ISSN XXXX XXXX © 2017 IJESC

Research Article Volume 7 Issue No.10

An Enhanced Evolutionary Model for Software Defect Prediction

Sukanya.V.S[1], Dr.S.Saraswathy[2]
Research Scholar[1], HOD[2]
Department of CSA & SS
Sri Krishna Arts and Science College, Coimbatore, India

Abstract:
To locate the defects in software and to reduce their occurrence is greatly done with the aid of developing an evolving model for predicting the software defect in an optimal manner. Recognizing how much of these defects are particularly owing to coding errors is a tough difficulty. Defect avoidance is the most brilliant but usually deserted characteristic of software quality guarantee in any project. Software defect prediction is the process of developing models for software projects is helpful for reducing the effort in locating defects. Software defect prediction assists to get better testing resources distribution by detecting defect-prone modules proceeding to testing. Software defect prediction models are developed based on a diverse set of metrics and defect data of earlier version of software. The main objective of this paper is to help developers to identify defects based on existing software metrics using enhanced genetic algorithm technique which improve the software quality. This paper focuses on predicting the software defect contributed by NASA repository dataset. The experimental result is done using matlab and the result shows the most promising output on proposed method in prediction of software defects.

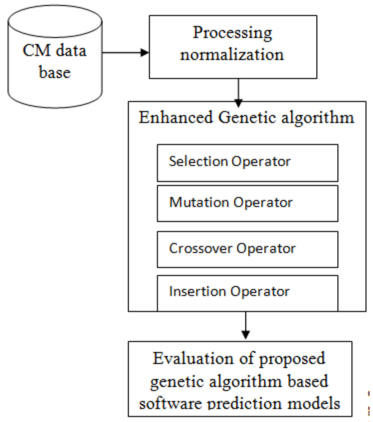
Keywords: Software, defect, prediction, NASA, genetic algorithm

I. INTRODUCTION

Software Defect can be defined as "Imperfections in software development process that would cause software to fail to meet the desired expectations" [11]. Software defects have a major impact of software development life cycle. Software defects are expensive. Moreover, the cost of finding and correcting defects represents one of the most expensive software development activities [15]. The software development team tries to increase the software quality by decreasing the number of defects as much as possible. Software defect prediction helps to optimize testing resources allocation by identifying defect-prone modules prior to testing. This paper develops a evolutionary algorithm based software defect detection model which optimizes its performance with the improvement in global search by adapting the concept of insertion operator with the mutation and cross over operation of the conventional genetic algorithm.

