ISSN: 2633-4828 Vol. 5 No.4, December, 2023

International Journal of Applied Engineering & Technology

A HYBRID FEATURE SELECTION ALGORITHM WITH NEURAL NETWORK FOR SOFTWARE FAULT PREDICTION

Khalaf Khatatneh[1], Nabeel Al-Milli[2], Amjad Hudaib[3] and Monther Tarawneh[4*]
[1]Khalaf Khatatneh, Prines Abdullah bin Ghazi for Communication and Information Technology College, Al-Balqa Applied University , Jordan
[2]Nabeel Al-Milli, Al-Zarqa Private University , IT College Jordan
[3]Amjad Hudaib, King Abullah II Iben Al-Husain College, Jordan University, Jordan
[4]Monther Tarawneh, College of IT, Isra University, ordan
Dr.khalf@bau.edu.jo, n.almilli@zu.edu.jo, ahudaib@ju.edu.jo and mtarawneh@iu.edu.jo

ABSTRACT

Software fault prediction identify potential faults in software modules during the development process. In this paper, we present a novel approach for software fault prediction by combining a feedforward neural network with particle swarm optimization (PSO). The PSO algorithm is employed as a feature selection technique to identify the most relevant metrics as inputs to the neural network. Which enhances the quality of feature selection and subsequently improves the performance of the neural network model. Through comprehensive experiments on software fault prediction datasets, the proposed hybrid approach achieves better results, outperforming traditional classification methods. The integration of PSO-based feature selection with the neural network enables the identification of critical metrics that provide more accurate fault prediction. Results shows the effectiveness of the proposed approach and its potential for reducing development costs and effort by detecting faults early in the software development lifecycle. Further research and validation on diverse datasets will help solidify the practical applicability of the new approach in real-world software engineering scenarios.

Index Terms – feature selection, neural network, particle swarm optimization, software fault prediction.

INTRODUCTION

Software fault prediction (SFP) problem is considered one of the most interesting classification tasks that determine whether a software module is faulty by considering some characteristics or parameters collected from software projects. Software faults leads the system to work differently compared with its expected behavior, and testing process for each activity in software system will be need high cost and time; therefore, SFP help in reducing unnecessary fault by finding efforts during the development of the software system [1]. Fixing defects typically consumes about 80% of the total budget of a software project [2]. Such cost can be significantly reduced if defects are fixed in an early stage [3], [4].

Several studies of SFP compare the performance of various methods (e.g., Artificial Neural Network (ANN), Support Vector Machine (SVM), Decision Trees (DTs) and Adaptive Neuro Fuzzy Inference System (ANFIS)) to determine the most applicable classification method for SFP. In addition to classification methods, several metric groups, such as process-level metrics, class-level metrics or method-level metrics, are defined, and some preprocessing techniques are suggested to extract the most useful techniques. The experiments in these studies are typically conducted on public datasets; the most popular datasets are available in PROMISE [5], [6], therefore the experiments in this study are performed on versions of the Ant, Camel, jEdit, Xalan, Log4j and Lucene [7] datasets.

In this paper, we have been compared between the results of fault prediction using Artificial Neural Network (ANN) with and without feature selection algorithm.

This paper is organized as follows: the next section presents the literature review of the previous studies on software fault prediction problem. Section 3 presents the software fault prediction problem. A description of the datasets is presented in Section 4. The proposed algorithm is presented in Section 5. Section 5 presents the results of the proposed algorithm. Finally, a conclusion is presented in Section 6.

LITERATURE REVIEW

The most difficult phase in SFP is to choose the best technique for classifying the datasets, since not all techniques lead to high accuracy in prediction. Challagulla et al. [8] performed an empirical study using various machine learning techniques for software fault prediction. The study was performed over four software projects taken from the NASA data repository. Results found that not a single machine learning technique is consistent in predicting faults with higher accuracy across different datasets. Furthermore, results suggested that the best choice of a fault prediction technique depends on the dataset available at a particular moment.

