggered AR technologies include four types: Markerbased, Location-based, Dynamic Augmentation, and Complex Augmentation. However, we consider that augmented reality can be grouped into two major categories: **marker-based AR** and **markerless AR**. The remaining types may consist of variations or modifications of these two. Below we will present a classification from our point of view, with a description and an example of its application in the education field.

3.1 Marker-Based approach

Marker-based augmented reality is one of the most common types; it uses markers to trigger an augmented experience. Due to its use of image recognition, this type of AR is sometimes also called recognition-based augmented reality. This type works when a camera or app is scanned over a visual marker. Markers are referenced to merge virtual extensions with real media. A QR code or a 2D code serves as a notable example of a visual marker. Before describing the variations of this type of AR, we need to know the answer to a few key questions, like: What content do we need to display in the digital world? What factor will trigger it in the physical world? Where exactly should we put the content within the user’s view?

To provide an example, consider a scenario where we aim to display an educational animation directly onto a page of a book. In this case,

it becomes crucial to determine the exact location on the page where the user should point their camera. The device needs to recognize the specific page and identify the marker present on that page to initiate the animation. One advantage of this recognition-based technology is that as we turn the page of the book, the animation will remain fixed to the page and move accordingly. Consequently, a marker can take various forms, as long as it possesses distinct visual features. This leads us to identify two variations of marker-based augmented reality:

1. *image-based recognition* – involves the placement of a “tracker” in the form of an encoded image that provides visual cues to the AR application, indicating where to position virtual content. The image marker can take various forms, such as pictures, logos, posters, QR codes, or any type of 2D objects. The crucial factor in image-based recognition is the quality of the image used as a marker. Quality does not solely refer to resolution but also encompasses factors like contrast, color accuracy, distortion, and texture, which aid camera recognition; more about markers requirements is presented in our research [7]. This type of AR can be employed in math worksheets, where QR code images are placed near each task. By triggering the image markers, the AR app can initiate video demonstrations on how to solve the tasks. Consequently, students can check their solutions or understand their mistakes by utilizing this interactive approach after completing exercises.
2. *object-based recognition* – refers to a type of marker in augmented reality (AR) that can take the form of various 3D shapes or objects. Similar to image-based recognition, it involves recognizing objects and displaying corresponding digital content on a screen. The recognition process relies on markers, which are replaced by 3D representations of the corresponding objects. This enables users to view the objects in greater detail and from different angles. As the user rotates the marker, the 3D image also rotates accordingly. One practical application of object-based recognition in the education field is the learning of geometry shapes. Students can use smartphones or tablets to scan 3D objects and

access relevant information about them. Another notable example is the implementation by CISCO, where they utilize this type of augmented reality. CISCO scans their devices, and through an AR app, they provide users with guidance on how to use those devices (see Fig. 2).



Figure 2. AR Solution for CISCO Technical Content [16]

Marker-based augmented reality (AR) predominantly relies on mobile applications, requiring users to download AR software before they can interact with augmented content. This may lead to a perceived loss of spontaneity in marker-based experiences. However, with the advancements in camera systems and precise sensors found in popular devices such as those from Apple and Google, the landscape of AR has shifted from primarily marker-based activations to markerless AR.

Notably, Apple's ARKit and Google's ARCore are software development kits that have been released in recent years, expanding the possibilities of markerless AR. These development kits enable the creation of AR experiences without the need for physical markers. The integration of advanced technologies within these platforms has facilitated the transition towards markerless AR, providing users with more seamless and immersive AR experiences.

3.2 Markerless Augmented Reality

Markerless AR refers to an augmented reality app that can overlay digital content onto the user's environment without requiring prior knowledge or the use of physical markers. According to [8], markerless AR systems integrate 3D virtual objects seamlessly into a real-time 3D environment, enhancing the user's perception and interaction with the world. This type of AR is more versatile compared to marker-based AR since it doesn't rely on markers as visual cues to position digital content in the real world.

Instead, markerless AR utilizes various hardware components of the device, such as GPS, gyroscope, velocity meter, digital compass, and accelerometer, to gather the necessary information for determining the user's location and other details. This approach is often referred to as Position-based AR. By leveraging the collected data, users have the freedom to decide where to place virtual objects within their environment. Additionally, markerless AR allows for experimentation with different styles of 3D virtual objects, enabling users to position them anywhere, regardless of the surroundings. It even allows for objects to appear as if they are floating in the air.

Markerless AR is at the forefront of augmented reality technology, primarily leveraging a powerful tracking system known as SLAM (Simultaneous Localization and Mapping). SLAM enables real-time tracking and mapping capabilities, allowing for the placement of 3D objects in both indoor and outdoor environments without the need for physical markers. This technology has opened up new possibilities for various applications in different settings, including indoor spaces, outdoor areas, aerial environments, and underwater scenarios.

Markerless AR encompasses several types of AR technologies that do not rely on specific markers to trigger digital content. These include location-based AR, superimposition-based AR, projection-based AR, and outlining AR. Each of these techniques utilizes different methods and approaches to integrate virtual content seamlessly into the user's environment.

Location-based or position-based AR is one of the most widely implemented applications of augmented reality, because of the easy

availability of smartphones that provide the needed data regarding the user's location by using GPS, compass, gyroscope, accelerometer, etc. In most cases, this type of AR is used for navigation support; it helps travelers in their journey.

Wikitude Navigation, as cited in reference [9], has been acknowledged as the pioneering augmented reality (AR) GPS navigation system globally. It has received numerous prestigious awards and has been hailed as a “**revolutionary step forward**”, recognizing its significant contribution as a groundbreaking advancement in the domain of navigation and guidance.

Location-based augmented reality (AR) holds significant potential in the field of education, offering a multitude of applications. With the aid of AR technology, teachers are liberated from the confines of the traditional classroom setting. Augmented reality breathes life into abstract concepts, enabling educators to guide students through immersive experiences with three-dimensional objects and highlighting intriguing landmarks and artefacts along the way. A noteworthy example of such technology is the AR application known as “Magical Parks,” presently utilized in various public parks across New Zealand and Australia and equally suitable for classroom implementation. By seamlessly integrating a virtual fantasy realm into the actual park landscape, “Magical Parks” captivates children's attention, presenting them with life-sized dinosaurs and interactive bears as they navigate through the park.

Projected-based AR, is the most exciting type of AR because of its futuristic experience; it consists of a physical three-dimensional model onto which a computer image is projected to create a realistic-looking object. Projected-based AR may be one of those technologies that might eliminate the use of special gear such as Google Glass or head-mounted displays (HMDS) for experiencing augmented reality. As is obvious by its name, projected-based AR works by using projection onto objects. One of the simplest is a projection of light on surfaces. The only difference between this type of AR and normal projection is that projected-based AR can detect touch and movement and to interact with programs. Although projected-based AR is being used by industry-learning manufacturers (see Fig. 3), it also can be used as

training students by developing real-life problem-solving puzzles. The strong point of this kind of training will help students to think and react quickly to make correct decisions.



Figure 3. Light Guide Systems [17]

Superimposition-based AR recognizes an object in the physical world and enhances it in some way to provide an alternative view. Many companies have used this type of AR to help their customers feel more connected to their brand. Superimposition-based technology can recreate or replace a portion of the image or object, or even a whole thing. An exemplary illustration of this type of augmented reality implementation can be seen in the case of IKEA. In 2017, IKEA introduced an augmented reality app that revolutionized the retail industry. The IKEA Place app empowers customers to bring any product from the IKEA catalogue into their own environment, allowing them to make informed purchase decisions. This innovative approach to product interaction enables customers to visualize how a particular item would fit within their space without the need to make a physical purchase. Furthermore, customers can modify colors and even multiply objects to enhance their own interior design. The application of superimposing virtual objects onto the real world can also prove advantageous in educational settings, such as learning about bone structures or providing immersive experiences in history and natural science classes.

Outlining augmented reality (AR) involves the utilization of

specialized cameras designed to mimic human vision by delineating specific objects, boundaries, and lines. Although the human eye is widely regarded as the most exceptional camera, it has inherent limitations. Prolonged focus on objects, poor visibility in low-light conditions, and the inability to perceive infrared light are a few of these limitations. To overcome these challenges, specialized cameras have been developed, which are employed in augmented reality (AR) applications that utilize outlining techniques. Object recognition serves as the foundation for the capabilities of outlining AR, which shares some resemblance to projection-based AR. This technique proves beneficial in various scenarios, leveraging object recognition to enhance the understanding of the surrounding environment. Notably, outlining AR finds application in car navigation systems, particularly for ensuring safer driving conditions during nighttime. By employing object recognition, the boundaries of the road can be accurately identified and outlined, aiding drivers in parking their vehicles. This technology shares some similarities with projection-based AR but focuses specifically on object recognition and outlining. Beyond automotive contexts, outlining AR holds potential in fields such as architecture and engineering, where it can assist in outlining buildings and identifying their supporting pillars. This type of augmented reality, combining object recognition and project-based techniques, represents an unparalleled technological solution with vast potential for connecting historical content through augmented reality experiences.

