Feature Selection Algorithm Used to Classify Faults in Turbine Bearings

Mohamad KHALIL[1], Joelle AL HAGE[1], Khaled KHALIL[2]

[1]Azm center for research in biotechnology, EDST, Lebanese University, Tripoli, Lebanon

[2]MMC Laboratory, Lebanese University, Engineering faculty, Tripoli, Lebanon

Mohamad.khalil@ul.edu.lb; joellealhage@hotmail.com; khkhalil@ul.edu.lb

Abstract

Feature Selection is a very important step that select a few number of feature used for the classification in order to reduce execution time, to improve accuracy and to enhance performance of the identification system. In this paper we propose new feature selection methods by combining between relief, mutual information and sequential selection. The new approach is compared with other existing and we demonstrate some improvement when they are applied to a random dataset and on real data acquired from wind turbine bearings aiming to detect fault in the turbine using vibration signal.

Keywords

Feature Selection; Sequential Selection; Relief; Mutual Information

Introduction

Feature selection removes any redundant or irrelevant features in order to improve the accuracy and enhance performance. The feature selection problem can be described as follow: Given a set S of original feature the purpose is to find a subset Y of S that well predicts the output. This subset Y must increases the classification accuracy rate and decreases the execution time.

In any feature selection algorithm we must take into account the evaluation function. Langley grouped different feature selection methods into two groups: filter and wrapper. In filter method we use only the property of the feature without taken into account the dependence between the features. On the other side the wrapper method is based on the classification accuracy; wrapper method produces a higher accuracy as the selected features correspond to the learning algorithms but they suffer of a higher risk of over-fitting than filter methods and they have high computational cost.

There are many methods for the feature selection, we distinguish between sequential methods that use the classification as an evaluation function, and relief based on the calculation of distance and the mutual information.

The sequential forward selection is the most used method because of its simplicity and its high accuracy. In this paper we used and developed the mutual information proposed by Huawen Liu et al., 2011.

The final purpose of this paper is to find new features selection methods and view their classification performance when they are applied to classification-based random dataset and to data from wind turbine bearings. The aim of this study is to detect the fault coming from the turbine using only its vibration signal. This signal is acquired using an accelerometer and we extracted all possible features from this signal that are used in the literature for classification. Then we developed new techniques to reduce the number of these parameters before applying these selected ones to the different classifiers.

Parameter Extracted from Vibration Signal

In our study 41 parameters have been extracted from the vibration signal.

Statistical Parameters

Statistical parameters that can be calculated are: the mean, (m), the variance (sigma), the RMS (rms) and the

2324-7037/15/01 001-08

doi: 10.12783/ijcsa.2015.0401.01

statistical skewness (ss) and the statistical kurtosis (sk) of the original signal.

Parameters Related to the Power Spectral Density:

Several frequency parameters have been extracted from the power spectral density (PSD), $S_x(f)$. In our work, we use the Welch Periodogram method to calculate the power spectral density of each vibration signal of duration 10 seconds. This Welch Periodogram uses a window of type nfft, with size equal to the length of signal/2, with 50% overlap. Therefore the number of used windows equals to three. 21 frequency parameters are extracted from this PSD: mean frequency MPF, Peak Frequency PF, Peak-to-Average Power Ratio, deciles D1…D9 which contain the median frequency D5 and the energies in ten equal frequency bands of $S_x(f)$ B1..B10. Deciles correspond to the frequencies D1…D9 that divide the power spectral density into parts containing 10% of the total energy.

Parameters Extracted from Wavelet Decomposition:

Some authors have also used time-frequency methods, such as wavelet decomposition, to characterize the non-stationary characteristics of the vibration signal. In our work, we used the wavelet Daubechies 4. This choice is done after studying compared several types of wavelets. After decomposition of each vibration signal into detail coefficients, we calculate the variances on the following details levels 2, 3, 4, 5 and 6 (named W1, W2, W3, W4 and W5). The choice of the details depends on the sampling frequency of our signal (sample rate equal to 1000 Hz). These selected details represent more than 96% of the signal energy and cover the frequency band of interest.

In total, we have 41 parameters extracted from the vibration signals: m, sigma, RMS, min, max, ss, sk, power of signal, mean frequency, median frequency, peak of frequency, dissymmetry coefficient, B1... B10 (energy in ten equal frequency bands), D1...D9 (the deciles), ratio H/L, spectral entropy H, envelope, the variance of the details and the approximation obtained after the decomposition of the signals into five levels.

Feature Selection

We will rapidly describe the principle of Sequential Forward Selection, Relief and Mutual Information.

Sequential forward Sequential (SFS)

This method is introduced by and it is a part of wrapper method. The principle of this method is to start with an empty subset and at each time we add a feature that maximizes the value of the objective function J (used to evaluate the candidates).In our case we use the classification error to evaluate the candidates. So the SFS method proposed the best subset of features starting with an empty set.

The SFS begin with an empty set of features $Y=\{\Phi\}$. The feature x^+ that give the best classification rate $J(Y+x^+)$ is added to Y. This process continue until the classification rate stops increasing.

The algorithm of SFS is shown below:

1. Start with an empty set of features $Y_0=\{\Phi\}$
2. Select the best next feature:
   $x^+=\arg\max_{x^+ \notin Y_K} [J(Y_K + x^+)]$.
3. If $J(Y_k + x^+)] > J(Y_k)$.
   a. Update $Y_{k+1}= Y_k + x^+$, $k=k+1$
   b. Go to step 2
4. End

Relief

Relief is initially proposed by Kira and Rendell. Relief is based on the calculation of distance and to each feature it calculates his weight, the feature with weight greater or equal to a defined threshold is selected. The original relief only works with binary classes. An extension of relief is Relief-F[4] which works with real classes.

In relief we must find the Near Hit and the Near Miss. Near Hit is a neighbor from the same class which have the minimum distance with selected instance and Near Miss is a neighbor from the other classes. In original relief we

find only one Near Hit and one Near Miss from every class. In Relief-F we can find k neighbors in order to reduce the noise.

The relief-F algorithm is as shown below:

Relief(D, S, No Sample, Threshold)

1. Start with an empty set Y=Φ
2. Initialise all weight, $W_{i,}$ to zero
3. for i=1:m (m=No Sample)
   a. Randomly choose an instamce R_i from D
   b. Find k Nearest Hits (H_t)
   c. For each class different from class(R_i) apply: From the class C find K Near Miss ($M_t(C)$)
   d. For j=1:N (N number of parameter)

$$W_j = W_j - \sum \frac{diff(R_j,H_{j,t})^2}{m*k} + \sum_{C \neq class(R_i)} \frac{\frac{P(C)}{1-P(class(R_i))} \sum_{t=1}^{k} diff(R_j,M_{j,t})^2}{m*k} \quad (1)$$

4. Choose the parameter with W_j>threshold
5. End

Case of binary variable:

$$diff(R_j,I_{j,t}) = \begin{cases} 0 \text{ if } R_j \text{ and } I_{j,t} \text{ are equal} \\ 1 \text{ if the values are different} \end{cases}$$

Case of continuous variable:

$$diff(R_j,I_{j,t}) = \frac{Value(R_j) - Value(I_{j,t})}{max(x_j) - min(x_j)} \quad (2),$$

$max(x_j)$ it's the maximal value of the feature x_j

$Value(R_j)$ is the value of the feature x_j for the instance R.

The relief method can't remove redundant feature.

Mutual Information with Clustering

The mutual information MI give the relevance between two variables and it's represented by the above formula:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \quad (3)$$

Instead of calculate only the mutual information between the feature and the classes we will take into account the dependence between the features. have proposed a method that use the clustering with mutual information for feature selection.

We define:

$$S_b(C_k,Y) = \sum I(y, C_k) \; y \in Y \quad (4)$$

which represent the mutual information between the selected features and the classes,

$$CR(y,f) = \frac{I(y,f)}{H(f)} \quad (5)$$

where: I(y,f) is the mutual information between the selected features y and the candidate feature f ,H the entropy,

$$S(f) = \sum_{y \in Y} CR(y,f) \quad (6)$$

and

$$S_w(YUf)=S_w(Y)+S(f) \quad (7)$$

which represents the within-cluster distance.

Finally we calculate an objective function for evaluating the candidate:

$$J(f)=\frac{S_b(C_k,Y)+I(f,C_k)}{|Y|+S_w(Y)+S(f)} \quad (8)$$

with $|Y|$ the number of elements in Y.

The proposed algorithm is given below:

1. Initialize : S=Set of all features, Y=Φ, C_k=C
2. Calculate the distance $S_b(C_k,f)=I(f;C)$ for each feature f
3. S=S-{f}/ f that give I maximal , Y={f}; S_w=0
4. while $|Y|<\delta$ do
   a. for each candidate f∈ S
      i) Calculate J(f)
      ii) Choose f that maximize J(f)
      iii) Y=YU{f}; S=S-{f}
      iv) $S_w=S_w+S(f)$, $S_b=S_b+I(f;C_k)$

   End

To calculate the mutual information we discretize the features values.

New Method for Feature Selection

Modify Relief

When using the relief method we should determine a threshold. But the choice of this threshold isn't easy. In order to deal with this problem we propose to use the classification as an evaluation function to select the best feature given by the relief method.

The proposed algorithm is as shown below:

1. Initialize the set of features Y={Φ};
   S={set of all the features }
2. while the classification error decrease and it's greater then a fixed value do:
   a. Calculate the weight of all features in S using then Relief-F algorithms
   b. Find the best feature, the one with maximal weight , x_i
   c. Y=Y U {x_i}
   d. Realize the classification by using only the features in Y
   e. Find the classification error by using the confusion matrix
   f. Y=Y-{x_i}

   End

SFS after Relief

In order to arrive to a method that improves the accuracy of the others we will try to use the relief algorithm with sequential forward selection (SFS). Relief does not take into account the dependence into the parameter and the SFS suffer of high computational cost. But when we apply the SFS only to the features selected by the relief algorithm the computational cost will decrease and we should arrive to better results than relief and SFS. In this case the problem of choice of threshold is solved.

Relief with Mutual Information

The feature selection using the mutual information as it given by take into account the dependence between the

features. But this method in some case selects unsuitable features. In order to arrive to a better method for feature selection we propose to use relief with mutual information (Relief+MI).

The proposed algorithm is as shown below:

1. Apply the mutual information algorithm in a way to select all the features
2. Save the objective function J of each feature using the mutual information
3. Find the weight of all the features using relief algorithm
4. Knowing J and the weight of each feature: multiply these values for each feature
5. Select the features with product value greater than a determined threshold or choose a number k of features.

Performance Evaluation on Random Dataset

To evaluate the performance of the proposed algorithms we must compare them with existing methods like relief, mutual information, and SFS.

At first we will try our algorithms on a classification-based random dataset and view the performance of each method. We suppose that we have 30 features over 200 observations. 26 features are chosen randomly and the other 4 features are chosen like way they distinguish between the classes. We suppose that we have 2 classes.

The number of features selected by the mutual information is fixed to 4 and the number of features selected by the other methods is determined by their proper conditions.

For the evaluation of performance we adopt the 10-fold cross validation apply to 10 generations of the 30 features.

We take several cases of intersection between the two classes. We begin with case where the 2 classes are separable, using the 4 significant features, then we increase this intersection.

The classification method used is the K nearest neighbors (KNN) with k=10.

After execution of our algorithm on this random dataset we conclude that the feature selection improve the classification accuracy after comparison with case without selection. We notice that the SFS after relief leads to very good results with the smaller number of features selected and to a lower computational cost. The method Relief With Mutual Information improve, when the intersection are not so large, the results given by the mutual information but it drawbacks that it selects a large number of features even when a small number is enough for a good classification and it suffers from high computational cost. The results given by Modify relief are not so propitious. Table 1 shows the errors of different selection methods in function of increasing the intersection between the classes, d represent the intersection, if d increase the intersection increase. Table 2 shows the number of features selected by each method.

Now, we will suppose that the number of features is previously set to 5. In this case the method relief with mutual information improves the results given by mutual information when the intersections are not so large. The method SFS after relief is one from the best methods and it improves in the most case the result given by the SFS method. The Table 3 shows the error of different methods when the number of features is set to 5 and it shows also the results given by the genetic algorithm. The results of the genetic algorithm are interesting but not optimal because of its random character.

From these tables we can conclude that the results given by our proposed methods improve in general the results given by the others existing methods.

Performance Evaluation on Wind Turbine Bearings

The method proposed in this work is tested on signals measured on a test bench of a wind turbine in our laboratory. (Figure 1).

The mobile part of the wind turbine has been simulated by a test bench but the propeller is replaced by a motor with controlling the speed of rotation.

We distinguish between four classes: normal operation, pulley default, large pulley default and bearing default.

We use a 3D accelerometer and we wish visualize the contribution of each direction in the classification rate. We placed the accelerometer in different position on this testing bunch.

We calculate 41 features over 280 observations where 70 came from normal operation, 70 with pulley default, 70 with large pulley default and 70 with bearing default. For each type of default, we acquired signals for three different rotation speeds. For this dataset we will apply our proposed algorithms and we will compare their performances with other existing. We use the "Labview" for acquisition and "matlab" for treatment.

TABLE 1. THE AVERAGE ERROR IN % OF DIFFERENT SELECTION METHOD IN FUNCTION OF INCREASING THE INTERSECTION BETWEEN THE CLASSES.

Method	d=0.1	d=0.2	d=0.3	d=0.4	d=0.5	d=0.6	d=0.7	d=0.8	d=0.9	d=1
Relief	**0**	**0**	**0**	**0.4**	**3.9**	**5.7**	**7.4**	**15.3**	**16.1**	**17.8**
Modify relief	**0**	**0.2**	**0.3**	**0.7**	**4.6**	**4.9**	**7.6**	**14.9**	**17.5**	**21.8**
MI	**0**	**0**	**0.3**	**0.4**	**4.0**	**4.9**	**7.5**	**14.7**	**15.3**	**18.2**
Relief+MI	**0**	**0**	**0**	**0.4**	**4.0**	**4.9**	**7.5**	**14.7**	**15.3**	**18.9**
SFS	**0**	**0.1**	**0**	**1.3**	**5.1**	**5.3**	**8.3**	**16.7**	**19.9**	**21.5**
SFS after relief	**0**	**0**	**0**	**0.5**	**3.3**	**5**	**7.2**	**14.6**	**17.9**	**20.4**
Without selection	**0.6**	**0.1**	**0.3**	**2.5**	**5.3**	**6**	**11.1**	**17**	**17.4**	**19.8**

TABLE 2. NUMBER OF SELECTED FEATURE IN FUNCTION OF INCEASING THE INTERSECTION BETWEEN THE CLASSES.

Method	d=0.1	d=0.2	d=0.3	d=0.4	d=0.5	d=0.6	d=0.7	d=0.8	d=0.9	d=1
Relief	**4**	**4**	**4**	**4**	**4.1667**	**4.1667**	**5.1667**	**5.9167**	**6.1667**	**9.5833**
Modify relief	**1**	**1.75**	**2.9167**	**4.5833**	**5.333**	**5**	**5.25**	**4.75**	**4.3333**	**3.8333**
MI	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
Relief+MI	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**10.667**
SFS	**1**	**1.333**	**2.25**	**3.9167**	**5.6667**	**5.1667**	**5.58**	**5.0833**	**5.1667**	**2.25**
SFS after relief	**1**	**1.333**	**2.333**	**3.75**	**3.8333**	**3.9167**	**4.33**	**4.5**	**3.75**	**2.5833**

TABLE 3. AVERAGE ERROR IN % WHEN NUMBER OF FEATURES IS SET TO 5.

Method	d=0.1	d=0.2	d=0.3	d=0.4	d=0.5	d=0.6	d=0.7	d=0.8	d=0.9	d=1
Relief	**0**	**0**	**0**	**1.2**	**1.7**	**4**	**9.3**	**10.3**	**16.2**	**18.7**
MI	**0**	**0**	**0**	**1.5**	**1.8**	**3.7**	**7.7**	**10.2**	**16.5**	**19.5**
Relief+MI	**0**	**0**	**0**	**1.3**	**1.8**	**3.5**	**8.7**	**10.3**	**17.7**	**19.2**
SFS	**0**	**0.3**	**0.3**	**1.5**	**3**	**4**	**11.3**	**12.7**	**16.8**	**21.7**
SFS after relief	**0**	**0**	**0**	**0.5**	**2.3**	**3.2**	**8.7**	**11.5**	**16.7**	**19.5**
GA	**0**	**0**	**0.1**	**1**	**2.2**	**2.8**	**7.8**	**10.3**	**16**	**19.2**

TABLE 4. AVERAGE ERROR IN %,AVERAGE NUMBER OF SELECTED FEATURE AND THE EXECUTION TIME OF EACH METHOD (SFFS: SEQUENTIAL FORWARD FLOATING SELECTION, LRS: PLUS L MINUS R, BDS: BIDIRECTIONAL SEARCH).