Dejaeger et al. [9] investigated the performance of Bayesian Network Classifiers for software fault prediction. They have used 15 different Bayesian Network based classifiers and compared them with other popular machine learning techniques. Results found that Naive Bayes and Random Forest are the most accurate predictors of software faults among the techniques considered and the performance of best technique depends on the development context. Kanmani et al. [10] investigated the effectiveness of Probabilistic Neural Network (PNN), Back Propagation Neural Network (BPN) and Discrimination Analysis over a dataset collected from student projects. The experiment was carried out on a small system and found that overall PNN based prediction models performed well in comparison to other used techniques. Hall et al. [11] have presented a review study on fault prediction performance in software engineering. The objective of the study was to appraise the context of fault prediction model, used software metrics, dependent variables, and fault prediction techniques on the performance of software fault prediction. The review included 36 studies published between 2000 and 2010. According to the study, fault prediction techniques such as Naive Bayes and Logistic Regression have produced better fault prediction results, while techniques such as SVM and C4.5 did not perform well. Similarly, for independent variables, it was found that object-oriented (OO) metrics produced better fault prediction results compared to other metrics such as LOC and complexity metrics, this work also presented the quantitative and qualitative models to assess the software metrics, context of fault prediction, and fault prediction techniques [12]-[16].

Previous studies highlight the challenges in selecting the most suitable technique for software fault prediction and emphasize the importance of considering the specific dataset and development context. The results vary depending on the techniques evaluated and the metrics used, indicating the need for careful evaluation and experimentation when approaching software fault prediction.

METHODOLOGY

Software fault prediction aims to predict faults in software system by using some variables in these systems; because early prediction for faults will be useful to reduce the cost, effort and time in later phases in the system development life cycle (SDLC). Fig. 1 gives an overview of the software fault prediction process. It can be seen from the figure that three important components of the software fault prediction process are: Software fault dataset, software fault prediction techniques, and performance evaluation measures.

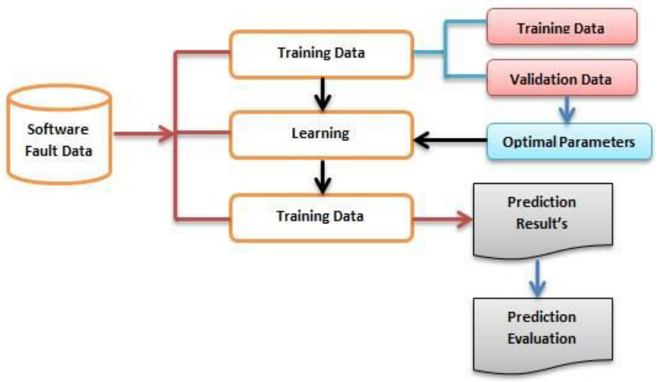


Fig. 1. Software Fault Prediction Process.

The software fault dataset is an important part in the process. It includes various relevant parameters for predicting faults. These parameters could be process-level metrics, class-level metrics, method-level metrics, or any other relevant information that captures the behavior or quality of the software.

The software fault predictions techniques is used to analyze the software fault dataset and make predictions about the presence or absence of faults in software modules. As mentioned earlier, these techniques can include Artificial Neural Network (ANN), Support Vector Machine (SVM), Decision Trees (DTs), Bayesian Network Classifiers, and others. The choice of the technique depends on the specific dataset and the context of the software development.

Values of various software metrics (e.g., LOC, Cyclomatic Complexity etc.) are extracted, which works as independent variables and the required fault information with respect to the fault prediction (e.g., the number of faults, faulty and non-faulty) work as the dependent variable. Generally, statistical techniques and machine learning techniques are used to build fault prediction models. Finally, the performance of the built fault prediction model is evaluated using different performance evaluation measures such as accuracy, precision, recall, and AUC (Area Under the Curve) [17]. The following equation presents the AUC equation:

$$AUC = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} 1_{pi>pj} \qquad (1)$$

Where i runs over all m data points with true label 1, and j runs over all n data points with true label 0; pi and pj denote the probability score assigned by the classifier to data point i and j, respectively. One is the indicator function: it outputs 1 if the condition (here pi>pj) is satisfied.