4 Learning styles

Each of us is different, having our own preferences for learning, leading to behavioral manifestations according to these preferences. Preferences may vary depending on the person, task, context, previous experience, education, etc. The frequency, stability, and constancy of the manifestation over time of a particular combination used in the execution of most tasks make it possible to differentiate so-called distinct learning styles. In the educational context, the most known and explored definition of learning styles is the one proposed by Kolb.

Definition 1 [10]. *The learning style designates the concrete ways*

by which individual changes in behavior are achieved through lived experience, reflection, experimentation, and conceptualization.

The most frequently operated in educational practice are typologies based on sensory encoding methods. Depending on the predominant sensory organ in receiving information and transmitting it to the brain, Barbe, Swassing, and Milone [13] differentiate the following learning styles:

1. *Visual Learner* – prefers to learn based on illustrations, images, maps, and diagrams; for the efficiency of understanding and storing new information, it is important to see the written text, because visual memory prevails; prefers written instructions; learns better in solitude;
2. *Auditory Learner* – learns better by listening to a speech or the explanations of others; associates concepts with various sounds and prefers to learn on a musical background; has a better auditory memory; prefers group discussions, debates; prefers verbal instructions for academic tasks; memorizes very well through repetitions out loud;
3. *Kinesthetic Learner* – prefers learning activities in which he/she can experiment, apply, and carry out practical actions; he/she needs to touch, to get involved through movements and manipulations in the learning activity; he/she remembers best what he does; shows a tendency to play with small objects while listening to classes or studying; he/she prefers physical/sports activities.

Stable individual differences in the way of learning affect the rhythms and quality of learning and especially determine the option for one or another learning strategy as one's own and personal way of approaching a learning situation.

In order to facilitate learning, increase study efficiency and successfully adapt to the multitude of learning situations, it is necessary to determine the specific preferences [11] of the personal learning style so that they can be applied in a targeted manner.

4.1 Distinct characteristics of learning styles

1. Visual learner

- observes especially the details of the environment
- remembers what he saw faster than what he heard
- the noise does not distract him
- forgets the verbal instructions
- is a good and fast reader
- prefers to read, not to be read
- speaks fast
- is a good organizer.

2. Auditory learner

- learns by listening to conversations or presentations
- speaks rhythmically
- talks to himself (in his mind)
- is easily distracted by noise
- moves his lips and says the words when he reads
- likes to learn out loud
- is a better storyteller than a writer
- is talkative and likes discussions.

3. Kinesthetic learner

- learns by handling objects
- wants to try objects and mechanisms
- speaks rarely
- has bad handwriting
- stays close to the person he is talking to
- uses body actions to demonstrate what he has learned
- is attentive to gestures

- likes to get involved in games – memorizes while walking
- does not retain geographical locations unless he was there
- uses action verbs

Knowing the person's learning style enables using his strengths during training. Many people show strength in more than one learning style but have a dominant learning style depending on the situation. In order to increase the efficiency of learning, it is recommended to use strategies adapted to learning styles. Suitable learning strategies for different styles are:

1. For visual learners

- highlighting the main ideas, words, and mathematical formulas with different colors
- providing sufficient time for viewing graphs, tables, and images
- using studio tools: maps, tables, graphs
- transcription of the information
- viewing the written information

2. For auditory learners

- explanation of new information, verbal expression of ideas
- reading aloud
- learning with tutors or in a group, where they can ask questions, provide answers, and express how they understand oral information

3. For kinesthetic learners

- handling of the objects to be learned
- arranging tables and diagrams in a correct order
- using movements, dramatization, dance, pantomime, or role-playing games for the development of long-term memory
- talking and walking during knowledge repetition
- learning by applying the learned knowledge in practice.

5 Augmented Reality in Mathematics Education

In the realm of mathematics education, traditional methods have relied on basic tools such as paper, pencils, and chalkboards or whiteboards. Despite advancements in educational technology, the integration of more advanced technological tools into mathematics instruction has been slow to progress. This lack of progress hinders the potential for improvement in mathematics education, even though educational technology has made significant strides.

While it is true that new technologies may not necessarily solve students' difficulties with arithmetic problem-solving techniques, it is crucial to take action rather than remain inactive. By embracing innovative teaching [21] and learning methodologies [22], educators can enhance conceptual understanding, scaffold learning, and foster opportunities for meaningful dialogue surrounding the application of mathematical problem-solving techniques in real-life contexts [14].

According to [18] study, the researchers focused on investigating the existing literature on augmented reality (AR) in mathematics education. The goal was to explore how AR can enhance interactive learning environments for mathematics in various educational settings, including classrooms.

To conduct their research, the researchers selected papers from 10 different databases, namely Scopus, Web of Science Core Collection, ERIC, IEEE Xplore Digital Library, Teacher Reference Center, SpringerLink, zbMATH Open, Taylor & Francis Online Journals, JSTOR, and MathSciNet. By employing the preferred reporting items for systematic reviews and meta-analysis (PRISMA) method, specifically PRISMA2020, they were able to identify 42 relevant studies from these databases. Upon analyzing the selected papers, the researchers found that the implementation of AR in mathematics education consistently yielded positive outcomes. The positive effects of AR on mathematics learning were demonstrated through the various studies reviewed.

AR technology in mathematics can be applied through various approaches, each catering to different learning objectives. For instance, researchers such as Cheng -Chih Wu's Lab [19] focus on utilizing AR to

enhance spatial abilities, while others like Cai [20] explore its potential for teaching probability. In our study, we took a unique approach by developing augmented content based on individual learning styles.

6 Implementation of scenarios in the AR Tool specific to a certain learning style

We develop an AR tool that enhances the learning effect by tracking 2D artefacts that trigger the visualization of the 3D geometry objects using a marker-based Augmented Reality approach, taking learning styles into account.

For this, there were designed 30 types of AR artefacts; when scanned by mobile devices camera, they trigger one of the augmented experiences, such as 3D objects, video content, audio content, text, formulae, and even virtual tutor. Using marker-based augmented developed system, learners can interact with the 3D information, objects, and events in a natural way.

Additionally, pupils can interact with artefacts to change the position, size, and color of 3D objects. Moreover, by interacting with virtual tutor, pupils can see the superimposed digital content that explains and demonstrates the basic theorems. The tool is created by using the Unity platform with the Vuforia database.

MB-AR tool will deliver a positive impact by keeping pupils' high engagement and by enhancing their learning abilities like problem-solving, collaboration, imaginative thinking, and spatial imagination.

The developed friendly interface allows even 2-3th grade kids to be used easily. The tools can be extendable to any age category, even for students. This is created by using the Unity platform with the Vuforia database. For more about workflow, see Fig. 4.

6.1 Scenarios for visual learners

For visual learners, there were developed several scenarios. For example, Figure 5 shows two of them. The first scenario allows viewing the definition of the object to be studied; in this case, it is the definition of the square. The second scenario allows viewing a video sequence in

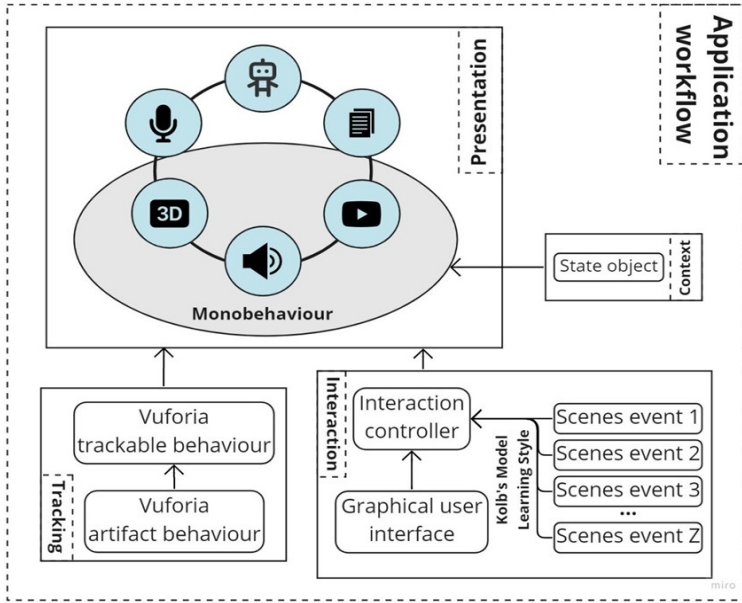


Figure 4. Workflow representation of AR tool

which the notion of the area of the square is explained, and its formula and examples of use are presented.

6.2 Scenarios for auditory learners

For auditory learners, a scenario has been developed that allows the playback of Musical visualisations of Pi (see Fig. 6), composed by Lars Erickson in the early 1990s.

The scenarios for this type of experience are as follows: 1) when the card is scanned, the pi symphony is played; 2) in the example in the middle, the user is asked to calculate the cube of the number 5; if the option 25 is selected, then in the sound form it is pronounced "you're wrong"; for option 255 it is pronounced "try again"; and for option 125 it is pronounced "is correct"; 3) the last example in this section generates the howl of the wolf.

