Implementation of Chaotic Gaussian Particle Swarm Optimization for Optimize Learning-to-Rank Software Defect Prediction Model Construction

M A Buchari, S Mardiyanto and B Hendradjaya
School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Bandung, Indonesia
Email : 23514008@std.stei.itb.ac.id

Keywords: software metrics; defect modules ranking; software defect prediction; learning-to-rank; meta-heuristic

Abstract. Finding the existence of software defect as early as possible is the purpose of research about software defect prediction. Software defect prediction activity is required to not only state the existence of defects, but also to be able to give a list of priorities which modules require a more intensive test. Therefore, the allocation of test resources can be managed efficiently. Learning to rank is one of the approach that can provide defect module ranking data for the purposes of software testing. In this study, we propose a meta-heuristic chaotic Gaussian particle swarm optimization to improve the accuracy of learning to rank software defect prediction approach. We have used 11 public benchmark data sets as experimental data. Our overall results has demonstrated that the prediction models construct using Chaotic Gaussian Particle Swarm Optimization gets better accuracy on 5 data sets, ties in 5 data sets and gets worse in 1 data sets. Thus, we conclude that the application of Chaotic Gaussian Particle Swarm Optimization in Learning-to-Rank approach can improve the accuracy of the defect module ranking in data sets that have high-dimensional features.

1. Introduction
Software defect prediction helps us in identifying defect prone software modules. This process is really helpful for software testing activity [1]–[3]. There are 3 different techniques on software defect prediction, that are classification purposes, defect number prediction and construct the defect modules ranking.

Defect ranking prediction and defect classification are the most used technique in the software defect domain. Defect modules ranking is important for the software testing phase, as they can be used to ensure that test resources are allocated efficiently [4]. There are two approaches to construct a ranking defect prediction. The first one is predicting the module defect number, then order it and the second one is Learning-to-Rank approach that directly optimizes the ranking performance [5].

Learning-to-Rank approach uses simple linear model as a prediction model with software metrics [6] as a linear model variable and α as the corresponding parameter for software metrics. The value for
α is obtained by training the software module’s data with known defect number. Learning-to-Rank approach utilizes Composite Differential Evolution (CoDE) to find the optimal value for α [5]. If α have already been trained, the model is ready to use for predicting the defect modules ranking.

Because α is a corresponding parameter for software metrics in learning-to-Rank linear model, the number of α increases as the number of usages of software metrics in the development of predictive models. CoDE is a meta-heuristic proposed by Wang et al [7], in their research CoDE was evaluated using 25 test function and each function run at 30 dimensional search space. The software defect prediction data sets used nowadays has more than 190 software metrics [3], [8].

Chaotic Gaussian Particle Swarm Optimization is meta-heuristic proposed by Jia et al [9] has good test results in solution search space with dimensions of 100 to 500. We propose Chaotic Gaussian Particle Swarm Optimization as an α determinant for the Learning-to-Rank approach by considering that improving the performance of prediction model construction can increase the prediction accuracy.

The remaining of this paper is organized as follows. In Section 2, we mention research that relate to our research. Section 3 describes the implementation of Chaotic Gaussian Particle Swarm Optimization in the Learning to Rank approach. We report and discuss our empirical evaluation in Section 4. We conclude our research in the last section.

2. Literature study

2.1. Meta-heuristic in software defect prediction
Can et al [10] constructed software defect prediction model using SVM and utilize Particle Swarm Optimization for determine some parameter value in SVM. Their research showed better accuracy for classification purpose. Coelho et al [11] applied Particle Swarm Optimization for both ensemble constructor and consensus function for ensemble clustering technique. This technique showed that it could predict defects as good as or better than the other algorithms. Wahono and Suryana [12] applied Particle Swarm Optimization and Bagging for noise feature and data imbalance problem. Their result showed that the classification accuracy had significantly improvement in almost classification method after they implement PSO and Bagging for noise attribute and data imbalance problem.

2.2. Ranking approach software defect prediction
Li et al [13] arranged top 4 subsystem that suspected to have most defect ratio for testing priority purposes. Zimmermann et al [8] used linear regression to predict defect number for further analysis correlation post and pre-release defect modules ranking. They used a combination of complexity metrics in constructing prediction models. The result showed that the complexity metrics were good indicators for software defect prediction. Li et al [14] classified the data to 3 classifications, defect prone modules, non-defect prone modules and deferment modules. They performed ranking method for each class separately. The result showed that the proposed method gave a good ranking prediction. Yang et al [5] showed to us that directly optimize the ranking performance can give good defect modules ranking. They used simple linear model with meta-heuristic algorithm to optimize the α for the model.