Overall, the software fault prediction process involves collecting and analyzing a software fault dataset, applying various prediction techniques, and evaluating their performance using appropriate measures. By predicting faults early in the software development life cycle (SDLC), organizations can potentially reduce costs, effort, and time associated with testing and fixing defects.

A. Datasets

In this paper, we select five different datasets from PROMISE repository: Ant, JEdit, Camel, Xalan, Log4j and Lucene projects. The datasets consist of 20 metrics that serve as input data for the software fault prediction task. Each metric provides information about a specific aspect of the software system. The table provides a brief summary of these metrics, and their descriptions. These metrics are used as the input variables, and the goal of the research is to predict the occurrence of faults based on these metrics. The output is a single binary value indicating whether a software module is faulty or not. By utilizing these datasets and their corresponding metrics, we aim to investigate the effectiveness of the proposed fault prediction algorithm in accurately predicting faults in software systems.

Proposed Hybrid Algorithm

The proposed methodology for solving the SFP problem in this paper is a hybrid approach that combines an ANN based on the feedforward algorithm PSO. The main objective is to utilize PSO as a feature selection technique to identify the most important metrics (features) for ANN. ANN is then employed as the classifier algorithm for fault prediction. Fig. 2 provides a visual representation of the proposed algorithm for software fault prediction. It illustrates the flow and integration of PSO and ANN in the hybrid methodology.

Table I A Brief Description of SFP Metric

Metric	Brief description
WMC	The value of the WMC is equal to the number of methods in the class.
DIT	The DIT metric provides for each class a measure of the inheritance levels from the object hierarchy top.

NOC	The NOC metric simply measures the number of immediate descendants of the class.
CBO	Metric represents the number of classes coupled to a given class.
RFC	Measures the number of different methods that can be executed when an object of that class receives a message
LCOM	Metric counts the sets of methods in a class that are not related through the sharing of some of the class fields.
LCOM3	Lack of cohesion in methods.
NPM	Simply counts all the methods in a class that are declared as public.
DAM	This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
MOA	The metric measures the extent of the part-whole relationship, realized by using attributes.
MFA	This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.
CAM	This metric computes the relatedness among methods of a class based upon the parameter list of the methods.
IC	This metric provides the number of parent classes to which a given class is coupled.
CBM	The metric measures the total number of new/redefined methods to which all the inherited methods are coupled.
AMC	This metric measures the average method size for each class
Ca	The Ca metric represents the number of classes that depend upon the measured class.
Ce	The Ce metric represents the number of classes that the measured class is depended upon.
CC	CC is equal to the number of different paths in a method (function) plus one.
Max(CC)	The greatest value of CC among methods of the investigated class.
Avg(CC)	The arithmetic mean of the CC value in the investigated class.

By combining PSO for feature selection and ANN as the classifier, the hybrid methodology aims to improve the accuracy and effectiveness of software fault prediction. The PSO component helps identify the most relevant metrics, reducing the dimensionality of the input data and enhancing the performance of the ANN classifier.

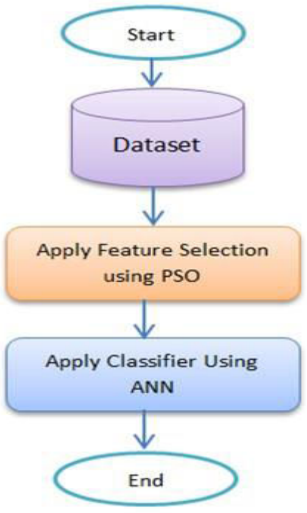
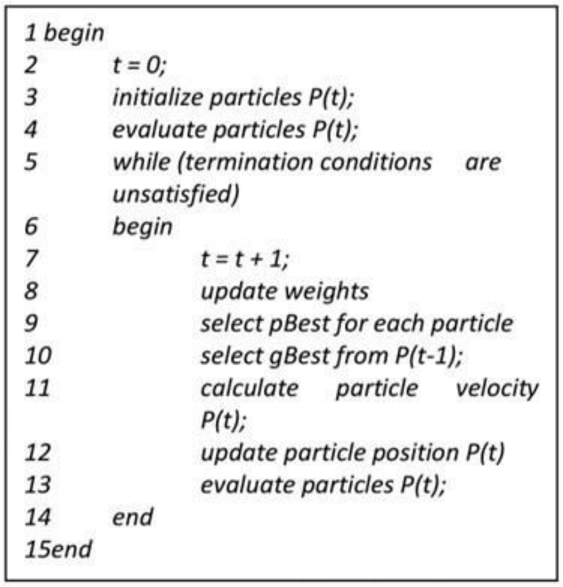