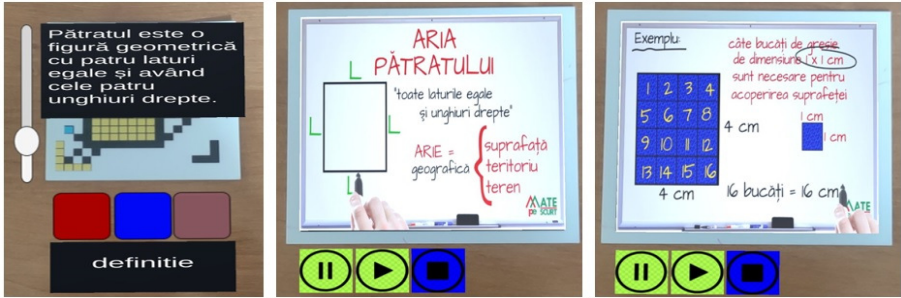


Figure 5. Scenarios for visual learners



Figure 6. Musical visualisations of Pi [12], Scenarios for auditory learners

6.3 Scenarios for kinesthetic learners

For kinesthetic learners, the first scenario has been developed that allows to change 3D object size, to rotate, and change RGB color, or even change color randomly by pressing a bigger green button (see Fig. 7) – handling the object to be learned, in this case, a cube. The second scenario allows viewing angle types.



Figure 7. Scenarios for kinesthetic learners

6.4 Evaluation

To carry out the evaluation process, a test consisting of 5 questions is proposed; after selecting the answer to each of the questions, the accumulated score and the answer for each question are visualized (see Fig. 8).

This allows obtaining an immediate feed-back and strengthening the accumulated knowledge.

7 Conclusion

In this paper, an approach to classifying augmented reality based on the characteristics of different techniques was done. Additionally, the application of a specific type of AR technology in the development of an educational application was demonstrated. Furthermore, the importance of designing augmented learning scenarios that align with the VAK learning styles, aiming to deliver personalized and immersive

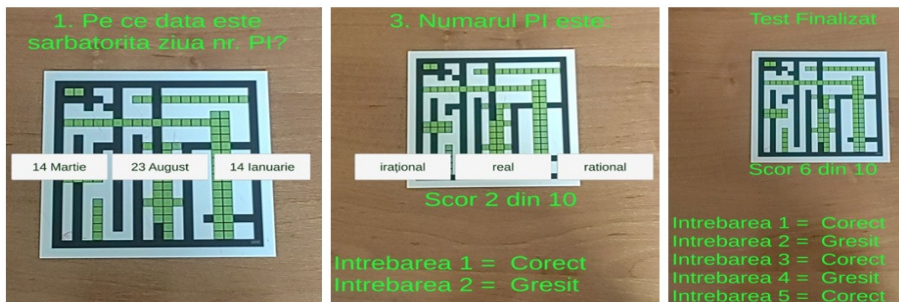


Figure 8. Some stages of scenario for evaluation

learning experiences, was emphasized. The potential for creating engaging and effective educational tools by integration of AR and VAK learning styles was shown.

Acknowledgments. The project Ref. Nr. 20.80009.5007.22 “Intelligent Information systems for solving ill structured problems, knowledge and Big Data processing” has supported part of the research for this paper.

References

- [1] D. Schafer and D. Kaufman, “Augmenting Reality with Intelligent Interfaces,” in *Artificial Intelligence – Emerging Trends and Applications*, 2018, ch. 11, pp. 221–242. DOI:10.5772/intechopen.75751.
- [2] T. P. Caudell and D. Mizell, “Augmented reality: An application of heads-up display technology to manual manufacturing processes,” in *IEEE Xplore Conference: System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference*, 1992, vol. 2, pp. 659–669. DOI:10.1109/HICSS.1992.183317.
- [3] Trekk, [Online]. Available: <https://www.trekk.com/augmented-reality>
- [4] Onirix, [Online]. Available: <https://www.onirix.com/learn-about-ar/what-is-augmented-reality/>

- [5] Yassir El Filali and Krit Salah-ddine, “Augmented reality types and popular use cases,” *International Journal of Engineering, Science and Mathematics*, vol. 8, no. 4, pp. 91–97, 2019.
- [6] Amanda Edwards-Stewar, Tim Hoyt, and Greg Reger, “Classifying different types of augmented reality technology,” *Annual Review of CyberTherapy and Telemedicine*, vol. 14, pp. 199–202. [Online]. Available: <https://www.researchgate.net/publication/315701832>
- [7] Osman Güler and Ibrahim Yucedag, “Developing an CNC lathe augmented reality application for industrial maintenance training,” in *Conference: 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2018, pp. 1–6. DOI: 10.1109/ISMSIT.2018.8567255.
- [8] Veronica Teichrieb, Joao Paulo Silva do Monte Lim, Eduardo Lourenco Apolinario, Thiago Souto Maior Cordeiro de Farias Marcio Augusto Silva Bueno, Judith Kelner, and Ismael H. F. Santos, “A Survey of Online Monocular Markerless Augmented Reality,” *International Journal of modeling and simulation for the petroleum industry*, vol. 1, no. 1, pp. 1–7, august, 2007.
- [9] Wikitude. [Online]. Available: <https://www.wikitude.com/showcase/wikitude-navigation/>
- [10] D. Kolb, *The Kolb Learning Style Inventory*, Version 3, Boston: Hay Group, 1999.
- [11] S. Focsa-Semionov, *Invatarea autoreglata. Teorie. Strategii de invatare*, Chisinau: Epigraf, 2010, pp. 95–116.
- [12] ThePiano.SG, “Musical visualisations of Pi,” [Online]. Available: <https://www.thepiano.sg/piano/read/musical-visualisations-pi>, Accessed on: November 2022.
- [13] W.B. Barbe, R.H. Swassing, and M.N. Milone, *Teaching through Modality Strengths: Concepts and Practices*, Columbus, Ohio: Zaner-Bloser, 1979. ISBN-10: 0883091003, ISBN-13: 9780883091005.

- [14] J. Lai and K. H. Cheong, “Adoption of virtual and augmented reality for mathematics education: A scoping review,” *IEEE Access*, vol. 10, pp. 13693–13703, 2022. DOI: <https://doi.org/10.1109/ACCESS.2022.3145991>.
- [15] “File:Virtual-Fixtures-USAF-AR.jpg,” Wikimedia Commons, [Online]. Available: <https://commons.wikimedia.org/wiki/File:Virtual-Fixtures-USAF-AR.jpg>.
- [16] Jon Judson, “Augmented Reality: A New Reality for Utilities,” Cisco Blogs. [Online]. Available: <https://blogs.cisco.com/energy/augmented-reality-a-new-reality-for-utilities>.
- [17] “6 Uses of Augmented Reality for Manufacturing In Every Industry,” Light Guide Systems, 23 February 2022. [Online]. Available: <https://www.lightguidesys.com/resource-center/blog/6-uses-of-augmented-reality-for-manufacturing-in-every-industry/>.
- [18] Nur Izza Nabila Ahmad and Syahrul N. Junaini, “Augmented Reality for Learning Mathematics: A Systematic Literature Review,” *International Journal of Emerging Technologies in Learning (iJET)*, vol. 15, no. 16, pp. 106. DOI: 10.3991/ijet.v15i16.14961.
- [19] Yi-Ting Liao, Chih-Hung Yu, and Cheng-Chih Wu, “Learning Geometry with Augmented Reality to Enhance Spatial Ability,” in *2015 International Conference on Learning and Teaching in Computing and Engineering*, (Taipei, Taiwan), 2015, pp. 221–222. DOI: 10.1109/LaTiCE.2015.40.
- [20] S. Cai, E. Liu, Y. Shen, L. Liu, S. Li, and Y. Shen, “Probability learning in mathematics using augmented reality: impact on student’s learning gains and attitudes,” *Interactive Learning Environments*, vol. 28, pp. 560–573, 2020. DOI: <https://doi.org/10.1080/10494820.2019.1696839>.
- [21] S. Schutera, M. Schnierle, M. Wu, T. Pertzelt, J. Seybold, P. Bauer, D. Teutscher, M. Raedle, N. Heß-Mohr, S. Röck, et al. “On the Potential of Augmented Reality for Mathematics Teaching with the Application cleARmaths,” *Education Sciences*, vol. 11,

no. 8, Article No. 368, 2021. DOI: <https://doi.org/10.3390/educsci11080368>.

- [22] E. Demitriadou, K. Stavroulia, and A. Lanitis, “Comparative evaluation of virtual and augmented reality for teaching mathematics in primary education,” *Education and Information Technologies*, vol. 1, 2020. [Online]. Available: <https://www.springerprofessional.de/en/comparative-evaluation-of-virtual-and-augmented-reality-for-teac/17028588>.