II. RELATED WORK

Software defect prediction is not a new thing in software engineering domain. To come out with the right defect prediction model various related studies and approaches have been conducted. Understanding what defect really means is important so that the term defect is not confused with error, mistake or failure. In the event the defect have taken place, when the software or system fails to perform its desired function [1]. Defect is also observed as the deviation from its specification [2] as well as any imperfection related to software itself and its related work product [3]. Consequently, defect can be referred as its work product and something that is not according to requirement for software. Since, the defects means it is the structure the prediction model for defects, it is used to know how defects are introduced as part of verification and validation (V&V) activities [3]. Defects predicting can be characterized in the proactive process of many types of defects that can be found in software's content, design and codes in producing high quality product [4]. To predict defect density Rayleigh model was also used for different phases of project life cycle [5]. In [6] product and project metrics collected from design review, code testing, code peer review as well as product release usage and defect validation can be constructed using the model to predict defects. Linear regression was applied to these metrics via product metrics only, project metrics only and both. As the result, both product and project metrics provided better correlation between defects and the predictors using linear regression. It demonstrated the feasibility of using regression analysis to build defect prediction model at the same time. To predict defects an approach was carried out using mathematical distributions that serve as quality prediction model [7]. In order to identify and predict the highest defects in the large software systems will prone to more defect is investigated was performed in it. The important factor for the prediction and its impact to the model quality is development information will be the result of the investigation, which focuses on three metrics: number of developers who modified the file during the prior release; the number of new developers who modified the file during the prior release; and the cumulative number of distinct developers who modified the file during all releases through the prior release [8]. We also study to investigate on how to defect fault-proneness in the source code of the open source Web and e-mail suite called Mozilla. To conduct the investigation it used object-oriented metrics proposed by Chidamber and Kemerer [9]. On the other hand, [10] to build defect prediction model was proposed several inputs to simulate the system test phase, in which those inputs could be considered as potential predictors. The defect prediction was based on simple Bayesian Network in a form of Defect Type Model (DTM) that predicts defects based on severity minor, major and minor was the another approach to defects prediction [11]. To come out with defect inflow prediction for large software projects either short-term defect inflow prediction or long-term defect inflow prediction [12] is used by Multivariate linear regression. [13] To predict defect density statistical approach in Six Sigma methodology is applied. In this case, Statistical method was used against the function point as the base metrics to predict defect density before releasing software to production. Defect prediction can also be observed from different perspective which is by predicting remaining total number of defects while the testing activities are still on-going [14], which is called as defect decay model. This model depends on on-going test execution data instead of historical data. [15] Case studies can be presented on building and assisting their organization to assess testing effectiveness and predict the quantity of post release defects and enables quantitative decision about production go-live readiness the defect prediction model was used. Their model was mostly focused on predicting defects in receiving test or manufacture which involves estimate total possible defects based on defined thorough requirements, applying defect elimination efficiency and finally estimates the defects per phase as well as post discharge defects. It display a 1% defect removal efficiency improvement which equals to $20,000 for implementing this model, The defect prediction would be difficult However, if past data is not available. Sample-based defect prediction was proposed to overcome this difficulty by using a small sample of modules to construct cost-effective defect forecast models for large scale systems, in which Co Forest, a semi supervised learning method was applied [16]. For defect prediction testing resources portion could be optimized, [17] on predicting defects of cross-project when chronological data is not in place possibility study must be conducted. The training data is very significant for machine learning based defect prediction provided that the data is carefully selected from the projects was demonstrated as results. Building of defect prediction system, it is necessary to couple with the technique to find its success. In [18] the authors proposed to compute the percent of faults establish in the recognized files as one of the ways to review the efficiency of the prediction Systems. In addition that, the model is said to be a good if it can help in the resource planning in order to maintain the software and insure based on the software system itself is insured [19]. However, it is firm to discover an recognized standard specific for defect prediction. An attempt was taken by given that an all-embracing contrast of well-known bug prediction approaches, jointly with narrative approaches using openly available dataset consisting of numerous software systems [20]. The findings showed that there is still a difficulty with observe to exterior soundness in defect prediction. It necessitate larger mutual data set towards having a noteworthy target of defect prediction Proposed Methodology of Enhanced Genetic Algorithm based Software defect Prediction This proposed method collects the dataset from the PROMISE repository of CM dataset which consist of 498 instances with 22 attributes. The selected raw dataset is preprocessed using normalization approach in which the dataset are converted to the values of the same range. The prediction of defect or no defect for the given instance is determined using the genetic algorithm in this work its performance is enhanced by adapting a new operator known as insertion operator. The genetic algorithm starts with selection process which selects the set of instances of promising result to act as the population set. From it the instances are performed with mutation and crossover to determine the different combination of the values of the given instances and the produced result are justified with the fitness value in each iteration. The instances with highest fitness value are sustained and remaining of them are removed and new set of instances are inserted with the process of insertion operator to overcome the problem of global optimization and thus it predicts the instances falls under the defect or no defect. The overall framework of the proposed method is depicted in the figure 1



III. ENHANCED GENTIC ALGORITHM FOR PREDICTION OF SOFTWRE DEFECT

To improve the performance of classic genetic algorithm this work adopted the concept of restarting procedure for the classic GA which was introduced by Grigorios N. Beligiannis et.al,[22] to achieve a better global exploration of the solution space while executing the minimum possible number of generations (function evaluations) in software defect prediction. In order to achieve this goal, they used the standard global exploration mechanism used by classic GAs (selection, crossover, mutation) but when the GA reaches the local refining phase, we restart the GA so as to preserve the global search procedure. This technique alleviates the enormous computational burden introduced by the local refining procedure, which is quite often useless in finding the optimal solution. The technique is described in Fig. 2. Of course, the new starting of the GA procedure should include all the valuable information gathered from the previous global search. Thus, we propose a new operation called "insertion" to be included in the classic GAs' evolution procedure.

