International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:12, 2010

Recurrent Radial Basis Function Network for Failure Time Series Prediction

Ryad Zemouri, Paul Ciprian Patic

***Abstract*—**An adaptive software reliability prediction model using evolutionary connectionist approach based on Recurrent Radial Basis Function architecture is proposed. Based on the currently available software failure time data, Fuzzy Min-Max algorithm is used to globally optimize the number of the k Gaussian nodes. The corresponding optimized neural network architecture is iteratively and dynamically reconfigured in real-time as new actual failure time data arrives. The performance of our proposed approach has been tested using sixteen real-time software failure data. Numerical results show that our proposed approach is robust across different software projects, and has a better performance with respect to next-step-predictability compared to existing neural network model for failure time prediction.

***Keywords*—**Neural network, Prediction error, Recurrent Radial Basis Function Network, Reliability prediction.

I. Introduction

SOFTWARE reliability is defined as the probability that the software will function during a specified period of time. The time between successive failures or the cumulative failure time is a vital indication of software reliability. Most of the existing analytical software reliability growth models depend on a priori assumptions about the nature of software faults and the stochastic behavior of software failure process [5], [6], [14], [15], [18], [22]. As a result, each model has a different predictive performance across various projects. A single universal model that can provide highly accurate predictions under all circumstances without any assumptions is most desirable [14], [15], [18]. It has been shown that a neural network approach is a universal approximator for any non-linear continuous function with an arbitrary accuracy [5], [8] and [17]. Consequently, it has become an alternative method in software reliability modeling, evaluation and prediction. Karunanithi N., Whitley D., Malaiya Y.K. in [14] and [15] were the first who propose a neural network approach for software reliability prediction. From other point of view Adnan W.A., Yaacob M.H. in [1] and, then, Adnan W.A., Yaacob M.H., Anas R., Tamjis M.R. in [2], Aljahdali S.H., Sheta A., Rine D. in [3] and [4], Ho S.L., Xie M., Goh T.N. in [11], Park J.Y., Lee S.U., Park J.H. in [18] and Sitte R. in[19] have also made contributions to software reliability prediction using neural networks, and have obtained better results compared to the traditional analytical models with respect to predictive performance [8].

Most of the published literature used single-input single-output neural network architecture to build software reliability growth models. Some examples are: cumulative execution time as input and the corresponding accumulated number of defects disclosed as desired output [14] and [15], and failure sequence number as input and cumulative failure time as desired output [18]. Recent studies focus on modeling software reliability based on multiple-delayed-input single-output neural network architecture. Aljahdali S.H., Sheta A., Rine D. in [3] used the most recent four day's faults observed before the current day as multiple delayed inputs to predict the number of faults initially resident at the beginning of testing process. Cai, K.Y., Cai, L., Wang, W.D., Yu, Z.Y., Zhang, D in [5] used the most recent 50 inter-failure time as multiple delayed inputs to predict the next failure time. However, there has been limited research on a systematic study on the impact of neural network architecture on performance.

Another potential problem of the existing neural network approach is arbitrary data partitioning of training and testing proportion. For example, 70% of the collected data were used in the training phase, and the remaining 30% of the collected data were used in the testing phase [3]. Approximately 20% of data were used for training, and all the remaining data were used for validation in [18].

The fixed number of collected data (last 30 failure data) was used for validation purposes in Cai's experiments irrespective of the total data set size [5]. Ho S.L., Xie M., Goh T.N. in [11] adopted a general 80–90% training and 10–20% testing proportion out of total 74 data points, and more specifically, 10 observations were used as out-of-sample testing set. For on-line applications, the number of failure data increases over time. The fixed neural network architectures do not address the effect on the performance of prediction as the number of input failure time data increases.

Many applications using radial basis function networks (RBF) for system output prediction use only one or two basis functions, the most popular being the Gaussian function. This function may not always be appropriate and the aim of this paper is to show the variation of test set error among six recognized basis functions. Each RRBF network has been

Ryad Zemouri. is with the Laboratoire d'Automatique du CNAM, 2 Rue Conté, 75003 Paris, France (corresponding author to provide phone: +33 01 40 27 21 35 / +33 01 58 80 88 56; fax: +33 01 40 27 21 97; e-mail: ryad.zemouri@cnam.fr).

Paul Ciprian Patic is with Valahia University of Targoviste, Electrical Engineering Faculty, Automatics, Informatics and Electrical Engineering Department, 18-24 Unirii Boulevard, 130082, Targoviste, Romania (e-mail: patic@valahia.ro).

trained using a two-stage approach, the k-means clustering algorithm for the first stage and singular value decomposition for the second stage. For this type of network configuration the results indicate that the choice of basis function (and, where appropriate, basis width parameter) is data set dependent and evaluating all recognized basis functions suitable for RRBF networks is advantageous [23].

In this paper we propose an on-line adaptive software reliability prediction model using evolutionary connectionist approach based on Fuzzy Min-Max algorithm. Unlike traditional neural network failure time data modeling approach, we want to model the inter-relationship among software failure time data instead of the relationship between failure sequence number and failure time data. The inputs and outputs of the neural network are all failure time data. During an on-line failure prediction process, Fuzzy Min-Max algorithm is used to globally optimize the neural network architecture after every occurrence of software failure time data. The optimization process determines the number of the k neurons in the hidden layer and then initializes the k centers. Thus the k-means technique is used to converge to a good and stable result. This improves the capability of the neural network to generalize well when the data is not known, mitigates the problem of over-fitting in earlier approaches, and enhances the accuracy of next-step prediction of the cumulative failure time. The corresponding globally optimized number of neurons in the hidden layer will be iteratively and dynamically reconfigured as new failure time data arrives. Our proposed on-line prediction approach is tested using different real time software failure data sets. The following sections describe the implementation of our approach.