	Direction x of the accelerometer			**Direction y of the accelerometer**			**Direction z of the accelerometer**		
Method	**Error in %**	**Number of selected feature**	**Execution time (s)**	**Error in %**	**Number of selected feature**	**Execution time (s)**	**Error in %**	**Number of selected feature**	**Execution time(s)**
Without selection	1.7	41	-	7.3	41	-	13.9	41	-
Relief	10.5	6	0.2652	4.6	5	0.3432	14.1	13	0.2652
Modify relief	12.9	7.4	2.8236	0.4	4	1.8096	58.7	3.8	2.3712
MI	19.4	4	1.2168	6.9	4	1.0764	44.6	4	1.1856
Relief+MI	19.8	4	40.0455	0.7	3	35.7242	44.1	3	34.0394
SFS	2.5	3.2	1.8252	0.3	3.2	1.716	6.9	5.8	2.0904
SBS	1.6	9.5	9.3289	5.2	18.4	6.2556	12.2	9.8	8.8297
SFFS	2.7	3.9	3.4008	0.4	4	4.8984	14.5	4	4.1184
BDS	2.2	5	46.2075	18.5	5	58.984	14.9	5	36.6134
LRS	1.9	6	8.2525	0.4	6	8.1433	14.2	6	6.8172
SFS after relief	0	4	0.936	0.6	3.3	1.4196	7.3	4.4	1.1232

The classification method used is the K nearest neighbors (KNN) with k=1 because it leads to lower errors than the case k=10.

The Table 4 shows the average error in %, the average number of selected feature and the execution time of each method for the 3 directions x, y and z of the accelerometer.

As we see the x direction is the most significant as it gives the smallest classification error. The z direction leads to the higher error so this direction is lowly affected by the default state of the machine. The z direction will not give us any propitious information.

The proposed method SFS after relief lead to the best performance with no classification error when we use only the x direction of the accelerometer with 4 features selected and an execution time of 0.936s which is lower than the case using the SFS algorithm . So SFS after relief lead to very good result and select a few numbers of features with an interesting execution time.

When we work on the y direction our proposed method; Modify relief; improves largely the results given by relief. In fact, the average error given by modify relief is 0.4% which is very low compare with relief that give an error of 4.6%. The execution time of modify relief is slightly higher than relief. The merge of Relief and MI improves the results given by MI and relief regarding the classification error but it leads to a very high computational cost. SFS after relief gives good results but they are not the optimal. So our proposed methods can improve the results given by the other methods in many cases.

After these studies we noticed that the most important features that are selected are: coefficient of dissymmetry, B1, B3,the variance of the fifth details, entropy, Kurtosis.

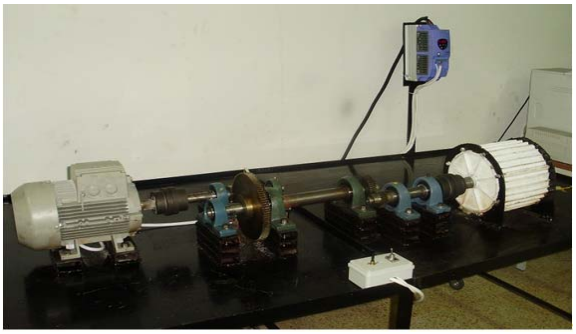


FIGURE 1. THE TESTING BUNCH.

Conclusion

In this paper we have proposed new methods for feature selection. At the beginning, we show that the feature selection decreases the classification error.

Then we perform feature selection on dataset from wind turbine elements and find that our proposed methods SFS after relief leads to a classification with no errors when we work on the x direction of the accelerometer. When we work on the y direction the Modify relief and the merge of Relief and MI improve largely the results given by MI and relief.

The most important selected features to detect the fault in the turbine are: coefficient of dissymmetry, B1, B3, the variance of the fifth detail (wavelet transform), entropy, spectral Kurtosis.

K nearest neighbour methods was used as classifier. Now, we have to increase our dataset and use a neural network to decrease the number of errors in classification. Test on more real signals must be done using only the six pertinent features extracted using the new above methods.

ACKNOWLEDGEMENTS

Authors would like to thank the National Council for Scientific Research, CNRS-Lebanon, and the Lebanese University for their financial support for realization of the testing bunch.

REFERENCES

[1] Antonio Arauzo-Azofra, José Manuel Benitez and Juan Luis CAstro. 'A feature set measure based on relief' ,pp104-109,2004Choi, Mihwa. “Contesting Imaginaires in Death Rituals during the Northern Song Dynasty.” PhD diss.,

University of Chicago, 2008.

[2] Chiementin Xavier,'Localisation et quantification des sources vibratoires dans le cadre d'une maintenance préventive conditionnelle en vue de fiabiliser le diagnostic et le suivi de l'endommagement des composants mécaniques tournants application aux roulements à billes' , pp26-27, 2007García Márquez, Gabriel. Love in the Time of Cholera. Translated by Edith Grossman. London: Cape, 1988.

[3] Huawen Liu, Yuchang Mo, Jiyi Wangand Jianmin Zhao. 'A New Feature Selection Method based on Clustering', pp 965-969,2011.

[4] Igot Kononenko. Estimating attributes:Analysis and extensions of RELIEF. In European Conference on machine learning, pages 171-182, 1994.

[5] Jean CATALIFAUD METRAVIB RDS, 'Acoustique industrielle analyse', pp 27-32 Pollan, Michael et al., The Omnivore's Dilemma: A Natural History of Four Meals. New York: Penguin, 2006.

[6] Kenji Kira and Larry A.Rendell. A pratical approach to feature selection. In Proceedings of ninth international workshop on machine learning, pages 249-256. Morgan Kaufmann Publishers Inc,1992.

[7] Langley, P. 'Selection of relevant features in machine learning. In: *Proceedings of the AAAI Fall Symposium on Relevance'*,pp 1–5, 1994.Ward, Geoffrey C., and Ken Burns. The War: An Intimate History, 1941–1945. New York: Knopf, 2011.

[8] Manoranj Dash, H. Liu, 'Feature Selection for Classification', Department of Information Systems & Computer Science, National University of Singapore, Singapore 119260, pp 131-156, 1997.

[9] Marie-Line Zani. MESURES 754 - , pp 40-43, Avril 2013

[10] Marque C, Leman H, Voisine ML, Gondry J, Naepels P. "Traitement de l'électromyogramme utérin pour la caractérisation des contractions pendant la grossesse". RBM-News 1999; 21(9):200–11.

[11] Roberto Battiti, "Using mutual information for selecting features in supervised neural net learning", IEEE Trans. Neural Network, vol.5, no.4, pp.537-550,1994.

[12] Thomas. M. Cover and Joy. A. Thomas, Elements of Information Theory. John Wiley & Sons, 1991.

[13] Whitney, A.W. A Direct Method of Nonparametric Measurement Selection. IEEE Trans. in Computational, 1100--1103 (1971).

[14] Y. Paquier, "Analyse harmonique de séries temporelles irrégulière", projet de semestre, pp 4-18, printemps 2009.

Mohamad Khalil was born in Akkar Atika, in Lebanon in 1973. He obtained an engineering degree in electrical and electricity from the Lebanese University, faculty of engineering, Tripoli, Lebanon in 1995.

He received the DEA in biomedical engineering from the University of Technology of Compiegne (UTC) in France in 1996. He received his Ph.D from the University of Technology of Troyes in France in 1999. He received his HDR (Habilitation a diriger des recherches) from UTC in 2006. He is currently researcher at Lebanese University. He is director of the Azm center for research in biotechnology at the doctoral school of sciences and technology at the Lebanese university. His current interests are the signal and image processing problems: detection, classification, analysis, representation and modeling of non-stationary signals, with application to biomedical signals and images.

Khaled KHALIL was born in Lebanon in 1964. He obtained from Jean Monnet University (St-Etienne, France) a master of sciences and technology on Plastics materials in 1991, a DEA in Macromolecular and composites materials in 1992 and a Ph.D in Mechanical engineering from F. Rabelais University (Tours, France) in 1995. He is researcher at F. Rabelais University between 1995 and 1997. He is actually researcher at Lebanese University since 1998 on collaboration with GeM institute at Ecole Centrale de Nantes (France). His current interests are the defaults of mechanical elements, the smart composite materials and Rheology.

Joelle AL HAGE was born in Lebanon in 1990. She received in 2013 a master 2 in fiability identification and diagnosis and an engineering degree in electrical engineering.

Now she's a PHD researcher at "université Lille 1" in France.

The Generalization Ability of Artificial Neural Networks in Forecasting TCP/IP Traffic Trends: How Much Does the Size of Learning Rate Matter?

Vusumuzi Moyo*[1], Khulumani Sibanda[2]

Department of Computer science, University of Fort Hare, P. O Box X1314, Alice, South Africa

*[1]vmoyo@ufh.ac.za; [2]ksibanda@ufh.ac.za

Abstract

Artificial Neural Networks (ANNs) have attracted increasing attention from researchers in many fields. They have proved to be one of the most powerful tools in the domain of forecasting and analysis of various time series. The ability to model almost any kind of function regardless of its degree of nonlinearity, positions ANNs as good candidates for predicting and modelling self-similar time series such as TCP/IP traffic. Inspite of this, one of the most difficult and least understood tasks in the design of ANN models is the selection of the most appropriate size of the learning rate. Although some guidance in the form of heuristics is available for the choice of this parameter, none have been universally accepted. In this paper we empirically investigate various sizes of learning rates with the aim of determining the optimum learning rate size for generalization ability of an ANN trained on forecasting TCP/IP network traffic trends. MATLAB Version 7.4.0.287's Neural Network toolbox version 5.0.2 (R2007a) was used for our experiments. The results are found to be promising in terms of ease of design and use of ANNs. We found from the experiments that, depending on the difficulty of the problem at hand, it is advisable to set the learning rate to 0.1 for the standard Backpropagation algorithm and to either 0.1 or 0.2 if used in conjunction with the momentum term of 0.5 or 0.6. We advise minimal use of the momentum term as it greatly interferes with the training process of ANNs. Although the information obtained from the tests carried out in this paper is specific to the problem considered, it provides users of Back-propagation networks with a valuable guide on the behaviour of ANNs under a wide range of operating conditions. It is important to note that the guidelines accrued from this paper are of an assistive and not necessarily restrictive nature to potential ANN modellers.

Keywords

Generalization Ability; Artificial Neural Networks; Learning Rate Size and Momentum

Introduction

Artificial Neural Networks (ANNs) have been used in many fields for a variety of applications, and proven to be reliable. Inspired by biological systems, particularly the observation that biological learning systems are built of very complex webs of interconnected neurons, ANNs are able to learn and adapt from experience. They have demonstrated to be one of the most powerful tools in the domain of forecasting and analysis of various time series. Time Series Forecasting (TSF) deals with the prediction of a chronologically ordered variable, and one of the most important application areas of TSF is in the domain of network engineering. As more applications vital to today's society migrate to TCP/IP networks it is essential to develop techniques that better understand and predict the behaviour of these systems.

TCP/IP network traffic forecasting is vital for the day to day running of large/medium scale organizations. By improving upon this task, network providers can optimize resources (e.g. adaptive congestion control and proactive network management), allowing an overall better Quality of Service (QoS). TCP/IP forecasting also helps to detect anomalies in the network. Security attacks like Denial-of-Service (DoS) or even an irregular amount of SPAM can be detected by comparing the real traffic with the values predicted by forecasting algorithms, resulting in economic gains from better resource management.

2324-7037/15/01 009-09

doi: 10.12783/ijcsa.2015.0401.02

Literature has shown that unlike all other TSF methods, ANNs can approximate almost any function regardless of its degree of nonlinearity. This positions them as good candidates for modeling non linear and self similar time series such as TCP/IP network traffic. Inspite of this huge advantage, ANNs are not completely absolved from any problems. One major issue that limits the applicability of ANN models in forecasting tasks is the selection of the optimal size of the learning rate. This has a profound influence on the generalization capabilities of the ANN. Generalization is a measure that tells us how well the ANN performs on the actual problem once training is complete. Once the ANN can generalize well, it means that it is capable of dealing with new situations such as a new additional problem or a new point on the curve or surface.

Although individual studies have been conducted and some form of heuristics provided for the selection of the size of the learning rate, none have been universally accepted as the results are largely contradictory.

In this paper the effect of different sizes of learning rates on the generalization ability of ANNs is empirically investigated. Although the results presented in this paper are for a particular case study, they provide a valuable guide for engineers and scientists who are currently using, or intend to use ANNs.

Related Work

The learning rate, also referred to as the step size parameter, determines how much the weights can change in response to an observed error on the training set. It is considered a key parameter for a successful ANN application because it controls the size of each step toward the minimum of the objective function. The learning rate is usually a value chosen between 0 and 1.

A general consensus amongst ANN practitioners is that a learning rate that is too large often moves too far in the correct direction, resulting in overshooting a valley or minimum in the error surface. This indelibly leads to longer training times, because it is continually overshooting its objective and unlearning what it has learned, leading to poor generalization ability of the ANN. On the other hand, a learning rate that is too small increases training time, resulting in the ANN taking many more steps than necessary to reach the goal and a greater likelihood of becoming trapped in a local minimum, or a plateau on the error surface, also resulting in poor generalization ability. Quite obviously, the best learning rate for good generalization is not so obvious. The selection of the appropriate size of the learning rate has been a huge problem for a number of ANN users. Studies pertaining to this aspect of ANN learning, though few, have presented mixed conclusions.

Wilson and Martinez (2001) conducted experiments on speech digit recognition. They conclude that a learning rate that is too large often hurts generalization accuracy and also slows down training. They further state that, once the learning rate is small enough, further reductions in size would result in a waste of computational resources without any further improvement in generalization ability. Richards (1991) also did extensive studies on the relationship between learning rate and the generalization ability of ANNs in phenome recognition. Their conclusion was that ANNs trained with higher learning rates tend to have a better generalization ability than those trained with lower learning rates. However they state that this assertion holds true only for ANNs trained with a single hidden layer; when the hidden layers are doubled the opposite is quite true. The authors do not offer much explanation for this phenomenon except suggesting that it's an area that needs further insight. Another group of authors, Mohammad and Luis (2007), using different learning rates, tested the performance of ANNs for several time series which had been previously reported to have worse results with ANNs. They conclude that high learning rates are good for less complex data and low learning rates should be used for more complex time series data.

In light of the uncertainty surrounding the selection of the most appropriate learning rate for ideal generalization ability of ANNs, several authors have attempted to offer some sort of heuristics. In practice these heuristics are frequently used as points of departure for subsequent search by trial and error. Plaut et al.(1986) proposed that the learning rate should be inversely proportional to the fan-in of a neuron. This approach has been theoretically justified through an analysis of the Eigen value distribution of the Hessian matrix of the objective function. Reports from Swingler (1996) suggest that starting with a large value for the learning rate of 0.75 and reducing to 0.25 and then to 0.1 as the network starts to oscillate is a good way of reaching the global minimum of error, leading to higher generalization ability of ANNs. Accounts from Becker and Cun (1986) suggest that for a given neuron, the

learning rate should be inversely proportional to the square root of the synaptic weights made to that neuron for an optimal generalization ability of the ANNs to be attained. Gonzalez (2000) reports that one- and two-layered networks with a learning rate of 0.2 and a momentum of 0.3 yield the best combination for good generalization ability. Unfortunately, this study was only based on simulated data. Richards (1991) shows that for fixed parameters, almost optimal generalization requires learning to go to zero at an appropriate rate, however Katidiotis et al. (2000) contend that a more nearly constant learning rate might well be preferable if parameters are time varying.

Traditionally, learning rates remain fixed during training, but some authors have come up with proposals to dynamically adjust the learning rate during training. One proposal comes from Jacobs (1998), who suggests that one should assume that each weight has a different learning rate n_{kj}. The following rule is then applied to each weight before that weight is updated: if the direction in which the error decreases at this weight change is the same as the direction in which it has been decreasing recently, then n_{kj} is increased; if not, n_{kj} is decreased. The direction in which the error decreases is determined by the sign of the partial derivative of the objective function with respect to the weight. An alternative is to use an annealing schedule to gradually reduce a large learning rate to a smaller value. This allows for large initial steps, in the region of the minimum. However, Shavlik et al.(1991) discourage this approach by stating that these methods require the selection of parameters which determine the rate at which the learning rate is to be adjusted. They also state that the optimization of these parameters is highly problem-dependent and "*… may lead to over adjustment of the weights, resulting in dramatic divergence".*

It is evident that the debate on the most appropriate learning rate for an optimal generalization ability of ANNs is largely ongoing. Analysis of the existing studies shows a lack of thorough probing on the part of researchers. For instance there is a widely held perception that the momentum attributed to smoothing bumps in the error surface by referring to the previous weight is closely linked to the learning rate. If this assertion is remotely true, as indeed proven by Attoh-Okine (1999) in predicting pavement conditions, and Tsai and Lee (2011) in a hand gesture recognition system, then any investigation done on the learning rate cannot exclude the momentum term. None of the authors mentioned thus far has dared to fully conduct systematic investigations in that direction. Although some heuristics on selecting the optimum combination of learning rate and momentum are found in literature most of them have not been universally accepted as they are largely based on toy problems with no relevance to the real world.