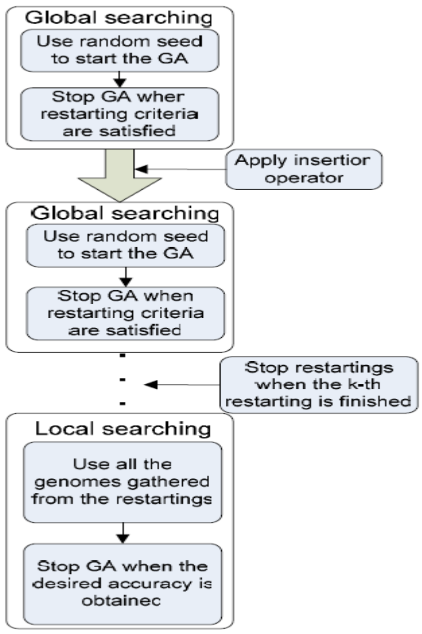


Figure.2. Procedure of the Enhanced Genetic Algorithm

The insertion operator works as follows. It chooses randomly a constant percentage of the genomes of the population of the last generation (before the restarting procedure takes effect) and inserts them into the new initial population of the GA as shown in Fig.3.

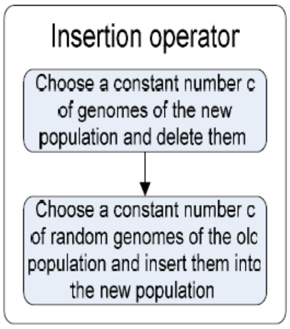


Figure.3. Insertion Operator

In this contribution, three different criteria for deciding when to apply restartings are proposed:

- Fitness function value
- Number of generations
- Mean fitness function value of population

Operator used in Genetic Algorithm Restartings

Crossover operator: Suppose if s1 and s2 are two chromosomes then they are represented as

$S_1 = \{S_{11}, S_{12}, S_{13 \dots\dots} S_{1n}\}$,

$S_2 = \{S_{21}, S_{22}, S_{23 \dots\dots} S_{2n}\}$ are

Two chromosomes, select a random integer number $0 \leq r \leq n$, S_3 and S_4 are offspring of crossover(S_1, S_2),

$S_3 = \{S_i \mid \text{if } i \leq r, S_i \in S_1, \text{else } S_i \in S_2\}$,

$S_4 = \{S_i \mid \text{if } i \leq r, S_i \in S_2, \text{else } S_i \in S_1\}$

Mutation Operator: Suppose a chromosome Si = $\{S_{11}, S_{12}, S_{13} \dots\dots S_{1n}\}$ Select a random integer number $0 \leq r \leq n$, S_3 is a mutation of S_1,

$S3 = \{Si \mid \text{if } i \neq r, S_i \in S_{1i}, \text{else } S_i \in \text{random}(S_{1i})\}$

Selection Operator: Suppose there are m individuals, we select $[\frac{m}{2}]$ individuals but erase the others, the ones we select are having more fitness that means their profits are greater.

Insertion Operator: Suppose there are m individuals, choose a constant number C having genomes of the new population and delete them. At the same time, choose a constant number C of random genomes of the old population and insert them into the new population.

Enhanced Genetic Algorithm:

1. Initialize the population: Producing a number of individuals randomly in software prediction dataset, each instance of dataset is a chromosome which is an n-length array, is the number of parameters.
2. Test if one of the stopping criteria (running time, fitness, generations etc) holds. If yes, stop the genetic procedure.
3. Selection: Select the better chromosomes. It means the profit under these parameters is greater.
4. Applying the genetic operators: such as crossover and mutation to the selected parents to generate an offspring.
5. Recombine the offspring and current population to form a new population with selection operator.
6. Insertion: Choose a 'C' constant number for new population and delete it. Add 'C' constant number of random population to form a new population.
7. Repeat step 2 to 6.

IV. EXPERIMENTAL RESULT

This work used NASA dataset namely CM1 [23, 24]. In order to classify the software module as defect or no defect using enhanced genetic algorithm. This dataset consists of a set of features characterized by static code metrics, such as LOC counts, Halstead and McCabe complexity metrics. These features are characterizing objectively the software quality. The In McCabe metrics are a collection of four software metrics: Essential complexity, cyclomatic complexity, design complexity and Lines of Code.CM1 dataset consists of 498 instances with 22 attributes. In the dataset 5 of the attributes are used for representing different lines of code measure, 3 attributes represents the McCabe metrics, 4 attributes refers to Halstead measures, 8 attributes refers derived Halstead measures, a branch-count attribute, and 1 goal field attribute which is called as class which classifies the instance as presence or absence of defect. Table 1 show the description of each attributes used in the four dataset of this research work.

