Computing and Informatics, Vol. 39, 2020, 1117–1147, doi: 10.31577/cai_2020_6_1117

FINDING EFFECTIVE COMPILER OPTIMIZATION SEQUENCES: A HYBRID APPROACH

Nilton Luiz Queiroz Junior, Anderson Faustino da Silva

Departament of Informatics
State University of Maringá
Maringá, Paraná, Brazil
e-mail: niltonlqjr@gmail.com, anderson@din.uem.br

Luis Gustavo Araujo Rodriguez

Institute of Mathematics and Statistics
University of São Paulo
São Paulo, São Paulo, Brazil
e-mail: luisgar1990@gmail.com

Abstract. The Optimization Selection Problem is widely known in computer science for its complexity and importance. Several approaches based on machine learning and iterative compilation have been proposed to mitigate this problem. Although these approaches provide several advantages, they have disadvantages that can hinder the performance. This paper proposes a hybrid approach that combines the best of machine learning and iterative compilation. Several experiments were performed using different strategies, metrics and hardware platforms. A thorough analysis of the results reveals that the hybrid approach is a considerable improvement over machine learning and iterative compilation. In addition, the hybrid approach outperforms the best compiler optimization level of LLVM.

Keywords: Compilers, optimization, optimization selection problem, iterative compilation, machine learning

Mathematics Subject Classification 2010: 68-N20

1 INTRODUCTION

Compilers are computer programs capable of transforming code written in a source language into a target language [1, 19, 20]. The final product is an equivalent program generated into an executable file.

This process is divided into several phases, and one of the most important is the *optimization phase*. This stage is fundamental because it improves the quality of the final executable program, reducing run-time, code size or even power consumption [1, 4, 10].

An important aspect of compilers is their ability to provide optimizations [13]. However, two optimizations (one after the other) can provide greater benefits. For example, *copy propagation* can generate *dead code*, which is *useless* code that will not be used in the future and does not affect program results. This type of code is removed by a compiler optimization called *Dead-code elimination*, and thus improves the performance.

The previous example provides an insight into how optimizations interact with each other. Based on these interactions, modern compilers (GCC [22], ICC [23], LLVM [24]) offer standard optimization levels (O1, O2, O3), which can be used to optimize the source code. However, the performance achieved by the aforementioned levels is different for each program. This is because optimization selection depends on program features. In addition, the most effective optimizations depend on the system architecture and input, however the latter is usually different and thus its effects are ignored.

The Optimization Selection Problem (OSP) consists in choosing the most effective optimizations for a given program. It is worth mentioning that this type of problem is classified as undecidable. This is because of the search-space size, which is related to the quantity of optimizations provided by the compiler and its possible combinations. Thus, mitigating this problem is highly desirable. There are several mitigating techniques for the OSP, and the most common are the following:

Selecting an optimization set: Optimizations are selected for a given program without considering their order of application. This approach is used frequently because compiler systems such as GCC and HotSpot VM cannot reorder optimizations based on the complexity of the intermediate code, which create dependencies between optimizations [6].

Selecting an optimization sequence: Optimization sequences are selected and their order of application is considered. Sequence selection considers whether or not to repeat optimizations within a sequence.

Parameterization of optimizations: Attempts to find the most effective combination of parameters for optimizations.

In general, researchers investigate only one approach to mitigate the OSP. This paper proposes to select an optimization sequence adopting automatic schemes. Our system either creates sequences for programs using Iterative Compilation (IC) [26]; or selects sequences from a knowledge base using Machine Learning (ML) [10, 15].

IC consists in evaluating the quality of the target code generated by different sequences, and therefore returns the most effective target code. However, ML based approaches attempt to predict sequences, from previously-successful compilations, that will have good performance in new programs.

ML has higher usage than IC based approaches because of its lower response time, which is spent mostly on the training phase. In this stage, it is necessary to evaluate the performance of several different sequences with example programs. Thus, the exact program is compiled and executed several times.

In this context, it is vital to characterize programs. Thus, one of the most difficult tasks is choosing a set of features that can effectively represent a program. Certain studies in the literature have shown that extracting features through control or data flow graphs are strategies that achieve good results; consequently, surpassing features extracted directly from the source code [11, 15]. Research studies also indicate that applying IC for each program function in Just-in-Time (JIT) environments yields better results than characterizing the function and predicting which sequence to use [6].

This paper describes a hybrid approach, already implemented on [8], that combines the best of IC and ML in order to mitigate the OSP. The objective is to describe an approach that initially uses ML to select potential optimization sequences. This is done considering the features of an unknown program. Afterwards, it applies IC to adapt potential optimization sequences to the said program. Thus, it is expected that performance will improve by adapting the solution rather than only using potential sequences.

Furthermore, the hybrid approach applies a learning scheme for recently-compiled programs. This is done using a Genetic Algorithm (GA) that creates new sequences, and thus the explored portion of the search space will always have these types of sequences. Therefore, these new sequences can be used for recent programs either after: feeding the knowledge base; compiler processing, or finding sequences for batch programs. In addition, this paper also includes the analysis, and propose a new approach to select the initial solution for the GA. It is called Centralized Sequence Selection, that choses all sequence from the most similar program.

The main contributions of this paper are as follows:

- a comparison between two different strategies to select the initial solution;
- different approaches for feeding a knowledge database; and
- an analysis of the performance impact of the approach with different hardware platforms and input sets.

The results indicate that the proposed hybrid approach outperforms both IC and ML. Furthermore, the average speedup achieved by the hybrid approach is superior to the best compiler optimization level of LLVM.

The rest of the paper is organized as follows. Section 2 introduces the hybrid approach for mitigating the OSP. Section 3 presents the instantiation of the hybrid approach. Section 4 describes the experimental environment and setup. Section 5 presents a discussion of the results, comparing them with other approaches proposed in the literature. Section 6 presents related works. Finally, Section 7 provides the conclusions of this paper.

2 A HYBRID SOLUTION FOR MITIGATING THE OSP

IC is an appealing option because it achieves better results than ML. However, ML is also interesting because it applies strategies capable of accelerating the convergence to a good solution. Thus, this paper describes a hybrid approach for solving the OSP. The objective is to combine the best of both IC and ML. The proposed hybrid approach can be described as follows.

2.1 Overview of the Hybrid Approach

Suppose there exists a training set, containing S good optimization sequences for P programs with their features. First, the approach creates a model based on the knowledge database (KD), which is used to predict good optimization sequences for a particular test program. Second, the approach selects N potential optimization sequences, using the created model, for the test program. Afterwards, the N sequences will feed a solution adapter, which utilizes a strategy based on IC to adapt (improve) the solution found in the ML phase. Finally, the best target code, found by the adapter, is returned to the user and the KD is updated with recent knowledge.

Although the system has the capacity of providing itself with feedback, an initial KD is necessary. In addition, a database generator is used just once, and thus the initial knowledge is built with both sequences and performances for some programs. Therefore, the system has the capacity to generate new sequences and provide itself with a feedback.

The system increments its KD as new programs are compiled. However, not all sequences will be possible candidates for the compilation of new programs. This occurs because the database is filtered, and thus poor performing sequences are discarded. It is important to highlight that the architecture is flexible enough to allow for a change of focus in terms of performance improvement. Therefore, it is necessary to store/record values in the KD.

The proposed hybrid approach is shown in Figure 1, which will be described more specifically in the following subsections.

As shown in Figure 1, the components of the hybrid architecture can be divided into 3 main groups:

1. Group of training components;
2. Group of prediction components; and

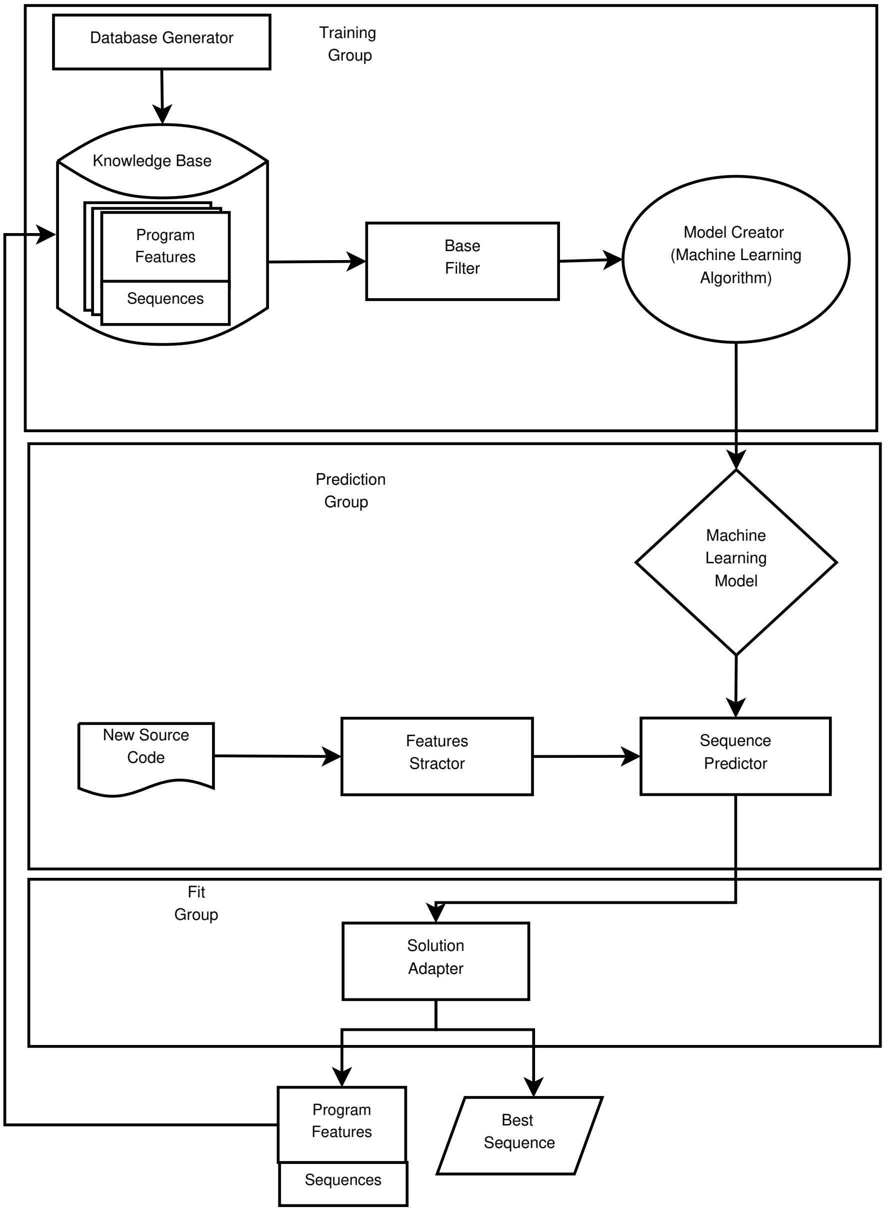


Figure 1. Architecture of our hybrid approach

3. Group of adaptive components.

These components will be described in Sections 2.2, 2.3 and 2.4. In addition, the approaches used to feed the knowledge database will be discussed in Section 2.5.

2.2 Group of Training Components

This group is comprised of components that handle the *background* process of the system. The aforementioned group has a component called the *database generator*, which is executed just once to create the KD. It is also comprised of a *database filter* and generative models in the context of ML, which the latter is used for prediction purposes.

Database Generator is an algorithm used for creating a knowledge database, which is usually created randomly [10, 2, 15]. However, there are several strategies for generating the knowledge database [7, 18]. The proposed hybrid approach uses a GA, which will be discussed in Section 3.1.1. The database generator will be used just once because of two reasons. First, it is a large IC process. Second, its objective is to generate a large database for initial programs, which will be compiled on the system.

Database Filter. The hybrid architecture considers similar programs, and thus chooses sequences that will compose the initial population of the GA. The low-performing sequences, in relation to the evaluation criteria, will be excluded from the model creation phase. However, these sequences will remain in the KD. The *database filter* is the component that executes these tasks. It discards bad sequences in terms of the performance goal, and associates the remaining sequences with the program features. It is consistently executed in the KD before the ML model is generated.

Generative Models in the Context of ML. This component collects program information given by the database filter. Afterwards, it creates the ML model using this information. This model will be utilized by the prediction components. Therefore, the parameters of the ML algorithm are defined in this phase. Subsequently, the model is created using sequences and information provided by the database filter. It is important to highlight that several ML algorithms can be used to create the model. The only restriction is that the algorithm must have a training and testing phase. In addition, the algorithm must be able to provide a classification based on a ranking scheme.

2.3 Group of Prediction Components

This group is comprised of components that predict sequences of the initial population. These components consist of a single feature extractor and sequence prediction scheme.

Feature Extractor. This component receives the source code, and afterwards extracts the features used in the prediction scheme. The extracted features are the same as those stored in the KD, and they will feed the KD using sequences generated in the solution adaptation stage. These features are also used for feeding purposes in the sequence prediction scheme.

Sequence Prediction Scheme. This module selects sequences that will compose the initial solution for the solution adapter. It receives a prediction model, created by the generative model, and features extracted from the test program. The selected sequences are chosen based on the similarity between program features in the KD and testing phase. The sequences can be chosen either from a single program or several programs.

2.4 Group of Adaptive Components

This group is comprised of just one component, which adapts the solution to the new program. This component is called the *solution adapter*.

Solution Adapter generates the final solution. Furthermore, this component creates solutions and feeds the knowledge database; in other words, it deeply explores the search space of the OSP. The initial solution of the algorithm is comprised of K sequences, which are selected by the sequence prediction scheme. Afterwards, the algorithm chosen to adapt the solution runs over these sequences. This algorithm must be capable of receiving and improving at least one sequence. The database generator stores all generated sequences in the KD. However, only the most effective sequence is considered the final solution, and thus it is given to the test program.

2.5 Approaches for Feeding the Knowledge Database

Although the hybrid approach is flexible enough to operate without feeding the database, a scheme for such a process generates knowledge for medium and long-term goals. Thus, the architecture of the hybrid approach allows the user to choose when the feedback will occur. Thus, two approaches are proposed for feeding the KD:

Constant Feeding: The hybrid approach stores new information in the KD (features and sequences) and recreates the model. This process is done for all compiled programs.

Batch Feeding: The hybrid approach stores new information in the KD, and recreates the model after K compiled programs. This is done for every compiled program.

These two approaches have different execution frequencies for creating the ML model.

The ML model based on the constant-feeding approach is created after a solution is found. Although, it has a higher cost than batch feeding, it has an intense feedback. This feeding mechanism produces a more non-deterministic approach than batch feeding. This result is produced because there are $N!$ forms of organizing N programs. Thus, altering the order of the programs can modify the initial population of the subsequent program; consequently, producing a different result.

However, batch feeding stores information and generated sequences in the database, and the model is recreated after K programs are compiled. The non-determinism of this feeding approach is less than constant feeding because altering the order of the programs that are in the same batch will not modify the initial population of the subsequent program. It is worth noting that constant feeding is basically a version of batch feeding with $K = 1$.

3 INSTANTIATION OF THE HYBRID APPROACH

The hybrid approach, described in the previous section, can be instantiated using different strategies. Thus, this section describes how it was implemented. The proposed strategy was implemented as a tool of LLVM [9], which was chosen because it allows full control over optimizations.

This means that it is possible to enable a list of optimizations through the command line. The position of each optimization indicates its order of application.

The infrastructure implemented can be viewed in Figure 2.

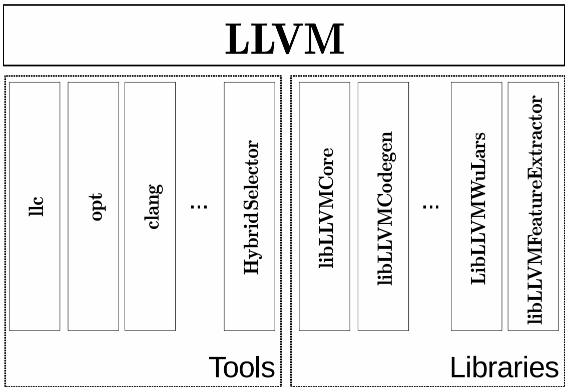


Figure 2. Infrastructure of the instantiations

Two plugins were implemented for LLVM:

1. libWuLars: used for extracting the hottest function of the program, as proposed by Wu and Larus (1994) [25]; and
2. libFeaturesExtractor: used for extracting features proposed by Namolaru et al. [14].

The remaining libraries (also known as *libs*) are standard for LLVM. In addition, they assist with developing plugins for LLVM.

The tools are provided by LLVM for compiling and optimizing code among other functions. Furthermore, a tool called HybridSelector is also used, which helps implement the proposed hybrid approach in Figure 1.

HybridSelector uses *Support Vector Machine* (SVM) as an ML algorithm. In addition, all IC phases are done using GAs. Finally, all features are extracted without the need to execute the program.

The following sections will present in detail the implementation of the HybridSelector tool. In addition, these sections will describe each component and the different parameterizations used for the experiments.

3.1 Iterative Compilation

IC is used in two phases of the hybrid approach. Its purpose is to create the KD and adapt the solution. Although the parameters are modified, the database generator and solution adapter use the same GA for each instantiation.

3.1.1 Database Generator

The algorithm 1 presents a pseudocode to the KD generation.

The GA presented in Algorithm 1 was used to create the database. This algorithm is executed with two distinct parametrization. Although these parameters are similar, they have two differences:

1. total number of individuals in a population;
2. and a stop criterion.

Both executions use *tournament selection*, and possess crossover and mutation operators. Although there are four mutation operators in the algorithm, only one can be applied to each individual. Thus, for every individual that mutation are applied, only one of four is chosen.

The mutation operators consist of:

1. substituting a randomly-positioned optimization for any other valid optimization;
2. permutation-based procedures for two optimizations that compose the sequence;
3. including a randomly-selected optimization into the sequence; and
4. excluding a randomly-positioned optimization.

Crossover takes a portion (specifically half) of each solution and concatenates them. The probability of applying mutation and crossover operators are 40 % and 60 %, respectively, as specified in [12]. In addition, this paper proposes an elitism-based algorithm, and thus the best/most effective solution is carried out to the

```
Data: TrainPrograms, MaxSeqSize, NumIndividuals, BaselineList
Result: $KD$
KD ← ∅;
foreach Program ∈ TrainPrograms do
    Generation ← ∅;
    NumGeneration ← 0;
    Sequences ← ∅;
    Size ← Random(1, MaxSeqSize);
    ProgramFeatures ← getFeaturesByFunction(Program);
    // Create a dictionary with functions names as keys and
       functions features as values
    Baselines ← GetBaseilnes(Program, BaselineList) // Compile with
       Compiler Baselines in BaselineList and get its execution
       time
    for $i$ ← 1 to NumIndividuals do
        Individual ← CreateIndividual(Size); // Create an optimization
           sequence
        IndividualFitness ← Fitness(Individual, Program); // Compile and
           execute program with an individual
        Generation ← Generation ∪ (Individual, IndividualFitness) ;
    end
    Sequences[NumGeneration] ← Generation;
    while not reach at least one stop criteria do
        Generation ← evolve(Generation); // do Crossover, Mutation
           and get fitness of each individual
        NumGeneration ← NumGeneration + 1;
        Sequences[NumGeneration] ← Generation;
    end
    KD ← KD + (Program, Baselines, ProgramFeatures, Sequences)
end
```

Algorithm 1: Genetic algorithm to create KD

next generation. The fitness function used in this paper refers to the run-time of the program given in seconds. This function calculates the arithmetic mean of 5 executions for each sequence.

Both initial populations are randomly generated and comprised of either 10 or 50 individuals. Each chromosome of the individuals is a string that specifies one optimization to the compiler (those string are presented in Table 2). The number of individuals is given by a specific parameter. The size of each individual is randomly generated as well, varying between 1 and 61. This range was chosen because it relates to the number of different optimizations available (O1, O2, O3). The algorithm has 3 stop criteria:

1. The standard deviation of the fitness function is less than 0.01.
2. The total number of generations is either:
   - 100 with an initial population of 50 individuals; or
   - 20 with an initial population of 10 individuals.
3. The best fitness value does not improve after three consecutive generations.

After the GA is executed with the aforementioned parametrizations, all sequences are gathered to create the KD. Thus for every experiment, the first model is built with the KD created in the previous phase.

3.1.2 Solution Adapter

The GA was also used for adaptation purposes. Its parameterization is very similar to the GA presented in Section 3.1.1.

The mutation and crossover operators are identical, and thus have equal probability. Furthermore, the fitness function is identical as well. Thus, the main difference between the algorithms is their initial population, which is not comprised of randomly-selected individuals. Instead, the initial population is selected from the KD, and is comprised of 10 individuals and 20 generations, and every individual is a optimization sequence that is applied on the whole program.

3.2 Feature Extraction

The features used for all instantiations are shown in Table 1, and were proposed by Namolaru et al.

These features are provided by two different scopes, which are based on:

1. The entire program structure: this indicates that the extracted features describe the entities of the entire program;
2. Hot functions: this indicates that the program will only be represented by its hottest function, which is highly beneficial for the compiler to optimize. The algorithm used to search for the hottest function was proposed by Wu and Lars (1994) [25]. This strategy is justified by Amdahl's Law [16].

In both cases, the features were not submitted to a prior preprocessing.

3.3 Machine Learning

SVM is a popular and widely-acclaimed ML algorithm, and thus was chosen for this experiment. A machine learning library called *Scikit-learn* was selected for implementing SVM [17]. The configuration, parametrization and implementation of the algorithm are described in the following sections.

Number of Instructions
Number of assignment instructions
Number of integer binop instructions
Number of float binop instructions
Number of terminator instructions
Number of bitwise binop instructions
Number of vector instructions
Number of memory access and addressing instructions
Number of aggregate instructions
Number of integer conversion instructions
Number of float conversion instructions
Number of call instructions
Number of call instructions that has pointers as arguments
Number of call instructions that have more than 4 arguments
Number of call instructions that return an integer
Number of call instructions that return a float
Number of call instructions that return a pointer
Number of switch instructions
Number of indirect branches instructions
Number of conditional branches instructions
Number of unconditional branches instructions
Number of load instructions
Number of store instructions
Number of GetElemPtr instructions
Number of other instructions
Number of PHI nodes
Number of BBs with no PHI nodes
Number of BBs with up to 3 PHI nodes
Number of BBs with more than 3 PHI nodes
Number of Basic Blocks (BB)
Average number of instructions per BB
Number of edges in a Control Flow Graph (CFG)
Number of critical edges in a CFG
Average number of PHI nodes per BB
Number of BBs with 1-successor
Number of BBs with 2-successor
Number of BBs with more than 2-successor
Number of BBs with 1-predecessor
Number of BBs with 2-predecessor
Number of BBs with more than 2-predecessor
Number of BBs with 1-successor and 1-predecessor
Number of BBs with 2-successor and 1-predecessor
Number of BBs with 1-successor and 2-predecessor
Number of BBs with 2-successor and 2-predecessor
Number of BBs with more than 2-successor and 2-predecessor
Number of BBs with less than 15 instructions
Number of BBs with more than 15 instructions and less than 500 instructions
Number of BBs with more than 500 instructions

Table 1. Features

3.3.1 Parametrization of SVM

SVMs are effective tools for binary classification. One-Versus-All (commonly referred to as OVA) is a strategy used for these types of problems, and was implemented for this experiment. Thus, the decision function of our SVM algorithm is capable of ranking test programs. In addition, we also analyzed the possibility of using a statistical SVM, however the results did not match our predictions because it had a low sample rate for each class.

The SVM kernel function used were linear function, and all the parameters of the kernel were the default of *Scikit-learn* library.

The adopted approach considers every program of the KD as a class for the SVM. Thus, each class is comprised of only one example because its program characterization was static. If a dynamic characterization occurs, every program execution is collectively seen as an example.

The features used to classify the program are extracted by the Feature extractor, which was previously discussed in Section 3.2.

3.3.2 Model Creator

The algorithm 2 presents the pseudocode to the model creator.

```
Data: KD, Representation, SVMParameters
Result: Model
SVMFeatures ← ∅;
KD ← FilterBase(KD)// Removes sequences worst than the best
    baseline also remove programs that have one baseline with
    execution time equals 0 from base
if Representation = HotFunction then
    foreach (Program, Baselines, Features, Sequences) ∈ KD do
        hot ← findHotFunction(Program);
        Vector ← toVector(Features[hot]);
        SVMFeatures ← SVMFeatures ∪ (Vector, Program.Name);
        // Vector is a vector of features and Program.Name is the
            program name string, that will be used as label in SVM
    end
else
    foreach (Program, Features, Sequences) ∈ KD do

    end
    ProgramFeatures ← sumDictFields(Features);
    Vector ← toVector(ProgramFeatures);
    SVMFeatures ← SVMFeatures ∪ (Vector, Program.Name);
end
SVMModel ← SVMTrain(SVMFeatures, SVMParameters);
Model.SVM ← SVMModel;
Model.Representation ← Representation;
Model.KD ← KD;
```

Algorithm 2: Algorithm to create a SVM model

It is worth highlighting that training the SVM is done with data filtered from the KD. In addition, features that do not appear in at least one program from the KD are excluded from the training phase. The number of features can increase as new programs are added to the KD.

The database filter uses speedup as a threshold compared to the LLVM optimization levels.

The model creator has two types of instantiations, and the scope of features given by the feature extractor is different for each type. These two scopes were previously discussed in Section 3.2.

3.3.3 Sequence Predictor

The main objective of the SVM prediction phase is to predict the initial population, and consequently the GA will improve the sequences. This GA is described in Section 3.1.2.

Thus, a new program P is given to the feature extractor as an input. Afterwards, the collected features with an ML model are given to the sequence predictor. Therefore, the predictor will select sequences that will compose the initial population of the GA.

However, some sequences can cause errors to the LLVM optimizer, and thus it is vital to validate the selected sequences. Therefore, each sequence is validated, and their error prone counterparts are discarded.

The sequence prediction was instantiated in two different ways:

- Centralized: All sequences of the population are provided by the program with the highest similarity. Thus, only the aforementioned program is predicted, and consequently its sequences are extracted. However if the most similar program is not able to provide all the sequences, the second most similar program is chosen, and so forth. This process repeats until the initial population is completely built.
- Distributed: Programs provide a number of sequences (N_p), and it is proportional to the value of the decision function (Decision_val_p). This value is given by the prediction function of the SVM model. In this case, the programs provide their best sequences, and each program provides N_p, which is calculated by the equation 1. This prediction strategy is implemented to generate diversity between the initial sequences of the GA, considering that the sequences will originate from different programs. The sequences are extracted until the size of the initial population is reached, and are ordered from the most similar to least similar program.

$$N_p = \left\lceil \text{Population\_size} \times \frac{\text{Decision\_val}_P}{\sum_{x \in \text{Base}} \text{Decision\_val}_x} \right\rceil \tag{1}$$

In the experiments presented in this paper, the GA for adapting solutions has an initial population of 10 individuals. In the Distributed sequence prediction, each program contributes with only a single sequence. This occurs because the decision function of the SVM has small differences between two programs that occupy consecutive positions. The Algorithm 3 presents the pseudocode of this phase.

```
Data: Model, PredictionType, K, NewProgram
Result: Population
SVMModel ← Model.SVMModel;
Representation ← Model.Representation;
KD ← Model.KD;
NewProgramFeatures ← getFeaturesByFunction(NewProgram);
if Representation = HotFunction then
    hot ← findHotFunction(Program);
    NewFeatures ← NewProgramFeatures[hot];
else
    NewFeatures ← sumDictFields(NewProgramFeatures);
end
SimilartyRank ← Classify(NewFeatures, SVMModel);
// Creates an array where each element is a tuple composed by
   the programs in KD and its decision function value according
   to SVMModel.  This array is sorted byt decision function
   value.
Population ← ∅;
rank ← 0;
S ← size(Population);
if PredictionType = Centralized then
    while S < K do
        MostSimilar ← SimilarityRank[rank].Name;
        Population ←
            Population ∪ SelectSequences(MostSimilar, K-S, KD)// Select
            the K-S best Sequences form the MostSimilar
        S ← size(Population);
        rank ← rank + 1;
    end
else
    SumDV ← SumDecisionValues(SimilarityRank);
    while S < K do
        Program ← SimilarityRank[rank].Name;
        DV ← SimilarityRank[rank].DecisionValue;
        numProgSeq ← ⌈K × (DV/SumDV)⌉
        Population ← Population ∪ SelecteSequences(Program, K-S, KD);
        S ← size(Population);
        rank ← rank + 1;
    end
end
```

Algorithm 3: Algorithm to generate the starter population

3.4 Approaches for Feeding the Knowledge Database

Three experiments were conducted to evaluate the approaches for feeding the database. These experiments were handled for each instantiation of the hybrid approach. Two of these experiments were conducted to evaluate batch feeding, while the other evaluates constant feeding.

For batch feeding, the conducted experiments were the following:

- The first experiment consisted in predicting and adapting the solution for every program from one benchmark. This is done using a model generated by the KD of micro-benchmarks. Although the KD was fed during this procedure, the model was not recreated. Afterwards, an entirely new model was created, and the same procedure was made for every program from a different benchmark. However, the latter does not build new models.
- The second experiment is very similar to the first, however the benchmark order was inverted.

For constant feeding, the test programs were ordered alphabetically, and the model was recreated after every prediction and adaptation procedure.

4 EXPERIMENTAL ENVIRONMENT

The following subsections describe the hardware platform, strategies, benchmarks and metrics used for the experiments.

4.1 Experimental Architecture

The experiments were conducted in the following environment:

Hardware: Intel Core i7-3770 processor with a frequency of 3.40 GHz, 8 MB cache and 8 GB of RAM;

Operating System: Ubuntu 15.10 with kernel 4.2.0-35-generic.

4.2 Compilation System

The compilation system used was LLVM, which has difficulties with certain sequences, and thus the LLVM optimizer (opt) hangs or crashes; consequently having unresponsive behavior. Therefore, this problem was mitigated by reducing the number of optimizations. Thus, sequences were comprised of optimizations from O1, O2, and O3. These optimizations are shown in Table 2.

These optimizations do not guarantee that problem-less sequences will generate, however it does reduce unresponsive behaviors and crashes/hangs.

adce	alignment-from-assumptions	always-inline	argpromotion
assumption-cache-tracker	barrier	basicaa	basiccg
bdce	block-freq	branch-prob	constmerge
correlated-propagation	deadargelim	domtree	dse
early-cse	elim-avail-extern	float2int	functionattrs
globaldce	globalopt	gvn	indvars
inline	inline-cost	instcombine	ipsccp
jump-threading	lazy-value-info	lcssa	licm
loop-accesses	loop-deletion	loop-idiom	loop-rotate
loop-simplify	loop-unroll	loop-unswitch	loop-vectorize
loops	lower-expect	memcpyopt	memdep
mldst-motion	no-aa	prune-eh	reassociate
scalar-evolution	sccp	scoped-noalias	simplifycfg
slp-vectorizer	sroa	strip-dead-prototypes	tailcallelim
targetlibinfo	tbaa	tti	verify

Table 2. Optimizations

4.3 Benchmarks Used

We used three benchmarks: two to evaluate strategies and one to evaluate the KD generation.

KD Generation. This phase uses micro-kernel applications, which in this paper are referred to as *micro-benchmarks*. These applications are available on the LLVM test-suite, and were used for experiments conducted by Purini and Jain (2013) [18]. The complete list of these applications is shown in Table 3.

Test Programs. We used the Collective Benchmark (cBench, with the dataset configured to 1; and the Polyhedral Benchmark (PolyBench), with the dataset configured to extralarge. These benchmarks are shown in Table 4.

4.4 Evaluation Metrics

Four metrics were used for analyzing the results:

1. Speedup over O0;
2. NPS: number of programs that achieve higher speedup than the best compiler optimization level. This process is also called coverage;
3. NoS: number of evaluated sequences; and
4. ReT: response time.

The speedup is calculated as follows:

$$\text{Speedup} = \text{Runtime\_Level\_O0}/\text{Runtime}.$$

ackermann	flops-8	perm
ary3	fp-convert	pi
binary-trees	hash	pidigits
bubblesort	heapsort	puzzle
chomp	himenobmtxpa	puzzle-stanford
dry	huffbench	queens
dt	intmm	queens-mcgill
fannkuch	lists	quicksort
fasta	lpbench	random
fasta-redux	mandel	realmm
fbench	mandel-2	recursive
ffbench	mandelbrot	reedsolomon
fib2	matrix	regex-dna
fldry	methcall	richards_benchmark
flops	misr	salsa20
flops-1	n-body	sieve
flops-2	nsieve-bits	spectral-norm
flops-3	oourafft	strcat
flops-4	oscar	towers
flops-5	partialsums	treesort
flops-6	perlin	whetstone
flops-7		

Table 3. Micro-benchmarks

cBench					
automotive_bitcount	bzip2d	consumer_mad	network_dijkstra	security_blowfish_e	security_sha
automotive_qsort1	bzip2e	consumer_tiff2bw	network_patricia	security_pgp_d	telecom_adpcm_c
automotive_susan_c	consumer_jpeg_c	consumer_tiff2rgba	office_ghostscript	security_pgp_e	telecom_adpcm_d
automotive_susan_e	consumer_jpeg_d	consumer_tiffdither	office_synth	security_rijndael_d	telecom_CRC32
automotive_susan_s	consumer_lame	consumer_tiffmedian	security_blowfish_d	security_rijndael_e	telecom_gsm

Polybench					
2mm	cholesky	durbin	gesummv	lu	syr2k
3mm	correlation	fdtd-2d	gramschmidt	mvt	syrk
adi	covariance	floyd-warshall	heat-3d	nussinov	trisolv
atax	deriche	gemm	jacobi-2d	seidel-2d	trmm
bicg	doitgen	gemver	ludcmp	symm	

Table 4. Test programs

4.5 Strategies

Several strategies were evaluated. Table 5 presents the strategies for mitigating the OSP.

IC.GA.50 and IC.GA.10 are used by the GA to create the database. IC.GA.50 and IC.GA.10 have 50 and 10 individuals, respectively. IC.Best10 consists in applying 10 sequences found by Purini and Jain [18], and consequently returns the best target code.

Approach	Sequence Selection	Program Representation	Feeding Strategy	Compilation Order	Maximum NoS
The Proposed Hybrid Approaches					
H.DHB.PC	Distributed	Hot	Batch	Poly-cBench	200
H.DHB.CP	Distributed	Hot	Batch	cBench-Poly	200
H.DHC.A	Distributed	Hot	Constant	Alphabetical	200
H.DFB.PC	Distributed	Full	Batch	Poly-cBench	200
H.DFB.CP	Distributed	Full	Batch	cBench-Poly	200
H.DFC.A	Distributed	Full	Constant	Alphabetical	200
H.CHB.PC	Centralized	Hot	Batch	Poly-cBench	200
H.CHB.CP	Centralized	Hot	Batch	cBench-Poly	200
H.CHC.A	Centralized	Hot	Constant	Alphabetical	200
H.CFB.PC	Centralized	Full	Batch	Poly-cBench	200
H.CFB.CP	Centralized	Full	Batch	cBench-Poly	200
H.CFC.A	Centralized	Full	Constant	Alphabetical	200
Machine Learning Approaches					
ML.DH	Distributed	Hot	–	–	10
ML.DF	Distributed	Full	–	–	10
ML.CH	Centralized	Hot	–	–	10
ML.CF	Centralized	Full	–	–	10
Iterative Compilation Approaches					
IC.GA.50	–	–	–	–	5 000
IC.GA.10	–	–	–	–	200
IC.Best10	–	–	–	–	10

Table 5. Strategies

We also evaluated four machine learning methods: ML.DH that selects sequences using the distributed strategy for the hottest function features; ML.DF that selects sequences using the distributed strategy for the full program features; ML.CH that selects sequences using the centralized strategy for the hottest function features; and ML.CF that selects sequences using the centralized strategy for the full program features.

5 EXPERIMENTS

The following subsections describe in detail the experimental results.

5.1 Quality of the Knowledge Database

The KD was generated from two executions of our GA, and it is presented in Figure 3.

The aforementioned figure presents a *violin plot*, which shows sequences created for each *microbenchmark*. In addition, the *violin plot* represents each LLVM

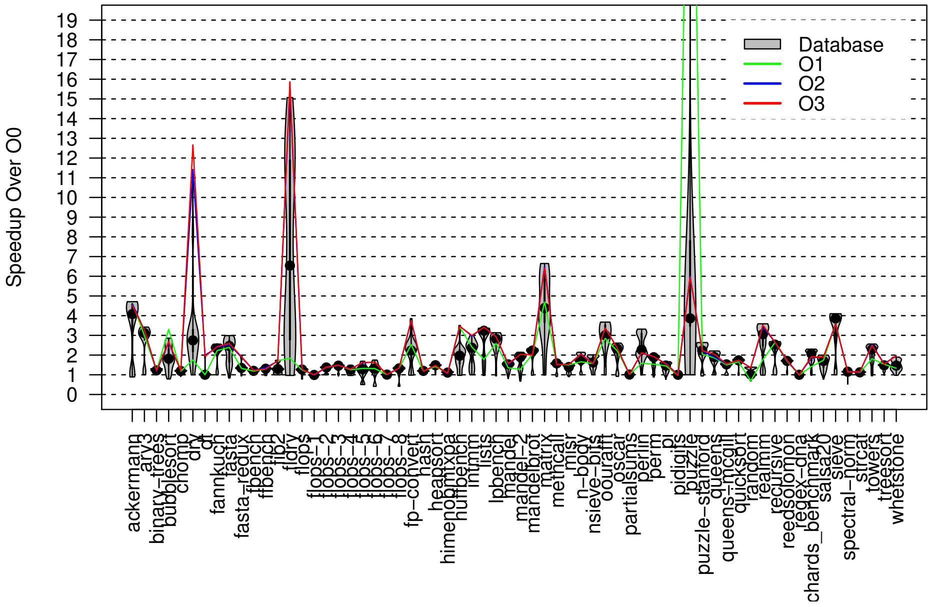


Figure 3. Knowledge Database

standard optimization level with lines. As shown in this figure, the best/most effective LLVM optimization level was superior to the GA in the following cases: *misr*, *bubblesort*, *dry*, *intmm*, *fldry*, *lists* and *whetstone*.

The genetic algorithm had a speedup rate slightly higher (not more than 3×) than the LLVM standard optimization levels. This indicates that although the base is not comprised of the best possible sequences, it can have sequences superior to the best/most effective LLVM optimization level (89.06 % of the programs) in terms of both quantity and quality.

In total, 23 410 sequences were generated. 25.18 % (5894) of these sequences were better, in terms of speedup over O0, than the most effective LLVM optimization level. The initial population can be comprised of these aforementioned sequences.

5.2 Performance

A summary of the results is presented in Table 6.

As shown above, "pure" or unaltered ML techniques had similar speedups in most cases, however their NPS is lower compared to distributed-selection-based hybrid approaches and IC. This does not apply for IC.Best10 approaches. ML.DH had the best/most effective speedups, however its NPS was worse compared to ML.DF.

A total of 59 programs were evaluated. ML.DH is superior to ML.DF in 35 programs, the latter overcame the former in just 24 programs. In addition, the best-case

Strategy	Speedup				NPS	NoS		
	Best	GMS	Worst	SDS		Max	AVG	Min
H.DHB.PC	4.47×	1.99×	1.06×	0.83×	46	110	51.09	10
H.DHB.CP	4.50×	1.99×	1.07×	0.79×	45	117	54.85	10
H.DHC.A	4.48×	1.97×	1,06×	0.83×	46	180	55.46	10
H.DFB.PC	6.00×	1.99×	1.06×	0.91×	45	140	50.14	10
H.DFB.CP	4.44×	1.96×	1.07×	0.75×	44	99	51.98	10
H.DFC.A	4.46×	1.95×	1.05×	0.78×	43	119	52.79	10
H.CHB.PC	4.42×	1.88×	1.00×	0.79×	38	150	50.53	10
H.CHB.CP	4.51×	1.85×	1.03×	0.70×	35	120	53.61	10
H.CHC.A	4.45×	1.87×	0.91×	0.77×	36	120	46.44	10
H.CFB.PC	4.20×	1.87×	1.02×	0.66×	36	149	52.98	10
H.CFB.CP	4.11×	1.85×	1.02×	0.66×	35	168	50.88	10
H.CFC.A	4.22×	1.88×	1.06×	0.66×	32	110	51.15	10
ML.DH	4.49×	1.91×	1.05×	0.77×	33	10	10	10
ML.DF	4.06×	1.90×	1.06×	0.68×	35	10	10	10
ML.CH	4.42×	1.84×	1.01×	0.72×	30	10	10	10
ML.CF	4.10×	1.82×	1.02×	0.63×	23	10	10	10
IC.GA.50	4.35×	2.083×	1.08×	0.83×	56	650	286.17	100
IC.GA.10	6.71×	1.930×	1.04×	0.92×	46	120	53.68	10
IC.Best10	3.75×	1.801×	1.05×	0.59×	24	10	10	10
O1	4.70×	1.69×	0.99×	0.63×	–	1	1	1
O2	4.37×	1.84×	1.03×	0.75×	–	1	1	1
O3	4.36×	1.84×	1.05×	0.76×	–	1	1	1

Best: the best result; GMS: geometric mean speedup; Worst: the worst result; SDS: standard deviation speedup; Max: maximun NoS; AVG: average NoS; Min: minimun NoS.

Table 6. Summary of experiments

scenario for ML.DH and ML.DF was 4.49× (gemm) and 4.06× (gemm), respectively. These results indicate that program characterization based on the hottest function provides a better initial solution to the solution adapter. In addition, another key element worth analyzing is ML. Distributed-selection-based approaches achieved a slightly higher NPS than Centralized-selection-based techniques. However, a thorough analysis revealed that ML strategies had a higher speedup than centralized-selection-based techniques in 9 programs (2mm, lame, covariance, doitgen, durbin, jacobi-2d, mvt, adpcm_d and CRC32). Finally, ML.DH had the best results in 2 cases.

Overall, IC.GA.50 had the best results. It had higher speedups than IC.GA.10 in 48 programs. Compared to other strategies, IC.GA.50 had higher speedups in 27 out of 59 programs. This result was highly anticipated because this strategy is the most aggressive; consequently, evaluating a high number of sequences. In addition, IC.GA.10 had higher speedups than IC.GA.50 in 11 programs (susan_e, bzip2e, jpeg_c, lame, correlation, doitgen, durbin, gesummv, mvt, trmm, and seidel-2d). Thus, this confirms that less aggressive IC techniques have satisfying results in some cases, and consequently surpass more aggressive IC strategies. Additionally, IC.GA.10 and IC.GA.50 had maximal speedups of 6.71× (durbin) and 4.35× (gemm), respectively. Furthermore, IC.GA.10 had the best results, in 4 programs, among all the strategies.

IC.Best10 surpassed all the other strategies in just 2 programs (bicg and rijndael_d). This result was also anticipated because the aforementioned strategy evaluates the same number of sequences. However, this strategy excels over ML in only 14 programs (adi, atax, bicg, correlation, gemver, gesummv, lu, dikstra, ghostscript, pgp_d, rijndael_d, rijndael_e, seidel-2d and trisolv).

An individual analysis reveals that the success rate of IC.Best10 does not increase significantly compared to ML.DH and ML.DF, and achieves 15 and 19 than the aforementioned strategies, respectively. The highest speedup reached by IC.Best10 was 3.75× (doitgen). However, this is the lowest speedup compared to other strategies including the LLVM optimization levels. Thus, we conclude and confirm that this strategy is the worst compared to others.

The hybrid approach reached its highest speedup by using H.DFB.PC. The speedup rate is 6.00× (durbin). The other strategies do not possess significant differences in terms of speedups. H.CHB.CP reached approximately half of the performance of the other hybrid strategies in 1 case (nussinov). Furthermore, H.DHB.PC and H.DHC.A had the highest value in the same case (bitcount). The centralized-selection based hybrid approach had worse performance than its distributed-selection based counterpart. This indicates that looking for sequences from different sources to obtain a more diversified population is an appealing option. Overall, the best-performing hybrid strategy was H.DHC.A because it had the highest speedup in 6 programs.

All strategies, except for IC.Best10 and both MLS's, had higher speedups than the optimization levels of LLVM. Specifically, IC.GA.50 had the best performance. These results were as expected, since IC.GA.50 is the most aggressive strategy in these experiments.

The behavior of the hybrid approach was altered for every strategy. However, the most important factor in improving performance is the initial solution. It is widely known that the initial selection of sequences is highly influential to the end results, and thus centralizing the selection of an initial solution is not beneficial. This can be confirmed because its performance was lower than IC.GA.10 in the majority of the programs. However, decentralizing the selection of an initial solution was better than IC.GA.10.

IC.GA.10 is appealing because it had a high geometric mean. However, it is approximately 3.35 % lower than the geometric mean of the hybrid approach. Nevertheless, it surpassed every strategy of the hybrid approach with centralized-selection-based techniques.

It is very important to highlight that the both MLF's approaches had speedups of approximately 1.03 % and 1.04 % lower than IC.GA.10. This result indicates that these methods have low-cost benefits.

The strategies can be categorized as follows:

1. Strategies that evaluate an unfixed number of sequences; and
2. Strategies that evaluate a fixed number of sequences.

IC.GA.50 was the best strategy among those in the first category. In addition, it had the largest number of explored sequences. Statistically, it evaluated 5.5 times more than the hybrid approach.

However, the hybrid approach and IC.GA.10 evaluated almost the same number of sequences. This indicates that the hybrid approach can reach higher speedups than a "pure" or unaltered IC strategy. In addition, GAs with a well selected population is an improvement over GAs with initial random populations.

IC.Best10 and ML.CF were the worst-performing strategies among those in the second category. They had the worst coverage, and does not even reach speedups obtained by optimization levels of LLVM (O2 and O3). Considering the number of evaluated sequences, ML.DH and ML.DF had acceptable improvements, however its coverage was low. Both of these strategies are an improvement over optimization levels of LLVM (O1, O2 and O3).

IC.GA.50 is superior in terms of NPS, and covers 95 % of the programs. Furthermore, IC.GA.10 covered 78 % of the programs. The distributed-selection based hybrid approach covered 73 % (at its worst) and 78 % (at its best) of the programs. The centralized-selection based hybrid approach covered 54 % (at its worst) and 64 % (at its best). ML.DH and ML.DF covered 56 % and 59 % of the programs, respectively, and ML.CH covered 51 % of the programs. Finally, IC.Best10 and ML.CF were the two worst cases in covering, achieving 41 % and 38 %, respectively. This indicates that aggressive exploration strategies increase the coverage; consequently, covering the majority of all tested programs.

The response time is the total time spent in sequence-selection and improvement phases, however the time spent on creating the database is ignored because this process is executed only once and it will not be necessary for future compilations. ML and IC.Best10 had the lowest response times. In addition, they are also the worst-performing strategies, as discussed previously.

However, IC.GA.10 and IC.GA.50 spent an average time of 2 hours and 18 minutes, and 14 hours and 30 minutes, respectively. Finally, the hybrid approach spent 3 hours and 35 minutes finding a solution for each program.

The hybrid approach outperforms IC.GA.10 by 3.47 % (in terms of speedup); consequently, consuming 56 % more of the time spent by IC.GA.10. In addition, IC.GA.50 outperforms IC.GA.10 by up to 7.93 %, and thus consuming 530 % more time than IC.GA.10.

Another important fact to be observed is that there is a soft relation between Standard Deviation and GMS. The approaches that reach low GMS tend to provide low Standard Deviations, and the approaches that reach high GMS tend to provide higher Standard deviation. This indicates that for some programs, the effort of iterative compilation, even with a selected start sequence set, may not achieve high improvements.

These results indicate that the most-time-consuming strategies reach the best speedups, however there are strategies (such as the hybrid approach) that increase speedups by slightly increasing the time consumption.

5.3 Different Hardware Platforms

Constant feeding experiments were executed on different hardware platforms as well. However, these experiments consisted of both cBench and Polybench benchmarks. Figure 4 presents the GMS of both the Core-i7 architecture (described in Section 4.1) and the following hardware platform: Intel Xeon E5504 processor with a frequency of 2.00 GHz, 4 MB of cache and 24 GB of RAM. The experiment performed on the Intel Xeon architecture considered an already-established KD. Thus, the results are based on the same database created on the Core-i7 architecture.

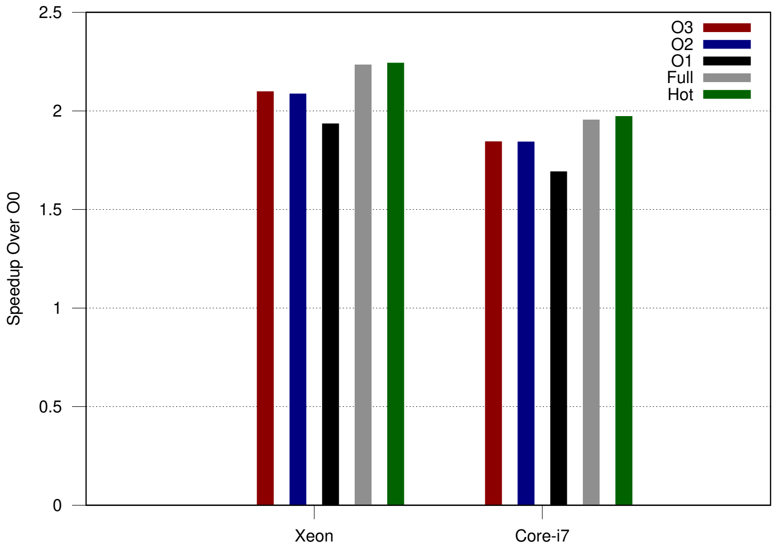


Figure 4. Results on different hardware platforms

Intel's Xeon processor had better results than the Core-i7. Compared to the best compiler optimization level (03), the hybrid approach gained performance by up to 6.49 % and 6.47 % on the Core-i7 and Xeon architecture, respectively. The performance gain at 02 and 01 optimization levels were very similar, varying no more than 1 %. The results indicate that, for both architectures, the performance gain of the hybrid approach was approximately the same proportion.

A thorough analysis of the results reveals the following conclusions.

- It is possible to obtain effective speedups using a database created on a different hardware platform.
- Representing programs based on their hot functions is an efficient strategy to reduce the performance loss when the data-set is altered, as well as the hardware platform.
- Using a hybrid approach is a smart and effective strategy to mitigate the OSP, regardless of the data-set or hardware platform.

5.4 Different Input Sets

An additional experiment was conducted with different input sets. These experiments were performed only with constant feeding and cBench because of the limited availability of several datasets. In addition, they are based only on distributed-selection based strategies. Figure 5 shows the results for these experiments. In addition, Table 7 presents speedups for each different input.

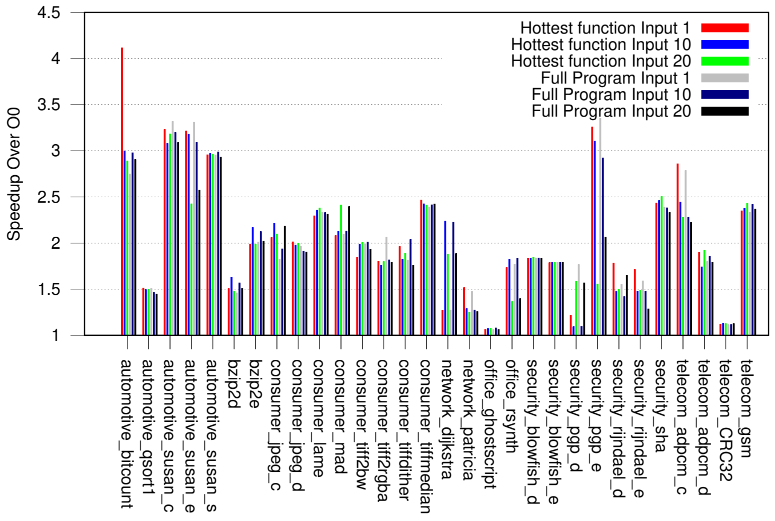


Figure 5. Different input sizes

Based on the aforementioned results, the performance was influenced by altering the data-sets, as reported in the literature [3]. The performance loss was only up to 4.86 %, regardless of the program characterization.

It is important to specify that only static features are considered when extracting good/effective sequences from the database. This process does not consider the

Input	Hot Function			Full Program		
	Best	GMS	Worst	Best	GMS	Worst
1	4.12×	1.99×	1.06×	3.35×	1.97×	1.06×
10	3.18×	1.96×	1.07×	3.20×	1.95×	1.08×
20	3.18×	1.90×	1.08×	3.09×	1.89×	1.06×

Best: the best result; GMS: geometric mean speedup; Worst: the worst result

Table 7. Speedups for each different input

program behavior when data-sets are exchanged. However, this process does not require the program execution to extract features, which will affect the system response time. Therefore, a performance loss of up to 4.86 % is appealing compared to the cost of executing a program.

6 RELATED WORKS

Zhou and Lin [26] used a genetic algorithm called NSGA-II to investigate multi-objective compilations, and thus compared randomly-selected sequences. The compiler used for this research study was GCC. The optimization levels -O1, -O2, -O3, -Os were selected for the experiments, thus totaling 54 optimizations and a search space size of 2^{54}. A set of 10 cBench programs were used, and the goal was to optimize the code size and run-time. The hyper-volumes created by NSGA-II had the best results, and random sequences had better performance than GCC optimization levels. A thorough analysis revealed that NSGA-II had the best results in terms of run-time and code size.

Jantz and Kulkarni [5] proposed an approach reducing the search-space of the optimization sequences. This is done by exploring dependencies between the optimizations. Applying cleanup phases such as dead-code elimination and dead-assignment elimination, which does not have much interaction with other optimizations, some optimizations can be removed from the search space and applied after each optimization.

The works presented by [5] and [26] focus on a pure IC strategy. They does not use a combination between ML and IC strategies as our work.

Malik [11] uses the concept of an histogram based on a data flow graph to establish the similarity between programs. This histogram was built considering the distance to a *sink* node (node without successor nodes) and a *root* node (node without predecessor nodes). Thus, Malik showed how data flow graphs can be beneficial to characterize programs using static information and SVM. However, these sets are not adapted to recently-established programs, which could provide greater benefits.

Park et al. [15] introduced a model to characterize programs based on *graph-model representation*, collecting instruction information for each node. In addition to the proposed model, the authors also implemented other techniques to characterize programs, which can be categorized into dynamic or static representations. The former requires program execution for classifications and the latter only needs to analyze the source code. The results showed that control flow graphs had better performance among other strategies.

Jantz and Kulkarni [6] addressed the Optimization Selection Problem in just-in-time compilers using Java Virtual Machine (JVM). This paper does not consider the order of optimizations because HotSpot (in JVM) does not require it. First, the analysis was done by recompiling the methods, removing only one optimization at a time from the standard sequence of JVM (used as *baseline*). Thus, it was possible to make the following observations: most optimizations do not have negative impacts for several methods of the program. However, some methods can hinder the performance more frequently; and most optimizations have a small individual influence on the performance. In addition, they also used a logistic regression thechnique to predict sequences and then compare it to an GA implementation.

Lima et al. [10] used machine learning to improve power and performance efficiency during compilation. They showed that sacrificing performance was unnecessary to reduce power consumption.

Junior and da Silva [7] evaluated the performance of different case-based reasoning configurations, which use dynamic and static features to find good/effective optimization sequences. The authors presented a measure of similarity among other strategies to build past experiences. The proposed algorithm is divided into two phases: an *offline* phase; and an *online phase.* The former creates sequences for prior experiences of the knowledge database. This phase can be done randomly or using a meta-heuristic. The *online* phase is based on case-reasoning, and thus all prior program experiences are analyzed, and its sequences are filtered. Afterwards, the input sequence is compiled with the given number of analogies. These sequences are extracted from the most similar program. The results show that the approach used to create the knowledge database influences the results.

The works presented by [7, 10, 11, 15, 6] uses IC to generate a knowledge database and also use ML to select a number of sequences and then pick up the best between them. Instead of the instruction: select the best one, our work adapts the sequence after ML phase.

Martins et al. [12] implemented a clustering approach, transforming code from program functions to symbolic representations, to mitigate the OSP. These representations refer to the DNA of the program, where *tokens* of the source code are transformed into characters. The clustering approach starts by extracting DNA from the program function. Thus, the distance matrix for programs is calculated using the normalized compression distance algorithm. This matrix will serve as a basis to build a tree topology in which the clustering algorithm will be capable of finding possible clusters. Finally, all optimizations are included in the reduced search space. The results showed that reducing the search space is significant and highly beneficial to the performance. This work differs from our because the initial population of GA is generated by the optimization of the reduced search space, while our work select the initial population to the program based on SVM prediction.

Purini and Jain [18] proposed a search strategy slightly different than machine learning and iterative compilation. The objective of this strategy was to build a set of good optimization sequences, where for each class of programs there exists a sequence of optimizations that reach a satisfying performance. While in this work the goal is to search a generic optimization sequence that fits well in different programs, our work focus on search specific optimization sequences.

Tartara and Crespi Reghizzi [21] proposed a continuous learning approach that does not require prior knowledge to create optimization sequences. These sequences are represented as mathematical formulas. Every formula can be represented using a grammar based on certain rules of binary operations, Boolean and numeric values, if-expressions, comparison operators, arithmetic and Boolean operators, and program features. This approach uses a knowledge database for storing and extracting sequences, however this information is randomly-generated if the number of sequences is insufficient. The results showed that converging for good performance does not require several executions, and in some cases it has better values than the highest optimization level in both compilers. This work is different from our work, because it does not have an *offline* phase. In our work we used the training group component that does the offline phase.

7 CONCLUSION

The selection of optimization sequences can greatly impact the run-time of a program. In addition, the OSP depends heavily on the hardware architecture.

This paper proposes a hybrid approach for mitigating the OSP. Since this problem is complex, this approach uses an ML approach (SVM) to feed the initial solution of a GA and then search for good/effective solutions. The results showed that the hybrid approach achieved significant improvements over "pure" or unaltered ML and IC strategies, achieving speed-ups over 2.00×, 1.93× and 2.08× with H.DHB.PC, IC.GA.10 and IC.GA.50, respectively. Machine learning strategies achieved the lowest speed-ups among the aforementioned approaches, 1.91.

It is extremely important to highlight that the strategy used for the initial sequences had the highest overall influence, and we conclude that distributed-selection is the best approach for selecting the initial solution. In addition, the experiments showed that the approach is portable, because it can be transferred from one architecture to another without a performance loss. Furthermore, the hybrid approach is beneficial because its solutions improve over time, whereas ML and IC do not provide this advantage. IC strategies need to restart the process on every occasion and ML cannot create new sequences.

Future works include investigating the influence of different schemes to generate the KD, and additional ML approaches to predict the initial population. In addition, instantiating the hybrid approach with more aggressive solution adapters can be proposed in the future.

REFERENCES

[1] Aho, A. V.—Lam, M. S.—Sethi, R.—Ullman, J. D.: Compilers: Principles, Techniques and Tools. Prentice Hall, 2006.

[2] Cavazos, J.—Fursin, G.—Agakov, F.—Bonilla, E.—O'Boyle, M. F. P.—Temam, O.: Rapidly Selecting Good Compiler Optimizations Using Performance Counters. Proceedings of the International Symposium on Code Generation and Optimization (CGO '07), IEEE, 2007, pp. 185–197, doi: 10.1109/cgo.2007.32.

[3] Chen, Y.—Huang, Y.—Eeckhout, L.—Fursin, G.—Peng, L.—Temam, O.—Wu, C.: Evaluating Iterative Optimization Across 1000 Datasets. SIGPLAN Notices, Vol. 45, 2010, No. 6, pp. 448–459, doi: 10.1145/1809028.1806647.

[4] Daud, S.—Ahmad, R. B.—Murthy, N. S.: The Effects of Compiler Optimisations on Embedded System Power Consumption. International Journal of Information and Communication Technology (IJICT), Vol. 2, 2009, No. 1-2, pp. 73–82, doi: 10.1504/ijict.2009.026431.

[5] Jantz, M. R.—Kulkarni, P. A.: Exploiting Phase Inter-Dependencies for Faster Iterative Compiler Optimization Phase Order Searches. 2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013, pp. 1–10, doi: 10.1109/cases.2013.6662511.

[6] Jantz, M. R.—Kulkarni, P. A.: Performance Potential of Optimization Phase Selection During Dynamic JIT Compilation. SIGPLAN Notices, Vol. 48, 2013, No. 7, pp. 131–142, doi: 10.1145/2517326.2451539.

[7] Queiroz Junior, N. L.—da Silva, A. F.: Finding Good Compiler Optimization Sets – A Case-Based Reasoning Approach. Proceedings of the 17th International Conference on Enterprise Information Systems, 2015, Vol. 2, pp. 504–515, doi: 10.5220/0005380605040515.

[8] Queiroz Junior, N. L.—Rodriguez, L. G. A.—da Silva, A. F.: Combining Machine Learning with a Genetic Algorithm to Find Good Compiler Optimizations Sequences. Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS), 2017, Vol. 3, pp. 397–404, doi: 10.5220/0006270403970404.

[9] Lattner, C.—Adve, V.: LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO '04), 2004, pp. 75–86, doi: 10.1109/cgo.2004.1281665.

[10] de Lima, E. D.—de Souza Xavier, T. C.—da Silva, A. F.—Ruiz, L. B.: Compiling for Performance and Power Efficiency. Proceedings of the 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2013, pp. 142–149, doi: 10.1109/patmos.2013.6662167.

[11] Malik, A. M.: Spatial Based Feature Generation for Machine Learning Based Optimization Compilation. 2010 Ninth International Conference on Machine Learning and Applications (ICMLA), 2010, pp. 925–930, doi: 10.1109/icmla.2010.147.

[12] Martins, L. G. A.—Nobre, R.—Cardoso, J. M. P.—Delbem, A. C. B.—Marques, E.: Clustering-Based Selection for the Exploration of Compiler Optimization Sequences. ACM Transactions on Architecture and Code Optimization, Vol. 13, 2016, No. 1, Art. No. 8, 28 pp., doi: 10.1145/2883614.

[13] Muchnick, S. S.: Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[14] Namolaru, M.—Cohen, A.—Fursin, G.—Zaks, A.—Freund, A.: Practical Aggregation of Semantical Program Properties for Machine Learning Based Optimization. Proceedings of the 2010 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, ACM, 2010, pp. 197–206, doi: 10.1145/1878921.1878951.

[15] Park, E.—Cavazos, J.—Alvarez, M. A.: Using Graph-Based Program Characterization for Predictive Modeling. Proceedings of the Tenth International Symposium on Code Generation and Optimization, ACM, 2012, pp. 196–206, doi: 10.1145/2259016.2259042.

[16] Patterson, D. A.—Hennessy, J. L.: Computer Organization and Design: The Hardware/Software Interface. Fourth Edition. The Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[17] Pedregosa, F.—Varoquaux, G.—Gramfort, A.—Michel, V.—Thirion, B.—Grisel, O.—Blondel, M.—Prettenhofer, P.—Weiss, R.—Dubourg, V.—Vanderplas, J.—Passos, A.—Cournapeau, D.—Brucher, M.—Perrot, M.—Duchesnay, E.: Scikit-Learn: Machine Learning in Python. The Journal of Machine Learning Research, Vol. 12, 2011, pp. 2825–2830.

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

[19] Scott, M. L.: Programming Languages Pragmatics. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2009.

[20] Sebesta, R. W.: Concepts of Programming Languages. Addison Wesley, San Francisco, CA, USA, 2009.

[21] Tartara, M.—Crespi Reghizzi, S.: Continuous Learning of Compiler Heuristics. ACM Transactions on Architecture Code Optimization, Vol. 9, 2013, No. 4, Art. No. 46, 25 pp., doi: 10.1145/2400682.2400705.

[22] GNU Compiler Collection. 2017, `http://gcc.gnu.org`.

[23] Intel C Compiler. 2017, `https://software.intel.com/en-us/c-compilers`, doi: 10.1145/62297.62412.

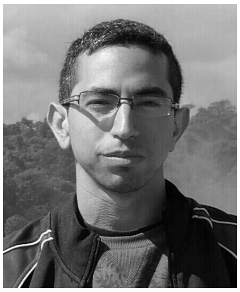
[24] The LLVM Compiler Infrastructure. 2017, `http://llvm.org`.

[25] Wu, Y.—Larus, J. R.: Static Branch Frequency and Program Profile Analysis. Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO 27), ACM, 1994, 11 pp., doi: 10.1145/192724.192725.

[26] Zhou, Y.—Lin, N.: A Study on Optimizing Execution Time and Code Size in Iterative Compilation. 2012 Third International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA), 2012, pp. 104–109, doi: 10.1109/ibica.2012.46.

Nilton Luiz Queiroz Junior is Professor in the Department of Informatics of the State University of Maringá located in Brazil. He is lecturing undergraduate courses. He received his Bachelor and Master degrees in computer science from the State University of Maringá, Brazil in 2014 and 2016, respectively. His research interests include parallel programming and compile techniques.

Anderson Faustino da Silva is Professor in the Department of Informatics of the State University of Maringá located in Brazil, lecturing undergraduate and graduate courses. He received his Bachelor's degree in computer science from the State University of West Paraná, Brazil in 2000. He then received his Master and Ph.D. degrees in systems engineering and computer science from the Federal University of Rio de Janeiro, Brazil in 2003 and 2006, respectively. His research interests include parallel programming and compile techniques.



Luis Gustavo Araujo Rodriguez is currently Ph.D. student in computer science at the University of São Paulo in Brazil. He received his Bachelor's degree in computer science from the Catholic University of Honduras in 2013. He then received his Master's degree in computer science from the State University of Maringá, Brazil in 2016. His research interests include high performance computing, parallel programming, and compiler.

Computing and Informatics, Vol. 39, 2020, 1148–1171, doi: 10.31577/cai_2020_6_1148

OPEN HYBRID MODEL: A NEW ENSEMBLE MODEL FOR SOFTWARE DEVELOPMENT COST ESTIMATION

Amin Moradbeiky, Vahid Khatibi Bardsiri

Department of Computer Engineering, Faculty of Science
Kerman Branch, Islamic Azad University
Kerman, Iran
e-mail: `ampayam@yahoo.com, kvahid2@live.utm.my`

Mehdi Jafari

Department of Electrical Engineering, Faculty of Engineering
Kerman Branch, Islamic Azad University
Kerman, Iran
e-mail: `mjafari@iauk.ac.ir`

Abstract. Given various features of a software project, it may face different administrative challenges requiring right decisions by software project managers. A major challenge is to estimate software development cost for which different methods have been proposed by many researchers. According to the literature, the capability of a proposed model or method is demonstrated in a specific set of software projects. Hence, the aim of this study is to present a model to take advantage of the capabilities of various software development cost estimation models and methods simultaneously. For this purpose, a new model called "open hybrid model" was proposed based on the firefly algorithm. The proposed model includes an extensible bank of estimation methods. The model also includes an extensible bank of rules to describe the relation between existing methods. Considering project conditions, the proposed model tries to find the best rule for combining estimation methods in the methods bank. Three datasets of real projects were used to evaluate the precision of the proposed model, and the results were compared with those of other 11 methods. The results were compared based on performance parmeters widely used to show the accuracy and stability of estimation models. According to the results, the open hybrid model was able to select the most appropriate methods present in the methods bank.

Keywords: Development cost estimation, firefly algorithm, software project

1 INTRODUCTION

According to the literature [30, 36, 5], various methods for software cost estimation can be classified as algorithmic and non-algorithmic strategies. Both strategies can be useful depending on the type of software projects. The efficiency will be increased by proper identification of the requirements of each method. Each method has its own advantages and disadvantages.

Moreover, different approaches have been recently used for software development cost estimation. For instance, back propagation learning algorithms on a multilayer perceptron and the genetic programming (GP) have been proposed for this purpose [24]. Reddy et al. [33] used the multi-objective particle swarm optimization (MOPSO) to develop a software cost estimation model which outperformed the standard constructive cost model (COCOMO). Andreou and Papatheocharous [2] employed fuzzy decision trees for software cost estimation.

A set of cost estimation methods are classified as algorithmic strategies. These methods employ mathematical models for estimating project costs. These models are defined as a function of cost factors.

Several algorithmic methods such as linear and nonlinear models, Putman's model, Seer-Sem model, function point model, Bailey and Basili model, Aron model, Doty model, COCOMO (constructive cost model), and COCOMO II have been proposed so far. COCOMO has been used by many researchers and thus is further discussed in this section.

COCOMO was first proposed by Boehm in 1981 as an empirical model by collecting data of different real-world software projects. The collected data are then analyzed to obtain certain formulas matching observations. In addition, Boehm et al. analyzed the developed version of COCOMO with more capabilities than the earlier version and released it as COCOMO II [7, 8, 9].

As mentioned earlier, algorithmic methods based on one or more mathematical formulas lack sufficient flexibility. Consequently, non-algorithmic models have been proposed for cost estimation.

Unlike algorithmic methods, non-algorithmic methods are based on analytical comparison and inference. The use of non-algorithmic methods requires accurate information on prior projects. Non-algorithmic methods make estimations by analyzing prior datasets without employing any specific relation or equation. The most common methods for estimating software criteria in non-algorithmic methods include exhaustive search, comparison, trial, error and inference. Some studies in this field are reviewed below:

Cuadrado-Gallego et al. [10] analyzed and compared machine learning and expert judgment methods that have been extensively used for cost estimation of software projects. To this end, they compared and pointed out advantages and disadvantages of seven machine learning methods including neural networks, fuzzy logic, comparative analysis, decision trees, case-based reasoning (CBR) and rule-based reasoning and hybrid systems. According to their results, CBR outperformed other methods. In fact, this method could be successful in the absence of statistical relationships.

Wen et al. [41] systematically analyzed machine learning models for software cost estimation from four perspectives. They analyzed different studies completed between 1991 and 2010 in terms of machine learning, estimation precision, comparison models and estimation field. Different machine learning models have their own advantages and disadvantages; thus, they can be employed in different fields. Huang et al. [17] analyzed preprocessing as a major step of machine learning in software project cost estimation. They aimed at empirical evaluation of the effect of data preprocessing on machine learning methods for software cost estimation. Bardsiri and Hashemi [6] employed the correlation analysis approach to evaluate the efficiency and precision of five major machine learning methods for software project cost estimation. They also analyzed the effect of feature selection on the estimation precision.

Idri et al. [18] classified the papers published on CBR cost estimation by IEE, ACM, ScienceDirect, and Google Scholar from 1990 to 2012 in terms of methodology, type and technique. According to their findings, ABE methods outperformed other techniques, especially when combined with the fuzzy logic or genetic algorithm. Sigweni and Shepperd [37] systematically reviewed and evaluated feature weighting techniques in the ABE method for software development cost estimation. They analyzed different aspects of each technique in addition to advantages, disadvantages and efficiency on different datasets. For this purpose, they comprehensively reviewed all papers in this area published from 2000 to 2014.

González-Ladrón-de-Guevara et al. [15] analyzed software development cost estimation in the ISBSG dataset. They reviewed different studies on the ISBSG dataset (2000–2013) to determine those ISBSG variables used in estimations and to examine the effect of ISBSG variables on the development of cost estimation models. They also determined dependent or independent variables. Out of 71 ISBSG variables, 20 variables were used as independent variables in most studies. Their findings can help other researchers in selecting appropriate variables in cost estimation models. General algorithms can be used to develop a model from databases. The EM clustering algorithm [14] was combined with the database segmentation method leading to an exact model and estimate for size-effort criterion and standard quality criterion. The estimation criterion improved the accuracy of expected parameters in each segment using EM clustering algorithm and local regression.

The use case points-activity based costing (UCPabc) method and function points (FP) can be employed for software development cost estimation. Azzeh and Nassif used UCPabc for software development cost estimation [4]; however, Dewi et al. used FP for this purpose [12]. Dewi et al. compared FP and UCPabc and found that the latter was much more accurate than the former [11].

Pospieszny et al. [31], Mensah et al. [26], Puspaningrum and Sarno [32], Moosavi and Bardsiri [27], Wani and Quadri [40], Arora and Mishra [3], Abnane et al. [1], and Khuat and Le [20] respectively employed a linear method integrated with neural networks, multivariate regression model, artificial neural networks integrated with the harmony algorithm, neural networks integrated with the fuzzy method, neural networks, the fuzzy model and the artificial bee colony algorithm for software development cost estimation.

Rastogi et al. [34] evaluated software cost estimation techniques and models extensively. They compared and classified different methods and considered cost estimation techniques resulting in a higher precision as a selection criterion. According to their results, all cost estimation techniques have their own advantages and disadvantages. They also believe that there is no single method which can be accepted by all researchers. Therefore, a combination of different methods should be employed to achieve realistic cost estimation. Shekhar and Kumar [36] reviewed and analyzed different software cost estimation techniques and models. For this purpose, they analyzed advantages and disadvantages of different methods and concluded that no single method could be used as the best cost estimation method and a more accurate estimate could be achieved by combining different methods. Pandey [30] classified software project cost estimation methods into parametric and nonparametric models by analyzing different cost estimation techniques and addressing their advantages and disadvantages. Khatibi and Jawawi [19] reviewed and analyzed different software cost estimation methods from different points of view. Each of the cost estimation methods can be employed efficiently in certain projects and situations. The performance of any estimation method depends on different parameters such as project complexity, project span, etc.

Ensemble models have been presented for estimating software development cost by another group of scholars. In these models, independent methods attempt to predict software development cost separately. The estimates from different methods are then combined with a method to calculate the final estimate. Various estimators and combinators have been used in the literature. Studies by Wu et al. [42], Kocaguneli et al. [22], Elish [13], and Hsu et al. [16] can be noted in this regard.

The reviewed studies suggest a single method or model can offer high estimation accuracy only in a limited number of datasets. Ensemble models allow benefiting from the advantages of different models and methods at the same time. A review of ensemble models reveals they are based on a limited number of estimators and combiners. Further, a survey of previous ensemble models shows using the same combiner fails to produce the most accurate estimations in all datasets. This study proposes an ensemble model that is not dependent on a set number of estimators and combiners, and enabling it to combine its estimators intelligently and by the best approach.

This article is organized as follows: Section 2 describes the background and related work. Section 3 describes the firefly algorithm. Different criteria are introduced to evaluate the precision of the model proposed in Section 4. The proposed model and its evaluation method are introduced in Section 5 and Section 6, respectively. The datasets of real projects introduced in Section 7 are used for testing. The results of testing the proposed model are compared to those of other 11 methods to determine the superiority of the proposed model. These 11 methods are presented in Section 8. Section 9 presents the results of testing the proposed model. The results are analyzed statistically in Section 10. Conclusions and future work are reported in Section 11.

2 BACKGROUND AND RELATED WORK

Ensemble models can be used in various data mining fields. Malgonde and Chari [25] used an ensemble model for estimating agile software development cost. Silva et al. [38] combined data mining techniques for predicting the export potential of a company. In an article on network security, Ochieng et al. [28] described identification of worms by an ensemble model. Salehi et al. [35] proposed an ensemble model by data mining techniques for cancer detection.

Various ensemble models have also been provided for software development cost estimation. According to the literature on software development cost prediction methods, ensemble models which include multiple predictors as a peer prediction model give more accurate results than each of the individual methods [22]. Ensemble models rely on multiple methods so that the inability of a method in providing an accurate estimate can be compensated by the accurate estimates provided by other methods and this is the main reason behind the success of these models [22].

Figure 1 shows the architecture of ensemble models. In general, ensemble models consist of two important parts. The first part includes a set of individual estimation models or methods ($f_{1 \ldots m}$). The second part includes a set of various methods for combining the estimations ($C_{1 \ldots k}$) obtained from estimators in the first part. Each combinator calculates an independent estimation. Accordingly, one can conclude that if Part 1 or 2 or both include a diverse and accurate set of various models or methods, the final model will be able to provide exact estimates in different datasets.