At present we are not aware of any major investigations on the effects of the learning rate on the generalization ability of ANNs, taking into account the possible influence of a momentum term. Most data used for previous research were simulated, making it difficult to generalize the conclusions of these studies. Furthermore, these studies tend to focus on network convergence without paying much dividend on generalization ability, which is an equally important aspect of ANN learning. The only rigorous studies we are aware of which have come close to addressing this issue are those conducted by Maier and Dandy (1999), who experimented with various learning rates and momentum combinations in forecasting salinity in Rivieree river, Australia. They empirically found that a learning rate of 0.6 and momentum of 0.2 ensured reasonably good convergence and generalization ability of their networks. Richards (1991), underscored a critical observation on how the sensitivity of networks to learning rates largely depends on the architecture of the networks. He found that multi hidden layer networks are more sensitive to the learning rate than single hidden layers. Although he conducted limited investigations of only two learning rates, 0.1 and 0.01. This is an area which, if fully investigated, has potential for significant strides insofar as the generalization ability of ANNs is concerned. Therefore continued research on this crucial aspect of ANN design remains an ardent necessity.

Data and Methods

In our approach for the study we used experimental method which is a proven method for testing and exploring cause and effect relationships. The benefit of using this method is that it allows the control of variables thereby enabling the isolation of a particular variable to observe the effects due to that variable alone. In this case our interest was on the effects of the size of the learning rate on ANN generalization. The software used for the purposes of this study is Matlab Version 7.4.0.287 (R2007a). Matlab is an application software and programming

language with interfaces to Java, C/C++ and FORTRAN. In this study, Matlab provides an environment for creating programs with built-in functions for performance metrics and forecasting using its Neural Networks toolbox Version 5.0.2 (R2007a). The computer used to conduct this study is an Intel(R) Core(TM) 2CPU6300@1.86GHz. The data was collected from the South African Tertiary Institutions Network (TENET) website (www.TENET.ac.za). We analysed network traffic data which comprised inbound traffic in (bits/ sec) from the University of Fort Hare VC Alice Boardroom 101 – Fa 0/1 router. The data spanned from the 1st of March 2010 from 02:00 hours to the 21st of September 2013 02:00 hours in daily intervals, equating to 700 observations. As in all practical applications the data suffered from several deficiencies that needed to be remedied before use for ANN training. Preprocessing was done which included Linear interpolation to fill in missing values, which amounted to 7 such observations. Matlab Neural Network toolbox has a built-in function, *mapminmax* which scales the data down before training so that it has 0 mean and unity standard deviation and then scales it up again after training, so as to produce outputs with 0 mean and unity standard deviation. The data was partitioned into training and testing sets. 547 samples were allocated to the train set whilst 182 were allocated to the test set.

To investigate the effect of the learning rate on the generalization ability of ANNs in predicting TCP/IP network traffic trends, various layered, fully connected ANNs with a single input neuron, Logistic sigmoid activation function in the hidden layers and a Linear output neuron were examined. We carried out investigations on both single and double hidden layer ANNs.

A supervising script was written to compute the ANN inputs and targets. On visual inspection of the time series a sliding time window of size 150 was arbitrarily chosen.

Training was stopped after 1000 epochs and the generalization performance of the ANNs tested by presenting the unknown test set to the ANNs and calculating the Root Mean Squared Error (RMSE) between the actual and predicted values. RMSE is a dimensionless value calculated to compare ANN performance. The RMSE on the test set (MSE_{te}) was calculated using the following equation:

$$RMSE_{te} = \sum_{p=1}^{P_{te}} (d_p - o_p)^2 \quad (1)$$

where d_p is the desired output for each input pattern and o_p is the actual output produced by the ANN. In order to minimize the random effect of the initial weights on results, for each experiment conducted, 4 training runs were made and the results averaged. We also ensured that all other variables that could potentially affect the quality of results remain constant. Hence throughout the duration of our investigations the size of the training set was fixed at 547 samples and of the test set at 182 samples, weights were randomly initialised in the range of [-0.5, 0.5], the BP (trainlm) training rule was the weight updating algorithm and the momentum was fixed at 0.2 (unless stated otherwise).

The Experiments

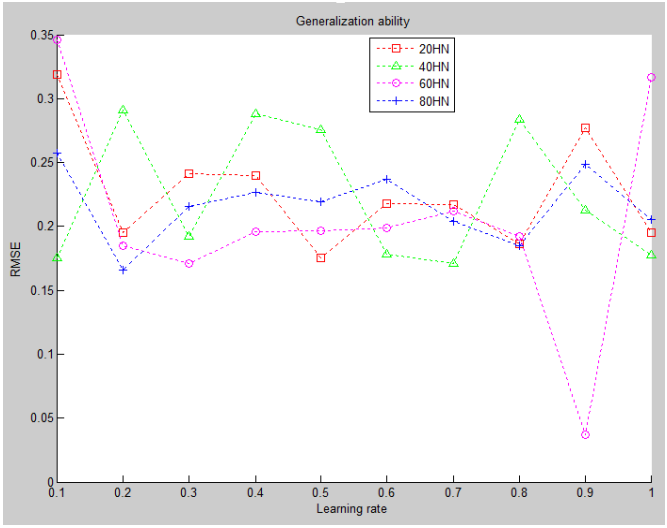


FIGURE 1. GENERALIZATION ERRORS (RMSE) FOR THE DIFFERENT LEARNING RATES AT VARIOUS NETWORK ARCHITECTURES.

The first set of experiments involved several single hidden layer ANNs trained on various learning rates on separate occasions. The ANNs selected for examination had 20, 40, 60 and 80 hidden neurons. These were selected based on stability times during preliminary investigations. The learning rates were varied according to 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1. The choice of the learning rates was purely done on an arbitrary basis. Figure 1 shows the results of those experiments.

In the second set of experiments, we trained a single hidden layer ANN of a network architecture of 60 hidden neurons, at learning rates of 0.2, 0.4, 0.6, and 0.8 for a different number of epochs in order to ascertain how the impact of learning rate on generalization ability varies with an increase in number of training iterations. The choice of the values of the learning rates was motivated by a similar case study by Attoh-Okine (1999) who utilised the same values of the learning rates for his experiments.

The reason a 60 hidden neuron ANN was selected for investigation was because it exhibited better generalization performance than other network architectures during preliminary experiments conducted prior to these investigations.

Figure 2 shows the generalization errors from the experiments.

In the third set of experiments, we assessed the performance of a 2 hidden layer architecture. After conducting several trial runs on various ANN architectures from preliminary experiments a 2 hidden layer ANN of architecture (5, 35) i.e 5 first hidden layer neurons and 35 second hidden layer neurons, was chosen as the bases for conducting these investigations. The bases for arriving at the chosen architecture was the fact that the (5, 35) network architecture exhibited better generalization performance amongst all the 2 hidden layer architectures examined in the preliminary experiments.

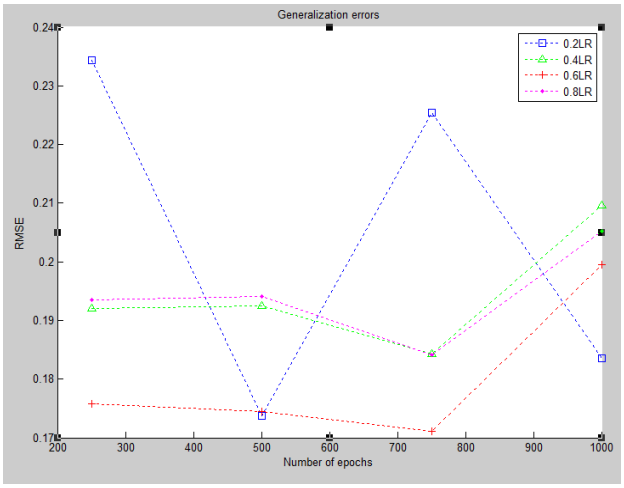


FIGURE 2. GENERALIZATION ERRORS (RMSE) FOR THE DIFFERENT LEARNING RATES AT VARIOUS TRAINING ITERATIONS.

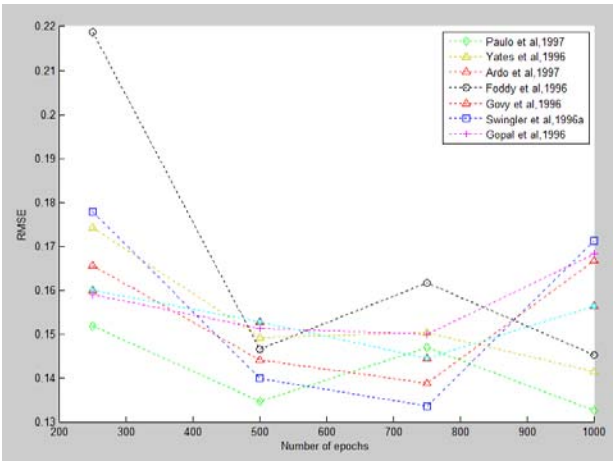


FIGURE 3. GENERALIZATION ERRORS (RMSE) FOR DIFFERENT LEARNING RATES AT VARIOUS TRAINING ITERATIONS FOR (5, 35) ANN.

We trained the ANN on 2 different learning rates of 0.1 and 0.01 on separate runs. The results of the generalization performance are shown in Figure. 3.

Finally, in order to make fair conclusions additional experiments were conducted using different combinations of learning rates and momentum values so as to assess whether the effect of different learning rates is the same regardless of momentum. These 2 parameters have been suggested to be quite closely related in literature. We trained as in the previous case, the same ANN of network architecture (5, 35) using different heuristics of learning rates and momentum combinations provided in literature. The generalization errors are shown in Figure. 4.

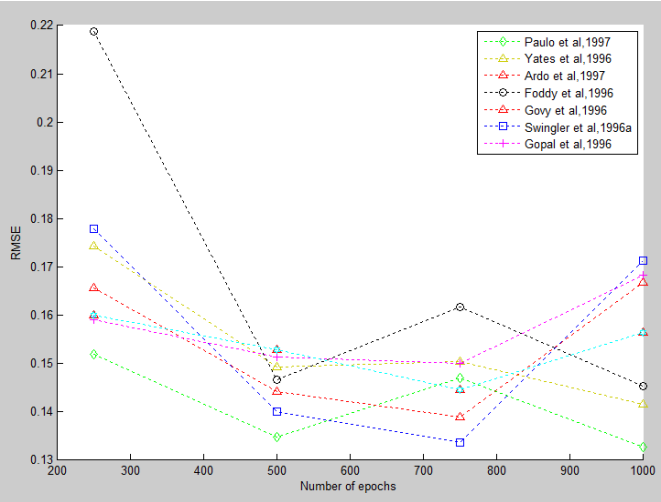


FIGURE 4. GENERALIZATION ERRORS (RMSE) FOR DIFFERENT LEARNING RATES AND MOMENTUM COMBINATIONS AT VARIOUS TRAINING ITERATIONS FOR (5, 35) ANN.

Results and Discussions

In this section we discuss the empirical results on experiments regarding the relationship between ANN generalization and learning rate. We begin our analysis by examining the results in Figure 1, which depict the generalization errors of various single hidden layer ANNs trained on various learning rates. From Fig. 1, it would appear that indeed selecting the optimum learning rate for a given problem is a complex task. From the results, we note that the best performing ANN is an architecture of 60 hidden neurons, trained on a learning rate of 0.9. We note that for many of the ANNs examined the lowest generalization errors occurred at lower learning rates mostly between 0.2 to 0.6, although it is difficult to pinpoint a single universal value which we can safely conclude to be effective in all cases. However, in many cases a learning rate of 0.6 seems to give the most reasonable generalization errors. Our results indicate that the smaller ANNs performed badly in these experiments, especially the 20 hidden neuron and 40 hidden neuron architectures. These two architectures also displayed the most erratic behaviour, whether trained on smaller or larger learning rates, indicating the intrinsic relationship between learning rate and network architecture, emphasizing the sensitivity of smaller ANNs to learning rate. Generally small learning rates produced consistent and better results, whereas large learning rates appeared to cause oscillations and inconsistent results.

We then trained a 60 hidden neuron ANN at different learning rates of 0.2, 0.4, 0.6, and 0.8 for a different number of epochs in order to ascertain how the impact of learning rates on generalization ability varies with an increase in the number of training iterations. The results of that endeavour are shown in Figure 2. From Figure 2 note that the best generalization performance is attained when the ANN is trained on a learning rate of 0.6. It is evident from the results that the generalization ability of the ANN increases as the size of the learning rate increases, however this is only true up to a certain threshold, beyond which any further increment in learning rate results in adverse effects. This is true judging by the generalization errors incurred in Figure 2. From Figure 2, note that the ANN performs the worst and is mostly erratic at a learning rate of 0.2, when the learning rate is increased to 0.4, the generalization ability of the ANN dramatically improves, at a learning rate of 0.6, the ANN is at its best in terms of generalization ability, however at a learning rate of 0.8, instead of following a similar antecedent, the generalization ability of the ANN decreases.

It is also quite interesting to note that for all the (learning rate-number of iterations) combinations the ANN performs the best in terms of generalization ability at 750 epochs. Additionally, training time was fast for learning rates of 0.2 and 0.4, and reasonably fast for rates of 0.6 and 0.8. Note that larger learning rates take many epochs to reach their maximum generalization ability, which is ultimately poor anyway. The small learning rates take longer to reach the same accuracy, but yield no further improvement in accuracy.

We now turn our discussions to the results shown in Figure 3 which show the generalization performance of a 2 hidden layer ANN of network architecture (5, 35). The results in Figure 3 indicate that for the task of forecasting a TCP/IP network traffic time series, a 2 hidden layer ANN is more sensitive to smaller leaning rates than larger ones. The generalization errors show that the ANN has significantly better generalization ability when trained on a learning rate of 0.01 than 0.1, this is more pronounced at an epoch size of 500. Although a learning rate of 0.1 outperforms the 0.01 learning rate between 750 to 800 epochs, we cannot not read much into this as it is largely short-lived

To conclude with the discussions we examine the results of different combinations of learning rates and momentum heuristics given by various authors. The reason for such was to ascertain whether the effect of different learning rates on generalization ability is the same regardless of momentum. The experiments were conducted on a (5, 35) network architecture. The results of this assessment are given in Figure 4. Several conclusions can be deduced from these results regarding the effect of different learning rates and momentum combinations on the generalization ability of the ANNs.

From the results, for the case study considered we note that the 0.1-0.9 (learning rate-momentum) combination given by Foody et al. (1996) fails to produce satisfactory generalization ability. We note extensive oscillatory behaviour in terms of generalization errors at those values of the learning rate and momentum, this as stated by Kavzoglu (1999) could be attributed to the fact that the use of large momentum term increases the effect of oscillations by extending the steps taken in faulty direction or perhaps the ANN could have been stuck in a local minima resulting from the large momentum term.

It can be seen from Figure 4 that the lowest generalization errors are produced by small learning rate and momentum combinations, such as 0.25-0.2 of Swinger (1996) and 0.1-0.3 of Ardö et al. (1997). Another important observation is that the addition of the momentum term to the training considerably slowed down the learning process.

Although the selection of an appropriate combination of the learning rate and momentum is a mammoth task, it appears that a very small learning rate, roughly 0.1 and a moderately high momentum term between 0.2–0.4 provided neo-optimal solutions for forecasting TCP/IP traffic trends.

Now, to answer the question: *How does the use of a momentum term affect the way in which an ANN responds to the learning rate*? From Fig. 4, we see that the generalization errors obtained in those tests where various combinations of learning rate and momentum values were tried out, were in essence not that significantly different from the generalization errors obtained when a momentum value of 0.2 was used as in the previous experiments. However, the behaviour of the ANNs was less controlled with increasing momentum, as a result of the larger steps taken in weight space. In fact, when a momentum value of 0.8 was used in conjunction with a learning rate of 0.2 and when a momentum value of 0.9 was used in conjunction with learning rates of 0.1, the steps taken in weight space were too large and divergent behaviour occurred during training. Discussions on the sizes of training epochs as a function of the network architecture and the resultant effect upon the learning rate and momentum needs further insight so that a more generalized result can be proposed.It is alos important to note that during these experiments,the ANNs did not show any signs of overfitting.

Conclusions and Future Work

The experimental results regarding the relationship between the learning rate and network generalization are discussed in section V. We discovered that for a single hidden layer network, a learning rate of 0.6 gave the most reasonable generalization errors, particularly at an epoch size of 750, and a 2 hidden layer ANN was more sensitive to larger learning rates than smaller ones. We also noted that smaller learning rates decreased the training time

quite significantly.

With regards to the impact the momentum term has on the relationship between the learning rate and generalization ability of ANNs, we discovered that a very small learning rate, roughly of about 0.1 and a moderate momentum between 0.2–0.4 provided neo-optimal solutions for the task considered. However, the behaviour of the ANNs became less controlled with increasing momentum, as a result of the larger steps taken in weight space. We conclude that the degree of effects of learning rate and momentum on the model differ as stated by Wilson and Martinez (2001). The learning rate is more powerful than momentum, as when a large learning rate and small momentum are achieved, the result is more precise than the opposite.

For researchers in this domain seeking simply for the learning rate that produces the fastest convergence should probably settle on a learning rate of 0.4. However, doing so would mean sacrificing generalization ability, which could be more efficiently achieved by using a larger learning rate, therefore trade-off between these two variables is an ardent necessity. Unfortunately for these experiments we did not require the training process to converge, rather, the training process is used to perform a direct search of a model with superior generalization performance.