Table.1. Attribute Description of the four Dataset

S.No	Variables	Description
1	Loc	McCabe's line count of code
2	v(g)	McCabe "cyclomatic complexity"
3	ev(g)	McCabe "essential complexity"
4	iv(g)	McCabe "design complexity"
5	N	Halstead total operators + operands
6	V	Halstead "volume"
7	L	Halstead "program length"
8	D	Halstead "difficulty"
9	I	Halstead "intelligence"
10	E	Halstead "effort"
11	B	Halstead
12	T	Halstead's time estimator
13	lOCode	Halstead's line count
14	lOComment	Halstead's count of lines of comments
15	lOBlank	Halstead's count of blank lines
16	lO Code And Comment	
17	uniq_Op	unique operators
18	uniq_Opnd	unique operands
19	total_Op	total operators
20	total_Opnd	total operands
21	branchCount	% of the flow graph
22	Defects	Yes/No module has/ has not one or more

Evaluation Metrics

His work used NASA dataset namely CM1. In order to classify the software module as defect or no defect using naïve bayes, simple logistics and Zero R classifier are applied and their performance is compared using the following metrics.

Table.2. the confusion matrix for the software defect prediction is depicted in the following table

		3Module actually has defects	
		No (P)	Yes (N)
Classifier predicts no defects	No (P)	a (TP)	b (FP)
Classifier predicts some defects	Yes (N)	C (FN)	d (TN)

TP = true positives: number of examples predicted positive that are actually positive
FP = false positives: number of examples predicted positive that are actually negative
TN = true negatives: number of examples predicted negative that are actually negative
FN = false negatives: number of examples predicted negative that are actually positive

$$\text{Accuracy}=\frac{a+d}{a+b+c+d}=\frac{tp+tn}{tp+tn+fp+fn} \quad (18)$$

$$\text{Probability of Detection} = PD = \text{Recall} = \frac{d}{b+d}=\frac{tp}{tp+fn}$$

(19)

$$\text{Probability of False Alarm} = PF = \frac{c}{a+c} = \frac{fp}{tp+fp} \quad (20)$$

$$\text{Precision} = \frac{d}{c+d}=\frac{tp}{tp+fp} \quad (21)$$

$$\text{F–Measure}=2*\frac{\Pr ecision.recall}{\Pr ecision+recall} \quad (22)$$

$$\textbf{Relative Absolute Error} = \frac{|p_1-a_1|+...+|p_n-a_n|}{|\bar{a}-a_1|+...+|\bar{a}-a_n|} \quad (23)$$

$$\textbf{Root relative squared error} = \frac{(p_1-a_1)^2+...+(p_n-a_n)^2}{(\bar{a}-a_1)^2+...+(\bar{a}-a_n)^2} \quad (24)$$

Where,

- Actual target values: a1 a2 … an
- Predicted target values: p1 p2 … pn
- Mean value of actual target values: $\bar{a}$

Table.3. Performance comparison of Software Defect Prediction techniques for CM1 dataset

	Enhanced Genetic Algorithm	Genetic Algorithm	KNN
Correctly classified	95.10	88.96	75.1606
Incorrectly classified	4.98	11.04	9.8394
Mean absolute error	0.1474	0.1774	0.1789
Root mean squared error	0.1979	0.3076	0.2979
Relative absolute error	99.1988	99.1569	100
Root relative squared error	99.9982	103.27	100
TPR	0.902	0.89	0.87
FP rate	0.902	0.9	0.86
Precision	0.93	0.812	0.80
Recall	0.902	0.89	0.83
F measure	0.855	0.849	0.78

The table 3 shows the performance comparison of the proposed method enhanced genetic algorithm with the conventional genetic algorithm and KNN. It is observed from the result that the correctly classified instances is high with 95.10 using enhanced genetic algorithm based approach in software defect prediction while the conventional genetic algorithm produces 88.9 and the worst case is performed by KNN with the value of 75.16. The highest error rate is produced by knn because of not handling the correct classification of instances in software defect prediction. The proposed method with its improvement in the global optimization it produces best result in software defect prediction with high true positive rate, precision, recall and f-measure.