Fig. 2. A pictorial diagram for proposed algorithm

A. Particle Swam Optimization (PSO) Algorithm

PSO is population-based algorithm, which is developed by Kennedy and Eberhart [18], and widely accepted in different fields either in industrial and scientific research, based on the social behavior metaphor. The PSO algorithm includes some variables that greatly influence the algorithm performance, often stated as the exploration–exploitation tradeoff: Exploration is the ability to test various regions in the problem space in order to locate a good optimum, hopefully the global one. Exploitation is the ability to concentrate the search around a promising candidate solution in order to locate the optimum precisely [19]. A simple pseudo code for PSO is presented in Fig. 3.

The algorithm starts by initialization of a set of particles (solutions) based on a predefined number. Each particle is evaluated based on a fitness function. Each particle is searching for the optimum, by moving in the search space and hence has a velocity. Each particle remembers the position it was in where it had its best result so far (its global best).



```
begin
    t = 0;
    initialize particles P(t);
    evaluate particles P(t);
    while (termination conditions are unsatisfied)
    begin
        t = t + 1;
        update weights
        select pBest for each particle
        select gBest from P(t-1);
        calculate particle velocity P(t);
        update particle position P(t)
        evaluate particles P(t);
    end
end
```

Fig. 3. Particle Swarm Optimization (PSO) algorithm

B. Artificial Neural Network (ANN)

ANN is indeed a powerful classification algorithm that is inspired by the functioning of the human brain. It has been extensively used in various domains, including weather forecasting, stock market prediction, currency price forecasting, and many others.

An ANN consists of a network of interconnected artificial neurons (nodes or units) that work together to process and analyze input data and produce the desired output. The structure of a neural network involves setting the strengths of the connections between neurons. There are different approaches to determining the connection strengths (weights) in a neural network. One common approach is to explicitly set the weights based on prior knowledge or domain expertise regarding the input data. This is often referred to as weight initialization.

Another approach is to train the neural network by providing it with a set of training patterns, along with their corresponding desired outputs. During the training process, the network adjusts its weights based on a learning rule, which determines how the weights are updated to minimize the error between the predicted output and the desired output. Fig. 4 illustrates a basic feedforward ANN [10]. It typically consists of an input layer, one or more hidden layers, and an output layer. The arrows between the layers represent the weighted connections. In a feedforward network, the information flows from the input layer through the hidden layers to the output layer, without any cycles or feedback connections

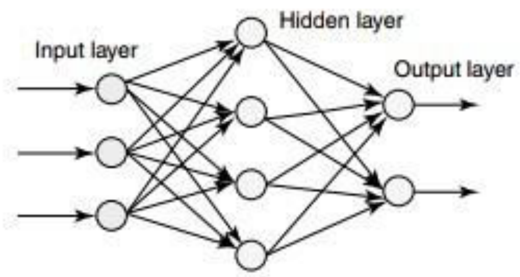


Fig. 4. Feedforward Artificial Neural Network

Experimental Results

The proposed hybrid algorithm, combining PSO for feature selection and ANN as the classification algorithm, was implemented using MATLAB R2014a. The simulations were conducted on a computer with an Intel Core i5 4.33 GHz processor. The algorithm was tested on standard datasets for software fault prediction, as described in Section 4. Table 2 provides the chosen parameters for the PSO algorithm and ANN, which were determined through preliminary experiments. These parameters play a crucial role in the performance of the algorithm.