We also advise, depending on the difficulty of the problem at hand to set the learning rate to 0.1 for the standard Backpropagation algorithm and to either 0.1 or 0.2 if used in conjunction with the momentum term of 0.5 or 0.6, it is important not to set the momentum term too large as it would cause the ANN to be greatly unstable. If possible we advise minimal use of the momentum term as it greatly interferes with the training process of ANNs.

As with almost any area of research, progress leads toward more questions. Based on the research carried out in this study, our results suggest considerable potential for future work. We plan in extending our investigations to new self-similar and chaotic time series and to other ANN models and learning parameters. In addition, more testing is needed to evaluate the applicability of our guidelines to other datasets to be able to make claims about their robustness and to validate the effectiveness of the conclusions reached in this research.

In order to improve and extend the investigations reported in this work, in addition to constant learning rates, the use of adaptive learning rate strategies could be examined and their effects on ANN generalization ability compared to those produced by their counterparts. Another issue which we can possibly look at is using a variable momentum value. A variable momentum value is currently being researched and its impact upon the generalization ability is not exactly known to date.

As this study was limited to Feed-forward ANN learning problems with the Backpropagation learning algorithm, it could be also beneficial to investigate the effects of the size of the learning rate on the performance and generalization ability of other ANN models, including Self Organizing Maps (SOM) and Learning Vector Quantization (LVQ), with the aim of deriving some general conclusions that can be used to construct some guidelines for users in the design of these particular network models.

ACKNOWLEDGMENT

This work is based on the research undertaken within the TELKOM Coe in ICTD supported in part by Telkom SA, Tellabs, SAAB Grintek Technologies, Eastell, Khula Holdings, THRIP, GMRDC and National Research Foundation of South Africa (UID: 86108). The opinions, findings and conclusions or recommendations expressed here are those of the authors and none of the above sponsors accepts liability whatsoever in this regard.

REFERENCES

[1] Ardö, J., Pilesjö, P. & Skidmore, A., 1997. Neural networks, multitemporal Landsat Thematic Mapper data and topographic data to classify forest damages in the Czech Republic. *Canadian Journal of Remote Sensing*, 23(2), pp.217–229.

[2] Attoh-Okine, N.O., 1999. Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance. *Advances in Engineering Software*, 30(4), pp.291–302. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0965997898000714.

[3] Becker, S. & Cun, L., 1986. Improving the Convergence of Back- Propagation Learning with Second Order Methods. In K. Morgan, ed. *Proceedings of the 1988 Connectionist Summer School*. pp. 67–56.

[4] Chen, C.J. & Miikkulainen, R., 2001. Creating Melodies with Evolving Recurrent Neural Networks. *In Proceedings of the 2001 International Joint Conference on Neural Networks*, 2(2), pp.20–60.

[5] Conway, A.J., 1998. Time series , neural networks and the future of the Sun. *New Astronomy Reviews*, 42(June), pp.343–394.

[6] E. Richards, "Generalization in Neural Networks,Experiments in Speech Recognition," University of Colarado, 1991.

[7] Foody, G., Lucas, R. & Curran, M., 1996. Estimation of the areal extent of land cover classes that only occur at a sub-pixel level. *Canadian Journal of Remote Sensing*, 22(4), pp.428–432.

[8] Gonzalez, S., 2000. *Neural Networks for Macroeconomic Forecasting.*, Canada.

[9] H. Tong, C. Li, J. He, and Y. Chen, "Internet Traffic Prediction by W-Boost : Classification and Regression ," *Neural Comput.*, vol. 2, no. 973, pp. 397–402, 2005.

[10] Jacobs, R., 1998. Increased rates of covergence throughlearning rate adaptation. *Neural Networks*, 1(4), pp.295–308. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=938431.

[11] Katidiotis, A., Tsagkaris, K. & Demestichas, P., 2000. *Performance Evaluation of Artificial Neural Network - based Learning Schemes for Cognitive Radio Systems*.

[12] Kavzoglu, T., 1999. Determining Optimum Structure for Artificial Neural Networks. In *In proceddings of the 25th Annual Technical Conference and Exhibition of the Remote Sensing Society*. Cardiff, UK, pp. 675–682.

[13] Mahmoud, O., Anwar, F. & Salami, M.J.E., 2007. LEARNING ALGORITHM EFFECT ON MULTILAYER FEED FORWARD ARTIFICIAL NEURAL NETWORK PERFORMANCE. *Journal of Engineering Science and Technology*, 2(2), pp.188–199.

[14] Maier, H.R. & Dandy, C., 1999. Empirical comparison of various methods for training feed-forward neural networks for salinity forecasting in the River. *Water Resources Journal*, 35(8), pp.2591–2596.

[15] Plaut, D., Nowlan, S. & Hinton, G., 1986. *Experiments on Learning by Back Propagation*.

[16] R. Aamodt, "Using Artificial Neural Networks To Forecast Financial Time Series," Norwegian university of science and technology, 2010.

[17] S. Chabaa, "Identification and Prediction of Internet Traffic Using Artificial Neural Networks," *J. Intell. Learn. Syst. Appl.*, vol. 02, no. 03, pp. 147–155, 2010.

[18] Shavlik, J.W., Mooney, R.J. & Towell, G.G., 1991. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2), pp.111–143. Available at: http://link.springer.com/10.1007/BF00114160.

[19] Swinger, K., 1996. Financial prediction. *Journal of Neural Computing and Applications*, 4(4), pp.192–197.

[20] Swingler, K., 1996. *Applying Neural Networks:A practical Guide*, New York,USA: Harcourt Brace and Company.

[21] Tsai, C.-Y. & Lee, Y.-H., 2011. The parameters effect on performance in ANN for hand gesture recognition system. *Expert Systems with Applications*, 38(7), pp.7980–7983. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0957417410014491 [Accessed July 18, 2014].

[22] Vogl, T. et al., 1988. Accelerating the Convergence of the Back-Propagation Method. *Biological Cybernetics*, Vol 5(9), pp.257–263.

[23] Wilson, D.R. & Martinez, T.R., 2001. The need for small learning rates on large problems. *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, 1, pp.115–119. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=939002

[24] Wilson, D.R. & Martinez, T.R., 2001. The need for small learning rates on large problems. *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, 1, pp.115–119. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=939002.

Vusumuzi Moyo received his B.Sc Honours (Computer science) from the University of Fort Hare and is presently studying towards his Master of Science degree at the same institution. His research interests include artificial intelligence and neural networks.

Further Considerations about Relationship between Framework and Application Components

Reishi Yokomori[1], Harvey Siy[2], Norihiro Yoshida[3], Masami Noro[4], Katsuro Inoue[5]

[1,4] Department of Software Engineering, Nanzan University, Japan

[2] Department of Computer Science, University of Nebraska at Omaha,USA

[3] Graduate School of Information Science, Nagoya University, Japan

[5] Graduate School of Information Science and Technology, Osaka University, Japan

[1]yokomori@nanzan-u.ac.jp; [2]hsiy@mail.unomaha.edu; [3]yoshida@ertl.jp; [4]yoshie@nanzan-u.ac.jp;

[5]inoue@ist.osaka-u.ac.jp

Abstract

A large number of software applications are used over ten years, and are subjected to continuous maintenance activities to improve their operability, functionality, stability and so on. Through such maintenance activities, the internal structure of the software system becomes more complex. We believe that studying how the complexity has evolved is important for understanding the actual maintenance activities. In a previous experiment, we analyzed how code clones and use relations between application and framework components change through a longitudinal study of open source software. However, we performed our analysis over only one set of framework and application, so we would like to discuss how our findings generalize. In this paper, we replicate the previous experiment, targeting several open source projects. By comparing with the result of the past experiment, we will discuss about generalities of our findings. Moreover, we study the differences between the trends for both incoming and outgoing edges, how use relations increase and decrease through long term development, and how code clones are introduced in the software in the early period of the development. These analyses have a certain level of commonality for understanding actual developers' activities.

Keywords

Component; Use Relation; Code Clone; Multi-Version Analysis

Introduction

Commonly used open source software undergo a long period of maintenance activities: fixing a bug, improving and adding functionality, handling new environment or new OS, introducing user's request, and so on. A large body of software are in use for over ten or more years, and the internal structure of such systems becomes more complex with each change. Each single modification affects its complication in small steps, however, "Little things make a big difference". As a result, the structure of such software becomes fragile, so developers understand the importance of refactoring activities and they seize the opportunity to improve the structure of the software while preserving its functionalities.

In the past study, we analyzed the history of one open source development project by using component graphs, and we compared how individual application components of each version use framework components, and conversely, how individual framework components are used by application components (Yokomori 2009). As a result, we confirmed that the number of application components that use a certain framework component increases over time. We also confirmed that the number of framework components used by a certain application component also increases slowly; however, the increase seems to be bounded. Application components that grow to use a lot of framework components are often decomposed into a group of classes in a subsequent refactoring activity. We confirmed such pattern of behavior by analyzing the JARP (application) and JHotDraw (framework) projects. We also analyzed code clones for each version of JARP application, and we introduced the one concerning the

2324-7037/15/01 018-14

doi: 10.12783/ijcsa.2015.0401.03

utilization of JHotDraw framework (Yokomori 2012). However we only identified the presence of various types of clones; we still have to investigate how they were produced and evolved over time. We also understand that analyses of other software systems are indispensable to generalize all these findings.

In this paper, we expand our past experiment to several open source projects, analyzing code clones and use relations between application and framework components for each release version. As the targets, we selected several open source projects that use JHotDraw. We also analyzed several applications that use the Hadoop framework for distributed processing. By comparing with the result of the past experiment, we will discuss general observations on the growth trends of use relations. We also characterize the differences in how the number of framework components used by a certain application component and similarly for the number of application components that uses a certain framework component changes over time. Moreover, we also analyze how outgoing use relations from applications increase. We believe the result yields some common insights for understanding actual developer's activities, and it would be efficient information when we analyze software at a view point of changes of use relations. With respect to clone relation, we extract framework-related code clones from several software projects, and we categorize them into three types of code clones. Through these activities, we study how code clones are introduced in the software in the early period of the development.

In the next section, we introduce background information, such as software components, component graph, relationship between components, and so on. In the section on Experiments, we introduce results of our new experiments. For purpose of comparison, we also show the result of JARP's analysis. In the section on Discussion, we discuss about the results and introduce related works.

Backgrounds

Component and Component Graph

In general, a component is a modular part of a system that encapsulates its content and whose manifestation is replaceable within its environment (Jacobson 1997, Krueger 1992). We model software systems by using a directed or a non-directed graph, called a Component Graph. A node in the graph represents a software component and an edge represents a relation between components. When we focus each component, each component uses other components and is used by other components as a variable, an instance, by method-call and so on, so we can consider use relation as a directed edge. We can also consider clone relation when two components have common or very similar code fragments.

In this study, a node in the graph represents a class or interface, an edge represents a relation between classes. In the case of the use relation, a directed edge from node x to y represents a use relation meaning that class x uses class y for declaration of variables, creation of instances, method calls, reference of fields, and inheritance. In the case of the clone relation, a non-directed edge between node x and y represents a clone relation meaning that class x and class y have similar code fragments that are longer than 25 tokens.

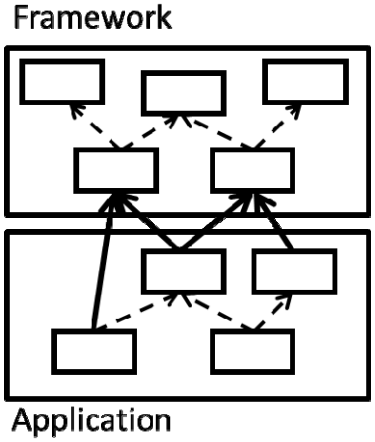


FIGURE 1. USE RELATION FROM APPLICATION TO FRAMEWORK.

Use Relations Spanning between Framework and Application

In our experiments, we focus how use relations spanning framework and application increase. Figure 1 depicts the component graph of an application utilizing a framework. Every solid edge crosses between framework and application, and dashed edges represent use relation inside of the application or the framework components. At first, we count how many framework classes each application class uses, by counting outgoing solid edges from each application class. Next, we count how many application classes gets used by each framework class, by counting incoming solid edges into each framework class. We obtain several versions of software and analyze them separately based on the above two criteria, after that we compare the result of each version.

Clone Relations Related to Utilization of Framework

A software framework is a reusable software platform used to develop applications, products and solutions. A lot of software projects are based on software frameworks, and the utilization of framework facilitates maintenance, shortening the development period.

When we consider code clone related to utilization of framework, we can imagine two kinds of clone relation. One kind crosses between framework and application, as shown in Figure 2. This kind of clone is often produced when application developers copied all of the required code out of a certain framework class. When application developers realize a framework-related feature by using a test code in a framework as a guide for implement, code clone between application class and the framework class that has a test code may be produced.

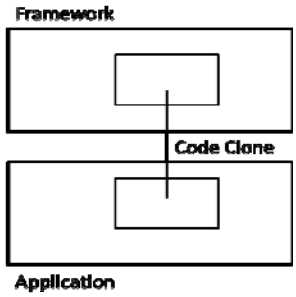


FIGURE 2. CLONE RELATIONS BETWEEN APPLICATION AND FRAMEWORK.

The other is a code clone inside of application components, as shown in Figure 3. In this case, each code fragment has a code that uses a framework feature directly or indirectly. This kind of code clone may be produced when application developers introduce a certain framework's feature under the condition that there is already similar code fragment that uses the feature in application.

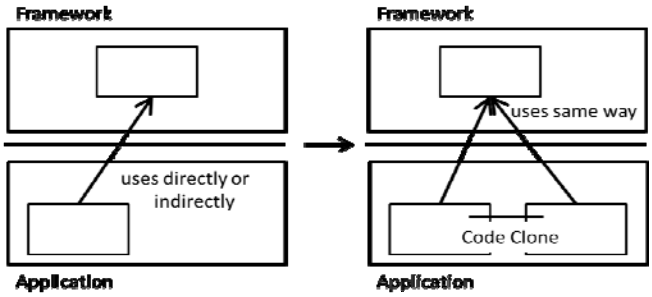


FIGURE 3. A FRAMEWORK-RELATED CLONE RELATION IN APPLICATION.

In our experiments, we focus on these two kinds of clones, and investigate how these are produced.

Previous works

We have investigated how software has evolved through analyses of relationships among software components. At first, we analyzed the history of one open source development project by using component graphs, and for each version, we counted how many outgoing edges each application component has and how many incoming edges each framework component has (Yokomori 2009).

We found that the number of edges increases throughout the application's lifetime and we confirmed that the number of incoming edges of each framework component increases, and the maximum value in each version also increases. We also confirmed that the number of outgoing edges of each application component increases slowly, however, the maximum value in each version is almost the same throughout the development. The number of application components that use a certain framework component increases over time, however, the number of framework components that an application component uses seems to be bounded. Application components that grow to use a lot of framework components are often decomposed into a group of classes in a subsequent refactoring activity. We confirmed such pattern of behavior by analyzing JARP (application) and JHotDraw (framework). We also analyzed code clones for each version of JARP application, and we introduced the one concerning the utilization of JHotDraw framework (Yokomori 2012). However, we studied only the introduction of each clone, and we have to classify them and get a tendency about how developers implant code clones into their application. We also understand that analyses of other software systems are indispensable to generalize such findings.

Experiments

Purpose

For each framework in the study, we prepared several software development projects that use it, and for every release of the application, we analyzed how each application component uses framework components. By comparing changes in the number of outgoing edges of each application component and the one of incoming edges of each framework component, we got a more general understanding of how outgoing and incoming edges between framework and application increase over time. We also analyzed how outgoing use relation increases. By collecting and classifying reasons of increasing, we hoped to prepare basic standard for explaining the result of this kind of use relation analysis. For clone relation, we extracted framework-related clone relations from the same set of software development projects, and classified them into three types of code clones. Through these activities, we confirmed how code clones are introduced in the software early in its lifetime.

Target Projects

In the past experiment, we analyzed JARP that uses JHotDraw as a GUI framework. JHotDraw is often used as a Java GUI framework for technical and structured graphics, so we select and analyze the other software projects that use JHotDraw in this experiment. These are Renew, that is a high-level Petri net editor and simulator, ChemSense, that is a software for sharing, viewing, and editing of a variety of chemistry representations, JStock, that is a software for real-time-monitoring stock markets in the world, and xmlBlaster, that is a MOM (Message oriented Middleware) with a lot of features, respectively. Information of these target projects (including JARP) is shown in Table 1.

TABLE 1. TARGET PROJECTS (JHOTDRAW).

Project	Ver.	Period	Classes	Framework-Application	JHD
Renew	11	1999/03-2012/03	300-> 1903	22-> 1182	5.1
ChemSense	3	2007/04-2008/07	553-> 558	291-> 375	5.2
JStock	70	2007/08-2012/10	582-> 660	127-> 112	7.1
xmlBlaster	40	2005/07-2011/10	1178->1229	47-> 42	5.2
JARP	11	2001/01-2006-07	41-> 215	91-> 325	5.1-5.4

In this table, we show the number of releases analyzed, the corresponding time period analyzed, the number of classes of the first and last versions, the number of edges between framework and application in the first and last versions, and release(s) of JHotDraw that the application used. From this table, we can confirm that Renew and JARP increase several times in size; on the other hand, ChemSense, JStock and xmlBlaster increase a little.