Inga Titchiev^{1,6}, Oleseca Caftanatov²,
Veronica Iamandi³, Dan Talambuta⁴,
Daniela Caganovschi⁵

Received May 23, 2023
Revised June 20, 2023
Accepted June 27, 2023

^{1,2,3,4}Vladimir Andrunachievici Institute of Mathematics
and Computer Science, SUM
5, Academiei street, Chisinau, Republic of Moldova, MD 2028

¹ORCID: <https://orcid.org/0000-0002-0819-0414>

E-mail: inga.titchiev@math.md

²ORCID: <https://orcid.org/0000-0003-1482-9701>

E-mail: olesea.caftanatov@math.md

³ORCID: <https://orcid.org/0000-0001-6827-1278>

E-mail: veronica.gisca@gmail.com

⁴ORCID: <https://orcid.org/0009-0008-7742-8597>

E-mail: dantalambuta@gmail.com

⁵ORCID: <https://orcid.org/0009-0002-3779-5129>

State University of Moldova

Alexei Mateevici 60, str

E-mail: dana.caganovschi@gmail.com

⁶ Ion Creanga State Pedagogical University of Chisinau

PN2Maude: An automatic tool to generate Maude specification for Petri net models

Ammar Boucherit Messaoud Abbas
Mohammed Lamine Lamouri Osman Hasan

Abstract

Currently, Model-Driven Engineering (MDE) plays a key role in the software development process as it aims to handle their increasing complexity and focuses on the automatic generation of code and/or specifications from system models. This paper presents a very useful tool for the automatic generation of Maude specifications from both Petri net PNML (Petri Net Markup Language) descriptions or incidence matrices. At the end of this paper, a simple but complete Petri net example will be presented to demonstrate the usefulness of the developed tool.

Keywords: Maude, Rewriting Logic, Petri Nets, Code Generation.

MSC 2020: 68N30, 68Q60.

ACM CCS 2020: Grammars and Other Rewriting Systems, Formal Languages.

1 Introduction

Nowadays, computer systems have become more and more indispensable not only in the industrial field, telecommunication, and energy production but in almost all areas of our daily life. On the other hand, because of the increasing complexity and the involvement of several heterogeneous and distributed components interactions in these systems, they are also responsible for an ever-increasing number of errors of varying severity. Therefore, it becomes necessary that the development of software systems should be based on powerful modeling formalisms facilitating the analysis and verification of these systems before

their actual implementation. Moreover, since graphical modeling techniques are of increasing interest, formal approaches accompanied with a graphical representation are becoming more interesting.

Petri nets [1] have been widely used as a graphical and semi-formal tool for specification and analysis of concurrent and complex systems [2], [3]. They are gaining more popularity in recent years since they allow an easy structural and behavioral description of studied systems [4], [5]. Due to their popularity, a large number of tools have been developed for editing, modeling, and analyzing Petri net model's properties [6]. However, the lack of interoperability between such tools, manual preparation, and ad hoc validation of system specifications represent some of their major shortcomings.

In this context, rewriting logic is a very expressive logic that is particularly suitable for formalizing concurrent, complex, and real-time systems [7]. In addition, it is a unifying framework for a wide range of Petri nets [8] and many other concurrency models [9]. Nevertheless, specifications based on the rewriting logic of Petri net-based systems are often voluminous and/or more difficult to refine or correct. Hence, it is necessary to automate such an operation to save time and avoid errors in the process of writing (preparing) specifications.

Model-Driven Engineering (MDE) is known as a promising solution to handle the increasing complexity of software architecture through the automatic generation of code and/or specification from system models. In fact, it helps the designer to integrate formal specification and verification techniques at earlier stages in the software development life-cycle. Therefore, looking for a solution integrating MDE technologies and allowing system designers to quickly, easily, and automatically generate formal specifications of Petri nets are expected to be very advantageous. In addition, we believe that the use of standard input format like the Petri Net Markup Language (PNML) and/or incidence matrices will enlarge the usefulness of this solution and allows designers to use Petri net tools while preparing their system models.

The main objective of this work is to present a tool (PN2Maude) that offers a simple and quick way for Maude practitioners to automatically generate the specification for their Petri net models from both PNML description and incidence matrices. Once the Maude specifi-

cation of a Petri net is generated, designers can then use the Maude LTL model checker or Maude reachability tool to analyze and verify its behavioral properties. This can greatly simplify the process of modeling and verifying concurrent and distributed systems and can help to ensure their correctness and reliability. In fact, this tool is an enhancement and extension of our previous work [10], in which we have proposed an algorithm for the automatic generation of Maude specification based only on the existing rewriting logic semantics by using incidence matrices for pure Petri nets without inhibitor arcs.

The remainder of this paper is structured as follows. In Section 2, we give a brief introduction to the Petri nets descriptions and the classical and improved semantics based on the rewriting logic for Petri nets. Section 3 discusses related works and is followed by Section 4 that presents a brief description of the process of translating the PNML Petri net description into rewriting logic as well as the structure of the tool PN2Maude and its main modules. In Section 5, we present the PN2Maude tool with a simple illustrative example. Finally, Section 6 concludes the paper and draws some perspectives.

2 Preliminaries

2.1 Petri nets Descriptions

Petri nets are typically regarded as one of the widely used modeling tools for the specification and analysis of concurrent and distributed systems. A Petri net can be viewed as a directed bipartite graph, in which arcs are labeled with their corresponding weights and connect nodes from different types. While the first type of nodes represents events, and they are depicted by rectangles or bars (called, Transitions), the second ones represent conditions or objects, and they are depicted by circles (called, Places). Places may contain a discrete number of dots (called, Tokens). Figure 1 below shows a simple Petri net consisting of four places and four transitions.

The distribution of tokens over the Petri net places represents its configuration and is called a marking. A particular transition t is enabled if all its input places (places leading to that transition) contain

sufficient tokens. Thereafter, an enabled transition t may be unconditionally fired, and, therefore, it changes the Petri net marking.

Practically, one can distinguish two sets of places for a transition t :

- $\text{pre-set}(t)$: is composed of the input places of t , also referred to as $\bullet t$. For instance, in our example of Petri net, the $\text{pre-set}(T_4) = \{P_3, P_4\}$.
- $\text{post-set}(t)$: is composed of the output places of t , also denoted by $t\bullet$. Consequently, the $\text{post-set}(T_4) = \{P_1\}$.

Accordingly, a tuple (p, t) is called a self-loop if p belongs to both the $\text{pre-set}(t)$ and $\text{post-set}(t)$. Accordingly, a Petri net that does not contain any self-loop is called pure. In addition, a transition that does not have any input place is called a source transition, and a transition that does not have any output place is called a sink transition. Moreover, a particular kind of arcs is called inhibitor arcs which is denoted by an arc with a small circle attached to a transition. Such kind of arcs is generally used for tests and does not consume tokens after firing. A transition connected by an inhibitor arc with a place will only be enabled if such a place contains fewer tokens than the weight of the inhibitor arc. Figure 1 gives an example of a Petri net.

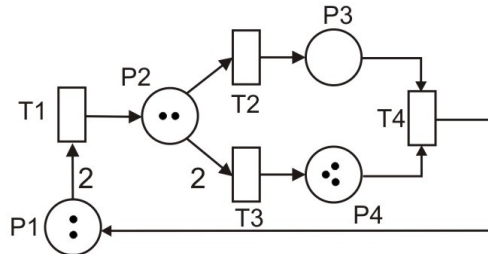


Figure 1. Example of Petri net

2.1.1 Matrix Representation

The easiest mathematical way to represent the structure of a Petri net with N places and M transitions is by using matrix representation. Besides the fact that matrices can provide an alternative way to describe

Petri nets from conventional graphical representation, one of the main motivations for this representation is that it facilitates mathematical analysis of the behavior and properties of Petri nets, such as deadlock and liveness. The three main types of matrices used for Petri net representation are given as follows:

- **PRE-Matrix (Input matrix):** It is a matrix (N,M) that captures all the input places to all transitions. An element of matrix (N,M) equals the weight of the arc linking a place p_i to the transition t_j if it exists and 0 otherwise. The PRE-Matrix may contain negative values in some particular cases, such as the Petri net with inhibitor arcs, to reflect the weights of the inhibitor arcs and to distinguish them from ordinary arcs.
- **POST-Matrix (Output matrix):** It is a matrix (N,M) that captures all the output places to all transitions. An element of matrix (N,M) equals the weight of the arc linking the transition t_j to a place p_i if it exists and 0 otherwise.
- **INC-Matrix (Incidence matrix):** It basically represents POST-Matrix - PRE-Matrix. It can only be used for pure Petri nets. Otherwise, the static structure of a non-pure Petri net will not be properly described, and, in that case, POST-Matrix and PRE-Matrix have to be used instead.

For example, the Petri net in Figure 1 can be specified in matrix representation as follows:

$$\begin{matrix} PRE \\ \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix} \quad \begin{matrix} POST \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} INC \\ \begin{bmatrix} -2 & 0 & 0 & 1 \\ 1 & -1 & -2 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \end{matrix} \quad (1)$$

In these matrices, each row represents a place, and each column represents an input or output of a transition. The values in the matrix represent the number of tokens that are either consumed or produced by transitions, i.e., a positive value in the matrix represents tokens being produced, while a negative value represents tokens being consumed.