Table II. Parameters for the PSO Algorithm and ANN

	Parameter	Value
PSO algorithm	GenerationNumber	100
	Populationsize	50
ANN	GenerationNumber	200
	Numberofhiddenneuron	5

To evaluate the performance of the proposed approach, 11 runs were performed on each dataset. The evaluation metric used is the Area Under the Curve (AUC) value [17]. AUC is a widely used metric in classification tasks and provides an overall measure of the model's discriminative power. Table 3 presents a brief description of the AUC values, which can be used to interpret the performance of the algorithm. A higher AUC value indicates better prediction performance.

Table III. Auc Value Descriptions

AUC Value	Description
AUC<0.5	Bad classification
0.5<=AUC<0.6	Poor classification
0.6<=AUC<0.7	Fair classification
.7<=AUC<0.9	Acceptable classification
0.8<=AUC<0.9	Excellent classification
AUC>=0.9	Outstanding classification

Table 4 presents the results obtained without feature selection and with feature selection. It is evident that the proposed hybrid algorithm outperforms the standard classification approach. For example, the AUC value for the Ant dataset improved from 0.622 to 0.778 after applying feature selection. Similar improvements were observed for nine out of 19 datasets after feature selection. Overall, the average of the obtained results is significantly higher than that of the standard classifier.

Table IV. Results of the Proposed Approach

Dataset	Without FS	With FS
Ant 1.7	0.622	0.778
Camel 1.0	0.71	0.985
Camel 1.2	0.517	0.546
Camel 1.4	0.828	0.773
Camel 1.6	0.653	0.608
Jedit 3.4	0.705	0.68
Jedit 4.0	0.825	0.78
Jedit 4.2	0.855	0.914
Jedit 4.3	0.68	0.989
log4j 1.0	0.782	0.913
log4j 1.1	0.77	0.895
log4j 1.2	0.858	0.641
lucene 2.0	0.637	0.62
lucene 2.2	0.621	0.702
lucene 2.4	0.55	0.514
xalan-2.4	0.796	0.855
xalan-2.5	0.694	0.64
xalan 2.6	0.694	0.64
xalan 2.7	0.336	0.518
Avg.	0.6912	0.7363

These findings indicate that the proposed hybrid algorithm, combining PSO feature selection and ANN classification, is effective in improving the accuracy of software fault prediction. It demonstrates its ability to overcome the limitations of traditional classification approaches and achieve better performance in terms of AUC values.

Table 5 provides comparisons between the results obtained from the proposed algorithm and the results reported in other relevant studies in the literature. The table allows for a comparison of the performance of different algorithms on various datasets. The proposed algorithm consistently outperforms several algorithms on specific datasets, indicating its effectiveness in software fault prediction. The hybrid approach of combining PSO feature selection with ANN classification brings improvements over the other methods in those particular cases. Furthermore, the average performance of the proposed algorithm is comparable to the results reported in other studies. This indicates that, on average, the proposed algorithm performs well compared to other algorithms used for software fault prediction

Table V. Comparisons Between The Results Obtained From The Proposed Algorithm And Other Relevant Algorithms.

Dataset	Without FS	With FS	ANN [1]	ANFIS [1]
Ant 1.7	0.622	0.778	0.8468	0.8184
Camel 1.0	0.71	0.985	0.9242	0.8939
Camel 1.2	0.517	0.546	0.6008	0.6009
Camel 1.4	0.828	0.773	0.7911	0.8132
Camel 1.6	0.653	0.608	0.6807	0.7143
Jedit 3.4	0.705	0.68	0.8796	0.8997
Jedit 4.0	0.825	0.78	0.8246	0.7826
Jedit 4.2	0.855	0.914	0.875	0.9755