We also select software projects that use a framework of another domain. We will compare these results with the results of the JHotDraw-based applications to confirm differences. We select Hadoop as a second framework. The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing, and a lot of

software projects use it as a framework. As target software projects, we select two projects. The one is Mahout, an application for machine learning and data mining with scalability, and the other is Nutch, an open source web-search software. Information on these 2 projects is shown in Table 2. From this table, we observe that Mahout increases several times in size. Nutch decreases its classes a little, however, use relation between framework and application doubles.

TABLE 2. TARGET PROJECTS (HADOOP).

Project	Ver.	Period	Classes	Framework-Application	Hadoop
Mahout	7	2009/04-2012/05	470-> 2274	775-> 1621	0.18.1 - 0.20.204
Nutch	10	2006/07-2012/07	410-> 373	730-> 1480	0.4 – 1.0.3

Experimental Procedure

For each development project, we collected source code of each release version from its website, and then we analyzed them and constructed a component graph for each version based on use relations and clone relations. We then extracted use relations from application class to framework class and clone relations related to the usages of framework.

In the experiment, the result of each version was compared with the previous or next version's result, and examined what classes are deleted or created, and how the result of classes existing in the both version changes. For extracting use relations, we used Classycle's Analyzer, an analyzer for the static class and package dependencies in Java applications or libraries. We prepared .jar files that include .class files as inputs. Classycle then reads them and extracts information about the usages of other classes and tallies how many classes each class uses and is used by. For extracting clone relations, we used CCFinder, a code clone detection tool with scalability. In some software project like ChemSense, JStock and xmlBlaster, these results were stable and we observed only a few changes about using a framework.

Result 1: How Framework Classes were Used by Application?

First, we show the most used framework classes in the five JHotDraw-based applications and the two Hadoop-based applications. For the five JHotDraw-based applications, Table 3 and Table 4 show the most used framework classes in the first version and the last version, respectively.

For example, Figure is used by only 3 classes in the first version of Renew, however, in the last version 104 classes in Renew use Figure. From these tables, we observe that the features used in application vary from one application to another, however, Figure, Drawing, DrawingView appears consistently on the list, and these classes are fundamental classes that treat the whole of the features.

TABLE 3. FREQUENTLY USED JHOTDRAW COMPONENTS IN APPLICATIONS (IN THE FIRST VERSION).

Renew First-ver.		ChemSense First-ver.		JStock First-ver.		xmlBlaster First-ver.		JARP First-ver.	
Figure	3	Figure	87	Figure	9	Figure	5	DrawingView	8
ColorMap	2	DrawingView	37	DOMInput	8	DrawingView	4	Drawing	6
Attribute Figure	1	Connector	20	DOMOutput	8	Drawing	4	StorableInput	5
Rectangle Figure	1	Locator	20	AttributeKey	6	DrawApplication	2	Figure	5
Child Figure	1	Tool	16	AttributeKeys	5	Figure ChangeEvent	2	Storable Output	4
Abstract Figure	1	Figure Enumeration	15	Drawing	5	TextAreaFigure	1	Tool	4
Drawing Editor	1	Drawing	13	DOMStorable	4	TextAreaTool	1	Drawing Editor	4
Creation Tool	1	Relative Locator	13	Resource BundleUtil	4	TextFigure	1	Connection Figure	3
ArrowTip	1	Drawing Editor	10	Locator	4	TextTool	1	Connector	3
TextFigure	1	Connection Figure	8	DrawingView	3	DrawingEditor	1	Abstract Figure	3

TABLE 4. FREQUENTLY USED JHOTDRAW COMPONENTS IN APPLICATIONS (IN THE LAST VERSION).

Renew Last-var.		ChemSense Last-var.		JStock Last-var.		xmlBlaster Last-var.		JARP Last-var.	
Figure	104	Figure	115	Figure	9	Figure	5	Figure	26
Drawing	95	DrawingView	66	Attribute Key	6	DrawingView	4	FigureAttribute Constant	25
DrawPlugin	57	Connector	21	Attribute Keys	5	Drawing	4	Command	23
Command	50	Locator	20	Drawing	4	Figure ChangeEvent	2	DrawingView	21
Figure Enumeration	46	Drawing	18	Resource BundleUtil	4	Storable	1	DrawingEditor	19
Drawing Editor	42	Tool	18	Locator	4	TextAreaFigure	1	UndoableComm and	17
DrawingView	40	Figure Enumeration	15	DrawingView	3	TextAreaTool	1	Drawing	17
TextFigure	34	Drawing Editor	14	Drawing Editor	3	TextFigure	1	FigureEnumerati on	10
Draw Application	34	Relative Locator	13	GroupFigure	3	TextTool	1	StandardStorage Format	9
Parent Figure	33	Connection Figure	8	Default Drawing	3	DrawingEditor	1	Tool	8

TABLE 5. FREQUENTLY USED HADOOP COMPONENTS IN APPLICATIONS (IN THE FIRST AND LAST VERSION).

Mahout First-var.		Mahout Last-var.		Nutch First-var.		Nutch Last-var.	
JobConf	87	Configuration	216	Configuration	141	Configuration	171
Text	83	Path	157	UTF8	61	Text	154
OutputCollector	47	Text	110	Writable	52	JobConf	83
Configuration	47	Writable	98	Path	52	Writable	82
Path	46	FileSystem	71	FileSystem	46	Path	60
Reporter	44	IntWritable	68	JobConf	43	Reporter	59
MapReduceBase	41	Mapper	66	Writable Comparable	39	OutputCollector	48
FileSystem	49	PathFilter	56	Reporter	31	FileSystem	46
Writable Comparable	26	Reducer	50	OutputCollector	21	Configured	45
Reducer	25	Tool	45	Configurable	20	StringUtils	37

For the two Hadoop-based applications, Table 5 shows the most used framework classes in the first version and the last version. As is the case with JHotDraw, there are some differences between two applications, Configration, Path, and FileSystem appears consistently on the list, and these are also fundamental classes. Moreover, Text, and Writable are used by a lot of classes in the last version, and these become also fundamental classes. On the other hand, JobConf was removed from the Mahout's list of last version, and we can confirm that JobConf was deprecated.

Result 2: Distributions of Incoming and Outgoing Edges

Next, we counted the number of outgoing edges to framework classes for each application class, and the number of incoming edges from application class for each framework class. We examined how the number of edges change over time, and plotted it on the graphs shown in Figure 4 and Figure 5. Figure 4 is for the projects that use JHotDraw and Figure 5 is for the projects that use Hadoop, respectively. We only plot components whose value is large.

At first, we explain the result of the five JHotDraw-based applications based on Figure 4. In the case of JStock and xmlBlaster, modification related to framework is done only once, so the lines on the graph are flat and values have been stable. We can confirm that framework-related features had been stable during our analysis period.

In the case of ChemSense, each application class uses less than 10 framework components ("ChemSense Outgoing"). However, there are a large number of such application classes, so Figure and DrawingView are used by a lot of application classes ("ChemSense Incoming"). We can get only 3 release versions for ChemSense; however, incoming edges of fundamental framework classes seem to increase. So we suspect that the result of

ChemSense would be similar to JARP or Renew if framework-related features are expanded in the later versions.

In the case of Renew, the number of outgoing edges of application classes appears to be increasing until ver. 2.0.1 at which point the values decrease ("Renew Outgoing"). In the decreasing point, we verified that some features in a class were decomposed into several classes. The number of incoming edges from several framework classes increases as a progress of development, and such classes are fundamental framework classes ("Renew Incoming"). Overall trend seems to be no significant change against JARP's one.

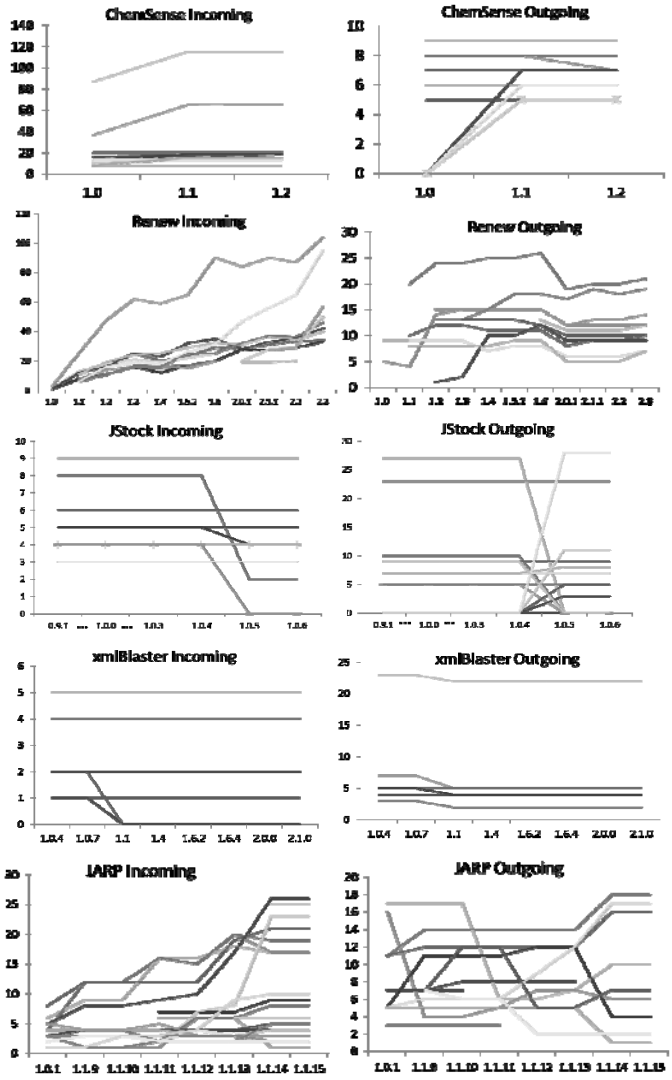


FIGURE 4. TRANSITIONS OF INCOMING AND OUTGOING EDGES (JHOTDRAW).

Next, we will show the result about applications that use Hadoop based on Figure 5. We confirmed that result of Mahout shows similar trends as Renew and JARP. Incoming edges from framework classes Configuration, Text, and Path increase and these classes are used as fundamental ones in the framework. We also confirmed that several large application classes were decomposed ("Mahout Outgoing"). These observations seem to be consistent with the trends from applications that used JHotDraw.

In the case of Nutch, the trends for outgoing edges are stable; however, the trends for incoming edges are somewhat different as the previous ones. We also studied the detail of each update, and framework-related classes are refactored in some updates. For example, an update into ver. 1.1 eliminated about 200 classes (550 -> 364) and use relations from application class to framework class also decreased (1681 -> 1353). An update into ver. 1.4 also includes a restructuring of the usage of framework, so about 400 use relations to framework were eliminated (1766 -> 1365). The graph was affected by the existence of large-scale refactoring ("Nutch Outgoing"); however, Nutch also has a trend of increasing incoming edges from framework ("Nutch Incoming"). Overall, we confirmed that feature addition causes increase of use relation.

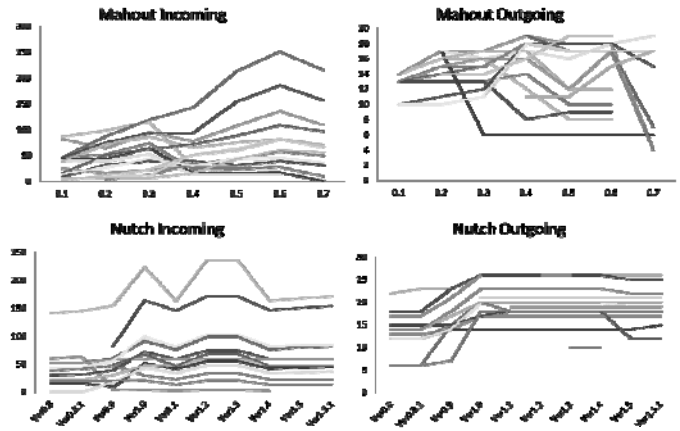


FIGURE 5. TRANSITIONS OF INCOMING AND OUTGOING EDGES (HADOOP).

Result 3: How Incoming Edges Increase?

In the case of incoming edges, increase of use relation is conducive to the increase of incoming edges to existing framework classes, especially about fundamental classes. In the case of outgoing edges, increase of use relation does not lead to the increase of outgoing edges from existing application classes. Next, we selected three JHotDraw-based applications and two Hadoop-based applications, the ones whose utilization of framework changed over time. We counted the number of framework classes that have incoming edges from application and a number of application classes that have outgoing edges to framework. The result is shown in the Table 6. We can confirm that the number of application classes that use framework ("out" rows) increases faster than the number of framework classes used by application ("in" rows). For Renew and Mahout, we also counted the increasing and decreasing of outgoing edges from existing classes ("Existing" rows) and the increasing of outgoing edges from added classes ("Added" rows), in Table 7. We confirmed that the increasing of outgoing edges comes from mainly new classes in an application, and new features are mainly implemented by new classes in an application. Some existing application classes can get new edges; however, other existing classes lose their edges by decomposition of their functionality. From a view point of framework classes, application sometimes uses new features in framework, however, a lot of new classes also uses fundamental classes and features which has already used.

TABLE 6. NUMBER OF FRAMEWORK RELATED CLASSES.

Project		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
ChemSense (JHotDraw)	out	108	148	150								
	in	42	47	68								
	#edges	291	372	375								
Renew (JHotDraw)	out	4	44	72	94	90	108	135	180	221	234	288
	in	19	52	64	64	63	79	82	97	119	124	128
	#edges	22	207	382	477	429	547	673	691	866	907	1182
JARP (JHotDraw)	out	14	14	14	24	28	34	56	73	80	80	80
	in	46	46	46	52	54	50	60	65	78	78	78
	#edges	91	91	91	123	135	143	198	218	325	325	325
Mahout (Hadoop)	out	125	186	246	228	327	401	332				
	in	48	78	93	82	78	79	68				
	#edges	775	1125	1521	1346	1687	2055	1621				
Nutch (Hadoop)	out	206	212	229	335	248	351	351	250	258	261	
	in	58	58	71	79	77	82	82	80	89	89	
	#edges	730	746	924	1681	1353	1766	1762	1367	1392	1400	

TABLE 7. HOW USE RELATIONS INCREASE.

Project		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
Renew (JHotDraw)	Existing		-1	-5	+2	-74	+4	-14	-185	-17	-191	+1
	Added		+186	+180	+93	+26	+113	+141	+203	+192	+233	+274
	Total		+185	+175	+95	48	+117	+127	+18	+175	+43	+275
	#edges	22	207	382	477	429	547	673	691	866	907	1182
Mahout (Hadoop)	Existing		-323	-17	-615	-180	-319	-544				
	Added		+673	+413	+440	+521	+687	+113				
	Total		+350	+396	-175	+341	+368	-431				
	#edges	775	1125	1521	1346	1687	2055	1621				

Result 4: Why Outgoing Edges Decrease?

In the past experiment, we confirmed characteristics of application components decomposed during refactoring activities (Yokomori2009). In earlier versions, such components play multiple roles that it controls and implements an abstract and large feature. In later versions, implemented codes in such components are split into fragments of components. It is very likely that the developer divides a large class into several small classes when a size of core class becomes too big to manage. In such situation, the use of framework is also divided into these small classes. So such component only controls the abstract and large feature, and uses only the essentials. In the latter period, classes for implementation and for handler, and so on use many framework components.

In this experiment, we also confirmed several points whose outgoing edges decreases, so we examined why outgoing edges decreases. We find a case example of decomposition mentioned above, that is **CPNApplication** in Renew ver. 2.0.1. In this update, developers extracted features about menu from the class, and new packages about menu are created and the extracted features are implemented in these packages. As a result, its LOC was reduced to almost quarter of the original (2000 -> 500), and outgoing edges to framework classes are also decreased (26->19).

Other examples are as follows:

- Developers decide to stop using features in framework, and start to develop the feature in the application. This is confirmed in SimpleCompositeFigure in Renew.
- Features implemented in early period had not been used much and become unnecessary, so the features are removed. This is confirmed in AbstractOperator in Jstock, and deleted features are about Input and Output.
- There are application classes that have very similar features in the framework. Developers create a new bridge part that mediate the use of framework, so direct use relations are wrapped up into use relation from the class. This is confirmed when CompositeFigure (JHotDraw) was used in XmlBlaster. We also confirmed in Mahout. At first, BayesDriver and CBayesDriver use the same Hadoop's feature severally, and after then these classes start to use the feature via HadoopUtil.
- Some framework classes provide very similar features. By replacing them into another one, the number of use relation from application to framework decreased. We also confirmed the uses of JobConf (Hadoop) were replaced by the uses of Configration (Hadoop) in Mahout, after Jobconf had become a deprecated class. We also confirmed that Mahout replaced the use of mapper and reducer into another mapper and reducer. Both of them are classes in the Hadoop framework, however, the numbers of their outgoing edges are affected.

In these examples, the number of the outgoing edges of a certain component decreases a lot; however, the scale of such kind of modifications was not so large. We believe that influential modifications to other components are mainly the decompositions of large components.

Result 5: How Code Clones were Produced and Removed?

We analyzed each version of 3 applications (Renew, JStock, and ChemSense) that use JHotDraw, and counted how many types of framework-related code clones through their developments. We also analyse whether these code clones are removed or not in the later versions. We classified them into the following three categories;

1. Code clones that span framework and application.
2. Code clones inside of application components, and they have code fragment that uses a framework feature directly or indirectly.
3. Code clones that span framework and application and the similar code fragments also spread into different application classes.