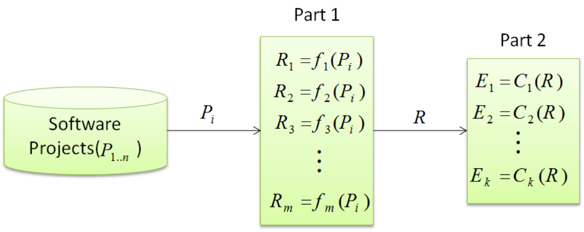


Figure 1. Architecture of ensemble models

Wu et al. [42] proposed an ensemble model for estimation of software development cost. For this purpose, they combined estimates obtained from different CBR methods by four linear combination techniques, namely median combination, mean combination, weighted mean combination (WMC) and outperformance combination (OC). The proposed model in their study was tested on two datasets. According to the results of these two datasets and based on the evaluation criterion MMRE, the WMC combination techniques provided more accurate results in one of the datasets, whereas the other combination technique (mean) led to more accurate results in the other dataset. The results of tests based on the evaluation criterion MdMRE also confirmed this point.

Elish [13] proposed an ensemble model for software development cost estimation. He combined the estimates obtained from three different methods by eight combination methods. The model proposed by Elish was tested by different datasets. According to the results, different combination methods led to best results based on the same evaluation criterion used in various datasets.

Hsu et al. [16] developed an ensemble model consisting of COCOMO, linear regression, CBR, grey relational analysis and artificial neural networks estimators. Equally weighted combination, median weighted combination and weighted adjustment based on a criterion were used for combining the estimations. The final model was tested by different datasets. The results obtained from different combination methods in various datasets indicated that no certain combination method was able to obtain the best results in all datasets.

Song et al. [39] provided an ensemble model of five machine learning methods for estimating software development cost. Kultur et al. [23] proposed an ensemble model using neural networks. Pahariya et al. [29] provided an ensemble model in which the results from computational intelligence techniques are combined by three different methods to calculate the final estimate.

According to the literature, no certain combination method was able to calculate the most accurate estimation in the different datasets, which raises the question whether it is possible to provide an intelligent model to detect the ability of different estimation methods in various datasets and select the best combination method.

The accuracy of ensemble estimation models is dependent on the accuracy and diversity of models and methods in the 1 and 2 parts (Figure 1, the architecture of ensemble models). According to the literature, the ensemble model is dependent on a certain and limited set of estimator and combinator methods. On the other hand, there are very diverse estimation and combination methods and new methods are still added to this set. The important question raised here is whether a proposed hybrid model can generate accurate results regardless of accuracy obtained from combination or estimation methods used in its parts of 1 and 2 (Figure 1, the architecture of ensemble models). Simply speaking, is it possible to eliminate factors limiting the accuracy of the final model due to limitations of models or methods used in the 1 or 2 part (Figure 1)? The model proposed in this article is able to eliminate the above-mentioned limitations.

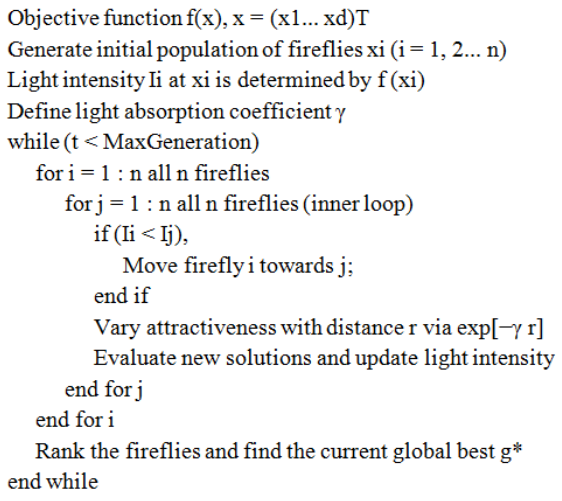
3 FIREFLY ALGORITHM

Firefly algorithm (FA) was first introduced by Xin-She Yang in 2009 [43]. FA is a meta-heuristic optimization algorithm that imitates the social behavior of fireflies flying in the tropical and temperate summer sky. FA algorithm has been used based on three principles:

1. Each firefly is capable of attracting other fireflies.
2. Attraction of each firefly depends on the level of its light so that the one with more light attracts a firefly with less light. If no firefly has more light, one is selected randomly.
3. The light of each firefly is under the influence of its distance from the goal.

As illustrated in Figure 2, each firefly generates a random solution. Then, some parameters as light intensity, primary attractiveness, and absorption coefficient are defined. Then, the most brilliant firefly is selected. Fireflies move toward a more brilliant firefly. Moving toward each other, fireflies light decreases and their attractiveness varies. Next, the best firefly is picked up for repetitive cycle according to an objective function. This process continues until the end condition is done.

The firefly attractiveness is due to its light determined by the objective function from the problem. As the most optimized method, the I light of firefly in the specific location of X can be selected as $I(x) \propto f(x)$. β is the relative attractiveness seen by each firefly. Therefore, it would change with r_{ij} distance between i firefly and j firefly. Additionally, light intensity decreases taking distance from the source. Light is absorbed in media. Therefore, attractiveness varies as its ratio varies.



```
Objective function f(x), x = (x1... xd)T
Generate initial population of fireflies xi (i = 1, 2... n)
Light intensity Ii at xi is determined by f (xi)
Define light absorption coefficient γ
while (t < MaxGeneration)
   for i = 1 : n all n fireflies
      for j = 1 : n all n fireflies (inner loop)
         if (Ii < Ij),
            Move firefly i towards j;
         end if
         Vary attractiveness with distance r via exp[−γ r]
         Evaluate new solutions and update light intensity
      end for j
   end for i
   Rank the fireflies and find the current global best g*
end while
```

Figure 2. Pseudocode of firefly algorithm [43]

Each firefly owns its specific attractiveness

$$\beta(r) = \beta_0 e^{-\gamma r^m}, m \geq 1. \tag{1}$$

The distance between two fireflies of i and j is calculated through following formula

$$r_{ij} = |x_i - x_j| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2}. \tag{2}$$

The movement of fireflies attracting more fireflies is calculated through the following formula

$$x_i = x_i + \beta_0 e^{\gamma r_{ij}^2}(x_j - x_i) + \alpha \left(rand - \frac{1}{2} \right). \tag{3}$$

Five parameters in the FA should be set for optimization purposes generally called FA configuration for problem solving. The settings of these parameters vary with the problem under study and expected objectives of the algorithm [43]. These settings determine the behavior of the algorithm during problem solving. The FA parameters are as follows:

1. N represents the number of fireflies used for problem solving.
2. *MaxGeneration* indicates the number of iterations.
3. *Alpha* (α) is a coefficient within the range $[0, 1]$ and is multiplied by the random number.
4. *Betamin* (β min) represents the minimum Beta (β) value, which is indicative of attractiveness of the light source.
5. *Gamma* (γ) is determined considering attractiveness variations. This parameter plays a key role in convergence rate and the behavior of FA.

A hyper-parameters tuning step was used for precise FA conguration. In this step, the best α and γ were obtained in the $[0, 1]$ and $[0.1, 20]$ ranges, respectively, by a comprehensive trial and error process.

4 ESTIMATION ERROR DETERMINATION EQUATIONS

Specific metrics were employed in this study to calculate the estimation error. These metrics have been used in several studies to compare the research results with those of other similar studies. The metrics include relative error (RE), magnitude of relative error (MRE), mean magnitude of relative error (MMRE), median magnitude of relative error (MdMRE) and prediction percentage (Pred), shown by metrics (4), (5), (6), (7) and (8):

$$\mathrm{RE} = \frac{\mathrm{Estimate} - \mathrm{Actual}}{\mathrm{Actual}}, \tag{4}$$

$$\mathrm{MRE} = \frac{|\mathrm{Estimate} - \mathrm{Actual}|}{\mathrm{Actual}}, \tag{5}$$

$$\mathrm{MMRE} = \mathrm{Mean}(\mathrm{MRE}), \tag{6}$$

$$\mathrm{MdMRE} = \mathrm{Median}(\mathrm{MRE}), \tag{7}$$

$$\mathrm{PRED}(X) = \frac{A}{N}. \tag{8}$$

5 PROPOSED MODEL

Different methods have been proposed for estimating important project parameters such as cost. According to the literature, the methods can only operate properly in limited datasets because of dependence on a certain estimation technique. According to this point, the challenge addressed in this study was to determine in which datasets a method would operate more successfully, to what extent it would be successful and how it would be possible to employ its high precision. For this purpose, a new model called "open hybrid model" was proposed based on the FA. It is also possible to add different estimation methods to the methods bank of the proposed model.

5.1 The Open Hybrid Model

The proposed model in this study includes a bank of various estimation methods. The main idea of the model is that an estimator machine, based on a specific method such as the ABE, is capable of proper cost estimation only in few datasets. Accordingly, the first objective of the proposed model is to test the precision levels of different methods for estimating a set of projects. Then it intends to allocate a specific value to each method with respect to its precision. This value indicates the effectiveness of a method on the final estimation. The model first divides the datasets of projects into basic, training and testing projects. The proposed model operates in two stages: training and testing. In the training stage, the open hybrid model operates on basic and training projects. The training stage aims at evaluating the precision of each method existing in the methods bank and also achieving the best configuration for the optimal use of each method. In the testing stage, the open hybrid model operates on the basic and testing projects to evaluate the precision of configuration obtained from the training stage.

5.2 Training

Figure 3 shows the training flowchart. As mentioned earlier, the training stage operates on the basic and training projects. There are two banks of methods and rules in this stage. The methods bank consists of different methods for software development cost estimation. The rules bank includes different rules for integration of estimation methods in the methods bank. The coordinator function (COF) in the training stage employs the methods bank and rules bank. The COF needs a set of projects for estimation. It makes use of other projects resembling the one which should be estimated (P') to increase the estimation precision. To find projects similar to P', it is clustered along with the basic projects based on specific features (f_list). The f_list is a set of projects features recommended by FA. Based on the accuracy of estimation from the training set, the FA attempts to recommend a better f_list in each iteration, selecting more similar projects to P' for its estimation. When proposing f_list, FA is only allowed to recommend from continuous features. The P'-containing cluster is then sent to the COF, which uses the received projects to estimate P' based on the rule (proposed by FA) taken from the rules bank (this specific rule determines how to use existing methods in the methods bank). Table 1 shows the rules in the rules bank where ES_i represents the value of current project estimated by i^{th} method, and W_i (proposed by FA) indicates the coefficient of the i^{th} method existing in the methods bank. In some rules, er and m have been used to reduce the estimation error. This study used the k-means clustering method in which the number of clusters, denoted by k, should be adjusted. This parameter is recommended by FA during the training stage and then improved in the later iterations. The obtained k value is then used for testing.

#	function
Rule 1	$Cost = W_1 \times ES_1 + W_2 \times ES_2 + \cdots + W_n \times ES_n$
	$Cost = Cost + Cost \times m$
	$Cost = Cost + er$
Rule 2	$Cost = Median(ES_{1 \dots n})$
	$Cost = Cost + Cost \times m$
	$Cost = Cost + er$
Rule 3	$Cost = Mean(ES_{1 \dots n})$
	$Cost = Cost + Cost \times m$
	$Cost = Cost + er$
Rule 4	$Cost = ES_i$ (ES_i is best method in training iteration)

Table 1. Rules bank

In each stage of the open hybrid model, the FA proposes k and f_list for clustering the projects and a rule with the required parameters and coefficients (W, er, m) of that rule to the COF in each iteration. The COF uses the proposed k, f_list and rule to estimate each and every project of the training set. Finally, the estimation error of the training set is determined and returned as the feedback of the proposed k, *f_list* and rule to the FA. In the next iteration, the FA tries to propose k, *f_list* and rule with more appropriate parameters and coefficients to the COF. The output of the training stage includes the best k, *f_list* and rule along with the proposed parameters and coefficients required by that rule.

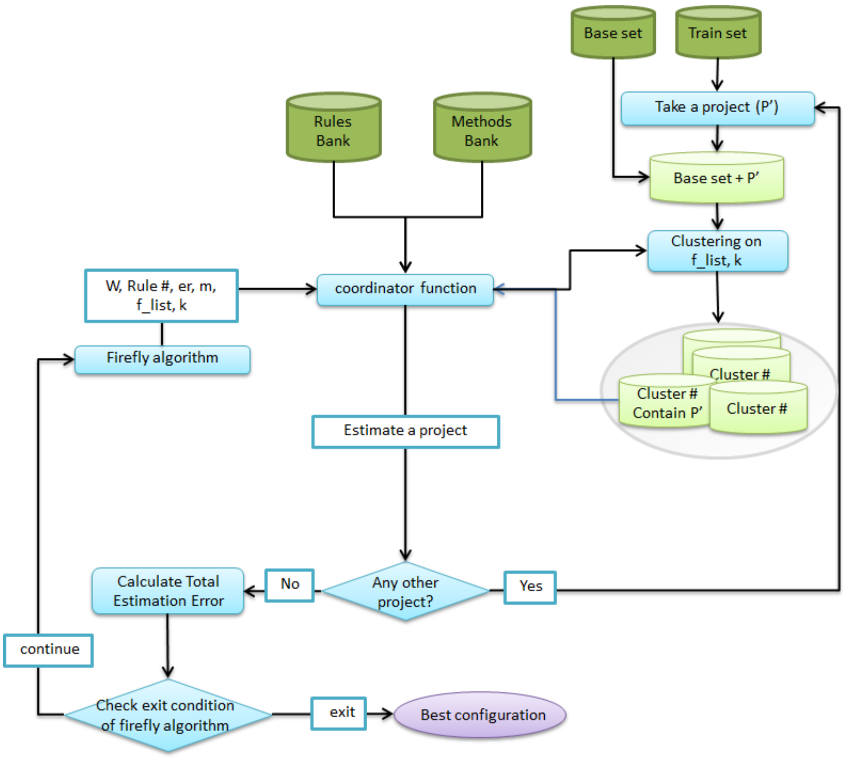


Figure 3. The training stage of the open hybrid model

5.3 Testing

As mentioned above, the testing stage operates on the basic and testing projects. According to Figure 4, the resulting configuration of the previous stage is given as the input to the model. The open hybrid model selects a project from the testing set (P') and then clusters P' with basic projects based on the *f_list* and k obtained from the previous stage. The cluster including P' is then given to the COF for estimating the development cost of P' based on the rule obtained from the previous stage. Another project is then selected and estimated through the previous stages. Iterations continue until no other projects remain in the testing set. Finally, the estimation error of each project is used to determine MdMRE, MMRE, and Pred. The results were used for comparing the models.

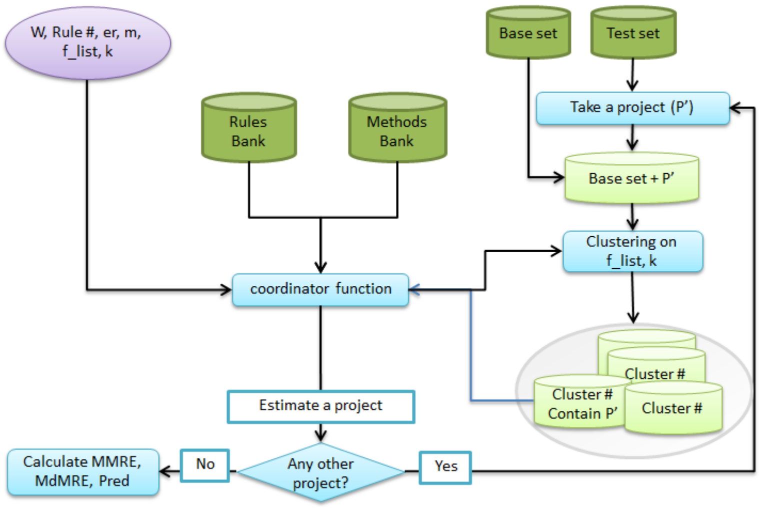


Figure 4. The testing stage of the open hybrid model

As the notable advantage of the open hybrid model over previous methods, it has been designed to add new methods and rules to the methods and rules banks. The model is also able to measure the efficiency of the new method intelligently and use it proportionately to its efficiency. Considering the methods used in the methods bank, the existing methods should be so diverse that model can operate dynamically in different modes and adapt to new conditions. The model proposed in this paper was obtained by analyzing the results of numerous studies indicating the success of any method in some datasets.

6 EVALUATION METHOD

The arrangement of samples in the training or testing stage may significantly affect the results of estimation models [21]. Therefore, a method is required to show the independency of results from the arrangement of samples to demonstrate the stability of results produced by the proposed model. Different evaluation methods such as 3-fold, 10-fold, and leave one out (LOO) have been proposed for this purpose. The LOO method was utilized in this study. In this method, when all projects except for one are used in the training stage, only one project is used in the testing stage. The nested LOO cross-validation was adopted in the compared methods. During training, different possible scenarios were considered for adjustable parameters by the LOO method, and the best quantities were used for testing. The process continued until tests completed for all projects.

7 DATASETS

The Desharnaise dataset contains data of 81 software projects collected from Canadian software houses. The software projects in this dataset have been described by 11 features. The dependent cost feature is based on 1000 person-hours. Ten independent features in this dataset include 'TeamExp', 'ManagerExp', 'YearEnd', 'Duration', 'Transactions', 'Entities', 'AdjFP', 'AdjFactor', 'RawFP', and 'Dev.Env'. Given that the data of 4 projects out of 81 projects in this dataset is missing and not available, the tests were conducted using the remaining 77 projects.

The Albrecht dataset contains data on software projects designed by third generation programming languages (3GLs). This dataset contains information on 24 projects. There is a dependent feature called 'work hours' in this dataset based on 1 000 hr. There are also 7 independent features ('input count', 'output count', 'query count', 'file count', 'line of code', 'RawFP', and 'function points') in this dataset.

The Kemerer dataset contains data on 15 software projects. The software projects in this dataset have been explained by 6 independent features and 1 dependent feature. The independent features include 'Language', 'hardware', 'RawFp', 'Duration', 'KSLOC' and 'AdjFP'. 'Cost' is considered as a dependent feature in this dataset and is measured by man-months. Table 2 shows the specifications of these datasets.

#	Dataset	Number of Projects	Number of Features	Mean of Cost
1	kemerer	15	7	219
2	desharnise	77	11	4 833
3	albrecht	24	8	21

Table 2. Datasets

8 TECHNIQUES

The results obtained from testing the proposed model were compared with those of different models to evaluate its precision. The following models were used for comparison:

Voting Ensemble: Elish [13] proposed an ensemble model for software development cost estimation. He combined the estimates obtained from three different methods by eight combination methods

Linearly Weighted Combinations Model (LWCM): Hsu et al. [16] developed an ensemble model consisting of COCOMO, linear regression, CBR, grey relational analysis and artificial neural networks estimators. Equally weighted combination, median weighted combination and weighted adjustment based on a criterion were used for combining the estimations.

Multilayer Perceptron (MLP): The neural network is a nonlinear modeling technique and MLP is a widely used neural network based on a network of neurons on an input layer with one or more hidden layers and an output layer.

Analog Based Estimation (ABE): ABE searches for the most similar sample to that which should be estimated. This method employs its internal functions to determine the resemblance of samples. The number of similar samples used for estimation is determined by the parameter K.

PSO + ABE: An estimation model based on ABE and particle swarm optimization (PSO) algorithm to increase the accuracy of software development cost estimation.

Ordinary Least Squares (OLS): OLS, as a regression-based method, is employed to determine the best regression line by minimizing the total squares.

Robust Regression (RoR): RoR is a regression-based method which can operate more accurately by weighting against unconventional data.

Multivariate Adaptive Regression Splines (MARS): MARS is a nonlinear nonparametric regression method with some interesting features such as ease of interpretability, modeling nonlinear complicated relationships and quick model development.

Classification and Regression Trees (CART): CART is an algorithm in which decision trees are employed for classification.

M5: This method can be regarded as a new type of CART. The model tree created by this method considers a linear regression for each leaf instead of determining a single value.

Least Squares SVM (LSSVM): The support vector machine (SVM) is a nonlinear machine learning method capable of mapping the input state space onto a space of higher dimensions leading to development of simpler linear regression functions.

9 TESTING THE OPEN HYBRID MODEL

Albrecht, Desharnise and Kemerer datasetes were employed to test the open hybrid model. The methods bank of the proposed model included CART, SWR, MLR, ABE and LSSVM. No normalization processes were performed on the data, and all of the methods were treated similarly and equally. The results of testing the proposed model on each dataset are described below.

Table 3 compares the results obtained from testing the open hybrid model on Albrecht with other methods. The proposed model was compared with other methods in terms of three different criteria (MMRE, MdMRE, and Pred). Accordingly, the proposed model was 51 %, and 65 % more precise than the most precise method in MMRE (Voting Ensemble), and the most precise method in MdMRE (LWCM), respectively. These results also indicated the higher precision of the proposed model than other methods. Figure 5 shows the results on a diagram for a better comparison. As clearly seen, the proposed model is able to increase the estimation precision simultaneously in terms of MdMRE, MMRE, and Pred.

Method	MMRE	MdMRE	Pred
Open Hybrid	0.22	0.08	0.62
LWCM	0.63	0.23	0.62
Voting Ensemble	0.45	0.37	0.41
ABE	0.85	0.38	0.29
CART	1.04	0.51	0.33
LSSVM	0.88	0.63	0.33
M5'	0.89	0.46	0.25
MARS	0.87	0.29	0.45
MLP	0.95	0.41	0.2
OLS	0.79	0.5	0.37
PSO + ABE	1.04	0.48	0.12
ROR	0.72	0.66	0.29

Table 3. The results of testing Albrecht in comparison with other methods

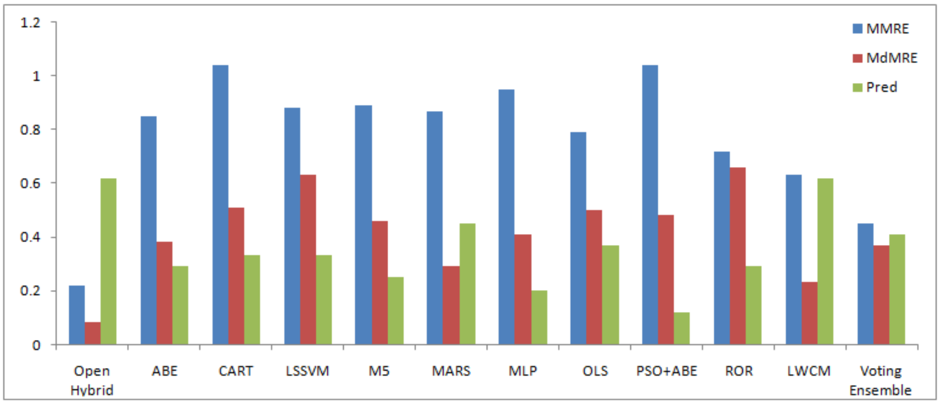


Figure 5. Comparing different methods in MMRE, MdMRE, and Pred on Albrecht

Table 4 compares the results of testing the open hybrid model on Kemerer with other methods. The results were evaluated in terms of MMRE, MdMRE, and Pred. Accordingly, the proposed model was nearly 54 %, 67 %, and 43 % more precise than the most precise method in MMRE (LWCM), the most precise method in MdMRE (LWCM), and the most precise method in Pred (Voting Ensemble), respectively. For a better comparison, the test results were shown on a diagram in Figure 6 which clearly depicts the precision of the proposed model on Kemerer in comparison with other methods.

Method	MMRE	MdMRE	Pred
Open Hybrid	0.23	0.09	0.66
LWCM	0.5	0.28	0.33
Voting Ensemble	0.54	0.36	0.46
ABE	0.82	0.46	0.2
CART	0.96	0.61	0.06
LSSVM	0.64	0.55	0.26
M5’	1.15	0.73	0.2
MARS	1.4	0.8	0.26
MLP	0.57	0.49	0.33
OLS	0.64	0.5	0.26
PSO + ABE	0.61	0.47	0.33
ROR	0.62	0.36	0.13

Table 4. The results of testing Kemerer in comparison with other methods

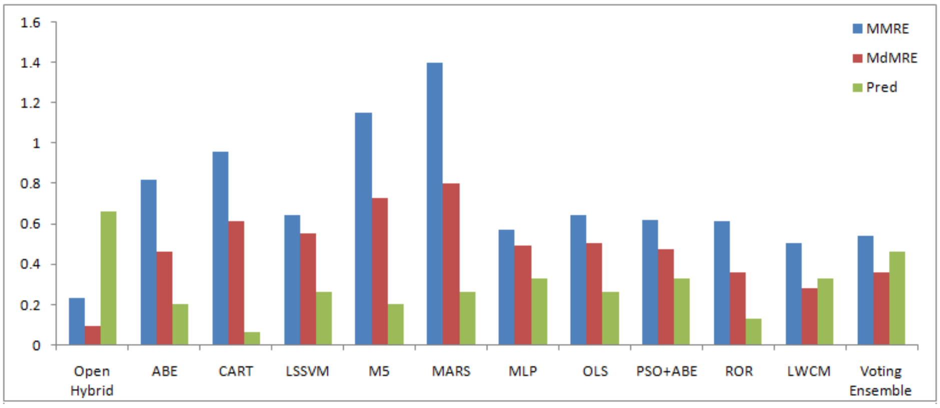


Figure 6. Comparing different methods in MMRE, MdMRE, and Pred on Kemerer

The proposed model was also tested on Desharnise and the results are presented in Table 5. 11 other methods were also tested on this dataset. The results of testing the proposed model were compared with those of other methods in terms of MMRE, MdMRE, and Pred. According to the results, the proposed method was nearly 35 %, 40 %, and 43 % more precise than the most precise method in MMRE (Voting Ensemble), the most precise method in MdMRE (Voting Ensemble), and the most precise method in Pred (Voting Ensemble), respectively. For a better comparison, the results were also shown on a diagram in Figure 7, indicating the proper capability of the proposed model.

Method	MMRE	MdMRE	Pred
Open Hybrid	0.26	0.17	0.59
LWCM	0.49	0.29	0.25
Voting Ensemble	0.4	0.28	0.41
ABE	0.74	0.4	0.28
CART	0.68	0.35	0.27
LSSVM	0.58	0.41	0.24
M5’	0.72	0.39	0.29
MARS	1.19	0.57	0.23
MLP	0.91	0.54	0.24
OLS	0.71	0.53	0.27
PSO + ABE	0.87	0.4	0.4
ROR	0.6	0.49	0.36

Table 5. The results of testing Desharnise in comparison with other methods

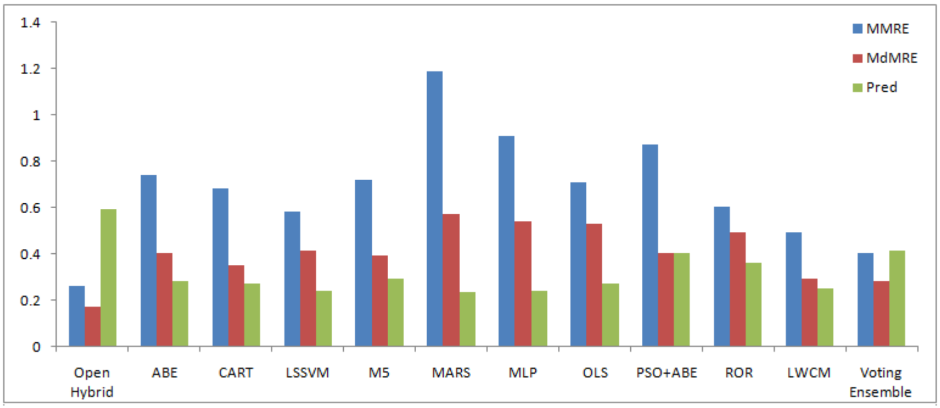


Figure 7. Comparing different methods in MMRE, MdMRE, and Pred on Desharnise

10 TEST RESULT ANALYSIS

Wilcoxon statistical test with two input statistical samples was used to determine the effectiveness of the proposed model. The output, P-value, shows the difference between the two input samples. A small P-value indicates the difference of the two samples, whereas a large value shows the similarity of both samples. A P-value less than 0.05 means a large difference between the two samples. Table 6 lists the Wilcoxon test results on MRE. It shows different P-values obtained from conducting the Wilcoxon test on the MRE of each method in comparison with the proposed model on different datasets. The results indicate the higher effectiveness of the proposed model in increasing estimation precision.

The box plot was employed for detailed analysis of the results of the proposed model. The box plot shows the MRE range obtained from testing the proposed

Method	Albrecht	Kemerer	Desharnise
ABE	0.0023	0.0021	0.0001
CART	0.0006	0.0014	7.95E−06
LWCM	0.019	8.00E−02	2.7E−02
Voting Ensemble	0.011	1.80E−02	4.20E−02
LSSVM	5.50E−04	4.70E−03	1.06E−05
M5'	0.0003	0.0012	1.17E−05
MARS	0.044	0.0018	5.36E−10
MLP	0.0005	3.40E−02	2.41E−08
OLS	0.013	0.027	1.58E−08
PSO + ABE	9.32E−05	5.60E−02	0.0002
ROR	5.00E−04	2.40E−03	1.19E−05

Table 6. P-values of Wilcoxon test

model on a dataset. Using this box plot, the MRE range and precision of the proposed model can easily be compared to those of other models. Figures 8, 9 and 10 show the box plots of MRE obtained from testing different methods on Albrecht, Kemerer, and Desharnise, respectively. As can be seen, the proposed model shows the lowest median and quartile range. All the results confirm the capability of the proposed model.

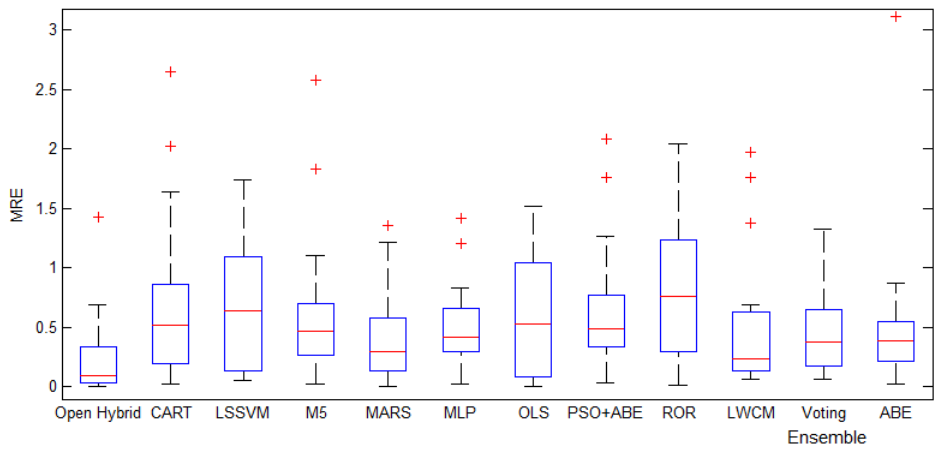


Figure 8. The box plot of MRE obtained by testing the proposed model on Albrecht

11 CONCLUSION

Software development cost estimation is an important issue resulting in an effective planning for software project management. Several methods and models have been proposed for this purpose. Despite acceptable precision of the proposed methods in

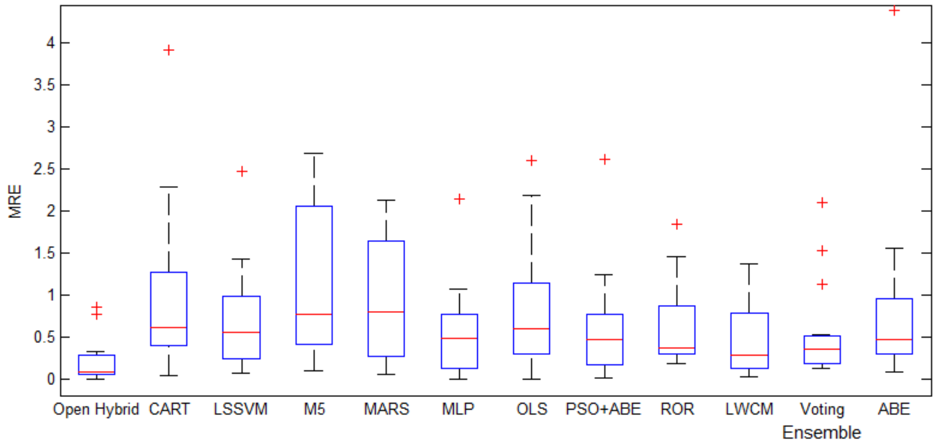


Figure 9. The box plot of MRE obtained by testing the proposed model on Kemerer

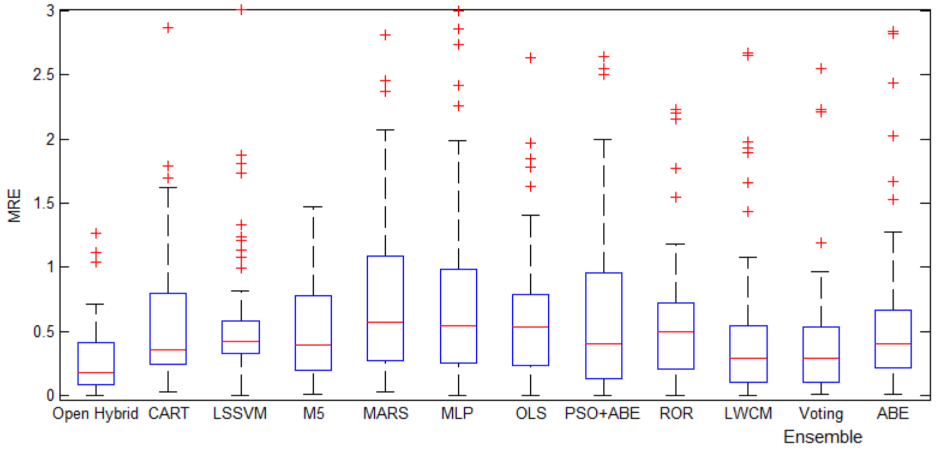


Figure 10. The box plot of MRE obtained by testing the proposed model on Desharnise

some cases, they fail operate accurately in other cases. The aim of this study was to combine these methods to take their advantages simultaneously to obtain a more accurate estimate. This idea led the authors to implement a model called the "open hybrid model".

The methods proposed in previous studies accurately estimate specific types of software projects when they depend on a specific estimation method. However, the challenge addressed here was to identify in what types of projects an estimation method can operate successfully, to what extent it can be successful, and how it is possible to take advantage of such a high level of precision. For this purpose, a new model called open hybrid model was developed in Section 5 based on the

FA. Different estimation methods can be added to the methods bank of this model. The proposed model is able to find the best rule for employing the methods in the methods bank using a variety of parameters and tools in the model.

The precision of the proposed model was evaluated by using three datasets of software projects. The results were compared with those obtained by testing 11 methods. The results were evaluated in terms of MMRE, MdMRE, and Pred. According to the results, the proposed model showed a high precision and ability to use capabilities of the methods existing in its methods bank. The results of tests were interpreted in detail in Section 9. Wilcoxon statistical test was conducted on the results to determine the effectiveness of the proposed model. The Wilcoxon results were analyzed in Section 10.

Regarding the other features of the proposed model, it was designed to include the new methods in its methods bank to use their advantages. Therefore, a new method with precise estimations in one or multiple datasets can be added to the methods bank. According to the results of multiple tests, the open hybrid model is able to intelligently identify a space in which a method can operate well. It can also effectively use every method under appropriate conditions. In future studies, more diverse and accurate methods and rules can be added to the methods and rules banks of this model.

REFERENCES

[1] Abnane, I.—Idri, A.—Abran, A.: Empirical Evaluation of Fuzzy Analogy for Software Development Effort Estimation. Proceedings of the Symposium on Applied Computing (SAC '17), ACM, 2017, pp. 1302–1304, doi: 10.1145/3019612.3019905.

[2] Andreou, A. S.—Papatheocharous, E.: Software Cost Estimation Using Fuzzy Decision Trees. 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, L'Aquila, Italy, 2008, pp. 371–374, doi: 10.1109/ase.2008.51.

[3] Arora, S.—Mishra, N.: Software Cost Estimation Using Artificial Neural Network. In: Pant, M., Ray, K., Sharma, T., Rawat, S., Bandyopadhyay, A. (Eds.): Soft Computing: Theories and Applications, Springer, Singapore, Advances in Intelligent Systems and Computing, Vol. 584, 2018, pp. 51–58, doi: 10.1007/978-981-10-5699-4_6.

[4] Azzeh, M.—Nassif, A. B.: A Hybrid Model for Estimating Software Project Effort from Use Case Points. Applied Soft Computing, Vol. 49, 2016, pp. 981–989, doi: 10.1016/j.asoc.2016.05.008.

[5] Bardsiri, A. K.—Hashemi, S. M.: Software Effort Estimation: A Survey of Well-Known Approaches. International Journal of Computer Science Engineering (IJCSE), Vol. 3, 2014, No. 1, pp. 46–50.

[6] Bardsiri, A. K.—Hashemi, S. M.: Empirical Evaluation of Different Machine Learning Methods for Software Services Development Effort Estimation Through Correlation Analysis. European Journal of Applied Sciences, Vol. 8, 2016, No. 4, pp. 257–269.

[7] Boehm, B. W.: Software Engineering Economics. New York, 1981.

[8] Boehm, B. W.—Abts, C.—Brown, A. W.—Chulani, S.—Clark, B. K.—Horowitz, E.—Madachy, R.—Reifer, D. J.—Steece, B.: Software Cost Estimation with Cocomo II. Prentice-Hall, 2000.

[9] Boehm, B. W.—Valerdi, R.: Achievements and Challenges in Cocomo-Based Software Resource Estimation. IEEE Software, Vol. 25, 2008, No. 5, pp. 74–83, doi: 10.1109/ms.2008.133.

[10] Cuadrado-Gallego, J. J.—Rodríguez-Soria, P.—Martín-Herrera, B.: Analogies and Differences Between Machine Learning and Expert Based Software Project Effort Estimation. 2010 11th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, IEEE, 2010, pp. 269–276, doi: 10.1109/snpd.2010.47.

[11] Sholiq—Dewi, R. S.—Subriadi, A. P.: A Comparative Study of Software Development Size Estimation Method: UCPabc vs Function Points. Procedia Computer Science, Vol. 124, 2017, pp. 470–477, doi: 10.1016/j.procs.2017.12.179.

[12] Dewi, R. S.—Subriadi, A. P.—Sholiq: A Modification Complexity Factor in Function Points Method for Software Cost Estimation Towards Public Service Application. Procedia Computer Science, Vol. 124, 2017, pp. 415–422, doi: 10.1016/j.procs.2017.12.172.

[13] Elish, M. O.: Assessment of Voting Ensemble for Estimating Software Development Effort. 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2013, pp. 316–321, doi: 10.1109/cidm.2013.6597253.

[14] Cuadrado-Gallego, J. J.—Sicilia, M.-Á.: An Algorithm for the Generation of Segmented Parametric Software Estimation Models and Its Empirical Evaluation. Computing and Informatics, Vol. 26, 2007, No. 1, pp. 1–15.

[15] González-Ladrón-de-Guevara, F.—Fernández-Diego, M.—Lokan, C.: The Usage of ISBSG Data Fields in Software Effort Estimation: A Systematic Mapping Study. Journal of Systems and Software, Vol. 113, 2016, pp. 188–215, doi: 10.1016/j.jss.2015.11.040.

[16] Hsu, C.-J.—Rodas, N. U.—Huang, C.-Y.—Peng, K.-L.: A Study of Improving the Accuracy of Software Effort Estimation Using Linearly Weighted Combinations. 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops, 2010, pp. 98–103, doi: 10.1109/compsacw.2010.27.

[17] Huang, J.—Li, Y.-F.—Xie, M.: An Empirical Analysis of Data Preprocessing for Machine Learning-Based Software Cost Estimation. Information and Software Technology, Vol. 67, 2015, pp. 108–127, doi: 10.1016/j.infsof.2015.07.004.

[18] Idri, A.—Azzahra Amazal, F.—Abran, A.: Analogy-Based Software Development Effort Estimation: A Systematic Mapping and Review. Information and Software Technology, Vol. 58, 2015, pp. 206–230, doi: 10.1016/j.infsof.2014.07.013.

[19] Khatibi, V.—Jawawi, D. N. A.: Software Cost Estimation Methods: A Review. Journal of Emerging Trends in Computing and Information Sciences, Vol. 2, 2011, No. 1, pp. 21–29.

[20] Khuat, T. T.—Le, M. H.: Applying Teaching-Learning to Artificial Bee Colony for Parameter Optimization of Software Effort Estimation Model. Journal of Engineering Science and Technology, Vol. 12, 2017, No. 5, pp. 1178–1190.

[21] Kocaguneli, E.—Menzies, T.: Software Effort Models Should Be Assessed via Leave-One-Out Validation. Journal of Systems and Software, Vol. 86, 2013, No. 7, pp. 1879–1890, doi: 10.1016/j.jss.2013.02.053.

[22] Kocaguneli, E.—Menzies, T.—Keung, J. W.: On the Value of Ensemble Effort Estimation. IEEE Transactions on Software Engineering, Vol. 38, 2012, No. 6, pp. 1403–1416, doi: 10.1109/TSE.2011.111.

[23] Kultur, Y.—Turhan, B.—Bener, A. B.: ENNA: Software Effort Estimation Using Ensemble of Neural Networks with Associative Memory. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16), 2008, pp. 330–338, doi: 10.1145/1453101.1453148.

[24] Kumari, S.—Pushkar, S.: Performance Analysis of the Software Cost Estimation Methods: A Review. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, 2013, No. 7, pp. 229–238.

[25] Malgonde, O.—Chari, K.: An Ensemble-Based Model for Predicting Agile Software Development Effort. Empirical Software Engineering, Vol. 24, 2019, No. 2, pp. 1017–1055, doi: 10.1007/s10664-018-9647-0.

[26] Mensah, S.—Keung, J.—Bosu, M. F.—Bennin, K. E.: Duplex Output Software Effort Estimation Model with Self-Guided Interpretation. Information and Software Technology, Vol. 94, 2018, pp. 1–13, doi: 10.1016/j.infsof.2017.09.010.

[27] Moosavi, S. H. S.—Bardsiri, V. K.: Satin Bowerbird Optimizer: A New Optimization Algorithm to Optimize ANFIS for Software Development Effort Estimation. Engineering Applications of Artificial Intelligence, Vol. 60, 2017, pp. 1–15, doi: 10.1016/j.engappai.2017.01.006.

[28] Ochieng, N.—Mwangi, W.—Ateya, I.: Optimizing Computer Worm Detection Using Ensembles. Security and Communication Networks, Vol. 2019, 2019, Art. No. 4656480, doi: 10.1155/2019/4656480.

[29] Pahariya, J. S.—Ravi, V.—Carr, M.: Software Cost Estimation Using Computational Intelligence Techniques. 2009 World Congress on Nature and Biologically Inspired Computing (NaBIC), IEEE, 2009, pp. 849–854, doi: 10.1109/nabic.2009.5393534.

[30] Pandey, P.: Analysis of the Techniques for Software Cost Estimation. 2013 Third International Conference on Advanced Computing and Communication Technologies (ACCT), IEEE, 2013, pp. 16–19, doi: 10.1109/acct.2013.13.

[31] Pospieszny, P.—Czarnacka-Chrobot, B.—Kobylinski, A.: An Effective Approach for Software Project Effort and Duration Estimation with Machine Learning Algorithms. Journal of Systems and Software, Vol. 137, 2018, pp. 184–196, doi: 10.1016/j.jss.2017.11.066.

[32] Puspaningrum, A.—Sarno, R.: A Hybrid Cuckoo Optimization and Harmony Search Algorithm for Software Cost Estimation. Procedia Computer Science, Vol. 124, 2017, pp. 461–469, doi: 10.1016/j.procs.2017.12.178.

[33] Prasad Reddy, P. V. G. D.—Hari, C. V. M. K.—Srinavasa Rao, T.: Multi Objective Particle Swarm Optimization for Software Cost Estimation. International Journal of Computer Applications, Vol. 32, 2011, No. 3, pp. 13–17.

[34] Rastogi, H.—Dhankhar, S.—Kakkar, M.: A Survey on Software Effort Estimation Techniques. 2014 5th International Conference – Confluence the Next Generation Information Technology Summit (Confluence), IEEE, 2014, pp. 826–830, doi: 10.1109/confluence.2014.6949367.

[35] Salehi, M.—Razmara, J.—Lotfi, S.: A Novel Data Mining on Breast Cancer Survivability Using MLP Ensemble Learners. The Computer Journal, Vol. 63, 2020, pp. 435–447, doi: 10.1093/comjnl/bxz051.

[36] Shekhar, S.—Kumar, U.: Review of Various Software Cost Estimation Techniques. International Journal of Computer Applications, Vol. 141, 2016, No. 11, pp. 31–34, doi: 10.5120/ijca2016909867.

[37] Sigweni, B.—Shepperd, M.: Feature Weighting Techniques for CBR in Software Effort Estimation Studies: A Review and Empirical Evaluation. Proceedings of the 10th International Conference on Predictive Models in Software Engineering (PROMISE '14), ACM, 2014, pp. 32–41, doi: 10.1145/2639490.2639508.

[38] Silva, J.—Borré, J. R.—Piñeres Castillo, A. P.—Castro, L.—Varela, N.: Integration of Data Mining Classification Techniques and Ensemble Learning for Predicting the Export Potential of a Company. Procedia Computer Science, Vol. 151, 2019, pp. 1194–1200, doi: 10.1016/j.procs.2019.04.171.

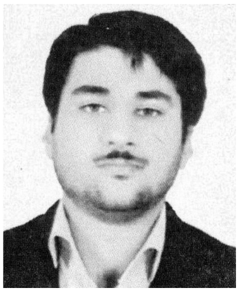
[39] Song, L.—Minku, L. L.—Yao, X.: The Impact of Parameter Tuning on Software Effort Estimation Using Learning Machines. Proceedings of the 9th International Conference on Predictive Models in Software Engineering (PROMISE '13), ACM, 2013, Art. No. 9, doi: 10.1145/2499393.2499394.

[40] Wani, Z. H.—Quadri, S. M. K.: Software Cost Estimation Based on the Hybrid Model of Input Selection Procedure and Artificial Neural Network. Artificial Intelligent Systems and Machine Learning, Vol. 10, 2018, No. 1, pp. 18–24.

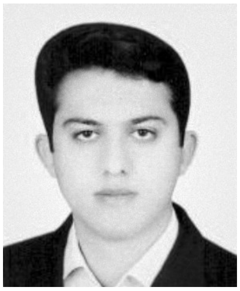
[41] Wen, J.—Li, S.—Lin, Z.—Hu, Y.—Huang, C.: Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models. Information and Software Technology, Vol. 54, 2012, No. 1, pp. 41–59, doi: 10.1016/j.infsof.2011.09.002.

[42] Wu, D.—Li, J.—Liang, Y.: Linear Combination of Multiple Case-Based Reasoning with Optimized Weight for Software Effort Estimation. The Journal of Supercomputing, Vol. 64, 2013, No. 3, pp. 898–918, doi: 10.1007/s11227-010-0525-9.

[43] Yang, X.-S.: Firefly Algorithms for Multimodal Optimization. In: Watanabe, O., Zeugmann, T. (Eds.): Stochastic Algorithms: Foundations and Applications (SAGA 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5792, 2009, pp. 169–178, doi: 10.1007/978-3-642-04944-6_14.



Amin MORADBEIKY received his B.Sc. from University of Sistan and Baluchestan, Iran in 2009 and M.Sc. from Islamic Azad University, Kerman Branch, Iran in 2014, both in engineering of information technology. He is currently Ph.D. student of software engineering in Islamic Azad University, Kerman Branch, Iran. His research interests are soft computing techniques, software test and software measurement.



Vahid Khatibi BARDSIRI is a lecturer at the Department of Computer Science, Islamic Azad University, Bardsir Branch, Iran. He received his B.Sc. and M.Sc. degrees in software engineering from Ferdowsi University of Mashhad, Iran in 2002 and from Science and Research Branch of Islamic Azad University, Iran in 2004, respectively. He received his Ph.D. in the area of software development effort estimation at Universiti Teknologi Malaysia (UTM) in 2013. He is a senior member of International Association of Computer Science and Information Technology (IACSIT). His research interests are agile software development methods, soft computing techniques and software measurement.

Mehdi JAFARI received his B.Sc. degree in electronic engineering and M.Sc. degree in communication systems from Shiraz University of Technology, Iran, in 1992 and 1996, respectively. During 1992–1998 he stayed in the Signal Processing Research Laboratory, Ministry of Telecommunications of Iran. He received Ph.D. degree in communication systems in Science and Research Branch of Islamic Azad University, Iran in 2008. He is currently Head of Electrical Engineering Department of Islamic Azad University, Kerman Branch, Iran. His main research interests are artificial intelligence, image and video processing, pattern recognition and prediction.

Computing and Informatics, Vol. 39, 2020, 1172–1202, doi: 10.31577/cai_2020_6_1172

EXPLANATION OF SIAMESE NEURAL NETWORKS FOR WEAKLY SUPERVISED LEARNING

Lev Utkin, Maxim Kovalev, Ernest Kasimov

Peter the Great Saint Petersburg Polytechnic University (SPbPU)
Saint Petersburg, Russia
e-mail: {lev.utkin, maxkovalev03, kasimov.ernest}@gmail.com

Abstract. A new method for explaining the Siamese neural network (SNN) as a black-box model for weakly supervised learning is proposed under condition that the output of every subnetwork of the SNN is a vector which is accessible. The main problem of the explanation is that the perturbation technique cannot be used directly for input instances because only their semantic similarity or dissimilarity is known. Moreover, there is no an "inverse" map between the SNN output vector and the corresponding input instance. Therefore, a special autoencoder is proposed, which takes into account the proximity of its hidden representation and the SNN outputs. Its pre-trained decoder part as well as the encoder are used to reconstruct original instances from the SNN perturbed output vectors. The important features of the explained instances are determined by averaging the corresponding changes of the reconstructed instances. Numerical experiments with synthetic data and with the well-known dataset MNIST illustrate the proposed method.

Keywords: Interpretable model, explainable AI, Siamese neural network, embedding, autoencoder, perturbation technique

Mathematics Subject Classification 2010: 68T10

1 INTRODUCTION

Machine learning models, especially, deep models play an important role in making prediction and decision for many applications. For example, computer-aided diagnosis systems using machine learning models for the diagnosis of various diseases have become a key element in medical imaging and personalized medicine over the past few years. However, many modern efficient machine learning techniques are not easily explainable, they are black boxes, i.e., they do not explain their predictions in a way that humans could understand. This may be an obstacle for incorporating the machine learning models into applied areas like medicine. Often, humans are unable to effectively use predictions provided by the machine learning models without their interpretation and explanation. As a result, a lot of models have been developed to explain predictions of the deep classification and regression algorithms, for example, deep neural network predictions [1].

A clear taxonomy for understanding the diverse forms of explanations is provided by Arya et al. [2]. Since a lot of machine learning models are black boxes, then we mainly consider the so-called post-hoc explanations which involve auxiliary models to explain the black-box models after it has been trained. These auxiliary or explanation models can also be divided into three types: example-based, local and global models. According to the example-based approach, an instance from the training set is selected, which could explain the behavior of a black-box model. One of the methods implementing the example-based explanations is nearest neighbors [3]. Local or prediction-level models focus on explaining how they make individual predictions, namely, what features lead to the individual prediction (see [4] for a definition). Explanations are derived by fitting an interpretable model locally around the considered instance. Global or dataset-level models explain importance of features across a dataset or a population [5]. They explain the global relationships between features and the model predictions.

A key component of general explanations for the local models as well as for the global models is the contribution of individual input features. A prediction is computationally explained by assigning to each feature a number which denotes its impact on the prediction.

One of the important machine learning tasks is to compare pairs of objects, for example, pairs of images, pairs of data vectors, etc. The task can be solved in the framework of the distance metric learning approach [6, 7, 8], which is based on computing a corresponding pairwise metric function that measures a distance between data vectors or a similarity between the vectors. The basic idea behind the metric learning solution is that the distance between similar objects should be smaller than the distance between different objects.

A powerful implementation of the metric learning dealing with non-linear data structures is the so-called Siamese neural network (SNN) [9, 10]. The SNN consists of two identical neural subnets sharing the same set of weights. The basic idea behind the SNN is to train the subnets to compare a pair of feature vectors in terms of their semantic similarity or dissimilarity. The SNN realizes a non-linear embedding of data with the objective to bring together similar instances and to move apart dissimilar instances. In a simplest case, when the training set is labelled, i.e., every instance belongs to a class, then two instances are similar if they belong to the same class. Two instances are dissimilar if they belong to different classes. At the same time, the SNN can be applied to the weakly supervised case when there are no labels of training instances, but there is only a side information of relationship of instances, i.e., of semantic similarity or dissimilarity of instances.

An approach to explain the Siamese neural network (SNN) as a black-box model is proposed. The approach is agnostic to the black-box model. This means that we do not know or do not use any details of the black-box model. Only its input and the corresponding output are used for training the explanation model. We also assume that the explained SNN is regarded as the black-box model in the sense that we know input feature vectors and embedded vectors, but we do not know peculiarities and tuning parameters of the SNN.

At the same time, to the best of our knowledge, there are no appropriate algorithms for explaining the SNN which is used as a weakly supervised learning model. Therefore, we aim to solve this explanation problem and to propose the corresponding approach which allows us to get subsets of features explaining similarity and dissimilarity of certain instances.

First of all, let us consider main difficulties of explaining the SNN. They are the following:

1. The input vectors (instances) are semantically similar or dissimilar. Therefore, direct distances between the input feature vectors do not have a sense. Semantically dissimilar vectors may be closer to each other than semantically similar vectors.
2. The perturbation technique cannot be used directly for input instances of the SNN due to the possible large dimensionality of input data.
3. Prototypes cannot be defined in case of weakly supervised data. Moreover, prototypes defined as mean values of the input vectors also do not have a sense because the input vectors may be too different. They can be defined only by using the output vectors (embeddings).
4. There is no an "inverse" map between the embedded vectors and the corresponding input feature vectors, i.e., we do not know subsets of features in the input vector corresponding to some features of the embedded vector.

Taking into account the fact that the distance measurement has a sense only for the SNN output vectors, direct ideas for explaining the SNN are the following:

1. to select a predefined number of features from a pair of embeddings of semantically similar instances, which have the smallest distance between each other;
2. to select a predefined number of features from a pair of embeddings of semantically dissimilar instances, which have the largest Euclidean distance between each other;
3. to perturb the selected features in a specific way;
4. to reconstruct the obtained perturbation in order to find which features of reconstructed instances are changed in accordance with the perturbation of embeddings;

5. maximally changed features of reconstructed instances explain similarity or dissimilarity of the considered original instances.

These ideas could be viewed as an approach for developing a SNN explanation method, but a bottleneck of the approach is the reconstruction of instances from the corresponding embeddings. Our experiments have shown that it is very difficult to train a reconstruction neural network, having the embedded vectors as its input and the original instances as its output, due to a large difference between the instance dimensionality and the embedded vector size. Therefore, we propose to apply an autoencoder (AE) of a special type in order to overcome the above difficulty and to help for reconstructing the instances from their embeddings. The proposed AE is trained in a way different from the standard AE. In contrast to the standard AE, it takes into account the proximity of its hidden representation and the SNN outputs by means of extending the corresponding loss function. The decoder part of the pretrained AE can be regarded as the reconstruction neural network after its additional training by using the SNN outputs. Our numerical experiments have demonstrated that this scheme allows us to reconstruct images (instances) with a high accuracy. By perturbing the SNN outputs in a special way and using the trained decoder of the AE, we can consider how these perturbations impact on the reconstructed instances.

The paper is organized as follows. Related work concerning with available explanation models and the SNN applications is considered in Section 2. A detailed description of the SNN architecture is given in Section 3. Main ideas of the proposed SNN explanation method are discussed in Section 4. Details of the AE implementation for improving the reconstruction of original images from the corresponding embeddings are given in Section 5. A scheme of the used perturbation technique and the instance reconstruction is studied in Section 6. Numerical experiments illustrating the proposed method on the basis of the synthetic dataset and the well-known MNIST dataset are given in Section 7. Concluding remarks are provided in Section 8.

2 RELATED WORK

2.1 Explanation Models

There are a lot of approaches to locally explain black-box models. One of the very popular methods is the Local Interpretable Model-agnostic Explanations (LIME) [11]. The main intuition of LIME is that the explanation may be derived locally from a set of synthetic instances generated randomly in the neighborhood of the instance to be explained such that every synthetic instance has a weight according to its proximity to the explained instance. Several modifications of LIME have been proposed due to success and simplicity of the method, for example, ALIME [12], NormLIME [13], DLIME [14], Anchor LIME [15], LIME-SUP [16], LIME-Aleph [17], SurvLIME [18]. Garreau and Luxburg [19] proposed a thorough theoretical analysis of LIME.

Another very popular method is the SHAP [20] and its modifications which take a game-theoretic approach by optimizing a regression loss function based on Shapley values [21, 22, 23, 24]. Alternative methods are influence functions [25], a multiple hypothesis testing framework [26], and many other methods.

A large part of methods can be united as counterfactual explanations [27]. They try to answer the question: "Why is the outcome Y obtained instead of Z?". A simplest approach to answer the question is to find the nearest training instance belonging to another class. As shown by Moore et al. [28], this approach strongly depends on the size and quality of the considered training set, and it cannot find a counterfactual that is not explicitly in the set. Therefore, a lot of new methods of the counterfactual explanation have been developed [29, 30, 31, 32, 33, 34, 35].

Dhurandhar et al. [36, 37] extended the counterfactual explanation by introducing the so-called contrastive explanation methods which produce explanations of the form [36]: "An input feature vector is classified in class y because features $f_{i_1}, \dots, f_{i_k}$ are present and because features $f_{j_1}, \dots, f_{j_l}$ are absent".

A large part of explanation methods uses a prototype technique which selects representative instances from training data, for instance, from instances belonging to the same class. These instances are called prototypes [38, 39]. The explanation methods using this technique determine how an explained instance is similar to a prototype.

It is important to point out also that most aforementioned explanation methods starting from LIME [11] are based on perturbation technique [40, 41, 42, 43, 44, 45]. These methods assume that contribution of a feature can be determined by measuring how prediction score changes when the feature is altered [46]. Strumbel and Kononenko [20] propose to perturb the input feature vectors and to observe how changes of the input features correspond to changes of the outcome. They rely on an assumption that a feature is important and strongly impacts the outcome if its change sufficiently changes the outcome. Perturbation techniques are model-agnostic, i.e., perturbations can be applied to a black-box model without any need to access the internal structure of the model. However, the perturbation technique may meet computational difficulties when perturbed input instances have a lot of features.

Several comprehensive surveys devoted to various explainable methods can be found in literature [47, 48, 49, 1, 50]. A very interesting critical review of main assumptions and statements accepted in explaining the black-box machine learning models is provided by Rudin [51]. The review [51] considers in detail how to avoid mistakes and incorrect assumptions in explanation models. The corresponding software packages [52] are developed in order to simplify the explanation process for various machine learning models.

2.2 Metric Learning and Siamese Neural Networks

A detailed description of the metric learning approaches is represented by Le Capitaine [53] and by Kulis [7]. One of the most important and popular approaches is to use the Mahalanobis distance as a distance metric which assumes some linear structure of data. However, this assumption significantly restricts the applicability of the Mahalanobis distance for comparing pairs of objects. Therefore, in order to overcome this restriction, the kernelization of linear methods is one of the possible ways for solving the metric learning problem. Bellet et al. [6] review several approaches and algorithms to deal with nonlinear forms of metrics. In particular, these are the support vector metric learning algorithm provided by Xu et al. [54], the gradient-boosted large margin nearest neighbors method proposed by Kedem et al. [55], the Hamming distance metric learning algorithm provided by Norouzi et al. [56]. Various methods and applications of the metric learning can be found in [57, 58, 59, 60, 61, 62, 54, 63].

SNNs realize a non-linear embedding of data [64] and were introduced in 90s by Bromley and LeCun to solve signature verification as an image matching problem [9]. SNNs have been widely spread in solving many application problems. In particular, they are applied to problems of image recognition and verification [10, 65, 66, 67, 68, 69], of speaker verification [70], of visual tracking [71, 72], of novelty and anomaly detection [73, 74], and to many different theoretical and practical problems [75, 76, 77, 78]. An important application of the SNN is one-shot learning [79] or few-shot learning [80, 81, 82, 83], when it is supposed that there are only a few training instances in some classes for training. A more detailed and general definition of the one-shot and few-shot learning is given in [84].

It should be noted that the above applications present only a small part of all applications of the SNNs. Many modifications of SNNs have been also developed, including fully-convolutional SNNs [85], SNNs combined with a gradient boosting classifier [86], SNNs with the triangular similarity metric [8].

3 SIAMESE NEURAL NETWORKS

Let $S = \{(\mathbf{x}_i, \mathbf{x}_j, z_{ij}), (i,j) \in K\}$ be a dataset consisting of N pairs of feature vectors $\mathbf{x}_i \in \mathbb{R}^m$ and $\mathbf{x}_j \in \mathbb{R}^m$ such that a binary label $z_{ij} \in \{0,1\}$ is assigned to every pair $(\mathbf{x}_i, \mathbf{x}_j)$. If both feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are semantically similar, then z_{ij} takes value 0. If the vectors are semantically dissimilar, i.e., they correspond to different classes, then z_{ij} takes value 1. This implies that the training set S can be divided into two subsets: a similar or positive set with $z_{ij} = 0$ and a dissimilar or negative set with $z_{ij} = 1$. It should be noted that knowledge of classes is not necessary if we have only weak information about similarity of pairs of instances.

The main idea of using the SNN can be formulated as follows. If there are two feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ being dissimilar, then the Euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j)$ between these feature vectors should be as large as possible. However, this may be not a case in practice. For instance, if to consider the medicine application, then tuberculosis and adenocarcinoma in the lung cancer diagnosis may have similar computed tomography patterns, but these are quite different diseases. Adenocarcinoma is cancer, but tuberculosis is not. At the same time, different forms of lung cancer may look quite differently, for instance, lepidic and squamous cell carcinoma. In this case, the Euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j)$ may be rather large, but it should be as small as possible. In other words, the Euclidean distance $d(\mathbf{x}_i, \mathbf{x}_j)$ often does not correspond to semantic similarity of objects. Therefore, we have to consider not the distance between feature vectors themselves, but the distance between new feature representations or embeddings denoted as $\mathbf{h}_i = (h_1^{(i)}, \dots, h_D^{(i)}) \in \mathbb{R}^D$ and $\mathbf{h}_j = (h_1^{(j)}, \dots, h_D^{(j)}) \in \mathbb{R}^D$, which fulfil the conditions of distances and similarity, i.e., the Euclidean distance $d(\mathbf{h}_i, \mathbf{h}_j)$ between vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ should be as small (large) as possible for a pair of objects with $z_{ij} = 0$ ($z_{ij} = 1$). At that, every vector $\mathbf{h}_i$ is a map f of $\mathbf{x}_i$ to a low-dimensional space, i.e., $\mathbf{h}_i = f(\mathbf{x}_i)$ and $\mathbf{h}_j = f(\mathbf{x}_j)$. The function f is implemented by every subnetwork in the SNN.

A standard architecture of the SNN given in the literature (see, for example, [10]) is shown in Figure 1. It consists of two identical neural subnets which are trained to compare a pair of feature vectors in terms of their semantic similarity or dissimilarity.

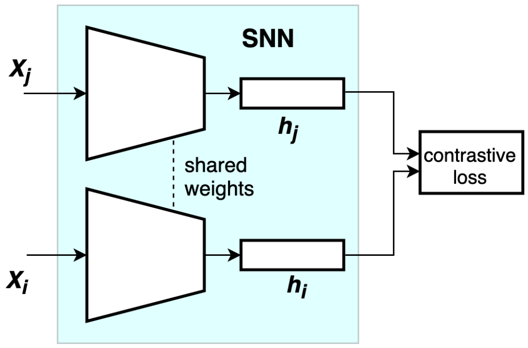


Figure 1. The architecture of the SNN

It should be noted that there are many specific loss function for training the SNN [6, 53, 8], which solve the problem of the object comparison. One of the functions is the contrastive loss function defined as

$$l(\mathbf{x}_i, \mathbf{x}_j, z_{ij}) = \begin{cases} \|\mathbf{h}_i - \mathbf{h}_j\|_2^2, & z_{ij} = 0, \\ \max(0, \tau - \|\mathbf{h}_i - \mathbf{h}_j\|_2^2), & z_{ij} = 1, \end{cases} \tag{1}$$

where τ is a predefined threshold.

Hence, the total error function for minimizing is defined as

$$L_{\text{Siam}}(W) = \sum_{(i,j)\in K} l(\mathbf{x}_i, \mathbf{x}_j, z_{ij}) + \mu R(W). \tag{2}$$

Here $R(W)$ is a regularization term added to improve generalization of the neural network; W is the matrix of the neural subnet parameters; μ is a hyper-parameter which controls the strength of the regularization.

It is assumed below that the outputs $\mathbf{h}_i$ and $\mathbf{h}_j$ are known for every pair $(\mathbf{x}_i, \mathbf{x}_j)$.

4 A GENERAL IDEA FOR EXPLAINING THE SNN

First of all, we have to define what is the meaning of the semantically similar and dissimilar instances from the interpretation point of view. In other words, we have to explain why two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ are semantically similar or dissimilar, i.e., which features of the instances make them similar or dissimilar. This is not a trivial question because the similarity of two instances and the distance between them in the input space may be not correlated. Therefore, it is difficult to apply various techniques, for example perturbation schemes, to the input examples. However, the similarity and the distance can be considered in the output space, where the distance between embeddings $\mathbf{h}_i$ and $\mathbf{h}_j$ corresponding to similar examples is supposed to be rather small, and the distance between embeddings corresponding to dissimilar examples is large. This implies that there are features of the vectors $\mathbf{h}_i$ and $\mathbf{h}_j$, which determine the similarity of input instances by comparing the corresponding distances between these features. These features can be called important features.

In order to explain the important features defining the semantic similarity and dissimilarity, we consider a simple example. Suppose that embeddings consist of two features h_1 and h_2, i.e., $D = 2$. Three 2-dimensional vectors are shown in Figure 2 in the form of small circles and a triangle. It can be seen from Figure 2 that the first and the second points correspond to semantically similar instances because they are close to each other. It is obvious that the first and the third points are semantically dissimilar because they are far from each other. The semantic similarity of points 1 and 2 is defined by feature h_2 because distance d_{similar} is smallest. This implies that the second feature can be viewed as important for semantic similarity of two instances. It should be noted that points 1 and 2 are close to each other also due to feature h_1. However, its impact is smaller in comparison with the impact of feature h_2. We can say the same about the first and the third points. They are dissimilar due to feature h_1 because the large distance $d_{\text{dissimilar}}$ defines the semantic dissimilarity of the instances. This implies that the first feature can be viewed as important for semantic dissimilarity of instances 1 and 3.

In sum, we have a rule for determining important features of embeddings, which define semantic similarity and dissimilarity of input examples. The main problem is that we do not know how these important features of embeddings are connected with the corresponding original instances because there is no an "inverse" map from embeddings to input instances. Therefore, in order to solve the interpretation problem, we have to construct this "inverse" map and to find features of the input instances which correspond to important features of embeddings. If we had such the "in-

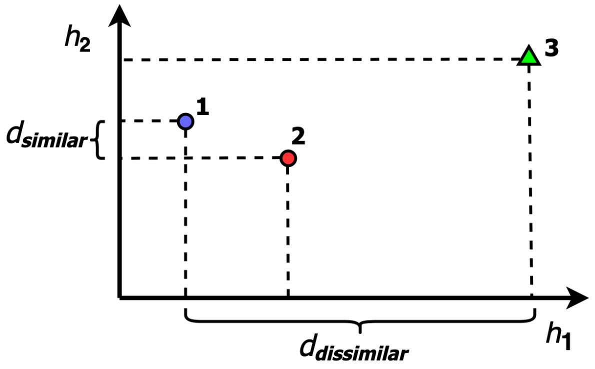


Figure 2. Pairs of semantically similar (1 and 2) and dissimilar (1 and 3) instances whose embeddings consist of two features (h_1 and h_2)

verse" map, then the problem would be solved by perturbing the important features of embeddings and analyzing the corresponding changes of the input instances.

One of the ways for implementing the "inverse" map is to train a reconstruction neural network with embeddings $\mathbf{h}$ as inputs and the corresponding input instances $\mathbf{x}$ as outputs. However, our experiments have shown that this reconstruction network is overfitted and does not allow to correctly reconstruct the input instances especially when the number of training instances is not large and the instances $\mathbf{x}$ are images of a high dimensionality. It turns out that it is simpler to train an AE and then to use its pre-trained decoder part for additional training and for reconstruction. Therefore, our idea is to train the AE whose inputs are instances $\mathbf{x}$, its code (the hidden representation) is close to the embedding vector $\mathbf{h}$. The trained decoder part of the AE can be used as a pre-trained reconstruction neural network which transforms embeddings $\mathbf{h}$ into instances $\mathbf{x}$. Moreover, this reconstruction neural network can be additionally trained by using embeddings $\mathbf{h}$ and instances $\mathbf{x}$.

Having important features of vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ and the trained reconstruction neural network, we perturb the features in accordance with the following rules. If the pair of similar instances is analyzed, then the features are perturbed to reduce the distance between these important features. The perturbation of important features of dissimilar instances is carried out to increase the distance between them. Changes of the reconstructed instances produce the corresponding heatmaps explaining similarity or dissimilarity of two input instances.

Perturbations aim at describing how the output of the explained model changes when one or more input features are perturbed. The intuition of the technique is that the more a model's response depend on a feature, the more predictions or some output score change with the corresponding feature changes. The perturbation scheme can be regarded as one of the interesting approaches to the model interpretation development and to the explainer evaluation [44]. Suppose there is a feature vector $\mathbf{x} \in \mathbb{R}^m$ that is slightly perturbed to a new vector $\mathbf{x} + \delta$, where δ is a small perturbation that does not alter the meaning of the data point, i.e., the vector $\mathbf{x} + \delta$ remains the similarity relationship with other vectors from the training set without changes. The perturbation scheme can be also viewed in the framework of a sensitivity analysis method which aims to consider how the output of the explainable model changes when one or more input features are perturbed. Perturbation methods have the advantages of a straightforward interpretation, as they are a direct measure of the marginal effect of some input features to the output [40]. Moreover, perturbation schemes can be simply implemented, and they can be applied to post-hoc models.

Finally, the proposed method for explaining the SNN can be represented by means of an algorithm consisting of two parts. The first part aims to train the additional AE with a special loss function, which plays a partial role of the explainer. It aims to reconstruct the input instances from the training set and to take into account the SNN output. The second part is to train the decoder of the pre-trained AE, to perturb the embedding vectors at the SNN output, to use the decoder in order to reconstruct the perturbed embeddings and to observe the features of the reconstructed vectors which are changed due to the perturbation of embeddings.

Let us consider every part of the above algorithm in detail. Suppose that we have a trained SNN as a black-box model. For every input instance $\mathbf{x}_i$, we have the corresponding embedding vector $\mathbf{h}_i \in \mathbb{R}^D$ such that $\mathbf{h}_i = f(\mathbf{x}_i)$. For the sake of clarity, we will call instances $\mathbf{x}_i$ as images whereas the embeddings will be called as vectors.

5 PRE-TRAINING OF THE AE

Suppose that inputs of the proposed AE are images $\mathbf{x}_i$. Then we expect to get reconstructed images $\mathbf{x}_i^*$ as its outputs. The corresponding loss function $L_{\text{recon\_AE}}$ for training the AE is defined, for example, as follows:

$$L_{\text{recon\_AE}}(W, \mathbf{x}_i, \mathbf{x}_i^*) = \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{x}_i^*\|_2^2 . \tag{3}$$

A regularization term is not written here because it will be used later. In order to use the pre-trained decoder for reconstruction of vector $\mathbf{h}_i \in \mathbb{R}^D$, the AE has to be trained in a special way. First of all, the length of a part of its hidden representation has to coincide with the length of vector $\mathbf{h}$, which is equal to D. Second, the loss function should take into account proximity of vectors $\mathbf{h}_i$ and the corresponding vectors of the AE hidden representation denoted as $\mathbf{b}_i \in \mathbb{R}^D$, i.e., we need to have the vectors $\mathbf{b}_i$ in the hidden layer coinciding with the vectors $\mathbf{h}_i$ obtained by means of the SNN. Therefore, we propose to change the loss function for training the AE by adding the loss function L_{close} in the following way:

$$L_{\text{autoen}}(W) = \gamma L_{\text{recon\_AE}}(W, \mathbf{x}_i, \mathbf{x}_i^*) + \mu L_{\text{close}}(W, \mathbf{h}_i, \mathbf{b}_i) + \lambda R(W)$$
$$= \gamma \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{x}_i^*\|_2^2 + \mu \sum_{i=1}^{n} \|\mathbf{h}_i - \mathbf{b}_i\|_2^2 + \lambda R(W). \qquad (4)$$

Here $R(W)$ is a regularization term, λ is a hyper-parameter which controls the strength of the regularization; W is the set of the neural network weights; γ and μ are parameters that control the interaction of the loss function terms.

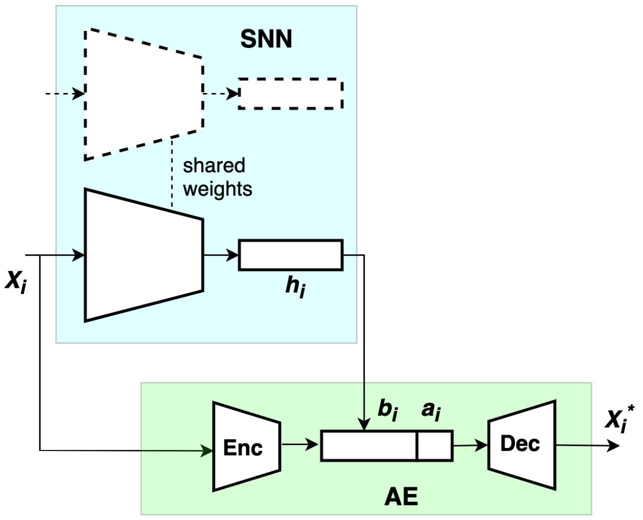


Figure 3. The autoencoder training scheme

One of the problems here is a case when D is small. Hence, the AE may provide the unsatisfactory reconstruction if the hidden representation is of a too small size. Therefore, for improving the presented architecture, it is proposed to enlarge the hidden representation of the AE, i.e., the vector $\mathbf{b}_i$ by means of its concatenation with a vector $\mathbf{a}_i \in \mathbb{R}^A$. As a result, we get the vector $\mathbf{c}_i = (\mathbf{b}_i||\mathbf{a}_i) \in \mathbb{R}^{D+A}$, where $(\mathbf{b}_i||\mathbf{a}_i)$ denotes the concatenation operation of vectors $\mathbf{b}_i$ and $\mathbf{a}_i$. The enlarged embedding allows us to improve the decoder training. In the same way, we later enlarge the outputs of the SNN, which contain vectors $(\mathbf{h}_i||\mathbf{a}_i) \in \mathbb{R}^{D+A}$ of the same dimensionality. Before training the AE, we assume that the vector $\mathbf{a}_i$ concatenated with the SNN output is arbitrary, for example, with zero-valued elements.