Jedit 4.3	0.68	0.989	0.4613	0.9115
log4j 1.0	0.782	0.913	0.8929	0.8857
log4j 1.1	0.77	0.895	0.9018	0.9018
log4j 1.2	0.858	0.641	0.7804	0.7719
lucene-2.0	0.637	0.62	0.8492	0.8651
lucene-2.2	0.621	0.702	0.7628	0.7457
lucene-2.4	0.55	0.514	0.8248	0.8148
xalan-2.4	0.796	0.855	0.8186	0.8197
xalan-2.5	0.694	0.64	0.6747	0.6633
xalan-2.6	0.694	0.64	0.6821	0.6782
xalan-2.7	0.336	0.518	0.8167	0.8589
Avg.	0.6912	0.7363	0.7835	0.8113

CONCLUSION

In conclusion, the paper presents a hybrid methodology for software fault prediction that combines Artificial Neural Network (ANN) with Particle Swarm Optimization (PSO) for feature selection. The proposed algorithm aims to improve the accuracy of fault prediction and reduce unnecessary costs, effort, and time in the software development life cycle. Through experiments conducted on standard datasets for software fault prediction, it is demonstrated that the proposed hybrid algorithm outperforms standard classification approaches. The results show significant improvements in the Area Under the Curve (AUC) values, indicating better prediction performance with the inclusion of feature selection.

Comparisons with other algorithms reported in the literature also highlight the effectiveness of the proposed algorithm. It shows that the proposed approach can outperform several algorithms on specific datasets, and its average performance is comparable to other existing methods. Overall, the findings suggest that the hybrid algorithm combining PSO feature selection and ANN classification offers a promising approach for software fault prediction. It provides improved accuracy and has the potential to reduce costs associated with software development by identifying faults at an early stage.

The proposed hybrid algorithm shows promise for software fault prediction, and its results warrant further investigation and potential application in practical software development scenarios.

REFERENCES

[1] E. Erturk and E. A. Sezer, "Iterative software fault prediction with a hybrid approach," Applied Soft Computing, vol. 49, pp. 1020-1033, 2016.

[2] S. Planning, "The economic impacts of inadequate infrastructure for software testing," National Institute of Standards and Technology, vol. 1, 2002.

[3] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," Applied Soft Computing, vol. 33, pp. 263-277, 2015.

[4] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in Proceedings of the 30th international conference on Software engineering, 2008, pp. 181-190.

[5] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," Applied Soft Computing, vol. 21, pp. 286-297, 2014.

[6] J. Chen, S. Liu, X. Chen, Q. Gu, and D. Chen, "Empirical studies on feature selection for software fault prediction," in Proceedings of the 5th Asia-Pacific Symposium on Internetware, 2013, pp. 1-4.

[7] "The promise repository of empirical software engineering data ", ed, 2015.

[8] V. U. Challagulla, F. B. Bastani, and I.-L. Yen, "A unified framework for defect data analysis using the mbr technique," in 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), 2006, pp. 39-46.

[9] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward comprehensible software fault prediction models using bayesian network classifiers," IEEE Transactions on Software Engineering, vol. 39, pp. 237-257, 2012.

[10] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," Information and software technology, vol. 49, pp. 483-492, 2007.

[11] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," Artificial Intelligence Review, vol. 51, pp. 255-327, 2019.

[12] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in Proceedings of the 6th international conference on predictive models in software engineering, 2010, pp. 1-10.

[13] C. Catal, B. Diri, and B. Ozumut, "An artificial immune system approach for fault prediction in object-oriented software," in 2nd International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX'07), 2007, pp. 238-245.

[14] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," IEEE Transactions on software Engineering, vol. 33, pp. 402-419, 2007.

[15] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," Models and methods of system dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, pp. 69-81, 2010.

[16] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," IEEE Transactions on software engineering, vol. 28, pp. 4-17, 2002.

[17] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on software engineering, vol. 20, pp. 476-493, 1994.

[18] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proceedings of ICNN'95-international conference on neural networks, 1995, pp. 1942-1948.

[19] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," Information processing letters, vol. 85, pp. 317-325, 2003.