2.1.2 PNML Description

The Petri Net Markup Language (PNML) is a standardized XML-based description of Petri net models. Indeed, a PNML description (see Figure 2) is generally made up of a set of main nodes such as “Place”, “Transition” and “Arcs” to describe the structure of a Petri net in the form of a labeled directed graph. In addition, PNML provides a universal interchange file format between various Petri net tools and a common language that allows users and developers to interchange Petri nets, which enables them to reuse models and integrate tools in the analysis of complex systems. Practically, there are many tools, such as WoPeD [11], P3 [12], and PIPE [13], that are capable of exporting the Petri net model in the PNML format.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<pnml>
  <net id="Net-One" type="P/T net">
    <place id="P0">
      <name> <value>P0</value> </name>
      <initialMarking> <value>Default,0</value> </initialMarking>
    </place>
    .....
    <transition id="T0">
      <name> <value>T0</value> </name>
      <timed> <value> false </value> </timed>
    </transition>
    .....
    <arc id="P0 to T1" source="P0" target="T1">
      <inscription> <value>Default,1</value> </inscription>
      <type value="normal"/>
    </arc>
    .....
  </net>
</pnml>
    
```

Figure 2. Example of the structure of a PNML description

As can be seen, the Petri net components are specified in detail with three nodes. For instance, the three attributes `id` (identifier), `name`, and `initial marking` are used to define a place (see node `<place id = ... > ... </place>`). A transition is also described with three attributes, `id` (identifier), `name`, and `timed`, a boolean attribute that determines whether the transition is timed or not (see

node `<transition id = ... > ... </transition>`). Lastly, the attributes `id`, `inscription`, and `type` are used to describe an arc (see node `<arc id = ... > ... </arc>`). The source and target of the arc are identified by the `id` attribute. The weight of the arc is written in the inscription. The type indicates whether the arc is normal or inhibitor.

2.2 Petri nets and Rewriting Logic

Since 1990, rewriting logic [14] has appeared as a promising logical and semantic framework within which Petri nets [8] and many other different concurrent systems and logics can be naturally specified [15]. Maude system is an implementation of rewriting logic that offers a powerful verification toolkit so that Maude's reachability analysis and model-checking can be used to formally analyze and verify the Petri net models with respect to different LTL properties [16].

Usually, a Maude specification is composed of a functional module and/or a system module. While the functional module describes the static part of a system by an equational theory, the dynamic part is described by a set of rewrite rules and, possibly, equations in a system module.

In the first proposed rewriting logic-based semantics for Petri nets [8], there are two main types (`Sorts`, `Place` and `Marking`). The `Marking` is a multiset of `Place` as it represents the distributions of tokens over the Petri net. In addition, the names of places are used to represent token instances. Moreover, each Petri net transition is expressed with a rewrite rule as follows: $[t] \Rightarrow [t']$, where:

- $[t]$ and $[t']$: are terms that represent a part of the global marking of the Petri net and describing the input and output of a transition, respectively.
- \Rightarrow : describes the change to be made while firing a transition ($[t]$ and $[t']$ are the parts to be consumed and produced, respectively).

Therefore, the Maude specification of Petri net in Figure 1 is given in Listing 1.


```

fmod FUNC_PETRINET is
sorts Place Marking .
subsorts Place < Marking .
op empty : -> Marking .
ops P1 P2 P3 P4 : -> Place .
op _ _ : Marking Marking -> Marking
    [assoc comm id: empty] .
endfm

mod SYSTEM_PETRINET is
including FUNC_PETRINET .
rl[T1]: P1 P1 => P2 .
rl[T2]: P2 => P3 .
rl[T3]: P2 P2 => P4 .
rl[T4]: P3 P4 => P1 .
endm
    
```

Listing 1. First Maude specification

In this context, it is worth noting that there is another enhanced rewriting logic-based semantics [17],[18] for Petri nets, and, therefore, the corresponding specification of our Petri net is given in Listing 2.

```

fmod NEW_FUNC_PETRINET is
protecting INT .
sorts PlaceName Place Marking .
subsorts Place < Marking .
op empty : -> Marking .
ops P1 P2 P3 P4 : -> PlaceName .
op <_,_> : PlaceName Int -> Place [ctor] .
op _ _ : Marking Marking -> Marking [ctor assoc comm id: empty] .
endfm

mod NEW_SYSTEM_PETRINET is
inc NEW_FUNC_PETRINET .
vars x1 x2 x3 x4 : Int .
crl[T1]: < P1,x1 > < P2,x2 > => < P1,x1 - 2 > < P2,x2 + 1 > if
    ( x1 >= 2 ) .
crl[T2]: < P2,x1 > < P3,x2 > => < P2,x1 - 1 > < P3,x2 + 1 > if
    ( x1 >= 1 ) .
crl[T3]: < P2,x1 > < P4,x2 > => < P2,x1 - 2 > < P4,x2 + 1 > if
    ( x1 >= 2 ) .
crl[T4]: < P1,x1 > < P3,x2 > < P4,x3 > => < P1,x1 + 1 > < P3,x2 -
    1 > < P4,x3 - 1 > if ( x2 >= 1 ) and ( x3 >= 1 ) .
endm
    