A scheme of the first part of the explanation algorithm is shown in Figure 3. It can be seen from the scheme that the AE is trained by using embedding vectors $\mathbf{h}_i$ from subnetworks of the SNN and the input images $\mathbf{x}_i$.

The pre-trained decoder part as well as the trained encoder part of the AE can be used for additional training and for reconstruction of the perturbed embeddings that is for implementing the second part of the algorithm. The use of the AE allows us to significantly simplify the training process and to get acceptable vector reconstructions. It should be noted that an architecture of the encoder differs from the architecture of a subnetwork of the SNN because we consider the SNN as a black box whose architecture is unknown.

6 PERTURBATION OF EMBEDDINGS AND THE INSTANCE RECONSTRUCTION

The second part of the explanation algorithm, including the perturbation and the image reconstruction is shown in Figure 4.

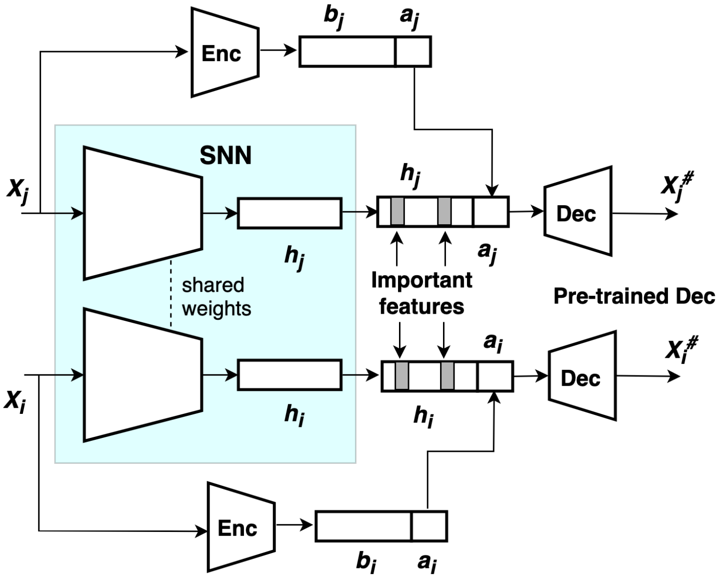


Figure 4. A scheme of the second part of the explanation algorithm

The pre-trained decoder can be again trained by using only vectors $\mathbf{h}_i$ from the SNN output and vectors $\mathbf{a}_i$ from the AE encoder output. The concatenated vector $(\mathbf{h}_i||\mathbf{a}_i)$ is the decoder input, the reconstructed image $\mathbf{x}_i^\#$ is the decoder output. The loss function is the standard Euclidean distance between images $\mathbf{x}_i^\#$ and $\mathbf{x}_i$. It is important to note that the vector $\mathbf{a}_i$ is computed by means of the encoder part of the AE. The set of all vectors $\mathbf{a}_i$ corresponding to all training instances could be separately stored in order to avoid the repeated use of the encoder. However, this can be done only for training. When we have new instances, the encoder has to be used. If the SNN outputs are not so small, then the AE hidden representation can have the same size as vectors $\mathbf{h}_i$. In this case, $A = 0$ and vectors $\mathbf{a}_i$ as well as the encoder are not needed, and the scheme of the second part of the explanation algorithm is simplified.

For a new pair of images $\mathbf{x}_i$ and $\mathbf{x}_j$, we find vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ as outputs of the SNN. If the images are semantically similar, then we find a predefined number of important features in $\mathbf{h}_i$ and $\mathbf{h}_j$ with smallest distances. There are different ways for choosing important features. The first way is just to define the number of important features $s < D$ such that the index set $J \subseteq \{1, \dots, D\}$ consists of s indices corresponding to smallest distances between features $h_k^{(i)}$ and $h_k^{(j)}$, $k = 1, .., D$. In this case, the value of s can be regarded as a tuning parameter. The second way is to define a threshold α of the relative Euclidean distances $r_k \in [0, 1]$ between important features of two vectors $\mathbf{h}_i$ and $\mathbf{h}_j$. It is supposed that features are important if there holds

$$r_k = \frac{\left| h_k^{(i)} - h_k^{(j)} \right|}{\max_{l=1,\dots,D} \left| h_l^{(i)} - h_l^{(j)} \right|} \leq \alpha, k = 1, \dots, D. \tag{5}$$

Then the index set J of important features is defined as

$$J = \{k : r_k \leq \alpha\}. \tag{6}$$

In the same way, we define the rule for important features of the semantically dissimilar images. In particular, features in $\mathbf{h}_i$ and $\mathbf{h}_j$ with largest distances can be viewed ad important features.

By having a set of important features, we can perturb them to study how the important features of the SNN outputs impact on the original images $\mathbf{x}_i$ and $\mathbf{x}_j$. The trained decoder is used to reconstruct images from $\mathbf{h}_i + \delta_i$ and $\mathbf{h}_j + \delta_j$ and to investigate how features of the reconstructed images $\mathbf{x}_i^{\#}$ and $\mathbf{x}_j^{\#}$ are changed. Here δ_i and δ_j are the perturbation vectors such that indices of their non-zero elements are from the index set J, other elements are equal to zero. In sum, we have the embeddings $\mathbf{h}_i$ and $\mathbf{h}_j$, the reconstructed images $\mathbf{x}_i^{\#}$ and $\mathbf{x}_j^{\#}$, the index set J of important features of embeddings. Important features as an example (two features) in $\mathbf{h}_i$ and $\mathbf{h}_j$ are shown by dashed cells in Figure 4. The perturbed vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ are fed to the corresponding decoders in order to get the reconstructed images $\mathbf{x}_i^{\#}$ and $\mathbf{x}_j^{\#}$ which depends on perturbations.

Suppose that the perturbation δ of an embedding leads to the changed i^{th} feature $x_i^{\#}(\delta)$ of the reconstructed image $\mathbf{x}^{\#}$. After generating the random vector δ many times, say N times, the mean value of the i^{th} feature changes is defined as

$$T_i = N^{-1} \sum_{j=1}^{N} \left(x_i^{\#}(\delta_j) - x_i^{\#} \right). \tag{7}$$

It should be noted that T_i may be positive as well as negative. If we consider the visual interpretation, then all values of T_i are scaled to be in interval $[-1, 1]$.

Finally, we compute absolute values T_i^* of T_i in order to visualize the largest changes. The heatmaps explaining the considered instances are determined from condition $T_i^* \geq \beta$, i.e., they have largest changes of the reconstructed instance. Here β is a threshold of relative changes of features, which can be regarded as another tuning parameter.

The perturbation vectors are randomly generated in the following way. Suppose that the index set J consists of s elements. First, we generate the vector δ in the s-sphere defined as $B = \{\delta \in R^s : |\delta| = R\}$ with some predefined radius R. There are several methods for the uniform sampling of points δ in the s-sphere with the unit radius $R = 1$, for example, [87, 88]. Then every generated point is multiplied by R. Moreover, we take vectors δ only from a part of the s-sphere. This part is defined by a hyper-quadrant, which the second embedding $\mathbf{h}_j$ is located in, under condition that the vector $\mathbf{h}_i$ is in the origin of coordinates. The radius is defined as a portion of the Euclidean distance $d(\mathbf{h}_j, \mathbf{h}_i)$ between $\mathbf{h}_j$ and $\mathbf{h}_i$, i.e., $R = q \cdot d(\mathbf{h}_j, \mathbf{h}_i)$, where q is also a tuning parameter.

7 NUMERICAL EXPERIMENTS

7.1 A Synthetic Example

One of the questions of the SNN explanation is to understand how the instances may be semantically similar or dissimilar and how to explain the important features of the instances, which are responsible for the semantics. This question is not trivial. Indeed, we simply and logically understand the similarity or dissimilarity at the embedding level because they depend on the distance between vectors. However, the similarity or dissimilarity of images are semantic and, therefore, not obvious. In order to see what the similarity and dissimilarity mean for images, we consider a synthetic example. We consider two types of randomly generated images: triangles and circles. Sizes of triangles and circles may be different and random. All images are of 28×28 pixels.

The SNN as well as the AE are implemented in Python by using the Keras package with Tensorflow. They are trained on the generated set of images with circles and triangles. The length of the hidden representation layer of the AE is 41, i.e., vector $\mathbf{h}$ consists of 32 features ($D = 32$), and vector $\mathbf{a}$ is of the length 9. The hyper-parameter μ, which controls the strength of the regularization is 0.02. The loss function for training the SNN is of the form:

$$l(\mathbf{x}_i, \mathbf{x}_j, z_{ij}) = \begin{cases} \left(\|\mathbf{h}_i - \mathbf{h}_j\|_2^2 - \omega\right)^2, & z_{ij} = 0, \\ \left(\max(0, \tau - \|\mathbf{h}_i - \mathbf{h}_j\|_2^2)\right)^2, & z_{ij} = 1. \end{cases} \tag{8}$$

Here ω is an intraclass margin which is introduced to diverse instances of the same class. It can be seen from (8) that the loss function differs from (1). The function (8) is used because embeddings of similar instances are too close to each other. A simple way to make them different is to introduce the margin ω which sets the minimal distance between embeddings equal to ω.

Numerical results are shown in Figures 5, 6, 7, 8 such that every figure contains numerical results of 6 experiments depending on the value of important features s for perturbation. Every experiment is represented by means of four pictures or two pairs of pictures. The first pair (vertically) consists of original images, whose similarity or dissimilarity has to be explained, overlapped by the corresponding heatmaps. The second pair of pictures is the heatmaps of features explaining the similarity or dissimilarity. It is important to note that the heatmaps are obtained by using the pre-trained autoencoder.

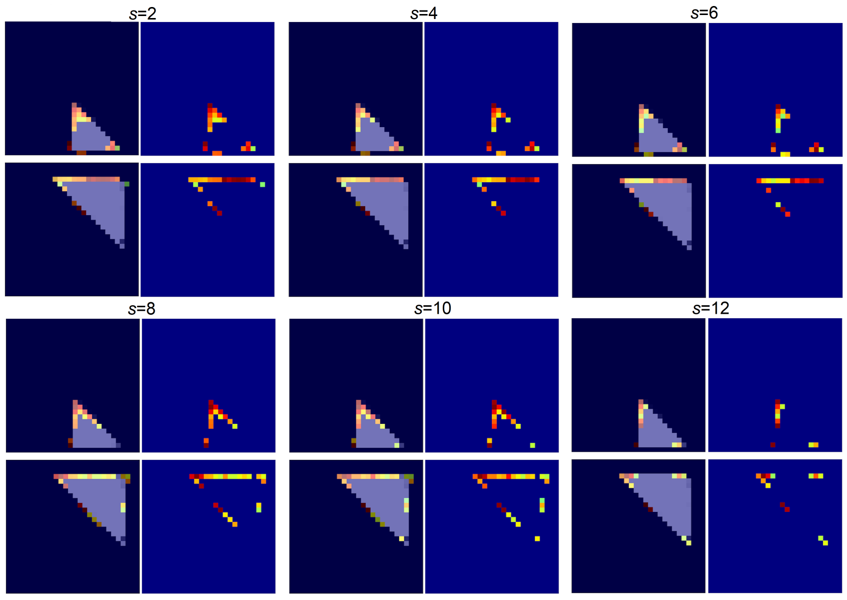


Figure 5. Examples of semantically similar images (triangles) and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

It can be seen from Figure 5 that the semantic similarity of triangles is defined starting from angles of the triangles (see the corresponding pictures of heatmaps by $s = 2$). This is a very interesting result. Indeed, the triangles are intuitively similar by angles. Sides of triangles also play some role in explanation, but they may be like some parts of small circles due to the low resolution of images. It can be seen from pictures by increasing the value of $s = 4, 6, 8$. We get sides of triangles as important features. It is interesting to note that insides of the triangles almost do not participate in the explanation though these are solid on the original images and differ from the background color. It is also interesting to see that that increasing of s does not necessarily leads to increasing the important features of the reconstructed images. For example, the number of important features by $s = 12$ is less than by

$s = 10$. This is due to the fact that some new perturbations mask changes the previous perturbations.

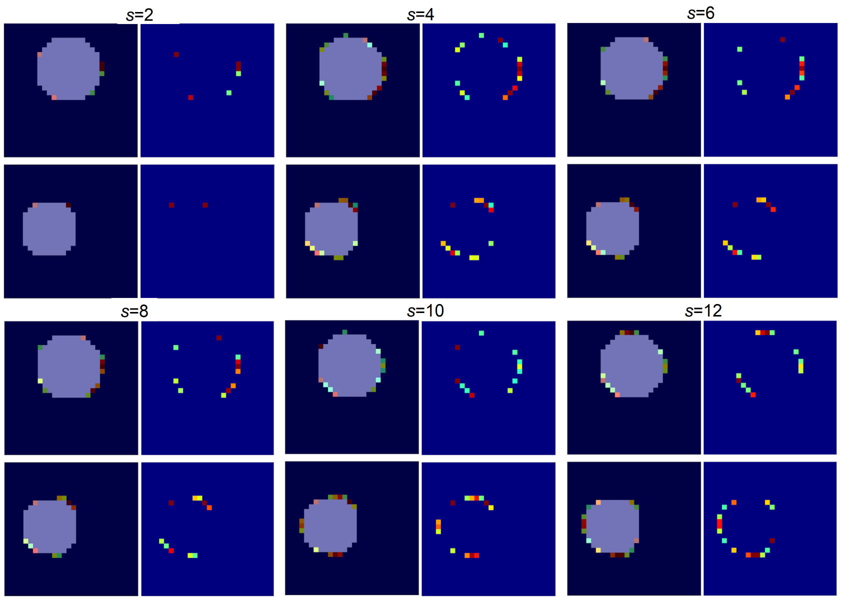


Figure 6. Examples of semantically similar images (circles) and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

When we look at Figure 6, it can be seen that the circles are like rectangles with chamferings due to the low resolution. It is also an interesting case, because the similarity is observed mainly in chamferings, and it covers sides only partially by increasing s.

But the most interesting cases are shown in Figures 7 and 8, where the semantic dissimilarity is studied. Two cases are studied:

1. The original triangle is small and the circle is large.
2. The original triangle is large and the circle is small.

It is clearly seen from the first pictures that the dissimilarity is explained by angles of triangles and by chamferings of circles. Indeed, triangles as well as "rectangles-circles" have sides as similar elements. They do not explain the semantic dissimilarity. Only angles of triangles and chamferings are really different. The sides of triangles and the whole circles become important only by large values of s. We intentionally consider the images of the low resolution in order to see some peculiarities of explaining the "wrong" rectangle and the "wrong" circle. It is again important to point out that insides of the pictures do not participate in the explanation.

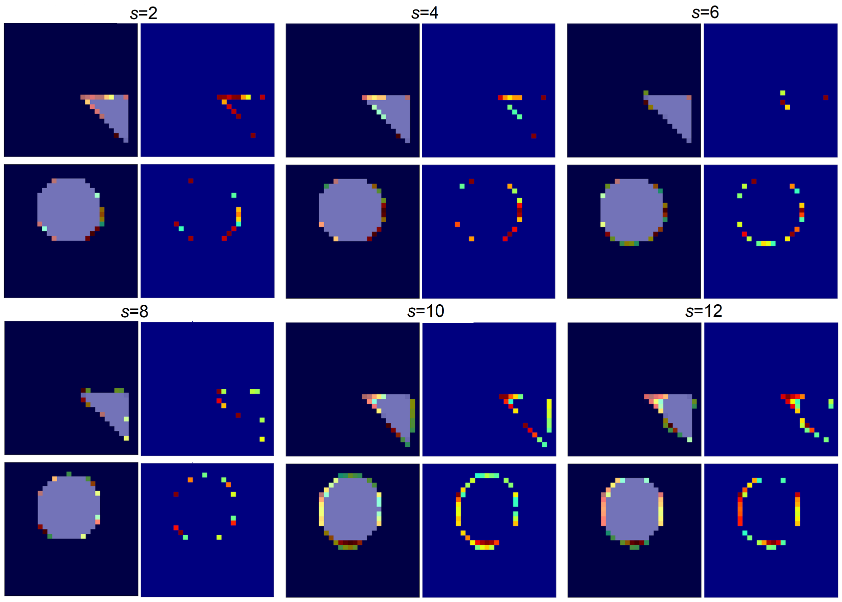


Figure 7. Examples of semantically dissimilar images (large circles and small triangles) and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

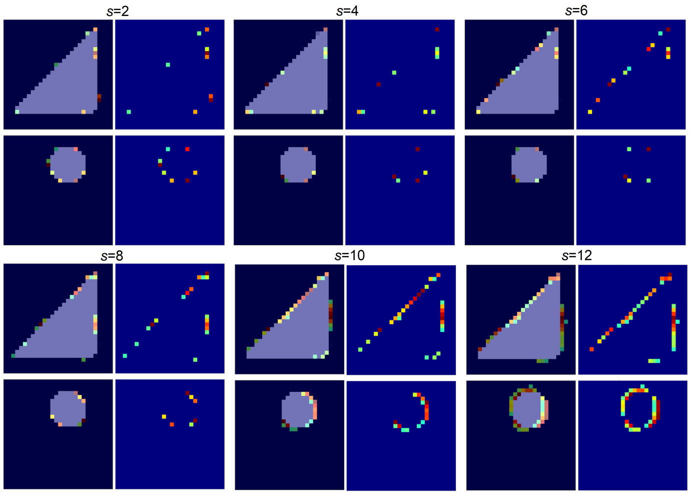


Figure 8. Examples of semantically dissimilar images (small circles and large triangles) and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

Architectures of the SNN and the AE are not provided for these numerical examples because they are stated to understand the meaning of the similarity and dissimilarity explanations. However, the corresponding architectures for explaining results obtained on the MNIST dataset are given below.

7.2 MNIST

The proposed explanation method is studied by applying the MNIST dataset which is a commonly used large dataset of 28x28 pixel handwritten digit images [89]. It has a training set of 60 000 instances, and a test set of 10 000 instances. The digits are size-normalized and centered in a fixed-size image. The dataset is available at `http://yann.lecun.com/exdb/mnist/`. If two digits from the MNIST dataset are identical, i.e., they belong to the same class, then they are semantically similar. If two digits are different, then they are semantically dissimilar.

7.3 Architecture of Neural Networks

Layer	Dimension	Activation
Input	28×28	–
Conv1	$28 \times 28 \times 4$	ReLU
Pooling1	$14 \times 14 \times 4$	–
Flatten	784	–
Dense1	128	ReLU
Output (Dense2)	20	Tanh

Table 1. An architecture of every subnetwork of the SNN for the MNIST dataset

Layer	Dimension	Activation
Input	28×28	–
Conv1	$28 \times 28 \times 8$	ReLU
Pooling1	$14 \times 14 \times 8$	–
Conv2	$14 \times 14 \times 16$	ReLU
Pooling2	$7 \times 7 \times 16$	–
Flatten	784	–
Hidden	26	Tanh
Dense	784	ReLU
Reshape	$7 \times 7 \times 16$	–
Up-Sampling	$14 \times 14 \times 16$	–
Conv3	$14 \times 14 \times 8$	ReLU
Up-Sampling	$28 \times 28 \times 8$	–
Output (Conv4)	20	Sigmoid

Table 2. An architecture of the AE for the MNIST dataset

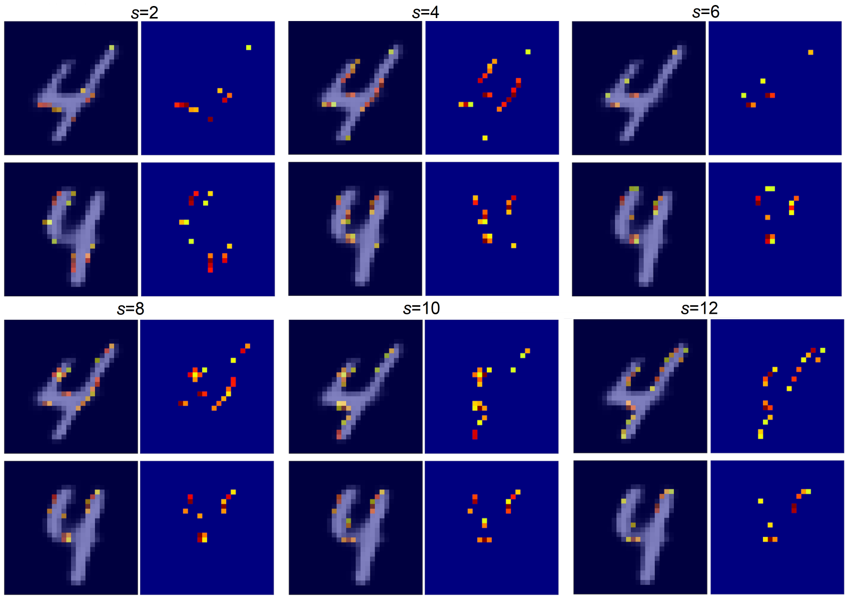


Figure 9. Examples of semantically similar images of digits 4 from the MNIST dataset and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

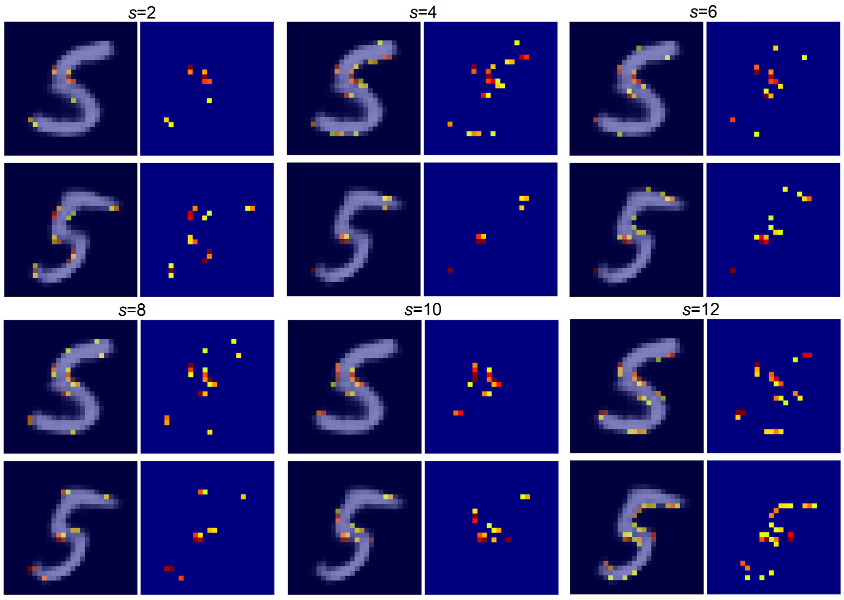


Figure 10. Examples of semantically similar images of digits 5 from the MNIST dataset and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

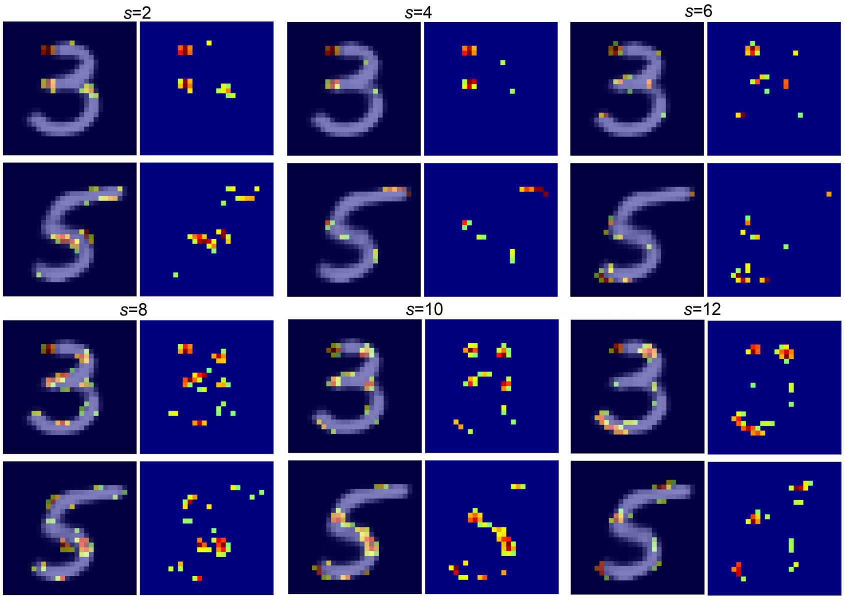


Figure 11. Examples of semantically dissimilar images of digits 3 and 5 from the MNIST dataset and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

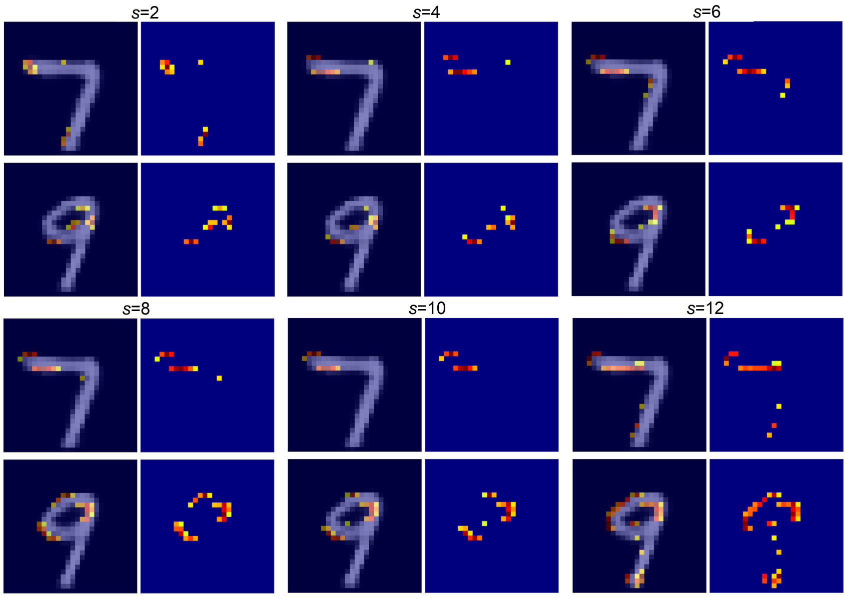


Figure 12. Examples of semantically dissimilar images of digits 7 and 9 from the MNIST dataset and the explanation heatmaps for $s = 2, 4, 6, 8, 10, 12$

An architecture of the SNN for experiments with the MNIST dataset is shown in Table 1. Every subnetwork is implemented as a convolutional network having a convolutional layer (Conv1), a max pooling layer (Pooling1), flatten layer (Flatten), which transforms a two-dimensional matrix into a vector, two dense layers (Dense1 and Dense2), which are represented by fully connected networks. Parameters of the loss function (8) are $\mu = 0.0005$, $\omega = 0.5$, $\tau = 3$.

An architecture of the AE is shown in Table 2. The encoder part consists of two convolutional layers (Conv1 and Conv2), two max pooling layers (Pooling1 and Pooling2), flatten layer (Flatten). The decoder part consists of a dense layer (Dense), a reshape layer to change dimensions (Reshape), two upsampling layers and two convolutional layers (Conv3 and Conv4).

The length of the hidden representation layer is 26, i.e., vector $\mathbf{h}$ consists of 20 features ($D = 20$), and vector $\mathbf{a}$ is of the length 6.

7.4 Results

Numerical results illustrating the explanation method on the MNIST dataset are shown in Figures 9, 10, 11, 12. The figures have the same structure as Figures 5, 6, 7, 8, i.e., every figure contains numerical results of 6 experiments depending on the value of important features s for perturbation.

Figures 9 and 10 show semantically similar digits 4 and 5, respectively. It can be seen from pictures in Figure 9 that the selected features indicate peculiarities of the digit 4, which differ from other digits. In particular, the important features are located at the upper part of the digit and its middle part. The same can be said about the digit 5 shown in Figure 10. The important features are concentrated at the middle part of the digits and partially select their upper curve. We again see from Figures 9 and 10 that the semantic similarity of pairs of original images is clearly exhibited by means of important features which explain this similarity.

Figures 11 and 12 show semantically dissimilar digits 3,5 and 7,9, respectively. The explanation of the semantic dissimilarity is very explicit in Figure 11. Indeed, the selected important features are inherent in the difference of two digits. Figure 12 is also very demonstrative. It can be seen from Figure 12 that only the parts of digits 7 and 9 are selected for explanation that characterize differences between the different digits. It is interesting to note that the slanting vertical slashes are not selected for explanation because they are common for these two digits. They are only partly selected when $s = 12$, i.e., features of embeddings responsible for the semantic similarity begin to act.

It can be also concluded from all the considered examples that the choice of a proper value of parameter s is not a trivial task. It can be solved by enumerating several values s and depends on many factors: datasets, the length of embeddings, the dimensionality of images, etc.

8 CONCLUSION

A new method for explaining the SNN results under condition of the weakly supervised learning has been presented in this paper. Basic ideas behind the method are comparisons of the explained instances at the embedding level and reconstruction of the embedding feature vectors by means of the separately trained decoder and the encoder of the AE in order to analyze the impact of the embedding vector perturbations on the reconstructed features of original instances. It should be noted that the main elements of the proposed method such as the AE with the extended loss function and the perturbation technique can be implemented independently of a structure of the explained SNN. This implies that the proposed method can be applied to various applications using SNNs. The method can be also used when there is information about classes of training data. This case can be viewed as a special case of the considered explanation approach.

To the best of our knowledge, there are no explanation methods applied to the SNN. Therefore, we do not compare the proposed method with the well-known methods such as LIME, SHAP, etc.

One of the limitations of the proposed approach is a possible small amount of training data in order to train the AE. The SNN is trained on pairs of instances such that the number of pairs may be large even by a small number of original instances. However, the AE has to be trained only by using the available data. One of the ways to overcome this problem is to train the AE on concatenated pairs of original instances. However, our experiments have shown that this obvious way may lead to the unsatisfactory reconstruction. Therefore, a modification of the method taking into account the lack of the sufficient amount of training data is a direction for further research. Another limitation is a rather large number of tuning parameters, including the number of important features, the length of the AE hidden representation, etc. Some efficient rules for restricting their values can be also regarded as a direction for further research.

It is important to note that the idea to reconstruct original instances from embeddings by means of the AE with the modified loss function can be applied not only to SNNs, but to different neural networks which implement the feature extraction procedures. If we have information about output feature extracted vectors of the neural networks, then the corresponding results can be explained in the same way. The main difference is that prototypes of classes should be used in order to select the important extracted features instead of output vectors of the SNN. In other words, in order to explain an original instance, the corresponding feature extracted vector is compared with the prototype of the class of this instance, and the nearest features are selected for their perturbation. In the same way, the counterfactual explanation technique can be applied by considering the prototypes of other classes. A detailed study of these extensions of the proposed method is another direction for further research.

Acknowledgement

The reported study was funded by RFBR, project No. 20-01-00154.

REFERENCES

[1] Guidotti, R.—Monreale, A.—Ruggieri, S.—Turini, F.—Giannotti, F.—Pedreschi, D.: A Survey of Methods for Explaining Black Box Models. ACM Computing Surveys, Vol. 51, 2019, No. 5, Art. No. 93, doi: 10.1145/3236009.

[2] Arya, V.—Bellamy, R. K. E.—Chen, P. Y.—Dhurandhar, A.—Hind, M.—Hoffman, S. C.—Houde, S.—Liao, Q. V.—Luss, R.—Mojsilović, A.—Mourad, S.—Pedemonte, P.—Raghavendra, R.—Richards, J.—Sattigeri, P.—Shanmugam, K.—Singh, M.—Varshney, K. R.—Wei, D.—Zhang, Y.: One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques. arXiv:1909.03012, 2019.

[3] Molnar, C.: Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. Published online, `https://christophm.github.io/interpretable-ml-book/`, 2019.

[4] Murdoch, W. J.—Singh, C.—Kumbier, K.—Abbasi-Asl, R.—Yu, B.: Interpretable Machine Learning: Definitions, Methods, and Applications. PNAS, Vol. 116, 2019, No. 44, pp. 22071–22080, doi: 10.1073/pnas.1900654116.

[5] Ibrahim, M.—Louie, M.—Modarres, C.—Paisley, J. W.: Global Explanations of Neural Networks: Mapping the Landscape of Predictions. Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society (AIES '19), 2019, pp. 279–287, doi: 10.1145/3306618.3314230.

[6] Bellet, A.—Habrard, A.—Sebban, M.: A Survey on Metric Learning for Feature Vectors and Structured Data. arXiv:1306.6709, 2013.

[7] Kulis, B.: Metric Learning: A Survey. Foundations and Trends in Machine Learning, Vol. 5, 2013, No. 4, pp. 287–364, doi: 10.1007/s11042-015-2847-3.

[8] Zheng, L.—Duffner, S.—Idrissi, K.—Garcia, C.—Baskurt, A.: Siamese Multi-Layer Perceptrons for Dimensionality Reduction and Face Identification. Multimedia Tools and Applications, Vol. 75, 2016, No. 9, pp. 5055–5073, doi: 10.1007/s11042-015-2847-3.

[9] Bromley, J.—Bentz, J.—Bottou, L.—Guyon, I.—LeCun, Y.—Moore, C.—Sackinger, E.—Shah, R.: Signature Verification Using a "Siamese" Time Delay Neural Network. International Journal of Pattern Recognition and Artificial Intelligence, Vol. 7, 1993, No. 4, pp. 669–688, doi: 10.1142/S0218001493000339.

[10] Chopra, S.—Hadsell, R.—LeCun, Y.: Learning a Similarity Metric Discriminatively, with Application to Face Verification. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), Vol. 1, 2005, pp. 539–546, doi: 10.1109/CVPR.2005.202.

[11] Ribeiro, M.—Singh, S.—Guestrin, C.: "Why Should I Trust You?" Explaining the Predictions of Any Classifier. arXiv:1602.04938v3, 2016.

[12] Shankaranarayana, S. M.—Runje, D.: ALIME: Autoencoder Based Approach for Local Interpretability. In: Yin, H., Camacho, D., Tino, P., Tallón-Ballesteros, A., Menezes, R., Allmendinger, R. (Eds.): Intelligent Data Engineering and Automated Learning – IDEAL 2019. Springer, Cham, Lecture Notes in Computer Science, Vol. 11871, 2019, pp. 454–463, doi: 10.1007/978-3-030-33607-3_49.

[13] Ahern, I.—Noack, A.—Guzmán-Nateras, L.—Dou, D.—Li, B.—Huan, J.: NormLime: A New Feature Importance Metric for Explaining Deep Neural Networks. arXiv:1909.04200, 2019.

[14] Zafar, M. R.—Khan, N. M.: DLIME: A Deterministic Local Interpretable Model-Agnostic Explanations Approach for Computer-Aided Diagnosis Systems. arXiv:1906.10263, 2019.

[15] Ribeiro, M. T.—Singh, S.—Guestrin, C.: Anchors: High-Precision Model-Agnostic Explanations. AAAI Conference on Artificial Intelligence, 2018, pp. 1527–1535.

[16] Hu, L.—Chen, J.—Nair, V. N.—Sudjianto, A.: Locally Interpretable Models and Effects Based on Supervised Partitioning (LIME-SUP). arXiv:1806.00663, 2018.

[17] Rabold, J.—Deininger, H.—Siebers, M.—Schmid, U.: Enriching Visual with Verbal Explanations for Relational Concepts – Combining LIME with Aleph. In: Cellier, P., Driessens, K. (Eds.): Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2019). Springer, Cham, Communications in Computer and Information Science, Vol. 1167, 2019, pp. 180–192, doi: 10.1007/978-3-030-43823-4_16.

[18] Kovalev, M. S.—Utkin, L. V.—Kasimov, E. M.: SurvLIME: A Method for Explaining Machine Learning Survival Models. Knowledge-Based Systems, Vol. 203, 2020, Art. No. 106164, doi: 10.1016/j.knosys.2020.106164.

[19] Garreau, D.—von Luxburg, U.: Explaining the Explainer: A First Theoretical Analysis of LIME. Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, PMLR, Vol. 108, 2020, pp. 1287–1296.

[20] Štrumbelj, E.—Kononenko, I.: An Efficient Explanation of Individual Classifications Using Game Theory. Journal of Machine Learning Research, Vol. 11, 2010, pp. 1–18.

[21] Aas, K.—Jullum, M.—Løland, A.: Explaining Individual Predictions When Features Are Dependent: More Accurate Approximations to Shapley Values. arXiv:1903.10464, 2019.

[22] Ancona, M.—Oztireli, C.—Gross, M.: Explaining Deep Neural Networks with a Polynomial Time Algorithm for Shapley Values Approximation. Proceedings of the 36th International Conference on Machine Learning, PMLR, Vol. 97, 2019, pp. 272–281.

[23] Lundberg, S. M.—Lee, S. I.: A Unified Approach to Interpreting Model Predictions. In: Guzon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.): Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017, pp. 4765–4774.

[24] Owen, A. B.—Prieur, C.: On Shapley Value for Measuring Importance of Dependent Inputs. SIAM/ASA Journal on Uncertainty Quantification, Vol. 5, 2017, pp. 986–1002, doi: 10.1137/16M1097717.

[25] Koh, P. W.—Liang, P.: Understanding Black-Box Predictions via Influence Functions. Proceedings of the 34th International Conference on Machine Learning, PMLR, Vol. 70, 2017, pp. 1885–1894.

[26] Burns, C.—Thomason, J.—Tansey, W.: Interpreting Black Box Models with Statistical Guarantees. arXiv:1904.00045, 2019.

[27] Wachter, S.—Mittelstadt, B.—Russell, C.: Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR. Harvard Journal of Law and Technology, Vol. 31, 2018, pp. 841–887, doi: 10.2139/ssrn.3063289.

[28] Moore, J.—Hammerla, N.—Watkins, C.: Explaining Deep Learning Models with Constrained Adversarial Examples. In: Nayak, A., Sharma, A. (Eds.): PRICAI 2019: Trends in Artificial Intelligence. Springer, Cham, Lecture Notes in Computer Science, Vol. 11670, 2019, pp. 43–56, doi: 10.1007/978-3-030-29908-8_4.

[29] Goyal, Y.—Wu, Z.—Ernst, J.—Batra, D.—Parikh, D.—Lee, S.: Counterfactual Visual Explanations. Proceedings of the 36th International Conference on Machine Learning, PMLR, Vol. 97, 2019, pp. 2376–2384.

[30] Hendricks, L. A.—Hu, R.—Darrell, T.—Akata, Z.: Grounding Visual Explanations. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.): Computer Vision – ECCV 2018. Springer, Cham, Lecture Notes in Computer Science, Vol. 11206, 2018, pp. 269–286, doi: 10.1007/978-3-030-01216-8_17.

[31] Laugel, T.—Lesot, M.-J.—Marsala, C.—Renard, X.—Detyniecki, M.: Comparison-Based Inverse Classification for Interpretability in Machine Learning. In: Medina, J. et al. (Eds.): Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations (IPMU 2018). Springer, Cham, Communications in Computer and Information Science, Vol. 853, 2018, pp. 100–111, doi: 10.1007/978-3-319-91473-2_9.

[32] Liu, S.—Kailkhura, B.—Loveland, D.—Han, Y.: Generative Counterfactual Introspection for Explainable Deep Learning. 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2019, doi: 10.1109/globalsip45357.2019.8969491.

[33] Van Looveren, A.—Klaise, J.: Interpretable Counterfactual Explanations Guided by Prototypes. arXiv:1907.02584, 2019.

[34] Poyiadzi, R.—Sokol, K.—Santos-Rodríguez, R.—De Bie, T.—Flach, P. A.: FACE: Feasible and Actionable Counterfactual Explanations. Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES '20), 2020, pp. 344–350, doi: 10.1145/3375627.3375850.

[35] van der Waa, J.—Robeer, M.—van Diggelen, J.—Brinkhuis, M.—Neerincx, M.: Contrastive Explanations with Local Foil Trees. arXiv:1806.07470, 2018.

[36] Dhurandhar, A.—Chen, P. Y.—Luss, R.—Tu, C. C.—Ting, P.—Shanmugam, K.—Das, P.: Explanations Based on the Missing: Towards Contrastive Explanations with Pertinent Negatives. arXiv:1802.07623v2, 2018.

[37] Dhurandhar, A.—Pedapati, T.—Balakrishnan, A.—Chen, P. Y.—Shanmugam, K.—Puri, R.: Model Agnostic Contrastive Explanations for Structured Data. arXiv:1906.00117, 2019.

[38] Bien, J.—Tibshirani, R.: Prototype Selection for Interpretable Classification. The Annals of Applied Statistics, Vol. 5, 2011, No. 4, pp. 2403–2424, doi: 10.1214/11-AOAS495.

[39] Kim, B.—Rudin, C.—Shah, J.: The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification. In: Ghahramani, Y., Welling, M., Cortes, C., Lawrence, N., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 27 (NIPS 2014), 2014, pp. 1952–1960.

[40] Ancona, M.—Ceolini, E.—Öztireli, C.—Gross, M.: Gradient-Based Attribution Methods. In: Samek, W., Montavon, G., Vedaldi, A., Hansen, L., Müller, K. R. (Eds.): Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Springer, Cham, Lecture Notes in Computer Science, Vol. 11700, 2019, pp. 169–191, doi: 10.1007/978-3-030-28954-6_9.

[41] Fong, R.—Vedaldi, A.: Explanations for Attributing Deep Neural Network Predictions. In: Samek, W., Montavon, G., Vedaldi, A., Hansen, L., Müller, K. R. (Eds.): Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Springer, Cham, Lecture Notes in Computer Science, Vol. 11700, 2019, pp. 149–167, doi: 10.1007/978-3-030-28954-6_8.

[42] Fong, R.—Vedaldi, A.: Interpretable Explanations of Black Boxes by Meaningful Perturbation. Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 3449–3457, doi: 10.1109/ICCV.2017.371.

[43] Petsiuk, V.—Das, A.—Saenko, K.: RISE: Randomized Input Sampling for Explanation of Black-Box Models. arXiv:1806.07421, 2018.

[44] Vu, M. N.—Nguyen, T. D.—Phan, N.—Gera, R.—Thai, M. T.: Evaluating Explainers via Perturbation. arXiv:1906.02032v1, 2019.

[45] Zeiler, M. D.—Fergus, R.: Visualizing and Understanding Convolutional Networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.): Computer Vision – ECCV 2014. Springer, Cham, Lecture Notes in Computer Science, Vol. 8689, 2014, pp. 818–833, doi: 10.1007/978-3-319-10590-1_53.

[46] Du, M.—Liu, N.—Hu, X.: Techniques for Interpretable Machine Learning. Communications of the ACM, Vol. 63, 2019, No. 1, pp. 68–77, doi: 10.1145/3359786.

[47] Adadi, A.—Berrada, M.: Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). IEEE Access, Vol. 6, 2018, pp. 52138–52160, doi: 10.1109/ACCESS.2018.2870052.

[48] Arrieta, A. B.—Díaz-Rodríguez, N.—Del Ser, J.—Bennetot, A.—Tabik, S.—Barbado, A.—Garcia, S.—Gil-Lopez, S.—Molina, D.—Benjamins, R.—Chatila, R.—Herrera, F.: Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges Toward Responsible AI. Information Fusion, Vol. 58, 2020, pp. 82–115, doi: 10.1016/j.inffus.2019.12.012.

[49] Gilpin, L. H.—Bau, D.—Yuan, B. Z.—Bajwa, A.—Specter, M.—Kagal, L.: Explaining Explanations: An Overview of Interpretability of Machine Learning. 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA), 2018, pp. 80–89, doi: 10.1109/DSAA.2018.00018.

[50] Mohseni, S.—Zarei, N.—Ragan, E. D.: A Survey of Evaluation Methods and Measures for Interpretable Machine Learning. arXiv:1811.11839v1, 2018.

[51] Rudin, C.: Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. Nature Machine Intelligence, Vol. 1, 2019, pp. 206–215, doi: 10.1038/s42256-019-0048-x.

[52] Nori, H.—Jenkins, S.—Koch, P.—Caruana, R.: InterpretML: A Unified Framework for Machine Learning Interpretability. arXiv:1909.09223, 2019.

[53] Le Capitaine, H.: Constraint Selection in Metric Learning. arXiv:1612.04853v1, 2016.

[54] Xu, Z.—Weinberger, K. Q.—Chapelle, O.: Distance Metric Learning for Kernel Machines. arXiv:1208.3422, 2012.

[55] Kedem, D.—Tyree, S.—Sha, F.—Lanckriet, G.—Weinberger, K.: Non-Linear Metric Learning. In: Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012, pp. 2582–2590.

[56] Norouzi, M.—Fleet, D. J.—Salakhutdinov, R. R.: Hamming Distance Metric Learning. In: Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012, pp. 1070–1078.

[57] Hoffer, E.—Ailon, N.: Deep Metric Learning Using Triplet Network. In: Feragen, A., Pelillo, M., Loog, M. (Eds.): Similarity-Based Pattern Recognition (SIMBAD 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9370, 2015, pp. 84–92, doi: 10.1007/978-3-319-24261-3_7.

[58] Huang, K.—Jin, R.—Xu, Z.—Liu, C. L.: Robust Metric Learning by Smooth Optimization. Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010), 2010, pp. 244–251.

[59] Li, C.—Georgiopoulos, M.—Anagnostopoulos, G. C.: Kernel-Based Distance Metric Learning in the Output Space. The 2013 International Joint Conference on Neural Networks (IJCNN), IEEE, 2013, pp. 1–8, doi: 10.1109/IJCNN.2013.6706862.

[60] Schultz, M.—Joachims, T.: Learning a Distance Metric from Relative Comparisons. In: Thrun, S., Saul, L., Schölkopf, B. (Eds.): Advances in Neural Information Processing Systems 16 (NIPS 2003), 2003, pp. 41–48.

[61] Weinberger, K. Q.—Saul, L. K.: Distance Metric Learning for Large Margin Nearest Neighbor Classification. Journal of Machine Learning Research, Vol. 10, 2009, pp. 207–244.

[62] Xing, E.—Jordan, M.—Russell, S.—Ng, A.: Distance Metric Learning with Application to Clustering with Side-Information. In: Becker, S., Thrun, S., Obermayer, K. (Eds.): Advances in Neural Information Processing Systems 15 (NIPS 2002), 2002, pp. 505–512.

[63] Yin, X.—Chen, Q.: Deep Metric Learning Autoencoder for Nonlinear Temporal Alignment of Human Motion. 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016, pp. 2160–2166, doi: 10.1109/ICRA.2016.7487366.

[64] Roy, S.—Harandi, M.—Nock, R.—Hartley, R.: Siamese Networks: The Tale of Two Manifolds. Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Vol. 2, 2019, pp. 3046–3055, doi: 10.1109/ICCV.2019.00314.

[65] He, K.—Zhang, X.—Ren, S.—Sun, J.: Deep Residual Learning for Image Recognition. Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[66] Hu, J.—Lu, J.—Tan, Y. P.: Discriminative Deep Metric Learning for Face Verification in the Wild. 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1875–1882, doi: 10.1109/CVPR.2014.242.

[67] Sun, Y.—Chen, Y.—Wang, X.—Tang, X.: Deep Learning Face Representation by Joint Identification-Verification. In: Ghahramani, Y., Welling, M., Cortes, C., Lawrence, N., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 27 (NIPS 2014), 2014, pp. 1988–1996.

[68] Yi, D.—Lei, Z.—Liao, S.—Li, S. Z.: Deep Metric Learning for Person Re-Identification. Proceedings of the 2014 22nd International Conference on Pattern Recognition (ICPR), 2014, pp. 34–39, doi: 10.1109/ICPR.2014.16.

[69] Zhang, C.—Liu, W.—Ma, H.—Fu, H.: Siamese Neural Network Based Gait Recognition for Human Identification. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016, pp. 2832–2836, doi: 10.1109/ICASSP.2016.7472194.

[70] Chen, K.—Salman, A.: Extracting Speaker-Specific Information with a Regularized Siamese Deep Network. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 24 (NIPS 2011), 2011, pp. 298–306.

[71] Jiang, C.—Xiao, J.—Xie, Y.—Tillo, T.—Huang, K.: Siamese Network Ensemble for Visual Tracking. Neurocomputing, Vol. 275, 2018, pp. 2892–2903, doi: 10.1016/j.neucom.2017.10.043.

[72] Zhan, H.—Ni, W.—Yan, W.—Wu, J.—Bian, H.—Xiang, D.: Visual Tracking Using Siamese Convolutional Neural Network with Region Proposal and Domain Specific Updating. Neurocomputing, Vol. 275, 2018, pp. 2645–2655, doi: 10.1016/j.neucom.2017.11.050.

[73] Masana, M.—Ruiz, I.—Serrat, J.—van de Weijer, J.—Lopez, A. M.: Metric Learning for Novelty and Anomaly Detection. arXiv:1808.05492, 2018.

[74] Utkin, L. V.—Zaborovsky, V. S.—Lukashin, A. A.—Popov, S. G.—Podolskaja, A. V.: A Siamese Autoencoder Preserving Distances for Anomaly Detection in Multi-Robot Systems. 2017 International Conference on Control, Artificial Intelligence, Robotics and Optimization (ICCAIRO), Prague, Czech Republic, IEEE, 2017, pp. 39–44, doi: 10.1109/ICCAIRO.2017.17.

[75] Berlemont, S.—Lefebvre, G.—Duffner, S.—Garcia, C.: Class-Balanced Siamese Neural Networks. Neurocomputing, Vol. 273, 2018, pp. 47–56, doi: 10.1016/j.neucom.2017.07.060.

[76] Dhami, D. S.—Kunapuli, G.—Page, D.—Natarajan, S.: Predicting Drug-Drug Interactions from Molecular Structure Images. AAAI Fall Symposium – AI for Social Good, AAAI, 2019, pp. 1–6.

[77] Shaham, U.—Lederman, R. R.: Learning by Coincidence: Siamese Networks and Common Variable Learning. Pattern Recognition, Vol. 74, 2018, pp. 52–63, doi: 10.1016/j.patcog.2017.09.015.

[78] Wang, J.—Fang, Z.—Lang, N.—Yuan, H.—Su, M. Y.—Baldi, P.: A Multi-Resolution Approach for Spinal Metastasis Detection Using Deep Siamese Neural Networks. Computers in Biology and Medicine, Vol. 84, 2017, pp. 137–146, doi: 10.1016/j.compbiomed.2017.03.024.

[79] Fei-Fei, L.—Fergus, R.—Perona, P.: One-Shot Learning of Object Categories. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 28, 2006, No. 4, pp. 594–611, doi: 10.1109/TPAMI.2006.79.

[80] Koch, G.—Zemel, R.—Salakhutdinov, R.: Siamese Neural Networks for One-Shot Image Recognition. Proceedings of the 32nd International Conference on Machine Learning, Lille, France, JMLR: W & CP, Vol. 37, 2015, pp. 1–8.

[81] Snell, J.—Swersky, K.—Zemel, R.: Prototypical Networks for Few-Shot Learning. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.): Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017, pp. 4077–4087.

[82] Triantafillou, E.—Zemel, R.—Urtasun, R.: Few-Shot Learning Through an Information Retrieval Lens. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.): Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017, pp. 2255–2265.

[83] Wang, Y.—Yao, Q.: Few-Shot Learning: A Survey. arXiv:1904.05046v1, 2019.

[84] Wang, Y.—Yao, Q.—Kwok, J.—Ni, L. M.: Generalizing from a Few Examples: A Survey on Few-Shot Learning. arXiv:1904.05046v2, 2019.

[85] Bertinetto, L.—Valmadre, J.—Henriques, J. F.—Vedaldi, A.—Torr, P. H. S.: Fully-Convolutional Siamese Networks for Object Tracking. In: Hua, G., Jégou, H. (Eds.): Computer Vision – ECCV 2016 Workshops. Springer, Cham, Lecture Notes in Computer Science, Vol. 9914, 2016, pp. 850–865, doi: 10.1007/978-3-319-48881-3_56.

[86] Leal-Taixé, L.—Canton-Ferrer, C.—Schindler, K.: Learning by Tracking: Siamese CNN for Robust Target Association. 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2016, pp. 418–425, doi: 10.1109/cvprw.2016.59.

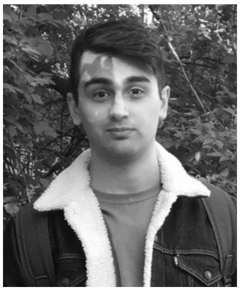
[87] Barthe, F.—Guédon, O.—Mendelson, S.—Naor, A.: A Probabilistic Approach to the Geometry of the ℓ_p^n-Ball. The Annals of Probability, Vol. 33, 2005, No. 2, pp. 480–513, doi: 10.1214/009117904000000874.

[88] Harman, R.—Lacko, V.: On Decompositional Algorithms for Uniform Sampling from n-Spheres and n-Balls. Journal of Multivariate Analysis, Vol. 101, 2010, pp. 2297–2304, doi: 10.1016/j.jmva.2010.06.002.

[89] LeCun, Y.—Bottou, L.—Bengio, Y.—Haffner, P.: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, Vol. 86, 1998, No. 11, pp. 2278–2324, doi: 10.1109/5.726791.

Lev Utkin is Head of the Institute of Computer Science and Technology in Peter the Great Saint Petersburg Polytechnic University, Saint Petersburg, Russia. He is Professor and the Head of the Research Laboratory of Neural Network Technologies and Artificial Intelligence in the same university. In 1986 he graduated from the Saint Petersburg State Electrotechnical University (former Leningrad Electrotechnical Institute). He holds Ph.D. in information processing and control systems (1989) from the same university and D.Sc. in mathematical modelling (2001) from the Saint Petersburg State Institute of Technology, Russia. He was awarded an Alexander von Humboldt Foundation Fellowship (2001–2003). He is a member of the Society for Imprecise Probability Theory and Applications (SIPTA) and the International Society on Multiple Criteria Decision Making (ISMCDM). He is author of more than 300 scientific publications, including AI journals: Neurocomputing, Neural Networks, Knowledge-Based Systems, Applied Soft Computing, AI in Medicine, etc. His research interests are focused on machine learning, imprecise probability theory, decision making.

Maxim Kovalev is Ph.D. student at the Institute of Applied Mathematics and Mechanics in Peter the Great Saint Petersburg Polytechnic University, Saint Petersburg, Russia. He is Research Assistant at the Neural Network Technologies and Artificial Intelligence Laboratory in the same university. In 2019, he graduated from Peter the Great Saint Petersburg Polytechnic University, holding M.Sc. in bioinformatics. His research interests are focused on machine learning, explainable artificial intelligence, computational biology.



Ernest Kasimov is Research Assistant at the Neural Network Technologies and Artificial Intelligence Laboratory in Peter the Great Saint Petersburg Polytechnic University. In 2020, he graduated from the same university, holding M.Sc. in mathematics and computer science. His research interests are focused on machine learning.

Computing and Informatics, Vol. 39, 2020, 1203–1228, doi: 10.31577/cai_2020_6_1203

ERROR ANALYSIS OF THE CHOLESKY QR-BASED BLOCK ORTHOGONALIZATION PROCESS FOR THE ONE-SIDED BLOCK JACOBI SVD ALGORITHM

Shuhei Kudo

RIKEN Center for Computational Science, 7-1-26
Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan
e-mail: shuhei.kudo@riken.jp

Yusaku Yamamoto

Department of Communication Engineering and Informatics
The University of Electro-Communications, 1-5-1, Chofugaoka, Chofu
Tokyo, 182-8585, Japan
e-mail: yusaku.yamamoto@uec.ac.jp

Toshiyuki Imamura

RIKEN Center for Computational Science, 7-1-26
Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan
e-mail: imamura.toshiyuki@riken.jp

Abstract. The one-sided block Jacobi method (OSBJ) has attracted attention as a fast and accurate algorithm for the singular value decomposition (SVD). The computational kernel of OSBJ is orthogonalization of a column block pair, which amounts to computing the SVD of this block pair. Hari proposes three methods for this partial SVD, and we found through numerical experiments that the variant named "V2", which is based on the Cholesky QR method, is the fastest variant and achieves satisfactory accuracy. While it is a good news from a practical viewpoint, it seems strange considering the well-known instability of the Cholesky QR method. In this paper, we perform a detailed error analysis of the V2 variant and explain why and when it can be used to compute the partial SVD accurately. Thus, our results provide a theoretical support for using the V2 variant safely in the OSBJ method.

Keywords: Singular value decomposition, one-sided Jacobi method, error analysis, parallel computing, orthogonalization

Mathematics Subject Classification 2010: 65F15, 65F25, 65G50

1 INTRODUCTION

Let $A \in \mathbb{R}^{m\times n}$, where $m \geq n$, be a dense rectangular matrix and consider computing its singular value decomposition (SVD) $A = U\Sigma V^\top$, where $U \in \mathbb{R}^{m\times n}$ is a matrix with orthonormal columns, $\Sigma \in \mathbb{R}^{n\times n}$ is a diagonal matrix and $V \in \mathbb{R}^{n\times n}$ is an orthogonal matrix. This type of SVD is referred to as the *thin SVD*, in contrast to the full SVD, where $U \in \mathbb{R}^{m\times n}$ and $\Sigma \in \mathbb{R}^{m\times n}$. There are two major approaches for this problem [1]. The first one consists of bi-diagonalization based methods like the QR [2], Divide-and-Conquer [3] and MRRR [4, 5] methods. The second one is the *one-sided Jacobi method* [6], which is an iterative method that starts from $A^{(0)} = A$. At the r^{th} step, the method chooses a pair of columns of $A^{(r)}$ and orthogonalizes them mutually by a Givens rotation [7] from the right, thereby producing $A^{(r+1)}$. If the column pair at each step is chosen judiciously, $A^{(r)}$ converges to a matrix $A^{(\infty)}$ with orthogonal columns. Then, by writing $A^{(\infty)} = U^{(\infty)}\Sigma^{(\infty)}$, where $U^{(\infty)} \in \mathbb{R}^{m\times n}$ is a matrix with orthonormal columns and $\Sigma^{(\infty)} \in \mathbb{R}^{n\times n}$ is a diagonal matrix, and denoting the accumulated Givens matrices by $V^{(\infty)}$, we have $A = U^{(\infty)}\Sigma^{(\infty)}(V^{(\infty)})^\top$, the thin SVD of A. Whereas the bi-diagonalization based approach is generally more efficient in terms of computational work[1], the one-sided Jacobi method has the advantage that small singular values can be computed to high relative accuracy under certain conditions [8]. Such an ability is important in applications like vibration analysis by finite element methods and quantum mechanical calculations, where the smallest singular values are of primary physical interest [9]. Moreover, thanks to the introduction of QR preprocessing [6, 10], the convergence speed of the method has been greatly improved. The numerical properties of the one-sided Jacobi method are well studied and a reliable and accurate SVD solver based on it has been implemented in LAPACK [10].

To further enhance the performance of the one-sided Jacobi method, two techniques, parallelization and blocking, can be employed. At each step of the algorithm, it is possible to orthogonalize multiple column pairs simultaneously as long as they

[1] When $m = n$, the bi-diagonalization based methods require at least $\frac{20}{3}n^3$ floating-point operations (FLOPs), while the OSBJ method requires $6n^3 \times n_{\text{sweep}} + 9n^3$ FLOPs, where n_{sweep} is the number of sweeps (see 2.1.2), which is typically between 1 and 10.

are disjoint, and this brings about inherent parallelism [11]. Blocking refers to orthogonalizing a pair of column blocks instead of a column pair. This requires partial SVD, as we will see later, but greatly enhances the computational intensity by replacing level-1 BLAS like operations such as the Givens rotation by level-3 BLAS operations [12]. The one-sided block Jacobi (OSBJ) method, which adopts both of these improvements, is highly competitive in terms of computational performance and sometimes outperforms the bi-diagonalization based ScaLAPACK SVD routine on modern parallel computers [13]. However, in contrast to the case of the point Jacobi method, still little is known about its convergence and numerical properties.

In this paper, we focus on the mutual orthogonalization of a pair of row blocks, which is a kernel operation in the OSBJ method, and perform its roundoff error analysis. The numerical errors arising in this operation influence both the convergence speed of the algorithm and the accuracy of the final results, so its analysis should be of great importance. However, to the best of our knowledge, no such analysis has been provided so far. There are several algorithms proposed for this mutual orthogonalization. Among them, the LHC method [14] is based on the Householder QR decomposition. On the other hand, Hari et al. propose three methods named V1, V2 and V3 [15]. In this paper, we focus on Hari's V2 method, which is based on the Cholesky QR algorithm. This method is superior to the LHC and V1 method in terms of parallel granularity or computational work and has better (experimental) numerical stability than the V3 method. We perform a detailed roundoff error analysis of the V2 method and derive a bound on the orthogonality of the column block pair updated by the V2 method, as well as a bound on the backward error of orthogonalization. If the QR preprocessing is applied to the OSBJ method, it is observed in many cases that the column-scaled and row-scaled condition numbers of $A^{(k)}$ approach to 1 quickly. Under these conditions, we show that both of the above bounds become $O(\mathbf{u})$, where $\mathbf{u}$ is the unit roundoff. Thus, our analysis will provide a necessary theoretical background for using Hari's V2 method safely in the OSBJ method for the SVD.

The rest of this paper is organized as follows. Section 2 summarizes the overall procedure of the one-sided block Jacobi method, as well as the details of orthogonalization methods of the column block pair. In Section 3, we present the roundoff error analysis of the V2 method for orthogonalization. Numerical results that support our theoretical results are provided in Section 4. Finally, Section 5 concludes the paper.

2 THE ONE-SIDED BLOCK JACOBI METHOD

2.1 The Overall Procedure of OSBJ

The overall procedure of the OSBJ method consists of three parts, namely, preprocessing, the SVD of the preprocessed matrix, and the postprocessing. For the first and the third parts, we use the QR pre/postprocessing proposed by Drmač and adopted in the LAPACK implementation of the one-sided point Jacobi method [10]. This will be explained in 2.1.1 below. The pre/postprocessing switches among several variants depending on the properties of the input matrix A, but here we explain only the most basic version. The SVD of the preprocessed matrix, which is the central part, will be described in 2.1.2.

In this section, we adopt the MATLAB notation for submatrices. Thus, for example, the j^{th} column vector of a matrix A is denoted by $A(:,j)$. The 2-norm condition number of A is denoted by $\kappa_2(A)$.

2.1.1 QR Pre/Postprocessing

The goal of QR pre/postprocessing is to reduce the condition number of the input matrix A, thereby accelerating the convergence of the OSBJ method. In the preprocessing, we first perform two QR decompositions (QRD) with column-pivoting on the input matrix A:

$$AP_1 = Q_1R_1, \tag{1}$$

$$R_1^\top P_2 = Q_2R_2, \tag{2}$$

where P_1 and P_2 are permutation matrices. Then, we let $B = R_2^\top \in \mathbb{R}^{n\times n}$. This is the preprocessed matrix. We compute its SVD, $B = \bar{U}\Sigma\bar{V}^\top$ by OSBJ. Finally, we recover the SVD of A by the following postprocessing:

$$U = Q_1P_2\bar{U}, \tag{3}$$

$$V = P_1Q_2(R_2^{-\top}\bar{U}\Sigma). \tag{4}$$

Figure 1 shows the pseudocode of the OSBJ method with QR pre/postprocessing. Here, U and V are the matrices of the left and right singular vectors, respectively, and S is a diagonal matrix whose diagonal elements are the singular values. "osbj" is the OSBJ method for the preprocessed matrix B to be explained in 2.1.2.

```
1: procedure POSBJ($A$)
2:     $[Q1, R1, P1] = \text{qr}(A)$
3:     $[Q2, R2, P2] = \text{qr}(R1')$
4:     $B = R2'$
5:     $[Ub, S, Vb] = \text{osbj}(B)$
6:     $U = Q1 * P2 * Ub$
7:     $V = P1 * Q2n(R2') * Ub * S$
8:     return $U, S, V$
```

Figure 1. Pseudocode of the OSBJ method with QR pre/postprocessing. We are using MATLAB-like notations. Thus, " ' " denotes the transposition, "*" denotes the matrix product and "\" denotes the solution of a linear system. "qr" is the MATLAB function to compute the QR decomposition with column-pivoting. "osbj" is the OSBJ code defined in Figure 2. Note that "Ub", "S" and "Vb" are used to denote $\bar{U}$, Σ and $\bar{V}$.

Thanks to the column-pivoting in the first QRD, the row-scaled condition number of R_1 is bounded by a constant independent of $\kappa_2(A)$, typically of $O(n)$ [10, Remark 3.2]. Here, the row-scaled condition number $\kappa_{\mathrm{R}}(A)$ and the column-scaled condition number $\kappa_{\mathrm{C}}(A)$ of A are defined as

$$\kappa_{\mathrm{R}}(A) := \kappa_2(D_r^{-1}A) \tag{5}$$

where $D_r = \mathrm{diag}(||A(1,:)||, \|A(2,:)||, \ldots, \|A(n,:)||)$,

$$\kappa_{\mathrm{C}}(A) := \kappa_2(AD_c^{-1}) \tag{6}$$

where $D_c = \mathrm{diag}(||A(:,1)||, \|A(:,2)||, \ldots, \|A(:,n)||)$.

The same holds true also for the second QRD, and thus both $\kappa_{\mathrm{R}}(B)$ and $\kappa_{\mathrm{C}}(B)$ become small. This explains why the QR preprocessing is so successful in reducing the number of sweeps of the one-sided Jacobi method. LAPACK implements some more tricks to improve performance or accuracy for special cases. For a well conditioned matrix, it uses the pivot-less QRD instead of (2) for better performance, and for a badly conditioned matrix, it may add one more QRD. The details are described in [10, Section 5]. We used LAPACK's QR preprocessing code in our numerical experiments.

2.1.2 SVD of the Preprocessed Matrix

Now we will explain the second (central) part, the computation of SVD of B by OSBJ. Let B be partitioned into column blocks as $B = [B_1 B_2 \ldots B_q] \in \mathbb{R}^{n\times n}$, where $B_i \in \mathbb{R}^{n\times n_i}$ and $n_1 + n_2 + \cdots + n_q = n$. The OSBJ method starts from $B^{(0)} = B$ and orthogonalizes a pair of column blocks at each step by post-multiplication by an orthogonal matrix. Let the indices of the column blocks chosen at step r be (I_r, J_r). Then, the orthogonalization is performed in the following manner.

1. The matrix $X = [B_{I_r}^{(r)}\, B_{J_r}^{(r)}]$ is formed.
2. The thin SVD of X is computed as $X = U_X \Sigma_X V_X^\top$.
3. $B_{I_r}^{(r)}$ and $B_{J_r}^{(r)}$ is updated as $[B_{I_r}^{(r+1)}\, B_{J_r}^{(r+1)}] = XV_X = U_X\Sigma_X$.
4. $B_{I_r}^{(r)}$ and $B_{J_r}^{(r)}$ are replaced by $B_{I_r}^{(r+1)}$ and $B_{J_r}^{(r+1)}$.

We call step 2. the "partial SVD." By post-multiplying X by V_X obtained in the partial SVD, its column vectors are orthogonalized, since $XV_X = (U_X\Sigma_X V_X^\top)V_X = U_X\Sigma_X$ and $U_X\Sigma_X$ is a column-scaled version of U_X, which has orthonormal columns. Steps 2. and 3. are the most time-consuming parts in the OSBJ algorithm and there are several approaches for performing them; they will be explained in Subsection 2.2 in detail. By choosing the sequence $\{(I_r, J_r)\}_{r=0,1,\ldots}$ properly (see the paragraph below) and repeating this orthogonalization process for $r = 0, 1, \ldots$, $B^{(r)}$ converges to a matrix with orthogonal columns [12].

The overall procedure of the OSBJ method for the preprocessed matrix is shown in Figure 2. Here, lines 4 through 6 correspond to the orthogonalization of the column block pair. After all the columns have been orthogonalized to a specified level, the singular triplet U, S and V are computed in lines 8 through 12. See [13] for details.

```
1: procedure OSBJ(B)
2:     $r = 0$; $I = 1$; $J = 2$; $B0 = B$; $S = O$
3:     while ortho($B$) > tol do
4:         $[I, J] = \text{next\_pivot}(I, J, r)$
5:         $X = [B[I], B[J]]$
6:         $[B[I], B[J]] = V2(X)$
7:     end while
8:     for $j = 1, n$
9:         $S(j, j) = \text{norm}(B(*, j))$
10:        $U(*, j) = B(*, j)/S(j, j)$
11:    end for
12:    $V = B\backslash B0$
13:    return $U, S, V$
```

Figure 2. Pseudocode of the OSBJ method for the preprocessed matrix. Here, "next_pivot" is a function to generate the indices of the column block pair to be orthogonalized at the r^{th} step. "ortho" is a function to compute the measure of orthogonality defined by Equation (8). "$B[I]$" is the I^{th} block column of B (that is, $B_I^{(r)}$). $[A, B]$ denotes the concatenation of two matrices A and B. In the pseudocode, the procedure $V2$ defined in Figure 3 is used for orthogonalization, but procedures $V1$ and $V3$ can be used as well.

Now, we will give some details on the choice of the sequence $\{(I_r, J_r)\}_{r=0,1,\ldots}$ and the stopping criterion.

Ordering of pairs. Many strategies have been proposed for choosing the sequence $\{(I_r, J_r)\}_{r=0,1,\ldots}$. Among them, we use the *row-cyclic ordering*, which belongs to the simplest class called *cyclic ordering*. In the cyclic ordering, we first choose a finite sequence $\{(I_r, J_r)\}_{r=0}^{q(q-1)/2}$ in such a way that every possible pair (I, J), where $1 \le I < J \le q$, appears exactly once in the sequence. This finite sequence is called *sweep*. Then, the iteration using this sweep is repeated until convergence. The pair in the row-cyclic ordering is defined as follows:

$$(I_r, J_r) = \begin{cases} (1, 2), & r = 0, \\ (I_{r-1}, J_{r-1} + 1), & r > 0, J_{r-1} < q, \\ (I_{r-1} + 1, I_{r-1} + 2), & \text{otherwise.} \end{cases} \tag{7}$$

Termination. As shown in the pseudocode in Figure 2, the iteration of the OSBJ method is terminated when the normalized column vectors of $B^{(r)}$ are orthogonal to working accuracy. For the one-sided point Jacobi method, Drmač recommends to use the following stopping criterion to achieve high relative accuracy of the computed singular values.

$$\mathrm{ortho}(B) \equiv \max_{1\le i<j\le n} \frac{|\mathbf{b}_i^{(r)} \cdot \mathbf{b}_j^{(r)}|}{\|\mathbf{b}_i^{(r)}\|_2 \|\mathbf{b}_i^{(r)}\|_2} \le \mathrm{tol}. \tag{8}$$

Here, $\mathbf{b}_i^{(r)}$ is the i^{th} column of $B^{(r)}$ and $\mathrm{tol} = \sqrt{n}\mathbf{u}$ [10, Remark 2.2]. We also adopt this criterion for our OSBJ. The dot products $\mathbf{b}_i^{(r)} \cdot \mathbf{b}_j^{(r)}$ for $1 \le i < j \le n$ are computed at once using a level-3 BLAS routine xSYRK for high performance.

2.1.3 Numerical Properties of Orthogonalization of the Column Block Pair

In concluding this subsection, we make two comments on the numerical properties of orthogonalization of the column block pair (steps 2. and 3. of 2.1.2), which will be useful in the error analysis in Section 3. First, X is a tall-and-skinny matrix whose aspect ratio is $q : 2$. Moreover, its column-scaled condition number $\kappa_{\mathrm{C}}(X)$ is usually small, because $\kappa_{\mathrm{C}}(X) \le \kappa_{\mathrm{C}}(B^{(r)})$ and it is usually observed that $\kappa_{\mathrm{C}}(B^{(r)})$ does not grow much during the computation. In fact, in our numerical experiments, we observe that $\kappa_{\mathrm{C}}(B^{(r)})$ converges to one. This observation is important in the error analysis to be given in the next section. Second, while the post-multiplication by the orthogonal matrix V_X in step 3. seems harmless, it can cause potential difficulties in finite precision arithmetic, as the following analysis by Drmač suggests [12]. Let $\hat{V}_X$ be the computed right singular vector matrix of X and assume that $\delta V_X = \hat{V}_X - V_X$ is small. Furthermore, Let X' be the matrix obtained by normalizing the columns of X and write $X = X'D$, where D is diagonal. Then, we have

$$X\hat{V}_X = XV_X + X\delta V_X = (U + X'\delta F)\Sigma_X, \tag{9}$$

$$\delta F = D\delta V_X \Sigma_X^{-1}. \tag{10}$$

Since δF can be large even if δV_X is small, this means that the normalized columns of the updated column block pair can be far from orthogonal. This will retard the convergence. The above observation suggests that a more intricate error analysis of steps 2. and 3. is necessary and it will be the main subject of this paper.

2.2 Methods for Orthogonalization of the Column Block Pair

As stated in 2.1.2, there are several methods for the partial SVD, or orthogonalization of the column block pair $X = [B_{I_r}^{(r)}\ B_{J_r}^{(r)}] \in \mathbb{R}^{n\times l}$, where $l = n_{I_r} + n_{J_r}$. Here, we review them briefly, discuss their advantages and disadvantages, and explain why we focus on Hari's V2 method in this paper.

The simplest approach is to apply the one-sided point Jacobi method directly to X, but it is inefficient because the whole X matrix must be updated by the Givens rotation, which is a level-1 BLAS-like slow operation. To avoid this, two approaches have been used. The first is to form the Gram matrix $C = X^{\top}X$ and compute its eigendecomposition $C = V_X D V_X^\top$ [22]. The second approach, known as the LHC method [14], is to compute the (thin) QR decomposition $X = QR$, where $Q \in \mathbb{R}^{n\times l}$ and $R \in \mathbb{R}^{l\times l}$, and compute its SVD, $R = U_R \Sigma_X V_X^\top$. The one-sided (point) Jacobi method can be used for this SVD. In either way, the orthogonalized column block pair is computed by $Y = XV_X$ or $Y = QU_R\Sigma_X$.

In the LAPACK implementation of the LHC method, the QR decomposition is computed by the Householder QR method and then the matrix QU_R is formed as the column-normalized version of $Y = XV_X$. This guarantees that the columns of QU_R are highly orthogonal. However, its computational cost is roughly twice that of the Cholesky QR-based methods to be described below. Moreover, since QU_R is not directly computed from X and V_X but from Q and U_R, it is not straightforward to show that the backward error $\|QU_R\Sigma_X - XV_X\|_2$ is small.

As an alternative, the Cholesky QR method can be used to compute the QR decomposition of X. This method forms the Gram matrix $C = X^\top X$, computes its Cholesky decomposition $C = R^\top R$ and finally obtain the orthogonal factor by $Q = XR^{-1}$. While the method is known to be unstable when the condition number of X is large, it requires only half as much computational work as the Householder QR method. Furthermore, it is suited to high performance computing since most of its computations can be done with the level-3 BLAS such as xSYRK and xTRSM.

Hari et al. propose three algorithms for using the Cholesky method in the partial SVD, which they call *V1*, *V2* and *V3* [15]. They all use the one-sided point Jacobi method to compute the SVD of R, $R = U_R\Sigma_X V_X^\top$, but differ in the way of computing the orthogonal matrix V_X. V1 computes V_X as a product of the Givens rotations used in the Jacobi method. In V2, the Givens rotations are not accumulated and V_X is computed as $V_X = R^{-1}U_R\Sigma_X$. In V3, V_X is computed as $V_X = R^\top U_R \Sigma_X$. These three algorithms are shown in Figure 3. From the viewpoint of high performance computing, V2 and V3, which do not require the accumulation of the Givens matrices, are desirable. However, Hari et al. report that OSBJ using V3 does not converge in their numerical experiments. They recommend V1 for accuracy, but also comment that V2 can be faster than V1. Hence it would be worthwhile to analyze the numerical properties of V2. If we can show by the roundoff error analysis that V2 has sufficient accuracy under certain conditions, it can be the method of choice, since it is both fast and accurate. This error analysis is the topic of the next section. We will also compare the V1 and V2 methods experimentally in Section 4.