2.3. Chaotic Gaussian Particle Swarm Optimization
Particle Swarm Optimization was proposed by Kennedy and Eberhart in 1995 [15]. PSO had simple and efficient characteristics when looking for solutions in the search space. However, Particle Swarm Optimization often experienced premature convergence when searching for solutions in high dimensional search space. Therefore Jia et al [9] performed a hybrid algorithm between Particle Swarm Optimization, Chaotic and Gaussian Local Search.
3. Methodology and implementation

In this section, we discuss the implementation of Learning-to-Rank approach with Chaotic Gaussian Particle Swarm Optimization. In this research we use Chaotic Gaussian Particle Swarm Optimization based on research conduct in [9].

3.1. Data sets

In this research, we have used 2 public benchmark data sets. The first data sets is 3 version of ECLIPSE data sets [8] and the second data sets is 5 open-source project AEEEM data sets [3]. The characteristics of these data sets are shown in Table 1.

The first data sets is public benchmark data that was submitted by D’Ambros et al consisted of 5 software projects Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Mylyn and Apache Lucene. In each software project D’Ambros extracted 6 categories of software metrics: process metrics, previous defect metrics, source code metrics, entropy of change metrics, metric source code and entropy (metric source code). This data can be used for defect prediction at the software class level.

The second data sets is ECLIPSE software project that was proposed by Zimmermann et al consisted of various source code metrics and divided into 3 versions of release i.e. version 2.0, 2.1 and 3.0. This data can be used to predict defects-prone at the file and software package level.

![Table 1. Experimental data](image)

| Data Sets          | Number of Module | Number of Software Metric | Defect Module Percentage | Number of Defect |
|--------------------|------------------|---------------------------|-------------------------|------------------|
| Apache Lucene      | 691              | 212                       | 9.30%                   | 97               |
| Eclipse JDT Core   | 997              | 212                       | 20.70%                  | 374              |
| Equinox Framework  | 324              | 212                       | 39.80%                  | 244              |
| Eclipse PDE UI     | 1497             | 212                       | 14%                     | 341              |
| Mylyn              | 1862             | 212                       | 13.20%                  | 340              |
| Eclipse/File2.0    | 6729             | 198                       | 14.5%                   | 1692             |
| Eclipse/File2.1    | 7888             | 198                       | 10.8%                   | 1182             |
| Eclipse/File3.0    | 10593            | 198                       | 14.8%                   | 2679             |
| Eclipse/Package2.0 | 377              | 207                       | 50.4%                   | 917              |
| Eclipse/Package2.1 | 434              | 207                       | 44.7%                   | 662              |
| Eclipse/Package2.0 | 661              | 207                       | 47.4%                   | 1534             |

3.2. Chaotic Gaussian Particle Swarm Optimization Implementation for Learning-to-Rank Approach

Learning-to-Rank [5] approach calculates relative defect number of a module that denote as f(x) use the following equation:

\[ f(x) = \sum_{i=1}^{d} \alpha_i x_i \]  

(1)

\( x_i \) is the i-th metric from software metrics vector that represent module \( x = (x_1, x_2, \ldots, x_d) \), when \( d \) is total software metrics and represent for the dimension of solution search space on Chaotic Gaussian PSO that is utilized to determine the value of vector \( \alpha \). For Chaotic Gaussian PSO fitness function, we have use Fault Percentile Average (FPA) [16] as suggested in [5]. FPA is also used for evaluating ranking prediction results. The Fault Percentile is defined as below

\[ \frac{1}{k} \sum_{i=1}^{k} \frac{1}{n} \sum_{l=k-m+1}^{k} n_l \]  

(2)

where \( n_i \) is actual defect number in module i, \( k \) is k-th module from top i actual defect modules ranking and \( m \) is top m relative defect modules ranking. Vector \( \alpha \) is translated into the position of each particle or usually called solution candidate in the swarm in order to determine the optimal value of
vector $\alpha$. Then Chaotic Gaussian PSO [9] begins the optimizing process for vector $\alpha$ using the parameter setting from table 2 and FPA as the fitness function.

3.3. Implementation scenario

Our experiment scenario for Learning-to-Rank software defect prediction approach is based on [5]. The implementation of Learning-to-Rank software defect prediction is divided into 2 scenarios. This division is based on different treatments on the data that will be used for construct prediction model and data used to predict the ranking of modules. Five data sets are single-release software project data (AEEEM data sets) and 6 other data sets are multiple-release software project data (Eclipse V2.0, 2.1, 3.0 in file and package level). The detail scenario is shown in figure 1 and 2.

| Chaotic Gaussian PSO Parameter | Value of Parameter |
|-------------------------------|--------------------|
| $W$                           | 0.7289             |
| $c_1$                         | 1.49618            |
| $c_2$                         | 1.49618            |
| population size               | 60                 |
| Particle initial position upper boundary$^a$ | 20                  |
| Particle initial position lower boundary$^a$ | -20                |
| Maximum Fes                   | Number of dimension $\times$ |
|                               | 5000               |

$^a$ only used in the initialization phase
The first step taken on the implementation of Learning-to-Rank is to normalize the data sets. It then transforms all values in each column to the range zero to one. After normalizing the data, we divide each AEEEM project data set based on 75% - 25% ratio. 75% of the data is used to train prediction models, the remaining 25% is used to evaluate the prediction model as test data. We then perform the splitting, because the 5 data sets from the AEEEM data set had only 1 version of each data set.