```

Listing 2. Second Maude specification

As can be seen, our Petri net has four places, P1, P2, P3, P4 and four transitions, T1, T2, T3, and T4. In addition, in Listing 1 (respectively, Listing 2), there are two modules.

While the first is a functional module that describes the static part of the Petri net, the second is a system module that describes the dynamic part of the Petri net.

However, the new semantics differs from the existing one in that it encodes the sets of tokens in a place by their cardinality, which allows for the natural expression of various high-level Petri nets such as Petri nets with inhibitor arcs, variable arc weights, colored Petri nets, etc. Additionally, the new semantics facilitates the expression and then checking of behavioral properties such as boundedness, Deadlock, and conservation by using the Maude LTL model checker.

2.3 XSLT Transformation

XSLT is a technology that transforms information from an XML document to another type of document, such as XML, HTML, XHTML, WML, PDF, etc. The process of transformation is mainly based on an XSLT document, called stylesheet, which is an XML format file that contains the information needed by the processor to perform the transformation. Practically, an XSLT Stylesheet is associated with an XML-based document in order to create a resulting document of a different or identical format. This principle is illustrated in Figure 3.

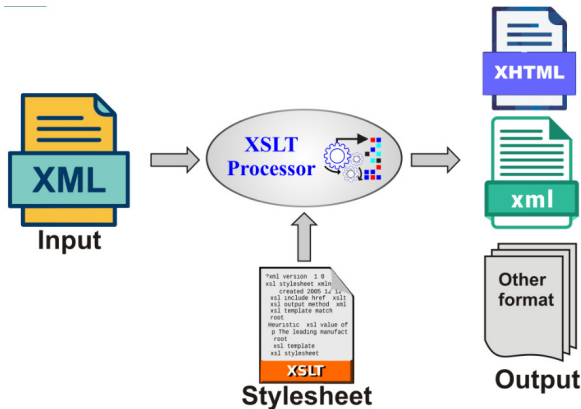


Figure 3. XSLT transformation principle

In the present work, XSLT functions and operators are used to navigate, select nodes from an PNML document, and then generate a purified file (see Figure 5(b)) that contains five blocks describing the set of Petri net places, transitions, and arcs. The purified file is thereafter

used for the generation of Petri net matrix description.

3 Related Works

Petri nets have been widely used for modeling and simulation of distributed and concurrent systems, including communication protocols and synchronization between system components. Unfortunately, most Petri net tools are rarely accompanied by model-checking algorithms. Therefore, Petri net models are often translated into other specific languages for formal analysis purposes [19], [20]. For instance, in [21], authors presented a transcription tool from Petri net to PLC programming languages. The tool makes the translation process more efficient and less error-prone and allows for greater flexibility in system design. Besides the translation process, the program supports stepwise simulation of the Petri net model, allowing for error-checking during the development. According to the authors, the proposed tool can be useful for dynamic integration projects by providing a more efficient and reliable process of design and implementation. In addition, in [22], PetriNet2NuSMV, a tool for the automatic translation of reachability graphs for colored Petri nets and place-transition Petri nets into the NuSMV language, was introduced by the authors. The reachability graphs used as input for this tool are those generated by the TINA and CPN Tools software. This tool allows the formal verification of Petri nets designed with these environments using model-checking techniques for LTL and CTL temporal logics.

In this context, Maude is a very powerful system for which some works have been realized for the transformation of Petri nets [8], [23]. However, very few works have been reported on the automatic translation of Petri nets into Maude. For instance, a tool for the editing, simulation, and analysis of a special extension of Petri nets, so-called ECATNets (Extended Concurrent Algebraic Terms Nets), is presented in [24]. Similarly, a graphic tool allowing a bidirectional translation of colored Petri nets to Maude and vice versa is reported in [25]. Then, a graph transformation-based approach is proposed in [26] for the automatic generation of ECATNets specification in Maude for simulation and analysis purposes.

Moreover, to the best of our knowledge, all the existing works are related to some extension of Petri nets (ECATNets and Colored Petri nets) and their authors did not give any details about the generation algorithm for their approaches, and none of them had made the presented tool available for users, which made the use of such tools too limited.

In addition, the most closely related work is that presented in [10], where authors have proposed an algorithm for the automatic generation of Maude specification based on the existing semantics of Petri nets. However, this work is limited to pure Petri nets without supporting inhibitor arcs.

Finally, the main advantage of the present work over the latter one is that now we generate Maude specification based on both existing and improved semantics to provide a large and solid theoretical basis and thus facilitate the simulation and formal analysis of Petri nets with inhibitor arcs using the Maude system analysis tools.

4 The Translation Approach

This section provides an overview about the proposed approach for translating PNML description into the Maude specification. The tool PN2Maude, developed based on this approach is accessible for users ¹.

4.1 Overview of the Process

The following Figure 4 shows a general description of the developed tool.

The main idea behind the PN2Maude tool is to generate Maude specification in six steps as follows:

1. Create a Petri net model and export it as a PNML file, or use incidence matrices and pass it to Step 5.
2. Translate the importation of the PNML file. . The current version of PNML2Maude supports Petri nets with inhibitor arcs and

¹PN2Maude is available at: https://drive.google.com/file/d/1I2DeysLPZzK5i-0sWtFd1PCuShc_0DqV

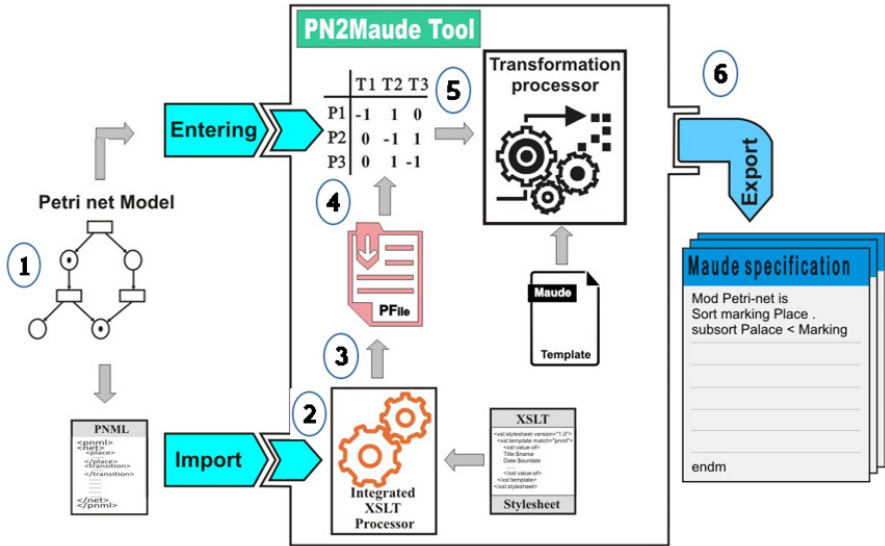


Figure 4. General Description of PN2Maude Tool
is exported by PIPE or P3 tools.

3. Extract data and values from the PNML file and create a purified file (PF) using XSLT.
4. Create incidence matrices from PF.
5. Start translation by using incidence matrices and Maude specification templates.
6. Export the generated specification into the Maude file.

4.2 Details of the transformation Process

There are two primary steps in the process of generating a Maude specification from a Petri net description. The first one deals with creating the incidence matrix from a PNML file, while the second one focuses on creating a Maude specification for a Petri net using the incidence matrices. The subsections that follow provide more details on these two steps.

4.2.1 From the PNML file to the incidence matrices

As we have previously noted, a PNML file typically contains information about the places, transitions, arcs, and other elements of the Petri net, which can be used to construct the incidence matrix. Therefore, the process of obtaining matrices from a PNML file (exported from the P3 or PIPE tools) is based on the use of XSLT language and goes through the following steps:

1. Purification: It allows extracting more or less essential information, such as Petri net places, transitions, arcs as well as the initial marking in order to facilitate the operation of creating the incidence matrices. Figure 5(a)) shows the XSLT code developed to navigate and select the essential nodes describing the Petri net from the input PNML file. Subsequently and in order to facilitate the creation of PRE and POST matrices, these nodes (see Figure 5(b))) are organized into five blocks describing lists of places with their initial markings, transitions, input arcs, inhibiting arcs, and out arcs, respectively.
2. Creation of matrices: This operation is based on the previous operation, where the purified file is manipulated to create and fill in the incidence matrices. This step can be summarized as follows:
 - i. The number of lines (N) of Block 1 (list of places) represents the number of lines for the incidence matrices, and the number of lines (M) of Block 2 (list of transitions) represents the number of columns for the incidence matrices.
 - ii. The elements of the incidence matrices are filled from Blocks 3, 4, and 5.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" encoding="utf-8" />
<xsl:strip-space elements="*" /> <xsl:template match="pnml/net">
<xsl:for-each select="place"> <xsl:value-of select="@id"/>|<xsl:value-
of select="name"/>|<xsl:value-of select="translate(initialMarking,
translate(initialMarking,'0123456789',''),'')"/> <xsl:text>|&#10;
</xsl:text>
</xsl:for-each>
<xsl:for-each select="transition"> <xsl:value-of select="@id"/>|<xsl:
value-of select="name"/> <xsl:text>|&#10;</xsl:text> <xsl:for-each>
<xsl:for-each select="arc[starts-with(@target,'t') or starts with
(@target,'T')]>
<xsl:if test="type/@value = 'normal'">inp_<xsl:value-of select=
"@target"/>:<xsl:value-of select="@source"/>(<xsl:value-of
select="translate(inscription, translate(inscription,
'0123456789',''),'')"/><xsl:text>)&#10;</xsl:text>
</xsl:if>
</xsl:for-each>
<xsl:for-each select="arc[starts-with(@target,'t') or starts-with(@
target,'T')]>
<xsl:if test="type/@value='inhibitor'">h_inp_<xsl:value-of select=
"@target"/>:<xsl:value-of select="@source"/>
<xsl:choose>
<xsl:when test="inscription = ''">
<xsl:text>|</xsl:text><xsl:text>&#10;</xsl:text>
</xsl:when>
<xsl:otherwise>
<xsl:text>(</xsl:text><xsl:value-of select="translate
(inscription,translate(inscription,'0123456789',''),'')"/>
<xsl:text>)&#10;</xsl:text>
</xsl:otherwise>
</xsl:choose>
</xsl:if>
</xsl:for-each>
<xsl:for-each select="arc[starts-with(@source,'t') or starts with
(@source,'T')]> out_<xsl:value-of select="@source"/>:<xsl:value-of
select="@target"/>(<xsl:value-of select="translate(inscription,
translate(inscription,'0123456789',''),'')"/> <xsl:text>)&#10;
</xsl:text>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

(a) : XSLT File

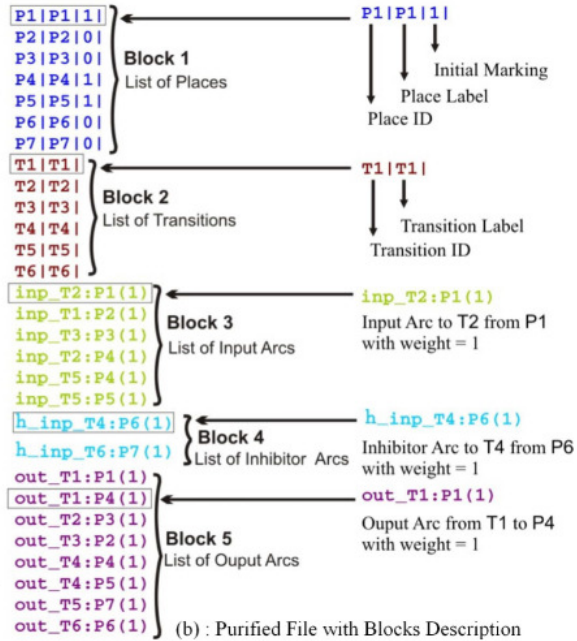


Figure 5. XSLT Code and Purified File with Blocs Description

4.2.2 From incidence matrices to Maude Specification

The principle of generating Maude specifications from incidence matrices can be summarized in the following steps:

- i. The number of rewrite rules is equal to the number of transitions of the Petri net (number of columns in the incidence matrices).
- ii. A rewrite rule is made to describe the firing of a transition. This rule may often be conditional to describe the enabling condition for such a transition, especially in the enhanced semantics. Therefore, such a rule is given as follows:

```
cr1[Label] : <Left_hand_side> => <Right_hand_side> if Cond .
```

- **Label**: a given name to represent the transition.
 - **Left-Hand-Side**: the left part of the rule (LHS), which describes the marking of the Petri net before firing.
 - **Right-Hand-Side**: the right part of the rule (RHS), which describes the marking of the Petri net after firing.
 - **Cond**: a logical expression represents the enabling condition of the transition and is picked up from the PRE-Matrix.
- iii. Each transition is represented by one column in both PRE-Matrix and POST-Matrix. Therefore, some special cases can be determined as follows:
 - **Source Transition**: It is a transition that has columns with null values in the PRE-Matrix.
 - **Sink Transition**: It is a transition that has columns with null values in the POST-Matrix.
 - **Transition with inhibitor arcs**: This transition must have negative values in the PRE-Matrix.
 - **Normal Transition**: It is a transition that is not source, sink, or with inhibitor arcs.

We recapitulate the Maude specification — by using the existing and improved semantics — of the different possible cases of a Petri net transition in Table 1.

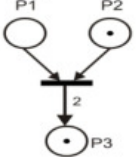
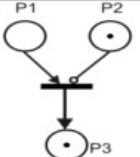
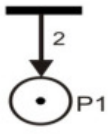
| Type of Transition | | Rewrite Rule | |
|--------------------|--|---|--|
| | | Existing Semantics | Improved Semantics |
| 1 | Ordinary Transition  | LHS: P1 P2 RHS: P3 | LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle \langle P3, X3 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 - 1 \rangle \langle P3, X3 + 2 \rangle$ |
| | | | Condition if $(X1 \geq 1)$ and $(X2 \geq 1)$. |
| 2 | Ordinary Transition with Inhibitor Arc  | it is not expressible without additional operators | LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle \langle P3, X3 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 \rangle \langle P3, X3 + 1 \rangle$ |
| | | | Condition if $(X1 \geq 1)$ and $(X2 < 1)$. |
| 3 | Source Transition  | LHS: M RHS: M P1 P1 | LHS: $\langle P1, X1 \rangle$ RHS: $\langle P1, X1 + 2 \rangle$ |
| | | | Condition this is an unconditional rule |

Table 1. Maude specification of different Petri net Transitions Cases (Continue)

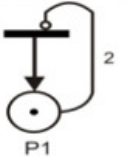
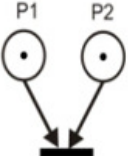
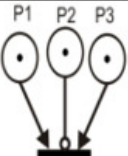
| Type of Transition | | Rewrite Rule | |
|--------------------|--|--|--|
| | | Existing Semantics | Improved Semantics |
| 4 | Source Transition with Inhibitor Arc  | it is not expressible without additional operators | LHS: $\langle P1, X1 \rangle$ RHS: $\langle P1, X1 + 1 \rangle$ |
| | | | Condition if $(X1 < 2)$. |
| 5 | Sink Transition  | LHS: $M \ P1 \ P2$ RHS: M | LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 - 1 \rangle$ |
| | | | Condition if $(X1 \geq 1)$ and $(X2 \geq 1)$. |
| 6 | Sink Transition with Inhibitor Arc  | it is not expressible without additional operators | LHS: $\langle P1, X1 \rangle \langle P2, X2 \rangle \langle P3, X3 \rangle$ RHS: $\langle P1, X1 - 1 \rangle \langle P2, X2 \rangle \langle P3, X3 - 1 \rangle$ |
| | | | Condition if $(X1 \geq 1)$ and $(X2 < 1)$ and $(X3 \geq 1)$. |

Table 1. Maude specification of different Petri net Transitions Cases

4.3 Generation Algorithm

As stated previously, the process of generating a Maude specification of a Petri net is based on its mathematical description or PNML. The general algorithm illustrating the generation process is given as follows in the Listing 3.

Algorithm 1 Automatic Generation of Maude Specifications (Existing and Improved semantics)

Inputs: XML File : Petri net PNML description
 PRE(N,M) : Input incidence Matrix
 POST(N,M) : Output incidence Matrix
 Maude_Spec : Maude specification Templates
 XSLT_Code : XSLT Stylesheet

Outputs: Maude Specification 1 : Existing semantics specification
 Maude Specification 2 : Improved semantics specification

Uses: MSXSL : Microsoft Command Line XSL Transformation

Begin

- 1: **if** (User_Choice = Import_PNML_File) **then**
- 2: SELECT_AND_IMPORT_PNML_FILE
- 3: CREATE_PF_IMPORT_PNML_FILE ▷ PF : is the purified file
- 4: AUTOMATIC_FILL_MATRICES (PRE(N,M),POST(N,M)) ▷ filling
 PRE-Matrix and POST-Matrix
- 5: **else**
- 6: FILL_PRE_MATRIX(PRE(N,M))
- 7: FILL_POST_MATRIX(POST(N,M))
- 8: **end if**
- 9: **NB_Rules** ← M ▷ M : number of columns of incidence matrices = number of
 transitions
- 10: **for** (each semantics_template) **do** ▷ templates for existing and improved
 semantics
- 11: **for** (j ← 1 to NB_Rules) **do**
- 12: TType ← Get_Transition_Type(j, PRE_Matrix, POST_Matrix) ▷
 source, sink, inhibitor or normal transition
- 13: RewriteRule ← Generate_Rule(j, TType)
- 14: Insert_Rule(semantics_template, RewriteRule)
- 15: **end for**
- 16: **end for**

End

Listing 3. First Maude specification

5 Presentation of the PN2Maude tool

PN2Maude is developed using the Delphi 7 language, which is a strongly typed language that supports the structured and object-oriented design, allowing rapid application development (RAD) running on Windows. In practice, the declarative programming language XSLT has been used to purify a PNML file and extract only the information necessary for the transformation and save it in the file purify (PF). This file is then used to create the incidence matrices, which will be used in the process of generating the Maude specification. The latter can be saved in a Maude file. On the one hand, we have used MSXSL, which is Microsoft's free command-line XSLT processor for performing XSL transformations. In the next subsections, we will present the PN2Maude interface windows with a case study for the following Petri net (see Figure 6).

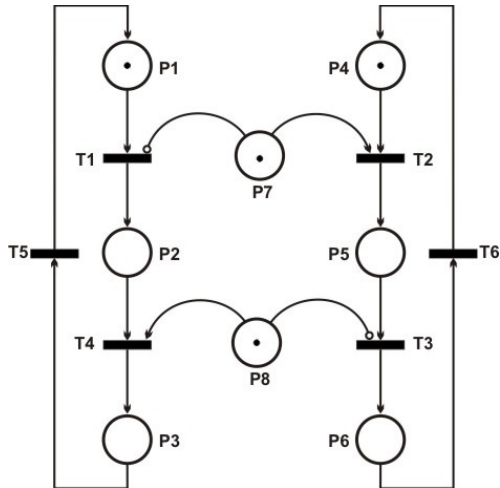


Figure 6. Example of Petri net (Case study)

5.1 Main Interface Window

The main interface window of PN2Maude tool is given in the following Figure 7).

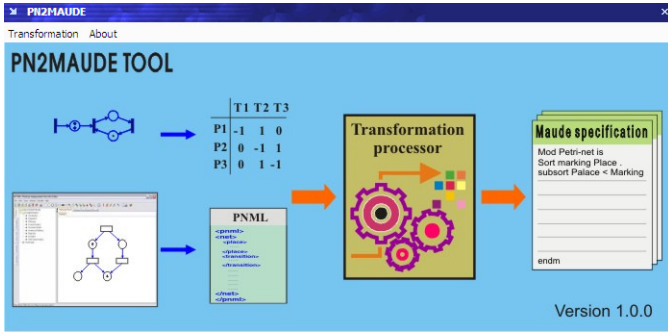


Figure 7. Main Interface Window of PN2Maude Tool

5.2 Using Incidence Matrices choice Interface Window

To introduce the Petri net, one can use the choice of “using incidence matrices” as shown in Figure 8.

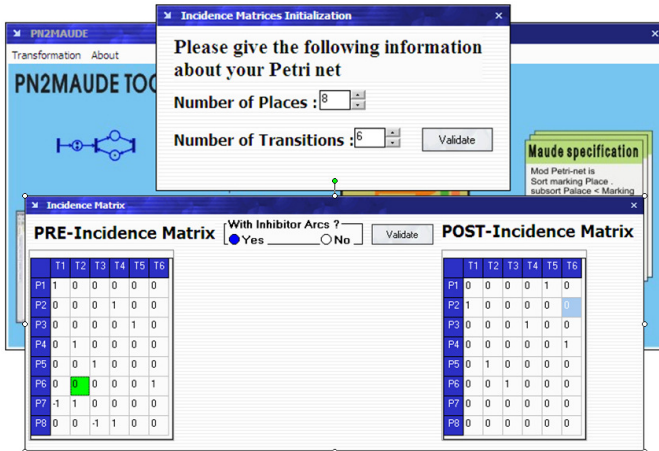


Figure 8. PN2Maude Interface Window for Using Incidence Matrices

5.3 Using PNML File choice Interface Window

The same Petri net may be introduced via a PNML file if it is edited by a Petri net tool, such as PIPE or P3. Therefore, the interface window

for importing a PNML file is given in Figure 9).