3 ERROR ANALYSIS

In this section, we perform roundoff error analysis of the Cholesky QR-based partial SVD, focusing on Hari's V2 variant. Our objective is to show that the V2 variant has sufficient accuracy under certain conditions, thereby establishing the competitiveness of the method not only in terms of speed but also in terms of accuracy. To this end, we need to evaluate two kinds of errors. The first error is the *orthogonality*

```
1: procedure V1($X$)
2:     $C = X' * X$
3:     $R = \mathrm{chol}(C)$
4:     $[Ux, Sx, Vx] = \mathrm{jsvd}(R)$
5:     return $X * Vx$
```

```
1: procedure V2($X$)
2:     $C = X' * X$
3:     $R = \mathrm{chol}(C)$
4:     $[Ux, Sx] = \mathrm{jsvd}(R)$
5:     $Vx = R \backslash Ux * Sx$
6:     return $X * Vx$
```

```
1: procedure V3($X$)
2:     $C = X' * X$
3:     $R = \mathrm{chol}(C)$
4:     $[Ux, Sx] = \mathrm{jsvd}(R)$
5:     $Vx = R' * Ux * (Sx.\hat{} -1)$
6:     return $X * Vx$
```

Figure 3. Pseudocodes of Hari's V1, V2 and V3 methods. We are using MATLAB-like notations as in Figure 1. ".^" denotes the element-wise power. "jsvd" is the same as MATLAB's "svd", which computes the thin SVD of the input matrix, except that it uses the Jacobi SVD algorithm. "jsvd" skips the computation of "Vx" if it is not needed.

error. Let $\hat{Y} \in \mathbb{R}^{n\times l}$ be the updated column block pair computed in finite precision arithmetic and assume that $\hat{Y}$ can be written as

$$\hat{Y} = (\bar{U} + \delta U)\hat{\Sigma} \tag{11}$$

where $\bar{U} \in \mathbb{R}^{n\times l}$ is an exactly orthogonal matrix and $\hat{\Sigma}$ is a diagonal matrix. Then we define δU as the orthogonalization error in the partial SVD. The second error is the *backward error*. Assume that the same $\hat{Y}$ can be written as

$$\hat{Y} = (X + \delta X)\bar{V}_X \tag{12}$$

where $\bar{V}_X$ is an exactly orthogonal matrix. This equation shows that $\hat{Y}$ is an exact (one-sided) orthogonal transformation of a perturbed matrix $X + \delta X$. Then we define δX the backward error in the partial SVD. The orthogonalization error is related to the stagnation of the convergence of OSBJ, because large δU means that the columns of $B^{(r)}$ have not been orthogonalized properly after the partial SVD. On the other hand, the backward error is related to the accuracy of the entire SVD, because large δX means that OSBJ is computing the SVD of a largely perturbed input matrix. The plan of this section is as follows. In Subsection 3.1, we derive an upper bound on the orthogonality error. The bound on the backward error will be provided in Subsection 3.2. Finally, we discuss the criterion for using the V2 method safely in Subsection 3.3.

Throughout this section, we use the following notations [16]. The symbol $fl(\cdot)$ is used to denote the result of floating-point computation. For any matrix A, we denote its computed counterpart by $\hat{A}$. The column scaled version of A is denoted by A'. We denote the (i, j) element of A by $A_{i,j}$ and the matrix whose (i, j) element is $|A_{i,j}|$ by $|A|$. The j^{th} column vector of A is denoted by $\mathbf{a}_j$. Inequalities like $A \le B$ mean element-wise inequality. The unit roundoff is denoted by $\mathbf{u}$ and $\gamma_m \equiv \frac{m\mathbf{u}}{1-m\mathbf{u}}$. In the following, we freely use the inequality like $\gamma_m < 1.01m\mathbf{u} = O(m\mathbf{u})$. We also assume that $n \ge l = n_{I_r} + n_{J_r}$ and $n^2\mathbf{u} \ll 1$.

3.1 Orthogonality Error of V2

The V2 variant computes the partial SVD in the following four steps.

- Compute the QR decomposition $X = QR$ by the Cholesky QR method.
- Apply the one-sided point Jacobi method to R and obtain $U_R\Sigma_X$.
- Compute V_X by $V_X = R^{-1}U_R\Sigma_X$.
- Update the column block pair by matrix multiplication $Y = XV_X$.

In the following, we will analyze the errors in these steps in this order.

3.1.1 Errors in the Cholesky QR Method

Let $d_j = \|\mathbf{x}_j\|_2$ for $1 \le j \le l$ and $D \equiv \operatorname{diag}(d_1, d_2, \ldots, d_l)$. Then, X can be written as $X = X'D$, where X' is the column scaled version of X. In the Cholesky QR method, we first form the Gram matrix $C = X^\top X$ and then compute its Cholesky decomposition, $C = R^\top R$. By denoting the computed version of C and R by $\hat{C}$ and $\hat{R}$, respectively, we have the following lemma.

Lemma 1. Let the forward error in the computation of $\hat{C}$ be E_1 and the backward error in the computation of $\hat{R}$ from $\hat{C}$ be E_2. That is,

$$\hat{C} = C + E_1 = X^\top X + E_1, \tag{13}$$

$$\hat{R}^\top \hat{R} = \hat{C} + E_2 = C + E_1 + E_2. \tag{14}$$

Then, the elements of $|E_1|$ and $|E_2|$ can be bounded as follows.

$$|E_1|_{i,j} \le \gamma_n d_i d_j = O(n\mathbf{u}) d_i d_j, \tag{15}$$

$$|E_2|_{i,j} \le \gamma_{l+1}\sqrt{\hat{C}_{i,i}\hat{C}_{j,j}} \le \gamma_{l+1}(1+\gamma_n) d_i d_j = O(l\mathbf{u}) d_i d_j. \tag{16}$$

Proof. The normwise error bounds on E_1 and E_2 are given in [17]. Here, however, we need component-wise error bounds. From the forward error bound of matrix multiplication, we have $|E_1| \le \gamma_n |X|^\top |X|$. Thus,

$$|E_1|_{i,j} \le \gamma_n \sum_{k=1}^{n} |X|_{k,i}|X|_{k,j} \le \gamma_n \left\||\mathbf{x}_i|\right\|_2 \left\||\mathbf{x}_j|\right\|_2 = \gamma_n d_i d_j. \tag{17}$$

The first inequality in (16) is due to Demmel [18, Lemma 2.1]. The second inequality can be proved by noting $\hat{C}_{i,i} \leq (1+\gamma_n)d_i^2$ from (15) and using

$$\gamma_{l+1}(1+\gamma_n) = O(l\mathbf{u})(1+O(n\mathbf{u})) = O(l\mathbf{u}). \tag{18}$$

$\square$

By letting $\hat{R}' = \hat{R}D^{-1}$, we have $\hat{R}'^{\top}\hat{R}' = X'^{\top}X' + E_3$, where

$$E_3 = D^{-1}(E_1+E_2)D^{-1}, \tag{19}$$

$$|E_3| \leq O(n\mathbf{u}) \ll 1. \tag{20}$$

Noting that E_3 is an $l \times l$ matrix, we also have

$$\|E_3\|_2 \leq \|\,|E_3|\,\|_F \leq O(nl\mathbf{u}) \ll 1. \tag{21}$$

In the following, we make the following important assumption on $\kappa_2(X')$:

$$O(nl\mathbf{u})\kappa_2^2(X') \ll 1. \tag{22}$$

We can justify this assumption because in the OSBJ method with QR preprocessing, the column-scaled condition number of $B^{(r)}$, and therefore of X, is usually small and approaches to 1 as the iteration proceeds. See Subsection 3.3 for the treatment of the case when (22) is not satisfied.

Let us denote the smallest and the largest eigenvalues of $X'^{\top}X'$ by $\lambda_{\min}(X'^{\top}X')$ and $\lambda_{\max}(X'^{\top}X')$, respectively. Since all the column vectors of X' has the unit length, $\lambda_{\max}(X'^{\top}X') \geq 1$. Thus, from (22), we have

$$\lambda_{\min}(X'^{\top}X') \gg \lambda_{\max}(X'^{\top}X')O(nl\mathbf{u}) \geq O(nl\mathbf{u}). \tag{23}$$

On the other hand, since $\hat{R}'^{\top}\hat{R}' = X'^{\top}X' + E_3$, we have from (21) and Weyl's theorem,

$$\lambda_{\min}(\hat{R}'^{\top}\hat{R}') \geq \lambda_{\min}(X'^{\top}X') - \|E_3\|_2 \geq \lambda_{\min}(X'^{\top}X') - O(nl\mathbf{u}) \geq O(nl\mathbf{u}). \tag{24}$$

This can be rewritten as

$$\left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \ll 1. \tag{25}$$

Now, we evaluate how close the computed upper triangular factor $\hat{R}$ is to the true upper triangular factor R. In particular, we express $\hat{R}^{-1}$ in terms of R^{-1} for later use. The following lemma holds.

Lemma 2. Under the assumption (22), there exist an orthogonal matrix W_1 and an error matrix E_4 that satisfy

$$\hat{R}^{-1} = R^{-1}(W_1+E_4), \tag{26}$$

$$\|E_4\|_2 \leq \left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}). \tag{27}$$

Proof. First, consider the following product:

$$\begin{aligned}\left(R\hat{R}^{-1}\right)^\top\left(R\hat{R}^{-1}\right) &= \hat{R}^{-\top}C\hat{R}^{-1}\\ &= \hat{R}^{-\top}(\hat{R}^\top\hat{R} - DE_3D)\hat{R}^{-1}\\ &= I - \hat{R}'^{-\top}E_3\hat{R}'^{-1} \equiv I + E_5. \end{aligned}\tag{28}$$

Since E_5 is a symmetric matrix, we consider its EVD, $E_5 = W_2\Gamma_1W_2^\top$. Then,

$$\|E_5\|_2 = \|\Gamma_1\|_2 \le \left\|\hat{R}'^{-1}\right\|_2^2\|E_3\|_2 \le \left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \ll 1 \tag{29}$$

where we used (21) and (25) in the second and the third inequalities, respectively. Using the same EVD, we rewrite the rightmost-hand side of (28) as

$$I + E_5 = W_2(I+\Gamma_1)W_2^\top = \left((I+\Gamma_1)^{\frac{1}{2}}W_2^\top\right)^\top\left((I+\Gamma_1)^{\frac{1}{2}}W_2^\top\right). \tag{30}$$

Then, since

$$\left[\left(R\hat{R}^{-1}\right)\left((I+\Gamma_1)^{\frac{1}{2}}W_2^\top\right)^{-1}\right]^\top\left[\left(R\hat{R}^{-1}\right)\left((I+\Gamma_1)^{\frac{1}{2}}W_2^\top\right)^{-1}\right] = I \tag{31}$$

from (28), there exists an orthogonal matrix W_3 such that

$$R\hat{R}^{-1} = W_3(I+\Gamma_1)^{\frac{1}{2}}W_2^\top. \tag{32}$$

Hence,

$$\hat{R}^{-1} = R^{-1}\left(W_3W_2^\top + W_3\left((I+\Gamma_1)^{\frac{1}{2}} - I\right)W_2^\top\right) = R^{-1}(W_1 + E_4) \tag{33}$$

where

$$W_1 = W_3W_2^\top, \quad E_4 = W_3\left((I+\Gamma_1)^{\frac{1}{2}} - I\right)W_2^\top. \tag{34}$$

Since Γ_1 is a diagonal matrix, E_4 can be bounded using the inequality $(1+x)^{\frac{1}{2}} \le 1 + \frac{x}{2}$, which holds when $|x| \le 1$, as

$$\|E_4\|_2 = \|\,(I+\Gamma_1)^{\frac{1}{2}} - I\|_2 \le \frac{1}{2}\|\Gamma_1\|_2 \le \frac{1}{2}\left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \tag{35}$$

where we used (29) in the last inequality. □

Now we evaluate the condition number of $\hat{R}'$. From $\hat{R}'^\top\hat{R}' = R'^\top R' + E_3$, we have

$$\left\|\hat{R}'\right\|_2^2 \le \|R'\|_2^2 + \|E_3\|_2 \le (1 + \|E_3\|_2)\|R'\|_2^2 = O(1)\|R'\|_2^2 = O(1)\|X'\|_2 \tag{36}$$

where we used $\|R'\|_2^2 = \lambda_{\max}(X'^{\top}X') \geq 1$ in the second inequality. On the other hand, we have from $\hat{R}'^{\top}\hat{R}' = X'^{\top}X' + E_3$, (23) and (21),

$$\left\|\hat{R}'^{-1}\right\|_2 \leq O(1)\left\|X'^{-1}\right\|_2. \tag{37}$$

Combining these leads to the following lemma.

Lemma 3. *Under the assumption of* (22),

$$\kappa_2(\hat{R}') = O(1)\kappa_2(R') = O(1)\kappa_2(X'). \tag{38}$$

3.1.2 Errors in the One-Sided Point Jacobi Method

Assume that the one-sided point Jacobi method on R ended successfully and the matrix $\hat{T} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_l]$ is obtained. $\hat{T}$ is an approximation to $U_R\Sigma_X$, where U_R is the left singular vector matrix of R and Σ_X is a diagonal matrix whose diagonal elements are the singular values of R (and therefore of X). We write $\hat{T}$ as $\hat{T} = \hat{U}\hat{\Sigma}$, where

$$\hat{U} = [\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_l] = \left[\frac{\hat{\mathbf{t}}_1}{\|\hat{\mathbf{t}}_1\|}, \frac{\hat{\mathbf{t}}_2}{\|\hat{\mathbf{t}}_2\|}, \ldots, \frac{\hat{\mathbf{t}}_l}{\|\hat{\mathbf{t}}_l\|}\right], \tag{39}$$

$$\hat{\Sigma} = \mathrm{diag}(\hat{\sigma}_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_l) = \mathrm{diag}(\|\hat{\mathbf{t}}_1\|, \|\hat{\mathbf{t}}_2\|, \ldots, \|\hat{\mathbf{t}}_l\|). \tag{40}$$

From (8), the stopping criterion in floating point arithmetic can be written as follows.

$$\left|fl(\hat{\mathbf{t}}_i^{\top}\hat{\mathbf{t}}_j)\right| \leq fl\left(\mathrm{tol}\sqrt{\hat{\mathbf{t}}_i^{\top}\hat{\mathbf{t}}_i}\sqrt{\hat{\mathbf{t}}_j^{\top}\hat{\mathbf{t}}_j}\right) \quad \text{for } 1 \leq i < j \leq l, \tag{41}$$

where we use $\mathrm{tol} = \sqrt{l}\mathbf{u}$ as noted in Subsection 2.1.

Lemma 4. *When the stopping criterion* (41) *is satisfied, the following inequality holds for* $1 \leq i < j \leq l$.

$$\left|\hat{\mathbf{u}}_i^{\top}\hat{\mathbf{u}}_j\right| \leq O(l\mathbf{u}). \tag{42}$$

Proof. We first bound the right-hand side of (41) from above as follows.

$$\begin{aligned} fl\left(\mathrm{tol}\sqrt{\hat{\mathbf{t}}_i^{\top}\hat{\mathbf{t}}_i}\sqrt{\hat{\mathbf{t}}_j^{\top}\hat{\mathbf{t}}_j}\right) &\leq (1+\mathbf{u})^4\,\mathrm{tol}\sqrt{fl(\hat{\mathbf{t}}_i^{\top}\hat{\mathbf{t}}_i)}\sqrt{fl(\hat{\mathbf{t}}_j^{\top}\hat{\mathbf{t}}_j)} \\ &\leq (1+\mathbf{u})^4(1+\gamma_l)^2\,\mathrm{tol}\left\|\hat{\mathbf{t}}_i\right\|_2\left\|\hat{\mathbf{t}}_j\right\|_2 \\ &\leq O(1)\mathrm{tol}\left\|\hat{\mathbf{t}}_i\right\|_2\left\|\hat{\mathbf{t}}_j\right\|_2. \end{aligned} \tag{43}$$

Here, the factor $(1+\mathbf{u})^4$ comes from the errors arising in the two square roots, their product, and the product by tol. In the second inequality, we used $fl(\hat{\mathbf{t}}_i^{\top}\hat{\mathbf{t}}_i) \leq$

$\left\|\hat{\mathbf{t}}_i\right\|_2^2 (1+\gamma_l)$. Next, we evaluate the left-hand side of (41) from below. From the error analysis of an inner product [16], we have

$$fl(\hat{\mathbf{t}}_i^\top \hat{\mathbf{t}}_j) = \hat{\mathbf{t}}_i^\top \hat{\mathbf{t}}_j + e, \tag{44}$$

$$|e| \le \gamma_l \left|\hat{\mathbf{t}}_i\right|^\top \left|\hat{\mathbf{t}}_j\right| \le \gamma_l \left\|\hat{\mathbf{t}}_i\right\|_2 \left\|\hat{\mathbf{t}}_j\right\|_2. \tag{45}$$

Hence,

$$\left|fl(\hat{\mathbf{t}}_i^\top \hat{\mathbf{t}}_j)\right| \ge \left|\hat{\mathbf{t}}_i^\top \hat{\mathbf{t}}_j\right| - \gamma_l \left\|\hat{\mathbf{t}}_i\right\|_2 \left\|\hat{\mathbf{t}}_j\right\|_2. \tag{46}$$

Combining (41), (43) and (43) gives

$$\left|\hat{\mathbf{t}}_i^\top \hat{\mathbf{t}}_j\right| \le (O(1)\mathrm{tol} + \gamma_l) \left\|\hat{\mathbf{t}}_i\right\|_2 \left\|\hat{\mathbf{t}}_j\right\|_2 = O(l\mathbf{u}) \left\|\hat{\mathbf{t}}_i\right\|_2 \left\|\hat{\mathbf{t}}_j\right\|_2. \tag{47}$$

Dividing both sides by $\left\|\hat{\mathbf{t}}_i\right\|_2 \left\|\hat{\mathbf{t}}_j\right\|_2$, we obtain (42). □

As for the orthogonality of the computed matrix $\hat{U}$, we have the following lemma.

Lemma 5. The matrix $\hat{U}$ can be written as

$$\hat{U} = \bar{U} + \delta\hat{U} \tag{48}$$

where $\bar{U}$ is an exactly orthogonal matrix and $\delta\hat{U}$ is an error matrix satisfying

$$\left\|\delta\hat{U}\right\|_2 \le O(l^2\mathbf{u}) \ll 1. \tag{49}$$

Proof. Since $\left|\hat{U}^\top \hat{U} - I\right| \le O(l\mathbf{u})$ from Lemma 4, we have

$$\left\|\hat{U}^\top \hat{U} - I\right\|_2 \le \left\|\left|\hat{U}^\top \hat{U} - I\right|\right\|_\mathrm{F} \le O(l^2\mathbf{u}). \tag{50}$$

Thus, by writing the EVD of $\hat{U}^\top \hat{U}$ as $\hat{U}^\top \hat{U} = Z(I+\Gamma_3)Z^\top$, we have $\|\Gamma_3\| \le O(l^2\mathbf{u}) \ll 1$. Now, let $(I+\Gamma_3)^{\frac{1}{2}} = I + \Gamma_4$, where Γ_4 is a diagonal matrix. Then, $\|\Gamma_4\|_2 \le \|\,(I+\Gamma_3)^{\frac{1}{2}}\,\| - 1 \le \frac{1}{2}\|\Gamma_3\|_2 = O(l^2\mathbf{u})$. Moreover, since

$$\left(\hat{U}\left((I+\Gamma_4)Z^\top\right)^{-1}\right)^\top \left(\hat{U}\left((I+\Gamma_4)Z^\top\right)^{-1}\right) = I, \tag{51}$$

there exists an orthogonal matrix W_4 such that

$$\hat{U} = W_4(I+\Gamma_4)Z^\top. \tag{52}$$

By letting $\bar{U} = W_4 Z^\top$ and $\delta\hat{U} = W_4\Gamma_4 Z^\top$, we have (48). The norm of $\delta\hat{U}$ can be bounded as $\|\delta\hat{U}\|_2 = \|\Gamma_4\|_2 = O(l^2\mathbf{u})$. □

3.1.3 Errors in the Computation of V_X

In the next step, we compute V_X by $V_X = R^{-1}U\Sigma$. In floating point arithmetic, we compute $\hat{V}_X = fl(\hat{R}^{-1}\hat{T})$ from the result $\hat{T}$ of the one-sided point Jacobi method by solving the triangular system with multiple right-hand sides. Now, let us denote the i^{th} column vector of $\hat{V}_X$ by $\hat{\mathbf{v}}_i$ and the backward error in the solution of the i^{th} triangular system by $\delta\hat{R}_i$. Then,

$$\begin{aligned}\hat{\mathbf{v}}_i &= fl(\hat{R}^{-1}\hat{\mathbf{t}}_i) = (\hat{R} + \delta\hat{R}_i)^{-1}\hat{\mathbf{t}}_i \\ &= \left(\left(I + \delta\hat{R}_i\hat{R}^{-1}\right)\hat{R}\right)^{-1}\hat{\mathbf{t}}_i \\ &= \hat{R}^{-1}\left(I + \delta\hat{R}_i\hat{R}^{-1}\right)^{-1}\hat{\mathbf{t}}_i. \end{aligned} \tag{53}$$

We further define $\hat{R}'$ and $\delta\hat{R}'_i$ as $\hat{R}' = \hat{R}D^{-1}$ and $\delta\hat{R}'_i = \delta\hat{R}_iD^{-1}$ using the diagonal matrix D defined in 3.1.1. Then, the following lemma holds.

Lemma 6. Assume that (22) holds and define an error matrix F_i by

$$I + F_i = \left(I + \delta\hat{R}_i\hat{R}^{-1}\right)^{-1}. \tag{54}$$

Then,

$$\|F_i\|_2 \leq O(l^{\frac{3}{2}}\mathbf{u}\kappa_2(\hat{R}')). \tag{55}$$

Proof. The backward error $\delta\hat{R}_i$ in the solution of the triangular system satisfies $\left|\delta\hat{R}_i\right| \leq \gamma_l\left|\hat{R}\right|$ [16]. Multiplying both sides by a nonnegative diagonal matrix D^{-1} gives

$$\left|\delta\hat{R}'_i\right| \leq \gamma_l\left|\hat{R}'\right|. \tag{56}$$

Hence,

$$\left\|\delta\hat{R}'_i\right\|_2 \leq \left\|\left|\delta\hat{R}'_i\right|\right\|_{\mathrm{F}} \leq \gamma_l\left\|\left|\hat{R}'\right|\right\|_{\mathrm{F}} \leq O\left(l^{\frac{3}{2}}\mathbf{u}\right)\left\|\hat{R}'\right\|_2. \tag{57}$$

Thus, we have

$$\begin{aligned}\left\|\delta\hat{R}_i\hat{R}^{-1}\right\|_2 &= \left\|\delta\hat{R}'_i\hat{R}'^{-1}\right\|_2 \\ &\leq \left\|\delta\hat{R}'_i\right\|_2\left\|\hat{R}'^{-1}\right\|_2 \\ &\leq O\left(l^{\frac{3}{2}}\mathbf{u}\right)\left\|\hat{R}'\right\|_2\left\|\hat{R}'^{-1}\right\|_2 = O\left(l^{\frac{3}{2}}\mathbf{u}\right)\kappa_2(\hat{R}') \ll 1\end{aligned} \tag{58}$$

where we used $O\left(l^{\frac{3}{2}}\mathbf{u}\right)\kappa_2(\hat{R}') \leq O\left(nl\mathbf{u}\right)\kappa_2(\hat{R}') = O\left(nl\mathbf{u}\right)\kappa_2(X') \ll 1$, which is a consequence of (22) and Lemma 3. As a result, the Neumann series expansion of

$I + F_i = \left(I + \delta\hat{R}_i\hat{R}^{-1}\right)^{-1}$ converges and we have

$$I + F_i = \sum_{k=0}^{\infty} \left(-\delta\hat{R}_i\hat{R}^{-1}\right)^k, \tag{59}$$

from which

$$\|F_i\|_2 = \frac{\left\|\delta\hat{R}_i\hat{R}^{-1}\right\|_2}{1 - \left\|\delta\hat{R}_i\hat{R}^{-1}\right\|_2} \le O\left(l^{\frac{3}{2}}\mathbf{u}\right)\kappa_2(\hat{R}') \tag{60}$$

follows immediately. □

Now we rewrite $\hat{\mathbf{v}}_i$ using $\hat{\mathbf{t}}_i = \hat{\sigma}_i\hat{\mathbf{u}}_i$ as

$$\hat{\mathbf{v}}_i = \hat{R}^{-1}(I + F_i)\hat{\mathbf{t}}_i = \hat{R}^{-1}(\hat{\mathbf{u}}_i + F_i\hat{\mathbf{u}}_i)\hat{\sigma}_i = \hat{R}^{-1}(\hat{\mathbf{u}}_i + \delta\hat{\mathbf{u}}_i)\hat{\sigma}_i \tag{61}$$

where we defined $\delta\hat{\mathbf{u}}_i = F_i\hat{\mathbf{u}}_i$. Letting $\delta\hat{U} = [\delta\hat{\mathbf{u}}_1\delta\hat{\mathbf{u}}_2 \ldots \delta\hat{\mathbf{u}}_l]$, we have

$$\begin{aligned}\hat{V}_X &= \hat{R}^{-1}(\hat{U} + \delta\hat{U})\hat{\Sigma} = \hat{R}^{-1}(I + \delta\hat{U}\hat{U}^{-1})\hat{U}\hat{\Sigma} \\ &= \hat{R}^{-1}(I + E_6)\hat{U}\hat{\Sigma}\end{aligned} \tag{62}$$

where $E_6 = \delta\hat{U}\hat{U}^{-1}$. The following lemma gives a bound on $\|E_6\|_2$.

Lemma 7. Under the assumption (22), the following inequality holds.

$$\|E_6\|_2 \le \left\|\delta\hat{U}\right\|_2 \left\|\hat{U}^{-1}\right\|_2 \le O(l^2\mathbf{u})\kappa_2(\hat{R}'). \tag{63}$$

Proof. From (52), the singular values of $\hat{U}$ are equal to those of $I + \Gamma_4$ and are therefore larger than or equal to $1 - \|\Gamma_4\| = 1 - O(l^2\mathbf{u})$. Thus,

$$\left\|\hat{U}^{-1}\right\|_2 \le 1 + O(l^2\mathbf{u}). \tag{64}$$

On the other hand, since $\|\delta\hat{\mathbf{u}}_i\|_2 \le \|F_i\|_2 \|\hat{\mathbf{u}}_i\|_2 \le O(l^{\frac{3}{2}}\mathbf{u})\kappa_2(\hat{R}') \cdot O(1)$,

$$\left\|\delta\hat{U}\right\|_2 \le \left\|\delta\hat{U}\right\|_{\mathrm{F}} \le \sqrt{\sum_{i=1}^{l} \|\delta\hat{\mathbf{u}}_i\|_2^2} \le O(l^2\mathbf{u})\kappa_2(\hat{R}'). \tag{65}$$

Multiplying these two bounds gives $\|E_6\|_2 \le O(l^2\mathbf{u})\kappa_2(\hat{R}')$. □

3.1.4 Errors in the Product $Y = XV_X$

Finally, we evaluate the errors in $\hat{Y} = fl(X\hat{V}_X)$. From the error analysis of matrix multiplication [16], we can write $\hat{Y}$ as

$$\hat{Y} = X\hat{V}_X + E_{MM}, \tag{66}$$

$$|E_{MM}| \leq \gamma_l |X| \left|\hat{V}_X\right|. \tag{67}$$

We first evaluate the error contained in $X\hat{V}_X$ itself and then the matrix multiplication error E_{MM}. From (62) and (26), $\hat{V}_X$ can be written as

$$\begin{aligned} X\hat{V}_X &= X\hat{R}^{-1}(I + E_6)\hat{U}\hat{\Sigma} \\ &= XR^{-1}(W_1 + E_4)(I + E_6)\hat{U}\hat{\Sigma}. \end{aligned} \tag{68}$$

By inserting $X = QR$ and (48) into the last expression leads to

$$\begin{aligned} X\hat{V}_X &= Q(W_1 + E_4)(I + E_6)\hat{U}\hat{\Sigma} \\ &= Q(W_1 + E_4 + W_1E_6 + E_4E_6)(\bar{U} + \delta\hat{U})\hat{\Sigma} \qquad (69) \\ &= \left(QW_1\bar{U} + E_7\right)\hat{\Sigma}. \qquad (70) \end{aligned}$$

Here, $E_7 = Q(E_4 + W_1E_6 + E_4E_6)(\bar{U} + \delta\hat{U}) + QW_1\delta\hat{U}$ and its norm is bounded as

$$\begin{aligned} \|E_7\|_2 &\leq \|E_4 + W_1E_6 + E_4E_6\|_2 \left\|\bar{U} + \delta\hat{U}\right\|_2 + \left\|\delta\hat{U}\right\|_2 \\ &\leq O(l^2\mathbf{u})\kappa_2(\hat{R}') + O(nl\mathbf{u}) \left\|\hat{R}'^{-1}\right\|_2^2 + O(l^2\mathbf{u}) \\ &\leq O(nl\mathbf{u})\kappa_2^2\left(\hat{R}'\right) \end{aligned} \tag{71}$$

where we used $\left\|\hat{R}'^{-1}\right\|_2^2 \leq \left\|\hat{R}'\right\|_2^2 \left\|\hat{R}'^{-1}\right\|_2^2 = \kappa_2^2\left(\hat{R}'\right)$, by noting that the column vectors of R' have the unit length.

To evaluate the matrix multiplication error E_{MM}, we use the relation:

$$|X| \left|\hat{V}_X\right| = |X'| DD^{-1} \left|R'^{-1}(W_1 + E_4)(I + E_6)\hat{U}\right| \hat{\Sigma}. \tag{72}$$

By inserting this into (67), we have

$$\begin{aligned} \left\|E_{MM}\hat{\Sigma}^{-1}\right\|_2 &\leq O(l^2\mathbf{u}) \|X'\|_2 \left\|R'^{-1}\right\|_2 \|W_1 + E_4\|_2 \|I + E_6\|_2 \left\|\hat{U}\right\|_2 \\ &\leq O(l^2\mathbf{u})\kappa_2(X'). \end{aligned} \tag{73}$$

Finally, we put (71) and (73) into (66) and replace $\kappa_2(R')$ with $\kappa_2(X')$ using Lemma 3. Then we arrive at the following theorem that bounds the deviation from orthogonality of the matrix $\hat{Y}$ obtained by the partial SVD.

Theorem 1. Assume that $X \in \mathrm{R}^{n\times l}$ is a full rank matrix with $n \geq l$ and the condition number of its column-scaled version X' satisfies $O(nl\mathbf{u})\kappa_2^2(X') \ll 1$. Assume further that Hari's V2 variant for the partial SVD has been applied to X successfully and the matrix $\hat{Y}$ is obtained. Then, there exist a matrix $\bar{U}$ with orthogonal columns, a diagonal matrix $\hat{\Sigma}$ and an error matrix δU such that

$$\hat{Y} = (\bar{U} + \delta U)\hat{\Sigma}, \tag{74}$$

$$\|\delta U\|_2 \leq O(nl\mathbf{u})\kappa_2^2(X'). \tag{75}$$

Since the column-scaled condition number of X approaches to 1 quickly in OSBJ with QR preprocessing, this result is highly satisfactory.

3.2 Backward Error of V2

We also need to evaluate how close to orthogonal the transformation matrix V_X is, because non-orthogonality of V_X causes deviation of the singular values of $Y = XV_X$ from those of X. To this end, the next theorem by Drmač can be used directly.

Theorem 2 (Drmač [10], Equations (5.3), (5.7), (5.8))**.** Assume that the one-sided point Jacobi method is applied to an upper triangular matrix $\hat{R}$ and the matrix $\hat{T}$, which is an approximation to the product of the left singular vector matrix of $\hat{R}$ and the diagonal matrix containing the singular values of $\hat{R}$, is obtained. Assume further that $\hat{V}_X$ is computed as $\hat{V}_X = fl(\hat{R}^{-1}\hat{T})$. Then, there exist an orthogonal matrix $\bar{V}_X$ and an error matrix $\delta\hat{V}_X$ such that

$$\bar{V}_X = \hat{V}_X + \delta\hat{V}_X, \tag{76}$$

$$\left\|\delta\hat{V}_X\right\|_2 \leq \kappa_{\mathrm{R}}(\hat{R}) \cdot O(sl^2\mathbf{u}) \tag{77}$$

where s is the number of iteration of the one-sided point Jacobi method until convergence and $\kappa_{\mathrm{R}}(\hat{R})$ is the row-scaled condition number of $\hat{R}$.

Using this result, we can evaluate the row-wise backward error of V2. Let the j^{th} row vectors of X, $\hat{Y}$ and E_{MM} be $\tilde{\mathbf{x}}_j$, $\tilde{\mathbf{y}}_j$ and $\tilde{\mathbf{e}}_j$, respectively. Note that

$$|\tilde{\mathbf{e}}_j| \leq \gamma_l\, |\tilde{\mathbf{x}}_j| \left|\hat{V}_X\right| \tag{78}$$

from (67). Then, we have from (66) and (76),

$$\begin{aligned}\tilde{\mathbf{y}}_j &= \tilde{\mathbf{x}}_j V_X + \tilde{\mathbf{e}}_j \\ &= \left(\tilde{\mathbf{x}}_j + \left(-\tilde{\mathbf{x}}_j \delta\hat{V}_X + \tilde{\mathbf{e}}_j\right)\bar{V}_X^\top\right)\bar{V}_X \\ &= (\tilde{\mathbf{x}}_j + \delta\tilde{\mathbf{x}}_j)\bar{V}_X \qquad (79)\end{aligned}$$

where $\delta\tilde{\mathbf{x}}_j = \left(-\tilde{\mathbf{x}}_j \delta\hat{V}_X + \tilde{\mathbf{e}}_j\right)\bar{V}_X^\top$ and

$$\begin{aligned}\|\delta\tilde{\mathbf{x}}_j\|_2 &\le \|\tilde{\mathbf{x}}_j\|_2 \left\|\delta\hat{V}_X\right\|_2 + \gamma_l \|\tilde{\mathbf{x}}_j\|_2 \left\|\hat{V}_X\right\|_{\mathrm{F}} \\ &\le \|\tilde{\mathbf{x}}_j\|_2 \left(\kappa_{\mathrm{R}}(\hat{R}) \cdot O(sl^2\mathbf{u}) + O(l^{\frac{3}{2}}\mathbf{u})\left\|\hat{V}_X\right\|_2\right) \\ &= \|\tilde{\mathbf{x}}_j\|_2 \, \kappa_{\mathrm{R}}(\hat{R}) \cdot O(sl^2\mathbf{u}). \qquad (80)\end{aligned}$$

Thus, we can conclude that the upper bound on the row-wise backward error $\delta\tilde{\mathbf{x}}_j$ is proportional to $\kappa_{\mathrm{R}}(\hat{R})$.

3.3 Criterion for Using the Variant V2

Drmač shows that when $\kappa_2(\hat{R}') = \kappa_{\mathrm{C}}(\hat{R})$ is very close to 1, $\kappa_{\mathrm{R}}(\hat{R})$ also becomes small as well [10, Proposition 3.1]. Thus, we can expect that as the iteration of OSBJ proceeds, $\kappa_{\mathrm{R}}(\hat{R})$ will get smaller. In fact, in the numerical experiments to be presented in the next section, we observed that $\kappa_{\mathrm{R}}(\hat{R})$ does not become much larger than $\kappa_{\mathrm{C}}(\hat{R})$, but frequently becomes smaller than the latter.

However, at intermediate steps, there is no theoretical guarantee that $\kappa_{\mathrm{R}}(\hat{R})$ is sufficiently small. Hence, in our implementation, we chose to switch from V2 to V1 when $\kappa_{\mathrm{R}}(\hat{R})$ is large, because $\hat{V}_X$ computed by V1 is guaranteed to be always nearly orthogonal. To estimate $\kappa_{\mathrm{R}}(\hat{R})$, we use LAPACK's xTRCON, which is an efficient condition number estimator in 1-norm or infinity norm. Specifically, we compute the row-scaled version $\hat{R}''$ of $\hat{R}$ and use the relation:

$$\kappa_{\mathrm{R}}(\hat{R}) = \kappa_2(\hat{R}'') \approx \kappa_1(\hat{R}''), \qquad (81)$$

which holds approximately when l is not too large. The criterion for using V2 is

$$\kappa_1(\hat{R}'') \le \sqrt{l} \qquad (82)$$

and V1 is used instead if this is not satisfied. In the numerical experiments to be given in the next section, this switching did not occur frequently. Thus we can say that the V2 variant, which is superior in terms of speed, can be used safely in place of the V1 variant most of the time.

4 NUMERICAL RESULTS

In this section, we experimentally evaluate the error of Hari's V2 method to support our theoretical analysis and compare them with those of Hari's V1 method. We used variety of test matrices which differ in the matrix size $m = n$, the number of blocks q, the 2-norm condition number $\kappa_2(A)$, and the distribution of the singular values. We generated five different matrices for each combination of the parameters listed below using LAPACK's DLATMS:

- $m = n = 200, 400, 800, 1\,600$
- $q = 10, 20, 40$
- $\kappa = 10^5, 10^{10}, 10^{15}$
- the distribution of singular values from DLATMS described in [19]
  - mode = 1: $\sigma_1 = 1$, $\sigma_2 = \sigma_3 = \cdots = \sigma_n = 1/\kappa$
  - mode = 2: $\sigma_1 = \sigma_2 = \cdots = \sigma_{n-1} = 1$, $\sigma_n = 1/\kappa$
  - mode = 3: $\sigma_i = \kappa^{-(i-1)/(n-1)}$
  - mode = 4: $\sigma_i = 1 - \frac{(i-1)(1-1/\kappa)}{n-1}$
  - mode = 5: the singular values are random numbers in the range $(1/\kappa, 1)$ such that their logarithms are uniformly distributed.

These matrices have singular value distributions that are often seen in real-world problems, such as singular values with high multiplicity and highly clustered singular values at the lower end of the spectrum. To organize the results in small spaces, we indexed the matrices using the formula:

$$\texttt{index} = 9\log_2(n/200) + 3\log_2(q/10) + \log_{10}(\kappa)/5 - 1. \tag{83}$$

We used double-precision floating-point numbers throughout the experiments, thus, the unit of round-off $\mathbf{u} \approx 1.01 \times 10^{-16}$.

4.1 Condition Numbers Observed During the Computation

Table 1 shows the maximum values of the estimated condition numbers, $\kappa_1(\hat{R}')$ and $\kappa_1(D_\mathrm{r}\hat{R})$, which are observed in the tests. We used the LAPACK's DTRCON to estimate the 1-norm condition numbers, thus, they are not exactly same as those used in the analysis, $\kappa(\hat{R}')$ and $\kappa_\mathrm{R}(\hat{R})$, but they provide a good estimate of the true values with small computation cost.

The condition numbers in the tables are drastically small (< 100) compared with those of the input matrices, which can be as large as 10^{15}, even in the first sweep, thanks to the QR preprocessing. It is also notable that the row-scaled condition numbers in the table are smaller than the column-scaled ones. These small figures make our theoretical error bounds (see (75) and (80)) of the order of $\mathbf{u}$. Moreover, because they are small, the switching from V2 to V1 described in Subsection 3.3

	Sweep #	mode = 1	mode = 2	mode = 3	mode = 4	mode = 5
$\kappa_1(\hat{R}')$	1	10.707	1.001	33.112	33.304	33.220
	2	1.233	1.000	1.756	3.763	1.743
	3	1.010	N/A	1.014	1.287	1.024
	4	1.002	N/A	1.000	1.021	1.000
$\kappa_1(D_\mathrm{r}\hat{R})$	1	10.213	1.000	24.093	21.942	18.454
	2	1.152	1.000	1.664	3.435	1.653
	3	1.010	N/A	1.014	1.269	1.024
	4	1.002	N/A	1.000	1.021	1.000

Table 1. The estimated condition numbers with LAPACK's DTRCON. We only listed the maximum values for each mode. N/A means the iteration has already converged.

did not occur for most of the matrices and even when it occurred, it was only a few times. In our tests, no switching occurred for 861 matrices out of 900, only once for 26, and up to five times for the rest. All the swithing, if any, occurred in the first sweep.

4.2 Orthogonality Error of $\hat{Y}$

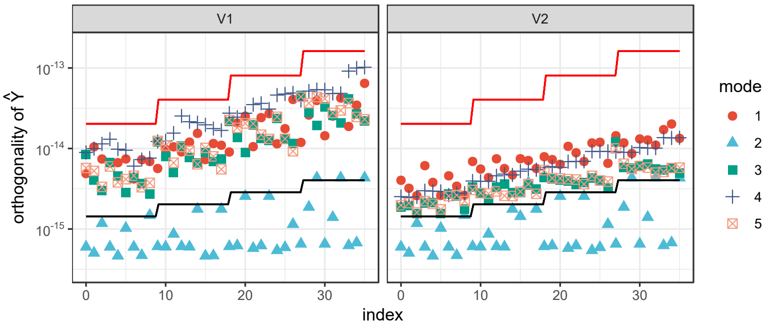


Figure 4. The maximum orthogonality error of $\hat{Y}$ for each parameter combination

Figure 4 shows the orthogonality error of $\hat{Y}$ defined as

$$\left\| (\hat{Y}\hat{\Sigma}^{-1})^\top \hat{Y}\hat{\Sigma}^{-1} - I \right\|_{\max}. \tag{84}$$

This error must be small for the convergence of the overall process. We only plotted the maximum values over all sweeps for each combination of the parameters. For both V1 and V2 methods, the errors are small or close to the $\sqrt{n}\mathbf{u}$ (the black lines in the figures). Generally, V2 has smaller errors than V1 in this test.

4.3 Residual and Orthogonality of the Factors

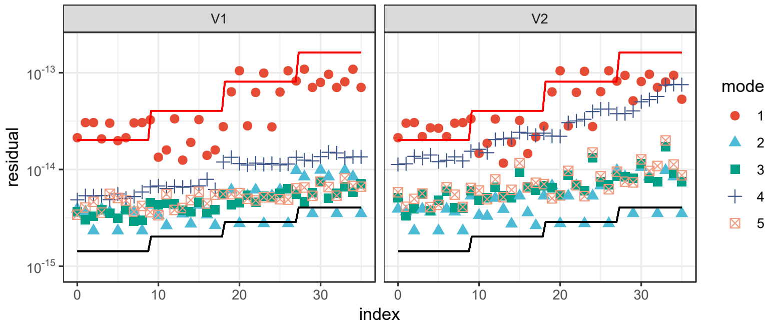


Figure 5. The maximum value of the residual for each parameter combination

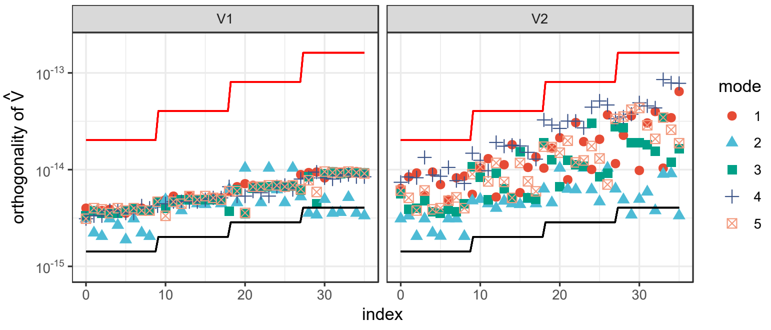


Figure 6. The maximum orthogonality error of $\hat{V}$ for each parameter combination

Figures 5, 6 show the residual of the decomposition, $\left\|A - \hat{U}\hat{\Sigma}\hat{V}\right\|_{\max} / \|A\|_{\max}$, and the orthogonality of the computed $\hat{V}$, $\left\|\hat{V}^{\top}V - I\right\|_{\max}$, respectively. The residuals are small for both V1 and V2, therefore, the Jacobi method can compute the SVD of the test matrices accurately even with the V2 method. The residuals of V2 are a bit larger than those of V1, but we think they are still acceptable because they are around $n\mathbf{u}$ (the red lines in the figures), the unit roundoff times a low degree polynomial of n. The situation is similar for the orthogonality of $\hat{V}$. The errors are small for both V1 and V2. The errors of V2 are a bit larger than those of V1, but they are still acceptable because they are smaller than $n\mathbf{u}$.

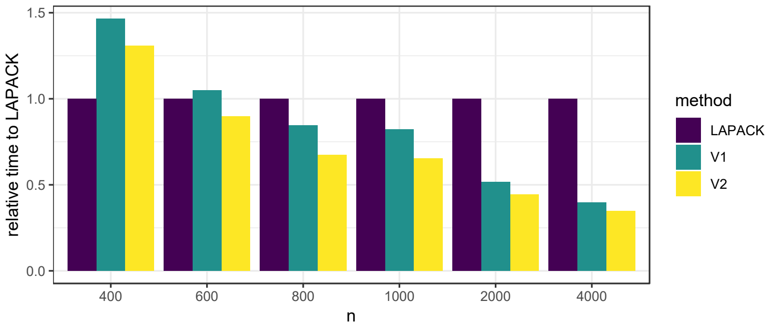


Figure 7. Normalized computation time of the three methods

4.4 Computation Time

Lastly, we compare the computation time of three methods, namely, V1, V2 and LAPACK DGESVJ, which is an implementation of the one-sided point Jacobi SVD method. We generated matrices with parameters $n = 400$ to $4\,000$, $\kappa = 10^{10}$, mode $=$ 3, and $q = 10, 20, 40$ (only for V1 and V2). The computation time was measured five times for each combination of the parameters, their average was calculated, and the value of q which attains the shortest average time was selected for each combination. The experiments were done on a 4-core desktop PC with Intel Core i7-7490 (3.6 GHz, 8 MiB cache) and dual-channel DDR3-1600 memories. Our program was compiled with gcc and gfortran version 7.5.0 and linked with OpenBLAS v3.9.0, with flags `-O3 -mtune=ntive -march=native`.

Figure 7 shows the normalized computation time, where the time of DGESVJ is set to 1. For large matrices, the OBSJ methods, V1 and V2, outperforms DGESVJ, and they achieve more than 2.5 times speedup over DGESVJ. V2 is always faster than V1 in the plot. The difference is larger for small matrices, but it is still more than 10 % when $n = 4\,000$.

5 CONCLUSION

In this paper, we presented a roundoff error analysis of the block orthogonalization process used in the one-sided block Jacobi SVD method. In particular, we focused on the so-called V2 method proposed by Hari and showed that the orthogonality error and the backward error are essentially bounded by the product of the unit roundoff and the column-scaled and row-scaled condition numbers, respectively, of the block to be orthogonalized. Since these condition numbers are usually small and approach one as the iteration proceeds, our results suggest that the V2 method is accurate in terms of both orthogonality and backward error. Numerical experiments confirm this theoretical prediction.

Our future work includes error analysis of the V1 method, which is another block orthogonalization method proposed by Hari, and a study on the impact of our present results on the convergence and accuracy of the one-sided block Jacobi SVD method.

Acknowledgement

We are grateful to Professor Marian Vajteršic for providing us with the opportunity to present part of the results in this paper at a workshop in the PPAM 2017 conference and at the PARNUM 2019 workshop.

REFERENCES

[1] Dongarra, J.—Gates, M.—Haidar, A.—Kurzak, J.—Luszczek, P.—Tomov, S.—Yamazaki, I.: The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale. SIAM Review, Vol. 60, 2018, No. 4, pp. 808–865, doi: 10.1137/17m1117732.

[2] Van Zee, F. G.—van de Geijn, R. A.—Quintana-Ortí, G.: Restructuring the Tridiagonal and Bidiagonal QR Algorithms for Performance. ACM Transactions on Mathematical Software, Vol. 40, 2014, No. 3, Art. No. 18, doi: 10.1145/2535371.

[3] Gates, M.—Tomov, S.—Dongarra, J.: Accelerating the SVD Two Stage Bidiagonal Reduction and Divide and Conquer Using GPUs. Parallel Computing, Vol. 74, 2018, pp. 3–18, doi: 10.1016/j.parco.2017.10.004.

[4] Willems, P. R.—Lang, B.—Vömel, C.: Computing the Bidiagonal SVD Using Multiple Relatively Robust Representations. SIAM Journal on Matrix Analysis and Applications, Vol. 28, 2006, No. 4, pp. 907–926, doi: 10.1137/050628301.

[5] Willems, P. R.—Lang, B.: Twisted Factorizations and qd-Type Transformations for the MR^3 Algorithm – New Representations and Analysis. SIAM Journal on Matrix Analysis and Applications, Vol. 33, 2012, No. 2, pp. 523–553, doi: 10.1137/110834044.

[6] Veselić, K.—Hari, V.: A Note on a One-Sided Jacobi Algorithm. Numerische Mathematik, Vol. 56, 1989, pp. 627–633, doi: 10.1007/bf01396349.

[7] Parlett, B. N.: The Symmetric Eigenvalue Problem. SIAM, Philadelphia, 1998, doi: 10.1137/1.9781611971163.

[8] Demmel, J.—Veselić, K.: Jacobi's Method Is More Accurate than QR. SIAM Journal on Matrix Analysis and Applications, Vol. 13, 1992, No. 4, pp. 1204–1245, doi: 10.1137/0613074.

[9] Demmel, J.—Gu, M.—Eisenstat, S.—Slapničar, I.—Veselić, K.—Drmač, Z.: Computing the Singular Value Decomposition with High Relative Accuracy. Linear Algebra and Its Applications, Vol. 299, 1999, No. 1-3, pp. 21–80, doi: 10.1016/s0024-3795(99)00134-2.

[10] Drmač, Z.—Veselić, K.: New Fast and Accurate Jacobi SVD Algorithm. I. SIAM Journal on Matrix Analysis and Applications, Vol. 29, 2008, No. 4, pp. 1322–1342, doi: 10.1137/050639193.

[11] BRENT, R. P.—LUK, F. T.: The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays. SIAM Journal on Scientific and Statistical Computing, Vol. 6, 1985, No. 1, pp. 69–84, doi: 10.1137/0906007.

[12] DRMAČ, Z.: A Global Convergence Proof for Cyclic Jacobi Methods with Block Rotations. SIAM Journal on Matrix Analysis and Applications, Vol. 31, 2009, No. 3, pp. 1329–1350, doi: 10.1137/090748548.

[13] KUDO, S.—YAMAMOTO, Y.—BEČKA, M.—VAJTERŠIC, M.: Performance Analysis and Optimization of the Parallel One-Sided Block Jacobi SVD Algorithm with Dynamic Ordering and Variable Blocking. Concurrency and Computation: Practice and Experience, Vol. 29, 2017, Art. No. e4059, doi: 10.1002/cpe.4059.

[14] CHAN, T. F.: An Improved Algorithm for Computing the Singular Value Decomposition. ACM Transactions on Mathematical Software, Vol. 8, 1982, No. 1, pp. 72–83, doi: 10.1145/355984.355990.

[15] HARI, V.—SINGER, S.—SINGER, S.: Full Block J-Jacobi Method for Hermitian Matrices. Linear Algebra and its Applications, Vol. 444, 2014, pp. 1–27, doi: 10.1016/j.laa.2013.11.028.

[16] HIGHAM, N. J.: Accuracy and Stability of Numerical Algorithms. 2nd Ed., SIAM, Philadelphia, PA, 2002, doi: 10.1137/1.9780898718027.

[17] YAMAMOTO, Y.—NAKATSUKASA, Y.—YANAGISAWA, Y.—FUKAYA, T.: Roundoff Error Analysis of the CholeskyQR2 Algorithm. Electronic Transactions on Numerical Analysis, Vol. 44, 2015, pp. 306–326.

[18] DEMMEL, J. W.: On Floating Point Errors in Cholesky. LAPACK Working Notes, No. 14, 1989, pp. 1–7.

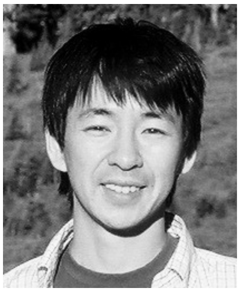
[19] The LAPACK Documentation. Available at: `http://www.netlib.org/lapack/explore-html/index.html`.

[20] LUK, F. T.—PARK, H.: A Proof of Convergence for Two Parallel Jacobi SVD Algorithms. IEEE Transactions on Computers, Vol. 38, 1989, No. 6, pp. 806–811, doi: 10.1109/12.24289.

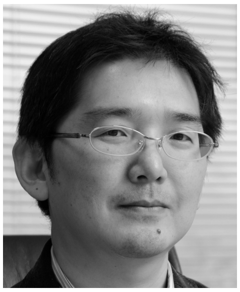
[21] SINGER, S.—SINGER, S.—NOVAKOVIĆ, V.—DAVIDOVIĆ, D.—BOKULIĆ, K.—UŠĆUMLIĆ, A.: Three-Level Parallel J-Jacobi Algorithms for Hermitian Matrices. Applied Mathematics and Computation, Vol. 218, 2012, No. 9, pp. 5704–5725, doi: 10.1016/j.amc.2011.11.067.

[22] BEČKA, M.—OKŠA, G.: New Approach to Local Computations in the Parallel One-Sided Jacobi SVD Algorithm. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9573, 2016, pp. 605–617, doi: 10.1007/978-3-319-32149-3_56.

Shuhei Kudo is Postdoctoral researcher of Large-Scale Parallel Numerical Computing Technology Research Team at RIKEN Center for Computational Science in Japan. He received his Bachelor's degree and Master's degree from The Kobe University in 2013 and 2015, respectively. He received his Ph.D. from the University of Electro-Communications in 2018. His current research interests include high performance computing and parallel numerical linear algebra algoritms in science and engineering.



Yusaku Yamamoto is Professor of high performance computing at the University of Electro-Communications in Japan. He received his Bachelor's degree and Master's degree from the University of Tokyo in 1990 and 1992, respectively. He received his Ph.D. from the Nagoya University in 2003. His current research interests include high performance algorithms in numerical linear algebra and applications of linear algebra in science and engineering.



Toshiyuki Imamura is currently a Team Leader of Large-Scale Parallel Numerical Computing Technology at RIKEN Center for Computational Science. He received his diploma and doctorate in applied systems and sciences in 1993 and 2000, respectively. He was a Researcher at Japan Atomic Energy Research Institute in 1996–2003, a visiting scientist at HLRS in 2001–2002, and Associate Professor at the University of Electro-Communications in 2003–2012. Since 2012 he has been with RIKEN. His research interests include high-performance computing, automatic-tuning technology, parallel eigenvalue computation. He is a member of IPSJ, JSIAM, and SIAM.

Computing and Informatics, Vol. 39, 2020, 1229–1249, doi: 10.31577/cai_2020_6_1229

PROBABILISTIC MEMORY MODEL FOR VISUAL IMAGES CATEGORIZATION

Linxia Xiao, Yanjiang Wang*, Baodi Liu, Weifeng Liu

College of Control Science and Engineering
China University of Petroleum (East China)
Qingdao, 266580, China
e-mail: `b17050163@s.upc.edu.cn, yjwang, liubaodi, liuwf@upc.edu.cn`

Abstract. During the past decades, numerous memory models have been proposed, which focused mainly on how spoken words are studied, whereas models on how visual images are studied are still limited. In this study, we propose a probabilistic memory model (PMM) for visual images categorization which is able to mimic the workings of the human brain during the image storage and retrieval. First, in the learning phase, the visual images are represented by the feature vectors extracted with convolutional neural network (CNN) and each feature component is assumed to conform to a Gaussian distribution and may be incompletely copied with a certain probability or randomly produced in accordance to an exponential distribution. Then, in the test phase, the likelihood ratio between the test image and each studied image is calculated based on the probabilistic inference theory, and an odd value in favor of an old item over a new one is obtained based on all likelihood values. Finally, if the odd value is above a certain threshold, the Bayesian decision rule is applied for image classification. Experimental results on two benchmark image datasets demonstrate that the proposed PMM can perform well on categorization tasks for both studied and non-studied images.

Keywords: Memory model, image categorization, probability distribution, probabilistic inference, Bayesian decision

Mathematics Subject Classification 2010: 68-Txx

* corresponding author

1 INTRODUCTION

As was known to all, computer vision research aims to make the computers easily perceive, understand, and remember the visual images effortlessly as humans do. With the rapid development of cognitive neuroscience and cognitive psychology, numerous new theories and methods have emerged [1], which constantly deepen the study of the complex mechanisms of the human brain and promote the progress of computer vision and artificial intelligence. The fact that the human may retrieve objects from their cluttered surroundings without any difficulties is closely related to the human memory mechanism. Since the time when the first psychology memory model put forward by Atkinson and Shiffrin [2], many memory models have been proposed to date, such as SAM (Search of Associative Memory) [3], BCDMEM (Bind Cue Decide Model of Episodic Memory) [4], TCM (Temporal Context Model) [5], and REM (Retrieving Effective from Memory) [6], etc. However, the majority of these memory models merely focus on the words list study, seldom, if ever, the research is conducted on the representation, storage, and retrieval of visual images.

Recently, machine learning algorithms have been found to be able to categorize visual images with high accuracy and the performances can be even, to some extent, comparable to human beings [7, 8, 9]. However, the key distinction between machine learning based image classification algorithms and human decision is that the machine learning based algorithms will classify an image from a new category into a studied category rather than report unknown like a human. In other words, machine learning based classification methods can distinguish one object from some limited studied objects, while humans can distinguish a certain category of objects from some infinite unknown categories. When one perceives an object never seen before, he will respond ‘unknown’ to it immediately instead of checking it against all the studied objects in his memory, indicating that the human brain memory works in a different way during performing classification tasks. Therefore, in this paper, we will shed light on how visual images are represented and stored as well as how the stored images are retrieved when performing categorization tasks.

Murdock et al. [10] strengthened that a reasonable memory model should address the following issues, i.e., how the information is organized, encoded and stored during learning and how the information could be retrieved or recalled when needed. These topics have gained strong concerns in research areas such as computer vision and image processing and have come a long way. However, the performances yielded are still far from human beings who can make decisions based on the experience and the knowledge learned in the past. Therefore, we can glean some insights from the human brain, especially the mechanism for making an inference and decision to enhance the robustness of the computer vision algorithms. In fact, in the early 1980s, Hinton and Sejnowski showed in their study that the human brain can be regarded as a machine which could infer and make decisions based on external uncertainties [11], suggesting that the brain represents knowledge and makes decisions in the form of probability distributions. Knill and Pouget [12] further developed the idea based on psychophysical evidence and proposed the concept of Bayesian brain which works on

Bayesian probability. In their research, they also described the neural representation mechanism of uncertainty in the human brain. Meanwhile, Lee and Mumford [13] also suggested that the hierarchical Bayesian inference might model the interactive computations in visual cortex and provided some neurophysiological evidences that support the plausibility of their study. Since then, Bayesian models were widely applied in inductive learning and reasoning [14], free recall of memory [15], visual search and recognition [16] as well as visual attention [17].

In this paper, in order to imitate the memory function of the human brain, we apply the probabilistic inference and Bayesian decision theories to the modeling of visual images storage and retrieval and put forward a probabilistic memory model (PMM) for visual images categorization. First, regarding the image representation and storage, we apply the deep convolutional neural network (CNN) to extract the image features, and each feature component is supposed to conform to a Gaussian distribution and may be incompletely copied with a certain probability or randomly produced according to an exponential distribution. The probability representation aims to mimic the sensory error as well as the memory noise of the human brain in the study phase. Then the likelihood ratio between the test image and all the studied images can be obtained, and an odd value can be computed based on all the likelihood values, which is regarded as a criterion for deciding whether the image is studied or not. For image retrieval, the Bayesian decision rule is performed. We evaluate the performance of the proposed PMM by applying it on image categorization tasks in which experiments are conducted on two benchmark image databases and the results are compared with the state-of-the-art image classification methods.

The structure of the paper is organized as follows. Section 2 elaborates the related work. Section 3 introduces the proposed PMM model using probabilistic inference and Bayesian decision. Section 4 provides the experiments and performances of PMM on visual image classification tasks. Finally, some conclusions are summarized in Section 5.

2 RELATED WORK

In this section, we will retrospect some relevant work about visual image representation, storage and recall, i.e., feature representation and the traditional memory model – REM model [6].

As for feature representation, currently a mass of methods for images feature extraction have been put forward, such as histogram of oriented gradient (HOG) [18], local binary patterns (LBP) [19], Haar-like [20] and gradient location-orientation histogram (GLOH) [21], etc. These methods can achieve good results when applied to simple images, but the effects will get worse when dealing with complex natural images. Recently, deep learning based feature extraction methods exhibit better performance when handling images with noise or cluttered background in comparison to the traditional approaches, such as autoencoder neural networks (ANN) [22], restricted Boltzmann machine (RBM) [23], recurrent neural network (RNN) [24] and convolutional neural networks (CNN) [25], etc. Notably, the hierarchical convolutional neural network (HCNN) [26] can model the neural single-unit and population responses in the visual cortical areas of the human brain, especially, the VGG-19 model [25] is known for its deep representation of visual images.

As for the traditional memory models, the REM model is the most typical memory model for words list study. The REM model supposed that human memory consisted of a series of separate images, with each one being a vector of feature values, which are stored randomly or incompletely copied from the studied vector with error. When a word was studied, the storage probability of each feature was μ^*. It was worth noting that when any feature value was stored, it did not change any longer. When some information was stored in the feature vector, the probability of stored value consisted of two parts: the rightly copied probability was c and the randomly chosen probability was $1 - c$ which satisfied a geometric probability distribution $P(V = j) = (1 - g)^{j-1}g,\ j = 1, 2, \ldots, \infty$. Given the test vector either studied or unstudied, it was matched against the studied feature vectors. The matching result $D = \{D_j\}_{j=1,2,\ldots,N}$ was then computed to find the matched and non-matched locations, and neglected the locations where the feature value was zero, where D_j denoted the aligning result between the test and the j^{th} word feature vector.

Next, calculated the likelihood ratio λ_j based on the matching result D_j:

$$\lambda_j = \frac{P(D_j|S_j)}{P(D_j|N_j)} \tag{1}$$

where S_j and N_j standed for the events that the j^{th} image was an s-image and a d-image, respectively. Finally, calculated the odds in favor of an old over a new item based on the likelihood values:

$$\Phi = \frac{1}{N}\sum_{j=1}^{N}\lambda_j. \tag{2}$$

If $\Phi > 1$, the test word was judged as an old one; or else, the test word was identified as new.

REM model offers a memory mechanism to elucidate a series of episodic memory phenomena, such as the list-length, list-strength, word-frequency and mirror effects, etc. However, it can only provide some accounts for word list study. Due to the complex characteristics of visual images, the research about the representation, storage, and retrieval of visual images is still a challenging task.

3 THE PROPOSED PMM MODEL

In what follows, we will elaborate the proposed PMM modeling process, including image representation and storage with probability, memory recall by probability reference, and memory retrieval with Bayesian decision.

3.1 Visual Image Representation and Storage with Probability

The primary part of the memory modeling process is feature representation for the information to be stored, which is essential for the performance of the model. In words list study, the feature of a word is expressed by integers, which is very simple and easy to handle, whereas the feature of an image is much more complicated. Neural evidence shows that the visual information in human brain is processed hierarchically [27, 28]. Thus a pre-trained convolutional neural network model with 19 layers, named VGG-19 [25], is adopted to extract the primitive image features.

More specifically, given a visual image set $X = \{X_{11}, X_{12}, \ldots, X_{1n_1}, X_{21}, X_{22}, \ldots, X_{2n_2}, \ldots, X_{C1}, X_{C2}, \ldots, X_{Cn_C}\}$, where X_{ij} denotes the j^{th} image coming from the i^{th} category, C represents the number of categories in the image set and is signified by $Z = Z_1, Z_2, \ldots, Z_C$, and the number of images in the i^{th} category Z_i is n_i. We first employ the VGG-19 model to extract the feature values of images during the study phase, which are represented as $F = \{F_{11}, F_{12}, \ldots, F_{1n_1}, F_{21}, F_{22}, \ldots, F_{2n_2}, \ldots, F_{C1}, F_{C2}, \ldots, F_{Cn_C}\}$, where $F_{ij} \in \mathbf{R}^{1\times D}$ indicates the feature vector of image X_{ij}, D implies the dimension of the feature vector.

To simulate the visual sensor error, for computational simplicity and convenience, each extracted image feature is modeled using a Gaussian distribution:

$$q\left(F_{ij}|\mu_i\right) \sim N\left(f_{ij} : \mu_i, \sigma^2\right) \;\; i \in [1, C] \tag{3}$$

where f_{ij} represents the observed feature values of the j^{th} image from the i^{th} category, μ_i is the mean value of the i^{th} category, and σ^2 is the precision parameter. Suppose each feature value of images of each category is independent for each other, then the probability distribution of the image features of each category takes the form:

$$q(F|\mu) = \prod_{i=1}^{C} q(F_{ij}|\mu_i). \tag{4}$$

To imitate the storage process of the human brain, we hypothesize that the feature vector of each studied image is stored either correctly copied or an incomplete and error prone copy, suggesting the image features extracted by CNN are incomplete, with some feature components produced randomly. Here, we assume that each stored feature value may be either correctly copied with probability ρ, satisfying a Gaussian distribution, or randomly produced with probability $1-\rho$, conforming to an exponential distribution (allowing the probability of accidentally selecting the correct value):

$$g(F = x) = \begin{cases} ge^{-gx}, & x > 0, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

where g is the exponential distribution parameter. The reason for choosing exponential distribution is that exponential distribution can describe the time probability distribution between the events in the Poisson process, which is broadly consistent with the neurons encoding process in the human brain [29].

In summary, the probabilistic storage process of the PMM can be described by a 4-tuple, i.e., the three parameters of the two probability distributions and one correctly copied probability ρ:

$$F \sim \{\mu, \sigma, g, \rho\}\,. \tag{6}$$

3.2 Visual Images Recall

Given a test image X_t, memory recall means determining whether it is studied or not. The feature vector $F_t \in \mathbf{R}^{1\times D}$ of the test image X_t is extracted by deep CNN first and then matched with the studied feature set $F = \{F_{11}, \ldots, F_{Cn_C}\}$. The matching rule between the test feature vector F_t and the learned feature vector F_{ij} is set as follows:

$$R_{ij}(d) = \begin{cases} 1, & \text{if } F_t(d) \neq 0 \text{ and } F_{ij}(d) \neq 0, \\ -1, & \text{if } F_t(d) \neq 0 \text{ and } F_{ij}(d) = 0, \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

where $1 \leq d \leq D$, in what follows, $F_t(d)$ and $F_{ij}(d)$ are abbreviated as F_d and $F_{d,ij}$ respectively. The matched result R_{ij} is used to find those positions whose values match or mismatch. When F_t and F_{ij} are matched, $R_{ij} = 1$, when F_t and F_{ij} are not matched, $R_{ij} = -1$, the other cases are ignored. The whole matching result is expressed by $R = \{R_{11}, R_{12}, \ldots, R_{Cn_C}\}$.

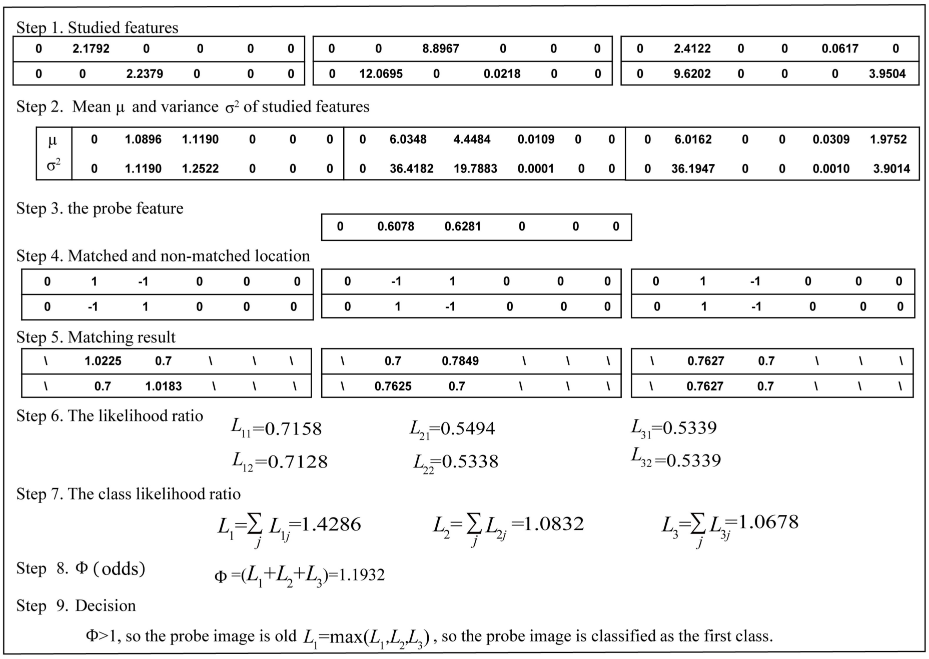
Then the likelihood ratio L_{ij} is calculated based on each matching result R_{ij} and equals the probability that image X_{ij} is an s-image (same as the test image) over the probability being a d-image (different from the test image). Suppose the prior probability of s-image and d-image is identical, then we have:

$$L_{ij} = \frac{P(Y_{ij}|R_{ij})}{P(N_{ij}|R_{ij})} = \frac{\frac{P(Y_{ij})P(R_{ij}|Y_{ij})}{P(R_{ij})}}{\frac{P(N_{ij})P(R_{ij}|N_{ij})}{P(R_{ij})}} = \frac{P(R_{ij}|Y_{ij})}{P(R_{ij}|N_{ij})} \tag{8}$$

where Y_{ij} and N_{ij} represent the events that image X_{ij} is an s-image and a d-image, respectively. Then,

$$L_{ij} = \prod_{d=1}^{D} \frac{P(F_{d,ij}|Y_{ij}, F_d)}{P(F_{d,ij}|N_{ij}, F_d)} \tag{9}$$

where F_d denotes the d^{th} feature value coming from the test image, $F_{d,ij}$ is the d^{th} feature value of image X_{ij}. $P(F_{d,ij}|Y_{ij}, F_d)$ is the probability when X_{ij} is an s-image, the d^{th} feature value in the test image is F_d, and the d^{th} feature value in the feature vector of the image X_{ij} is $F_{d,ij}$. $P(F_{d,ij}|N_{ij}, F_d)$ is the probability when X_{ij} is an d-image, the d^{th} feature value in the test image is F_d, and the d^{th} feature value in



Step 1. Studied features

0	2.1792	0	0	0	0	0	0	8.8967	0	0	0	0	2.4122	0	0	0.0617	0
0	0	2.2379	0	0	0	0	12.0695	0	0.0218	0	0	0	9.6202	0	0	0	3.9504

Step 2. Mean μ and variance σ² of studied features

μ	0	1.0896	1.1190	0	0	0	0	6.0348	4.4484	0.0109	0	0	0	6.0162	0	0	0.0309	1.9752
σ²	0	1.1190	1.2522	0	0	0	0	36.4182	19.7883	0.0001	0	0	0	36.1947	0	0	0.0010	3.9014

Step 3. the probe feature

0	0.6078	0.6281	0	0	0

Step 4. Matched and non-matched location

0	1	-1	0	0	0	0	-1	1	0	0	0	0	1	-1	0	0	0
0	-1	1	0	0	0	0	1	-1	0	0	0	0	1	-1	0	0	0

Step 5. Matching result

\	1.0225	0.7	\	\	\	\	0.7	0.7849	\	\	\	\	0.7627	0.7	\	\	\
\	0.7	1.0183	\	\	\	\	0.7625	0.7	\	\	\	\	0.7627	0.7	\	\	\

Step 6. The likelihood ratio $L_{11}=0.7158$ $L_{21}=0.5494$ $L_{31}=0.5339$

$L_{12}=0.7128$ $L_{22}=0.5338$ $L_{32}=0.5339$

Step 7. The class likelihood ratio

$L_1=\sum_j L_{1j}=1.4286$ $L_2=\sum_j L_{2j}=1.0832$ $L_3=\sum_j L_{3j}=1.0678$

Step 8. Φ (odds) $\Phi=(L_1+L_2+L_3)=1.1932$

Step 9. Decision

$\Phi>1$, so the probe image is old $L_1=\max(L_1,L_2,L_3)$, so the probe image is classified as the first class.

Figure 1. An illustration of the storage and retrieval process for PMM

the feature vector of image X_{ij} is $F_{d,ij}$. According to the matching results in R, Equation (9) can be decomposed into

$$L_{ij} = \prod_{d\in Q} \frac{P(F_{d,ij}|Y_{ij}, F_d)}{P(F_{d,ij}|N_{ij}, F_d)} \prod_{d\in M} \frac{P(F_{d,ij}|Y_{ij}, F_d)}{P(F_{d,ij}|N_{ij}, F_d)} \tag{10}$$

where M and Q are the sets of exponents for the matched and nonmatched non-zero features, respectively. When $d \in Q$, for s-images, the d^{th} feature of X_{ij} does not match with the test image, meaning the feature is wrongly copied, i.e.,

$$P(F_{d,ij}|Y_{ij}, F_d) = (1-\rho)P(F_{d,ij}|N_{ij}, F_d). \tag{11}$$

Let $n_{ij,Q}$ be the number of all mismatched non-zero feature values in image X_{ij}, then by substituting Equation (11) into Equation (10), we obtain:

$$L_{ij} = (1-\rho)^{n_{ij,Q}} \prod_{d\in M} \frac{P(F_{d,ij}|Y_{ij}, F_k)}{P(F_{d,ij}|N_{ij}, V_d)}. \tag{12}$$

When $d \in M$, the d^{th} feature of X_{ij} matches with the test image, for d-images, the probability of storing value $F_{d,ij}$ satisfies an exponential distribution $g(F)$, as shown in Equation (5). Whereas for s-images, the probability of storing value $F_{d,ij}$ consists of two parts: the correctly copied probability which satisfies the Gaussian distribution and the randomly chosen probability which is distributed exponentially, i.e.,

$$P(F_{d,ij}|Y_{ij}, F_d) = \rho q(F_{d,ij}) + (1-\rho)g(F_{d,ij}), \tag{13}$$

$$\frac{P\left(F_{d,ij}|Y_{ij}, F_d\right)}{P\left(F_{d,ij}|N_{ij}, F_d\right)} = \frac{\rho q(F_{d,ij}) + (1-\rho)g(F_{d,ij})}{g(F_{d,ij})} \tag{14}$$

where $q(F_{d,ij}) = \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{1}{2\sigma^2}(F_{d,ij}-\mu_i)^2]$ and $g(F_{d,ij}) = gexp(-gF_{d,ij})$. Then, we have

$$\begin{aligned} L_{ij} &= (1-\rho)^{n_{ij,Q}} \prod_{d\in M} \frac{\rho q(F_{d,ij}) + (1-\rho)g(F_{d,ij})}{g(F_{d,ij})} \\ &= (1-\rho)^{n_{ij,Q}} \prod_{d\in M} \frac{\rho q(F_{d,ij}) + (1-\rho)ge^{-gF_{d,ij}}}{ge^{-gV_{d,ij}}}. \end{aligned} \tag{15}$$

In summary, the likelihood ratio set symbolized as $L = \{L_{ij}\}_{i=1,\dots,C,j=1,\dots,n_C}$ can be obtained by checking against the given test image X_t with all studied images $X = \{X_{ij}\}_{i=1,\dots,C,j=1,\dots,n_C}$.