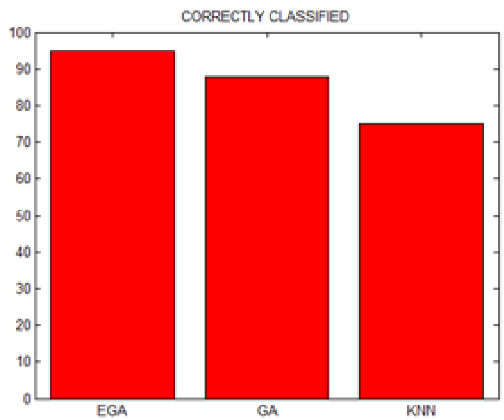


Figure.4. Performance comparison of correctly classified and incorrectly classified instances in software defect prediction

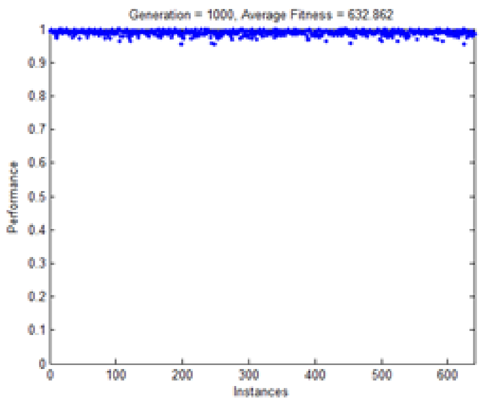
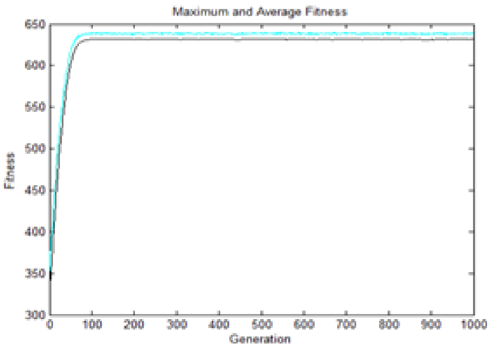


Figure.5. Performance of enhanced Genetic algorithm in each generation



The figure 5 shows the performance of the proposed method during each generation, the process is performed till 1000 iterations it is goal criteria to get the optimized result. The instances which contribute more is determined with the help of the fitness value and the results are obtained. The highest fitness value instances are sustained for the generation of new population and produce the more accuracy in determination of the defect or non defect of the testing samples.

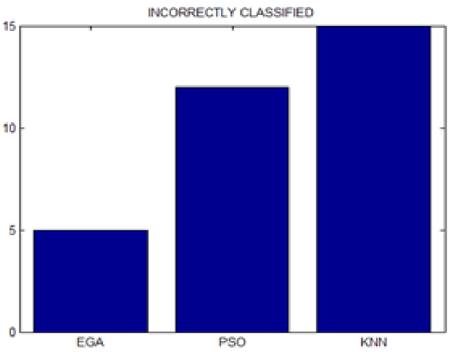


Figure.6. Generation versus fitness value of the instances

In this figure 5 the maximum fitness value obtained during each generation is depicted the instances which hold the highest fitness value alone is selected for the next iteration and remaining of them are eliminated from the process and a insertion operator is used for selecting new set of population to mingle with the existing population for overcoming the problem of global optimization and thus its performance is better compared to the existing conventional genetic algorithm

V. CONCLUSION

A vital role in improving the quality of the software is fulfilled by software defect prediction. The portability of the software can assist in reducing the time taken and the cost of the product. Developing software defect detection model for software projects is helpful for reducing the effort in locating defects. Software defect prediction models are built based supervised learning and unsupervised learning. This work proposed the enhanced genetic algorithm based software prediction model by overcoming the problem of global search in conventional genetic algorithm by introducing the insertion operator. The performance comparison is done with the knn and the genetic algorithm and the result shows the optimal performance of the classification of defect and non defect instances of the software defect detection dataset.

VI. REFERENCES

[1]. Graham, E.V. Veenendaal, I. Evans, R. Black, “Foundations of Software Testing: ISTQB Certification”, Thomson Learning, United Kingdom, 2007.

[2]. N.E. Fenton, M. Neil, “A Critique of Software Defect Prediction Models”, IEEE Transactions on Software Engineering, vol. 25, no.5, pp.675-689, 1999.