The next step is to train prediction models using Chaotic Gaussian PSO. With the vector α that is produced by Chaotic Gaussian PSO, model is ready to be used to predict modules rank. We use the prediction model that constructs from former release data sets to predict next release data sets such as a prediction model construct from Eclipse_File2.0 data sets will predict the defect modules ranking for Eclipse_File2.1 and Eclipse3.0 data sets. For single release data sets we predict the defect modules ranking of data test with the prediction model construct by corresponding training data sets.

| Data Sets               | CGPSO (Train) | CoDE (Train) | CGPSO (predict) | CoDE (predict) | Ideal   |
|------------------------|---------------|--------------|-----------------|----------------|---------|
| Eclipse_File2.0-2.1    | 0.8470±.001   | 0.8362±.000(0.00) | 0.7568±.004     | 0.7441±.004(0.00) | 0.9586  |
| Eclipse_File2.0-3.0    | 0.7867±.001   | 0.7801±.000(0.00) | 0.7452±.008     | 0.7490±.013(7.3)  | 0.9506  |
| Eclipse_File2.1-3.0    | 0.8032±.002   | 0.7872±.001(0.00) | 0.7763±.005     | 0.7706±.003(0.02) | 0.8870  |
| Eclipse_Package2.0-2.1 | 0.8223±.001   | 0.8080±.001(0.00) | 0.7983±.007     | 0.7825±.009(91)   | 0.8944  |
| Eclipse_Package2.0-3.0 | 0.8123±.001   | 0.7962±.001(0.00) | 0.7903±.005     | 0.7884±.009(47)   | 0.8944  |
| Eclipse_Package2.1-3.0 | 0.8123±.001   | 0.7962±.001(0.00) | 0.7903±.005     | 0.7884±.009(47)   | 0.8944  |

Table 3. Experiment result for multiple release data sets
### Table 4. Experiment result for single release data sets

| Data Sets      | CGPSO (train) | CoDE (train) | CGPSO (predict) | CoDE (predict) | Ideal  |
|----------------|---------------|--------------|-----------------|----------------|--------|
| Eclipse        | 0.8776±.001   | 0.8667±.000  | 0.8396±.004     | 0.8460±.005    | 0.9323 |
| Equinox        | 0.8299±.002   | 0.8204±.001  | 0.7925±.008     | 0.7779±.019    | 0.8620 |
| Apache Lucene  | 0.9145±.002   | 0.8948±.001  | 0.8203±.029     | 0.8455±.019    | 0.9672 |
| Mylyn          | 0.8525±.002   | 0.8300±.001  | 0.7798±.010     | 0.7611±.009    | 0.9451 |
| Eclipse PDE UI | 0.8362±.002   | 0.8160±.001  | 0.8164±.011     | 0.8179±.016    | 0.9453 |

### 4. Result and discussion

As we mentioned earlier, we have used FPA value for evaluating both training performance and prediction accuracy. We have performed Wilcoxon rank-sum test at 0.05 significance to determine whether models produce by two meta-heuristic has statistically different prediction accuracy. Wilcoxon rank-sum test has been performed for both training phase FPA and prediction accuracy (FPA value as prediction result). Experimental results for multiple-release data sets are shown in table 3.

Eclipse_File2.0-2.1 in table 3 means the prediction model is trained using Eclipse_File2.0 data sets and then the prediction model is used to predict defect modules ranking for Eclipse_File2.1 data sets and so on. We have trained 10 prediction models for each data set. Each value in the table column represents the average of 10 FPA value from training process or average of 10 predicted result along with its deviation standard, while the number in the bracket is p-value of nonparametric test results of Wilcoxon rank-sum.

Table 3 shows that the prediction model trained using Chaotic Gaussian Particle Swarm Optimization has significantly better accuracy on 3 data sets and has an equivalent accuracy in 3 other data sets. This proves that Chaotic Gaussian Particle Swarm Optimization has a better ability than CoDE to find optimal solutions in data sets with large dimensions. It is confirmed by CGPSO FPA train value that is statistically better than CoDE FPA train value in all data sets.