With the above likelihood ratios, whether the test image X_t is studied (old) or not (new) during the learning phase can be determined.

Let Φ denote the odds, which equals the probability that the test image X_t is old over the probability being new. Then we have,

$$\Phi = \frac{P(O|R)}{P(N|R)} = \frac{\frac{P(O)P(R|O)}{P(R)}}{\frac{P(N)P(R|N)}{P(R)}} = \frac{P(R|O)}{P(R|N)} \tag{16}$$

where $P(R|O) = \sum_{i=1}^{C}\sum_{j=1}^{n_i} P(R|Y_{ij})P(Y_{ij}))$, C is the number of image categories, n_i is the number of images of the i^{th} category Z_i.

Assume the probability of being an s-image for each class is identical and the probability of being an s-image for each test image is identical too, i.e. $P(Y_{ij}) = \frac{1}{C}\frac{1}{n_i}$. Then,

$$\Phi = \sum_{i=1}^{C}\sum_{j=1}^{n_i} \frac{1}{C}\frac{1}{n_i}\frac{P(R|Y_{ij})}{P(R|N)} = \frac{1}{C}\sum_{i=1}^{C}\frac{1}{n_i}\sum_{j=1}^{n_i}\frac{P(R|Y_{ij})}{P(R|N)}. \tag{17}$$

When the studied X_{ij} is an s-image, i.e., event Y_{ij} happens, the other images are d-image, i.e., N event happens, then

$$P(R|Y_{ij}) = P(R_{ij}|Y_{ij}) \prod_{i\neq i', j\neq j'} P(R_{i'j'}|N_{i'j'}). \tag{18}$$

Figure 2. Exemplars in Caltech-101 dataset

Finally, we have

$$\begin{aligned}
\Phi &= \frac{1}{C}\sum_{i=1}^{C}\frac{1}{n_i}\left(\sum_{j=1}^{n_i}\frac{P(R_{ij}|Y_{ij})\prod_{i\neq i',j\neq j'}P(R_{i'j'}|N_{i'j'})}{P(R_{ij}|N_{ij})\prod_{i\neq i',j\neq j'}P(R_{i'j'}|N_{i'j'})}\right)\\
&= \frac{1}{C}\sum_{i=1}^{C}\frac{1}{n_i}\left(\sum_{j=1}^{n_i}\frac{P(R_{ij}|Y_{ij})}{P(R_{ij}|N_{ij})}\right)\\
&= \frac{1}{C}\sum_{i=1}^{C}\frac{1}{n_i}\sum_{j=1}^{n_i}L_{ij}\\
&= \frac{1}{C}\sum_{i=1}^{C}\sum_{j=1}^{n_i}\frac{1}{n_i}L_{ij}. \qquad (19)
\end{aligned}$$

Given a threshold θ, if $\Phi > \theta$, the test image is judged as 'studied' (old), otherwise 'unstudied' (new).

3.3 Memory Retrieval with Bayesian Decision

When the test image is studied, the Bayesian decision rule can be adopted to determine its category, i.e., memory retrieval.

Given a test image X_t, assume there are two different classes Z_a and Z_b, $1 \leq a$ and $b \leq C$. Assume the posterior probability of being class Z_a and Z_b are $P(Z_a|X_t)$ and $P(Z_b|X_t)$, $1 \leq a$ and $b \leq C$, respectively, then according to Bayesian decision theory,

$$X_t \in \begin{cases} Z_a, & if\ P(Z_a|X_t) > P(Z_b|X_t),\\ Z_b, & \text{otherwise.}\end{cases} \qquad (20)$$

Let $\{X_{aj}\}_{j=1,\ldots,n_a}$ and $\{X_{bj}\}_{j=1,\ldots,n_b}$ denote the studied images for the two classes Z_a and Z_b, respectively. Hence, for two-category problem, the Bayesian decision rule takes the form,

$$I_t \in \begin{cases} Z_a, & \text{if } \frac{P(Z_a|X_t)}{P(Z_B|X_t)} > 1, \\ Z_b, & \text{otherwise.} \end{cases} \tag{21}$$

Figure 3. Exemplars in Caltech-256 dataset

$$\begin{aligned} \frac{P(Z_a|X_t)}{P(Z_b|X_t)} &\propto \frac{P(R|Y_a)}{P(R|Y_b)} \\ &= \frac{\sum_{j=1}^{n_a} P(R|Y_{aj})P(Y_{aj})}{\sum_{j=1}^{n_b} P(R|Y_{bj})P(Y_{bj})} \\ &= \frac{\frac{1}{C}\frac{1}{n_a}\sum_{j=1}^{n_a} P(R|Y_{aj})}{\frac{1}{C}\frac{1}{n_b}\sum_{j=1}^{n_b} P(R|Y_{bj})} \\ &\propto \frac{\frac{1}{n_a}\sum_{j=1}^{n_a} P(R_{aj}|Y_{aj})}{\frac{1}{n_b}\sum_{j=1}^{n_b} P(R_{bj}|Y_{bj})} \\ &\propto \frac{\frac{1}{n_a}\sum_{j=1}^{n_a} L_{aj}}{\frac{1}{n_b}\sum_{j=1}^{n_b} L_{bj}} \end{aligned} \tag{22}$$

Applying the Bayesian decision rule, we have,

$$X_t \in \begin{cases} Z_a, & \text{if } \frac{1}{n_u}\sum_{j=1}^{n_a} L_{aj} > \frac{1}{n_b}\sum_{j=1}^{n_b} L_{bj}, \\ Z_b, & \text{otherwise.} \end{cases} \tag{23}$$

Similarly, for multi-category classification problem, given the studied class set $Z = Z_1, Z_2, \ldots, Z_C$, then the final Bayesian decision rule can be defined as

$$X_t \in Z_a, \text{ if } \frac{1}{n_a}\sum_{j=1}^{n_a} L_{aj} > \frac{1}{n_b}\sum_{j=1}^{n_b} L_{bj} \tag{24}$$

for all $b \neq a$, $1 \leq a, b \leq C$. The storage and retrieval process for PMM is illustrated in Figure 1, where the parameters ρ, g and θ are assumed to be 0.1, 0.01 and 1, respectively. The process includes nine steps:

The storage and retrieval process for PMM
Step 1: Given the studied feature vectors coming from three different classes with each consisting of two images.
Step 2: Estimate the corresponding means μ and variance σ^2 of the two studied feature values in each class, respectively.
Step 3: Present the feature vector of the test image whose category needs to be judged.
Step 4: Find the matched and nonmatched locations between the feature values of the studied images and the test image according to Equation (7), where 1 represents the matched features, -1 stands for the nonmatched features.
Step 5: Given the matching results. For the positions where the value is 0, the matching result is ignored. For nonmatched positions where the value is -1, the matching result is set as 1-ρ, whereas for the matched positions where the value is 1, the matching result is computed according to Equation (14).
Step 6: Calculate the likelihood ratio L_{ij} by multiplying all the matching results for each image according to Equation (15).
Step 7: Calculate the class likelihood ratio based on the summation of all the ratios of the two studied images from the same category.
Step 8: Compute the odd value Φ based on all likelihood values according to Equation (19).
Step 9: Make the decision. If the odd value is larger than a certain threshold, the test image is decided as old, and the Bayesian decision rule is applied to the image classification, otherwise decided as new.

4 EXPERIMENTS

For the purpose of performance evaluation of the proposed PMM, we conduct the experiments on two benchmark image databases, Caltech-101 [30] and Caltech-256 [31]. The experiments are implemented on MATLAB R2017b. The test environment is a laptop computer with Intel (R) Core (TM) i7-4770 CPU @ 3.40 GHz, 16 GB memory, and Windows10 operating system. First, the parameters of the PMM are tuned to reach the optimal performance, and then the effects on image classification by PMM are compared with several image classification methods, including the traditional support vector machine (SVM) [32] and some state-of-the-art methods such as VGG-19 [33], ResNet-50 [34], SLRC [35], ICS-DLSR [36], WKRBM [37], Euler-SRC [38] and NRC [39]. Finally, the memory recognition performance of PMM is evaluated. All the experimental results are averaged by 10 runs.

4.1 Datasets

4.1.1 Caltech-101 Dataset

The Caltech-101 dataset contains a total of 9 144 images from 102 classes, one of which is the background, the number of images for each class ranges from 31 to 800. Figure 2 shows some exemplars in Caltech-101 dataset.

a) Exemplars of training set

b) Exemplars of studied images for testing

c) Exemplars of unstudied images for testing

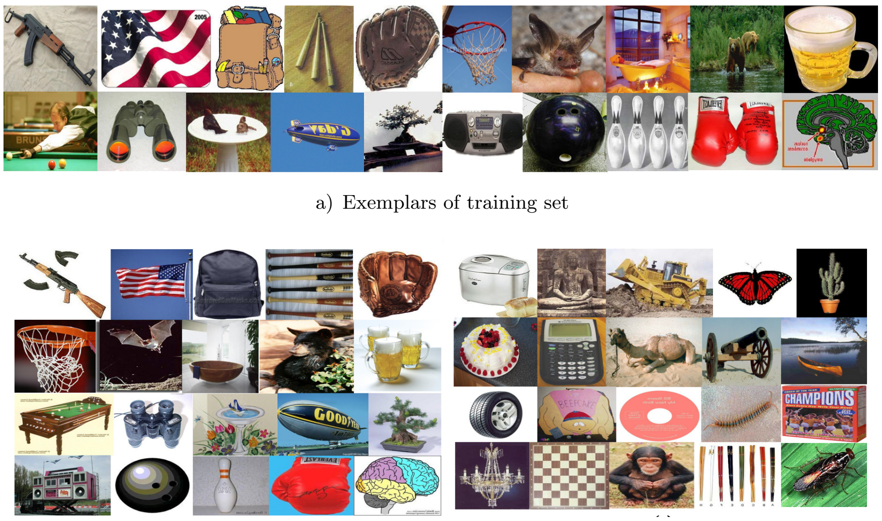
Figure 4. Exemplars of training images and test images on Caltech-101 dataset

4.1.2 Caltech-256 Dataset

The Caltech-256 dataset contains 257 categories with a total of 30 607 images with each category more than 80 different images. Figure 3 shows some exemplars in Caltech-256 dataset.

4.2 Parameters Optimization

As mentioned above, the probabilistic storage process of the PMM can be described by 4 parameters, two Gaussian distribution parameters μ and σ, and two other probability parameters ρ and g. For the Gaussian distribution parameters μ and σ, they can be estimated by the images of each category after feature extraction. While only the parameters μ and g need to be optimized. In the experiment, for each dataset, we randomly choose 20 categories and 20 images per category for training,



a) Exemplars of training set

b) Exemplars of studied images for testing c) Exemplars of unstudied images for testing

Figure 5. Exemplars of training images and test images on Caltech-256 dataset

then randomly choose 10 images of each category for hit rate evaluation, meanwhile randomly choose 20 new categories (10 images of each category) for false alarm evaluation, that is, the test images include 200 studied images and 200 new categories images. Some image exemplars are shown in Figure 4 and Figure 5. Figure 6 and Figure 7 demonstrate the comparison results of the recognition performance under varying parameters g and ρ. As shown in Figure 6 and Figure 7, for both datasets, when $g = 0.01$ and $g = 0.02$, the hit rates are higher than other g, while when $g = 0.02$, the false alarm rates are obviously lower than $g = 0.01$. When $\rho = 0.1$, the hit rates are higher and the false alarm rates are lower than other ρ. Accordingly, the optimal parameters are set as $g = 0.02$ and $\rho = 0.1$ for both datasets.

4.3 Image Classification with PMM

For further validation of the pure classification performance, we compare the PMM model with SVM [32] and some state-of-the-art methods such as VGG-19 [33], ResNet-50 [34], SLRC [35], ICS-DLSR [36], WKRBM [37], Euler-SRC [38] and NRC [39] on both datasets. In the experiment, for each dataset, we randomly select 20, 40, 80 categories and 30 images per category, including 10 images as training set and 20 images as test set or 25 images as training set and 5 images as test set, respectively. The image features are extracted with the trained convolutional neural network VGG-19 model except the ResNet-50 method, in which the image features are extracted with the trained residual neural network model. In SVM, the extracted features are fed into a linear support vector machine (LIB-SVM [32]). As shown in Tables 1 and 2, the classification performance of PMM has the highest classification rate in comparison to all the other eight models with both datasets.

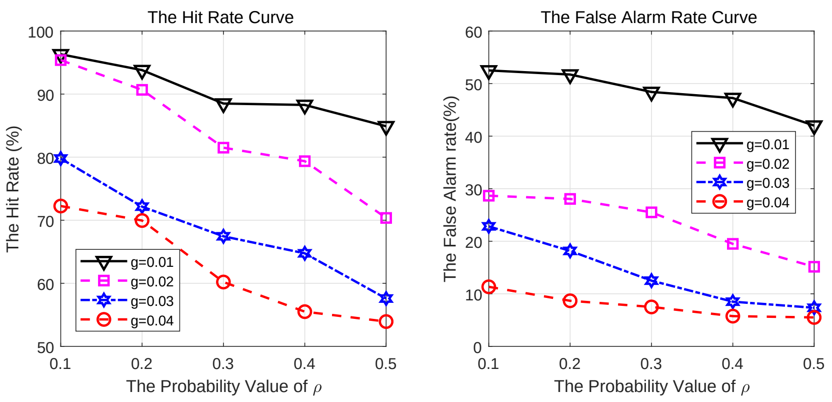


Figure 6. Comparison results of the memory recognition performance under varying parameters ρ and g on Caltech-101 dataset

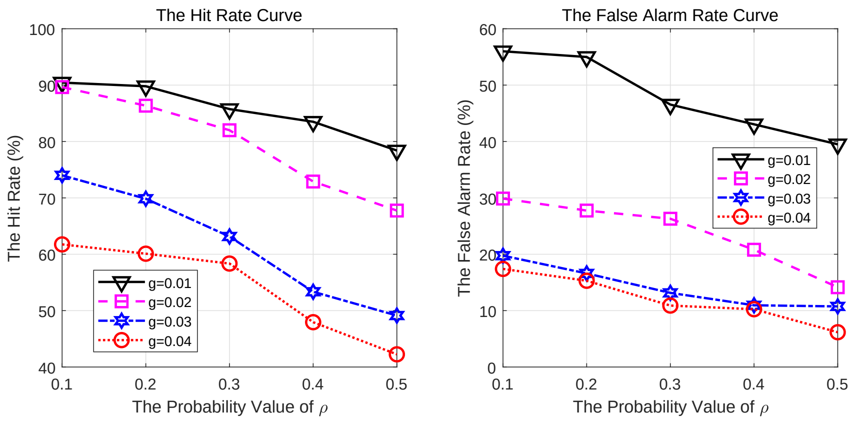


Figure 7. Comparison results of the memory recognition performance under varying parameters ρ and g on Caltech-256 dataset

4.4 Recognition Performance Evaluation

Memory recognition performance means the rate of responding 'old' or 'new' to a probe image. To evaluate the effects of recognition, we compare the PMM model with some machine learning based approaches including VGG-19 [33], ResNet-50 [34] and NRC [39] on both datasets, which also show good classification performance. In

Images	Methods	20 categories	40 categories	80 categories
10 train/20 test	PMM	95.80	94.55	91.65
	SVM	91.73	87.94	83.96
	VGG-19	92.72	91.34	91.31
	ResNet-50	95.10	91.63	91.35
	SLRC	90.24	88.97	87.60
	ICS-DLSR	89.61	87.82	86.80
	WKRBM	88.51	87.31	85.22
	Euler-SRC	88.11	87.16	86.77
	NRC	93.55	91.47	90.32
25train/5test	PMM	97.80	96.40	92.85
	SVM	92.80	91.85	88.95
	VGG-19	96.52	94.62	92.75
	ResNet-50	96.60	94.70	92.92
	SLRC	92.81	90.61	88.53
	ICS-DLSR	89.91	88.32	87.34
	WKRBM	92.51	90.43	89.53
	Euler-SRC	91.89	90.13	88.64
	NRC	94.11	93.72	91.96

Table 1. Comparison classification performance of PMM with the other eight methods on Caltech-101 dataset (%)

Images	Methods	20 categories	40 categories	80 categories
10 train/20 test	PMM	91.83	87.31	84.89
	SVM	85.72	80.45	76.65
	VGG-19	88.22	87.09	83.56
	ResNet-50	91.05	87.23	83.96
	SLRC	88.69	86.25	81.81
	ICS-DLSR	87.95	85.83	80.62
	WKRBM	88.53	84.41	81.38
	Euler-SRC	88.43	85.75	82.12
	NRC	89.25	86.33	83.31
25 train/5 test	PMM	93.78	90.44	86.44
	SVM	89.56	88.61	82.81
	VGG-19	90.01	89.72	86.15
	ResNet-50	92.40	89.95	86.50
	SLRC	90.86	87.50	84.50
	ICS-DLSR	91.31	88.83	84.08
	WKRBM	91.38	87.13	83.66
	Euler-SRC	91.85	87.65	84.25
	NRC	91.80	88.75	85.75

Table 2. Comparison classification performance of PMM with the other eight methods on Caltech-256 dataset (%)

the experiment, for each dataset, we randomly choose 20 categories and 20 images per category for training, then randomly choose 10 images of each category for testing, meanwhile randomly choose 20 new categories (10 images of each category) for testing, that is, the test images include 200 studied images and 200 new category images. Tables 3 and 4 list the hit rates and false alarm rates for each of the methods for both datasets. It can be seen that these machine learning based methods fail to decide whether a test image is studied or not, whereas the proposed PMM is able to recognize most of the unstudied images.

Methods	P(H)	P(F)
PMM	96.07	29.28
VGG-19	91.85	100
ResNet-50	93.33	100
NRC	92.25	100

Table 3. Comparison recognition results of the hit rates P(H) (%) and the false alarm rates P(F) (%) of PMM, VGG-19, ResNet-50 and NRC methods on Caltech-101 dataset

Methods	P(H)	P(F)
PMM	91.67	29.98
VGG-19	87.37	100
ResNet-50	90.33	100
NRC	88.78	100

Table 4. Comparison recognition results of the hit rates P(H) (%) and the false alarm rates P(F) (%) of PMM, VGG-19, ResNet-50 and NRC methods on Caltech-256 dataset

5 CONCLUSIONS

In this paper, we have presented a probabilistic memory model (PMM) for visual image storage and recall based on probabilistic inference and Bayesian decision. We assume that the original visual image feature values extracted by deep CNN are stored stochastically, with being correctly copied with some probability or randomly produced conforming to an exponential distribution. The probabilistic generative model can facilitate the matching process between the test image and the studied images during learning. Thus, when a new object of a new category is presented, PMM can judge it as an unknown object instead of identifying it from a studied category. Additionally, the proposed PMM can also be applied to the classification of natural multi-category images. We have conducted experiments on Caltech-101 and Caltech-256 databases to validate the effectiveness of PMM. The results manifest that PMM achieves better classification performance in image classification tasks than most state-of-the-art image classification methods. It is worth highlighting that PMM can also determine the images from an unstudied category, whereas the other methods fail.

The current PMM can imitate the human brain memory for visual images learning to some extent and offer a new way for visual images categorization, however, there are also some limitations to PMM. First, the probabilistic model parameters are chosen experimentally and might be influenced by the image datasets used. Secondly, the matching rule between the test feature vector and the learned feature vectors seems simple and might not be consistent with the way of humans memorizing visual images. These two issues will remain for further investigation in our future work.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (No. 62072468), the Shandong Provincial Natural Science Foundation, China (Nos. ZR2018MF017, ZR2019MF073), and the Fundamental Research Funds for the Central Universities, China University of Petroleum (East China) (No. 18CX060-60A).

REFERENCES

[1] Lee, M. D.—Vanpaemel, W.: Determining Informative Priors for Cognitive Models. Psychonomic Bulletin and Review, Vol. 25, 2018, No. 1, pp. 114–127, doi: 10.3758/s13423-017-1238-3.

[2] Atkinson, R. C.—Shiffrin, R. M.: Human Memory: A Proposed System and Its Control Processes. Psychology of Learning and Motivation, Vol. 2, 1968, No. 1, pp. 89–195, doi: 10.1016/s0079-7421(08)60422-3.

[3] Raaijmakers, J. G. W.—Shiffrin, R. M.: SAM: A Theory of Probabilistic Search of Associative Memory. Psychology of Learning and Motivation, Vol. 14, 1980, pp. 207–262, doi: 10.1016/s0079-7421(08)60162-0.

[4] Dennis, S.—Humphreys, M. S.: A Context Noise Model of Episodic Word Recognition. Psychological Review, Vol. 108, 2001, pp. 452–478, doi: 10.1037/0033-295x.108.2.452.

[5] Polyn, S. M.—Kahana, M. J.: Memory Search and the Neural Representation of Context. Trends in Cognitive Sciences, Vol. 12, 2008, No. 1, pp. 24–30, doi: 10.1016/j.tics.2007.10.010.

[6] Shiffrin, R. M.—Steyvers, M.: A Model for Recognition Memory: REM – Retrieving Effectively from Memory. Psychonomic Bulletin and Review, Vol. 4, 1997, No. 2, pp. 145–166, doi: 10.3758/bf03209391.

[7] Liu, B.—Jing, L.—Li, J.—Yu, J.—Gittens, A.—Mahoney, M. W.: Group Collaborative Representation for Image Set Classification. International Journal of Computer Vision, Vol. 127, 2019, No. 2, pp. 181–206, doi: 10.1007/s11263-018-1088-0.

[8] Wan, W.—Zhong, Y.—Li, T.—Chen, J.: Rethinking Feature Distribution for Loss Functions in Image Classification. Proceedings of 2018 IEEE/CVF Conference

on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 2018, pp. 9117–9126, doi: 10.1109/cvpr.2018.00950.

[9] Liu, B. D.—Gui, L.—Wang, Y.—Wang, Y. X.—Shen, B.—Li, X.—Wang, Y. J.: Class Specific Centralized Dictionary Learning for Face Recognition. Multimedia Tools and Applications, Vol. 76, 2017, No. 3, pp. 4159–4177, doi: 10.1007/s11042-015-3042-2.

[10] Murdock, B. B.: A Theory for the Storage and Retrieval of Item and Associative Information. Psychological Review, Vol. 89, 1982, No. 6, pp. 609–626, doi: 10.1037/0033-295x.89.6.609.

[11] Hinton, G. E.—Sejnowski, T. J.: Optimal Perceptual Inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Washington, D. C., June 1983, pp. 448–453.

[12] Knill, D. C.—Pouget, A.: The Bayesian Brain: The Role of Uncertainty in Neural Coding and Computation. Trends in Neurosciences, Vol. 27, 2004, No. 12, pp. 712–719, doi: 10.1016/j.tins.2004.10.007.

[13] Lee, T. S.—Mumford, D.: Hierarchical Bayesian Inference in the Visual Cortex. Journal of the Optical Society of America A – Optics Image Science and Vision, Vol. 20, 2003, No. 7, pp. 1434–1448, doi: 10.1364/josaa.20.001434.

[14] Tenenbaum, J. B.—Griffiths, T. L.—Kemp, C.: Theory-Based Bayesian Models of Inductive Learning and Reasoning. Trends in Cognitive Sciences, Vol. 10, 2006, No. 7, pp. 309–318, doi: 10.1016/j.tics.2006.05.009.

[15] Socher, R.—Gershman, S.—Sederberg, P.—Norman, K.—Perotte, A.—Blei, D.: A Bayesian Analysis of Dynamics in Free Recall. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., Culotta, A. (Eds.): Advances in Neural Information Processing Systems 22 (NIPS 2009), 2009, pp. 1714–1722.

[16] Elazary, L.—Itti, L.: A Bayesian Model for Efficient Visual Search and Recognition. Vision Research, Vol. 50, 2010, No. 14, pp. 1338–1352, doi: 10.1016/j.visres.2010.01.002.

[17] Chikkerur, S.—Serre, T.—Tan, C.—Poggio, T.: What and Where: A Bayesian Inference Theory of Attention. Vision Research, Vol. 50, 2010, No. 22, pp. 2233–2247, doi: 10.1016/j.visres.2010.05.013.

[18] Song, S.—Xu, B.—Yang, J.: SAR Target Recognition via Supervised Discriminative Dictionary Learning and Sparse Representation of the SAR-HOG Feature. Remote Sensing, Vol. 8, 2016, No. 8, Art. No. 683, doi: 10.3390/rs8080683.

[19] Kaplan, K.—Kaya, Y.—Kuncan, M.—Minaz, M. R.—Ertunç, H. M.: An Improved Feature Extraction Method Using Texture Analysis with LBP for Bearing Fault Diagnosis. Applied Soft Computing, Vol. 87, 2020, Art. No. 106019, doi: 10.1016/j.asoc.2019.106019.

[20] Ai, J.—Tian, R.—Luo, Q.—Jin, J.—Tang, B.: Multi-Scale Rotation-Invariant Haar-Like Feature Integrated CNN-Based Ship Detection Algorithm of Multiple-Target Environment in SAR Imagery. IEEE Transactions on Geoscience and Remote Sensing, Vol. 57, 2019, No. 12, pp. 10070–10087, doi: 10.1109/tgrs.2019.2931308.

[21] Mikolajczyk, K.—Schmid, C.: A Performance Evaluation of Local Descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, 2005, No. 10, pp. 1615–1630, doi: 10.1109/tpami.2005.188.

[22] Chicco, D.—Sadowski, P.—Baldi, P.: Deep Autoencoder Neural Networks for Gene Ontology Annotation Predictions. Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB '14), 2014, pp. 533–540, doi: 10.1145/2649387.2649442.

[23] Larochelle, H.—Mandel, M.—Pascanu, R.—Bengio, Y.: Learning Algorithms for the Classification Restricted Boltzmann Machine. The Journal of Machine Learning Research, Vol. 13, 2012, No. 1, pp. 643–669.

[24] Gregor, K.—Danihelka, I.—Graves, A.—Rezende, D.—Wierstra, D.: DRAW: A Recurrent Neural Network for Image Generation. Proceedings of the 32nd International Conference on Machine Learning, PMLR, Vol. 37, 2015, pp. 1462–1471.

[25] Simonyan, K.—Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, 2015. arXiv:1409.1556v6, 2015.

[26] Yamins, D. L. K.—DiCarlo J. J.: Using Goal-Driven Deep Learning Models to Understand Sensory Cortex. Nature Neuroscience, Vol. 19, 2016, No. 3, pp. 356–365, doi: 10.1038/nn.4244.

[27] Fukushima, K.: Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biological Cybernetics, Vol. 36, 1980, pp. 193–202, doi: 10.1007/bf00344251.

[28] Riesenhuber, M.—Poggio, T.: Hierarchical Models of Object Recognition in Cortex. Nature Neuroscience, Vol. 2, 1999, No. 11, pp. 1019–1025, doi: 10.1038/14819.

[29] Beck, J. M.—Ma, W. J.—Kiani, R.—Hanks, T.—Churchland, A. K.—Roitman, J.—Shadlen, M. N.—Latham, P. E.—Pouget, A.: Probabilistic Population Codes for Bayesian Decision Making. Neuron, Vol. 60, 2008, No. 6, pp. 1142–1152, doi: 10.1016/j.neuron.2008.09.021.

[30] Li, F. F.—Fergus, R.—Perona, P.: Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. Computer Vision and Image Understanding, Vol. 106, 2007, No. 1, pp. 59–70, doi: 10.1016/j.cviu.2005.09.012.

[31] Griffin, G.—Holub, A.—Perona, P.: Caltech-256 Object Category Dataset. California Institute of Technology, 2007.

[32] Chang, C. C.—Lin, C. J.: LIBSVM: A Library for Support Vector Machines. ACM Transactions on Intelligent Systems and Technology (TIST), Vol. 2, 2011, No. 3, Art. No. 27, doi: 10.1145/1961189.1961199.

[33] He, K.—Zhang, X.—Ren, S.—Sun, J.: Deep Residual Learning for Image Recognition. Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770–778, doi: 10.1109/cvpr.2016.90.

[34] Deng, W.—Hu, J.—Guo, J.: Face Recognition via Collaborative Representation: Its Discriminant Nature and Superposed Representation. IEEE Transactions on Pat-

tern Analysis and Machine Intelligence, Vol. 40, 2018, No. 10, pp. 2513–2521, doi: 10.1109/tpami.2017.2757923.

[35] Wen, J.—Xu, Y.—Li, Z.—Ma, Z.—Xu, Y.: Inter-Class Sparsity Based Discriminative Least Square Regression. Neural Networks, Vol. 102, 2018, No. 10, pp. 36–47, doi: 10.1016/j.neunet.2018.02.002.

[36] Qin, Y.—Tian, C.: Weighted Feature Space Representation with Kernel for Image Classification. Arabian Journal for Science and Engineering, Vol. 43, 2018, No. 12, pp. 7113–7125, doi: 10.1007/s13369-017-2952-x.

[37] Song, Y.—Liu, Y.—Gao, Q.—Gao, X.—Nie, F.—Cui, R.: Euler Label Consistent K-SVD for Image Classification and Action Recognition. Neurocomputing, Vol. 310, 2018, pp. 277–286, doi: 10.1016/j.neucom.2018.05.036.

[38] Xu, J.—An, W.—Zhang, L.—Zhang, D.: Sparse, Collaborative, or Nonnegative Representation: Which Helps Pattern Classification? Pattern Recognition, Vol. 88, 2019, pp. 679–688, doi: 10.1016/j.patcog.2018.12.023.

[39] Sederberg, P. B.—Norman, K. A.: Learning and Memory: Computational Models. Encyclopedia of Behavioral Neuroscience, 2010, pp. 145–153, doi: 10.1016/b978-0-08-045396-5.00140-8.

Linxia Xiao received her Master's degree in communication and information system from Shandong University of Science and Technology in 2017. Currently, she is studying for her Ph.D. degree in control science and engineering in China University of Petroleum. Her research interests include pattern recognition, computer vision, and cognitive science.

Yanjiang Wang is currently Full Professor with the College of Control Science and Engineering, China University of Petroleum (East China), China. He received his M.Sc. degree in communication and electronic system from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 1989 and the Ph.D. degree in signal and information processing from the Beijing Jiaotong University, Beijing, China, in 2001. He was Postdoctoral Researcher with the Institute of Drilling Engineering, Shengli Oilfield, Dongying, China, from 2003 to 2006. From 2013 to 2014, he was Visiting Scholar with the Department of Psychological and Brain Sciences, Indiana University, Bloomington, Indiana, USA. His current research interests include bio-inspired pattern recognition, cognitive memory modeling and human brain connectivity. He has authored or co-authored more than 200 papers in top journals and prestigious conferences.

Baodi Liu received his Ph.D. degree in electronic engineering from the Tsinghua University, Beijing, China. He is currently Associate Professor with the College of Information and Control Engineering, China University of Petroleum, Qingdao, China. His research interests include computer vision and machine learning.

Weifeng Liu received his double B.Sc. degree in automation and business administration and the Ph.D. degree in pattern recognition and intelligent systems from the University of Science and Technology of China, Hefei, China in 2002 and 2007, respectively. His current research interests include pattern recognition and machine learning.

Computing and Informatics, Vol. 39, 2020, 1250–1281, doi: 10.31577/cai_2020_6_1250

MITIGATING DRAWBACKS OF LOGISTIC MAP FOR IMAGE ENCRYPTION ALGORITHMS

Jakub ORAVEC, Ľuboš OVSENÍK, Ján TURÁN, Tomáš HUSZANÍK

Department of Electronics and Multimedia Communications
Technical University of Košice
Němcovej 32
040 01 Košice, Slovakia
e-mail: {jakub.oravec, lubos.ovsenik, jan.turan, tomas.huszanik}@tuke.sk

Abstract. This paper identifies and analyses some drawbacks of the logistic map which is still one of the most used chaotic maps in image encryption algorithms. As some of the disadvantages are caused by inappropriate implementations of the logistic map, this paper proposes a set of rules which should lead to enhancement of the desired chaotic behavior. Probably the most important rule introduces alternating value of parameter utilized by the logistic map. With careful choice of values and an adapted quantization technique, some of the issues should be fixed and theoretically also the values of numerical parameters should be improved. These assumptions are verified by applying the proposed set of rules on an algorithm from our prior work. Effects of the proposed rules on the used algorithm are investigated and all necessary modifications are thoroughly discussed. The paper also compares obtained values of commonly used numerical parameters and computational complexity with some other image encryption algorithms based on more complex chaotic systems.

Keywords: Fixed point, image encryption, logistic map, Lyapunov exponent, periodic cycle

Mathematics Subject Classification 2010: 94A60, 68U10

1 INTRODUCTION

Nowadays, there are many possible ways for establishing security of multimedia data. Each solution has certain advantages and drawbacks determined by its design and expected applications. This statement is valid even for well-established "conventional" encryption algorithms such as Advanced Encryption Standard (AES) [1]. While AES could be used in a wide amount of applications, it was designed primarily for operations with small blocks of hexadecimal data. Careless implementation of AES for image encryption, especially using simpler modes of operation could lead to undesired results. One example of this situation is shown in Figure 1 where a grayscale image with a resolution of 256×128 pixels was encrypted by AES using Electronic CodeBook (ECB) mode of operation [2]. This example used key $0\times$ `C4 EB 50 BC 0E C5 EB 50 BC 0E C5 EB 50 BC 0E C5` for the encryption.

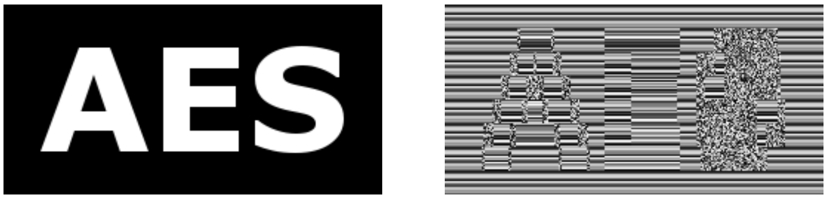


Figure 1. Visible artifacts in an image encrypted by AES ECB

The encrypted image displayed in Figure 1 shows that AES in ECB mode could create visible artifacts that are correlated with the plain image. This drawback is not present in other modes of operation [2], but not all users are familiar with it.

Presented issue of AES in ECB mode is caused by its design – the image is processed as a set of blocks. Each block is replaced by a corresponding block from a code book during the encryption. If the plain image contains several identical blocks of image pixel intensities, all these blocks are replaced by the same block from the code book. This situation is undesirable since the images are known for rather high redundancy and they can contain multiple groups of identical blocks.

On the other hand, encryption algorithms that were designed specifically for image encryption try to adjust their steps to characteristics of image data. This means that the dedicated image encryption algorithms are robust against attacks that are feasible on image data. Also, performance of the dedicated algorithms should be better, however this is hard to prove as the conventional algorithms use various tricks such as hardware acceleration [3] for increasing their performance.

The dedicated image encryption algorithms found their applications mainly in areas which are sensitive to presence of even small artifacts such as encryption of medical images [4] or encryption of secret data in steganographic systems [5]. Also some biometric systems could benefit from lower computational complexity of carefully designed image encryption algorithms [6].

The history of dedicated image encryption algorithms goes back to 1989, when Matthews proposed an encryption algorithm that utilized logistic map (LM) [7]. While this scheme did not encrypt images, it described some techniques that later became essential for the image encryption algorithms. The choice of used chaotic map was explained by Matthews mainly by good understanding of the LM at the time. One of the first extensive studies of LM was published already in 1976 by May [8] and since then, multiple works investigated properties of LM in general [9, 10, 11], or its properties in certain applications [12, 13, 14, 15, 16, 17].

The work of Matthews was later continued by Fridrich, who described one of the first encryption algorithms designed specifically for operations with images in 1998 [18]. Probably the biggest advantage of Fridrich's algorithm is its architecture which was later adopted by many other image encryption algorithms. Some of the drawbacks of this algorithm and similar ones were described by Solak et al. in 2010 [19] and later also by Xie et al. in 2017 [20]. The newest image encryption algorithms take into account the drawbacks of Fridrich's architecture and they utilize techniques to overcome these issues [21, 22, 23, 24]. However, less work has been done in mitigating vulnerabilities of the LM in image encryption algorithms even if they are already described [12, 14].

These vulnerabilities such as occurrence of periodic cycles [10], existence of a relation between successive iterates or their uneven distribution [12] negatively affect generated pseudo-random (PR) sequences. If the generated sequences do not possess required statistical properties and they are still used for encryption, the analysis of encrypted data can reveal information about algorithm's architecture or a part of used key. Furthermore, some attacks can obtain parts of the plain data and this could eventually lead to a break of whole encryption algorithm [13, 14, 15, 16].

The LM is not the only chaotic map used for image encryption, however it can be considered as one of the most popular. Some approaches utilize various modifications of LM or other chaotic maps. Cao et al. described a new custom map in their paper from 2018 [25] that was a result of cascading LM and other chaotic map. Similar solution was proposed by Alawida et al. in 2019 [26]. In both these cases authors review basic properties of the resulting maps by means of their chaotic behavior. However, long-time experience with LM shows that new vulnerabilities could be found out even after multiple years of research [14]. Therefore usage of a newly derived chaotic map without exhaustive analysis of its properties could be viewed as a disadvantage of whole image encryption algorithm.

Other notable approach for enhancement of chaotic behavior is usage of chaotic maps with far too many dimensions. Usage of multidimensional systems could lead to big key spaces, however in general they have a negative impact on computational complexity of the algorithms. Probably the most extreme cases are algorithms by Zhu and Zhu from 2018 [27] or by Sun et al. from 2019 [28]. The first proposal uses six dimensional system, while the second one uses seven dimensional system.

In this paper, we would like to address some of the basic defects regarding usage of LM in the image encryption algorithms. While some are caused by the map itself (or its discretized version), some can be viewed as an inappropriate implementation by the researchers. We would like to propose some fixes for issues that still do not have an efficient solution. As these fixes can affect other techniques used in the field (such as quantization [29]), this paper will briefly describe also some well-known practices in the field of image encryption. Furthermore, in order to provide a complex insight to all necessary blocks of an image encryption algorithm, we will shortly mention also some techniques described in our prior works (e.g. in [24]).

The second goal of this paper is to compare values of commonly used numerical parameters and measure the increase of computational complexity caused by the proposed fixes. As equation of the LM is considered to be quite simple and therefore the map is considered as relatively fast [10], the proposed fixes should not greatly affect the computational complexity. The increase of computational complexity would be measured by applying the proposed fixes to our prior algorithm [24].

The rest of the paper is organized as follows: Section 2 analyzes the behavior and general properties of the LM. Section 3 describes proposed fixes and the way how they interact with other used techniques. Also some other useful methods are mentioned. Section 4 experimentally verifies impact of proposed fixes and contains measurements of numerical parameters and computational complexity. Lastly, Section 5 summarizes the paper and gives a brief overview of possible future work.

2 LOGISTIC MAP AND ITS PROPERTIES

Logistic map (LM) could be characterized as an one dimensional chaotic map with one control parameter r. When $r \in (0, 4)$, each iteration of LM produces an iterate $x_n \in (0, 1)$. Calculation of the first iterate x_1 requires an initial value $x_0 \in (0, 1)$ (also known as an initial point). Equation for the LM can be expressed as (1):

$$x_{n+1} = r \cdot x_n(1 - x_n). \tag{1}$$

Equation (1) shows that successive iterates x_{n+1} use values of previous iterates x_n in their calculations. In order to suppress noticeable relationship between initial value x_0 and successive iterates, some of them are used only for providing other initial value. This concept is known as a *transient period* and its usual length is at least 1 000 iterates. With this length, the first 1 000 iterates are discarded (they are not used for an application of the LM) and the first usable iterate is x_{1001}.

2.1 Bifurcations, Self Similarity and Islands of Stability

The behavior of LM with various values of parameter r can be illustrated by a bifurcation diagram. An example of a bifurcation diagram obtained with 1 000 samples for $r \in (0, 4)$ is shown in Figure 2.

First bifurcation takes place at $r \sim 3$, where predictable trajectory of iterate values divides into two trajectories. Iterate values switch between these trajectories, called orbits based on value of parameter r. Second bifurcation happens at $r =$

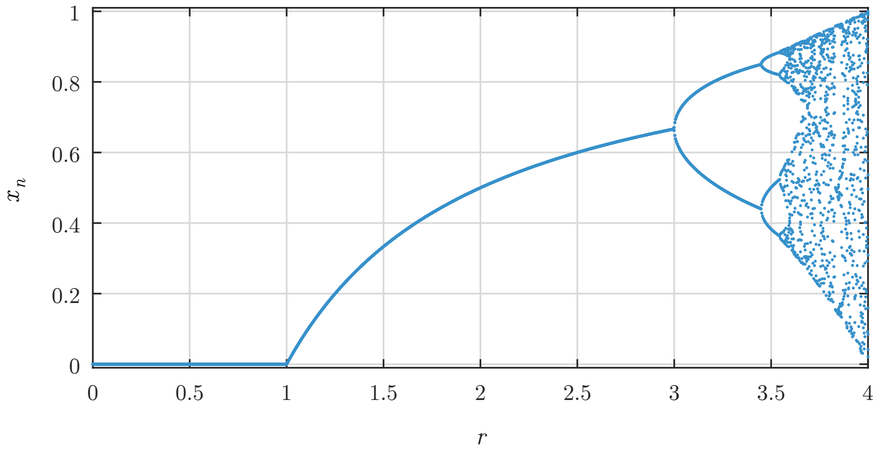


Figure 2. A bifurcation diagram of the logistic map

$1 + \sqrt{6} \sim 3.44949$. The distance between following bifurcations decreases each time by a ratio approximately given by Feigenbaum constant $\delta_f \sim 4.6692$ [30].

It needs to be pointed out that the shape of orbits after second bifurcation resembles shape of orbits after first bifurcation. This is one example of *self similarity* or *fractal behavior* caused by the LM.

The point where $r \sim 3.56995$ is considered as "an onset of chaos" [31]. However, the iterates still do not obtain values from whole interval of $(0, 1)$. Moreover, there are some areas known as *islands of stability* where increasing value of r does not cause multiplication of orbit amount, but the amount of orbits is significantly decreased. The self similarity of LM and an island of stability around $r = 1 + \sqrt{8} \sim 3.82843$ are illustrated in Figure 3 on a magnified part of the bifurcation diagram. The magnified part contains 1 000 samples for $r \in (3.5, 4)$.

2.2 Measurement of Chaotic Behavior

Quantification of chaotic behavior is important for comparing various chaotic maps. The dynamics of chaotic maps are usually expressed by values of maximal Lyapunov exponents (LEs) λ_{max}. As the LM is only one dimensional chaotic map, it does have only one LE $\lambda_{max} = \lambda$.

The basic idea behind computation of LEs investigates relationship of two orbits: the first one starts with an initial point p_0 and initial point for the second orbit denoted as $p_0 + \delta_0$ is achieved by applying small perturbation δ_0 to the point p_0. If orbits for these two initial points diverge from each other over time ($\lambda > 0$), investigated system is considered as chaotic. Otherwise, when the orbits are converging to each other ($\lambda < 0$), it may indicate that investigated system at a certain point tends to have periodic orbit or a fixed point. These two basic situations are displayed in Figure 4. Generally speaking, the higher

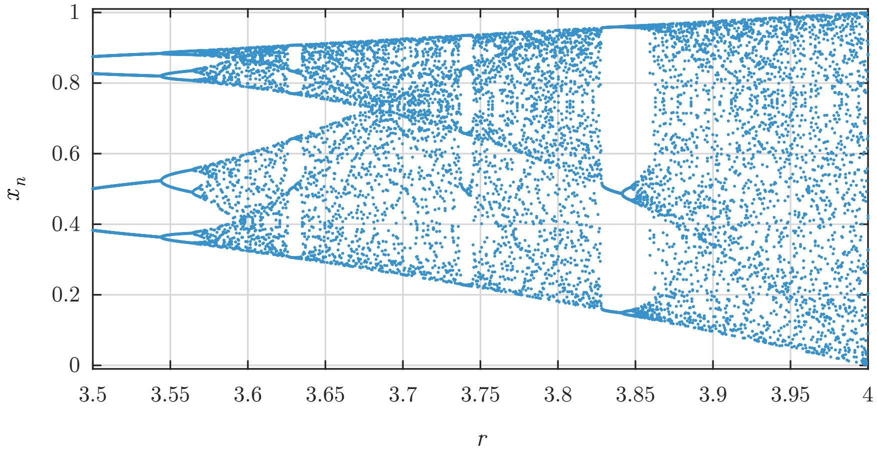


Figure 3. An illustration of self similarity and islands of stability

value of positive λ is, the better chaotic behavior can be expected from chaotic map.

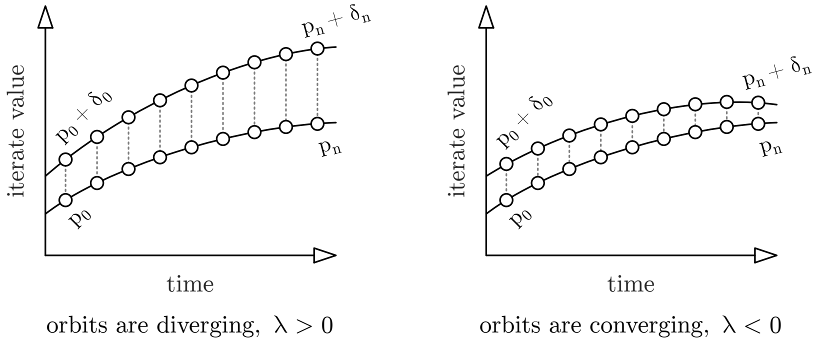


Figure 4. An illustration of diverging and converging orbits

In general, the values of λ can be computed via (2). Please note that practical computations are limited to finite amount of iterates n. Therefore, the resulting value of λ is only an estimation [32]. Commonly used values of n are 1 000 or 10 000 iterates.

$$\lambda = \frac{1}{n} \cdot \left|\frac{\delta_n}{\delta_0}\right| = \frac{1}{n} \cdot \ln|(f^n)'(x_0)| \sim \lim_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} \ln|f'(x_i)| \tag{2}$$

where brackets $|a|$ denote absolute value of a, δ_0 is an initial perturbation – an initial difference between orbits, δ_n is the difference between orbits after n iterations, $\ln(b)$

is a natural logarithm of b, (f^n) is a value of function f at point n, $(f^n)'(x_0)$ is a derivation of f^n with respect to x_0 and i is the iterate index number.

The calculations of λ for the LM substitute arbitrary function f with (1) that could be rearranged and its derivation with respect to x_i equals to $r(1-2x_i)$. Therefore, the estimation of LE for the LM λ_{LM} could be expressed as (3):

$$\lambda_{LM} \sim \lim_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} \ln|r(1-2x_i)|. \tag{3}$$

Please note that the value of λ_{LM} depends on both r and x_i which is a substitution for x_n from (1). While the dependence on r is pretty clear from the bifurcation diagram (see Figure 2 as the amount of bifurcations increases with greater values of r), the dependence on x_n is not that clear. It is caused by the fact that x_n is a result of n iterations of the LM which started with an initial value of $x_0 \in (0, 1)$. Various values of x_0 therefore can affect the behavior of the LM [32].

The obtained estimated values of LE for the LM with $x_0 = 0.5$, $r \in \langle 0, 4\rangle$ and $n = 1\,000$ are displayed in Figure 5. The interval for r is represented by $40\,000$ samples. The red line at $\lambda = 0$ marks the boundary which determines if the behavior of the LM is considered as chaotic or not. Please note that the predictable regions of the bifurcation diagram (see Figures 2 and 3) result in negative values of λ.

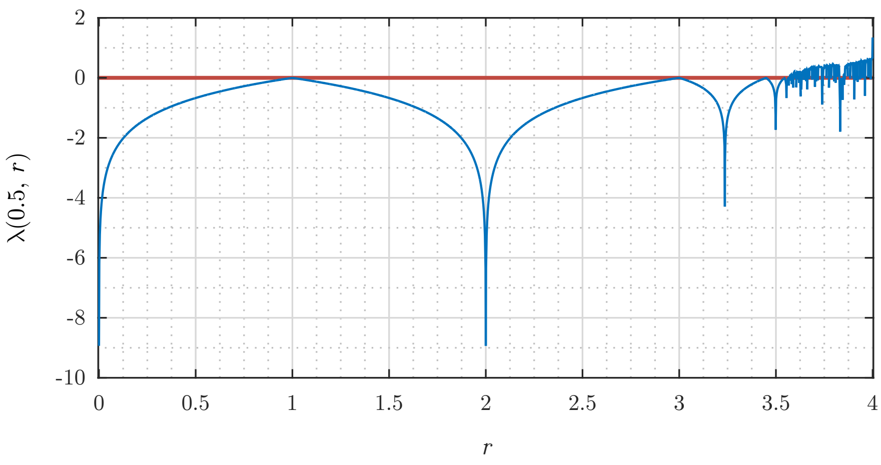


Figure 5. Estimated Lyapunov exponents for $r \in \langle 0, 4\rangle$

A detail of the estimated values of LE for $x_0 = 0.5$, $r \in \langle 3.5, 4\rangle$ and $n = 1\,000$ is shown in Figure 6. The interval investigated in Figure 6 uses finer resolution as it contains $50\,000$ samples (equivalent to $400\,000$ samples for $r \in \langle 0, 4\rangle$). It needs to be pointed out that the spike around island of stability at $r = 1 + \sqrt{8} \sim 3.82843$ resembles spikes visible in Figure 5. This is also caused by self similarity of the

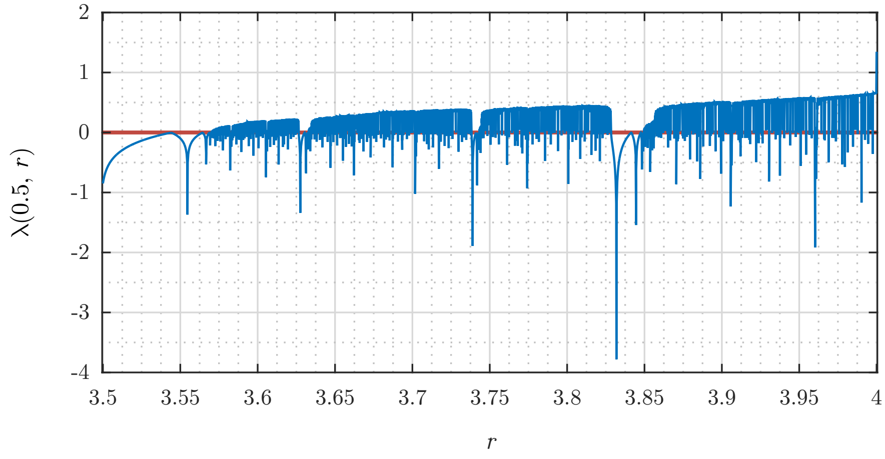


Figure 6. Estimated Lyapunov exponents for $r \in \langle 3.5, 4\rangle$

LM [31]. Therefore usage of finer resolution could reveal some spikes that are present at values of r that could be otherwise not sampled.

Please note that while the equation of the LM (1) is given for $r \in (0,4)$, the plot of estimated LE in Figure 4 uses $r \in \langle 0,4\rangle$. This is due to fact that global minimum and global maximum of the estimated LE values are located on the boundary points of this interval. Some researchers use $\lambda_{max} \sim 1.3447$ which is present for $x_0 = 0.5$ at $r = 4$ with $n = 1\,000$, however this value of r could cause occurrence of a fixed point $x_n = 0$ in systems with finite precision (used in a majority of practical applications).

With usage of the *double precision* data type defined by IEEE 754 standard [33], the decimal part of number is represented by 52 bits. This gives a precision of $\log_{10} 2^{52} \sim 15.6536$ decimal places, therefore double precision data type reliably represents only numbers with 15 or less decimal places. If some number has more decimal places, it is rounded to the closest double precision value.

The largest value of λ for $x_0 = 0.5$, $r \in (0,4)$ and $n = 1\,000$ in the double precision data type is located at $r = 4 - 9 \cdot 10^{-15}$ and its estimated value is approx. 0.67601. This measurement used resolution equivalent to $4 \cdot 10^{16}$ samples for the interval of $r \in \langle 0,4\rangle$, therefore the set of possible x_n has more samples than the amount of numbers in interval $(0,1)$ represented by the double precision data type.

The effect of various initial values x_0 on estimated values of λ is illustrated in Figure 7. This measurement used amount of samples equivalent to $4 \cdot 10^{13}$ for $r \in \langle 0,4\rangle$ and $n = 1\,000$. Please note that the two used values of x_0 cause different behavior of the LM only around a "self similar window". Also the values of λ were estimated with relatively fine resolution, coarser resolutions with less samples could suppress some areas with significant spikes.

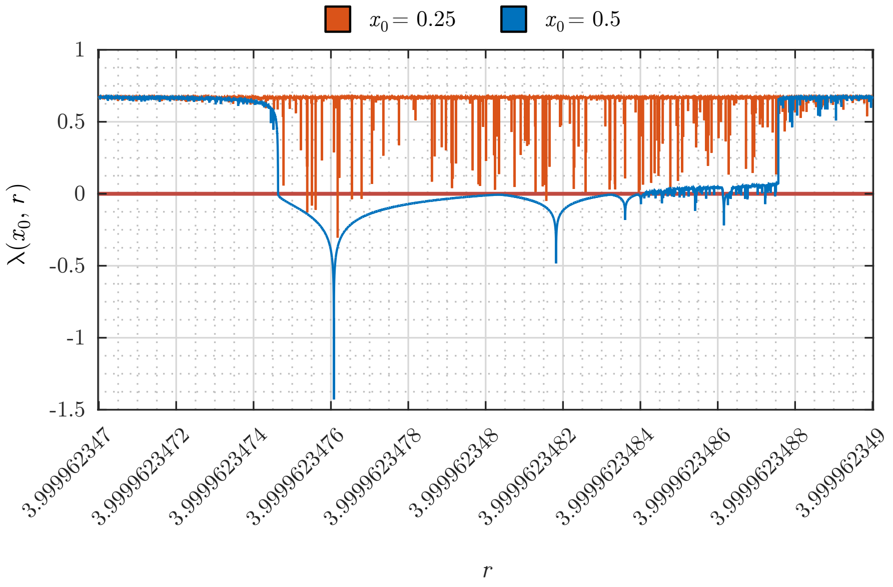


Figure 7. Dependence of estimated Lyapunov exponents on initial value x_0

2.3 Periodic Cycles and Fixed Points

As it was already mentioned, negative values of λ could lead to inappropriate behavior of the LM. One example of this behavior can be demonstrated using the local minimum of λ for $x_0 = 0.5$ from Figure 7. Successive iterates generated with parameter r belonging to this minimum of λ are shown in Table 1.

$x_0 = 0.5,\ r = 3.999962347607499$		
$x_1 = 0.999990586901875$	$x_6 = 0.009607660983953$	$x_{10} = x_1 = 0.999990586901875$
$x_2 = 0.000037651683653$	$x_7 = 0.038061057061643$	$x_{11} = x_2 = 0.000037651683653$
$x_3 = 0.000150599646393$	$x_8 = 0.146448273443029$	$x_{12} = x_3 = 0.000150599646393$
$x_4 = 0.000602302194975$	$x_9 = 0.500000000000973$	$x_{13} = x_4 = 0.000602302194975$
$x_5 = 0.002407735043706$		$x_{14} = x_5 = 0.002407735043706$

Table 1. An example of periodic cycles generated by logistic map

The situation presented in Table 1 is known as a *periodic cycle* of the LM. The presented example is caused by finite precision as a value of iterate x_9 is very similar to the value of x_0 and while their successive iterates are different in systems with higher precision, double precision systems represent them both with the same value that eventually becomes $x_{10} = x_1$. Situations like this are undesirable in cryptographic applications. In this case the length of this periodic cycle is just 9 iterates.

Analytical solution for finding out periodic cycles and their length for arbitrary values of r and x_0 has not been proposed yet. Persohn and Povinelli provided an overview of amount of periodic cycles with certain length, however, only for the single precision data type [10]. Therefore, possible occurrence of periodic cycles in the LM could be indicated only by negative values of λ.

Another issue with the LM is existence of *fixed points* for $x_n \in (0,4)$. The fixed points do not change their values in successive iterations which could be expressed mathematically as $x_n = x_{n+1} = x_{n+2} = \cdots = x_{n+m}$ where m is amount of iterates computed after x_n. The fixed points for the LM could be obtained analytically by setting x_{n+1} to x_n in (1). The modified equation has two solutions: x_n equals either 0 or $1 - \frac{1}{r}$. While the first case is easily suppressed by using $x_n \in (0,4)$, the second case is a bigger issue as value of this fixed point changes depending on used r. The issue with this fixed point can be illustrated by an example provided in Table 2.

$x_0 = 0.25$, $r = 4 - 10^{-15}$, fixed point at $x_n = 1 - \frac{1}{r} \sim 0.75$
$x_1 = 0.75$
$x_2 = x_1 = 0.75$
$x_3 = x_2 = x_1 = 0.75$

Table 2. An illustration of a fixed point of logistic map

An interesting thing about the example presented in Table 2 is that the value of $1 - \frac{1}{4-10^{-15}}$ is considered to be equal to $1 - \frac{1}{4} = 0.75$. In this case the small difference could not be preserved due to finite precision of double precision data type.

The issue with fixed points needs to be solved for enabling proper usage of the LM for encryption. The probability of "hitting" a fixed point among all other iterate values may seem low, but after reaching the value of the fixed point, all successive iterates will have the same value.

2.4 Relation Between Successive Iterates, Their Distribution and the Need for Quantization

As it is visible in (1), the LM is an iterative function as $x_n = f(x_{n-1})$. Equation (1) could be rearranged and there are two solutions for calculating value of previous iterate x_{n-1} from value of current iterate x_n with known value of parameter r (4):

$$x_{n-1} = \frac{1}{2}\left(1 \mp \sqrt{1 - 4\frac{x_n}{r}}\right). \tag{4}$$

The dependence of iterate values on values of the previous iterates could be also illustrated by a Poincaré plot. An example of this plot is shown in Figure 8 where a sequence of 2 000 iterates was computed with $r = 4 - 10^{-15}$ and $x_0 = 0.5$.

Two dimensional Poincaré plots have values of actual iterate x_n on their vertical axis and values of previous iterates x_{n-1} on their horizontal axis. If a horizontal line representing x_n would be drawn over the plot, two perpendicular lines drawn at interceptions with the parabola lead to values of x_{n-1}. Figure 8 shows the example presented in Table 2, where iterate value $x_n = 0.75$ could be obtained from two previous iterate values $x_{n-1,1} = 0.25$ and $x_{n-1,2} = 0.75$ with usage of $r = 4 - 10^{-15}$.

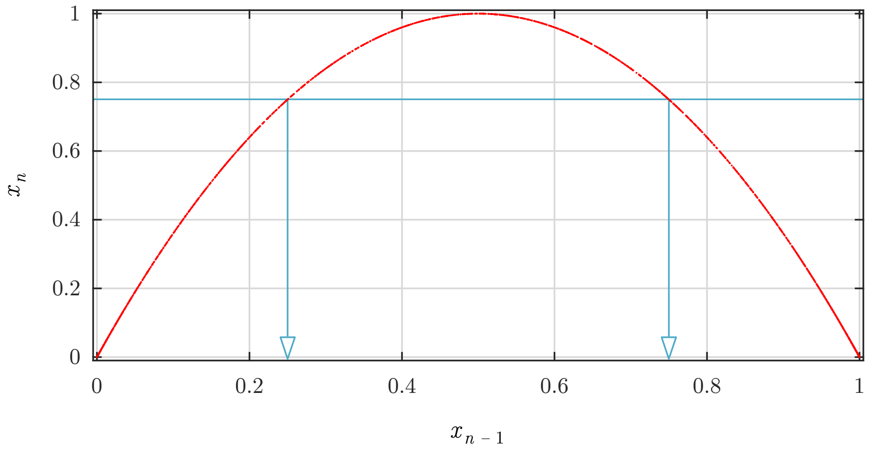


Figure 8. A Poincaré plot showing relation between two successive iterates

While this relation is crucial for computing successive iterates, it needs to be suppressed for usage in cryptographic applications. The possibility of reconstructing previous iterate values from one iterate is investigated by *phase space reconstruction* attacks [34, 35], however none of the proposed attacks could be practically used yet.

Other important feature of a set of computed iterates is their distribution among the interval $(0, 1)$. Ideally, the distribution should be uniform. However, as it was already mentioned and as it is visible in the bifurcation diagrams (see Figures 2 and 3), some values of r do not produce iterate values x_n that cover whole interval $(0, 1)$. An illustration of this case is shown in Figure 9 where blue bins use $r_1 = 3.75$, $x_0 = 0.5$ and transient period of 1 000 iterates in order to generate 10^6 iterates. The second set of bins, red ones were generated by the same settings of the LM except for using $r_2 = 4 - 10^{-15}$. The histogram has 20 bins with equal size.

Figure 9 shows that the first non-zero bin for $r_1 = 3.75$ is located in interval $(0.2, 0.25\rangle$ and the last one is present in interval $(0.9, 0.95\rangle$. The four bins at the beginning and the last bin are empty in the case of r_1 while they are populated for r_2. Also, please note that the bins close to boundaries of the histogram have bigger amount of samples and that the histogram for r_2 is approximately symmetric with respect to its center. These findings could be used for analysis, so they need to be suppressed for enabling usage of the LM for image encryption.

There is also a problem with the usage of computed iterate values in image encryption algorithms. As the iterate values x_n are decimal numbers from interval $(0, 1)$ and the image encryption algorithms process images either as a set of bytes (256 possible integer values) or bits (2 possible integer values), it is useful to perform

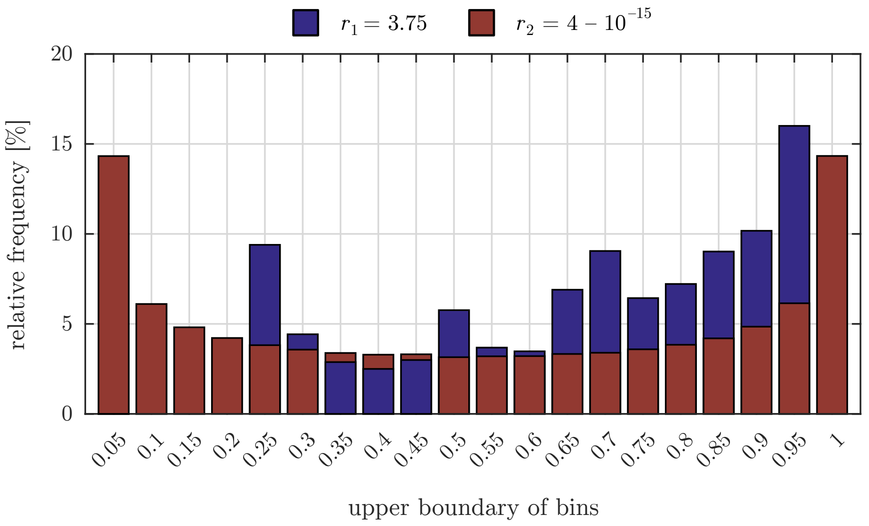


Figure 9. Distribution of iterate values for two different values of parameter r

quantization of the iterate values. Multiple ways of quantization were proposed, some of our prior work even solves the already mentioned issue of relation between successive iterates [24].

3 PROPOSED SOLUTION

Based on the findings from analysis of the LM, we tried to mitigate its drawbacks by using a following set of rules:

- usage of combinations of parameter r and initial value x_0 that have positive λ, so the occurrence of periodic cycles should be suppressed,
- the fixed points should be suppressed by alternating two values of r – each value of r will have its counterpart, denoted as r_c (from "complementary r") which would have a fixed point located at other value of x_n,
- each pair of values of r and r_c should be selected in a way that results in quick divergence form their respective fixed points,
- the elements of resulting PR sequences should not have any distinctive relation between pairs of successive elements and their distribution should be close to uniform (this point has been partially solved in our prior works [24, 35]),
- implementation of these rules should have minimal impact on the computational complexity of image encryption algorithms.

3.1 Suitable Parameters for Logistic Map in Double Precision Data Type

In order to meet given requirements, we chose to use interval of parameter r that obtains positive values of λ and does not have many significant spikes. As most image encryption algorithms use double precision data type, the upper boundary of chosen interval was set the closest it can get to $r = 4$, so the upper boundary was chosen as $4 - 10^{-15}$.

The selection of a lower boundary was motivated by enabling as large interval of parameter r as possible. The size of this interval is important, as key elements are used directly for computation of values of r. Therefore, the larger interval of r enables larger amount of usable keys (known as a key space).

The keys are represented in a hexadecimal or binary form. Conversion between these two forms is simple as one hexadecimal character contains $16 = 2^4$ binary characters. For enabling usage of integer amount of key elements as values of r, also the number of samples in interval of r needs to be a power of 16. The chosen value of number of samples was $16^4 = 65\,536$ as bigger powers of 16 resulted in intervals of r that have significant spikes in plots of their λ. The plot of λ estimated for the resulting interval of $r \in \langle 4 - 65\,536 \cdot 10^{-15}, 4 - 10^{-15} \rangle$, $x_0 = 0.5$ and $n = 1\,000$ is shown in Figure 10. While the analysis was done with resolution equivalent to $4 \cdot 10^{16}$ samples for interval $\langle 0, 4 \rangle$, the plot uses coarser resolution equivalent to $4 \cdot 10^{13}$ samples for the same interval for the sake of better readability.

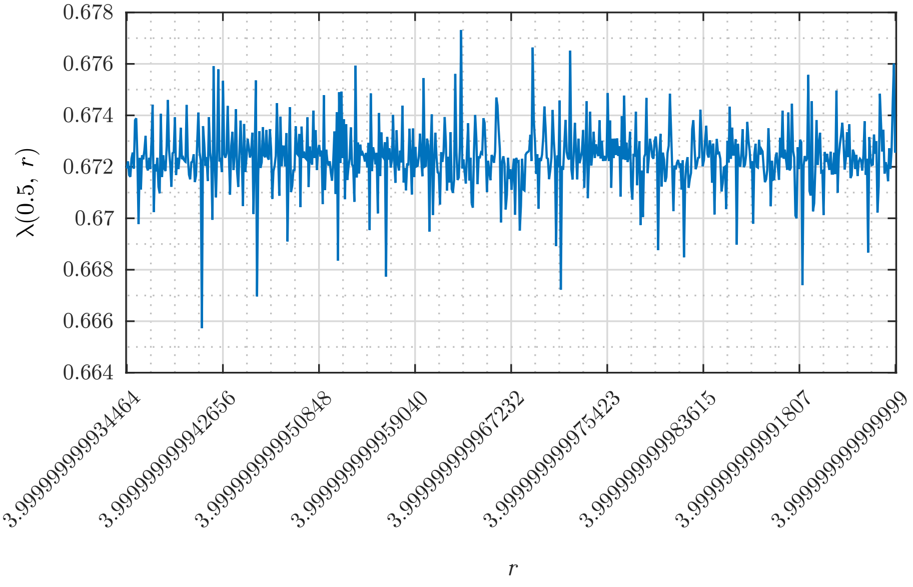


Figure 10. Lyapunov exponents for chosen interval of values for parameter r

The reason behind the choice of x_0 being fixed at 0.5 is the rather small effect on the resulting iterate values x_n. Its impact is present mostly in areas with smaller or negative λ as it was explained in Section 2.2 and especially in Figure 7.

As 4 hexadecimal characters are mapped to 65 536 very similar values of r, the impact of a slightly different key on the resulting iterate values x_n might raise some concern. As it was pointed out in Section 2.1, the LM uses a concept of the transient period. In the case of our proposal, its length is set to 100 iterates for each used value of r. As our solution alternates between r and its complement r_c, the total length of the transient period for each used r is 200 iterates. The resulting values of x_{200} obtained by similar values of r are displayed in Table 3.

	$r_c = r - 10^{-5}$		
used r	$4 - 32\,768 \cdot 10^{-15}$	$4 - 32\,767 \cdot 10^{-15}$	$4 - 32\,766 \cdot 10^{-15}$
x_0	0.5	0.5	0.5
x_1	0.999999999991808	0.999999999991808	0.999999999991809
x_2	0.000000000032768	0.000000000032767	0.000000000032766
x_3	0.000000000131072	0.000000000131068	0.000000000131063
x_4	0.000000000524286	0.000000000524271	0.000000000524250
...	...	...	...
x_{200}	0.845629440355832	0.933763238939059	0.189452959055630

Table 3. Different values of x_{200} obtained with similar values of r

The results presented in Table 3 show that even small differences in value of r result in rather big differences in values of x_{200}. This is a property of chaos, known as a butterfly effect [31].

Image encryption algorithms usually utilize multiple PR sequences for their operations. Therefore, multiple values of r are necessary which brings up the key length to multiples of 4 hexadecimal characters or 16 bits. However, current encryption algorithms utilize key lengths that exceed even 256 bits. In order to enable usage of longer keys (and thus larger key space), our proposal uses multiple values of r and their respective r_c during the transient period. These are denoted as r_i and $r_{c,i}$ and their amount depends on desired size of the key space.

The example of multiple r_i and $r_{c,i}$ used in an image encryption algorithm is illustrated in Table 4. In this case, the algorithm uses 8 values of r_i and their respective $r_{c,i}$ in order to produce 4 final PR sequences. The rule which determines purpose of individual key parts is also known as a *key schedule*.

This example uses 8 values of r_i and their respective $r_{c,i}$. As each r_i is computed from 16 bits of key, the total length of a key for the example is $8 \cdot 16 = 128$ bits. As all of the r_i and $r_{c,i}$ are used during the transient period, even a small change in one part of the key affects all generated PR sequences. This concept is known as *key diffusion* [35, 36]. Total size of the transient period in this case is 1 600 iterates (100 for each of the r_i and $r_{c,i}$).

sequence	indexes i of r_i and $r_{c,i}$ used during the transient period	indexes i of r_i and $r_{c,i}$ used after the transient period
seq_1	1 to 8	1
seq_2	2 to 8, then 1	2
seq_3	3 to 8, then 1 and 2	3
seq_4	4 to 8, then 1 to 3	4

Table 4. An example of a simple key schedule

3.2 Suppression of Periodic Cycles and Fixed Points

As it was mentioned earlier in Section 2.3, the occurrence of periodic cycles could be greatly suppressed by usage of values of r that have positive λ. Analysis of the periodic cycles is quite computationally exhaustive and has been performed only for the single precision data type yet [10]. On the other hand, the values of fixed points could be easily derived for each value of r as it was already shown.

In order to suppress these drawbacks, our proposal alternates between two related parameter values r and r_c. Each odd iteration of the LM uses r, while each even iteration utilizes respective r_c. This approach should be able to solve both mentioned problems.

Because the rules proposed in our approach should have minimal impact on the computational complexity of whole image encryption algorithms, it is important to have a fast way for computing r_c from r. As the chosen interval for value of r is $\langle 4-65\,536\cdot 10^{-15}, 4-10^{-15}\rangle$, we decided to select interval for r_c as $\langle 4-10^{-5}-65\,536\cdot 10^{-15}, 4-10^{-5}-10^{-15}\rangle$. This interval of r_c should provide several advantages: first of all the chosen r_c should have favorable values of λ as they are still close to $r = 4$. The plot of values of λ for the mentioned interval of r_c with $x_0 = 0.5$, $n = 1\,000$ and resolution equivalent to $4\cdot 10^{13}$ samples for interval $\langle 0, 4\rangle$ is shown in Figure 11.

Secondly, the selection of interval for r_c leads to a simple formula for its calculation from r (5):

$$r_c = r - 10^{-5}. \tag{5}$$

Lastly, as the values of r_c and r are different on their fifth decimal place, a slightly adapted quantization block presented in our previous work [24] should be able to ensure quick divergence from any fixed points. An example of fixed points for one pair of parameters r and r_c is shown in Table 5.

parameter	r	r_c
value	$4-10^{-15}$	$4-10^{-5}-10^{-15}$
fixed point at $1-\frac{1}{r}$	0.75	0.749999374998438

Table 5. Fixed points for a pair of parameters r and r_c

While the difference between fixed points displayed in Table 5 may seem negligible, it is big enough to ensure quick divergence from fixed points with usage of

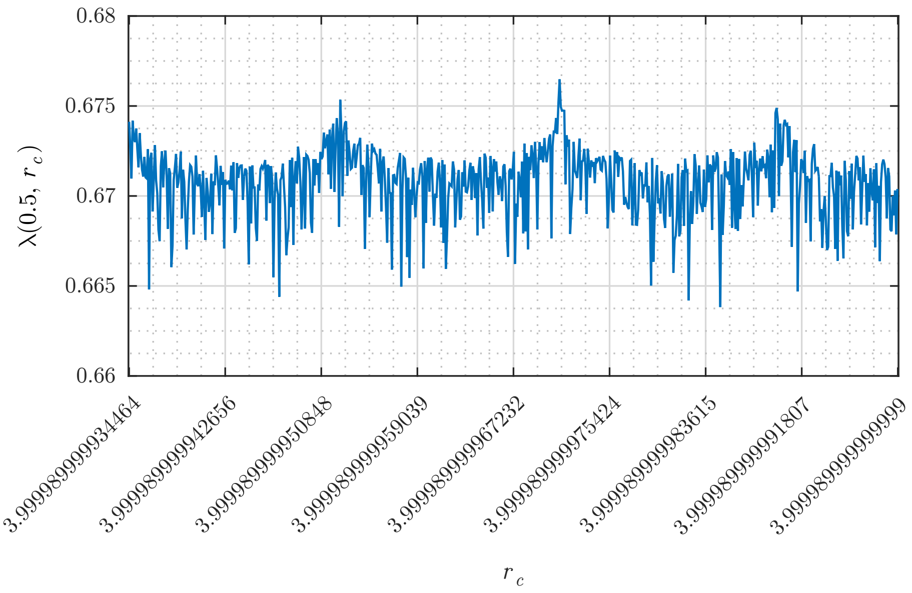


Figure 11. Lyapunov exponents for chosen interval of values for parameter r_c

suitable quantization approach. If value of r_c would be obtained by modification of a higher decimal place, the plot of resulting λ could contain significant spikes and it could also negatively affect the distribution of iterate values after their quantization.

3.3 Quantization and Successive Iterate Relation

In our previous work, it was found out that the non-uniform distribution of iterate values x_n is caused mainly by higher amount of iterate values that are close to the boundaries of interval $(0, 1)$ [24, 35]. This could be easily fixed by dismissing several decimal places of iterate values after whole sequence of iterates has been computed.

A quantization technique that dismisses first four decimal places of each iterate value x_n was proposed in [35]. Its equation could be expressed as (6):

$$x'_n = \left\lfloor (max_{val} + 1) \cdot \left(10^4 \cdot x_n \pmod 1\right) \right\rfloor \tag{6}$$

where x'_n is a sequence of quantized iterate values, brackets $\lfloor a \rfloor$ denote the greatest integer less than or equal to a, n is a sequential number of an iterate and max_{val} is maximal allowed value after quantization.

The effects of quantization by (6) are shown in Figures 12 and 13. The sequences for these plots were made in the same way as for Figures 8 and 9. Figure 13 uses only one value of $r = 4 - 10^{-15}$ so please compare it only with red bars from Figure 9.