[3]. B. Clark, D. Zubrow, “How Good is the Software: A Review of Defect Prediction Techniques”, Carnegie Mellon University, USA, 2001.

[4]. V. Nayak, D. Naidya, “Defect Estimation Strategies”, Patni Computer Systems Limited, Mumbai, 2003.

[5]. M. Thangarajan, B. Biswas, “Software Reliability Prediction Model”, Tata Elxsi Whitepaper, 2002.

[6]. D. Wahyudin, A. Schatten, D. Winkler, A.M. Tjoa, S. Biffl, “Defect Prediction using Combined Product and Project Metrics: A Case Study from the Open Source “Apache” MyFaces Project Family” In Proceedings of Software Engineering and Advanced Applications (SEAA '08), 34th Euromicro Conference, pp. 207-215, 2008.

[7]. I. Sinovcic, L. Hribar, “How to Improve Software Development Process using Mathematical Models for Quality Prediction and Element of Six Sigma Methodology”, In Proceedings of the 33rd International Conventionions 2010 (MIPRO 2010), pp. 388-395, 2010.

[8]. E.J. Weyuker, T.J. Ostrand, R.M. Bell, “Using Developer Information as a Factor for Fault Prediction”, In Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), pp.8, 2007.

[9]. T. Gyimothy, R. Ferenc, I. Siket, “Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction”, IEEE Transactions on Software Engineering, vol. 31, no.10, pp. 897-910, 2005.

[10]. J.S. Collofello, “Simulating the System Test Phase of the Software Development Life Cycle”, In Proceedings of the 2002 Summer Software Computer Simulation Conference, 2002.

[11]. L. RadliRski, “Predicting Defect Type in Software Projects”, Polish Journal of Environmental Studies, vol.18, no. 3B, pp. 311-315, 2009.

[12]. M. Staron, W. Meding, “Defect Inflow Prediction in Large Software Projects”, e-Informatica Software Engineering Journal, vol. 4, no. 1, pp. 1-23, 2010.

[13]. T. Fehlmann, “Defect Density Prediction with Six Sigma”, Presentation in Software Measurement European Forum, 2009.

[15]. S.W. Haider, J.W. Cangussu, K.M.L. Cooper, R. Dantu, “Estimation of Defects Based on Defect Decay Model: ED3M”, IEEE Transactions on Software Engineering, vol. 34, no. 3, pp. 336-356, 2008.

[16]. L. Zawadski, T. Orlova, “Building and Using a Defect Prediction Model”, Presentation in Chicago Software Process Improvement Network, 2012.

[17]. M. Li, H. Zhang, R. Wu, Z.H. Zhou, “Sample-based Software Defect Prediction with Active and Semi-supervised Learning”, Journal of Automated Software Engineering, vol. 19, no. 2, pp. 201-230, 2012.

[18]. Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, “An Investigation on the Feasibility of Cross-Project Defect Prediction”, Journal of Automated Software Engineering, vol. 19, no. 2, pp. 167-199, 2012.

[19]. T.J. Ostrand, E.J. Weyuker, “How to Measure Success of Fault Prediction Models”, In Proceedings of Fourth International Workshop on Software Quality Assurance 2007 (SOQUA ’07), pp. 25-30, 2007.

[20]. L.P. Li, M. Shaw, J. Herbsleb, “Selecting a Defect Prediction Model for Maintenance Resource Planning and Software Insurance”, In Proceedings of 5th Workshop on Economics-Driven Software Engineering Research (EDSER '03), pp. 32-37, 2003.

[21]. M. D’Ambros, M. Lanza, R. Robbes, “Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison, Journal of Empirical Software Engineering, vol. 17, no. 4-5, pp. 531-577, 2012.

[22]. Das, Sufal. Saha, Banani (2009). “Data Quality Mining using Genetic Algorithm”. International Journal of Computer Science and Security, (IJCSS) Volume (3): Issue (2).

[23]. GN Beligiannis, CN Moschopoulos, GP Kaperonis, SD Likothanassis, Applying evolutionary computation to the school timetabling problem: The Greek case Computers & Operations Research 35 (4), 1265-1280

[24]. http://promise.site.uottawa.ca/SERepository

[25]. http://mdp.ivv.nasa.gov