Table 8 is the result of the categorization, and we confirmed many code clones still exist in the latest version. We also found that only a few clones are removed, and most of the resolved clones disappeared by upgrading framework code, or by modifying one side of the code, not both of the code clones.

TABLE 8. CLASSIFICATIONS OF FRAMEWORK-RELATED CLONES.

	(1) Total	Resolved	(2) Total	Resolved	(3) Total	Resolved
JARP	8	3	13	7	1	1
Renew	10	8	12	3	2	0
JStock	1	1	2	0	0	0
ChemSense	5	0	0	0	1	0

In the early period of the development, several code clones are produced by copying from a class of framework. Copied classes are fundamental classes and utility classes, and it is likely that these classes have examples of using a certain framework features. For example, some utility classes in JHotDraw were entirety copied to ChemSense. Such code clones are not resolved till the introducing feature would be expanded in the application. We also investigated whether code clones between framework and application will spread into the application and found that some code clones do spread into the application, however, this occurs less frequently.

In the later version, code clones appear in only a few methods of several application classes, and this kind of code clone is easy to spread in the application. This is because the structure of the application had already been established, so developers tend to write new code that aligns with the existing application's structure, and new classes are created by inheriting a common parent class. We also consider these code clones are comparatively manageable ones because developers can easily grasp them based on the inheritance relations.

For 2 applications (Mahout, Nutch) that use Hadoop, we studied what kind of framework-related code clones exist in these applications. By introducing these code clones, we identify what kind of code clones are difficult to be solved.

- In the early version of Mahout, several classes have similar descriptions for distributed processing by using JobConf. Some framework classes have such common descriptions, and we confirmed them as code clones. After Jobconf had become a deprecated class, these classes started to use Configration. As a result, code clones between framework and application disappeared; however, code clones between application classes still exist.
- Also in the case of Nutch, several classes have similar descriptions for distributed processing. In the later version, these classes have undergone major re-design and are arranged by its functionality.
- In the early version of Mahout, several algorithms, such as Bayes and CBayes, are implemented separately, so they have a lot of common descriptions. In the later version, these implementations are re-designed and integrated, and then these code clones are resolved.
- There are several combiner and reducer in Mahout, and these combiners (reducers) have several common descriptions.
- From the result, code clones that exist in similar algorithms are relatively easy to solve by combining algorithms. On the other hand, code clones that exist in several descriptions for distributed processing are essentially difficult to solve, because such descriptions have some communality and differences at the same time.

Discussion

Distributions of Incoming and Outgoing Edges

We analyzed distributions of incoming and outgoing edges for several projects, and some projects are consistent with the result of JARP. Maximum value of outgoing edges tends to reach a ceiling during development while maximum value of incoming edges grows unbounded. This kind of ceiling seems to come from the maximum size of the class description. Our analysis result shows that new code is mainly implemented with new components, and not added to the existing components in many cases. On the other hand, we also found that results of XMLBlaster and JStock stay approximately constant. For such projects, the usage of the framework has already

been stabilized, and developer's actions do not affect the usage.

We also get similar results analyzing applications for the Hadoop framework. In the case of Nutch, class structure was revised several times and at a large scale. Actually the number of use relation from application to framework decreases about 400 for each update. Its result was strongly affected by these trimming, however, we can find similar tendency from the result.

Increase of Edges from Application to Framework

We mentioned in the previous section that the number of use relation increases through feature addition. We can interpret the use relation's increase from the transition of incoming edges, however, it is difficult to interpret the use relation's increase from the transition of outgoing edges. From the experiment, we find that the increase of edges is mainly caused by new classes, and the number of classes that have use relations to framework increases significantly.

We consider a situation that developers introduce new features. New features are implemented by using new framework features in some cases, and also by using the already introduced features in another cases. In both cases, fundamental classes are always used heavily. So incoming edges of such fundamental classes grows. We also consider a situation that a large class is decomposed into several small classes. In this case, at first the target class plays multiple roles that it controls and implements an abstract and large feature. After decomposition, individual classes for each feature are created and the target class only controls the ambiguous and large feature, and uses only the essentials. The target class lose incoming edges about the extracted features; however, some classes created by the decomposition get the edges. Each decomposed class also has to use fundamental framework class for management of its own features. Also in this case, the number of application class that need framework features increases, and fundamental framework classes become to have a lot of incoming edges.

Code Clones Related to Framework

In the experiment, we classified framework-related code clones into several categories. We confirmed that some code clones are produced by copying the entire class from framework, and such code clones are produced mainly in the early stage of development. This is because there is no restriction about how developers introduce framework features, and developer's chief priority was to ensure that their products are realizable. When implementing similar algorithms, these algorithms are often implemented independently.

As the development goes underway, developers start to be concerned with management of existing source code. We observed that developers have to write very similar methods in several application classes in the organized class structure, according to their inheritance relationship. The class structure in the application is re-organized in preparation for an implementation of a lot of classes that treat very similar features. After then, such features are implemented along with the new application's structure, and code clones sometimes appear as a result. The respective classes carry out different processing after all, so these code clones are sometimes difficult to be solved. In the early stage of development, it is difficult to decide a method for utilization of framework, so it is likely that this re-organization is important for maintaining long term quality of code.

We expected a situation that there are code clones that span between framework and application, and such code clones also spread into different application classes; however, this occurred less frequently. We believe that this is because developers will re-organize application's class structure before code clones spread, to prevent a large decline in its quality, and that "Rule Of Three" for refactoring is an appropriate concept and is practiced adequately (Fowler1999).

How Do We Utilize The Findings of Evaluations?

From a viewpoint of use relation analysis, our ultimate goal is establishment of the technique that can point out application components which should be modified or carefully inspected, and can roughly estimate the cost of upgrading to a new framework version from several viewpoints.

For example, software components related to management of added components grew through maintenance

activities as reflected by the increase in outgoing edges. Such components often become bloated and developers divide such components into several small classes in the refactoring. Transitions of outgoing edges of each component would be good information for making such decision, and we consider a development of support environment.

For another example, we consider a situation where existing framework classes are deprecated in the new version of framework. Because of this, existing application classes need to be migrated based on recommended upgrade plan. In such case, developer has to specify where the deprecated classes are used, to manage how to carry out the necessary changes, and to specify their sphere of influence. We plan to establish a support tool for these modifications, and then to add the function to estimate those costs.

The analysis results for use relations can also contribute towards an understanding of good framework design. We observe that the applications grew much faster than the frameworks. Furthermore, use relations continued to increase while outgoing edges appear to be bounded. This indicates that even as the applications grew, became more complex and increasingly depended on the framework, the framework APIs remained relatively stable, able to accommodate the increasing application needs. Thus both frameworks we studied appear to have APIs that are designed well.

Code clone analysis can be used to understand how to manage code clones already existing in the applications, through maintenance activities. Even if the structure of application has already organized, certain types of code clones still exist because of the stylized description or very similar description for implementation of similar features. When new related features are added under such environment, existing code clones derives and have a possibility to get more variousness, and management becomes more difficult. By assessing which framework components are used by the existing code clones, we can notify developers that the existing code clones may derive when a certain framework component is used. We consider this kind of support is useful for actual maintenance activities.

Related Works

Previous research on analysis of software repositories have focused on understanding reasons of software changes(German 2003), identifying the effects of communication delays among developers on software development(Herbsleb 2001), detecting potential software changes and incomplete changes (Zimmermann 2004), and so on. Johnson proposed an approach that automatically records developer's activities with the objective of finding a relation between the internal characteristics (size and time, etc.) and the external characteristics (quality and reliability of products, etc.) rather than measuring updates (Johnson 2004). Tamura proposed a new approach to software reliability assessment based on deterministic chaos theory and confirmed its effectiveness by applying it to several development histories of open source software (Tamura 2008). Black investigated fault data extracted from multiple versions of the open source system, by using two of Weiser's original slice-based metrics (Tightness and Overlap) (Black 2009). Bhattacharya represented a way of modelling software structure by using two graphs that represent products and process, and suggested a way of extracting metrics for predicting bug severity, high-maintenance software parts, and failure-prone releases, and so on(Bhattacharya 2012). They also confirmed its effectiveness by applying on 11 open source projects. Our research objective is to obtain new findings through the experiment that examines how clone and use relations become complex as the development goes underway.

From the viewpoint of use relation (component dependency) analysis, there is active research in architecture recovery. Zhang represents OO system by using WDCG (Weighted Directed Class Graph), and suggested a clustering algorithm for recovering high-level software architecture (Zhang 2010). Constantinou represents hierarchical relationships between components as D-layer, by contracting closed paths in component graph, and investigated relationships between architecture layer and design metrics (Constantinou 2011). We consider that our contribution is applying dependency analysis to multi-version analysis.

From the viewpoint of clone analysis, Mondal analyzed the stability of several kinds of cloned codes, and they reported that Type3 clones, known as gapped clones, have higher stability than other clones (Mondal 2012). In our experiment, we also find a lot of such gapped clones from classes in an organized class structure, and their indication also matches our result. Antoniol analyzed the evolution of code duplications in 19 versions of the Linux

kernel (Antoniol 2002). Their target system seems to be relatively stabilized, so the evolution of common ratio is also stabilized through these versions. They also reported that recently-introduced architectures tend to exhibit a slightly higher cloning ratio. From our experiments, code clones produced after the middle period has similar characteristics; however, code clones produced in the early period seems to have another characteristics. We consider that there is significant value to analyse clones produced in the early period.

Conclusions

In this paper, we target several open source projects that use JHotDraw or Hadoop as a framework, and analyze code clone and use relation between application and framework components for each release version. We find that the results for most projects are similar to the result of JARP. Maximum value of outgoing edges tend to reach a ceiling during development, on the other hand, maximum value of incoming edges grows through development. As a different tendency, some results keep approximately constant value. For such projects, the usage of the framework has already been stabilized, and developer doesn't change the usage.

We confirmed the use relation's increase affects the increases of the number of classes that treats framework features, and each such application class uses fundamental framework classes, so some framework classes have a lot of incoming edges. This suffices to explain how distributions of incoming and outgoing edges change as the development goes underway. We examined several points where the number of outgoing edge of the application class decreases. As a result, we find several cases, such as a creation of a new bridge class, stopping a use of framework feature, unification and replacement of the usage, and so on. We can consider that decompositions of large components are performed in the point that outgoing edges decreases a lot. Finally, we confirmed that some code clones are produced by copying entire classes from the framework in the early stage of development. As the development goes underway, the class structure in the application is re-organized, after then such features are implemented along with the new application's structure, and code clones sometimes appear as a result. We consider this re-organization is important for maintaining long term code quality.

Our ultimate goal is establishment of the support tools based on findings from use relation analysis and code clone analysis. We are now implementing a tool that shows transitions of clone and use relation as a table and a graph, and we plan to expand upon it as an analysis infrastructure.

REFERENCES

[1] Antoniol, Villano, Merio, Penta. "Analyzing cloning evolution in the Linux kernel". Information and Software Technology, vol.44, no. 13, pp 755-765, 2002.

[2] Bhattacharya, Iliofotou, Neamtiu, Faloutsos. "Graph-Based Analysis and Prediction for Software Evolution". Proceedings of the 2012 International Conference on Software Engineering, pp 419-429, 2012.

[3] Black, Counsell, Hall, Biwes. "Fault analysis in OSS based on program slicing metrics". Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, pp 3-10, 2009.

[4] Constantinou, Kakarontzas, Stamelos. "Towards Open Source Software System Architecture Recovery Using Design Metrics". Proceedings of the 15th Panhellenic Conference on Informatics, pp 166-170, 2011.

[5] Fowler, Beck, Brant, Opdyke, Roberts. "Refactoring: Improving the Design of Existing Code." Addison Wesley, 1999.

[6] German, Mockus. "Automating the measurement of open source projects". Proceedings of the 3rd Workshop on Open Source Software Engineering, pp 63-67, 2003.

[7] Herbsleb, Mockus, Finholt, Grinter. "An empirical study of global software development: Distance and speed". Proceedings of the 23rd international conference on Software Engineering, pp 81-90, 2001.

[8] Jacobson, Griss, Jonsson. "Software Reuse". Addison Wesley, 1997.

[9] Johnson, Kou, Agustin, Zhang, Kagawa, Yamashita. "Practical automated process and product metric collection and analysis in a classroom setting: lessons learned from Hackystat- UH". Proceedings of the 2004 intl. Symposium on Empirical Software Engineering, pp 136-144, 2004.

[10] Krueger. "Software Reuse". ACM Computing Surveys, vol.24, no. 2, pp 131-183, 1992.

[11] Mondal, Roy, Rahman, Saha, Krinke, Schneider. "Comparative Stability of Cloned and Non-cloned Code: An Empirical Study". Proceedings of the 27th ACM Symposium on Applied Computing, pp 1227-1234, 2012.

[12] Tamura, Yamada. "A Method of Reliability Assessment Based on Deterministic Chaos Theory for an Open Source Software". Proceedings of the Second International Conference on Secure System Integration and Reliability Improvement, pp 60-66, 2008.

[13] Yokomori, Siy, Noro, Inoue."Assessing the Impact of Framework Changes Using Component Ranking". Proceedings of 25th IEEE International Conference on Software Maintenance, pp 189-198, 2009.

[14] Yokomori, Siy, Yoshida, Noro, Inoue. "Evolution of Component Relationships between Framework and Application". Journal of Computers, Computer Society of the Republic of China, vol.23, no. 2, pp 61-79, 2012.

[15] Zhang, Qiu, Tian, Sun. "Object-oriented software architecture recovery using a new hybrid clustering algorithm". Proceedings of the Seventh International Conference on Fuzzy Systems and Knowledge Discovery, pp 2546-2550, 2010.

[16] Zimmermann, Weissgerber, Diehl, Zeller. "Mining version histories to guide software changes". Proceedings of the 26th international conference on Software Engineering, pp 563-572, 2004.

Generating A New S-Box Inspired by Biological DNA

Auday H. Saeed Al-Wattar*[1], Ramlan Mahmod[2], Zuriati Ahmad Zukarnain[3], Nur Izura Udzir[4]

Computer Science and Information Technology, University Putra Malaysia/Affiliation, Universiti Putra Malaysia 43400 UPM SERDANG SELANGOR MALAYSIA

*[1]ahsa.alwattar@gmail.com; [2]ramlan@upm.edu.my; [3]zuriati@upm.edu.my; [4]izura@upm.edu.my

Abstract

Many scholars have attempted to use new methods inspired by DNA bio-techniques in the domains of cryptography and steganography. In this article, a new S-Box was designed inspired by biology DNA techniques to be used for SPN symmetric block ciphers. The new S-Box is used in order to make use of biological process as inspiration in creating the S-Box as simple and secure approach. This article uses the new S-Box within the AES (Advanced Encryption Standard). The National Institute of Standard and Technology (NIST) tests have been used to test the cipher which uses this new S-Box. The results of the tests demonstrate that it effectively passed all the randomness tests of NIST. In addition, the S-Box testing criteria were conducted to test the security of the new S-Box; the results of these tests indicate that the new S-Box has good security.

Keywords

Algorithm; Bblock Cipher; DNA; AES; S-Box; Randomness

Introduction

cryptography has been and is still by far the most efficient means used to achieve secrecy.In the cryptography domain and for any symmetric cryptographic algorithm, the S-Box(substitution Box) is the non-linear unit of symmetric encryption algorithms, that carries out substitution (Kazlauskas & Kazlauskas, 2009). Usually, the cipher uses the S-Box to build the association of the key and the cipher, which is called confusion according to Shannon (Braeken, 2006). Since the security of the whole cipher is dependent on the S-Box, the better the design of the S-Box will result in the most secure cipher as a whole (Adams & Tavares, 1990; Detombe & Tavares, 1993; Leander & Poschmann, 2007). Depending on this concept it can be considered that one of most significant reasons for designing, modifying or working with the cipher S-Box is to enhance the entire cipher and make it totally immune and secure.

There are numerous techniques used by the researchers in designing and modifying the S-Box. as in (Clark, Jacob, & Stepney, 2005), (Tang, Liao, & Chen, 2005), (Tran, Bui, & Duong, 2008), (Canright & Batina, 2008), (Chen, 2008). The employing of DNA as a way of cryptography remains in the preliminary phase. One of the most important reasons lies in the need for a high tech lab in addition to a method that obviates the highly labor intensive means of extrapolation.

However, this challenge has led researchers to find an alternate process in utilizing DNA cryptography, by the use of digital DNA cryptography or pseudo DNA cryptography. This kind of cryptography was inspired by the real DNA process.