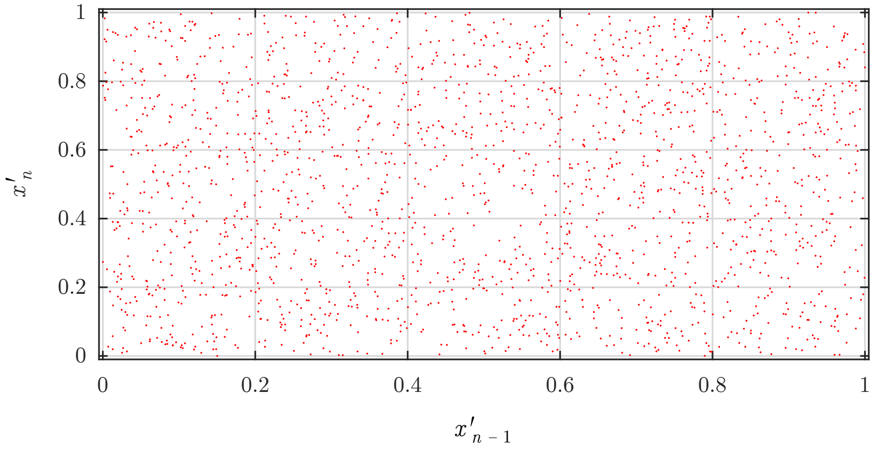


Figure 12. A Poincaré plot of iterates after quantization

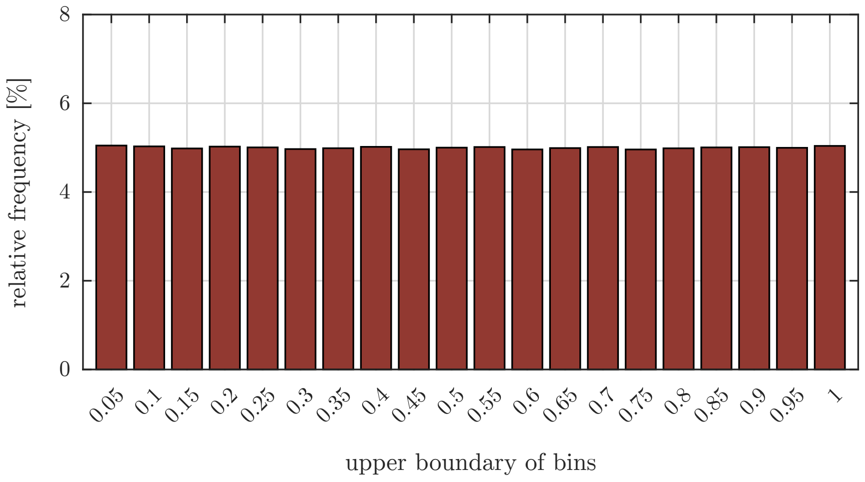


Figure 13. Distribution of iterate values after quantization

Please note that the quantization by using (6) suppresses relations between successive iterates. While the relations can be found in a sequence of computed iterates, the removal of the first four decimal places by quantization results in a more uniform distribution of the points shown by a Poincaré plot in Figure 12. Also, the amount of points on the horizontal lines belonging to x'_n is not clear so any attempts to obtain x'_{n-1} from x'_n are even harder.

The histogram of values of quantized iterates displayed in Figure 13 shows that the distribution is much more uniform than in the previous case (see red bars on

Figure 9). Removal of more decimal places did not yield better results in terms of more uniform distribution of iterate values [35, 24].

As it was mentioned, the quantization technique greatly affects the speed of divergence from fixed points. An example is presented in Table 6. Please note that the same values of x_0 and r were used in Table 2. However the previous case did not alternate between r and r_c and therefore ended in an occurrence of a fixed point.

$r = 4 - 10^{-15}$, fixed point at $x_n = 0.75$
$r_c = 4 - 10^{-5} - 10^{-15}$, fixed point at $x_n = 0.749999374998438$

iterate	value	value after removal of first four decimal places	value after quantization by (6), $max_{val} = 255$
x_0	0.75	–	–
x_1	0.75	0.999999999998181	255
x_2	0.749998125000000	0.981250000002547	251
x_3	0.750003749985937	0.037499859367927	9
x_4	0.749990624990627	0.906249906268386	231
x_5	0.750018749667183	0.187496671829649	47
x_6	0.749960624353186	0.606243531856308	155
x_7	0.750078745091862	0.787450918623108	201
...	...	...	...
x_{50}	0.202236261140394	0.362611403942537	92
x_{51}	0.645347023281394	0.470232813938310	120
x_{52}	0.915494682550523	0.946825505234301	242
x_{53}	0.309456675088959	0.566750889588548	145
...	...	...	...

Table 6. An illustration of quick divergence from fixed point

Values reported in Table 6 show that the first usage of r_c which has a different fixed point introduces a slight divergence from the fixed point for r. Also, if the value of computed iterate would be equal to the fixed point for r_c, the alternation to r would help to provide a small perturbation which would result in a divergence.

There are two interesting facts about values presented in Table 6. Firstly, the value of $x_1 = 0.75$ after removal of the first 4 decimal places becomes ~ 0.99999. This is due to fact that x_1 is represented by its closest possible representation in double precision data type which is actually 0.74999999999999978. The first four decimal places of this number are removed and the other places are shifted towards the decimal point. The last two decimal places (78) are changed in order to meet the closest possible number after the quantization in double precision data type (8181).

The second interesting fact is that the values of x_1, x_2 and x_3 are close to the boundaries of set $\{0, 1, \ldots, 255\}$. However, after rather small amount of iterates, the distribution becomes more uniform.

4 EXPERIMENTAL RESULTS

Usage of the proposed rules should result in a more suitable behavior of the LM (1) and therefore it can positively affect the performance of whole image encryption algorithms. On the other hand more complex computations can raise computational complexity. In order to investigate these assumptions, the proposed set of rules was applied to one of our prior image encryption algorithms published in [24]. The mentioned algorithm was already highly optimized for achieving good performance, so the resulting values of numerical parameters should clearly show the effect of the proposed set of rules.

All performed measurements were done on a PC with 2.5 GHz CPU, 12 GBs of RAM in computing environment MATLAB R2015a on Windows 10 OS. A set of experimental plain images *lena*, *lenaG* and *peppersG* is shown in Figure 14. Their parameters are described in Table 7. A set of experimental keys K_1 and K_2 are presented in Table 8. The first one was derived from binary representation of decimal part of number π. The second key has a minimal difference, as the sixth hexadecimal character is changed from 0 to 1.

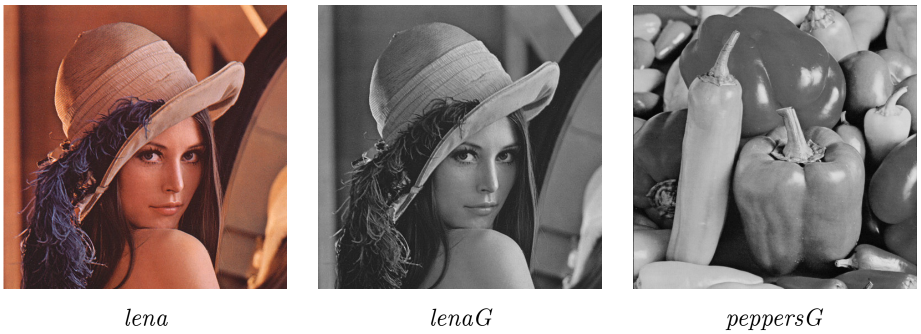


Figure 14. A set of experimental plain images

image	*lena*	*lenaG*	*peppersG*
height [px]	512	512	512
width [px]	512	512	512
color depth [bits/px]	24	8	8

Table 7. Parameters of used plain images

key	value
K_1	`0×` `C4 EB 50 BC 0E C5 EB 50 BC 0E C5 EB 50 BC 0E C5`
K_2	`0×` `C4 EB 51 BC 0E C5 EB 50 BC 0E C5 EB 50 BC 0E C5`

Table 8. A set of experimental keys

4.1 Brief Description of Used Algorithm

The image encryption algorithm proposed in [24] processes images with arbitrary resolution higher than 16×16 pixels. This restriction is caused by Mojette transform (MoT) that is used during a plaintext related chaining stage. The color depths of input images could be either 8 or 24 bits per pixel. Used key is 128 bits long, entered in a hexadecimal form and it is used for generation of six PR sequences by means of the LM (1).

Described image encryption algorithm has 5 stages. The first and the last stage combine image pixel intensities with PR sequences generated by the LM (1). Since these combinations protect the other stages against attacks, the security of whole examined image encryption algorithm relies heavily on the first and last stage.

The second stage rearranges image pixels in order to suppress correlation between adjacent image pixel intensities. The rearrangement is carried out by cyclic shifts in individual columns of the processed image followed by cyclic shifts in the rows of the image.

Stages three and four introduce dependencies between pixel intensities in order to increase sensitivity to small differences between various plain images. While the third stage scans and processes the image in all four possible directions for spreading the differences, the fourth stage uses pixel intensities as parameters for the MoT. The previously loaded pixel intensities choose values that are combined with actually processed pixel intensities.

The decryption algorithm utilizes analogous stages in a reversed order. More details about this solution could be found in [24].

4.2 Modifications of Used Algorithm

Due to previous optimization of the used algorithm a rather small amount of modifications was necessary for adhering to the proposed set of rules. First of all, the concept of shifting between values of r and r_c was introduced. The intervals for these parameters were set as it was recommended in Section 3.1: $r \in \langle 4 - 65\,536 \cdot 10^{-15}, 4 - 10^{-15} \rangle$ and $r_c \in \langle 4 - 10^{-5} - 65\,536 \cdot 10^{-15}, 4 - 10^{-5} - 10^{-15} \rangle$.

Secondly, the whole key schedule was reworked as the original approach used parts of the key with 8 hexadecimal characters. This was not longer possible as the 65 536 possible values of r or r_c are represented only by 4 hexadecimal characters. The eight values of r and their respective values of r_c are computed by (7):

$$\begin{aligned} r_i &= 4 + 10^{-15} \cdot (K_i - 65\,536), \\ r_{c,i} &= r_i - 10^{-5} \end{aligned} \tag{7}$$

where K_i represents a decimal form of i^{th} four hexadecimal character group from key K, $i = 1, 2, 3, \ldots, 8$. The conversion from a hexadecimal to a decimal form uses big endian ordering scheme.

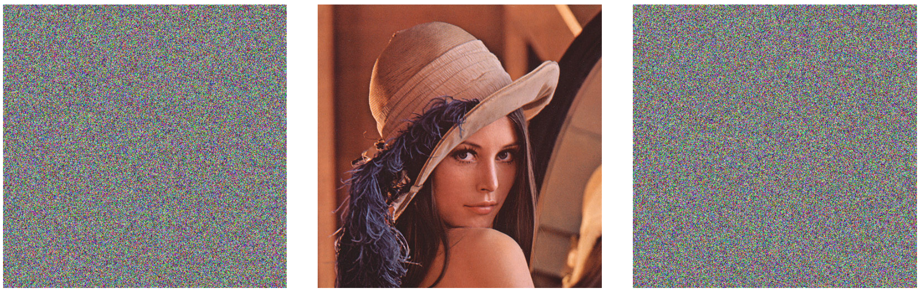
The key schedule used in the modified algorithm is presented in Table 9. Each parameter is used for 100 iterations in the transient period (each pair of r ad r_c therefore for 200 iterations). The total amount of iterations during the transient period is extended from previously used 1 000 iterates to 1 600.

sequence	indexes i of r_i and $r_{c,i}$ used during the transient period	indexes i of r_i and $r_{c,i}$ used after the transient period
seq_1	1 to 8	1
seq_2	2 to 8, then 1	2
seq_3	3 to 8, then 1 and 2	3
seq_4	4 to 8, then 1 to 3	4
seq_5	5 to 8, then 1 to 4	5
seq_6	6 to 8, then 1 to 5	6

Table 9. Key schedule used in the modified algorithm

Finally, the quantization does not shift decimal places of iterates by five places anymore. In order to provide both quick divergence from fixed points and suppression of relations between successive iterations, the quantization shifts decimal places by four places by (6) and r_c is set in a way that it is different from r on fifth decimal place. These steps were selected because of suitable values of λ for r and r_c with x_0 fixed at 0.5. More details on this issue were presented in Section 2.2.

A simple verification that the proposed set of rules did not significantly harm the basic features of the original image encryption algorithm can be performed through encryption and decryption an image. The example presented in Figure 15 used plain image *lena* and key K_1. The first decryption used correct key and its result is the same as the plain image. However second decryption deliberately used a wrong key (K_2) and the decryption results in an image similar to noise.



encrypted by K_1 — encrypted by K_1, decrypted by K_1 — encrypted by K_1, decrypted by K_2

Figure 15. Preserved key sensitivity of the modified algorithm

4.3 Used Numerical Parameters

There are several commonly utilized numerical parameters that can be used for measuring performance of the image encryption algorithms. This paper utilizes values of correlation coefficients ρ, entropy H, Number of Pixel Change Ratio (NPCR) and Unified Average Changing Intensity (UACI). The last two parameters were described by Wu et al. in [37].

In addition to these parameters, this paper also utilizes a measure of computational complexity known as number of cycles necessary for an operation (either encryption or decryption of an image). This measure takes into account resolution and color depth of the testing images and also a configuration of used PC.

Values of *correlation coefficients* ρ are computed separately for each color plane in three directions – horizontally (denoted as ρ_h), vertically (ρ_v) and diagonally (ρ_d). The computation of ρ is usually done on a finite amount of randomly chosen pixel pairs. Each pixel pair contains intensities of two image pixels that are adjacent in a chosen direction. In general, ρ are computed by using (8):

$$\rho = \frac{\sum_{pp=1}^{num_{pp}} \left(vec_1(pp) - \overline{vec_1}\right) \cdot \left(vec_2(pp) - \overline{vec_2}\right)}{\sqrt{\sum_{pp=1}^{num_{pp}} \left(vec_1(pp) - \overline{vec_1}\right)^2 \cdot \sum_{pp=1}^{num_{pp}} \left(vec_2(pp) - \overline{vec_2}\right)^2}} \; [–] \tag{8}$$

where $pp = 1, 2, \ldots, num_{pp}$ is an index of pixel pair, num_{pp} is total amount of pixel pairs (usually selected as 1 000), vectors vec_1 and vec_2 represent intensities of pixels from pixel pairs and $\overline{vec}$ denotes arithmetic mean of vector vec.

The entropy H as it was described by Shannon [38] can be viewed as a randomness measure of a information source. It is computed individually for each color plane of an image by applying (9):

$$H = -\sum_{in=0}^{2^L-1} p(in) \cdot \log_2\left(p(in)\right) \; \text{[bits/px]} \tag{9}$$

where L is a color depth of investigated color plane, in denotes intensity of image pixel and $p(in)$ stands for a probability of occurrence of pixel with intensity in. The ideal value of entropy H is close to the color depth of investigated color plane.

Each computation of NPCR and UACI utilizes two encrypted images E_1 and E_2. These were made by encryption with the same algorithm and the same key from plain images P_1 and P_2. The first plain image P_1 was arbitrarily chosen, while the second plain image P_2 is a copy of P_1 with a difference in an intensity of one image pixel. Furthermore, the size of this difference is minimal (only one intensity level). Values of NPCR and UACI therefore measure the differences done by encryption of two almost identical plain images.

The location of the difference introduced in plain image P_2 is usually randomly chosen. In order to suppress the impact of certain locations on the resulting values, the values of NPCR and UACI are presented as a mean of larger set of measurements (usually mean of 100 computed values).

Number of Pixel Change Ratio (NPCR) is calculated individually for each color plane of the investigated image by using (10):

$$NPCR = \frac{100}{h \cdot w} \sum_{l=1}^{h} \sum_{k=1}^{w} Diff_{mat}(l,k) \ [\%] \tag{10}$$

where h and w represent height and width of images E_1 and E_2, l and k are line and column indexes and $Diff_{mat}$ is a difference matrix, $Diff_{mat}(l,k) = 1$ if $E_1(l,k) \neq E_2(l,k)$, $Diff_{mat}(l,k) = 0$ otherwise.

Unified Average Changing Intensity (UACI) is also computed individually for each color plane of the tested image via (11):

$$UACI = \frac{100}{h \cdot w} \sum_{l=1}^{h} \sum_{k=1}^{w} \frac{|E_1(l,k) - E_2(l,k)|}{2^L - 1} \ [\%] \tag{11}$$

where brackets $|a|$ denote absolute value of a and L is a color depth of the investigated color plane.

Please note that while NPCR only sums amount of pixels with different intensities after encryption of P_1 and P_2, UACI also takes into account the size of individual differences. Also, the proposal of NPCR and UACI by Wu et al. [37] mentions expected values of the two parameters. The expected values are presented as intervals and they depend on a resolution of used images. If a computed value of NPCR or UACI belongs to the interval of the expected value, the resulting encrypted image can be considered as robust against differential attacks with certain confidence (determined by significance level α) [37].

For one of the most used significance levels at $\alpha = 0.001$ which results in confidence of 99.9 %, the intervals of NPCR and UACI for an image with resolution of 512×512 pixels are $(99.5717\,\%, 100\,\%\rangle$ and $(33.1594\,\%, 33.7677\,\%)$, respectively.

The *computational complexity* of the image encryption algorithms can be measured via time t_{oper} necessary for completing an operation (encryption or decryption) on an image. Usually, a mean of 100 repeated measurements is used for suppressing effect of some faster or slower times (e.g. due to other processes running on a testing PC). These means are denoted as t_{enc} for encryption times and t_{dec} for decryption times in the following text.

Configuration of the testing PC is taken into account by computing number of CPU cycles that are necessary for an operation with one byte of image data. The equation for this measure is given as (12):

$$cyc_{oper} = \frac{f_{CPU} \cdot t_{oper} \cdot 2^3}{h \cdot w \cdot d} \ [\text{cycles/B}] \tag{12}$$

where f_{CPU} is a clock frequency of CPU in the testing PC, t_{oper} is the operation (encryption or decryption) time in seconds, h and w stand for height and width of the used image, d is its color depth given in bits per pixel and a constant of

2^3 represents amount of bits in a byte. Please note that most image encryption algorithms utilize computational environments that use only one CPU core.

4.4 Measured Values and Discussion

The measured values of correlation coefficients ρ, entropy H, NPCR and UACI for plain images *lena*, *lenaG* and *peppersG* encrypted by keys K_1 and K_2 are included in Table 10. Please bear in mind, that this comparison is done only for investigating impact of the proposed set of rules, therefore the values for plain images (not encrypted) are not presented. The letters R, G and B stand for red, green and blue color plane. Symbol '–' denotes that the presented results are obtained for the only color plane of grayscale images.

image and color plane		key	ρ_h [–]	ρ_v [–]	ρ_d [–]	H [bits/px]	NPCR [%]	UACI [%]
algorithm from reference [24]								
lena	R	K_1	0.0004	0.0013	0.0005	7.9993	99.6101	33.4738
	G		−0.001	−0.0021	−0.0037	7.9994	99.6109	33.4742
	B		−0.0032	−0.002	0.0017	7.9992	99.6103	33.4746
	R	K_2	0.0033	0.0026	−0.001	7.9994	99.6105	33.4742
	G		0.0006	0.0015	−0.0016	7.9993	99.6113	33.4749
	B		0.0016	0.0013	0.003	7.9993	99.6101	33.4743
lenaG	–	K_1	0.0008	0.0005	−0.0004	7.9992	99.61	33.4714
		K_2	0.0015	0.001	−0.0012	7.9993	99.6099	33.4725
peppersG	–	K_1	0.0051	0.0007	−0.0001	7.9991	99.6106	33.4721
		K_2	0.0019	−0.0011	−0.0008	7.9991	99.6121	33.4729
algorithm from [24] improved by the proposed solution								
lena	R	K_1	0.0019	0.0015	0.0019	7.9993	99.6191	33.4908
	G		−0.0005	0.0021	0.0034	7.9993	99.6204	33.49
	B		−0.0018	0.0005	−0.0004	7.9993	99.6183	33.4873
	R	K_2	−0.0001	−0.001	−0.0006	7.9992	99.6202	33.4884
	G		0.0009	−0.0001	−0.0038	7.9993	99.6198	33.4894
	B		0.0001	0.0007	−0.0017	7.9992	99.621	33.4872
lenaG	–	K_1	0.0021	0.0006	0.0019	7.9992	99.6206	33.4867
		K_2	−0.0032	0.0013	0.0002	7.9993	99.6196	33.4904
peppersG	–	K_1	0.0003	−0.0001	−0.0008	7.9994	99.6187	33.4856
		K_2	−0.0001	−0.0034	0.0027	7.9993	99.618	33.4892

Table 10. Measured values of ρ, H, NPCR and UACI

The results presented in Table 10 show that difference in values of correlation coefficients ρ (computed from 1000 pairs of adjacent pixel intensities) did not change much by application of the proposed set of rules. Also, the change in values of entropy H might be viewed as negligible. This could be caused by a good performance of the original algorithm as it was discussed in the paper [24].

On the other hand, the values of NPCR and UACI were slightly improved. Please bear in mind, that there is only a slight improvement, but it could greatly help in the case of a differential attack. Also, since the proposed set of rules mitigates vulnerabilities of the LM, the possibility of other attacks should be suppressed. On top of that, all values of NPCR and UACI for both the original algorithm and its improved version belong to intervals of the expected values [37].

The increase of computational complexity is illustrated by values included in Table 11. It is clearly seen that application of the proposed set of rules has only a small impact on computational complexity of the whole image encryption algorithm. Furthermore, the used algorithm was highly optimized so any new modifications could result in a visible increase of computational complexity.

image	key	t_{enc} [ms]	increase [%]	t_{dec} [ms]	increase [%]	cyc_{enc} [cycles/B]	cyc_{dec} [cycles/B]
algorithm from reference [24]							
lena	K_1	436.2245	–	420.5649	–	1 386.72	1 336.94
	K_2	436.3845		420.3934		1 387.23	1 336.39
lenaG	K_1	126.2499	–	122.7206	–	1 204.01	1 170.35
	K_2	126.9473		122.6332		1 210.66	1 169.52
peppersG	K_1	127.1529	–	122.8078	–	1 212.62	1 171.19
	K_2	126.8485		122.7695		1 209.72	1 170.82
algorithm from [24] improved by the proposed solution							
lena	K_1	448.7342	2.87	429.9181	2.22	1 426.49	1 366.67
	K_2	447.9295	2.65	431.0329	2.53	1 423.93	1 370.22
lenaG	K_1	129.5253	2.59	126.5223	3.1	1 235.25	1 206.61
	K_2	130.9827	3.18	125.8389	2.61	1 249.15	1 200.09
peppersG	K_1	130.8076	2.87	125.5048	2.2	1 247.48	1 196.91
	K_2	129.9531	2.45	125.7442	2.42	1 239.33	1 199.19

Table 11. Measured values of computational complexity

The measured increase of the computational complexity was ranging approximately from 2.2 % to 3.2 % (with the worst case of almost 13 ms for a true color image with a resolution of 512 × 512 pixels). This is a relatively interesting result for as complex changes as those described in Section 4.2. This finding can also imply that well-tailored fixes or bigger patches could further improve the behavior of simpler chaotic maps like the LM.

4.5 Comparison with Some Other Proposals

The impact of the proposed set of rules on the LM could be pointed out by a comparison of numerical results with some other approaches. Some of the relatively new algorithms were selected for the comparison, with a strong emphasis on algorithms that use multidimensional systems. Usage of these systems in image encryption is interesting as some researchers directly relate the robustness of whole image encryption algorithm to behavior of its chaotic system given by its LEs. However, the finite precision and some bad implementations can degrade their results. Therefore, we would like to compare results obtained by our well-tuned algorithm based on a simple chaotic map with the numerical results and computational complexity of image encryption algorithms that utilize far more complex chaotic systems.

An approach using a cascade of two chaotic maps was described by Cao et al. in 2018 [25]. The authors discuss that in certain interval, the parameter of used chaotic system results in λ close to value of 6. A similar solution was proposed by Alawida et al. in 2019 [26], where resulting value of λ is close to 2. Moreover, these researchers performed a brief investigation regarding relation of successive iterates.

A paper by Sun et al. [28] from 2019 describes an algorithm that uses seven dimensional chaotic system. Also, the computational complexity of this algorithm is increased by usage of hash function for introducing a plaintext related operation.

Liu et al. published a paper [39] in 2020 that characterizes their algorithm as "fast". However they still use the combination of two relatively complex chaotic maps that negatively affect the resulting computational complexity.

The comparison of numerical results including the computational complexity given in amount of cycles necessary for an encryption of one byte of image data cyc_{enc} is shown in Table 12. The value in *italics* marks that it was the best among all compared approaches and value '–' stands for not reported measurement.

approach	ρ_h [–]	ρ_v [–]	ρ_d [–]	H [bits/px]	NPCR [%]	UACI [%]	cyc_{enc} [cycles/B]
plain image *lenaG*							
proposed	0.0021	*0.0006*	0.0019	*7.9992*	99.6206	33.4867	*1 235.25*
ref. [25]	0.0019	0.0012	*0.0009*	7.9973	99.6096	33.4574	9 402.59
ref. [26]	−0.0017	−0.0084	−0.0019	7.9975	99.62	*33.505*	24 644.78
ref. [39]	0.0106	−0.0012	*0.0009*	–	*99.6216*	33.4994	3 484.18
plain image *peppersG*							
proposed	*0.0003*	−0.0001	−0.0008	*7.9994*	99.6187	33.4856	*1 247.48*
ref. [26]	0.0024	−0.0131	*0.0002*	7.997	99.617	33.391	–
ref. [28]	0.0017	0.0012	−0.0128	7.9978	*99.62*	*33.63*	27 694.7

Table 12. A comparison of the obtained numerical results

Obtained values of correlation coefficients ρ show just little differences between compared approaches. On the other hand, the results for entropy H are much more different. It could be noticed that the proposed solution achieves the best values of H, very close to the theoretical boundary of 8 bits per pixel.

Probably the most interesting values for comparison are those of NPCR and UACI. While [25] has quite low values of both NPCR and UACI, the other three algorithms that used plain image *lenaG* have similar values of NPCR. The solution described in this paper obtains the lowest value of UACI among these three approaches even after the value was increased by applying the proposed set of rules.

Results for encryption algorithms that utilized plain image *peppersG* show an example of an unbalanced performance. While the value of NPCR obtained by [26] is quite high, the value of UACI is the lowest among all reported values. Even more interesting is a fact that the same algorithm achieved best value of UACI for previous plain image *lenaG*. The solution proposed in this paper may have far worse value of UACI than [28], however its performance is uniform over both plain images used for a comparison. Values of UACI reported in [28] for other plain images (*lenaG* was not used) varied from 33.36 % to 33.63 %.

The measurement of amount of CPU cycles necessary for encryption of one byte of image data shows that our proposal is clearly the fastest one among the discussed algorithms. A rather interesting contrast is shown by values of cyc_{enc} for quite similar approaches [25] and [26]. The first design was much more optimized as it used more than 2.5 times less CPU cycles to complete the encryption than the second approach. Also, the second fastest solution from [39] took almost three times more CPU cycles than the proposed solution. The slowest algorithm [28] is negatively affected mainly by high complexity of used operations.

5 CONCLUSIONS

This paper described and extensively analyzed some of the vulnerabilities related to the usage of logistic map in image encryption algorithms. Some of the drawbacks were identified and suppressed already in the past, however, some others are present also in the newer proposals. This paper namely dealt with the usage of parameter values that result in a predictable behavior of the map, its periodic cycles and fixed points.

These disadvantages are mitigated by a proposed set of rules. One of the rules describes a way to choose a pair of suitable parameter values. Usage of appropriate parameter values in a data type with finite precision should prevent occurrence of periodic cycles. Also the speed of divergence from a fixed point was investigated. A rather quick divergence was reached by a combination of alternating parameter values and a modified quantization technique.

In order to verify the mentioned assumptions, the proposed set of rules was applied on an algorithm from our prior work. Obtained numerical results show that the usage of the rules helped to enhance the chaotic behavior of a rather simple logistic map to the level that the image encryption algorithm based on it achieves almost the same values of numerical parameters as algorithms based on more complex chaotic systems. Moreover, the logistic map still preserves its advantages such as a relatively simple usage leading to lower computational complexity. Also, as the properties of the logistic map are quite well studied, the probability of finding some new vulnerabilities is smaller than in newer, more complex chaotic systems.

Because the usage of the proposed set of rules did not significantly increase the computational complexity (the results show an increase ranging approx. from 2.2 % to 3.2 %), it gives a possibility for even more work in this area. However, it needs to be done with great care, as there are many examples of image encryption algorithms that have a higher computational complexity, as they rely on computationally exhaustive techniques such as cascades of chaotic maps, their coupling or hash functions.

Acknowledgment

This work was supported by research grants APVV-17-0208 and VEGA 1/0584/20.

REFERENCES

[1] Advanced Encryption Standard (AES). Federal Information Processing Publication 197 (FIPS 197), 2001, doi: 10.6028/NIST.FIPS.197.

[2] Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A, National Institute of Standards and Technology, 2001, doi: 10.6028/NIST.SP.800-38A.

[3] Gueron, S.: Intel® Advanced Encryption Standard (AES) New Instructions Set. Intel, White Paper, 2010. Available at: `https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf`.

[4] Chen, X.—Hu, C. J.: Adaptive Medical Image Encryption Algorithm Based on Multiple Chaotic Mapping. Saudi Journal of Biological Sciences, Vol. 24, 2017, No. 8, pp. 1821–1827, doi: 10.1016/j.sjbs.2017.11.023.

[5] Oravec, J.—Turán, J.: Substitution Steganography with Security Improved by Chaotic Image Encryption. Proceedings of 2017 IEEE 14th International Scientific Conference on Informatics (INFORMATICS 2017), Poprad, Slovakia, 2017, pp. 284–288, doi: 10.1109/INFORMATICS.2017.8327261.

[6] Abundiz-Pérez, F.—Cruz-Hernández, C.—Murillo-Escobar, M. A.—López-Gutiérrez, R. M.—Arellano-Delgado, A.: A Fingerprint Image Encryption Scheme Based on Hyperchaotic Rössler Map. Mathematical Problems in Engineering, Vol. 2016, 2016, Art. No. 2670494, doi: 10.1155/2016/2670494.

[7] Matthews, R.: On the Derivation of a "Chaotic" Encryption Algorithm. Cryptologia, Vol. 13, 1989, No. 1, pp. 29–42, doi: 10.1080/0161-118991863745.

[8] May, R. M.: Simple Mathematical Models with Very Complicated Dynamics. Nature, Vol. 261, 1976, No. 5560, pp. 459–467, doi: 10.1038/261459a0.

[9] Phatak, S. C.—Rao, S. S.: Logistic Map: A Possible Random-Number Generator. Physical Review E, Vol. 51, 1995, No. 4, pp. 3670–3678, doi: 10.1103/PhysRevE.51.3670.

[10] Persohn, K. J.—Povinelli, R. J.: Analyzing Logistic Map Pseudorandom Number Generators for Periodicity Induced by Finite Precision Floating-Point Representation. Chaos, Solitons and Fractals, Vol. 45, 2012, No. 3, pp. 238–245, doi: 10.1016/j.chaos.2011.12.006.

[11] Özkaynak, F.: A Novel Method to Improve the Performance of Chaos Based Evolutionary Algorithms. Optik, Vol. 126, 2015, No. 24, pp. 5434–5438, doi: 10.1016/j.ijleo.2015.09.098.

[12] Arroyo, D.—Alvarez, G.—Fernandez, V.: On the Inadequacy of the Logistic Map for Cryptographic Applications. Proceedings of 10th Spanish Meeting on Cryptology and Information Security, Salamanca, Spain, 2008, pp. 77–82.

[13] Rhouma, R.—Solak, E.—Belghith, S.: Cryptanalysis of a New Substitution-Diffusion Based Image Cipher. Communications in Nonlinear Science and Numerical Simulation, Vol. 15, 2010, No. 7, pp. 1887–1892, doi: 10.1016/j.cnsns.2009.07.007.

[14] Alvarez, G.—Amigó, J. M.—Arroyo, D.—Li, S.: Lessons Learnt from the Cryptanalysis of Chaos-Based Ciphers. In: Kocarev, L., Lian, S. (Eds.): Chaos-Based Cryptography. Springer, Berlin-Heidelberg, Studies in Computational Intelligence, Vol. 354, 2011, pp. 257–295, doi: 10.1007/978-3-642-20542-2_8. ISBN 978-3-642-20541-5.

[15] Li, C.—Li, S.—Lo, K.-T.: Breaking a Modified Substitution-Diffusion Image Cipher Based on Chaotic Standard and Logistic Maps. Communications in Nonlinear Science and Numerical Simulation, Vol. 16, 2011, No. 2, pp. 837–843, doi: 10.1016/j.cnsns.2010.05.008.

[16] Li, C.—Xie, T.—Liu, Q.—Cheng, G.: Cryptanalyzing Image Encryption Using Chaotic Logistic Map. Nonlinear Dynamics, Vol. 78, 2014, No. 2, pp. 1545–1551, doi: 10.1007/s11071-014-1533-8.

[17] Preishuber, M.—Hütter, T.—Katzenbeisser, S.—Uhl, A.: Depreciating Motivation and Empirical Security Analysis of Chaos-Based Image and Video Encryption. IEEE Transactions on Information Forensics and Security, Vol. 13, 2018, No. 9, pp. 2137–2150, doi: 10.1109/TIFS.2018.2812080.

[18] Fridrich, J.: Symmetric Ciphers Based on Two-Dimensional Chaotic Maps. International Journal of Bifurcation and Chaos, Vol. 8, 1998, No. 6, pp. 1259–1284, doi: 10.1142/S021812749800098X.

[19] Solak, E.—Çokal, C.—Yildiz, O. T.—Biyikoğlu, T.: Cryptanalysis of Fridrich's Chaotic Image Encryption. International Journal of Bifurcation and Chaos, Vol. 20, 2010, No. 5, pp. 1405–1413, doi: 10.1142/S0218127410026563.

[20] Xie, E. Y.—Li, C.—Yu, S.—Lü, J.: On the Cryptanalysis of Fridrich's Chaotic Image Encryption Scheme. Signal Processing, Vol. 132, 2017, pp. 150–154, doi: 10.1016/j.sigpro.2016.10.002.

[21] Fu, C.—Hou, S.—Zhou, W.—Liu, W. Q.—Wang, D. L.: A Chaos-Based Image Encryption Scheme with a Plaintext Related Diffusion. Proceedings of 2013 9th International Conference on Information, Communications and Signal Processing (ICICS 2013), Tainan, Taiwan, 2013, pp. 1–5, doi: 10.1109/ICICS.2013.6782914.

[22] Zhang, Y.: The Image Encryption Algorithm with Plaintext-Related Shuffling. IETE Technical Review, Vol. 33, 2016, No. 3, pp. 310–322, doi: 10.1080/02564602.2015.1087350.

[23] Li, Z.—Peng, C.—Li, L.—Zhu, X.: A Novel Plaintext-Related Image Encryption Scheme Using Hyper-Chaotic System. Nonlinear Dynamics, Vol. 94, 2018, No. 2, pp. 1319–1333, doi: 10.1007/s11071-018-4426-4.

[24] Ovseník, Ľ.—Turán, J.—Huszaník, T.—Oravec, J.—Kováč, O.—Oravec, M.: An Image Encryption Algorithm with Plaintext Related Chaining. Computing and Informatics, Vol. 38, 2019, No. 3, pp. 647–678, doi: 10.31577/cai_2019_3_647.

[25] Cao, C.—Sun, K.—Liu, W.: A Novel Bit-Level Image Encryption Algorithm Based on 2D-LICM Hyperchaotic Map. Signal Processing, Vol. 143, 2018, pp. 122–133, doi: 10.1016/j.sigpro.2017.08.020.

[26] Alawida, M.—Samsudin, A.—Teh, J. S.—Alkhawaldeh, R. S.: A New Hybrid Digital Chaotic System with Applications in Image Encryption. Signal Processing, Vol. 160, 2019, No. 15, pp. 45–58, doi: 10.1016/j.sigpro.2019.02.016.

[27] Zhu, S.—Zhu, C.: Image Encryption Algorithm with an Avalanche Effect Based on a Six-Dimensional Discrete Chaotic System. Multimedia Tools and Applications, Vol. 77, 2018, No. 21, pp. 29119–29142, doi: 10.1007/s11042-018-6078-2.

[28] Sun, S.—Guo, Y.—Wu, R.: A Novel Image Encryption Scheme Based on 7D Hyperchaotic System and Row-Column Simultaneous Swapping. IEEE Access, Vol. 7, 2019, pp. 28539–28547, doi: 10.1109/ACCESS.2019.2901870.

[29] Mihálik, J.—Gladišová, I.—Michalčin, V.: Two Layer Vector Quantization of Images. Radioengineering, Vol. 10, 2001, No. 2, pp. 15–19.

[30] Feigenbaum, M. J.: Universal Behavior in Nonlinear Systems. Physica D: Nonlinear Phenomena, Vol. 7, 1983, No. 1-3, pp. 16–39, doi: 10.1016/0167-2789(83)90112-4.

[31] Gleick, J.: Chaos: Making a New Science. Vintage Books, London, 1998, 380 pp., ISBN 978-07-4938-606-1.

[32] Ibarra Olivares, E.—Vázquez-Medina, R.—Cruz-Irisson, M.—Del-Rio-Correa, J. L.: Numerical Calculation of the Lyapunov Exponent for the Logistic Map. Proceedings of 2008 12th International Conference on Mathematical Methods in Electromagnetic Theory (MMET 2008), Odessa, Ukraine, 2008, pp. 409–411, doi: 10.1109/MMET.2008.4581011.

[33] IEEE: 754-2019 – IEEE Standard for Floating-Point Arithmetic. doi: 10.1109/IEEESTD.2019.8766229.

[34] Liu, L.—Miao, S.: A New Image Encryption Algorithm Based on Logistic Chaotic Map with Varying Parameter. SpringerPlus, Vol. 5, 2016, No. 1, Art. No. 289, doi: 10.1186/s40064-016-1959-1.

[35] Oravec, J.—Turán, J.—Ovseník, Ľ.—Huszaník, T.: A Chaotic Image Encryption Algorithm Robust Against the Phase Space Reconstruction Attacks. Acta Polytechnica Hungarica, Vol. 16, 2019, No. 3, pp. 37–57, doi: 10.12700/aph.16.3.2019.3.3.

[36] Oravec, J.—Turán, J.—Ovseník, Ľ.: Image Encryption Technique with Key Diffused by Coupled Map Lattice. Proceedings of 2018 28th International Conference Radioelektronika, Prague, Czech Republic, 2018, pp. 1–6, doi: 10.1109/RADIOELEK.2018.8376374.

[37] Wu, Y.—Noonan, J. P.—Agaian, S.: NPCR and UACI Randomness Tests for Image Encryption. Journal of Selected Areas in Telecommunications (JSAT), Vol. 2, 2011, No. 4, pp. 31–38.

[38] Shannon, C. E.: Communication Theory of Secrecy Systems. The Bell System Technical Journal, Vol. 28, 1949, No. 4, pp. 656–715, doi: 10.1002/j.1538-7305.1949.tb00928.x.

[39] Liu, L.—Lei, Y.—Wang, D.: A Fast Chaotic Image Encryption Scheme with Simultaneous Permutation-Diffusion Operation. IEEE Access, Vol. 8, 2020, pp. 27361–27374, doi: 10.1109/ACCESS.2020.2971759.

Jakub Oravec received his M.Sc. and Ph.D. degrees from the Department of Electronics and Multimedia Communications, Technical University of Košice in 2015 and 2019, respectively. Since April 2020 he has served there as Assistant Professor. His research interests include image encryption, steganography and digital image processing.

Ľuboš Ovseník received his M.Sc. and Ph.D. degrees from the Department of Electronics and Multimedia Communications, Technical University of Košice in 1990 and 2002, respectively. Currently, he works at the Technical University of Košice as Associate Professor. His research interests are fiber optic communication systems and sensor networks.

Ján Turán received Ing. (M.Sc.) degree in physical engineering with honours from the Czech Technical University, Prague, Czech Republic, in 1974, and RNDr. degree in experimental physics with honours from the Charles University, Prague, Czech Republic in 1980. He received his CSc. (Ph.D.) and Dr.Sc. (D.Sc.) degrees in radioelectronics from the Technical University of Košice, Slovakia in 1983 and 1992, respectively. Since March 1979, he has been at the Technical University of Košice as Full Professor for electronics and information technology. His research interests include digital signal processing and fiber optics, communication and sensing.

Tomáš Huszaník received his M.Sc. degree from the Department of Electronics and Multimedia Communications, Technical University of Košice in 2017. Currently, he is Ph.D. student in the same department. His research interests include all optical networks and degradation mechanisms in all optical WDM systems.

Computing and Informatics, Vol. 39, 2020, 1282–1310, doi: 10.31577/cai_2020_6_1282

TIME-SENSITIVE ADAPTIVE MODEL FOR ADULT IMAGE CLASSIFICATION

Mohammad Reza Mazinani, Seyyed Mojtaba Hoseini
Kourosh Dadashtabar Ahmadi

Faculty of Electrical and Computer Engineering
Malek Ashtar University of Technology
15875-1774, Iran
e-mail: `mazinany@gmail.com, mojtabahoseini@aut.ac.ir, dadashtabar@mut.ac.ir`

Abstract. Images play an important role in modern internet communications, but not all of the images shared by the users are appropriate, and it is necessary to check and reject the inappropriate ones. Deep neural networks do this task perfectly, but it may not be necessary to use maximum power for all images. Many easier-to-identify images may be classified at a lower cost than running the full model. Also, the pressure on the system varies from time to time, so an algorithm that can produce the best possible results for different budgets is very useful. For this purpose, a deep convolutional neural network with the ability to generate several outputs from its various layers has been designed. Each output can be considered as a classifier with its own cost and accuracy. A selector is then used to select and combine the results of these outputs to produce the best possible result in the specified time budget. The selector uses a reinforcement learning model, which, despite the time-consuming learning phase, is fast at execution time. Our experiments on challenging social media images dataset show that the proposed model can reduce the processing time by 32 % by sacrificing only 1.4 % of accuracy compared to the VGG-f network. Also, using different metrics such as F1-score and AUC (the Area Under the Curve in the accuracy vs. time budget chart), the superiority of the proposed model at different time budgets over the base model is shown.

Keywords: Adult content recognition, time-sensitive, cost-sensitive model, convolutional neural network, deep learning, image classification

Mathematics Subject Classification 2010: 68T10

1 INTRODUCTION

In image and video sharing platforms, users upload images or videos to the multimedia sharing platform's servers and share them publicly. In such platforms, latency in the appearance of shared data is not acceptable to the users, and images are expected to be visible immediately after upload. Also, according to the rules of most sites, some images and videos are in the category of unacceptable images and must be deleted. These categories can vary depending on the different platform policies, for example, violence, racism, and nudity [1, 2]. Due to the huge number of images uploaded to the image-sharing platforms, it is not possible to manually check all images. So, a high-speed automatic image recognition system becomes a necessity for them.

Additionally, by increasing the speed of the Internet and improving video transfer technologies, live video streaming is becoming more and more popular. Live streaming provides video frames instantly. Therefore, in live video streaming platforms, in addition to being careful in filtering objectionable video frames, processing online is also very important, and this filtering should not cause lag or delay in the live streaming. [3, 4] have tried to identify people who abuse these platforms and then attempt to remove this content or restrict the streamer user. Many providers who publish adult content on the live streaming platform with the intention of generating revenue or harassment, are broadcasting sex-related content intelligently. They may only show nude content in part of the video, or may only produce sexual content through gesture or voice, which is harder to detect. Hence, because it is not feasible to review the video by human operators, it is necessary to use algorithms to do the review thoroughly.

The image recognition process in image sharing platforms has other challenges. Some photos are taken by professional users in the right lighting conditions and are high-quality images, but some photos are taken by amateur users with mobile phones and are blurry, poorly lit, oversaturated, or nearly dark. Another type of challenge is because of the content of the images. Some sports, such as boxing and swimming, involve half-naked bodies but are not sexual images. Nude pictures of babies also show a naked picture of a human being, but it is not considered porn. People in the image may have clothes, but due to their facial or body posture, the whole image is considered sexual.

One way to overcome these challenges is to use deep convolutional neural networks (CNN). In recent years, very deep networks have been presented with acceptable accuracy in image classification on databases with 1 000 image categories, such as googlenet [5], vgg-verydeep [6], and resnet-152 [7]. We also use CNN in our model to accurately detect adult images. But the use of neural networks causes a new challenge. These networks usually take a long time to process each image, and the deeper the network, the longer the processing time. In this article, our main focus is on reducing processing time.

Despite the various challenges, many images on image sharing servers have simple content and can be detected with simple algorithms. For example, images of objects, animals, buildings, etc. can be accurately categorized into normal classes using simple convolutional neural networks with a small number of layers. Using simple networks saves computational time and cost, but is less accurate on more complex images. Therefore, it will be very advantageous to design a model that can provide tags for simple images at a lower computational cost using a simple network. This model should use more complex networks to accurately detect complicated images.

Also, in some cases, the classifier must be able to produce the best result given the limitations of available computational sources. Conditions like:

- The volume of requests to the server: In some hours, many requests are sent to the servers, and in other hours the requests may be much less than server processing capacity.
- Limited processing capacity of different devices: Processing power on portable devices such as mobile phones, wearables, IoT devices, or laptops is different. So the algorithm must have the ability to adapt to processing power.
- Increasing the processing accuracy by a cloud processor: Simple images can be processed by the user's device, and if the result is desirable, it can be displayed to the user; otherwise, the data can be sent to a cloud server to continue processing. [8].

Given the above conditions, it will be necessary to create a model that can process images with a limited source. In this article, we consider processing time as our limited resource. Time constraints can take many forms.

- The fixed time limit for each image (hard budget): The classifier should provide the best result based on the specified time limit. This model can be useful in cases where images are given to different devices for processing. As an example, in [9] each image is assigned to a separate processor, and when it reaches the time limit for each image, these processes end, and the obtained result is reported.
- The time limit for a set of images (average budget): A set of images is sent to the processor, and it is necessary to process them in a limited time interval. In this case, the processing time of each image is not limited separately, and the only limitation is the average processing time. This model is useful when the complexity of input images varies.

The model introduced in this article is designed to meet the needs of the average budget. The general structure of the model is shown in Figure 1. As shown in Figure 1, images uploaded by users are inserted in a buffer. The server reads a set of images and sends them with the time limit to our model. If the number of images in the buffer become too large, the server considers a shorter time limit, to increase the output rate of the buffer and reduce the size of the buffer.

Our goal is to provide a novel model with the ability to classify a set of images in a specified time limit, with the highest achievable accuracy. Our method does not apply the same processing to all images like static methods [10, 11, 12, 13],

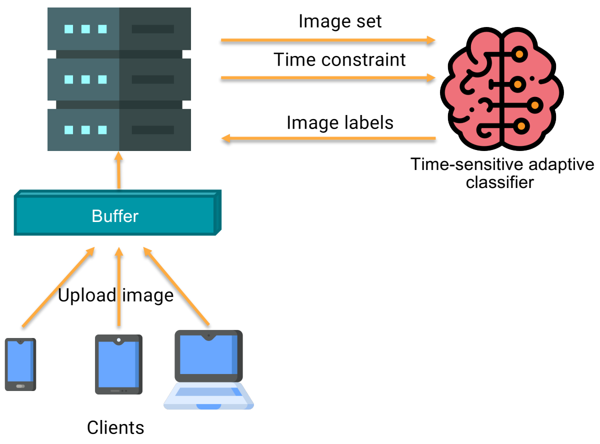


Figure 1. The overall structure of the time-sensitive adaptive model application on the image-sharing platform server (Icons made by Freepik from `https://www.flaticon.com`)

but it tracks a different process for each input instance. So our method could be considered as an input dependent method.

In this paper, a model is presented that can process a set of images with any time budget. The innovation of this research is the specific design of its selector, by which, in addition to the ability to finish the process at different time budgets, also the accuracy can be increased by combining the outputs of the different layers of the CNN classifier. The increase in speed and accuracy is due to the fact that the selector uses outputs of the earlier layers of the CNN classifier for simpler images, and continues to the deeper layers of the network for the more complex images. We used reinforcement learning to design the selector, which has many advantages for this task. The advantages of our selector model are described below.

- Improving the average speed of adult image classification: Our basic CNN contains several outputs from the middle layers. We trained the selector by the q-learning algorithm. The reinforcement learning agent tries to minimize the time cost using earlier layers of CNN, so the overall speed increased.
- Increasing accuracy by combining outputs: In cascading methods, the last classifier result is presented as the final model output. But in our model after observing the results of the outputs, the reinforcement learning agent decides which category is better to be presented as the correct final result, based on past experiences.
- Flexibility to select any subset of classifier outputs in any order: The selection of each of the classifier outputs is defined as an action in the reinforcement learning model. Therefore, unlike cascading models, there is no limit on the

order of classifier outputs selection. So, for example, when we have a lot of processing capacity, the selector may choose the most complex classifier output. This ability to freely choose between outputs allows the selector to be applied to any structure, such as a CNN with complex tree structure, or even several separate CNNs.

- The ability of the selector to work with different time budgets: The selection of classifier outputs is done in such a way that the cost of the classifier error and the cost of processing time are minimized. By increasing the time cost factor, the selector tends to select classifiers with lower cost. Therefore, by changing this factor, the selector will be trained for different time budgets. At runtime, any of these trained models can be easily used based on the given time budget.

The rest of the paper is organized as follows: Section 2 provides an overview of the previous works in the field of adult image recognition, as well as some related articles on cost-sensitive classification subject. Section 3 first demonstrates the contribution and justification of our model, and then explains the overall structure of the model and describes its parts in detail. Section 4 presents the obtained experimental results, and Section 5 discusses the properties of the proposed model and plans for future work.

2 RELATED WORK

Since our model consists of two primary parts, one is the base classifier, and the other is the selector, which is responsible for managing the outputs of this classifier. Therefore, articles related to these two areas and their advantages and disadvantages are described separately.

2.1 Adult Content Detection

Adult image recognition methods can be divided into three general categories: skin as the main feature, local features, and convolutional neural network (CNN). Due to the limitations of skin-based methods and local features, we chose deep neural networks for the classification part of the model. The limitations and weaknesses of these methods are discussed below.

2.1.1 Skin as the Main Feature

In these methods, the main focus is on finding skin regions, and then use the characteristics of these regions, such as the distribution of skin pixels, the area of skin regions, the number of skin regions, etc., to classify images [18, 19]. These methods do not recognize black and white images and do not have enough recognition capabilities in finding skin in different lighting conditions. To increase the ability to find skin, Lee et al. [20] find the faces in the image, and by sampling the skin pixels from the face, they detect skin pixels in other areas of the image. In skin-based methods, despite the increased accuracy in finding skin areas, the accuracy in detecting adult images was very low. So different methods added new features, such as the number of faces and the regions of the faces [21], texture [22], and shape [23] to increase accuracy. Although the classification error was reduced with the addition of new features, the processing time was increased significantly.

In addition to the problems mentioned, skin-based methods also have an intrinsic drawback because they usually do not take into account people's position. For instance, in some images, the skin area is very small; however, it is considered an adult image due to people's positions. Therefore, these methods have very low accuracies in practice.

2.1.2 Local Features

Classifiers based on local features, extract attributes and shapes from different parts of the image. Lopes et al. [24] use the scale invariant feature transform (SIFT) feature. First, these features are extracted locally from the image, and then all the features are formed in the form of histograms of visual features. It then uses this new feature to classify images. They use the standard SIFT feature, which is extracted from non-color images. They then use the Hue-SIFT feature, which adds color information to the previous feature and increases the accuracy of the algorithm. Hue-SIFT does not use the color characteristic properly, so to take advantage of the color feature, Deselaers et al. [25] use image patches as local features. They then reduce the size of image patches using PCA, and use the bags of visual words method to produce the final feature vector. They finally use created feature vector to classify images.

These methods are more accurate than skin based methods. Still, because they often use handcrafted features, they do not produce an accurate result on new and intricate images.

2.1.3 Convolutional Neural Network (CNN)

Recently, deep learning methods have much higher accuracy than previous algorithms, especially in image classification [26]. [27] uses deep learning to classify adult images, using a simple combination of two – Alex-net [28] and googlenet [29] networks. This paper demonstrates that the use of CNN is more accurate than pre-selected features such as SIFT.

Deep neural network training requires a large number of training images. Some articles use data augmentation, for example, flipping the image or using different images crop [30]. Another way to overcome the lack of training images problem is to use the weights of the pre-trained network to train a similar network (fine-tuning). Vitorino et al. [32] use this method to detect child pornographic images. They first choose a CNN that was trained on ImageNet [33] and fine-tuned it using 200 000 training adult images. Then, with a small dataset from children, they fine-tuned the trained network to detect child pornographic images.

We used data-augmentation and fine-tuning in our work and will explain them in Section 3.2.

Another way to increase the accuracy of classification is to combine multiple classifiers. Shen et al. [31] combined the results of several CNNs using Bayesian networks. Cheng et al. [34] combined the features extracted by the two CNNs and adopted the final feature vector for classification. In these methods, due to the processing of all CNNs, the processing time is greatly increased. Also, this processing is done for all images, regardless of the difficulty and ease of the image. In our method, the amount of processing depends on the input difficulty, so the overall processing time will be optimized.

Due to the superiority of the convolutional neural network for classification, we used this method to build our base classifier. But to generate output at different times, we added several intermediate outputs to the structure of one of the known CNNs. Table 1 shows a comparison of existing work in the field of adult image classification.

Model	Advantages	Disadvantages
adult image classification using skin region [18, 19]	classify high quality naked image	unable to recognize grayscale images. very low accuracy
adult image classification using skin plus face [20, 21] texture [22], or shape [23]	acceptable accuracy in color image	unable to recognize grayscale images. very time consuming
adult image classification using local features [24, 25]	medium accuracy on color and grayscale image	time consuming. not accurate on intricate images
adult image classification using CNN [27]	accurate	requires a large number of training images
adult image classification using CNN plus data augmentation [30], or fine-tuning CNN [32]	accurate. trainable by small training set	time consuming
adult image classification using combination of CNNs [31, 34]	very accurate	very time consuming

Table 1. Adult image classification methods comparison

2.2 Cost-Sensitive Models

In addition to trying to reduce the processing time of previous deep neural networks in a static way [10, 11, 12, 13], some input dependent work has been done to reduce time. In this section, we will review some related cost-sensitive articles. For a more informative description, these methods are divided into three categories.

2.2.1 Dynamic Pruning

Some works remove unnecessary nodes or layers within a CNN. For example, Bengio et al. [36] remove some nodes from the layers during the training and testing to reduce computations. Their method works like a dropout layer in CNN but has tried to estimate the best path in the neural network for different samples and deactivate unnecessary nodes. The other work by [16, 17] tries to reduce the processing time of the network by pruning the channels. For each input image, after each convolutional layer, the output channels that are not considered useful for image classification are pruned.

Because these methods use an integrated structure, they do not produce any results between processing steps. Therefore, after determining the result, it is not possible to increase the accuracy with more processing along the new path, and the whole network must be processed with new parameters from the beginning.

2.2.2 Decision Making with Threshold

Some works use a cascade structure to make decisions with the threshold. The cascade model is CNN with some output from the middle layers. They use the confidence value (obtained from the softmax layer output), which shows a number between zero and one for each class. At the time of inference, if the value of confidence is greater than a specific value, the model terminates the processing and shows the last obtained image label as a result of the whole network. Therefore, for some images, fewer processing steps are performed. Berestizshevsky and Even [14] used this method and selected the ResNet [7] network as the cascade model, with three outputs in the middle layers. The same method is used in [8], but the focus is on the use of CNN on devices that either have little memory to maintain all network parameters or do not have the computational power required to process in a specified short time. Therefore, fewer layers are processed on the portable device, and if more processing is required, the information is sent to the cloud server.

One of the advantages of this method is that they could change the decision threshold at test time. If the output accuracy is more important, increase the value of the decision threshold, and when the time budget is low, reduce the threshold value to increase the speed. And these methods are fast in selecting a processing path. But the disadvantage is that it can only be applied to the cascading model and cannot be applied to the network structure in the form of a tree. Also, specifying a fixed decision threshold may reduce the accuracy of the decision. Our method uses one decision-maker (selector) for all output and path of the network, and it could be used on any structure, and the final class is selected accurately using all available outputs.

2.2.3 Classifiers as Decision-Makers

The use of classifiers to make decisions increases accuracy and allows the use of non-cascade structures. In different network structures in each branch, a classifier has the task of choosing the path. For example, Bolukbasi et al. [35], used a cascading structure. At each branch, the classifier decides to exit the correctly identified samples from the processing path. Odena et al. [37] introduce a structure that has three metalayers, each consisting of two modules. Before processing each metalayer, a function uses previous outputs to decide which module to use in the metalayer so that the network can achieve the desired accuracy and speed. Liu and Deng [15] use control modules within the network to select the best processing path. Control modules are trained using the backpropagation algorithm and reinforcement learning. They tested their model on high-low, cascade, chain, and hierarchy structures.

These methods use complex classifiers to select the best route on CNN, so they have two drawbacks. First: using a complex classifier adds a time cost to the whole process. Second: the selector classifier error is added to the total image classification error. Our method at the time of testing is a lookup table, and the required processing time is very low. Also, to prevent errors, the next path is selected, after the output of the classifier in the current path is specified. Therefore, the confidence obtained in this output is used for a better decision in choosing the future path or exit the network. And unlike some methods [35], we do not predict this confidence that may cause an error.

Table 2 shows a comparison of existing work in the field of cost-sensitive image classification. To estimate the correctness of the classification, we use the criterion presented in [14]. We use a smart selection of outputs instead of using the threshold. The details of our selector algorithm are explained in Section 3.3.

Model	Advantages	Disadvantages
Dynamic pruning using pruning nodes [36], or pruning channels [16, 17]	speed up the normal CNN	unable change speed or accuracy at runtime
Decision making with threshold [14, 8]	change speed or accuracy at runtime easily	only could be applied to the cascading model. may reduce the accuracy of the decision
Classifiers as decision-makers in specified structure [35, 37]	select path in network more accurate	just used in specified structure. time consuming. may increase error.
Classifiers as decision-makers in any structure [15]	work in any structure	time consuming. may increase error.

Table 2. Cost-sensitive methods comparison

3 PROPOSED MODEL

3.1 Time-Sensitive Adaptive Classifier Structure

Our model proposes to use a dynamic classifier selection technique. The paper [39] states that the dynamic classifier selection is composed of three phases:

1. Classifiers generation,
2. Selection, and
3. Fusion.

Our proposed model combines two latter phases, so it consists of two main parts, the Classifiers generation and the Selector (a combination of selection and fusion). The Classifiers are the outputs of different layers of CNN, which are generated at different times from the start of the network execution. The Selector is a reinforcement learning agent that selects some of these outputs to achieve the best performance, taking into account the time budget.

Figure 2 shows the general structure of the model and the connection between these two parts.

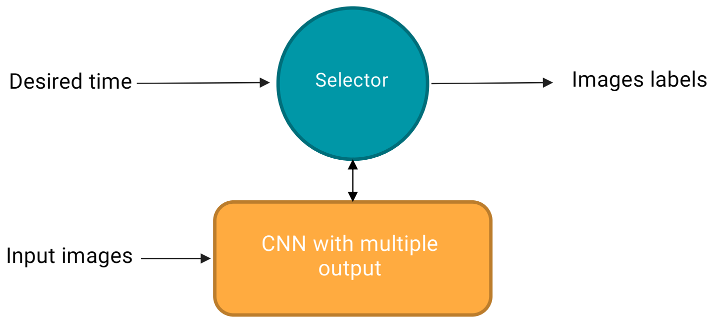


Figure 2. Structure of the proposed method: A model based on reinforcement learning is used for the selector part, and a deep convolutional neural network with several outputs from the middle layers is used for the classifier part

The goal of our cost-sensitive problem is to reduce the total cost of the classification, for any given time budget (t'). The total cost of the model ($Cost_{total}(t')$) is shown as:

$$Cost_{total}(t') = Cost_{error}(t') + Cost_{evaluation\text{-}time}(t') \qquad (1)$$

where $Cost_{error}(t')$ is the classifier error cost that is shown in Equation (2) and $Cost_{evaluation\text{-}time}(t')$ is the time of running the algorithm on a set of data that is shown in Equation (3).

$$Cost_{error}(t') = \sum_{i=1}^{N} |W'(x_i) - W(x_i|t')| \tag{2}$$

$$Cost_{evaluation\text{-}time}(t') = \sum_{i=1}^{N} T(x_i|t') + T_{selector}(t') \tag{3}$$

In these formulas, x_i is the input image, N is the number of input images, $W(x_i|t')$ is the output label of the model given time budget t', and $W'(x_i)$ is the true label of image x_i, $T(x_i|t')$ is the CNN processing time for input image x_i given time budget t', and $T_{selector}(t')$ is processing time of our selector module given time budget t'. At runtime, the selector easily accesses a Q-table and selects the action with the highest value, so its processing time could be ignored. CNN processing time is calculated based on the number of operations performed in the network layers. More precisely, we show $T(x_i|t')$ as follows:

$$T(x_i|t') \simeq \sum_{l=1}^{L} P_l(x_i|t').n_{l-1}.s_l^2.n_l.m_l^2, \tag{4}$$

$$P_l(x_i|t') = \begin{cases} 1, & l^{\text{th}} \text{ layer processed,} \\ 0, & l^{\text{th}} \text{ layer not processed,} \end{cases} \tag{5}$$

where L is the number of all layers in the CNN, $P_l(x_i|t')$ is the coefficient that determines whether each layer is processed or not, n_{l-1} is the number of the input channels of the l^{th} layer, n_l is the number of output channels (filters), s_l is the spatial size of the filter, and m_l is the spatial size of the output channels. In a regular CNN, all layers are processed, so $P_l(x_i|t')$ is equal to 1 for all layers. But in our proposed method, we intend to prevent some layers from being processed. The selector accomplishes this goal using a reinforcement learning method. If the input image can be correctly recognized by the outputs of the first layers, it prevents the execution of the final layers of the network. So $P_l(x_i|t')$ will be equal to zero for the final layers of the network if the input image is easy.

According to the expressed formulas, the time complexity of the final model could be written as:

$$O\left(\sum_{l=1}^{L} P_l.n_{l-1}.s^2.n_l.m_l^2\right). \tag{6}$$

Here, P_l $(0 \leq P_l \leq 1)$ is the probability of using each layer of the model. Our method causes P_l to be less than 1 for a number of layers, as described in Section 3.3.

In the following sections, we illustrate two main parts of our proposed method named time-sensitive adaptive classifier.

3.2 Classifiers Generation

As stated in the previous section, the classifiers generation is the first phase of the dynamic classifier selection technique. We need a range of classifiers, some with faster output but less accurate, and some with slower output but more accurate. For this purpose, a deep convolutional neural network (CNN) with several outputs in the middle layers is designed. From the popular and state-of-the-art CNNs such as Googlenet [5], Resnet-152 [7], and VGG network [26], we chose the VGG network. Because this network has fewer layers and is faster than the rest, and also our problem is a two-class problem, so it does not need a complex structure for high-accuracy classification. There are different types of VGG network, and we chose the lightest and fastest one called vgg-f. For the need of the final model, we added three more outputs to the base network.

The designed network, which has four outputs, is shown in Figure 3. The output layers are shown with a red rectangle and are named softmax1, softmax2, softmax3, softmax4. In the convolutional and fully connected layers, we show the characteristic of each layer by "$Convs * s * n$" and "$FCs * s * n$" respectively, where $s * s$ is the size of the filter and n is the number of the input channel. The processing time required to produce each output is the sum of the processing times of the layers on the path to it. Therefore, to achieve the softmax1 output, the convolutional layer, the pooling layer, the fully connected layer, and finally, the softmax1 layer must be processed, so according to Figure 3, the processing time will be $t_1 + t_3$. Normally to reach the softmax3 output, the processing time will be $t_1 + t_2 + t_4 + t_7$. However, if softmax1 output is already generated on the same image, the output of the first layer of pooling is ready, so there is no need to spend time t_1. In this case, the processing time for the softmax3 output will be $t_2 + t_4 + t_7$.

Different network outputs produce their results with different accuracies at different times. The softmax1 output has the highest processing speed and the lowest accuracy, and the softmax4 output has the lowest processing speed and the highest accuracy.

To train our network, we use fine-tuning of the vgg-f network [26]. The vgg-f network has been trained on the ImageNet dataset [33] containing 1 000 classes. We train each of the four outputs separately. To fine-tune the network, first, remove the last fully connected layer before softmax4 from the vgg-f network, and replace it with a fully connected layer with two output nodes for both normal and adult classes. To train only the newly inserted layer, we keep the weights of all the previous layers constant, then train the network with our dataset. The fine-tuning method is also used to train the branch layers. For example, to train the fully connected layer before softmax 1, the fully connected layer with two output nodes is connected to the end of the first pooling layer, as shown in Figure 3. Then, by keeping the weight of the main network layers constant, this new layer is trained with our training dataset. The same goes for training the other two branches.

Before training or testing, we resized all the images to 224×224. We also used the data augmentation method to improve the training of the neural network. So

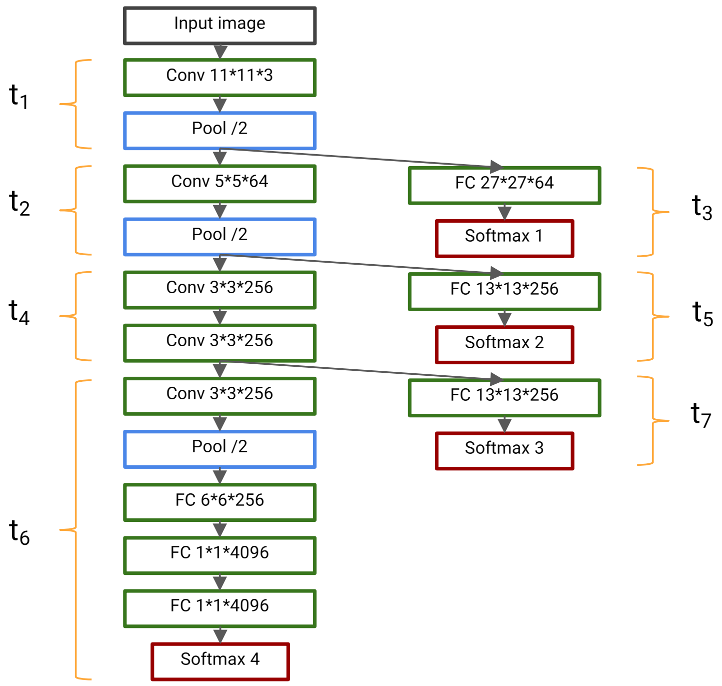


Figure 3. Designed CNN as a classifier with multiple outputs: the green rectangles show the convolutional and fully-connected layers, the blue rectangles represent the pooling layers, and the red rectangles represent the softmax (Network output) layers. $t_1, t_2, \ldots, t_7$ indicate the processing time of each part of the network.

each time the images were called during the training, a slight shift in image cropping, a change in image brightness, and image flipping has been applied.

3.3 Selector

The second phase of a dynamic classifier selection technique is the design of the selector that task is to select and fuse the outputs. After training our CNN, a selector is required to select and combine the best outputs that can get the best result in the shortest time for each image. In other words, for simpler images, output with less processing time should be selected, and for more difficult images, more complex outputs with higher accuracy should be selected. The selector should also be able to keep the average processing time of the images within a specified time budget.

Among the reinforcement learning (RL) methods, Q-learning has been selected. By embedding the accuracy and the time-cost to the reward of the RL method, we lead the RL agent to select the outputs in which the answer with the highest accuracy and the lowest cost will be achieved. At execution time, when the RL agent selects an output, the CNN executes the path to that output, then the RL agent observing the output decides whether to select another output or terminate the process and produce a resulting label. The agent observes and considers both the selected classifier output and its confidence about that output.

The degree of confidence in the output is obtained using the softmax layer output. The softmax formula produces a number between zero and one for each class so that the sum of all the probabilities is equal to one. This number represents the score that determines how much each input belongs to the class:

$$P_i = \frac{e^{s_i}}{\sum_{c=1}^{N} e^{s_c}} \tag{7}$$

where P_i is the probability of each output, s_i is the score value of class i, and N is the number of classes.

The probability given to each class is considered as the network's confidence in the output of that class. Therefore, if this confidence is high, the path selected in CNN has achieved the desired result, and the selector can report the final selected class. But, if this probability, for both classes is close to 0.5, it indicates that the classifier is not sure of the correctness of its output, so the selector selects another output of the neural network to get a more accurate result.

The overall structure of the reinforcement learning algorithm and a number of its states are shown in Figure 4. To further explain our reinforcement learning model, action, state, and reward for the designed model are described below.

Action: In the beginning, the agent can select each of the CNN outputs and process the path leading to that output. In the next step, if more processing is required, the agent can choose another output among the unselected outputs. For example, action $a1$ means to select the softmax1 output. After performing the action $a1$, we reach one of the states related to this output according to the resulting confidence. Also, in each state, the agent can choose one of the normal and adult classes and then go to the final state, which itself is defined as an action. Therefore, according to the confidence obtained at each output, the agent decides to select one of the classes or to continue processing a new path in the neural network.

State: Different states are defined for the proposed model. There is a start and finish state displayed by red rectangles in Figure 4. The two states before the finish state represent the number of model classes (normal and adult). After leaving each of the middle states, the agent (to complete the processing) could decide to choose one of these two classes. Other states also specify which output had been selected in the neural network, and what classes with what confidence is selected. Since we have only two classes in this case, if the confidence level for the output of the normal class is above 0.5, the normal class is selected, otherwise, the adult class is selected. In Figure 4, $S1 = R1$ shows the selection of the normal class at the softmax1 output with confidence above 0.5, and $S1 = R2$

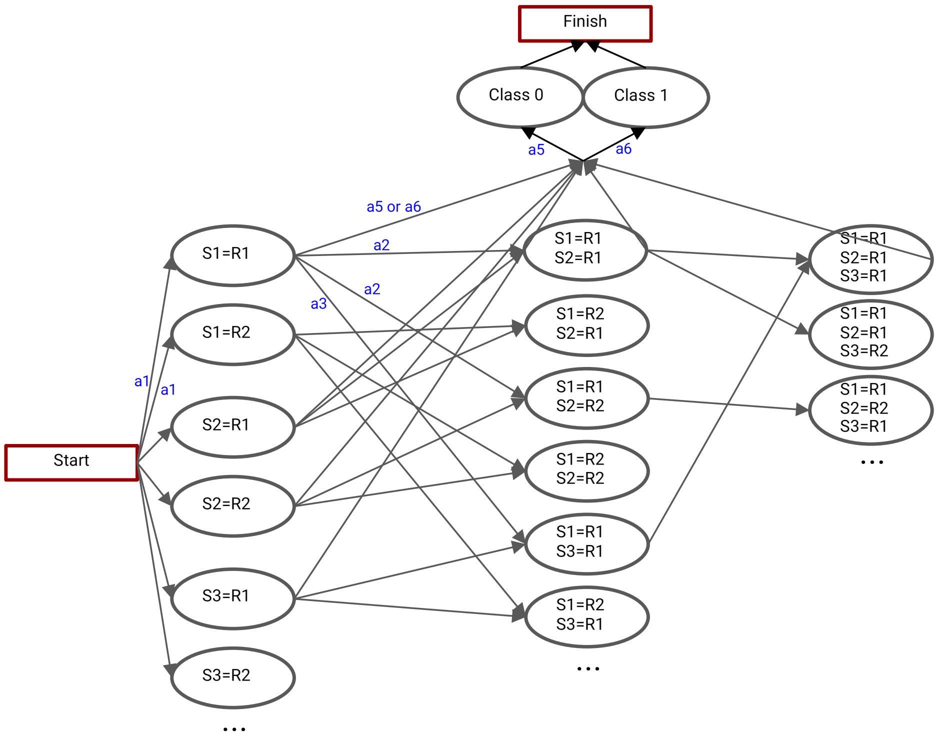


Figure 4. The general structure of the reinforcement learning algorithm in a simple mode: The start and finish states are displayed with red rectangles, and middle states are displayed with ellipses. The actions that shown by arrows named as $a1$, $a2$, $a3$, $a4$ indicate the selection of each of the CNN outputs, and $a5$, $a6$ specify the actions of selecting each of the final classes. S1 represents softmax1 output, S2 represents softmax2 output, and so on. $R1$ and $R2$ represent two value interval for confidence.

shows the selection of the adult class with confidence above 0.5 (equal to the confidence below 0.5 at the output of the normal class). That is the easiest way to quantize the confidence value into two intervals. In Section 4, this quantization is done with more intervals, and its effect on the achieved accuracy is shown.

Reward: In the Q-learning algorithm, the value of each state and action is updated according to the current reward and the highest reward obtained in the next state. Our presented reward depends on two factors: the calculation time of the processed CNN layers, and the misclassification error. Q-table values are updated using the following equation [41]:

$$Q_{new}(s_i, a_i) \leftarrow Q_{old}(s_i, a_i) + \beta.(Reward_i + \gamma.max_a Q(s_{i+1}, a) - Q_{old}(s_i, a_i)) \quad (8)$$

where $Q(s_i, a_i)$ is the value of state s_i by performing action a_i, β is learning rate, γ is the discount factor, $max_a Q(s_{i+1}, a)$ is the maximum value that can be obtained from state s_{i+1}, and $Reward_i$ is the reward received by performing action a_i in state s_i. According to Equation (1), to simultaneously consider the accuracy and time cost, the $Reward_i$ is determined as follows:

$$Reward_i = Accuracy_i - \alpha . TimeCost_i \tag{9}$$

where $Accuracy_i$ is achieved in final action based on whether the classification result is correct or not. If the class assigned to each instance in pre-finish states is correct, $Accuracy_i$ is set with a positive constant value (we used 10) and otherwise with a negative constant value (we used -10). $TimeCost_i$ for each action indicates the extra time spent to get the output in the CNN up to this point. The α parameter is a factor that determines the balance between time and accuracy. If we look for a faster result, this value is increased, so the results that are generated over a longer time will be more penalized.

By changing the parameter α, the model is trained for different time constraints. So, for various time budgets, we set the α to various values and we will have different Q tables.

According to the times $t_1, t_2, \ldots, t_7$ shown for different parts of the network in Figure 3, Equation (4) can be simplified for our model as follows:

$$T(x_i|t') \simeq \sum_{l=1}^{7} P_l(x_i|t').t_l \tag{10}$$

where $T(x_i|t')$ is the CNN processing time for input image x_i given time budget t', l indicates the l^{th} part of the CNN, and $P_l(x_i|t')$ is the phrase that determines whether each part is processed or not. By penalizing longer times using Equation (9), more layers will be neglected and processing time $T(x_i|t')$ will be reduced.

4 EXPERIMENT RESULTS AND ANALYSIS

4.1 Dataset Description

We created a dataset containing 18 000 different images to train and test the model. We wanted the database to have diverse and challenging images, so we collected many samples from public social networks or image sharing sites. Social network images are often produced by non-professional users, and many of them do not have proper lighting or quality. Dataset images were selected to include both simple and hard images. In the adult class, 38 % are simple images, but the rest of the images have various challenges. About 22 % of the images have poor illumination, in 40 % of the images the important parts of the image are covered with clothes or with text on the image, 20 % of the images are not naked and can only be detected by body positions. The dataset can be downloaded from [40].