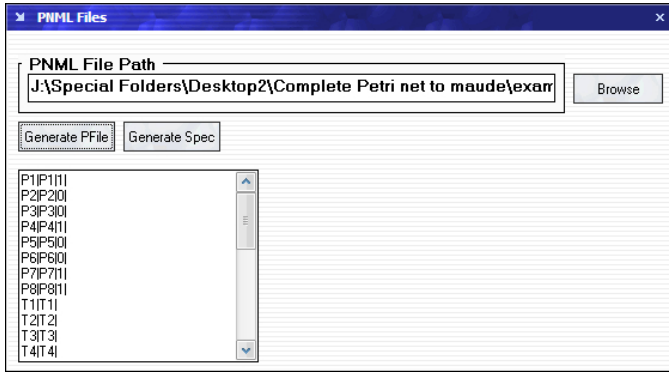


Figure 9. PN2Maude Interface Window for Importing PNML File

5.4 Generated Specification Interface Window

Figure 10 shows the interface window with the generated specification.

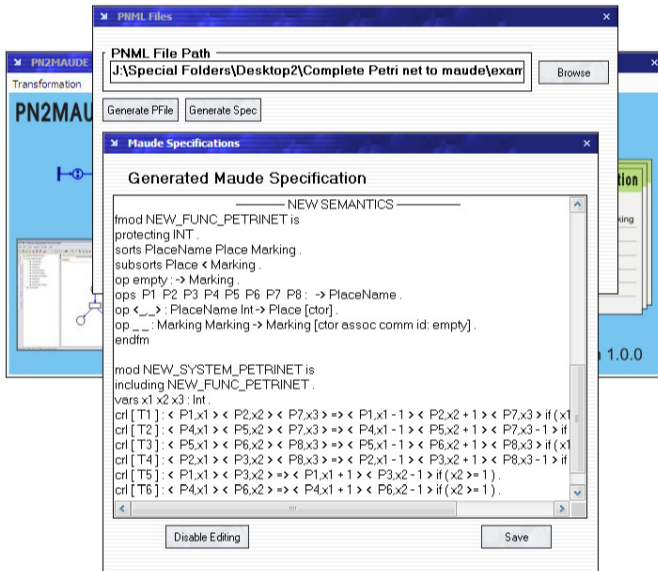


Figure 10. PN2Maude Interface for the generated Specifications

6 Conclusion and Future Work

In this paper, we presented PN2Maude, which is a developed tool for the automatic generation of the Maude specification for a Petri net following the existing and improved semantics based on its mathematical or PNML descriptions. We intend — in the near future — to include a Petri net editor within PN2Maude in order to facilitate the process of Petri net-based modeling for developers and users of our tool. In addition, we aim to develop a web version of PN2Maude and/or a plug-in for Eclipse in order to ensure its wide distribution for users.

References

- [1] W. Reisig and G. Rozenberg, *Carl Adam Petri: Ideas, Personality, Impact*. Springer, 2019.
- [2] E. Huang, L. F. McGinnis, and S. W. Mitchell, “Verifying sysml activity diagrams using formal transformation to petri nets,” *Systems Engineering*, vol. 23, no. 1, pp. 118–135, 2020.
- [3] P. Singh and L. Singh, “Verification of safety critical and control systems of nuclear power plants using petri nets,” *Annals of Nuclear Energy*, vol. 132, pp. 584–592, 2019.
- [4] D. Buchs, S. Klikovits, and A. Linard, “Petri nets: A formal language to specify and verify concurrent non-deterministic event systems,” in *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer, 2020, pp. 177–208.
- [5] J. Dong, J. Jiao, H. Xia, and J. Chu, “Safety simulation and analysis for complex systems concurrency based on petri net and stateflow model,” in *2019 Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, 2019, pp. 1–7.
- [6] W. J. Thong and M. Ameen, “A survey of petri net tools,” in *Advanced Computer and Communication Engineering Technology*. Springer, 2015, pp. 537–551.

- [7] M. Alpuente, D. Ballis, F. Frechina, and J. Sapiña, “Exploring conditional rewriting logic computations,” *Journal of Symbolic Computation*, vol. 69, pp. 3–39, 2015.
- [8] M.-O. Stehr, J. Meseguer, and P. C. Ölveczky, “Rewriting logic as a unifying framework for petri nets,” in *Unifying Petri Nets*. Springer, 2001, pp. 250–303.
- [9] J. Meseguer, “Conditional rewriting logic as a unified model of concurrency,” *Theoretical computer science*, vol. 96, no. 1, pp. 73–155, 1992.
- [10] A. Boucherit, A. Khababa, and L. M. Castro, “Automatic generating algorithm of rewriting logic specification for multi-agent system models based on petri nets,” *Multiagent and Grid Systems*, vol. 14, no. 4, pp. 403–418, 2018.
- [11] T. Freytag, “Woped–workflow petri net designer,” *University of Cooperative Education*, pp. 279–282, 2005.
- [12] D. Gasevic and V. Devedzic, “Software support for teaching petri nets: P3,” in *Proceedings 3rd IEEE International Conference on Advanced Technologies*. IEEE, 2003, pp. 300–301.
- [13] P. Bonet, C. M. Lladó, R. Puijaner, and W. J. Knottenbelt, “Pipe v2. 5: A petri net tool for performance modelling,” in *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, 2007.
- [14] J. Meseguer, “Rewriting as a unified model of concurrency,” in *International Conference on Concurrency Theory*. Springer, 1990, pp. 384–400.
- [15] —, “Twenty years of rewriting logic,” *The Journal of Logic and Algebraic Programming*, vol. 81, no. 7-8, pp. 721–781, 2012.
- [16] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott, “Maude manual (version 3.0),” *SRI International–University of Illinois at Urbana-Champaign*. URL: <http://maude.cs.uiuc.edu>, 2019.

- [17] A. Boucherit, K. Barkaoui, and O. Hasan, “An enhanced rewriting logic based semantics for high-level petri nets,” in *The International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2021)*, 2021.
- [18] A. Boucherit, L. M. Castro, A. Khababa, and O. Hasan, “Petri net and rewriting logic based formal analysis of multi-agent based safety-critical systems,” *Multiagent and Grid Systems*, vol. 16, no. 1, pp. 47–66, 2020.
- [19] K. Korenblat, O. Grumberg, and S. Katz, “Translations between textual transition systems and petri nets,” in *International Conference on Integrated Formal Methods*. Springer, 2002, pp. 339–359.
- [20] O. R. Ribeiro and J. M. Fernandes, “Translating synchronous petri nets into promela for verifying behavioural properties,” in *Industrial Embedded Systems, 2007. SIES’07. International Symposium on*. IEEE, 2007, pp. 266–273.
- [21] A. T. F. de Mello, M. C. Barbosa, D. J. dos Santos Filho, P. E. Miyagi, and F. Junqueira, “A transcription tool from petri net to clp programming languages,” in *ABCM Symposium Series in Mechatronics—Vol. 5, Section IV—Industrial Informatics, Discrete and Hybrid Systems*, 2012.
- [22] M. Szpyrka, A. Biernacka, and J. Biernacki, “Methods of translation of petri nets to nusmv language.” in *CS&P*, 2014, pp. 245–256.
- [23] L. J. Steggles, “Rewriting logic and elan: prototyping tools for petri nets with time,” in *International Conference on Application and Theory of Petri Nets*. Springer, 2001, pp. 363–381.
- [24] N. Boudiaf, A. Chaoui, and H. Bakha, “A rewriting logic based tool for ECATNet’s analysis: Edition and simulation steps description,” *European Journal of Scientific Research*, vol. 6, no. 2, pp. 16–27, 2005.

- [25] N. Boudiaf and A. Djebbar, “Towards an automatic translation of colored petri nets to maude language,” *International Journal of Computer Science & Engineering*, vol. 3, no. 1, pp. 1078–1083, 2009.
- [26] E. Kerkouche and A. Chaou, “A graphical tool support to process and simulate ecatnets models based on meta-modelling and graph grammars,” *INFOCOMP*, vol. 8, no. 4, pp. 37–44, 2009.

Ammar Boucherit, Messaoud Abbas,
Mohammed Lamine Lamouri,
Osman Hasan

Received April 17, 2023
Revised June 15, 2023
Accepted June 15, 2023

Ammar Boucherit¹, Messaoud Abbas², Mohammed Lamine Lamouri³
^{1,2,3}LIAP Laboratory, University of El Oued,
PO Box 789, El Oued 39000, Algeria

¹ORCID: <https://orcid.org/0000-0002-1617-0050>

E-mail: ammam-boucherit@univ-eloued.dz

²ORCID: <https://orcid.org/0000-0002-7998-9020>

E-mail: messaoud-abbas@univ-eloued.dz

³ORCID: <https://orcid.org/0000-0002-1074-624X>

E-mail: lamouri-mohamedlamine@univ-eloued.dz

Osman Hasan

ORCID: <https://orcid.org/0000-0003-2562-2669>

SEECS, National University of Sciences and Technology (NUST),
Islamabad, Pakistan

E-mail: osman.hasan@seecs.nust.edu.pk