A number of previous work have been done within the context of DNA cryptography,(Gehani, LaBean, & Reif, 1999) proposed DNA One-Time Pad, that hides information in DNA strands as a steganography, and(Amin, Saeb, & El-Gindi, 2006) proposed a virtual DNA cryptographic method employed the principal initiatives of the central dogma molecular biology. in addition to (Ning, 2009) which launched a new cryptographic technique that depends on the central dogma of molecular biology. Many other researchers have adopted the proposing of different new cryptographic techniques that are inspired by the techniques of real DNA, such as (Leier, Richter, Banzhaf, & Rauhe, 2000) (G. Cui, Qin, Wang, & Zhang, 2008) (Tornea & Borda, 2009) (Sadeg, Gougache, Mansouri, & Drias, 2010) (Sabry, Hashem, & Nazmy, 2012) (Kartalopoulos, 2005; Singh, Chugh, Dhaka, & Verma, 2010).Although, all

2324-7037/15/01 032-11

doi: 10.12783/ijcsa.2015.0401.04

previous works on DNA have concentrated on proposing a cryptographic method inspired from real DNA, no one has proposed or suggested a cryptographic system (cipher) that depends on the S-Box that has been designed or created using bio-inspired techniques to avoid the algebraic, differential and linear attacks, as well as increase the security of the key cipher by depending on non-classical means for the generating process.

This paper proposes a novel technique for gaining a powerful (8×8) S-Box based on operations that have been inspired from real biological processes, especially some operations of molecular DNA processes and structure. Subsequently, it tests the new S-Box using the S-Box testing criteria and the NIST randomness tests for the cryptosystem that used the DNA-based S-Box.

Cryptosystem

Cryptography system or Cryptosystem is a technique that permits some parties to communicate in secure way. Usually the cryptosystem is composed of a number of items as: plaintext P_t , ciphertext C_t , K is the key, the encryption method E_k ,finally the decryption method D_k .

If E_k is encryption method, and D_k is decryption method the mathematic represent is as follow:

$$\forall\, k \in K\ \exists\ E_k \text{ and } D_k : D_k(E_k(s)) = s\ \forall\ s \in P_t.$$

Substitution Box (S-Box)

S-Box is the main non-linear transformation of an encryption algorithm. It substitutes a set of input bits with a different set of bits known as its output bits. If S-Box denoted by π_s then:

π_s: $\{0,1\}^n \longrightarrow \{0,1\}^n$ where n represent the number of input bits for S-Box.

DNA Background

DNA (Deoxyribo Nucleic Acid). DNA is considered as the genetic drawing of living or existing creatures. All individual body cells have a complete set of DNA. DNA is exceptional for every being. It is a polymer made out of monomers named deoxyribo nucleotides. This nucleotide comprises three fundamental components

A single-strand of DNA is composed of a sequence of molecules named bases, which stick out from a sugar-phosphate backbone, the bases are defined as four characters {A, C, G, and T} (Webster & Tavares, 1986; Zhang, Cheng, & Tarn, 2006). One of the most basic features of the DNA strand sequence is that it is oriented; accordingly, ACCT is distinct from TCCA. Typically the DNA strands exist as paired, reverse complementary words or strands: The Watson-Crick Double helix, with its four letters, A, C, G and T paired via $A^- = T$ and $C^- = G$. Corresponding DNA codes could involve the insertion-deletion metric — with bounded similarity between two strands (Bell & Torney, 1993).The reverse complement process is one of the popoular in DNA strands for example the reverse complement of TTGCAC is equal to GTGCAA. The adjacent reverse complementary strands in reverse directions produces the double strands (D'yachkov et al., 2003) .

Central dogma is one of the most important methods for biological molecules. It is includes some processes of DNA, such as replication, transcription and translation, Figure 1.

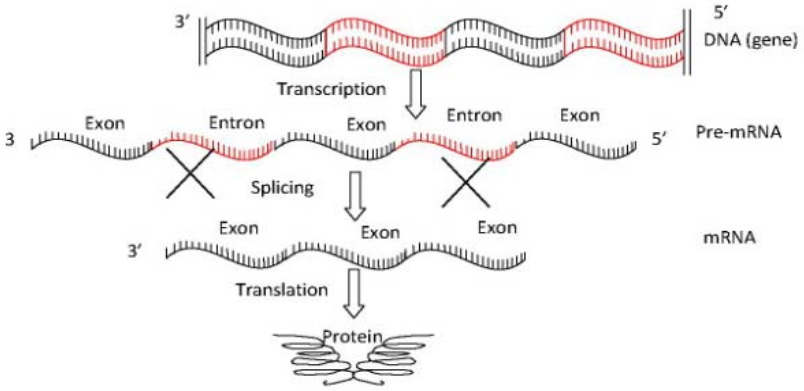


FIGURE 1. CENTRAL DOGMA.

NIST Suite Randomness test

The NIST Test Suite is a statistical test suite for randomness by NIST employed to appraise the ciphers (cryptographic algorithm). It includes 15 tests (Bassham III et al., 2010).

The NIST test suite is one of the security test tools that is used to evaluate the confusion and diffusion properties for the new cipher, as conducted by (Soto & Bassham, 2000) and (Katos, 2005).The test judges whether the production of the algorithms under convincing test conditions shows features that would suggest that the outputs are generated randomly.

S-Box tests Criteria

There are a number of criteria that the S-Boxes should satisfy to be considered as being a good S-Box.

Balanced

If the S-Boxes have the same number of one's and zeroes, it indicates that they are balanced, which is one of most important features of an S-Box.

Completeness

The S-Boxes are complete if every output bit depends on all of the input bits (Webster & Tavares, 1986). The function Y is considered complete if there is at least one pair of plaintext vectors (z and z_i), such that:

(z and z_i) are n bit vectors that vary in just one bit i , and, $Y(z)$and $Y(z_i)$ vary at least in bit h, for all $\{i, h : 1 \leq i, h \leq n\}$.

Avalanche Criterion

A block cipher is considered to reveal the avalanche effect if for a single change in a single bit of the input, the output varies drastically (Feistel, 1973; Feistel, Notz, & Smith, 1975; Webster & Tavares, 1986).

The avalanche value should be within the range [0, 1]. The ideal value for avalanche is 0.5, which indicates that the S-Box satisfies the avalanche criterion. However, it is preferred to take the error interval {- ∈A,+ ∈A} into account for the experimental results (Vergili & Yücel, 2001).

The avalanche of the transformation function (S-Box) can be obtained by using the following equation (Ramanujam & Karuppiah, 2011):

$$\text{Avalanche Effect} = \frac{\text{Number of flipped bits in (output)ciphertext}}{\text{Number of All bits in the (output)ciphertext}}$$

Strict Avalanche (SAC)

According to A. Webster and S. E. Tavares in (Webster & Tavares, 1986) the transformation function (S-Box) satisfies the strict avalanche criterion if each bit of its output bits is changed by a probability of one half when a single bit of its outputs is complemented. This criterion merges both the completeness and avalanche criteria.

Bit Independence (BIC)

Another criterion called Bit Independence (BIT) was declared by A. Webster and S. E. Tavares as one of the criteria used to check the security of the designed S-Boxes.

For all u, v, l ∈ (1,2, ... m), such as v ≠ l, a function f : $\{0,1\}^m \rightarrow \{0,1\}^m$ satisfies the bit independence criterion if complementing input bit u makes the output bits v and l to alter independently.

Commonly the values of BIC range between 0 and 1 as:

1: means the worse state is completely dependent on the relation between v and l bits.

0: means the ideal state is completely independent in the relation between v and l bits

Differential Uniformity

The differential Uniformity δ(S) for a function S(x) is defined as (J. Cui, Huang, Zhong, Chang, & Yang, 2011):

$$\delta(S) = \max_{\substack{\alpha \in F_2^n \\ \beta \in F_2^n \\ \alpha \neq 0}} |\{x | S(x) + S(x+\alpha) = \beta| \quad (1)$$

Where: $S(x) = (s_1(x), \dots, s_n(x))$ is a multiple output Boolean function from $F_2^n \rightarrow F_2^n$. The minimum value for δ(S) = 1, and its value for AES S-Box= 4, the low value of differential uniformity means it is resistant to differential attack (Gong, Tan, & Zhu).

The Proposed Method

According to the DNA base binary coding, every two bit is considered as one DNA base. Since a byte consists of eight bits, so, each byte represents four bases, for example, 01101100 will be as CGTA.

The numbers 0,255 will deal with as DNA bases to treat them as a biological molecule performing a number of processes, inspired from the real biological system. Therefore, those the numbers will be represented as in 1

$$i \in \{A, C, G, T\} \; \forall \; i = \{0, . . ,255\} \quad (2)$$

The following steps stand for creating the proposed S-Box

First:

1) step 1

Split the numbers (0, 255) into 4 sets(w ,x ,y and z), where, every set represents a single DNA strand, and the elements (bytes) of each set represent the DNA bases as each byte has 4 bases. These sets are named a, b, c, and d respectively, each set is consisting of 256 bases, with different start initial value for each set. as follow:

(*w*) With initial value = *i*, (*x*) with initial value = *i+f, (y)* with initial value = *i+f1*, and, *(z)* with initial value =*i+f2*. where *i,f,f1,f2* are random values.

All of their values are increased by *m*, each iteration round as following: (suppose *i=0,f=1,f1=2,f2=3* and *m = 4*)

$$a = j, b = j + 1, c = j + 2, d = j + 3 \; \forall \; j \in \{0,255\} : \; j = j + 4 ,$$

so

$$a = \; (0, 4 \dots,252), b = (1,5,..,253), c = (2,6,..,254)$$

and d = (3,7,..,255).

The DNA bases are represented in binary as:

- Convert the bytes of sets to DNA base code (A, *C*,*G*, and *T*), using the coding :

((*00 =A*), (*01 =C*), (*10 = G*), (*11 = T*)).

- Recode the byte of sets follows: ((A=10), (*C=11*), (*G=01*), (*T=00*))

2) Step 2

Complete the DNA strands structure by Recombine these sets as the following order as some DNA strands direction 3″ to 5″, while the others are from 5″ to 3″: into one set (Strand-All) as:

Strand-All = Forward (w) + Reverse (x) + Forward (y) + Reverse (z).

Forward ( ): The elements stay with same order.

Reverse ( ): Reverse the order of the element within the set.

The process of representing the DNA strands and bases digitally is illustrated in Figure 2.

second:

Performs the reverse complement process inspired by real DNA strands reverse complement,(Watson-Crick

complement), for byte level. The Watson Crick of a DNA strand is gained by converting each A to T, each C to G and vice-versa and toggling the 3′ and 5′ ends of the strand as:

Let B be any byte B: $B = b_4b_3b_2b_1$, where $b_i \in \{A, C, G, T\}$, and $1 \le i \le 4$,also

$\forall b_i \in \{A, C, G, T\}$: $b_i = X_{i2}X_{i1}$, and, $X_{i2}, X_{i1} \in \{0, 1\}$.

Complement of B is $B^c = b_4'b_3'b_2'b_1'$, as ACTG ⇔ TGAC as A⇔T, C⇔G, and $b_i = X_{i2}X_{i1} \Leftrightarrow b_i' = X_{i2}'X_{i1}'$, as 00⇔11, 10⇔01. so, $X_{42}X_{41}X_{32}X_{31}X_{22}X_{21}X_{12}X_{11} \Leftrightarrow X_{42}'X_{41}'X_{32}'X_{31}'X_{22}'X_{21}'X_{12}'X_{11}'$

DNA reverse of B is B^r as $B^r = b_2b_1b_4b_3$, and,

Reverse complement is B^{rc} where $B^{rc} = b_2'b_1'b_4'b_3'$. *So, For B*= $b_4b_3b_2b_1 \rightarrow X_{42}X_{41}X_{32}X_{31}X_{22}X_{21}X_{12}X_{11}$, $B^{rc} = b_2'b_1'b_4'b_3'$. → $X_{22}'X_{21}'X_{12}'X_{11}'X_{42}'X_{41}'X_{32}'X_{31}'$

This action applied for all sets elements as shown in Figure 3.

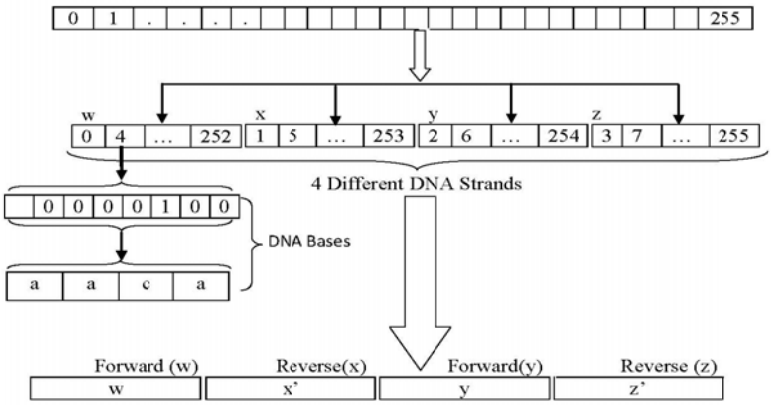


FIGURE 2. SIMULATE THE DNA STRANDS AND BASES STRUCTURE TO BE USED FOR S-BOX DESIGN.

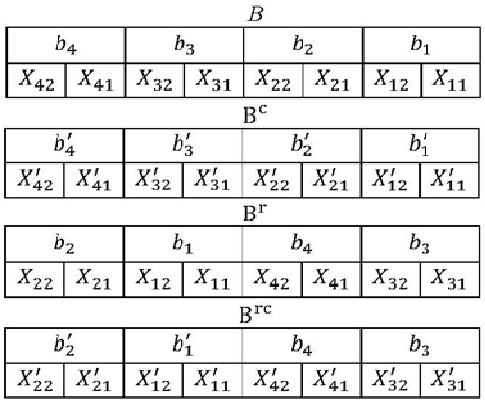


FIGURE 3. REVERSE–COMPLEMENT PROCESS (BYTE –LEVEL).

Third:

To get an extra key for the new static S-Box, an XOR operation between the resulted DNA strands of second stage (B^{rc} of the bytes) and another segment of DNA is perrformed to double and strengthen the randomness of the resulted S-Box. There are many ways to get this DNA strands such as employ a random DNA segment generated by some random generator techniques or use a particular segment obtained from the one GenBank (Benson et al., 2012).

Accordingly, the result of each number will be as follows:

$$Dh = B^{rc} \oplus St_i \quad (3)$$

where St_i represent the segment,

Fourth:

Apply the central dogma process on the result set of numbers Dh_i. The central dogma operation includes a number of important processes: transcription, splicing, and translation besides the reverse transcription. The resulting numbers from the previous steps can be defined as:

$$Dhi = dh_0\, dh_1 \ldots dh_n \; where\; n = 255$$

As the central dogma transcription process includes removing the Introns and keeping the Exon to get the RNA sequence. By considering the set Dh contains both the Exon and the Intron, it could keep the Exon and remove the Intron by separating the resulting set Dh set into two sets called Intron and Exton. In the real DNA strands the number of Exon is more than the number of Exon, so, the separate process would be as:

For i = 1, n // where n is the whole Dh elements
Begin

```
J=0; k=0;
If (j<m)  where m= the length of Exon segment
   Begin    Intron [ j ] = Dh[ i ]; j=j+1 end
        If (k<m/2)
             Begin
                Exon [ k ] = Dh   [i] ; k=k+1
             End
End //
```

Finally, it gets the New DNA S-Box byperform the following two steps:

1) Reverse (Exon[ ])// Reverse the order of the Exon [ ] set
2) Merge (Reverse (Exon[ ]), Intron[]). // Merge the reverse of Exon[] set with the Intron[] set, as:

$$Y[\,] = Merge\,(Reverse\,(Exon[\,]),\ Intron[]).$$

The results of the array Y will be the New-DNA-based S-Box, which is obtained by the inspiration of the DNA processes.

All the above methods have a reverse according to the nature of DNA techniques, so the reverse can be calculated using same operation, but in opposite order and with using the same random DNA segment which used for encryption.

Methodology

Two types of experiments were conducted

1) Experiments measure the security of the AES algorithm that used the New -S-Box by using the statistical NIST Suite Randomness test.
2) Experiments measure the security of the new -S-Box using S-Box test criteria.

The laboratory experiments were performed on the Windows Operating system. For the NIST Suite Randomness test all the data for the 128-byte block of plain-text and 16-byte key was generated and evaluated off-line. The data included a random plaintext with random 128 bit keys.

This works only deals with the situation where the block ciphers run in ECB mode, where the plaintext is divided into blocks, and each block is encrypted separately using the same secret key.

The values of the key were based on generating random data, while the plaintext was different file types, image, video and text. Many images, videos and text files were chosen. According to (Soto & Bassham, 2000), as a minimum, 128 sequences with 1,000,000 bits for each sequence should be used for an NIST test suite. This paper uses 128 sequences of length 1,044,096 bits per sequence in length, which were tested and plotted in the laboratory experiments' action as a random plaintext with random keys of 128 bits.

The tested ciphertext for the experiments is the output of round (3) of the AES algorithm with DNA-based S-Box.

The entire randomness testing relied upon the use of the NIST Statistical Test Suite, which comprises 15 tests that, under special factor, can be observed as 188 statistical tests (Katos, 2005).

The majority of the 15 tests have a one p-value; nevertheless, some of the tests have more than one p-value Table (1). Every p-value matches to the function of a random statistical test on a distinct block, this block is a binary sequence (Rukhin, Soto, Nechvatal, Smid, & Barker, 2001).

The significant level α used for analysis of its value = 0.01, as proposed by NIST, for the study of p-values gained from a variety of statistical tests. Depending on the p-value the following states can be concluded:

- The sequence is shown to be completely non-random if a (p-value = 0).
- The sequence is shown to be non-random if a (p-value <0.01).
- The sequence is shown to be random if a (p-value ≥0.01).