Figure 5 shows some samples from this dataset. The upper row shows normal images; the three images on the left are simple, but the two images on the right are more difficult to recognize because of their large area of skin and high similarity to adult images. The lower row shows adult images; the three images on the left side include the naked bodies and are easy to recognize. But the two images on the right side are difficult to recognize, these two images are taken outdoors, and the peoples in them are not completely naked, but they are adult images because of showing certain parts of the body.

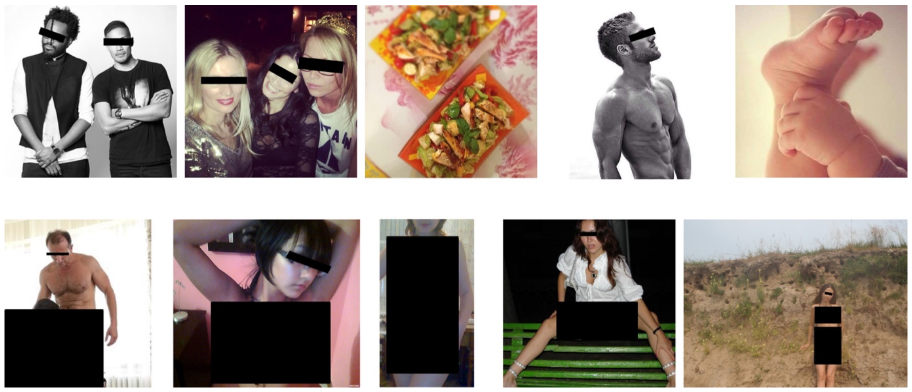


Figure 5. Sample images of our dataset: The first row shows the images of the normal class, and the second row shows the adult class

The entire dataset is divided into three parts. The neural network is trained using the first part of the dataset, the second part of the dataset is used to train the selector, And finally, the third part (test data) is used to report the accuracy of the network.

4.2 Experimental Setup

To train and test the deep neural network and other parts of the model, a device with 2.7 GHz Intel Core i5 processor and 8 GB RAM has been used. MatConvNet software package [38] has been used for training and testing the CNN.

Our CNN is trained using data from the first part of the dataset, then the results are reported on the images of the third part (test data). The values obtained for the various outputs of the CNN are shown in Table 3. The accuracy of the first output (softmax1) is about 77 % and the accuracy of the last output (softmax4) is about 93 %. At the output of the last layer, a 16 % increase in accuracy has been achieved, but the processing time has increased almost sevenfold. The times related to different parts of the CNN are shown with $t_1, t_2, \ldots, t_7$ according to Figure 3.

The reported processing time of each output is the average processing time for all images. And the accuracy is presented according to the following formula using the average accuracy of different classes.

$$Accuracy = \frac{\frac{TP}{totalNumberP} + \frac{TN}{totalNumberN}}{2} * 100 \tag{11}$$

where TP is the number of correctly classified images of the adult class and *totalNumberP* is the total number of images of the adult class, similarly, TN is the number of correctly classified images of the normal class and *totalNumberN* is the total number of images of the normal class. Using this formula, the accuracy of the classification does not change according to the imbalance between the number of samples in each class.

Output name	Accuracy (percent)	Time (ms)	Time according to Figure 3
Softmax1	77.13	4.65	$t_1 + t_3$
Softmax2	82.1	12.03	$t_1 + t_2 + t_5$
Softmax3	85.07	17.27	$t_1 + t_2 + t_4 + t_7$
Softmax4	92.53	33.36	$t_1 + t_2 + t_4 + t_6$

Table 3. Accuracy and the processing time of the outputs of the proposed method's CNN

4.3 Quantizing the Confidence into Four Intervals

As explained in the previous sections, to determine the different states in the selector, it is necessary to quantize the numerical values of the softmax output. We also want to define different states for different classes. Of course, the output nodes of the last layer of the neural network have the same number as classes, so in our case, it has two output nodes. Since the sum of the values of these two outputs is equal to one, the specified classes can be extracted using one of the output nodes. Assuming we display the output range of the normal output node with $[0, 1]$, if the output value is in the range $(0.5, 1]$, we consider it as the normal class, and if it is in the range $[0, 0.5]$, we consider it as the adult class.

In this section, the output range is quantized into four intervals. Therefore, the quantization is considered as follows:

$$R1 = [0, a], R2 = (a, 0.5], R3 = (0.5, a'], R4 = (a', 1] \tag{12}$$

where $R1, R2, R3, R4$ are different intervals. To make this intervals balanced, we set $a' = 1 - a$. Thus, if the output is in the range of $R1$ or $R4$, the confidence of the output is high, otherwise, the confidence is low. For example, if the output for the normal class is in the range $R4 = (a', 1]$, it indicates that the normal class is selected with high confidence, and if it is in the range $R3 = (0.5, a']$, the normal class is selected with low confidence.

Reinforcement learning models are trained using the second part of the database (as a training set for the selector) and different models are obtained for different values of a. The results for these models on the train set as well as the test set are shown in Figure 6.

In each model, by changing the α parameter in Equation (12), the resulting accuracies and their corresponding processing times are obtained.

According to Figure 6, the corresponding line to the all presented models is above the line of the base network, so the given models performed better at the same time budget. Models with a greater area under the curve are generally more accurate. For a better comparison, we have shown the area under the curve in the accuracy vs. time budget chart (AUC) for different models in Table 4. According to the table, the best result is 279.5 that is obtained on the test set for the parameter $a = 0.1$.

Model	Area Under the Curve
base net	201.3
$a = 0.05$	268.5
$a = 0.1$	**279.5**
$a = 0.2$	274.1
$a = 0.3$	260.6
$a = 0.4$	237.8

Table 4. Area under the curve for models with four quantization intervals with different parameter of a

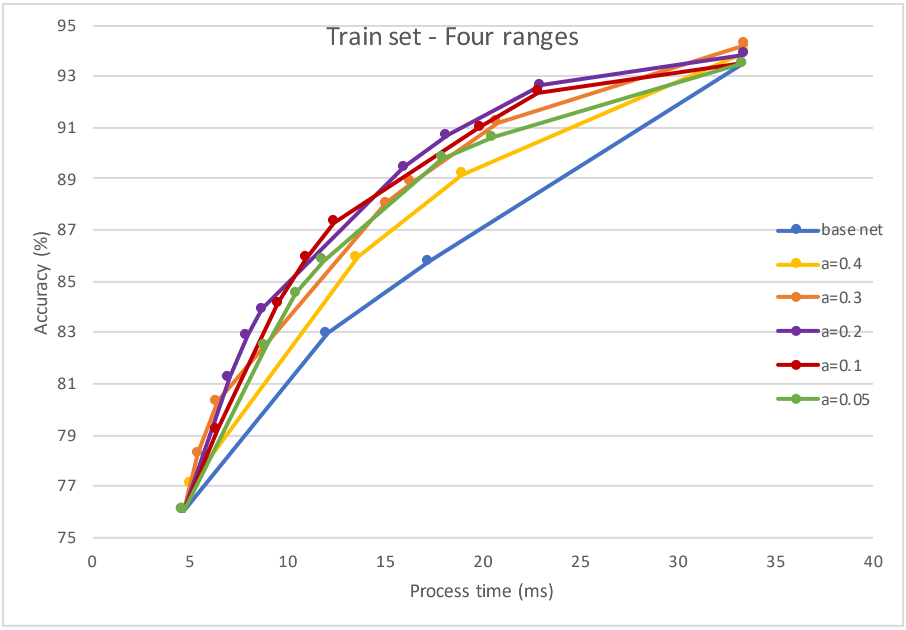
When only the accuracy is important and the penalty for image processing time is low, the selector uses all network outputs to select the appropriate class. Since most of the processing path is common between the outputs of the base network, using all outputs does not significantly increase processing time. Although the use of all outputs increased the accuracy on the training data, there is no significant increase in the accuracy on the test data. This is because most of the image processing is done in the convolution layers, our three branches consist only of fully connected layers.

4.4 Different Number of Quantization Intervals

In this section, we have increased the number of quantization intervals and measured its effect on accuracy. Below our different quantization intervals are shown.

$$3part\ model = \{R1 = [0, 0.3], R2 = (0.3, 0.7], R3 = (0.7, 1]\}, \tag{13}$$

$$4part\ model = \{R1 = [0, 0.2], R2 = (0.2, 0.5], R3 = (0.5, 0.8], R4 = (0.8, 1]\}, \tag{14}$$



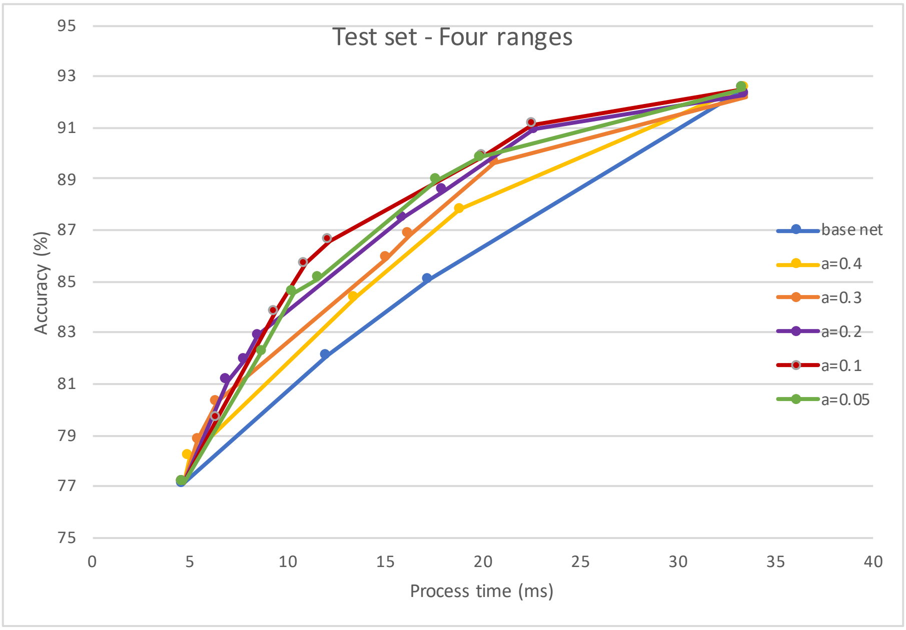


Figure 6. Accuracy versus time budget for models with four quantization intervals: The blue line shows the connection between four points related to the resulting points of the four outputs of the base network. For different values of a, the results are shown by a different color. The top chart shows the results on the train set, and the bottom chart shows the results on the test set.

$$\begin{aligned}
\textit{6part model} = \{&R1 = [0, 0.1], R2 = (0.1, 0.2], R3 = (0.2, 0.5],\\
&R4 = (0.5, 0.8], R4 = (0.8, 0.9], R4 = (0.9, 1]\}, \qquad (15)\\
\textit{8part model} = \{&R1 = [0, 0.1], R2 = (0.1, 0.2], R3 = (0.2, 0.3],\\
&R4 = (0.3, 0.5], R5 = (0.5, 0.7], R6 = (0.7, 0.8],\\
&R7 = (0.8, 0.9], R8 = (0.9, 1]\}, \qquad (16)\\
\textit{10part model} = \{&R1 = [0, 0.1], R2 = (0.1, 0.2], R3 = (0.2, 0.3], R4 = (0.3, 0.4],\\
&R5 = (0.4, 0.5], R6 = (0.5, 0.6], R7 = (0.6, 0.7],\\
&R8 = (0.7, 0.8], R9 = (0.8, 0.9], R10 = (0.9, 1]\} \qquad (17)
\end{aligned}$$

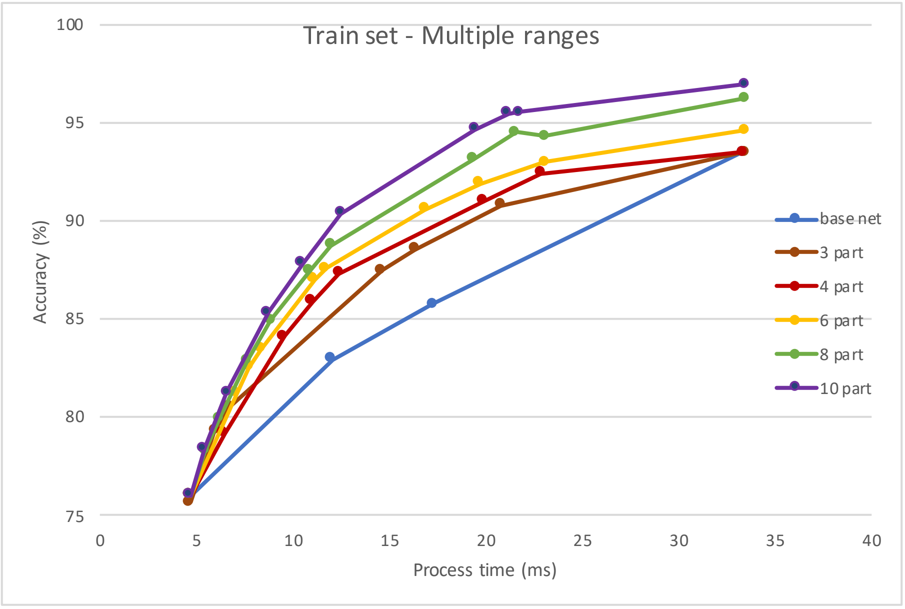
where $3\textit{part model}$, $4\textit{part model}$, $6\textit{part model}$, $8\textit{part model}$, $10\textit{part model}$ are five models that are trained with different quantization intervals. For each of these models, different divider numbers have been tested, and the value with the highest accuracy has been placed in the above equation as the selected model. The result on the test and train set for all models is shown in Figure 7. We notice that most of the points of the proposed models are above the resulting line of the base network. So our models perform better in these conditions.

In the case of 3 quantization interval, in about five milliseconds, the model's accuracy is lower than the first output of the CNN, because in this model only when the output confidence is above 70 %, the class is specified; otherwise, the output value is in the range $R2$, which does not show a specific class and only indicates that the classifier is not confident about its output. As the number of quantization intervals increases, the accuracy of the training data increases, but after increasing the intervals to more than four parts, due to the overfitting of the model to the training data, the accuracy of the test data decreases. As in the previous section, it was found that using all the outputs of the neural network in the selector does not significantly increase the accuracy on the test data, and this shows that the output of the last layer of the base neural network contains the information of the other outputs.

For a more accurate comparison of the models, we also have calculated the area under the curve of accuracy vs. time budget chart (AUC), as shown in Table 5. According to this table, the best number of quantization intervals for the model is four part quantization. The worst model is the 10 part, but even this model has performed better than the base network.

4.5 Final Model Evaluation

To verify that the final selected model performs better than the base network for all different time budgets, we calculated the precision, recall, and F1-score values for a number of different time budgets, and showed them in Table 6. In the least and most time-consuming cases, the results of our model are equal to the base network. But



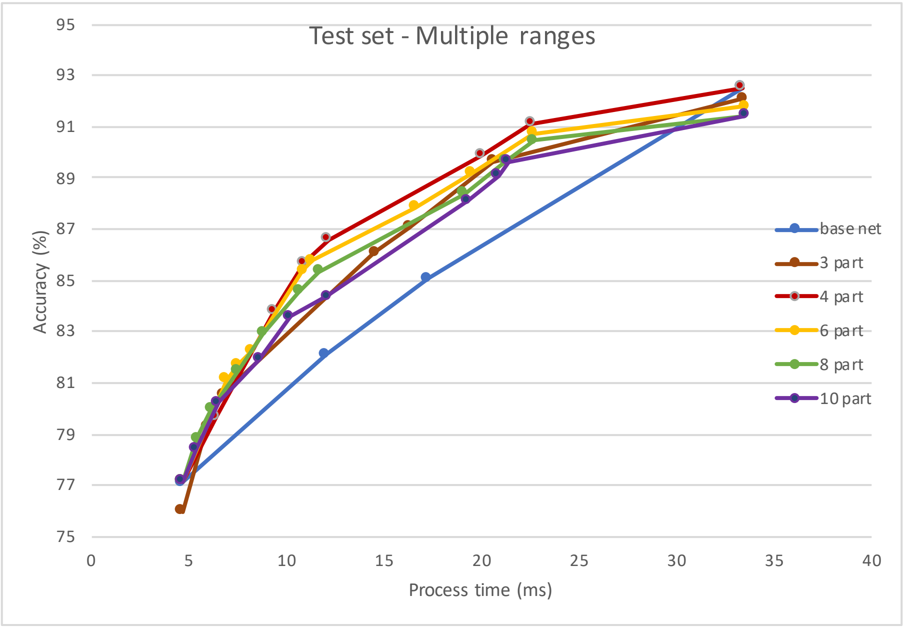


Figure 7. Accuracy versus time budget for models with different quantization intervals: The blue line shows the connection between four points related to the resulting points of the four outputs of the base network. Models by changing the quantization intervals into 3, 4, 6, 8, and 10 intervals are shown in the figure with different colors. The top chart shows the results on the train set, and the bottom chart shows the results on the test set.

Model	Area Under the Curve
base net	201.3
3 part	252.2
4 part	**279.5**
6 part	269.1
8 part	258.9
10 part	244.8

Table 5. Area under the curve for models with different quantization intervals

for all intermediate time budgets, the results are improved. For example, for 12.03 milliseconds for the base network, the precision, recall, and F1-score are 0.8141, 0.832, and 0.8229, respectively, while for the proposed model, in less time, i.e. 8.76 milliseconds, the performance is better, and the precision, recall, and F1-score are 0.819, 0.8406, and 0.8297, respectively. Therefore, as explained in the previous sections, the combination of network outputs as performed by the proposed model is able to improve the final result, in addition to the ability to produce output at the given time budgets.

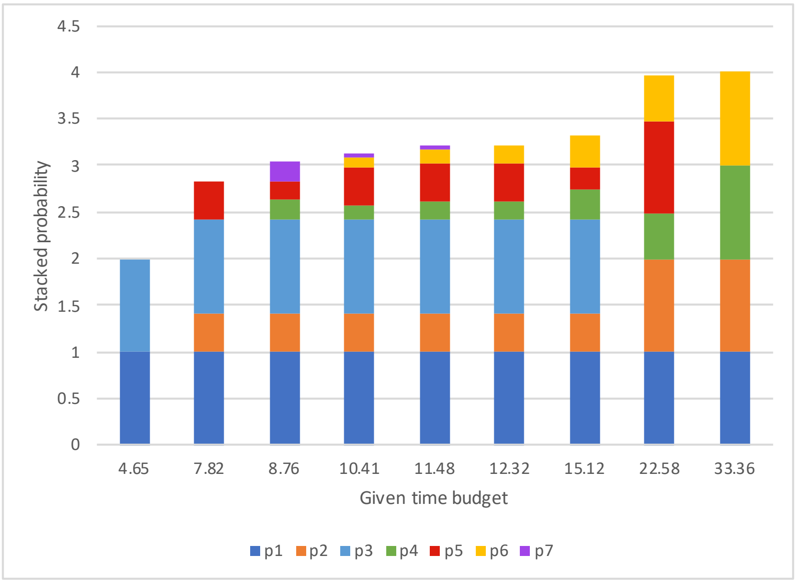
Model	Process Time (ms)	Precision	Recall	F1-score
base net	4.65	0.7602	0.7927	0.7761
	12.03	0.8141	0.832	0.8229
	17.27	0.8465	0.8567	0.8516
	33.36	0.9288	0.9213	0.925
our 4 part model	4.65	0.7603	0.793	0.7763
	8.76	0.819	0.8406	0.8297
	10.41	0.8367	0.849	0.8428
	12.32	0.8724	0.8506	0.8614
	16.55	0.8782	0.8876	0.8829
	22.58	0.9127	0.9096	0.9111
	33.36	0.9287	0.9213	0.925

Table 6. F1-score comparison for 4 part model and base model

Reduction of processing time is achieved by reducing the values of $P_l(x_i|t')$ for different parts of the CNN, in Equation (10). The first chart in Figure 8 shows the average of P_l values for different time budgets t' for all images in the test set. The second chart shows the processing time for different time budgets. P_l and t_l corresponds to a part of the CNN in Figure 3.

In Figure 8, P_3, P_5, P_7, P_6 correspond to the parts leading to the output softmax1, softmax2, softmax3, and softmax4 of the CNN, respectively. In the following, we will explain two charts of the Figure 8.

In the top chart, when the time budget is very low, only softmax1 output is used (i.e. $p1 = 1$ and $p3 = 1$). As the budget increases, the probability of using layers with higher computational cost increases. Therefore, the hypothesis of re-



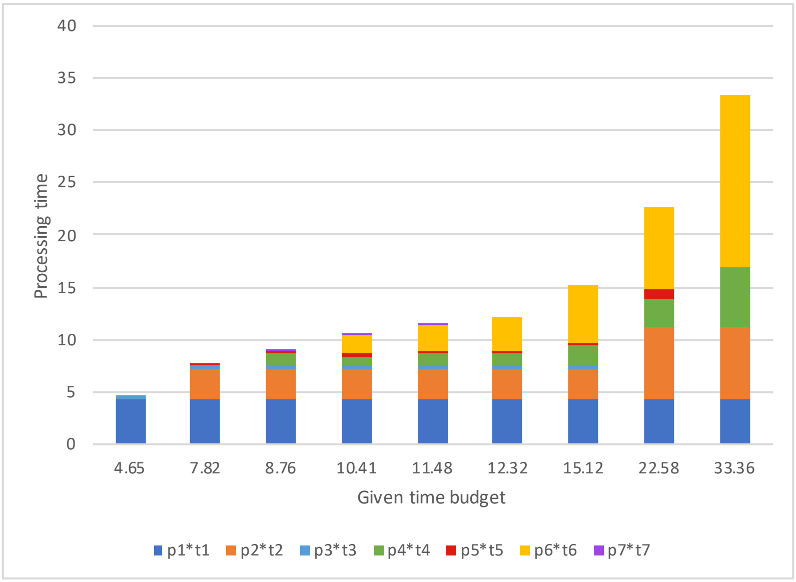


Figure 8. The top chart shows the stacked probability of P_l given different time budgets. The bottom chart shows the stacked time of each part of the CNN (i.e. $P_l * t_l$) given different time budgets.

ducing time complexity by reducing the use of layers (presented in Equation (6)) is verified.

When time budgets are less than 15.12 milliseconds, the first output (P_3) is used, but the sum of probability of using the other outputs ($P_5 + P_7 + P_6$) is less than one. That is, some images are simple and can be correctly classified using only the first output of the CNN.

The bottom chart shows that the outputs P_3, P_5, and $P7$ consume very little processing time and most of the processing is related to the middle layers. When the maximum time budget is given, the selector does not use outputs from the earlier parts of the network and only uses the last output P_6. Because the outputs are not completely independent, the latest output contains information about previous outputs, and the use of earlier outputs does not improve the result.

By increasing the time budget, the selector has used the layers with more time cost, which are also more accurate. But this use is not linear and the selector tries to use the best combination for different given budgets. For example, the selector has used softmax3 (correspond to P_7) very little, and by increasing the time budget, the selector has increased the share of softmax4 (correspond to P_6).

5 CONCLUSIONS AND FUTURE WORKS

This paper presents a method to adaptively resolve the trade-off between accuracy and time budget for adult image classification. We used a dynamic classifier selection technique. First, we created a CNN with three outputs from the middle layers, then we trained a selector using the Q-learning method to select and combine the best CNN outputs for each image based on the available time budget. The results confirm that our selector can increase the accuracy of the classifier by using the appropriate combination of network outputs. The final model also has the ability to generate results for different time budgets.

In the future, we intend to automize the quantization of the outputs of the softmax layers using a linear classifier. So, the error of manually quantizing the ranges can be eliminated. We also plan to extend this method for classifying adult videos. By considering the correlation of consecutive frames and using the appropriate selector, we could reduce the processing time of the entire video.

REFERENCES

[1] Community Guidelines, Instagram Help Center. Available at: `https://help.instagram.com/477434105621119`, July 17, 2020.

[2] Community Standards, Facebook. Available at: `https://www.facebook.com/communitystandards/objectionable_content`, July 17, 2020.

[3] Wang, L.—Zhang, J.—Tian, Q.—Li, C.—Zhuo, L.: Porn Streamer Recognition in Live Video Streaming via Attention-Gated Multimodal Deep Features. IEEE

Transactions on Circuits and Systems for Video Technology, Vol. 30, 2020, No. 12, pp. 4876–4886, doi: 10.1109/TCSVT.2019.2958871.

[4] Lykousas, N.—Gómez, V.—Patsakis, C.: Adult Content in Social Live Streaming Services: Characterizing Deviant Users and Relationships. 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2018, pp. 375–382, doi: 10.1109/ASONAM.2018.8508246.

[5] Liu, H.—Simonyan, K.—Yang, Y.: DARTS: Differentiable Architecture Search. 7th International Conference on Learning Representations (ICLR 2019), 2019, pp. 1–13.

[6] Simonyan, K.—Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. 3rd International Conference on Learning Representations (ICLR 2015). ArXiv Preprint ArXiv:1409.1556v6, 2015.

[7] He, K.—Zhang, X.—Ren, S.—Sun, J.: Deep Residual Learning for Image Recognition. Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[8] Leroux, S.—Bohez, S.—De Coninck, E.—Verbelen, T.—Vankeirsbilck, B.—Simoens, P.—Dhoedt, B.: The Cascading Neural Network: Building the Internet of Smart Things. Knowledge and Information Systems, Vol. 52, 2017, No. 3, pp. 791–814, doi: 10.1007/s10115-017-1029-1.

[9] Karayev, S.—Baumgartner, T.—Fritz, M.—Darrell, T.: Timely Object Recognition. In: Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012, pp. 890–898, `http://papers.nips.cc/paper/4712-timely-object-recognition`.

[10] Iandola, F. N.—Han, S.—Moskewicz, M. W.—Ashraf, K.—Dally, W. J.—Keutzer, K.: SqueezeNet: AlexNet-Level Accuracy with 50× Fewer Parameters and < 0.5 MB Model Size. 2016, pp. 1–13, `http://arxiv.org/abs/1602.07360`.

[11] Howard, A. G.—Zhu, M.—Chen, B.—Kalenichenko, D.—Wang, W.—Weyand, T.—Andreetto, M.—Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017, `http://arxiv.org/abs/1704.04861`.

[12] Zhang, X.—Zhou, X.—Lin, M.—Sun, J.: ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856, doi: 10.1109/CVPR.2018.00716.

[13] Ma, N.—Zhang, X.—Zheng, H.-T.—Sun, J.: ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (Eds.): Computer Vision – ECCV 2018. Springer, Cham, Lecture Notes in Computer Science, Vol. 11218, 2018, pp. 122–138, doi: 10.1007/978-3-030-01264-9_8.

[14] Berestizshevsky, K.—Even, G.: Dynamically Sacrificing Accuracy for Reduced Computation: Cascaded Inference Based on Softmax Confidence. In: Tetko, I. V., Kůrková, V., Karpov, P., Theis, F. (Eds.): Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning. Springer, Cham, Lecture Notes in Computer Science, Vol. 11728, 2019, pp. 306–320, 2019, doi: 10.1007/978-3-030-30484-3_26.

[15] Liu, L.—Deng, J.: Dynamic Deep Neural Networks: Optimizing Accuracy-Efficiency Trade-Offs by Selective Execution. 32nd AAAI Conference on Artificial Intelligence (AAAI-18), 2018, pp. 3675–3682.

[16] Lin, J.—Rao, Y.—Lu, J.—Zhou, J.: Runtime Neural Pruning. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.): Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017, pp. 2182–2192.

[17] Rao, Y.—Lu, J.—Lin, J.—Zhou, J.: Runtime Network Routing for Efficient Image Classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 41, 2019, No. 10, pp. 2291–2304, doi: 10.1109/TPAMI.2018.2878258.

[18] Zheng, H.—Daoudi, M.—Jedynak, B.: Blocking Adult Images Based on Statistical Skin Detection. In Electronic Letters on Computer Vision and Image Analysis, Vol. 4, 2004, No. 2, 14 pp., doi: 10.5565/rev/elcvia.78.

[19] Zheng, H.—Liu, H.—Daoudi, M.: Blocking Objectionable Images: Adult Images and Harmful Symbols. 2004 IEEE International Conference on Multimedia and Expo (ICME), Vol. 2, 2004, pp. 1223–1226, doi: 10.1109/icme.2004.1394442.

[20] Lee, J. S.—Kuo, Y. M.—Chung, P. C.: The Adult Image Identification Based on Online Sampling. Proceedings of the 2006 IEEE International Joint Conference on Neural Network, 2006, pp. 2566–2571, doi: 10.1109/IJCNN.2006.247111.

[21] Zheng, Q. F.—Zhang, M. J.—Wang, W. Q.: A Hybrid Approach to Detect Adult Web Images. In: Aizawa, K., Nakamura, Y., Satoh, S. (Eds.): Advances in Multimedia Information Processing – PCM 2004. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3332, 2004, pp. 609–616, doi: 10.1007/978-3-540-30542-2_75.

[22] Zhu, H.—Zhou, S.—Wang, J.—Yin, Z.: An Algorithm of Pornographic Image Detection. Proceedings of the 4th International Conference on Image and Graphics (ICIG 2007), 2007, pp. 801–804, doi: 10.1109/ICIG.2007.29.

[23] Zheng, Q. F.—Zeng, W.—Wang, W. Q.—Gao, W.: Shape-Based Adult Image Detection. International Journal of Image and Graphics, Vol. 6, 2006, No. 1, pp. 115–124, doi: 10.1142/S0219467806002082.

[24] Lopes, A. P. B.—de Avila, S. E. F.—Peixoto, A. N. A.—Oliveira, R. S.—De A. Araújo, A.: A Bag-of-Features Approach Based on Hue-SIFT Descriptor for Nude Detection. 2009 17th European Signal Processing Conference (EUSIPCO), Glasgow, UK, 2009, pp. 1552–1556, https://ieeexplore.ieee.org/abstract/document/7077625/.

[25] Deselaers, T.—Pimenidis, L.—Ney, H.: Bag-of-Visual-Words Models for Adult Image Classification and Filtering. 2008 19th International Conference on Pattern Recognition, 2008, pp. 1–4, doi: 10.1109/ICPR.2008.4761366.

[26] Chatfield, K.—Simonyan, K.—Vedaldi, A.—Zisserman, A.: Return of the Devil in the Details: Delving Deep into Convolutional Nets. Proceedings of the British Machine Vision Conference 2014, 2014, pp. 6.1–6.12, doi: 10.5244/C.28.6.

[27] Moustafa, M.: Applying Deep Learning to Classify Pornographic Images and Videos. Proceedings of the 7th Pacific-Rim Symposium on Image and Video Technology (PSIVT 2015), Auckland, New Zealand, 2015, http://arxiv.org/abs/1511.08899.

[28] Krizhevsky, A.—Sutskever, I.—Hinton, G. E.: Imagenet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012, pp. 1097–1105, `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks`.

[29] Szegedy, C.—Liu, W.—Jia, Y.—Sermanet, P.—Reed, S.—Anguelov, D.—Erhan, D.—Vanhoucke, V.—Rabinovich, A.: Going Deeper with Convolutions. Proceedings of the 2015 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[30] Nian, F.—Li, T.—Wang, Y.—Xu, M.—Wu, J.: Pornographic Image Detection Utilizing Deep Convolutional Neural Networks. Neurocomputing, Vol. 210, 2016, pp. 283–293, doi: 10.1016/j.neucom.2015.09.135.

[31] Shen, R.—Zou, F.—Song, J.—Yan, K.—Zhou, K.: EFUI: An Ensemble Framework Using Uncertain Inference for Pornographic Image Recognition. Neurocomputing, Vol. 322, 2018, pp. 166–176, doi: 10.1016/j.neucom.2018.08.080.

[32] Vitorino, P.—Avila, S.—Perez, M.—Rocha, A.: Leveraging Deep Neural Networks to Fight Child Pornography in the Age of Social Media. Journal of Visual Communication and Image Representation, Vol. 50, 2018, pp. 303–313, doi: 10.1016/j.jvcir.2017.12.005.

[33] Deng, J.—Dong, W.—Socher, R.—Li, L. J.—Li, K.—Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2009), 2009, pp. 248–255, doi: 10.1109/CVPR.2009.5206848.

[34] Cheng, F.—Wang, S. L.—Wang, X. Z.—Liew, A. W. C.—Liu, G. S.: A Global and Local Context Integration DCNN for Adult Image Classification. Pattern Recognition, Vol. 96, 2019, Art. No. 106983, doi: 10.1016/j.patcog.2019.106983.

[35] Bolukbasi, T.—Wang, J.—Dekel, O.—Saligrama, V.: Adaptive Neural Networks for Efficient Inference. 34th International Conference on Machine Learning (ICML 2017), PMLR, Vol. 70, 2017, pp. 527–536.

[36] Bengio, E.—Bacon, P.-L.—Pineau, J.—Precup, D.: Conditional Computation in Neural Networks for Faster Models. 2015, `http://arxiv.org/abs/1511.06297`.

[37] Odena, A.—Lawson, D.—Olah, C.: Changing Model Behavior at Test-Time Using Reinforcement Learning. Proceedings of the 5th International Conference on Learning Representations (ICLR 2017), Workshop Track, 2017, pp. 1–6, `https://arxiv.org/pdf/1702.07780.pdf`.

[38] Vedaldi, A.—Lenc, K.: MatConvNet: Convolutional Neural Networks for MATLAB. Proceedings of the 23rd ACM International Conference on Multimedia (MM 2015), 2015, pp. 689–692, doi: 10.1145/2733373.2807412.

[39] Cruz, R. M. O.—Sabourin, R.—Cavalcanti, G. D. C.: Dynamic Classifier Selection: Recent Advances and Perspectives. Information Fusion, Vol. 41, 2018, pp. 195–216, doi: 10.1016/j.inffus.2017.09.010.

[40] Noktedan, A.: Adult Content Dataset. Figshare, 20-Dec-2020, doi: 10.6084/m9.figshare.13456484.v1.

[41] Watkins, C. J. C. H.—Dayan, P.: Q-Learning. Machine Learning, Vol. 8, 1992, pp. 279–292, doi: 10.1007/BF00992698.

Mohammad Reza Mazinani is Ph.D. candidate in computer engineering, Malek Ashtar University of Technology, Iran. He received his M.Sc. degree in artificial intelligence from the Computer Engineering Department, Sharif University of Technology, Iran. His research interests are deep learning, cost-sensitive computing, image and video processing, and pattern recognition.

Seyyed Mojtaba Hoseini received his B.Sc. degree in electronic engineering from Malek Ashtar University of Technology in 1991. He also received his M.Sc. and Ph.D. degrees in computer architecture engineering from Amirkabir University of Technology in 1995 and 2011, respectively. His research interest is in wireless sensor networks, with an emphasis on target coverage and tracking applications, image and signal processing, and evolutionary computing.

Kourosh Dadashtabar Ahmadi is Chairman of Computer and Artificial Intelligence Research Institute, Head of AI group, and Assistant Professor in Malek Ashtar University of Technology, Iran. He received his M.Sc. in information technology from Tarbiat Modares University in 2007, and Ph.D. in artificial intelligence from Malek Ashtar University of Technology, Iran in 2015. His research interests are information fusion, cyber security, image processing, deep learning, and reinforcement learning.

Computing and Informatics, Vol. 39, 2020, 1311–1335, doi: 10.31577/cai_2020_6_1311

DECODING FIVE TIMES EXTENDED REED SOLOMON CODES USING SYNDROMES

Peter Farkaš

Institute of Multimedia ICT, FEI STU
Ilkovičova 3
812 19 Bratislava, Slovakia
&
Institute of Applied Informatics
Faculty of Informatics Pan-European University, Slovakia
Tematínska 10
851 05 Bratislava, Slovakia
e-mail: p.farkas@ieee.org

Martin Rakús

Institute of Multimedia ICT, FEI STU
Ilkovičova 3
812 19 Bratislava, Slovakia
e-mail: martin.rakus@stuba.sk

Abstract. Recently a new family of five times extended Reed Solomon codes constructed over certain finite fields $GF(2^\zeta)$, where $\zeta \geq 3$ is an odd integer, was discovered. Until now only an erasure decoding algorithm for these codes was published. In this paper a new decoding algorithm is presented, which allows correcting up to two errors in a codeword from the five times extended Reed Solomon codes. The proposed decoding algorithm is based on syndrome usage.

Keywords: Algorithm, extended Reed Solomon codes, error correction decoding, syndromes

Mathematics Subject Classification 2010: 58F15, 58F17, 53C35

1 INTRODUCTION

Reed Solomon (RS) codes were discovered more than 60 years ago [3]. Their practical importance was proven by mass applications in such systems as CD or DVB [4]. More surprisingly, in recent years a new interest in these codes has risen in the research community. It is caused by the fact that they are extensively used in cloud technology, namely in distributed storage systems [5, 6, 7, 8, 9]. In [1] it was shown that RS codes could be extended not only three times, but even five times when they are constructed over certain finite fields. For such five times extended RS codes a decoding algorithm is known [2] which is suitable only for erasure decoding. In case of erasures the positions are known in a received vector and therefore only the erasure values have to be found out during decoding. In contrast to this, the positions of the errors in received vectors are unknown; therefore both error values and error positions have to be determined during decoding. The error correction is useful in many applications of RS codes [4]. For example, in [10] syndrome decoding is patented for random access-based computer memory systems by IBM Corp. Interestingly, it contains a reference to [11] in which the authors of this paper proposed a related code which inspired the construction of five times extended Reed Solomon codes [1].

In this paper a decoding algorithm is presented which allows correcting at least 2 errors in five times extended RS codes from [1].

The paper is organized as follows. In Section 2 a short introduction to the original (not extended) RS codes error correcting decoding via syndromes is presented. In Section 3 the error correcting decoding of the original RS codes is given. In Section 4 the novel error correcting algorithm for five times extended RS codes introduced in [1] is presented. In Section 5 some remarks on the decoding implementation are given. In Section 6 the complexity estimation of the algorithm is made. Conclusions in Section 7 summarize the results of the paper.

2 A SHORT INTRODUCTION TO ORIGINAL RS CODES

RS codes are linear block codes which could be described by many mathematical tools and which have many interesting connections with different mathematical branches. In this section only a short introduction to RS codes is given. The interested reader could find a more detailed explanation of RS codes and their applications for example in [4]. To explain the proposed decoding algorithm a basic description of RS codes as linear block codes via matrices and as cyclic codes using polynomials is given.

A linear block code is defined as a k-dimensional subspace of an n-dimensional vector space over a finite field $GF(q)$. Basic parameters of linear block codes are: the codeword length n, the number of information symbols k in each codeword and the code distance d_m which is a minimal Hamming distance between any two codewords. Sometimes linear block codes are in shorthand using a triple $[n, k, d_m]$, which contains these fundamental parameters. The code distance d_m and the number of correctable errors t in a linear code are connected by the following inequality:

$$2t + 1 \leq d_m.$$

Because a linear block code is defined as a k-dimensional subspace of an n-dimensional vector space over a finite field $GF(q)$ it could be described via a $k \times n$ generator matrix $\mathbf{G}_{k\times n}$ which contains as rows k linearly independent vectors with length n. In systematic form:

$$\mathbf{G}_{k\times n} = \left[\mathbf{I}_{k\times k} \mid \mathbf{P}_{k\times(n-k)}\right] \tag{1}$$

where $\mathbf{I}_{k\times k}$ and $\mathbf{P}_{k\times(n-k)}$ are the identity and parity matrices, respectively. For decoding purposes a control matrix is defined as:

$$\mathbf{H}_{(n-k)\times n} = \left[\mathbf{P}^T_{(n-k)\times k} \mid \mathbf{I}_{(n-k)\times(n-k)}\right] \tag{2}$$

where $\mathbf{I}_{(n-k)\times(n-k)}$ and $\mathbf{P}^T_{(n-k)\times k}$ are the identity and transposed parity matrices, respectively. The following equation is valid:

$$\mathbf{G}_{k\times n}.\mathbf{H}^T_{n\times(n-k)} = \mathbf{0}_{k\times(n-k)} \tag{3}$$

where $\mathbf{H}^T_{n\times(n-k)}$ and $\mathbf{0}_{k\times(n-k)}$ are the transposed matrix $\mathbf{H}$ and all zeros matrix, respectively.

In [1] a new family of codes constructed over $GF(2^\zeta)$, where $\zeta \geq 3$ is an odd integer, using the $\mathbf{H}$ matrix (4) was proposed.

$$\mathbf{H} = \begin{bmatrix} \alpha^0 & \alpha^0 & \cdots & \alpha^0 & \alpha^0 & \alpha^0 & 1 & 0 & 0 & 0 & 0 \\ \alpha^{(q-2)} & \alpha^{(q-3)} & \cdots & \alpha^2 & \alpha^1 & \alpha^0 & 0 & 1 & 0 & 0 & 0 \\ \alpha^{2(q-2)} & \alpha^{2(q-3)} & \cdots & \alpha^4 & \alpha^2 & \alpha^0 & 0 & 0 & 1 & 0 & 0 \\ \alpha^{3(q-2)} & \alpha^{3(q-3)} & \cdots & \alpha^6 & \alpha^3 & \alpha^0 & 0 & 0 & 0 & 1 & 0 \\ \alpha^{4(q-2)} & \alpha^{4(q-3)} & \cdots & \alpha^8 & \alpha^4 & \alpha^0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

The basic parameters of this family of block codes can be characterized by a triple $[q+4, q-1, 5]$.

3 ERROR CORRECTING DECODING OF ORIGINAL RS CODES

The definition of original RS codes correcting t errors requires that each codeword of these cyclic codes has $2t$ consecutive and distinct elements of a finite field as roots. In other words the generating polynomial should also have the same property. In $GF(2^\zeta)$, $\zeta \geq 2$ we get:

$$g(x) = \left(x + \alpha^j\right)\left(x + \alpha^{j+1}\right)\left(x + \alpha^{j+2}\right)\ldots\left(x + \alpha^{j+2t-1}\right) \tag{5}$$

where j is an integer (for convenience usually equal to 0 or 1). In the following explanation we will use $j = 0$ for simplicity. Then the generator polynomial can be written as follows:

$$g(x) = \left(x+\alpha^0\right)\left(x+\alpha^1\right)\left(x+\alpha^2\right)\dots\left(x+\alpha^{2t-1}\right). \tag{6}$$

The original RS codes could be described using the following control matrix:

$$\mathbf{H}_{RS} = \begin{bmatrix} \alpha^0 & \alpha^0 & \cdots & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^{(q-2)} & \alpha^{(q-3)} & \cdots & \alpha^2 & \alpha^1 & \alpha^0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ \alpha^{(2t-1)(q-2)} & \alpha^{(2t-1)(q-3)} & \cdots & \alpha^{(2t-1)2} & \alpha^{(2t-1)} & \alpha^0 \end{bmatrix} \tag{7}$$

which is in fact a Vandermonde matrix $\mathbf{V}_{2t\times n}$ over $GF(2^\zeta)$.

There are numerous different decoding algorithms for original RS codes. In this section, we will describe only the basic algorithm for error correction of codewords of RS codes based on syndromes. The reason is that this algorithm is the most related to the proposed decoding algorithm for the five times extended RS codes, which will be described later.

The main goal of error correcting codes and their decoding is to decrease the number of errors and/or erasures which can occur during the information transmission or storage. Next, we will focus on error correction. The codewords of RS codes can be described as vectors in which the coordinates are symbols from the underlying finite field:

$$\mathbf{c} = (c_{n-1}, c_{n-2}, \dots, c_1, c_0). \tag{8}$$

Another possibility is to use polynomials for descriptions:

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_1x^1 + c_0x^0. \tag{9}$$

The symbols c_i in (8) and 9 are elements from $GF(q)$. In the present digital era the most practical choice are finite fields $GF(2^\zeta)$, which are extensions of the binary finite field. $\zeta \geq 2$ is positive integer.

In such fields the i^{th} error in a codeword could be described using two unknowns – the so called error value $Y_i \in GF(2^\zeta)$ and its position, which is given by a corresponding error locator $X_i \in GF(2^\zeta)$. The decoder needs to calculate both of these values for each error in order to correct one error in a codeword. On the other hand, each polynomial 9) representing a codeword from the RS code has $2t$ roots which are consecutive elements in $GF(2^\zeta)$. This allows the formation of $2t$ equations in order to correct t errors in one codeword from the original RS code. The following explanation and example of the basic decoding algorithm makes it more obvious.

The model of an additive error channel is illustrated in Figure 1.

Because for any codeword of a cyclic code the following two equations hold:

$$c(x) \,\texttt{mod}\, g(x) = 0 \tag{10}$$

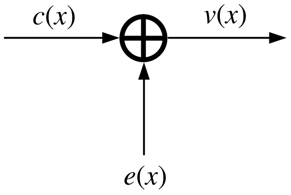


Figure 1. Additive error channel model. $\oplus$ denotes a polynomial addition over $GF(2^\varsigma)$, $c(x)$ is the transmitted codeword, $e(x)$ is an error polynomial and $v(x)$ is the received polynomial which can contain errors.

and

$$g(\alpha^k) = 0; \quad k = j, (j+1), \ldots, (j+2t-1) \tag{11}$$

then the following equation is also valid:

$$c(\alpha^k) = 0; \quad k = j, (j+1), \ldots, (j+2t-1). \tag{12}$$

One of the many algorithms which are known for error correcting decoding for original RS codes is the following syndrome method. The first step consists of inserting the roots, which define the concrete RS code via (5) into the received polynomial $v(x) = v_{n-1}x^{n-1} + v_{n-2}x^{n-2} + \ldots + v_1x^1 + v_0x^0$:

$$S_k = v(\alpha^k) = c(\alpha^k) + e(\alpha^k); \quad k = j, (j+1), \ldots, (j+2t-1). \tag{13}$$

If $j = 0$:

$$\begin{aligned} S_0 &= v(\alpha^0), \\ S_1 &= v(\alpha^1), \\ &\vdots \\ S_{2t-1} &= v(\alpha^{2t-1}) \end{aligned} \tag{14}$$

where S_j denotes the j^{th} syndrome, $j = 0, 1, \ldots, 2t-1$. From (12) and (13):

$$S_k = e(\alpha^k); \quad k = j, (j+1), \ldots, (j+2t-1). \tag{15}$$

Therefore each syndrome is dependent only on the error polynomial written as $e(x) = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + \ldots + e_1x^1 + e_0x^0$ and it could be expressed also via the unknowns which have to be calculated in order to correct the errors. Namely the error locators X_i; $i = 0, 1, 2, \ldots, t$ and error values Y_i; $i = 0, 1, 2, \ldots, t$:

$$S_k = \sum_{i=1}^{t} Y_i X_i^k; \quad k = 0, 1, \ldots, 2t-1. \tag{16}$$

(At this stage it is not known how many errors τ occurred in reality, but we suppose that $\tau \leq t$ and so we start with the worst case assumption, that $\tau = t$.)

Actually (16) is a system of nonlinear equations which could not be solved directly. It could be better seen after writing (16) in a more detailed fashion:

$$\begin{array}{ccccccccc} Y_1 & + & Y_2 & \cdots & + & Y_t & = & S_0, \\ Y_1X_1 & + & Y_2X_2 & \cdots & + & Y_tX_t & = & S_1, \\ Y_1X_1^2 & + & Y_2X_2^2 & \cdots & + & Y_tX_t^2 & = & S_2, \\ & & & & \vdots & & & \\ Y_1X_1^{2t-1} & + & Y_2X_2^{2t-1} & \cdots & + & Y_tX_t^{2t-1} & = & S_{2t-1}. \end{array} \tag{17}$$

A clever way to deal with this difficulty is to first find the error locators and then after introducing them into (16) the system turns into a system of linear equations which could be solved by standard algorithms, for example Gauss elimination.

The error locator polynomial could be used for this purpose. The error locator polynomial: $\lambda(x) = \lambda_t x^t + \lambda_{t-1}x^{t-1} + \ldots + \lambda_1 x + 1$ is a polynomial which has roots, which are error locators. It could be expressed as follows:

$$\lambda(X_i) = 0 \quad i = 0, 1, \ldots, t. \tag{18}$$

In order to find an error locator polynomial we can rewrite (18):

$$\lambda_t X_i^t + \lambda_{t-1}X_i^{t-1} + \ldots + \lambda_1 X_i + 1 = 0 \quad i = 0, 1, \ldots, t. \tag{19}$$

(19) could be multiplied by $Y_iX_i^z$; $i = 1, 2, \ldots, t$; $z \in Z$:

$$\lambda_t Y_i X_i^{t+z} + \lambda_{t-1} Y_i X_i^{t+z-1} + \ldots + \lambda_1 Y_i X_i^z + Y_i X_i^z = 0 \quad i = 0, 1, \ldots, t. \tag{20}$$

Now we can write (18) for fixed values of $z \in Z$. For $z = 0$ we get:

$$\begin{array}{ccccccccc} Y_1X_1^0 & + & \lambda_1Y_1X_1^1 & + & \cdots & + & \lambda_tY_1X_1^t & = & 0, \\ Y_2X_2^0 & + & \lambda_1Y_2X_2^1 & + & \cdots & + & \lambda_tY_2X_2^t & = & 0, \\ & & & & & & & \vdots & \\ Y_tX_t^0 & + & \lambda_1Y_tX_t^1 & + & \cdots & + & \lambda_tY_tX_t^t & = & 0. \end{array} \tag{21}$$

Summation of (21) gives us:

$$S_0 + \lambda_1 S_1 + \ldots + \lambda_t S_t = 0. \tag{22}$$

Similarly for $z = 1$ we get:

$$\begin{array}{ccccccccc} Y_1X_1^1 & + & \lambda_1Y_1X_1^2 & + & \cdots & + & \lambda_tY_1X_1^{t+1} & = & 0, \\ Y_2X_2^1 & + & \lambda_1Y_2X_2^2 & + & \cdots & + & \lambda_tY_2X_2^{t+1} & = & 0, \\ & & & & & & & \vdots & \\ Y_tX_t^1 & + & \lambda_1Y_tX_t^2 & + & \cdots & + & \lambda_tY_tX_t^{t+1} & = & 0. \end{array} \tag{23}$$

Summation of (23) gives us:

$$S_1 + \lambda_1 S_2 + \ldots + \lambda_t S_{t+1} = 0. \tag{24}$$

We can continue in a similar way until we get the following system of equations:

$$\begin{array}{ccccccccc} S_0 & + & \lambda_1 S_1 & + & \cdots & + & \lambda_t S_t & = & 0, \\ S_1 & + & \lambda_1 S_2 & + & \cdots & + & \lambda_t S_{t+1} & = & 0, \\ & & & & & & & \vdots & \\ S_{t-1} & + & \lambda_1 S_t & + & \cdots & + & \lambda_t S_{2t-1} & = & 0. \end{array} \tag{25}$$

This system of linear equations could be solved by any standard method, for example by Gauss-Jordan elimination. As a result we get the coefficients of the error locator polynomial.

The error locator polynomial determination from the calculated syndromes could also be done using different approaches, for example the Berlekamp-Massey algorithm [12, 13].

In the third step the locators are found via Chien search [14]. It is, in principle, a slightly augmented brute force algorithm in which the symbols are inserted consecutively into the locator polynomial, evaluated and tested if:

$$\lambda(X_i) = 0 \quad i = 0, 1, \ldots, \tau; \quad \tau \leq t \tag{26}$$

until we get τ locators. This algorithm is possible because the underlying field is finite. It will stop when we get enough solutions. In other words it means that with high probability we do not usually have to insert all nonzero elements.

Nevertheless, the Chien search has high complexity. If the error locator has a small degree it could be solved directly [15, 16, 17]. Or theorem 3.2.15 in [18] can be used with much smaller computational complexity. For example the method in [16] has very low complexity.

The result of this step will be the set of error locators: X_i; $i = 0, 1, \ldots, \tau$, where τ is the number of errors which actually occurred during transmission in the decoded codeword.

The fourth step is the calculation of error values. This can be done thanks to known syndromes, which could be expressed in another form:

$$S_k = \sum_{i=1}^{\tau} Y_i X_i^k \quad k = 0, 1, \ldots, 2t-1. \tag{27}$$

In this step it remains to calculate the error values Y_i; $i = 0, 1, \ldots, \tau$. This could be accomplished simply by solving the set of equation given by (27). At first we have to insert the values of error locators into it which will transform it into a set of linear equations. (The error locator values are elements of the finite field and therefore also their powers are elements of the same finite field.) Then this set of linear equations could be solved by the Sarus rule or by Gauss-Jordan elimination.

In this step the attempt to correct the received word could be done by adding the error values to received symbols in positions determined by error locators.

One may ask how the actual number of errors $\tau \leq t$ could be obtained in the third step of the above algorithm. When using Gauss-Jordan elimination it is quite straightforward. When using the Sarus rule we have to test if the determinant of the system is zero. We have to suppose that $\tau = t$ first. If the determinant of the matrix corresponding to the system of equations is zero, we will have to suppose that $\tau = t - 1$ and again compute the new determinant and test if it is zero and if not then we can conclude that $\tau = t - 1$. If it is zero we can continue to decrease τ and modify the corresponding system matrix by deleting the last row and last column until we get a nonzero determinant. It has to be said that there exists a much more efficient method to find the error locator, namely the Berlecamp Massey algorithm. The details can be found in [12, 13].

The last step is obtaining the estimation of the transmitted or stored codeword from the received word – the actual error correction of the received word. This is done by adding the error values to the positions determined by error locators in the received codeword.

4 A DECODING ALGORITHM FOR ERROR CORRECTION FOR FIVE TIMES EXTENDED RS CODES

In [1] it was proven that the code distance of each code from this family is 5. This code distance potentially allows for correcting up to two errors in a codeword. This is a necessary but insufficient condition. The additional condition which has to be fulfilled is the knowledge of a decoding algorithm.

Unfortunately the approach presented in the previous section is not applicable to codes proposed in [1]. It is because (4) contains not only a Vandermonde matrix as a submatrix, but also an additional identity matrix in juxtaposition. It becomes obvious by inspection of the matrix (4), which can be represented in a compact form as:

$$\mathbf{H}_{5\times n} = \left[\mathbf{V}_{5\times(q-1)} \mid \mathbf{I}_{5\times 5}\right]. \tag{28}$$

Therefore in this section a new specialized error correcting algorithm will be presented for codes from [1]. The main problem is that the classical syndrome method in this case is not able to distinguish between different error patterns by analyzing the values of syndromes. The approach which will overcome this difficulty is similar to using sieves for mechanical separation by size. In other words the algorithm deals first with error patterns detectable by syndromes. The rest of the error patterns are processed at the end of decoding.

At first we will introduce notation which will allow us to underline some properties, which will be exploited in the decoding algorithm. The codewords of the codes from [1] will be denoted as follows:

$$\mathbf{c} = (c_{k-1}, c_{k-2}, \ldots, c_1, c_0, p_0, p_1, p_2, p_3, p_4) \tag{29}$$

where the left part contains information vector: $\mathbf{c} = (m_{k-1}, m_{k-2}, \ldots, m_1, m_0)$, or in other words:

$$c_i = m_i \quad i = 0, 1, \ldots, k-1 \tag{30}$$

and

$$p_I = \sum_{i=0}^{k-1} m_i \alpha^I; \quad I = 0, 1, \ldots, 4. \tag{31}$$

It is obvious that (30) and (31) could also be used for systematic encoding of codes from [1]. The received vector which has to be decoded and therefore can contain errors will be denoted as follows:

$$\mathbf{v} = (v_{k-1}, v_{k-2}, \ldots, v_1, v_0, r_0, r_1, r_2, r_3, r_4). \tag{32}$$

The decoding algorithm starts with the calculation of 5 syndromes via multiplying the received vector by the transposed control matrix (4):

$$\mathbf{s} = \mathbf{v}.\,\mathbf{H}^T_{5\times n}. \tag{33}$$

The resulting vector will contain, as its coordinates, the desired syndromes:

$$\mathbf{s} = (S_0, S_1, S_2, S_3, S_4). \tag{34}$$

The second step consists of analysing $\mathbf{s}$. The algorithm will continue in depending on this analysis.

1. If all coordinates of $\mathbf{s}$ are zeros:

$$\mathbf{s} = (0, 0, 0, 0, 0) \tag{35}$$

   the algorithm ends, and it is supposed that no errors occurred, therefore:

$$\hat{\mathbf{c}} = \mathbf{v} \tag{36}$$

   where $\hat{\mathbf{c}}$ is the estimation of the transmitted or stored codeword.

2. If only one syndrome S_I (one coordinate) is nonzero and all other four are zeros, then it is supposed that only one error occurred in the parity symbol corresponding to the position of the nonzero element in $\mathbf{s}$. This symbol can be corrected simply by calculating the corresponding parity symbol again using the received non corrupted symbols corresponding to information symbols. Assuming that $S_I \neq 0$ then $S_I = Y$ and:

$$\hat{\mathbf{p}}_I = S_I + r_I. \tag{37}$$

   For the other coordinates of the decoded word (36) is valid.

3. If only two syndromes e.g. $S_I \neq 0$ and $S_J \neq 0$ and all other are zeros, then it is supposed that only two errors occurred and only in the parity positions. Otherwise, if one error is in the parity part and the other in the information part or both errors are in the information part, or only one error occurred in the information part then at least one other syndrome would be nonzero. This follows from the properties of Vandermonde matrices. In this case the two corresponding parity symbols could be computed similarly as in case 2. Now $S_I \neq 0 \Longrightarrow S_I = Y_1$ and $S_J \neq 0 \Longrightarrow S_J = Y_2$ then:

$$\begin{aligned} \hat{\mathbf{p}}_I &= S_I + r_I, \\ \hat{\mathbf{p}}_J &= S_J + r_J. \end{aligned} \tag{38}$$

Note: In the following steps a situation when one or two errors can occur in the information part, or one error can occur in the information part and one in the parity part, has to be investigated and decided. For these tests a set of auxiliary variables is computed:

$$\begin{aligned} A_1 &= \tfrac{S_1}{S_0}, \\ A_2 &= \tfrac{S_2}{S_1}, \\ A_3 &= \tfrac{S_3}{S_2}, \\ A_4 &= \tfrac{S_4}{S_3}. \end{aligned} \tag{39}$$

In connection with (39) possible division by 0 has to be taken into account in any practical implementation e.g. by substituting a value not present in the given $GF(q)$.

4. At first let us assume that one error occurred in the information part. In this case all of the following equations must be fulfilled:

$$\begin{aligned} S_0 &= Y, \\ S_1 &= XY, \\ S_2 &= X^2Y, \\ S_3 &= X^3Y, \\ S_4 &= X^4Y. \end{aligned} \tag{40}$$

From (40) follows that:

$$A_1 = A_2 = A_3 = A_4 = X = \alpha^j \tag{41}$$

where j gives the position of the error in the information part to which the error value $Y = S_0$ (from (40)) has to be added in order to correct the received word:

$$\hat{c}_j = v_j + Y. \tag{42}$$

5. If in one or in two cases $A_i \neq A_{i+1}$, $i = 1, 2, 3$ then one error occurred in the information part and one in the parity part. In order to test in which position in the parity part the error occurred, one of the following 5 sets of equations is valid. In order to determine the error position in the parity part one of the following tests has to hold.

   (a) If $J = 0$ then:

$$\begin{aligned} S_0 &= Y_1 + Y_2, \\ S_1 &= X_1 Y_1, \\ S_2 &= X_1^2 Y_1, \\ S_3 &= X_1^3 Y_1, \\ S_4 &= X_1^4 Y_1. \end{aligned} \tag{43}$$

   Test if the following is true:

$$A_1 \neq A_2 \wedge A_2 = A_3 \wedge A_3 = A_4. \tag{44}$$

   From (43) follows that:

$$\begin{aligned} X_1 &= \tfrac{S_2}{S_1} = \alpha^j, \\ X_2 &= \alpha^0, \\ Y_1 &= \tfrac{S_1^2}{S_2}, \\ Y_2 &= S_0 + \tfrac{S_1^2}{S_2}. \end{aligned} \tag{45}$$

   (b) If $J = 1$ then:

$$\begin{aligned} S_0 &= Y_1, \\ S_1 &= X_1 Y_1 + Y_2, \\ S_2 &= X_1^2 Y_1, \\ S_3 &= X_1^3 Y_1, \\ S_4 &= X_1^4 Y_1. \end{aligned} \tag{46}$$

   Test if the following is true:

$$A_1 \neq A_2 \wedge A_2 \neq A_3 \wedge A_3 = A_4. \tag{47}$$

   From (46) follows that:

$$\begin{aligned} X_1 &= \tfrac{S_3}{S_2} = \alpha^j, \\ X_2 &= \alpha^1, \\ Y_1 &= S_0, \\ Y_2 &= S_1 + \tfrac{S_3}{S_2} S_0. \end{aligned} \tag{48}$$

(c) If $J = 2$ then:

$$\begin{aligned} S_0 &= Y_1, \\ S_1 &= X_1Y_1, \\ S_2 &= X_1^2Y_1 + Y_2, \\ S_3 &= X_1^3Y_1, \\ S_4 &= X_1^4Y_1. \end{aligned} \tag{49}$$

Test if the following is true:

$$A_1 \neq A_2 \wedge A_2 \neq A_3 \wedge A_1 = A_4. \tag{50}$$

From (49) follows that:

$$\begin{aligned} X_1 &= \tfrac{S_1}{S_0} = \alpha^j, \\ X_2 &= \alpha^2, \\ Y_1 &= S_0, \\ Y_2 &= S_2 + \tfrac{S_1^2}{S_0}. \end{aligned} \tag{51}$$

(d) If $J = 3$ then:

$$\begin{aligned} S_0 &= Y_1, \\ S_1 &= X_1Y_1, \\ S_2 &= X_1^2Y_1, \\ S_3 &= X_1^3Y_1 + Y_2, \\ S_4 &= X_1^4Y_1. \end{aligned} \tag{52}$$

Test if the following is true:

$$A_1 = A_2 \wedge A_2 \neq A_3 \wedge A_3 \neq A_4. \tag{53}$$

From (52) follows that:

$$\begin{aligned} X_1 &= \tfrac{S_1}{S_0} = \alpha^j, \\ X_2 &= \alpha^3, \\ Y_1 &= S_0, \\ Y_2 &= S_3 + \tfrac{S_1^3}{S_0^2}. \end{aligned} \tag{54}$$

(e) If $J = 4$ then:

$$\begin{aligned} S_0 &= Y_1, \\ S_1 &= X_1Y_1, \\ S_2 &= X_1^2Y_1, \\ S_3 &= X_1^3Y_1, \\ S_4 &= X_1^4Y_1 + Y_2. \end{aligned} \tag{55}$$

Test if the following is true:

$$A_1 = A_2 \wedge A_2 \neq A_3 \wedge A_3 \neq A_4. \tag{56}$$

From (55) follows that:

$$\begin{aligned} X_1 &= \tfrac{S_1}{S_0} = \alpha^j, \\ X_2 &= \alpha^4, \\ Y_1 &= S_0, \\ Y_2 &= S_4 + \tfrac{S_1^4}{S_0^3}. \end{aligned} \tag{57}$$

Correction of the received word:

$$\begin{aligned} \hat{c}_j &= v_j + Y_1, \\ \hat{r}_J &= r_J + Y_2, \quad J = 0, 1, \ldots, 4. \end{aligned} \tag{58}$$

(f) In this step an assumption is made that both errors are in the information part. In this case the following is true:

$$A_1 \neq A_2 \wedge A_2 \neq A_3 \wedge A_3 \neq A_4 \tag{59}$$

and the following equations must be fulfilled:

$$S_0 = Y_1 + Y_2, \tag{60}$$

$$S_1 = X_1Y_1 + X_2Y_2, \tag{61}$$

$$S_2 = X_1^2Y_1 + X_2^2Y_2, \tag{62}$$

$$S_3 = X_1^3Y_1 + X_2^3Y_2, \tag{63}$$

$$S_4 = X_1^4Y_1 + X_2^4Y_2. \tag{64}$$

The unknown variables X_1, X_2, Y_1, Y_2 can be computed using a standard method described in Section 3. The correction of the received word can be done similarly as in case 4. But now, two positions in the information part have to be corrected corresponding to the two locators X_1 and X_2 by adding the calculated values Y_1 and Y_2 of the errors to the received symbols:

$$\begin{aligned} \hat{c}_i &= v_i + Y_1, \\ \hat{c}_j &= v_j + Y_2. \end{aligned} \tag{65}$$

6. If none of the cases: 1.–6. is confirmed, then it has to be assumed that more than two errors occurred and the decoding failure has to be declared.

In Figure 2 the proposed decoding algorithm is depicted in a compact way using a flow chart.

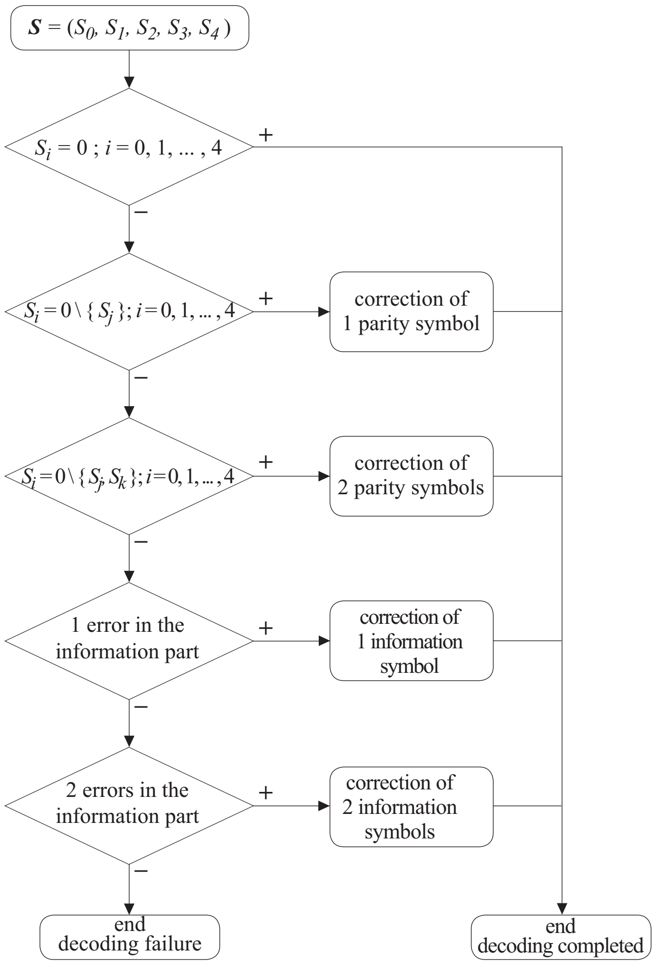

S = (S0, S1, S2, S3, S4)
Si = 0 ; i = 0, 1, ... , 4
+
–
Si = 0 \ { Sj }; i = 0, 1, ... , 4
correction of 1 parity symbol
Si = 0 \ { Sj, Sk }; i = 0, 1, ... , 4
correction of 2 parity symbols
1 error in the information part
correction of 1 information symbol
2 errors in the information part
correction of 2 information symbols
end decoding failure
end decoding completed

Figure 2. Flow chart of decoding algorithm

5 SOME REMARKS ON DECODING IMPLEMENTATION

In the previous section the basic principles of the decoding algorithm were presented for error correction in five times extended Reed Solomon codes. In this section we will explain some properties which were used in particular steps of the decoding algorithm and describe in more detail how to solve the different, earlier mentioned systems of equations. We will start with polynomial notation for (29) and (32):

$$c(x) = c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x^1 + c_0x^0 + (p_0 + p_1 + p_2 + p_3 + p_4)x^0, \quad (66)$$

$$v(x) = v_{k-1}x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x^1 + v_0x^0 + (r_0 + r_1 + r_2 + r_3 + r_4)x^0 \quad (67)$$

which will help to express the formulas needed to explain the proposed decoding algorithm. Similarly the error vector could be expressed using a polynomial:

$$e(x) = e_{k-1}e^{k-1} + e_{k-2}x^{k-2} + \ldots + e_1x^1 + e_0x^0 + (\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4)x^0. \quad (68)$$

Note: This notation will also be useful later in the section dealing with the analysis of the decoding algorithm complexity. This is because the complexity could be very often decreased by ideas inspired by hardware realizations of decoders. On the other hand the hardware for cyclic codes is conveniently described by polynomials.

By observing (66), (67) and the calculation of the syndromes together with knowing the role of locators during decoding ordinary RS codes, it is obvious that the description (66) and (67) is correct. However, it is not helpful since it does not allow us to distinguish the positions of v_0 and $r_0, r_1, \ldots, r_4$. It is because in this polynomial description all these symbols appear in sum as a coefficient of x^0. Therefore, we will introduce the following new sets of polynomials which are better adapted to the decoding of five times extended RS codes. This approach enables us to solve the mentioned problem with location distinction.

$$\begin{aligned}
c_0(x) &= c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x^1 + c_0x^0 + p_0x^0,\\
c_1(x) &= c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x^1 + c_0x^0 + p_1x^0,\\
c_2(x) &= c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x^1 + c_0x^0 + p_2x^0,\\
c_3(x) &= c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x^1 + c_0x^0 + p_3x^0,\\
c_4(x) &= c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x^1 + c_0x^0 + p_4x^0,
\end{aligned} \quad (69)$$

$$\begin{aligned}
v_0(x) &= v_{k-1}x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x^1 + v_0x^0 + r_0x^0,\\
v_1(x) &= v_{k-1}x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x^1 + v_0x^0 + r_1x^0,\\
v_2(x) &= v_{k-1}x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x^1 + v_0x^0 + r_2x^0,\\
v_3(x) &= v_{k-1}x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x^1 + v_0x^0 + r_3x^0,\\
v_4(x) &= v_{k-1}x^{k-1} + v_{k-2}x^{k-2} + \ldots + v_1x^1 + v_0x^0 + r_4x^0,
\end{aligned} \quad (70)$$

$$\begin{aligned}
e_0(x) &= e_{k-1}x^{k-1} + e_{k-2}x^{k-2} + \ldots + e_1x^1 + e_0x^0 + \epsilon_0x^0,\\
e_1(x) &= e_{k-1}x^{k-1} + e_{k-2}x^{k-2} + \ldots + e_1x^1 + e_0x^0 + \epsilon_1x^0,\\
e_2(x) &= e_{k-1}x^{k-1} + e_{k-2}x^{k-2} + \ldots + e_1x^1 + e_0x^0 + \epsilon_2x^0,\\
e_3(x) &= e_{k-1}x^{k-1} + e_{k-2}x^{k-2} + \ldots + e_1x^1 + e_0x^0 + \epsilon_3x^0,\\
e_4(x) &= e_{k-1}x^{k-1} + e_{k-2}x^{k-2} + \ldots + e_1x^1 + e_0x^0 + \epsilon_4x^0.
\end{aligned} \quad (71)$$

Using these polynomials we can express the syndromes as follows:

$$\begin{aligned} S_0 &= v_0(\alpha^0) = c_0(\alpha^0) + e_0(\alpha^0) = e_{k-1} + \ldots + e_1 + e_0 + \epsilon_0, \\ S_1 &= v_1(\alpha^1) = c_1(\alpha^1) + e_1(\alpha^1) = e_{k-1}\alpha^{k-1} + \ldots + e_1\alpha^1 + e_0\alpha^0 + \epsilon_1\alpha^0, \\ S_2 &= v_2(\alpha^2) = c_2(\alpha^2) + e_2(\alpha^2) = e_{k-1}\alpha^{2(k-1)} + \ldots + e_1\alpha^2 + e_0\alpha^0 + \epsilon_2\alpha^0, \\ S_3 &= v_3(\alpha^3) = c_3(\alpha^3) + e_3(\alpha^3) = e_{k-1}\alpha^{3(k-1)} + \ldots + e_1\alpha^3 + e_0\alpha^0 + \epsilon_3\alpha^0, \\ S_4 &= v_4(\alpha^4) = c_4(\alpha^4) + e_4(\alpha^4) = e_{k-1}\alpha^{4(k-1)} + \ldots + e_1\alpha^4 + e_0\alpha^0 + \epsilon_4\alpha^0. \end{aligned} \tag{72}$$

Now we can explain some properties on which the particular steps of the proposed decoding algorithm are based depending on error location.

Case 1 is trivial and known for syndrome decoding of linear block codes.

In Case 2, the following property is used: assuming that the error occurs in one of the parity symbols, or in other words, if one of the symbols from a set $\{\epsilon_0, \epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4\}$ in the error polynomial is nonzero, it means that only one corresponding syndrome is nonzero. It follows from (72).

Similarly in Case 3, the following property is used: assuming that only two syndromes e.g. S_I and S_J are nonzero and all others are zeros, then it is supposed that exactly two errors occurred in the parity positions, which follows from (72).

In Case 4, it is supposed that:

$$\epsilon_0 = \epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4 = 0 \tag{73}$$

and only one of the symbols from the set $\{e_{k-1}, e_{k-2}, \ldots, e_1, e_0\}$ is nonzero. From these assumptions it follows that:

$$c_0(x) = c_1(x) = c_2(x) = c_3(x) = c_4(x) = c(x). \tag{74}$$

And therefore (40) follows from (17).

In Case 5, it is supposed that the first error ocured in the information part of the codeword and the second one in the parity part. Therefore, for the first error we can use error locator X_1 as usually defined inside the information part which is mapped on $c(x)$ because in this case: $\epsilon_0 = \epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon_4 = 0$ and $c_0(x) = c_1(x) = c_2(x) = c_3(x) = c_4(x) = c(x)$. However, for the second error the assumptions are different. Namely that one of the symbols from a set $\{e_{k-1}, e_{k-2}, \ldots, e_1, e_0\}$ in the error polynomial is nonzero. On the other hand, the location and value of the second error is not given by a standard locator in this case. These are determined indirectly using tests: (44),(47),(50),(53),(56) and the system of equations: (45),(48),(51),(54),(57). The reason is that the single nonzero element from the set $\{e_{k-1}, e_{k-2}, \ldots, e_1, e_0\}$ will cause the second summand on the right side to be equal to zero in only one of the above mentioned systems of equations.

In Case 6, the reasoning is similar, because it is assumed that (73) is valid and two of the symbols from set $\{e_{k-1}, e_{k-2}, \ldots, e_1, e_0\}$ are nonzero. Therefore (73) also holds. Consequently (60), (61), (62), (63) and (64) follow from (17) again.