Table 4 represents the experimental results of Learning-to-Rank software defect prediction implementation in single-release data sets. All prediction models in table 4 have been trained with 75% of the total data sets, and then the trained models have been used to predict the defect module ranking for 25% of unused data on the model training process.

In Table 4, Chaotic Gaussian Particle Swarm Optimization has significantly better accuracy in 2 data sets, has an equivalent accuracy on 2 data sets and has worse accuracy results in 1 data sets. The result is approximately the same as the result in table 3. However, in Table 4 Chaotic Gaussian PSO has worse accuracy than CoDE on eclipse data sets. However, we still have not been able to find the exact reason CoDE has a higher accuracy on predicted defect modules ranking on eclipse data sets. We suspect that the model that we have trained with eclipse data sets using Chaotic Gaussian Optimization is overfitting.

### 5. Conclusion

In this research, we have presented and evaluated the implementation of Chaotic Gaussian Particle Swarm Optimization on the Learning-to-Rank software defect prediction approach for train model parameter. The results show the prediction model construct using Chaotic Gaussian Particle Swarm Optimization has better FPA train value on all data sets used in the model training process. An improved prediction accuracy is also seen in some data sets. Although one of the prediction results on Eclipse data sets produces worse accuracy than the prediction results produces by CoDE.
The overall results show prediction models construct using Chaotic Gaussian Particle Swarm Optimization gets better accuracy on 5 data sets, more or less the same results in 5 data sets and worse results in 1 data sets. Finally, we conclude that the application of Chaotic Gaussian Particle Swarm Optimization in Learning-to-Rank approach can improve the accuracy of the defect module ranking in data sets that have high-dimensional features (over 190 software metrics).

In the future, we plan to modify the fitness function on meta-heuristic to increase the prediction accuracy and explore the impact of various software metrics we have not used in this research on learning to rank software prediction approach.

References
[1] Jiang Y, Cukic B and Ma Y 2008 Techniques for evaluating fault prediction models Empir. Softw. Eng. 13 561–95
[2] Mende T and Koschke R 2011 Effort-aware defect prediction models Proc. Eur. Conf. Softw. Maint. Reengineering, CSMR 107–16
[3] D’Ambros M, Lanza M and Robbes R 2012 Evaluating defect prediction approaches: a benchmark and an extensive comparison Empir. Softw. Eng. 17 531–77
[4] Yang X, Tang K and Yao X 2012 A Learning-to-Rank Algorithm for Constructing Defect Prediction Models Intelligent Data Engineering and Automated Learning - IDEAL ed H Yin, J A F Costa and G Barreto (Natal, Brazil: Springer Berlin Heidelberg) pp 167–75
[5] Yang X, Tang K and Yao X 2015 A Learning-to-Rank Approach to Software Defect Prediction IEEE Trans. Reliab. 64 234–46
[6] Fenton N and Bieman J 2014 Software Metrics: A Rigorous and Practical Approach, Third Edition (CRC Press)
[7] Wang Y, Cai Z and Zhang Q 2011 Differential evolution with composite trial vector generation strategies and control parameters IEEE Trans. Evol. Comput. 15 55–66
[8] Zimmermann T, Premraj R and Zeller A 2007 Predicting defects for eclipse Third International Workshop on Predictor Models in Software Engineering, PROMISE’07
[9] Jia D, Zheng G, Qu B and Khan M K 2011 A hybrid particle swarm optimization algorithm for high-dimensional problems Comput. Ind. Eng. 61 1117–22
[10] Can H, Jianchun X, Ruide Z, Juelong L, Qiliang Y and Liqiang X 2013 A new model for software defect prediction using Particle Swarm Optimization and support vector machine 25th Chinese Control Decis. Conf. 4106–10
[11] Coelho R A, Guimaraes F dos R N and Esmin A A A 2014 Applying Swarm Ensemble Clustering Technique for Fault Prediction Using Software Metrics 13th Int. Conf. Mach. Learn. Appl. 356–61
[12] Wahono R S and Suryana N 2013 Combining particle swarm optimization based feature selection and bagging technique for software defect prediction Int. J. Softw. Eng. its Appl. 7 153–66
[13] Li P L, Herbsleb J, Shaw M and Robinson B 2006 Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc. Proceeding of the 28th international conference on Software engineering - ICSE ’06 (New York, New York, USA: ACM Press) p 413
[14] Li W, Huang Z and Li Q 2016 Three-way decisions based software defect prediction Knowledge-Based Syst. 91 263–74
[15] Kennedy J and Eberhart R 1995 Particle swarm optimization IEEE Int. Conf. Neural Networks 6 1942–8
[16] Weyuker E J, Ostrand T J and Bell R M 2010 Comparing the effectiveness of several modeling methods for fault prediction Empir. Softw. Eng. 15 277–95