- The sequence is shown to be perfect-random if a (p-value =1).

The proportion of sequences that passed a specific statistical test should lie above the proportion value p′ described in the following equation:

$$p' = \mp 3\sqrt{\frac{\propto (1-\propto)}{m}} \quad (4)$$

Where m = 128.

TABLE 1: BREAKDOWN OF 15 STATISTICAL TESTS APPLIED DURING EXPERIMENTATION

Test ID Number	NIST Statistical Test	Number of p-value
1	Frequency	1
2	Frequency-Within-Block	1
3	Runs	1
4	Longest Runs of Ones	1
5	Binary Matrix Rank	1
6	Discrete Fourier Transformation	1
7	Non-Over Lapping Template Matching	1
8	Overlapping Template Matching Test	1
9	Maurer's Universal Statistical	1
10	Linear Complexity	1
11-12	Serial	2
13	Approximating Entropy	1
14-15	Cumulative Sums (Cusums)	2
16-23	Random Excursions	8
24-41	Random Excursions Variant	18

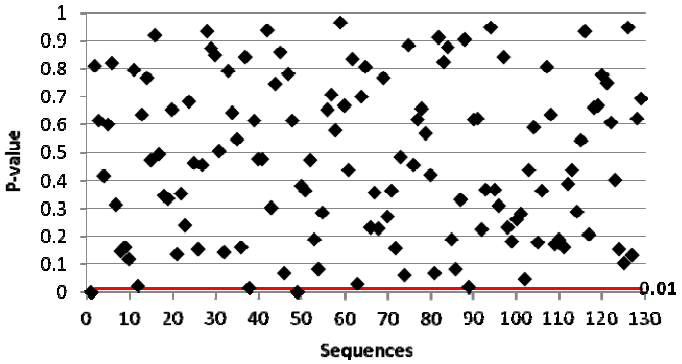


FIGURE 4. P-VALUES OF FREQUENCY TEST AT ROUND 3.

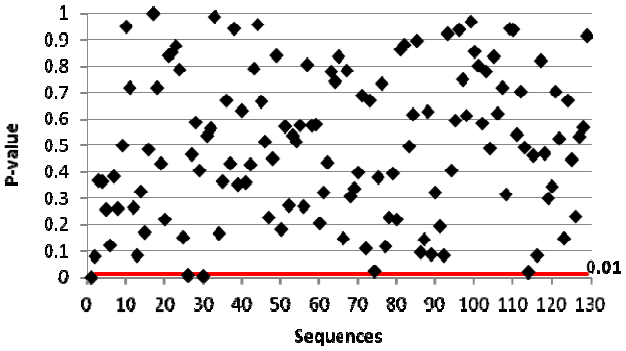


FIGURE 5. P-VALUES OF FREQUENCY WITHIN BLOCK TEST AT ROUND 3.

Results and Discussion

The process of generating the new S-Box used less calculation and mathematical operations than that used in generating the origin AES S-Box.

Randomness Test

This section assesses and analyses the randomness of the AES algorithm used New S-Box. A number of experiments were performed using the NIST Test Suite randomness test function. It is concerned with proving the success of the encryption algorithm that used the proposed S-Box by showing that this cipher has successfully passed all 15 NIST Statistical Test Suite randomness tests for some chosen sequences.

According to (Doganaksoy, Ege, Koçak, & Sulak, 2010), The Frequency Test, Frequency Test Within Block and Run Test are associated to SAC for examine randomness and avalanche effect for ciphertext .

The Frequency Test results of the block cipher for rounds 3 is illustrated in Figure 4. It was reported that 127 out of 128 sequences enrolled a p-value greater than 0.01, for round 3, which indicates that the block cipher passes the

Frequency Test for round 3 with proportion of 0.992.

The Frequency Within Block Test results of DNAB block cipher for rounds 3 is illustrated in Figure 5. It was reported that 126 out of 128 sequences enrolled a p-value greater than 0.01, for round 3, which indicates that the block cipher passes the Frequency Test for round 3 with proportion of 0.984.

The Run Test results of DNAB block cipher for rounds 3 is illustrated in Figure 6. It was reported that 127 out of 128 sequences enrolled a p-value greater than 0.01, for round 3, which indicates that the block cipher passes the Frequency Test for round 3 with proportion of 0.992.

In reference to Table 1 Random Excursion Test is a chain of eight tests and conclusions, single test and conclusion for each of the states: -4,-3,-2,-1 and +1, +2, +3+4. Random Excursions Variant Test is a chain of eighteen tests and conclusions, single test and conclusion for each of the states:-9,-8,…,-1 and +1,+2,… , +9. State +1 from Random Excursions Test (test ID number = 20) and state -1 from Random Excursions Variant Test (test ID number =32) were selected to register in this experiment. Figure 7 shows the p-values for 15 NIST at round 2 of the AES cipher used the proposed S-Box.

From the test results above, the AES algorithm with New -S-Box has good randomness, which is considered as one of the most important measures of the security of the block cipher algorithms. The high p-value of the frequency test is a big indicator of the success of the proposed algorithm, especially when using multiple types of file of different sizes.

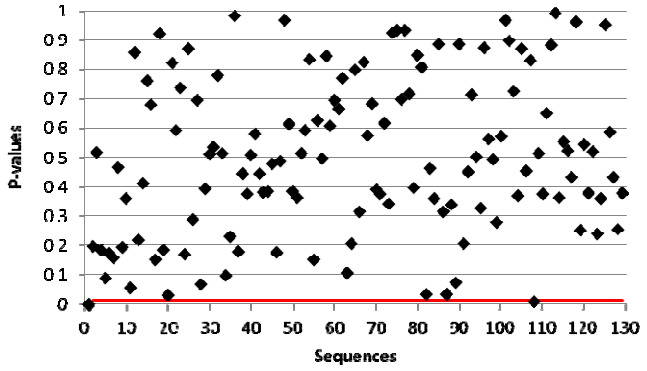
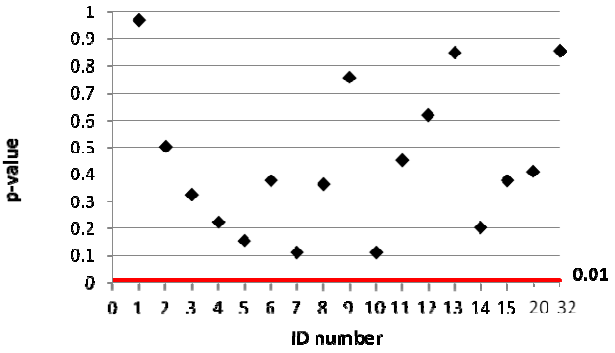


FIGURE 6. P-VALUES OF RUN TEST AT ROUND 3.



FEGURE 7. P-VALUES OF STATISTICAL NIST TEST AT ROUND 3.

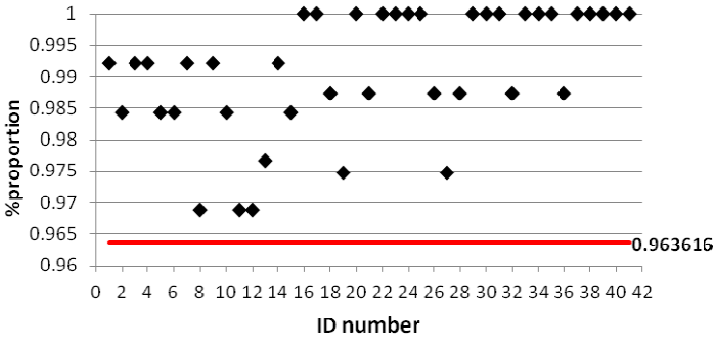


FIGURE 8. RANDOMNESS TESTS RESULTS OF AES BLOCK CIPHER USING THE PROPOSED S-BOX AT ROUND 3.

Proportion

The proportion of the sequences that exceeded a particular statistical test must be greater than the proportion value $p^{'}$. As defined in equation (4), the proportion value of these sequences is:

$$p' = (1 - 0.01) - 3\sqrt{\frac{0.01(1-0.01)}{128}} = 0.96361$$

As m = 128 and. $\propto = 0.01$

Figure 8 demonstrate the randomness test for 15 statistical tests for rounds 3 of AES block cipher that used the

proposed S-Box. From this figure, at the end of the third round, all of the 41 statistical tests fall over 96.36%, which is evident that the output from the algorithm is completely random.

S-Box Test Criterion

Balanced

The new DNA-based S-Box, which is generated using the proposed method, is balanced since it has equal numbers of both 0's and 1's.

Completeness

The new generated New -S-Box has the completeness criteria since each bit of the new S-Box is dependent on all of the input bits. For the DNA-based S-Box, it is clear that if there is at least one pair of 8-bit input vectors, *Z* and *Zi* that are differ in only one bit (*i*), then the output *f(z)* and *f(zi)* are differ at least in bit *j*.

Avalanche

A number of DNA-based S-Boxes were tested to get their avalanche values. The experiments show that the avalanche values of the new S-Boxes range between 0.4694 and 0.51. For example the avalanche value for the new S-Boxis (0.5).

Strict Avalanche (SAC:

The Strict Avalanche criterion for the New -S-Box is 127. The values of strict avalanche for the S-Boxes generated by the proposed algorithms are ranging between 123 and 129.

Bit Independence:

The experiments show that the bit independence value for the new new S-Box does not exceed 0.07. The bit independence value of the New -S-Box= 0.03.

Differential Uniformity

The experiments result shows that differential uniformity of the new S-Boxes is range between (4 and 5). The resistance against differential cryptanalysis is measured by the Differential Uniformity, which indicates that the new S-Boxes generated by the proposed method are resistant against differential attacks(J. Cui, et al., 2011).

The results of the NIST suite randomness test besides the S-Box criteria tests prove that the new DNA based S-Boxes have a strong security since they are bijective, balanced and complete; furthermore, they have perfect avalanche, strict avalanche, good independence and low differential uniformity. Based on all of the foregoing reasons, and supported by the simple processes used in the generation of these S-Boxes it can be deduced that this technique is successful.

Conclusions

This paper proposed a new method to generate new S-Boxes inspired by real biological techniques, specifically DNA. The proposed method tried to take advantage of the DNA properties in generating a new S-Box that satisfies the security criterion with simple mathematical operations. It proves that real biology techniques could be used as the inspiration to build the main components used in the encryption algorithms like the S-Box, since the mechanism of these techniques has different concepts from traditional methods. The generated S-Box can be used within SPN symmetric block cipher as AES cipher. The data used for testing the proposed algorithm were Image, Video, Text, and BBS, which are considered as being among the most difficult and important data types in terms of encryption. The new S-Boxes were tested using the NIST test Suite and the S-Box test criteria. The results of the experiments and tests show that both the new S-Box and the cipher used have strong security. For future work, some modifications could be made for the proposed generating method to build a dynamic DNA-based S-Box, in which the whole cipher completely depends on or is inspired by DNA techniques. Finally, this work opens the door wide to capitalize on the numerous features available in DNA and adopting them within the encryption field.

REFERENCES

[1] Adams, C., & Tavares, S. (1990). *Good S-Boxes are easy to find.* Paper presented at the Advances in Cryptology—CRYPTO'89 Proceedings.

[2] Amin, S. T., Saeb, M., & El-Gindi, S. (2006). *A DNA-based Implementation of YAEA Encryption Algorithm.* Paper presented at the IASTED International Conference on Computational Intelligence, San Francisco.

[3] Bassham III, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., et al. (2010). SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.

[4] Bell, G. I., & Torney, D. C. (1993). Repetitive DNA sequences: some considerations for simple sequence repeats. *Computers & chemistry, 17*(2), 185-190.

[5] Benson, D. A., Cavanaugh, M., Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., et al. (2012). GenBank. *Nucleic acids research*, gks1195.

[6] Braeken, A. (2006). *Cryptographic properties of Boolean functions and S-Boxes.* phd thesis-2006.

[7] Canright, D., & Batina, L. (2008). *A very compact "perfectly masked" S-Box for AES.* Paper presented at the Applied Cryptography and Network Security.

[8] Chen, G. (2008). A novel heuristic method for obtaining< i> S</i>-boxes. *Chaos, Solitons & Fractals, 36*(4), 1028-1036.

[9] Clark, J. A., Jacob, J. L., & Stepney, S. (2005). The design of S-Boxes by simulated annealing. *New Generation Computing, 23*(3), 219-231.

[10] Cui, G., Qin, L., Wang, Y., & Zhang, X. (2008). *An encryption scheme using DNA technology.* Paper presented at the Bio-Inspired Computing: Theories and Applications, 2008. BICTA 2008. 3rd International Conference on.

[11] Cui, J., Huang, L., Zhong, H., Chang, C., & Yang, W. (2011). An improved AES S-Box and its performance analysis. *International Journal of Innovative Computing, Information and Control, 7*(5).

[12] D'yachkov, A. G., Erdös, P. L., Macula, A. J., Rykov, V. V., Torney, D. C., Tung, C.-S., et al. (2003). Exordium for DNA codes. *Journal of Combinatorial Optimization, 7*(4), 369-379.

[13] Detombe, J., & Tavares, S. (1993). *Constructing large cryptographically strong S-Boxes.* Paper presented at the Advances in Cryptology—AUSCRYPT'92.

[14] Doganaksoy, A., Ege, B., Koçak, O., & Sulak, F. (2010). Cryptographic Randomness Testing of Block Ciphers and Hash Functions. *IACR Cryptology ePrint Archive, 2010*, 564.

[15] Feistel, H. (1973). Cryptography and computer privacy. *Scientific american, 228*, 15-23.

[16] Feistel, H., Notz, W. A., & Smith, J. L. (1975). Some cryptographic techniques for machine-to-machine data communications. *Proceedings of the IEEE, 63*(11), 1545-1554.

[17] Gehani, A., LaBean, T., & Reif, J. (1999). *DNA-based cryptography.* Paper presented at the 5th DIMACS workshop on DNA Based Computers, MIT.

[18] Gong, G., Tan, Y., & Zhu, B. Enhanced Criteria on Differential Uniformity and Nonlinearity of Cryptographically Significant Functions.

[19] Kartalopoulos, S. V. (2005). *DNA-inspired cryptographic method in optical communications, authentication and data mimicking.* Paper presented at the Military Communications Conference, 2005. MILCOM 2005. IEEE.

[20] Katos, V. (2005). A randomness test for block ciphers. *Applied mathematics and computation, 162*(1), 29-35.

[21] Kazlauskas, K., & Kazlauskas, J. (2009). Key-dependent S-Box generation in AES block cipher system. *Informatica, 20*(1), 23-34.

[22] Leander, G., & Poschmann, A. (2007). On the Classification of 4 Bit S-Boxes *Arithmetic of Finite Fields* (pp. 159-176): Springer.

[23] Leier, A., Richter, C., Banzhaf, W., & Rauhe, H. (2000). Cryptography with DNA binary strands. *BioSystems, 57*(1), 13-22.

[24] Ning, K. (2009). A pseudo DNA cryptography method. *arXiv preprint arXiv:0903.2693*.

[25] Ramanujam, S., & Karuppiah, M. (2011). Designing an algorithm with high avalanche effect. *IJCSNS, 11*(1), 106.

[26] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., & Barker, E. (2001). *A statistical test suite for random and pseudorandom number generators for cryptographic applications*: DTIC Document.

[27] Sabry, M., Hashem, M., & Nazmy, T. (2012). Three Reversible Data Encoding Algorithms based on DNA and Amino Acids' Structure. *International Journal of Computer Applications, 54*(8).

[28] Sadeg, S., Gougache, M., Mansouri, N., & Drias, H. (2010). *An encryption algorithm inspired from DNA.* Paper presented at the Machine and Web Intelligence (ICMWI), 2010 International Conference on.

[29] Singh, H., Chugh, K., Dhaka, H., & Verma, A. (2010). DNA based Cryptography: an Approach to Secure Mobile Networks. *International Journal of Computer Applications, 1*(19).

[30] Soto, J., & Bassham, L. (2000). *Randomness testing of the advanced encryption standard finalist candidates*: DTIC Document.

[31] Tang, G., Liao, X., & Chen, Y. (2005). A novel method for designing S-Boxes based on chaotic maps. *Chaos, Solitons & Fractals, 23*(2), 413-419.

[32] Tornea, O., & Borda, M. (2009). *DNA Cryptographic Algorithms.* Paper presented at the International Conference on Advancements of Medicine and Health Care through Technology.

[33] Tran, M. T., Bui, D. K., & Duong, A. D. (2008). *Gray S-Box for advanced encryption standard.* Paper presented at the Computational Intelligence and Security, 2008. CIS'08. International Conference on.

[34] Vergili, I., & Yücel, M. (2001). Avalanche and Bit Independence Properties for the Ensembles of Randomly Chosen× S-Boxes. *Turk J Elec Engin, 9*(2), 137-145.

[35] Webster, A., & Tavares, S. E. (1986). *On the design of S-Boxes.* Paper presented at the Advances in Cryptology—CRYPTO'85 Proceedings.

[36] Zhang, M., Cheng, M. X., & Tarn, T.-J. (2006). A mathematical formulation of DNA computation. *NanoBioscience, IEEE Transactions on, 5*(1), 32-40.