However, in this case it is worth giving more details of the method how to solve the system of Equations (60), (61), (62), (63) and (64). In order to decode the received word it is necessary to find two unknown locators X_1, X_2 and two unknown error values Y_1, Y_2, together 4 unknowns. Therefore at least 4 equations are needed to be calculated. However, the Equations (60), (61), (62), (63) and (64) are not linear. Therefore a direct analytical solution is not easy or viable at all. The following notification allows arguing that the standard approach as in decoding ordinary RS codes is possible. Because in Case 5, it is supposed that two errors occurred, the locator polynomial has degree two:

$$\lambda(x) = \lambda_2 x^2 + \lambda_1 x^1 + x^0. \tag{75}$$

Therefore:

$$\lambda_2 X_i^2 + \lambda_1 X_i^1 + 1 = 0; \quad i = 1,2. \tag{76}$$

After multiplying (75) with $Y_i X_i^z$; $i = 1,2$; $z \in Z$ we get:

$$\lambda_2 Y_i X_i^{z+2} + \lambda_1 Y_i X_i^{z+1} + Y_i X_i^z = 0; \quad i = 1,2 \quad z \in Z. \tag{77}$$

Now we can write (76) for a fixed value of $z \in Z$. For $z = 0$ we get:

$$\begin{aligned} \lambda_2 Y_1 X_1^2 + \lambda_1 Y_1 X_1^1 + Y_1 X_1^0 &= 0, \\ \lambda_2 Y_2 X_2^2 + \lambda_1 Y_2 X_2^1 + Y_2 X_2^0 &= 0. \end{aligned} \tag{78}$$

Summation of (78) gives us:

$$\lambda_2 S_2 + \lambda_1 S_1 + S_0 = 0. \tag{79}$$

For $z = 1$, we get similarly:

$$\begin{aligned} \lambda_2 Y_1 X_1^3 + \lambda_1 Y_1 X_1^2 + Y_1 X_1^1 &= 0, \\ \lambda_2 Y_2 X_2^3 + \lambda_1 Y_2 X_2^2 + Y_2 X_2^1 &= 0. \end{aligned} \tag{80}$$

Summation of (80) gives us:

$$\lambda_2 S_3 + \lambda_1 S_2 + S_1 = 0. \tag{81}$$

Now (79) and (81) form a system of linear equations with two unknowns: λ_1 and λ_2. Such a system is easily solvable by standard approaches. For example:

$$\lambda_2 = \frac{S_1 + S_0 S_2}{S_2^2 + S_1 S_3}, \tag{82}$$

$$\lambda_1 = \frac{S_0 + \lambda_2 S_2}{S_1}. \tag{83}$$

As a result we have a concrete error locator polynomial and therefore the Chien algorithm could be used in order to get locators X_1 and X_2. Or (75) can be solved directly [15, 16, 17, 18].

For example in [17] after substituting $x = \frac{\lambda_1}{\lambda_2}u$ into (75), the error locator polynomial will be:

$$\lambda(x) = \frac{\lambda_1^2}{\lambda_2}\left(u^2 + u + \frac{\lambda_2}{\lambda_1^2}\right). \tag{84}$$

In order to find its roots it is necessary to solve:

$$u^2 + u + \frac{\lambda_2}{\lambda_1^2} = 0. \tag{85}$$

Because $GF(2^m)$ is, by squaring, transformed linearly over $GF(2)$, Equation (85) could be reformulated as follows:

$$\mathbf{u}\left(\mathbf{\Theta} + \mathbf{I}\right) = \frac{\mathbf{\Lambda}_2}{\mathbf{\Lambda}_1^2} \tag{86}$$

where $\mathbf{u}$, $\frac{\mathbf{\Lambda}_2}{\mathbf{\Lambda}_1^2}, \mathbf{I}$ and $\mathbf{\Theta}$ are binary vectors, identity matrix and $m \times m$ square operator matrix, respectively. After adding the two matrices in (86) the following equality is obtained:

$$\mathbf{u} \times \mathbf{T} = \frac{\mathbf{\Lambda}_2}{\mathbf{\Lambda}_1^2} \tag{87}$$

where $\mathbf{T} = \mathbf{\Theta} + \mathbf{I}$. To make the computation fast the pseudo inverse of $\mathbf{T}$ could be implemented via lookup table in advance for the specific finite field $GF(2^m)$. The calculation of error locators then consists of the following steps:

- calculating $\frac{\mathbf{\Lambda}_2}{\mathbf{\Lambda}_1^2}$;
- reading out the two roots: U_1, U_2 from the lookup table;
- transforming U_1, U_2 into locators X_1, X_2 by using:

$$X_1 = \frac{\mathbf{\Lambda_1}}{\mathbf{\Lambda_2}}U_1; \tag{88}$$

- thanks to Vieta's formulas we can use addition instead of multiplication for the second locator transformation:

$$X_2 = U_1 + \frac{\mathbf{\Lambda_1}}{\mathbf{\Lambda_2}}. \tag{89}$$

After calculating X_1, X_2 they could be inserted into (61). By using the resulting equation and (60) the error values Y_1 and Y_2 could be calculated from this linear system of equations:

$$Y_2 = \frac{S_1 + S_0X_1}{X_1 + X_2}, \tag{90}$$

$$Y_1 = S_0 + Y_2. \tag{91}$$

Case 7 is a consequence of the fact that all correctable configurations of errors determined by the code distance of the codes are covered by previous cases: 1–6. However, if non correctable configuration of errors occurs, for example containing more than two errors, then the decoding can end in failure or incorrect decoding. The failure can be detected in Case 7. After this detection some remedy mechanism can be started, for example the request for re-sending the codeword again could be generated. Therefore this situation is better than the incorrect decoding caused by more than two errors. This can happen if under the influence of more than two errors the received combination is closer to some other codeword than to one which was sent. Namely its Hamming distance is smaller than or equal to two.

6 COMPLEXITY ESTIMATION OF THE ALGORITHM

In this section the complexity estimation of the proposed decoding algorithm is presented. The number of operations needed for calculating syndromes will not be counted, because these operations are necessary in any syndrome based decoding and they are usually done by hardware. It is obvious that the number of operations necessary for correct decoding is dependent on the number of errors which occur in the transmission channel and on their positions in the received codeword. Therefore an average number of operations per codeword was chosen for this estimation with the assumption that the error occurrence mechanism is modeled as a q-ary symmetric channel depicted in Figure 3.

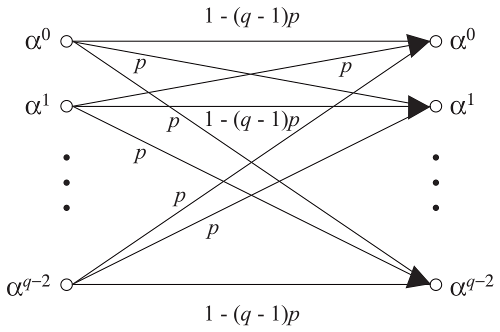


Figure 3. q-ary symmetric channel

If we denote by $P_e = (q-1)p$ the probability of error in this channel model, then:

$$P_0 = (1-P_e)^{q+4}, \tag{92}$$

$$P_1 = \binom{q+4}{1} P_e(1-P_e)^{q+3}, \tag{93}$$

$$P_2 = \binom{q+4}{2} P_e^2(1-P_e)^{q+2} \tag{94}$$

are the probabilities that 0, 1 and 2 errors occur in a received codeword, respectively. However the complexity also depends on the constellation of the error or errors. Therefore the following probabilities for the decoding complexity estimation will be defined:

- probability of one error in parity symbol:

$$P_{1P} = P_1 \frac{5}{q+4} = 5P_e(1-P_e)^{q+3}, \tag{95}$$

- probability of one error in information symbol:

$$P_{1I} = P_1 \frac{q-1}{q+4} = (q-1)P_e(1-P_e)^{q+3}, \tag{96}$$

- probability of two errors in parity symbol:

$$P_{2P} = P_2 \frac{\binom{5}{2}}{\binom{q+4}{2}} = 10P_e^2(1-P_e)^{q+2}, \tag{97}$$

- probability of two errors in information symbol:

$$P_{2I} = P_2 \frac{\binom{q-1}{2}}{\binom{q+4}{2}} = \frac{(q-1)(q-2)}{2} P_e^2(1-P_e)^{q+2}, \tag{98}$$

- probability of one error in parity symbol and one error in information symbol:

$$P_{1P1I} = P_2 \frac{5(q-1)}{(q+4)^2} = \frac{5(q-1)(q-2)}{2(q+4)} P_e^2(1-P_e)^{q+2}, \tag{99}$$

- probability of more than two errors in codeword:

$$P_{>2} = \sum_{j=3}^{q+4} \left(\begin{array}{c} q+4 \\ j \end{array} \right) P_e^j (1 - P_e)^{q+4-j}. \tag{100}$$

After the algorithm gets as an input 5 syndromes, five comparisons of the calculated syndromes are done and depending on the results one of the following subroutines are performed. The complexity for each case can be given by the number of operations. Next, the average number of operations could be calculated using the cumulative probability theorem. We will introduce the vector $\mathbf{\Upsilon} = (\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4, \upsilon_5)$ in which the particular non-negative integer coordinates: $\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4, \upsilon_5$ denote the number of additions, multiplications, divisions, comparisons and look up table accesses, respectively, which are needed in order to perform the particular subroutines. The detailed estimation of the necessary operations (except calculation of syndromes) follows:

- one error in parity symbol: 1 addition:

$$\mathbf{\Upsilon}_{1P} = (1, 0, 0, 0, 0), \tag{101}$$

- two errors in parity symbol: 2 additions:

$$\mathbf{\Upsilon}_{2P} = (2, 0, 0, 0, 0), \tag{102}$$

- one error in information symbol: 1 addition, 4 divisions and 3 comparisons:

$$\mathbf{\Upsilon}_{1I} = (1, 0, 4, 3, 0), \tag{103}$$

- one error in parity symbol and one error in information symbol. Now the calculations and mutual comparison of auxiliary variables: A_1, A_2, A_3, A_4 was done in the previous case. In the worst case 3. additional additions, one additional multiplication and 3 additional divisions have to be performed. Therefore:

$$\mathbf{\Upsilon}_{1P1I} = (4, 1, 7, 3, 0), \tag{104}$$

- two errors in information symbols: For the calculations and mutual comparison of auxiliary variables: A_1, A_2, A_3, A_4 holds the same as in the previous case. Then the correction of two errors will need for solutions of (88), (89),(90) and (91): 4 additions, 8 divisions, 5 multiplications and 2 readouts from the look up table:

$$\mathbf{\Upsilon}_{2I} = (4, 5, 12, 3, 2), \tag{105}$$

- more than two errors in information symbol: In this case the complexity of procedures quantified by $\mathbf{\Upsilon}_{1I}, \mathbf{\Upsilon}_{2I}, \mathbf{\Upsilon}_{1P1I}$ can be added as the worst case:

$$\mathbf{\Upsilon}_{>2I} = (16, 6, 23, 9, 2). \tag{106}$$

We will characterize the overall complexity using a vector: $\mathbf{\Gamma} = (\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5)$ in which the particular coordinates: $\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5$ denote average numbers of: additions, multiplications, divisions, comparisons and look up table accesses, respectively, which are needed in order to perform the particular subroutines. The value of $\mathbf{\Gamma}$ can be evaluated from (101), (102), (103), (104), (105) and (106) using the cumulative probability formula:

$$\begin{aligned}\mathbf{\Gamma} = {} & \mathbf{\Upsilon}_{1P} \times 5P_e(1-P_e)^{q+3} \\ & + \mathbf{\Upsilon}_{2P} \times 10P_e^2(1-P_e)^{q+2} \\ & + \mathbf{\Upsilon}_{1I} \times (q-1)P_e(1-P_e)^{q+3} \\ & + \mathbf{\Upsilon}_{2I} \times \frac{(q-1)(q-2)}{2}P_e^2(1-P_e)^{q+2} \\ & + \mathbf{\Upsilon}_{1P1I} \times \frac{5(q-1)(q-2)}{2(q+4)}P_e^2(1-P_e)^{q+2} \\ & + \mathbf{\Upsilon}_{>2I} \times \sum_{j=3}^{q+4} \binom{q+4}{j} P_e^j(1-P_e)^{q+4-j}. \end{aligned} \tag{107}$$

After evaluation using (107) it is possible to get information about the average number of particular operations: additions, multiplications, divisions, comparisons and accessing the look up table, necessary for decoding the received codewords. By observing (107) it is also obvious that these average numbers of operations depend on the error probability P_e of the q-ary symmetric channel and on q – the number of elements in the finite field.

The total average number of operations per codeword is given by the sum of all coordinates of (107):

$$\bar{\mathbf{\Gamma}} = \sum_{i=1}^{5} \gamma_i. \tag{108}$$

A tabular representation of (108) for selected values of P_e for $GF(64)$ and $GF(256)$ is shown in Table 1.

	GF(64)	GF(256)
P_e	Γ	Γ
10^{-2}	7.058	34.89
10^{-3}	5.280×10^{-1}	2.370
10^{-4}	5.109×10^{-2}	2.077×10^{-1}
10^{-5}	5.092×10^{-3}	2.048×10^{-2}

Table 1. The average number of operations

It has to be stressed out that the complexity estimations in Table 1 do not include the complexities of syndrome calculations. These complexities are constant for particular finite fields. In $GF(q)$ if we use the Horner scheme for syndrome calculations we will need $5 \times (q-2)$ additions and $5 \times (q-2)$ multiplications. E.g. in $GF(64)$ and in $GF(256)$ the overall number of operations needed for calculation of syndromes will be 640 and 2 560, respectively. As was already mentioned, the syndromes always have to be calculated in any syndrome decoding algorithm and usually this is done via hardware. The average number of operations needed in the proposed algorithm for example where $P_e = 10^{-3}$ is less than 1 and less than 3 in $GF(64)$ and $GF(256)$, respectively.

7 CONCLUSIONS

In this paper a new decoding algorithm was proposed for the five times extended Reed Solomon codes proposed in [1], which allows correcting up to two errors in a codeword. In contrast to known error correcting syndrome decoding algorithms, the method proposed in this paper has to deal with the problem that the control matrix is not a pure Vandermonde matrix, but it is a juxtaposition of a Vandermonde and an identity matrix. Therefore, the proposed algorithm had to (to some extent) use strategy somewhat like a set of sieves in order to separate different possible error patterns. This strategy is explained in detail using specially created polynomial and matrix analytical descriptions. This algorithm was also verified using Mathematica software for all distinct cases and all typical combinations of error patterns.

Acknowldegement

This work was supported by the Slovak Research and Development Agency under the Contract No. APVV-19-0436 and it was also supported by the Scientific Grant Agency of the Ministry of Education of the Slovak Republic and the Slovak Academy of Sciences (grant VEGA No. 1/0477/18).

REFERENCES

[1] Rakús, M.—Farkaš, P.—Páleník, T.—Daniš, A.: Five Times Extended Reed-Solomon Codes Applicable in Memory Storage Systems. IEEE Letters of the Computer Society, Vol. 2, 2019, No. 2, pp. 9–11, doi: 10.1109/LOCS.2019.2911517.

[2] Rakús, M.—Farkaš, P.—Páleník, T.: Erasure Decoding of Five Times Extended Reed-Solomon Codes. Journal of Electrical Engineering, Vol. 70, 2019, No. 3, pp. 256–258, doi: 10.2478/jee-2019-0035.

[3] Reed, I. S.—Solomon, G.: Polynomial Codes over Certain Finite Fields. M.I.T. Lincoln Laboratory Group Report 47.23, 31 December 1958.

[4] Wicker, S. B.—Bhargava, V. K.: Reed-Solomon Codes and Their Applications. Wiley-IEEE Press, 1999, 336 pp., doi: 10.1109/9780470546345. ISBN 978-0-780-35391-6.

[5] Li, W.—Wang, Z.—Jafarkhani, H.: Repairing Reed-Solomon Codes over $GF(2^l)$. IEEE Communications Letters, Vol. 24, 2020, No. 1, pp. 34–37, doi: 10.1109/LCOMM.2019.2950922.

[6] Zhang, G.—Liu, H.: Constructions of Optimal Codes with Hierarchical Locality. IEEE Transactions on Information Theory (Early access), doi: 10.1109/TIT.2020.2977636.

[7] Chen, Z.—Barg, A.: Explicit Constructions of MSR Codes for Clustered Distributed Storage: The Rack-Aware Storage Model. IEEE Transactions on Information Theory, Vol. 66, 2020, No. 2, pp. 886–899, doi: 10.1109/TIT.2019.2941744.

[8] Zhang, Y.—Zhang, Z.: An Improved Cooperative Repair Scheme for Reed-Solomon Codes. Proceedings of 2019 19th International Symposium on Communications and Information Technologies (ISCIT), Ho Chi Minh City, Vietnam, 2019, pp. 525–530, doi: 10.1109/ISCIT.2019.8905159.

[9] Li, W.—Wang, Z.—Jafarkhani, H.: On the Sub-Packetization Size and the Repair Bandwidth of Reed-Solomon Codes. IEEE Transactions on Information Theory, Vol. 65, 2019, No. 9, pp. 5484–5502, doi: 10.1109/TIT.2019.2917425.

[10] Jagmohan, A.—Lastras-Montano, L. A. (Inventors): Mis-Correction and No-Correction Rates for Error Control. US patent: US 8 806 295 B2, Aug. 12, 2014 (Assignee: IBM Corp.).

[11] Rakús, M.—Farkaš, P.: Double Error Correcting Codes with Improved Code Rates. Journal of Electrical Engineering, Vol. 55, 2004, No. 3-4, pp. 89–94.

[12] Berlekamp, E. R.: Nonbinary BCH Decoding. International Symposium on Information Theory, San Remo, Italy, 1967, doi: 10.1109/tit.1968.1054109.

[13] Massey, J.: Shift-Register Synthesis and BCH Decoding. IEEE Transactions on Information Theory, Vol. 15, 1969, No. 1, pp. 122–127, doi: 10.1109/TIT.1969.1054260.

[14] Chien, R.: Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes. IEEE Transactions on Information Theory, Vol. 10, 1964, No. 4, pp. 357–363, doi: 10.1109/TIT.1964.1053699.

[15] Berlekamp, E. R.—Rumsey, H.—Solomon, G.: On the Solution of Algebraic Equations over Finite Fields. Information and Control, 1967, pp. 553–564, doi: 10.1016/s0019-9958(67)91016-9.

[16] Walker, C. W.: New Formulas for Solving Quadratic Equations over Certain Finite Fields. IEEE Transactions on Information Theory, Vol. 45, 1999, No. 1, pp. 283–284, doi: 10.1109/18.746816.

[17] Pommerening, K.: Quadratic Equations in Finite Fields of Characteristic 2. May 2000, English Version, February 2012, available online: `https://www.staff.uni-mainz.de/pommeren/MathMisc/QuGlChar2.pdf`.

[18] Goresky, M.—Klapper, A.: Algebraic Shift Register Sequences. Cambridge University Press, 2005.

Peter Farkaš is with the Institute of Multimedia Information and Communication Technologies, Slovak University of Technology in Bratislava (STU) and also with the Institute of Applied Informatics, Faculty of Informatics, Pan European University in Bratislava as Professor. From 2002 until 2007 he was Visiting Professor at Kingston University, UK and Senior Researcher at SIEMENS PSE. In 2003 SIEMENS named him VIP for his innovations and patents. In 2004 he was awarded with the Werner von Siemens Excellence Award for research results on two-dimensional complete complementary codes. From 2008 to 2009 he worked also as Consultant in the area of software defined radio for SANDBRIDGE Tech. (USA). He was the Leader of a Team from STU in projects funded by the European Community under the 5FP and 6FP "Information Society Technologies Programs": NEXWAY IST-2001-37944 (Network of Excellence in Wireless Applications and Technology) and CRUISE (Creating Ubiquitous Intelligent Sensing Environments) FP6 IST-2005-4-027738 (2006–2007). His research interests include coding, communications theory and sequences for CDMA. He has published 1 book, about 45 papers in reviewed scientific journals and about 100 papers in international conferences. He is the author or co-author of 7 patents. He is and was serving in TPC of about 60 international conferences and presented 12 invited lectures. As an IEEE volunteer, he was serving in the IEEE Czechoslovakia Section Executive Committee in different positions from 1992 to 2014 and from 2005 to 2006 he served as Chair of the Conference Coordinator Subcommittee in IEEE Region 8. He organized the IEEE R8 Conference EUROCON 2001 and was Chairman of SympoTIC '03, SympoTIC '04, SympoTIC '06 and co-organizer of the Winter School on Coding and Information Theory 2005. From 2016 he has been serving as a vice-chair of the Computer chapter in the IEEE Czechoslovakia Section.

Martin Rakús studied radio electronics and graduated from the Slovak University of Technology in 2001. In 2004 he received his Ph.D. from the Slovak University of Technology and in 2020 he became Associate Professor at the same institute. Since 1995 he has been with the Institute of Multimedia Information and Communication Technology, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, Slovakia. His primary research interests are error control coding and digital communication systems. He is a member of the IEEE.

Computing and Informatics, Vol. 39, 2020, 1336–1359, doi: 10.31577/cai_2020_6_1336

EVENT DETECTION IN TWITTER USING MULTI TIMING CHAINED WINDOWS

Mohammad Mahdi Mojiri, Reza Ravanmehr*

Department of Computer Engineering
Central Tehran Branch, Islamic Azad University
Tehran, Iran
e-mail: `mojiry@gmail.com, r.ravanmehr@iauctb.ac.ir`

Abstract. Twitter is a popular microblogging and social networking service. Twitter posts are continuously generated and well suited for knowledge discovery using different data mining techniques. We present a novel near real-time approach for processing tweets and detecting events. The proposed method, Multi Timing Chained Windows (MTCW), is independent of the language of the tweets. The MTCW defines several Timing Windows and links them to each other like a chain. Indeed, in this chain, the input of the larger window will be the output of the smaller previous one. Using MTCW, the events can be detected over a few minutes. To evaluate this idea, the required dataset has been collected using the Twitter API. The results of evaluations show the accuracy and the effectiveness of our approach compared with other state-of-the-art methods in the event detection in Twitter.

Keywords: Event detection, microblogging, twitter, timing windows, MTCW

1 INTRODUCTION

Nowadays, social networks are much used in everyday lives, and the users of these networks usually share a lot of information with others about their events and incidents, their opinions on various issues, and so on. Meanwhile, Twitter has an essential role due to the number of users and its specific structure. This microblog had 336 million monthly active users in the first quarter of 2018 who sent 500 million

* Corresponding author

tweets per day [1, 2]. The high number of posts on this social network and the extent of its users have attracted the interest of researchers. In recent years, this huge amount of data has provided a hotbed for data mining research to discover facts, trends, events, and even predict specific incidents [3, 4, 5, 6, 7].

The data streaming process, including social networks, produces millions of documents (image, text, audio, etc.) per hour. Since these documents arrive at high speed, there is not much time to process them, and they cannot be stored in memory, either. Therefore, it is very difficult to compare the current and previous documents and find any similarities or changes between them.

On social networks such as Twitter, data is produced in an unlimited and endless stream. Therefore, the process for storing and analyzing this inexhaustible stream of information should not be dependent on the hardware specification of the system, such as memory capacity. In other words, if the event detection algorithm is applied to tweets for two months or two days, its accuracy should not be reduced, as well as the limitations of hardware such as memory to prevent it [5]. So, we need to propose an approach to detect events employing the analysis of tweets while they are received and require no further storage and processing.

The research conducted by Petrovic et al. [8] and Osborne and Dredze [9] confirm the effectiveness of employing Twitter for an event detection system. However, there are at least two severe challenges of doing so. Firstly, not all data from Twitter are correct and may include spam and personal information without processing capability. In studies such as [10], news agencies are the source of valid information, but the posts of users in social networks are not limited to factual data and events. Indeed, many posts are spam and trivial statements. The majority of tweets are not true stories, some concern personal life, some conversation and friendly chats, and some spam. The second challenge is the message length limit on Twitter. Twitter does not allow users to post a text longer than 280 characters, which causes changes in the syntax of the language. For example, short sentences are written instead of full sentences, which makes it difficult to process the tweets.

Many event detection approaches take advantage of the keywords contained in the users' post.The Burst Keyword is a group of keywords that represents a trend or the occurrence of a new event. Certainly, the period of the occurrence of these Burst Keywords is also very important; for example, if a Burst Keyword is observed over a few minutes, it represents an event, but if it occurs over several days, it indicates a new trend. The main idea of this research is to use multiple Timing Windows simultaneously and link them together. In this method, the smaller Timing Windows removes the words in the next (larger) Timing Windows and sends the remaining words to that Timing Windows. By applying this method, the history of repetitive words will not affect the detection of a sudden increase in windows at present. Moreover, processing of incoming tweets as well as other calculations are performed only for the first window, and there will be no waste of time for preliminary calculations in other Windows.

In the proposed method, Multi Timing Chained Windows (MTCW), sudden events can be detected over a few minutes. The suggested method is also independent of the user language. It benefits from the constant time and space complexity for any number of tweets, as explained in the next sections.

In the following, the research fundamentals of this study are described. The related works in the field of event detection are discussed in Section 3. The proposed approach is described in Section 4. Next, experimental results and evaluation of MTCW are discussed employing the collected Twitter dataset. Then, our proposed approach is compared with other state-of-the-art methods in event detection on Twitter. Finally, we conclude the paper and provide future works.

2 FUNDAMENTALS OF RESEARCH

In this section, the foundations of this research, including concepts, objectives, and quality of service parameters in event detection, are described.

2.1 Concepts in Event Detection

In recent years, the detection of an event has been a very trending topic. The underlying assumption is that an increasing use of some related keywords shows that an event is happening. Event is an occurrence of interest among users of the social media to discuss a real-world event-associated topic, usually after the incident or even sometimes prediction of it [11]. In the domain of Topic Detection and Tracking (TDT) [12], an event is defined as "Something that happens at a specific time and place along with all necessary conditions and unavoidable consequences".

In fact, there is no distinct segregation between trending topic detection and event detection in many papers. In some papers discussing the subject of trending topic detection, the term event detection is frequently mentioned, including [5, 13, 14, 15, 16, 17]. It is also pretty much the same in Twitter. In Twitter, there is a proprietary algorithm to detect and display the trending topics, consisting of terms and phrases that express the "trending" behavior. While Twitter's trending topics sometimes reflect current events (e.g., "iPhone X announcement"), they often include keywords for popular conversation topics (e.g., cryptocurrency).

As another example, suppose a celebrity who asks the fans in a tweet to comment on purchasing an apartment or a villa. A large number of tweets are generated discussing the disadvantages and benefits of each option. In this case, the "house" or "apartment" is raised as a trending topic, but has any event occurred in fact?

Indeed, the current systems usually are not intelligent enough to be able to recognize the difference between an event or a widespread discussion about a topic; therefore, most papers do not distinguish between an event and a trending topic. However, in articles such as [4], it has been attempted to use temporal and local tags to differentiate between urgent events with trends. It should be noted that Becker, Naaman, and Gravano [18] were the first researchers who considered this distinction between a trending topic and event detection.

2.2 Objectives of Event Detection

Considering the concepts and requirements of an event detection system, four significant objectives can be specified for an ideal user event detection system, including Generality, Scalability, Real-time processing, and No supervision. Apart from these four objectives, there are other criteria for perfect event detection. For example, no use of additional data and information or use of a dictionary could be good targets.

Generality: Many systems of event detection can only detect specific topics. In fact, just a series of topics already given to the system are detected. For example, an approach has been presented in [4] which detects whether the event is urgent or not and then classifies the event from among the preset categories (flood, earthquake, fire, etc.). In fact, this approach cannot detect such things as the death of a famous person.

Scalability or independence from space and time: The event detection system should not be dependent upon memory. This system should not slow down over time and should process each data upon arrival. This target states the consistency of the space required over time. Obviously, the system grows over time, which should not increase the required memory. Also, in solving a problem like event detection in Twitter, the growth rate is not fixed. In an approach like EDCoW [5] which works with signal processing, elimination of several tweets has been considered to reduce the space, while it cannot be a guarantee for scalability.

Real-time processing: This objective means the detection of the event as soon as it is observed. The real-time concept has been observed in a small number of approaches. Almost all of them conducted their assessment by collecting a dataset, but there is no discussion about the processing time in a real environment. In TwitterMonitor [19],the system works online, but even in this case, only a small number of tweets are tested due to lack of access to all tweets.

No supervision: The final event detection system must be able to operate without user monitoring, verification, and feedback. There should be no need for a training phase in the system since training is reserved for some tweets in the dataset whose behavior will change over time.

No use of a dictionary or additional data: Approaches such as [20] employed a dictionary as well to improve the accuracy of their system. The use of such things can result in a loss of generality of the approach because it is not possible to find and use a comprehensive dictionary for all the languages. Even if one finds and uses all the dictionaries, the system performance is reduced due to their high volume. Besides, the common languages in the world are generally dynamic and produce new words, so the use of a dictionary can be considered a weakness of the method.

2.3 Quality of Services in Event Detection

An effective event detection system is the one with a high recall to be able to retrieve each related event, as well as high precision to reject all the unrelated events [21].

In an event detection system, precision represents the usefulness of the output list (diagnostic events). The closer the events in the system output to real answers, the better this criterion. The precision is introduced as follows:

$$\text{Precision} = \frac{CDE}{TDE}. \tag{1}$$

In this equation, CDE (Correctly Detected Events) represents the number of correctly detected events, and TDE (Total Detected Events) is the total number of detected events. The recall represents the completeness of the output list. The recall is equal to:

$$\text{Recall} = \frac{CDE}{TE}. \tag{2}$$

In this equation, TE (Total Events) is equal to the total number of events that occurred during the period in question. Recall shows how well the system works in finding the events.

3 RELATED WORKS

Atefeh and Khreich generally classified event detection approaches in Twitter into document-pivot and feature-pivot techniques depending on whether they rely on a document or temporal features [22]. However, another classification has been recently proposed by Hasan et al. on Twitter-centric event detection systems [11].

Hasan et al. classified the event detection approach based on their common features into term-interestingness, topic-modeling, and incremental-clustering. The term-interestingness methods rely on tracking the terms from the Twitter data stream likely to be related to an event. The topic-modeling approaches associate each tweet with a probability distribution over the different latent topics to discover the hidden semantic structures from a stream of tweets to detect the related events. The incremental-clustering methods employ an incremental clustering strategy in order to avoid having a fixed number of clusters due to the high-volume, real-time Twitter data where a wide variety of topics are discussed.

Considering the above categories and taking into account the proposed approach in this article, we classify the research associated with Twitter event detection into the following two categories: Burst/Hot Keywords Frequency and Statistical Analysis.

A majority of event detection techniques use the frequency of keywords in users' posts to detect the related events. Some other approaches like Benhardus employ TF-IDF (Term Frequency–Inverse Document Frequency) factor, again related to the frequency. The extraction of Burst Keywords using Timing Windows can be used for event detection in many approaches, such as [19, 23, 24, 25]. The timing Window is a time slice in which the words are collected, and their sudden increase is studied. Also, looking for special keywords ("Hot Keywords") is another method to detect events. If the tweet or group of tweets include these hot keywords, then the associated event will be discovered (Section 3.1).

Some other approaches employ various statistical techniques such as LDA or Bayesian (Section 3.2). In these approaches, each tweet is associated with a probability distribution over various latent topics to discover the hidden semantic structures from a collection of tweets, such as [33, 34, 35].

3.1 Burst/Hot Keywords Frequency

As defined in articles, an event is expressed as a group of keywords, and the event detetcion system goal is to find this group that appears simultaneously in a stream of data.

A method for event detection using text mining techniques has been presented by Benhardus in [20]. This approach takes advantage of the frequency with TF-IDF factor as well as Entropy test. In this approach, the tweets are grouped in packages. Depending on when the tweets are sent, groups are normalized., which means that each document is linked to a fixed time interval.

Massive Online Analysis (MOA) TweetReader detects a trend in three steps [26]. In the first step, upon the arrival of each tweet from Twitter API, pre-processing is done, and then the feature vector is formed using TF-IDF parameter. In the second step, the tweets are tagged using a trained component, and finally, if a change is observed, the trend is discovered.

The SABESS (Social Awareness Based Emergency Situation Solver) approach employs the frequency method with some modifications [4]. Upon the arrival of each tweet, SABESS determines whether it is an urgent event or not, what category it is (flood, earthquake, fire, etc.), and where is the location of the event.

Recently, Frequent Pattern Mining (FPM) has been employed for event detection in Twitter. The FPM helps to find patterns of words that frequently occur in the Twitter data stream. The method called SFPM (Soft FPM) is also applied for this purpose by mitigating the requirement that all items must be frequent in the pattern [27]. Gaglio et al. extended the SFPM method to deal with dynamic environments of Twitter by splitting the streams into dynamic windows whose size depends both on the volume of tweets and time [28]. Huang et al. proposed an event detection based on the High Utility Pattern Clustering (HUPC) framework by clustering all patterns generated by the FP-Growth algorithm [29].

In the approach presented by Choi and Park, emerging topics on Twitter have been detected based on High Utility Pattern Mining (HUPM). The goal of HUPM is to find itemsets that have high frequency and high utility at the same time [30].

In [31], six different methods for topic detection have been evaluated (i.e., LDA, Doc-p, Gfeat-p, FPM, SFPM, and Bngram) on three datasets. Finally, a comparison of these methods concluded that the combination of DF-IDF and nGram in Bngram method showed the best results.

TwitterMonitor attempts to find the trend of users using the frequency method [19]. It first detects the Burst Keywords (keywords that suddenly appear in an unusually large number of tweets), and these words are then placed in their related groups. In other words, a trend is detected as a set of Burst Keywords that have frequently appeared together in tweets. Once a trend is detected, TwitterMonitor extracts additional information from the tweets related to the detected trend.

Guzman et al. have introduced an approach in [23] to detect sudden keywords. In this approach, suddenly rising words are detected using a five-stage algorithm. Each stage is written with a standalone module. Three modules are used for preprocessing, and the remaining two modules detect sudden words.

In EDCoW method, an event is indicated by keywords that are suddenly increased [5]. This approach attempts to detect new and important events using a signal processing technique. In EDCoW approach, a signal is generated only when a word shows a sudden behavior. The signal is then quickly processed without the need for considerable memory by Wavelet analysis. In fact, after receiving a signal, trivial words are discarded, and only the signal and its affiliates are considered. Then, the cross-correlation between signals is calculated, followed by event detection through signal clustering via graph partitioning.

TopicSketch is a framework for real-time detection of bursty topics on Twitter [32]. This approach utilizes two main techniques; a sketch-based topic model and a hashing-based dimension reduction technique. TopicSketch is not suitable for topic detection in a stream of documents with multiple topics.

Burnap et al. attempted to categorize public posts on Twitter according to their tension [3]. In this paper, tension is defined as follows: "any event that seriously disrupts a normal relationship between individuals or groups, which also spreads to groups or individuals not involved in the relationship". In their paper, first, the incoming tweet is tagged, and then the Burst keyword (if present) is detected in tension level. Their method determines the level of tension using simple rules. For example, if the tweet includes one or more words from vulgar and profane words, the tension level is detected as high.

Zhang et al. proposed a Pattern-based Topic Detection and Analysis System (PTDAS) on Weibo, a Chinese Twitter-like platform [33]. For this purpose, they have developed three different modules: Topic detection, Evolutionary analysis, and Sentimental analysis. A key component of their method is to employ an FP-growth-like algorithm to mine cosine interesting patterns from a set of tweets.

3.2 Statistical Analysis

Given a set of keywords, various statistical models such as pLSI (probabilistic Latent Semantic Indexing) [34] and LDA (Latent Dirichlet Allocation) [35] can be used for topic detection in Twitter data streams. However, pLSI was based on the likelihood principle, and it can not assign probabilities to new documents. This was resolved by LDA, which models each document as a mixture of topics and topics as a mixture of words. Indeed, LDA is a Bayesian network that generates a document using a mixture of topics and words.

In [13], the authors used the Online LDA approach, which has been developed for modeling several latent variables (titles) in a series of texts, including words. In this approach, new events make sense with new words (for example, names of people, parties, etc.), so that the collection of words is updated each time a document arrives. The words with a lower threshold are removed from the end.

Ahuja et al. have proposed Spatio-Temporal Event Detection (STED), a probabilistic model that detects events using information from news and Twitter [36]. For this purpose, they employed timestamps and geolocation information from tweets to estimate the temporal and regional distributions of events.

Huang et al. applied LDA to identify potential topics in a Twitter data stream [37]. They first employed ST-DBSCAN (an unsupervised data clustering algorithm) to cluster the tweets of every day. Moreover, spatial, temporal, and textual patterns for every cluster have been generated. Then, they applied LDA to identify potential topics in the cluster and analyze the structure of every tweet.

Gupta et al. collected the tweets and then, for unprocessed tweets run the LDA streaming and retrieved the tweet based on the domain to which it belongs [38]. If the tweet belongs to a certain event category, then it extracts tweets using domain-based classification. Moreover, the scoring function is used to correctly identify whether the tweet is belonging to that domain or not.

Another statistical model, namely Bursty Biterm Topic Model (BBTM), has been proposed by Yan et al. to solve the data sparsity problem in topic modeling over the short texts [39]. Their work is devoted to Biterm Topic Model (BTM), which models biterms (i.e., word pairs) rather than words for effective topic modeling in short texts.

Mehrotra et al. proposed an approach for aggregating tweets in order to improve the quality of LDA-based topic modeling in microblogs [40]. They achieved this through various pooling schemes that aggregate tweets in a data preprocessing step for LDA. Their pooling schemes included Author-wise, Burst-score-wise, Temporal, and Hashtag-based Pooling.

4 THE PROPOSED METHOD: MTCW

As discussed in the previous section, the use of Timing Windows is a convenient method for event detection. The Timing Window is a slice of time that could be a quarter of an hour, an hour, or even a day. If time slices are represented with k_t

then k_0 is the first slice of time. In Table 1, different time slices from ten minutes to one day have been shown.

Time slices	10 minutes	100 minutes	One day
k_0	From 1:00 AM to 1:10 AM	From 1:00 AM to 2:40 AM	From 2016-02-01 to 2016-02-02
k_1	From 1:10 AM to 1:20 AM	From 2:40 AM to 4:00 AM	From 2016-02-02 to 2016-02-03
k_2	From 1:20 AM to 1:30 AM	From 4:00 AM to 5:40 AM	From 2016-02-03 to 2016-02-04

Table 1. Examples of time slices

L is a Timing Window that keeps the words in a fixed number of time slices. When a new word arrives, the earlier word is removed from the beginning of the Timing Window. Therefore, the size of L will always be constant. Upon the arrival of a new word, the model is rebuilt. Table 2 presents an example of a Timing Window in size of 5, indicating that L can only contain five words; if a new word arrives, the earlier word is removed.

L in k_t		Content of L in size of 5 words				
L in k_0	Word	Iran	Rouhani	ART	100$	Planes
	Frequency	735	352	439	259	199
L in k_1	Word	Unfrozen	Claims	Iran	Rouhani	ART
	Frequency	689	598	497	356	241
L in k_2	Word	Unfrozen	Iran	Claims	Rouhani	ART
	Frequency	741	656	568	522	345
L in k_3	Word	Planes	Iran	Rouhani	Unfrozen	Claims
	Frequency	659	612	563	456	450

Table 2. Examples of timing window

The words in tweets of each time slice together with their frequency make up the content of a Timing Window. When a short slice of time is selected, a sharp increase in the number of certain words in this time slice indicates a sudden event.

We define several Timing Windows and link them to each other like chains in a way that the input of the larger window will be the output of the previous smaller window. This new concept has been introduced as Multi Timing Chained Windows (Figure 1).

A stop word is defined as a word that contains no meaning or relevance by itself [20]. In other approaches, a fixed list of Stop Words is usually used. Our approach, namely MTCW, is capable of simultaneously detecting sudden events and user events as well as making a dynamic list of Stop Words in the language of a query. The idea of dynamic generation of Stop Words helps the algorithm to be compatible with the environment under any conditions. More precisely, if a new word is spread among users, new words are added to the collection of Stop Words over time.

The developed approach for event detection comprises six modules (Figure 2). Each module has its specific task, and instead of saving its output, it directly delivers

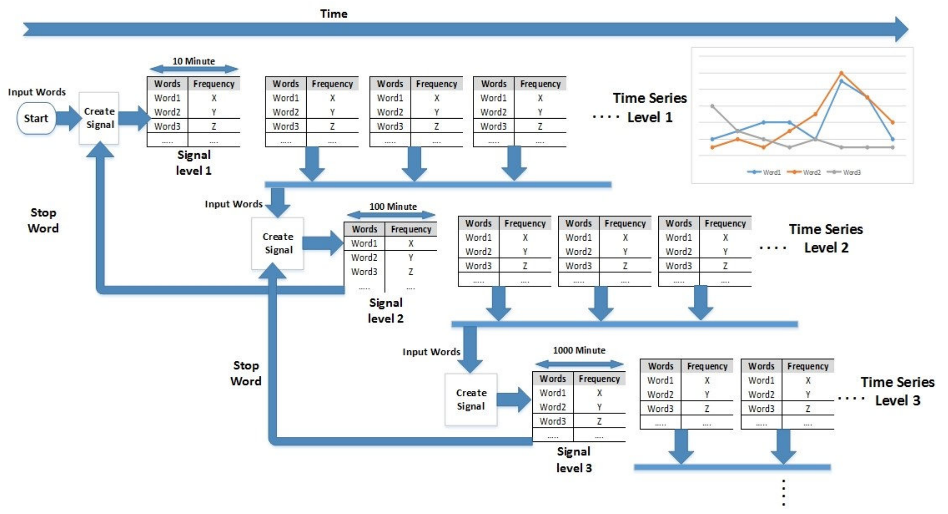


Figure 1. Different signal levels in MTCW approach

it to the next module to increase the speed and reduce the use of memory. In the following, each module is briefly explained.

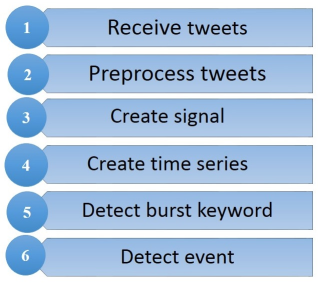


Figure 2. The layered architecture of proposed trend detection

4.1 The First Module: Receive Tweets

This module receives tweets from the Twitter Streaming API. The Twitter Streaming API continuously delivers the details of tweets requested by users in JSON format. Based on the track and location filters, this API only delivers a portion of tweets to the user.

Although MTCW, in the first step, collects tweets using Twitter Streaming API, the methodology of the proposed approach is not user-centric. In fact, user's tweets affect the functionality and the correctness of MTCW (same as all social-aware event detection systems), but the user/operator could not impose any preferences directly on the system's performances.

4.2 The Second Module: Preprocess Tweets

This module processes the text of a tweet in a straightforward way. In fact, every effort is made to spend the minimum possible time for initial processing. This module receives the text of a tweet and finally produces a sorted array of tweets along with the frequency. This module performs six operations on the text of a tweet, as shown in Pseudocode 1.

```
TweetPreprocess(string s){
    String r;
    Array result[][];
    r = ConvertLowCase(s);
    r = ConvertSpace(r);
    r = SplitBasedonSpace(r);
    r = RemoveWordsLessthan3char(r);
    for (i=0; i< length(r); i++)
        if (r[i] isInArray result)
            result[r[i]]++;
        else
            add r[i], 1 to result;
    AsortArray(result);
    return result;
}
```

Pseudocode 1: Preprocess tweets module

4.3 The Third Module: Create Signal

The assorted array of words from tweets and the frequency of each word is called signal. The signal is assorted in a descending order based on the frequency. In this study, each signal is constructed in four levels. In fact, four different Timing Windows are considered to construct a signal.

The first level signal: It should be noted that the first signal is received directly from the words of the first Timing Window tweets, while in the next levels, words from a previous level are used as the input. In this level, tweets are received in a ten-minute interval, and their text is delivered to the second module for processing. The arrays received from the second module are combined with each other to produce the signal.

The process is as follows: words in the fourth level of Timing Windows (Dynamic Stop Words) are deleted from arrays. Then, the words in the next level time series (second level) are also deleted from the array. In fact, the words that have reached the next level are those with a high frequency in the recent past, which cannot thus represent an urgent event. Frequencies of similar words are added together. Afterwards, the remaining words of the signal are arranged in descending order based on the sum of frequencies. Finally, the signal is sliced according to first level limitation parameter of the signal word. In MTCW, this parameter is set to 100 for the first level signal, i.e., 100 frequently repeated words are maintained. In this way, space limitation is observed, and there is no need for further space with an increasing number of incoming tweets.

The pseudocode of the above process has been listed in Pseudocode 2.

```
CreateSignals(inputArray, lastLevelWords, nextLevelWords, limitationSignalLevel){

    inputArray = LastLevelRemoveWords (inputArray, lastLevelWords);
    inputArray = nextLevelRemoveWords (inputArray, nextLevelWords);
    outArray = AsortArray (outArray);
    signal = CutArray (outArray , limitationSignalLevel);

    return result;
}
```

Pseudocode 2: Create signal module

Making the second, third, and fourth level signal: After few iterations of the first level signal, the second level signal is constructed. After generating ten signals in the first level, a new signal for the second level is constructed (every 100 minutes). This process is repeated to create next level signals. Indeed, after ten iterations of the second level signal, the third level signal is constructed (every 1 000 minutes), and after five iterations of third level signal, the fourth level signal is constructed.

The procedure for signal generation is as follows:

- Considering the current signal level, five to ten signals of the previous level are combined to create the signal of the next level.
- The signal integration is performed by summing up the total number of similar words (frequency).
- For the second level signals, the words that exist in the next time series (third) is deleted. The same happens for the third level signal.
- The result is assorted in a descending order based on the frequency.
- Finally, the assorted result is sliced based on the words limitations.

4.4 The Fourth Module: Create Time Series

A number of signals generated at successive time intervals is called time series. There is a distinct time series for each level. The series is updated upon the creation of each signal. The update is done as follows:

If a word is absent in a time series, it is added to the time series, and iteration is set to zero in previous signals for that word. If the word exists in the time series, a new iteration is added as the last word signal, and the oldest signal of that word is deleted. If both previous conditions are false, then a zero repeat is added to the word, and the earliest signal of that word is deleted.

The pseudocode for creating time series is listed in Pseudocode 3.

```
CreateTimeSeries(signals){
        array timeSeries[][];
        foreach (signals as signal){
                CreateNewSignalTimeSeries (timeSeries);
                foreeach (signal as word => frequency)
                        if (word is not in timeSeries){
                                AddToTimeSeries(timeSeries, word, frequency);
                                AddZeroToLastSignalTimeSeries (timeSeries, word);
                        } else{
                                AddNewFrequencyWordToOld (timeSeries, word , newFrequency);
                        }
                AddZeroOtherWordsNewSignal(timeSeries);
        }
}
```

Pseudocode 3: Create time series module

No time series is constructed for the fourth level, but its signal is updated. The reason for this is that Stop Words are kept at this level and that the highest iterations should always remain at this level. Therefore, upon the arrival of a new signal at the fourth level, the number of iterations is compared with the previous signal, and if there is a higher value in the new signal, the previous signal is updated. Finally, based on the limitation of signal words, only the most frequently repeated ones remain.

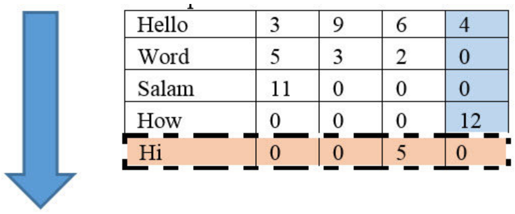
In other words, in the previous levels, the words are horizontally and vertically removed from Timing Windows, but in the fourth level, the words are removed vertically.

Vertical and horizontal update: The following time series in Table 3 includes four words and four-signal limitations. If the new signal arrives with a "How" word and iteration of 12, the word "Hi" is excluded due to the limitation of words. In this case, we say the word was out vertically (Figure 3).

Now, suppose the arrival of a new signal, including the word "Hello" with five iterations. In this case, all iterations of the word "Salam" are set to zero and are deleted from the set of time series words. Now, we say the word was out horizontally (Figure 4).

Hello	0	3	9	6
Word	2	5	3	2
Salam	0	11	0	0
Hi	1	0	0	5

Table 3. Time series in the last level



Hello	3	9	6	4
Word	5	3	2	0
Salam	11	0	0	0
How	0	0	0	12
Hi	0	0	5	0

Figure 3. Vertical update of the last level time series

4.5 The Fifth Module: Detect Burst Keywords

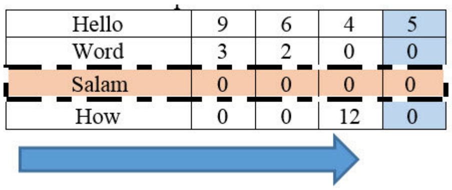
After updating the time series, the fifth module investigates a sudden increase in the iteration of words. The output of this module is the candidate of Burst Keywords.

The following criterion is used to find candidate Burst keywords:

$$\left| \frac{F_w - M_w}{M_w} \right| > BP. \tag{3}$$

In this formula,F_w represents the frequency of the word W in the new signal. M_w is the average frequency of the word W in previous signals. In addition, BP (burst parameter) is a measure of Burst Keyword that can be different for each level. In our experiments, the BP value has been experimentally considered 0.3 for the first level and 0.7 for the second and third levels, respectively.

For example, if the frequency of the word "planes" in the first level of the new signal is 1 200 ($F_w = 1\,200$) and the average frequency value of this word in previous signals is 1 000 ($M_w = 1\,000$), then this word is a candidate as a Burst Keyword in the first level of the signal ($BP = 0.3 > 0.2$). The candidate Burst Keywords are



Hello	9	6	4	5
Word	3	2	0	0
Salam	0	0	0	0
How	0	0	12	0

Figure 4. Horizontal update of the last level time series

returned together with their entire signal to calculate the similarity of their keywords in the next module.

4.6 The Sixth Module: Detect Event

We detect events by grouping a set of candidate Burst Keywords with similar burst patterns. This set includes a few candidate Burst Keywords with a sudden increase in their iteration with a similar pattern of burst in recent signals.

The similarity in the time series of candidate Burst Keywords is checked for each signal. Finding the level of similarity in the time series of two candidate Burst Keywords can indicate the similarity of their iteration pattern. If the two candidate Burst Keywords have the same iteration pattern, it can be stated that they are at the same set.

The cross-correlation measure is used to examine the behavioral similarity of two candidate Burst Keywords. The cross-correlation is a criterion to detect the similarity of two time series in signal processing. In the discrete domain, the following equation is used to calculate the cross-correlation for two time series of x and y with the length of n [41]:

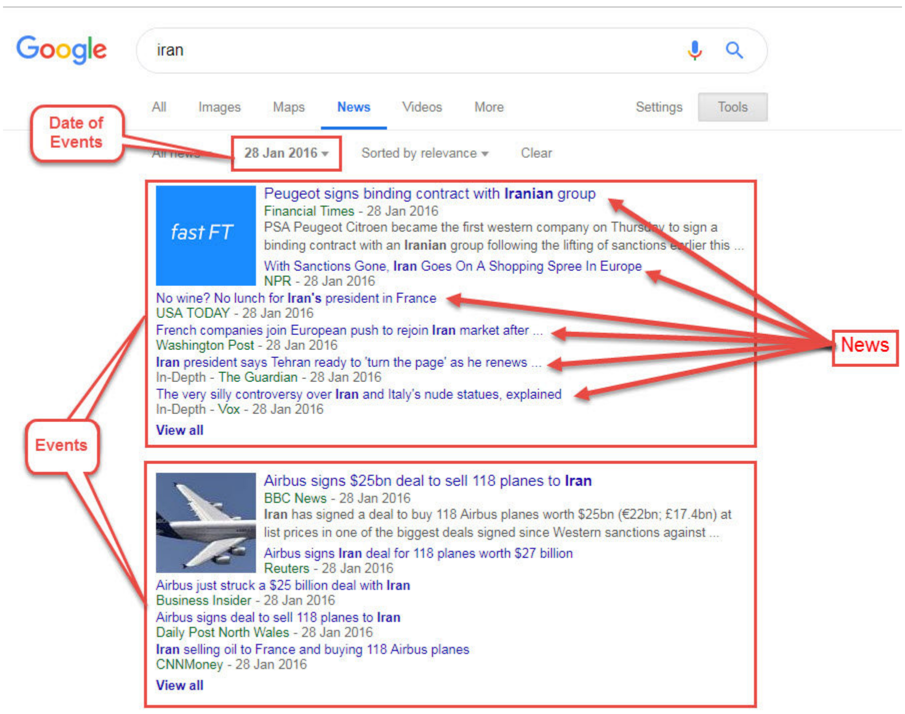
$$r = \frac{\sum_i^n [(x(i) - mx) \times (y(i-d) - my)]}{\sqrt{\sum_i^n (x(i) - mx)^2}\sqrt{\sum_i^n y(i-d) - my)^2}} \tag{4}$$

where x and y are actually two time series supposed to be measured in terms of similarity, with m_x and m_y presenting the mean of two time series. d is the lag parameter for which the value of 1 has been chosen in this algorithm.

5 EXPERIMENTAL RESULTS

A dataset has been collected using Twitter streaming API to analyze and evaluate this idea. The tweets were collected from 28. 1. 2016 to 27. 2. 2016 based on track = “iran”, “tehran”, “thr”. Over one and a half million tweets (1 524 493) were collected. Tweets from the same day were classified together, and every day was divided into ten-minute intervals. For each interval, the tweets were stored as a text file. The information collected from every tweet included ID, date, text, origin, User ID, username, and location of the user posting the tweet.

It should be mentioned that most proposed approaches in event detection use their proprietary datasets for evaluation purposes. To this end, Twitter streaming API has been employed to collect and create our proprietary dataset. In fact, to evaluate MTCW, we compared the News items related to Iran, which were detected by our proposed approach with the Google News search service. The search was done by examining Google News, searching for the word Iran, and setting the time interval from day to day. However, several methods benefit from publicly available datasets. In order to precisely evaluate MTCW and eliminate any possible inconsistency or bias in the dataset, MTCW has also been evaluated using a publicly available dataset, which has been employed by many articles. The results are reported in Section 6.2.



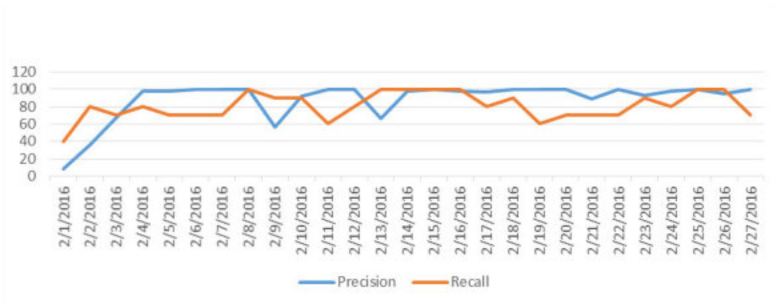
Peugeot signs binding contract with **Iranian** group
Financial Times - 28 Jan 2016
PSA Peugeot Citroen became the first western company on Thursday to sign a binding contract with an **Iranian** group following the lifting of sanctions earlier this ...
With Sanctions Gone, **Iran** Goes On A Shopping Spree In Europe
NPR - 28 Jan 2016
No wine? No lunch for **Iran's** president in France
USA TODAY - 28 Jan 2016
French companies join European push to rejoin **Iran** market after ...
Washington Post - 28 Jan 2016
Iran president says Tehran ready to 'turn the page' as he renews ...
In-Depth - The Guardian - 28 Jan 2016
The very silly controversy over **Iran** and Italy's nude statues, explained
In-Depth - Vox - 28 Jan 2016

Airbus signs $25bn deal to sell 118 planes to **Iran**
BBC News - 28 Jan 2016
Iran has signed a deal to buy 118 Airbus planes worth $25bn (€22bn; £17.4bn) at list prices in one of the biggest deals signed since Western sanctions against ...
Airbus signs **Iran** deal for 118 planes worth $27 billion
Reuters - 28 Jan 2016
Airbus just struck a $25 billion deal with **Iran**
Business Insider - 28 Jan 2016
Airbus signs deal to sell 118 planes to **Iran**
Daily Post North Wales - 28 Jan 2016
Iran selling oil to France and buying 118 Airbus planes
CNNMoney - 28 Jan 2016

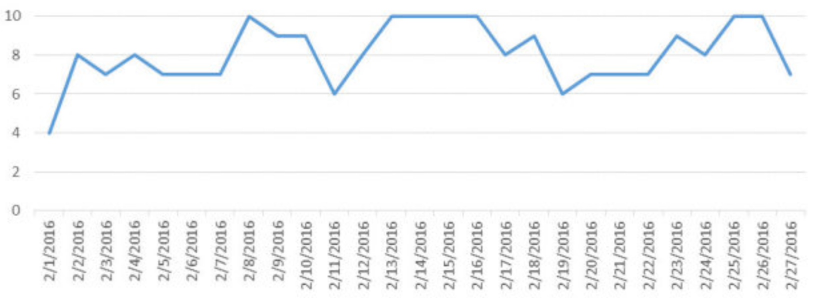
Figure 5. Google News service

As mentioned above, for the first phase of evaluation, Google New search service returns ten events related to Iran on a respective day. An event may include 1–7 news items from different News agencies. The collection of a number of news from an event contributes to a better analysis of the results. We may have detected an event but with different words and syntax. Different headlines from various news agencies concerning a single event are an idea for a better comparison of results. We combine these headlines, and if the detected event is close to this combination, then we declare the detection of that event.

The results of the evaluation related to the first level series and Google News headlines are shown in Figure 5. As it can be seen in Figure 6 b, for some days, all ten events have been detected. Also, the results have been improved over time.



a) Precission and recall for detected events



b) Detected events day by day

Figure 6. Experimental results

6 EVALUATION AND COMPARISON

In Section 5, we provided the precision and recall of our event detection approach compared to the Google News service. For further evaluation of the proposed method, quantitative and qualitative comparisons have been performed with other state-of-the-art methods in event detection in Twitter, as discussed in the next subsections.

6.1 Qualitative Comparison

In the following Table, the approaches assessed in the Related Work section have been compared based on the five objectives described in Section 2.2.

MTCW approach has the Generality feature. Since the algorithm is not sensitive to a specific subject and no word or phrase has already been tagged, it recognizes an event based on frequency patterns of words (which can be related to each topic).

Our solution does not depend on time and space (Scalability feature). In the proposed algorithm, the signal length is constant at all levels, so the algorithm only needs a constant space for storing information and the relevant history, and this space will not increase over time. The processing done in the algorithm does not change over time due to the constancy of the input, so the complexity of the time is constant.

The proposed solution requires limited preprocessing, and little time is needed to process the tweets and generate the result (Real-time processing feature). In fact, the algorithm produces the result of an event within the specified time interval without any delay at each signal level.

The use of the idea of dynamic Stop Words forms an unsupervised system (No Supervision feature). In this system, based on user behavior, meaningless words are removed over time, and meaningful words are added to the system. MTCW does not use any dictionary or additional words and data, either.

No.	**Approach**	**Generality**	**Scalability**	**Real-time processing**	**No super-vision**	**Non-use Additional Data**
1	Benhardus [20]	✓	✓	✗	✓	✗
2	TwitterMonitor [19]	✓	✓	✓	✓	✓
3	EDCoW [5]	✓	✓	✓	✓	✓
4	SasaPetrovic [42]	✗	✓	✓	✓	✓
5	Bifet [26]	✗	✗	✗	✓	✓
6	Pete Burnap [3]	✗	✓	✗	✗	✗
7	Takeshi Sakaki [43]	✗	✗	✓	✓	✗
8	Aielloi [31]	✓	✗	✓	✗	✗
9	Hyeok Jun Choi [30]	✓	✓	✓	✓	✓
10	MTCW	✓	✓	✓	✓	✓

Table 4. Qualitative comparison of different approaches

6.2 Quantitative Comparison

To evaluate MTCW, we utilized the same evaluation framework proposed by Aiello et al. [31]. They have extracted tweets about three major real-world events that occurred in 2012, which includes the FA Cup Final (FA), Super Tuesday (ST), and US Elections (US). Since they are not allowed to publicly distribute the original tweets, the distributions only contain the tweet IDs and the ground truth topics which have been organized in timeslots as explained in [1]. Indeed, Aiello et al. generated a ground truth for the dataset consisting of a manual review of published media reports about the event. This ground truth includes 13 topics for FA, 22 topics for ST, and 64 topics for US datasets [31].

The "FA Cup" dataset contains tweets posted during the final match of the Football Association Challenge Cup held on May 5, 2012. The ground truth for the FA Cup dataset comprises 13 topics, including kick off, goals, half-time, fouls, bookings, and the end of the match.

The "Super Tuesday" dataset consists of tweets posted during the US primary elections, which were held on the first Tuesday of March 2012 in ten US states. The ground truth comprises 22 topics, which represent the key moments of the elections and projections of the voting results in different states.

The "US Election" dataset contains tweets posted during the United States presidential election of 2012, which was held on November 6, 2012. The ground truth consists of 64 topics. The topics were related to the outcomes of the presidential election, derived from mainstream media.

Totally, we extracted about 200k tweets for FA, 500 k tweets for ST, and 1 100 k tweets for the US. It should be mentioned that some tweets were not downloaded as they are not available anymore. Sequiera and Lin [44] performed experiments on the long term effect of tweet removal from Tweets2013 corpus. They observed that the deletions would less likely have an impact on the ranking of systems. The details of the three datasets are given in Table 5.

Dataset	**Temporal Coverage**	**No. of Tweets**	**Total Topics**
FA Cup	6 HOURS	124 524	13
Super Tuesday	24 HOURS	540 241	22
US Election	36 HOURS	2 335 105	64

Table 5. Datasets details

We have compared the precision and recall of MTCW with some well-known methods based on the above datasets in Table 6. The baselines selected for evaluation include the state-of-the-art event detection systems and cover a wide range of techniques in this domain, such as BNgram [31], Frequent Pattern Mining (FPM) [31], Soft Frequent Pattern Mining (SFPM) [27], Graph-based feature-pivot (GFeat-p) [31], and a probabilistic topic model-based LDA [45]. Moreover, we have compared the performance of MTCW with a recently published method, HUPM [30]. As shown in Table 6, our approach proposes competitive or even better results with state-of-the-art event detection approaches.

The experimental results for the FA dataset show that both precision and recall are the highest for MTCW. Similarly, for the ST dataset, MTCW precision is the best one after FPM, and MTCW recall is very close to SFPM, which offers the best recall among others. The evaluation results from the US dataset indicates that MTCW precision is the highest, and its recall is very competitive among the compared methods.

Method	**FA Cup**			**Super Tuesday**			**US Elections**		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
BNgram	0.2989	0.5778	0.394	0.6286	0.6471	0.6377	0.4050	0.5632	0.4712
FPM	0.7500	0.4286	05455	1.0000	0.4091	05807	0.0000	0.0000	0
SFPM	0.2336	0.6579	0.3448	0.4717	0.8929	0.6173	0.2412	0.6953	0.3582
GFeat-p	0.0000	0.0000	0	0.3750	0.6000	0.4615	0.3750	0.4839	0.4225
LDA	0.1637	0.6829	0.2641	0.0000	0.0000	0	0.1654	0.6286	0.2619
HUPM	0.3200	0.6000	0.4174	0.4860	0.7080	0.5764	0.3520	0.5650	0.4338
MTCW	0.8500	0.7250	0.7825	0.7750	0.8234	0.7985	0.4650	0.6125	0.5287

Table 6. Quantitative comparison of different approaches

7 CONCLUSIONS

In this paper, we proposed a new approach for event detection in Twitter using Multi Timing Chained Windows (MTCW). In our method, the time and space complexity are constant for any number of tweets, and the approach is also independent of user language. We examined MTCW on Iran-related tweets over a period of 27 days. More than 1.5 million tweets related to Iran were collected in this period. Using the Google News service, news about Iran were categorized within the period of tweets collection, which were used to evaluate the results. The results indicate the high precision of the proposed method. Moreover, common datasets such as FA Cup Final, Super Tuesday, and US Elections have been employed to compare MTCW with baselines and recent approaches.

As future works, it is suggested to collect the tweets based on their location instead of their subject to improve the effectiveness of the results, but we should be aware of the complexity of this dataset. In addition, the TF-IDF approach can be used instead of frequency to create signals. Also, by increasing the number of windows, a closer examination of the precision of this method will become possible.

REFERENCES

[1] Twitter by the Numbers. 2021, available at: `https://www.omnicoreagency.com/twitter-statistics`.

[2] Company – About – Twitter. 2021, available at: `https://about.twitter.com/company`.

[3] Burnap, P.—Rana, O. F.—Avis, N.—Williams, M.—Housley, W.—Edwards, A.—Morgan, J.—Sloan, L.: Detecting Tension in Online Communities with Computational Twitter Analysis. Technological Forecasting and Social Change, Vol. 95, 2015, pp. 96–108, doi: 10.1016/j.techfore.2013.04.013.

[4] Klein, B.—Castanedo, F.—Elejalde, I.—López-de-Ipiña, D.—Prada Nespral, A.: Emergency Event Detection in Twitter Streams Based on Natural Language Processing. In: Urzaiz, G., Ochoa, S. F., Bravo, J., Chen, L. L., Oliveira, J. (Eds.): Ubiquitous Computing and Ambient Intelligence. Context-Awareness and Context-Driven Interaction. Springer, Cham, Lecture Notes in Computer Science, Vol. 8276, 2013, pp. 239–246, doi: 10.1007/978-3-319-03176-7_31.

[5] Weng, J.—Yao, Y.—Leonardi, E.—Lee, F.: Event Detection in Twitter. Development, 2011, pp. 401–408.

[6] Gerber, M. S.: Predicting Crime Using Twitter and Kernel Density Estimation. Decision Support Systems, Vol. 61, 2014, pp. 115–125, doi: 10.1016/j.dss.2014.02.003.

[7] Dong, G.—Yang, W.—Zhu, F.—Wang, W.: Discovering Burst Patterns of Burst Topic in Twitter. Computers and Electrical Engineering, Vol. 58, 2017, pp. 551–559, doi: 10.1016/j.compeleceng.2016.06.012.

[8] Petrović, S.—Osborne, M.—McCreadie, R.—Macdonald, C.—Ounis, I.—Shrimpton, L.: Can Twitter Replace Newswire for Breaking News? Proceedings of

the Seventh International AAAI Conference on Weblogs and Social Media (ICWSM-13), Boston, MA, USA, 2013.

[9] Osborne, M.—Dredze, M.: Facebook, Twitter and Google Plus for Breaking News: Is There a Winner? Proceedings of 8th International Conference on Weblogs and Social Media (ICWSM 2014), 2014, pp. 611–614.

[10] Allan, J.—Lavrenko, A. V.—Jin, H.: First Story Detection in TDT Is Hard. Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM 2000), 2000, pp. 374–381, doi: 10.1145/354756.354843.

[11] Hasan, M.—Orgun, M. A.—Schwitter, R.: A Survey on Real-Time Event Detection from the Twitter Data Stream. Journal of Information Science, Vol. 44, 2018, No. 4, pp. 443–463, doi: 10.1177/0165551517698564.

[12] Allan, J.: Topic Detection and Tracking: Event-Based Information Organization. Kluwer Academic Publishers, 2002, doi: 10.1007/978-1-4615-0933-2.

[13] Lau, J. H.—Collier, N.—Baldwin, T.: On-Line Trend Analysis with Topic Models: #twitter Trends Detection Topic Model Online. International Conference on Computational Linguistics (COLING 2012), Vol. 2, 2012, pp. 1519–1534.

[14] Rafea, A.—GabAllah, N. A.: Topic Detection Approaches in Identifying Topics and Events from Arabic Corpora. Procedia Computer Science, Vol. 142, 2018, pp. 270–277, doi: 10.1016/j.procs.2018.10.492.

[15] Madani, A.—Boussaid, O.—Zegour, D. E.: Real-Time Trending Topics Detection and Description from Twitter Content. Social Network Analysis and Mining, Vol. 5, 2015, Art. No. 59, doi: 10.1007/s13278-015-0298-5.

[16] Gaglio, S.—Lo Re, G.—Morana, M.: A Framework for Real-Time Twitter Data Analysis. Computer Communications, Vol. 73, 2016, Part B, pp. 236–242, doi: 10.1016/j.comcom.2015.09.021.

[17] Comito, C.—Forestiero, A.—Pizzuti, C.: Bursty Event Detection in Twitter Streams. ACM Transactions on Knowledge Discovery from Data, Vol. 13, 2019, No. 4, Art. No. 44, 28 pp., doi: 10.1145/3332185.

[18] Becker, H.—Naaman, M.—Gravano, L.: Beyond Trending Topics: Real-World Event Identification on Twitter. Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media (ICWSM), 2011.

[19] Mathioudakis, M.—Koudas, N.: TwitterMonitor: Trend Detection over the Twitter Stream. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10), 2010, pp. 1155–1158, doi: 10.1145/1807167.1807306.

[20] Benhardus, J.—Kalita, J.: Streaming Trend Detection in Twitter. International Journal of Web Based Communities (IJWBC), Vol. 9, 2013, No. 1, pp. 122–139, doi: 10.1504/ijwbc.2013.051298.

[21] Fawcett, T.: An Introduction to ROC Analysis. Pattern Recognition Letters, Vol. 27, 2006, No. 8, pp. 861–874, doi: 10.1016/j.patrec.2005.10.010.

[22] Atefeh, F.—Khreich, W.: A Survey of Techniques for Event Detection in Twitter. Computational Intelligence, Vol. 31, 2015, No. 1, pp. 132–164, doi: 10.1111/coin.12017.

[23] Guzman, J.—Poblete, B.: On-Line Relevant Anomaly Detection in the Twitter Stream: An Efficient Bursty Keyword Detection Model. Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description (ODD '13), 2013, pp. 31–39, doi: 10.1145/2500853.2500860.

[24] Zhang, C.—Lei, D.—Yuan, Q.—Zhuang, H.—Kaplan, L.—Wang, S.—Han, J.: GeoBurst+: Effective and Real-Time Local Event Detection in Geo-Tagged Tweet Streams. ACM Transactions on Intelligent Systems and Technology, Vol. 9, 2018, No. 3, Art. No. 34, 24 pp., doi: 10.1145/3066166.

[25] Zhang, C.—Zhou, G.—Yuan, Q.—Zhuang, H.—Zheng, Y.—Kaplan, L.—Wang, S.—Han, J.: GeoBurst: Real-Time Local Event Detection in Geo-Tagged Tweet Streams. Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '16), 2016, pp. 513–522, doi: 10.1145/2911451.2911519.

[26] Bifet, A.—Holmes, G.—Pfahringer, B.: MOA-TweetReader: Real-Time Analysis in Twitter Streaming Data. In: Elomaa, T., Hollmén, J., Mannila, H. (Eds.): Discovery Science (DS 2011). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6926, 2011, pp. 46–60, doi: 10.1007/978-3-642-24477-3_7.

[27] Petkos, G.—Papadopoulos, S.—Aiello, L.—Skraba, R.—Kompatsiaris, Y.: A Soft Frequent Pattern Mining Approach for Textual Topic Detection. Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS '14), 2014, Art. No. 25, 10 pp., doi: 10.1145/2611040.2611068.

[28] Gaglio, S.—Lo Re, G.—Morana, M.: Real-Time Detection of Twitter Social Events from the User's Perspective. 2015 IEEE International Conference on Communications (ICC), 2015, pp. 1–6, doi: 10.1109/icc.2015.7248487.

[29] Huang, J.—Peng, M.—Wang, H.: Topic Detection from Large Scale of Microblog Stream with High Utility Pattern Clustering. Proceedings of the 8th Workshop on Ph.D. Workshop in Information and Knowledge Management, 2015, pp. 3–10, doi: 10.1145/2809890.2809894.

[30] Choi, H.-J.—Park, C. H.: Emerging Topic Detection in Twitter Stream Based on High Utility Pattern Mining. Expert Systems with Applications, Vol. 115, 2019, pp. 27–36, doi: 10.1016/j.eswa.2018.07.051.

[31] Aiello, L. M.—Petkos, G.—Martin, C.—Corney, D.—Papadopoulos, S.—Skraba, R.—Göker, A.—Kompatsiaris, I.—Jaimes, A.: Sensing Trending Topics in Twitter. IEEE Transactions on Multimedia, Vol. 15, 2013, No. 6, pp. 1268–1282, doi: 10.1109/tmm.2013.2265080.

[32] Xie, W.—Zhu, F.—Jiang, J.—Lim, E.-P.—Wang, K.: TopicSketch: Real-Time Bursty Topic Detection from Twitter. IEEE Transactions on Knowledge and Data Engineering, Vol. 28, 2016, No. 8, pp. 2216–2229, doi: 10.1109/tkde.2016.2556661.

[33] Zhang, L.—Wu, Z.—Bu, Z.—Jiang, Y.—Cao, J.: A Pattern-Based Topic Detection and Analysis System on Chinese Tweets. Journal of Computational Science, Vol. 28, 2018, pp. 369–381, doi: 10.1016/j.jocs.2017.08.016.

[34] Hofmann, T.: Probabilistic Latent Semantic Indexing. Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '99), 1999, pp. 50–57, doi: 10.1145/312624.312649.

[35] Blei, D. M.—Ng, A. Y.—Jordan, M. I.: Latent Dirichlet Allocation. Journal of Machine Learning Research, Vol. 3, 2003, pp. 993–1022.

[36] Ahuja, A.—Baghudana, A.—Lu, W.—Fox, E. A.—Reddy, C. K.: Spatio-Temporal Event Detection from Multiple Data Sources. In: Yang, Q., Zhou, Z. H., Gong, Z., Zhang, M. L., Huang, S. J. (Eds.): Advances in Knowledge Discovery and Data Mining (PAKDD 2019). Springer, Cham, Lecture Notes in Computer Science, Vol. 11439, 2019, pp. 293–305, doi: 10.1007/978-3-030-16148-4_23.

[37] Huang, Y.—Li, Y.—Shan, J.—Huang, Y.—Li, Y.—Shan, J.: Spatial-Temporal Event Detection from Geo-Tagged Tweets. ISPRS International Journal of Geo-Information, Vol. 7, 2018, No. 4, Art. No. 150, 21 pp., doi: 10.3390/ijgi7040150.

[38] Gupta, M.—Gupta, P.: Research and Implementation of Event Extraction from Twitter Using LDA and Scoring Function. International Journal of Information Technology, Vol. 11, 2019, No. 2, pp. 365–371, doi: 10.1007/s41870-018-0206-0.

[39] Yan, X.—Guo, J.—Lan, Y.—Xu, J.—Cheng, X.: A Probabilistic Model for Bursty Topic Discovery in Microblogs. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI '15), 2015, pp. 353–359.

[40] Mehrotra, R.—Sanner, S.—Buntine, W.—Xie, L.: Improving LDA Topic Models for Microblogs via Tweet Pooling and Automatic Labeling. Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '13), 2013, pp. 889–892, doi: 10.1145/2484028.2484166.

[41] Montgomery, D. C.—Jennings, C. L.—Kulahci, M.: Introduction to Time Series Analysis and Forecasting. Second Edition. Wiley, 2015.

[42] Petrović, S.: Real-Time Event Detection in Massive Streams. Ph.D. Thesis, University of Edinburgh, 2012.

[43] Sakaki, T.—Okazaki, M.—Matsuo, Y.: Earthquake Shakes Twitter Users: Real-Time Event Detection by Social Sensors. Proceedings of the 19th International Conference World Wide Web (WWW '10), 2010, pp. 851–860, doi: 10.1145/1772690.1772777.

[44] Sequiera, R.—Lin, J.: Finally, a Downloadable Test Collection of Tweets. Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17), 2017, pp. 1225–1228, doi: 10.1145/3077136.3080667.

[45] Teh, Y. W.—Newman, D.—Welling, M.: A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. In: Schölkopf, B., Platt, J., Hoffman, T. (Eds.): Advances in Neural Information Processing Systems 19 (NIPS 2006), 2006, pp. 1353–1360.

Mohammad Mahdi Mojiri received his B.Sc. degree in software engineering from the University of Kashan, Iran, in 2008 and his M.Sc. degree in software engineering from the Central Tehran Branch, Islamic Azad University, Tehran in 2017. His research interests include data mining, social network analysis, and stream processing.

Reza Ravanmehr graduated in computer engineering from the Shahid Beheshti University, Tehran, in 1996. After that, he gained M.Sc. and Ph.D., both in computer engineering from the Islamic Azad University, Science and Research Branch, Tehran, in 1999 and 2004, respectively. His main research interests are distributed/parallel systems, large-scale data management systems, and social network analysis. He is a faculty member of the Computer Engineering Department at the Central Tehran Branch, Islamic Azad University, since 2001.