II. On-line reliability prediction

The proposed on-line adaptive software failure prediction system shown in Fig. 1 consists of a failure history database and an evolutionary connectionist model. When a software failure, x_i, occurs, the failure history database is updated and the accumulated failure data ($x_i, x_2, ..., x_i$) is made available to the evolutionary connectionist model. The Fuzzy Min-Max algorithm optimization process determines the optimal or near-optimal number of the Gaussian hidden neurons. This information is then used to dynamically reconfigure the neural network architecture for predicting the next-step failure, $\hat{x}_{i+1}$.

Most of the traditional neural network approaches use an arbitrary data partition for training and testing [3], [5], [11], [18]. Since our proposed algorithm is tailored for on-line applications, we investigate the effect of the size of training patterns on the next-step prediction error. When the number of software failure time data is large, the amount of time taken for training with all available data can be a limiting factor. The rate of occurrence of the failure data depends on the maturity of the software. If the failure occurrence rate is high, the amount of time available to accurately predict the next failure is small. With these practical constraints, trade-offs between the size of data to be used for training and the next-step prediction error become critical. Also, when selecting a subset of data for training, we determine if the data from the earliest failure observations or the most recent occurrences yields lower prediction error.

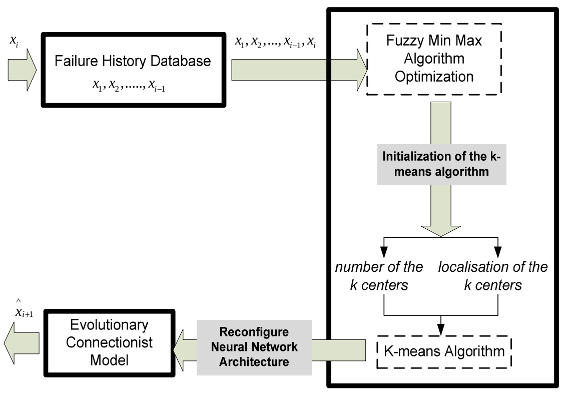
x_i
Failure History Database
$x_1, x_2, ..., x_{i-1}$
$x_1, x_2, ..., x_{i-1}, x_i$
Fuzzy Min Max Algorithm Optimization
Initialization of the k-means algorithm
number of the k centers
localisation of the k centers
K-means Algorithm
Reconfigure Neural Network Architecture
Evolutionary Connectionist Model
$\hat{x}_{i+1}$

Fig. 1. On-line adaptive failure prediction framework

III. Connectionist Implementation

A. The RBF network

The RBF consists of two layers observed in Fig. 2 with architecture similar to that of a two-layer MLP. The distance between an input vector and a prototype vector determines the activation of the hidden layer with the nonlinearity provided by the basis function. The nodes in the output layer usually perform an ordinary linear weighted sum of these activations, although non-linear output nodes are an option. Training RBF can be accomplished by two different approaches depending on whether the output functions are linear or nonlinear. In the nonlinear case, the advantage of training simplicity is lost since a nonlinear optimization of the output weights is required. Computationally this method is considerably more demanding, with training times equivalent to MLP. This approach is not considered here.

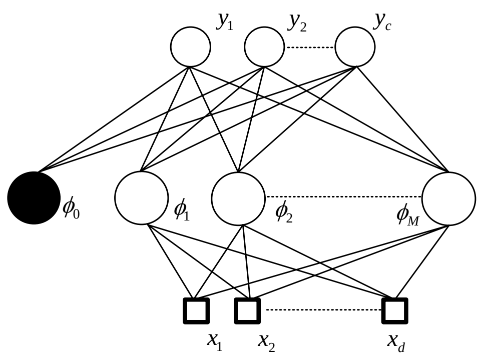

Fig. 2. Typical RBF network configuration

Mathematically, the network output for linear output nodes is expressed as follows:

$$y_k = \sum_{j=1}^{M} w_{kj}\phi_j(\|\mathbf{x}-\mathbf{u}_j\|) + w_{k0}$$

where $\mathbf{x}$ is the input vector with elements x_i (where i is the dimension of the input vector); $\mathbf{u}_j$ is the vector determining the centre of the basis function ϕ_j with elements u_{ji}; w_{kj} are the final layer weights and w_{k0} is the bias. The basis function $\phi_j(.)$ provides the non-linearity and is discussed in the following section.

Training RBF with linear outputs is very fast and is accomplished in two stages. The first stage is unsupervised and accomplished by obtaining cluster centers of the training set input vectors.

A popular method is k-means clustering, the use of which was first proposed by Moody and Darken, although other methods can be employed such as Max–Min, Kohonen network [16] or simply using a random selection of centers from the data.

The second stage consists in solving a set of linear equations, the solution of which can be obtained by a matrix inversion technique such as singular value decomposition or by least squares.

B. *The Recurrent Radial Basis function network (RRBF)*

The Recurrent RBF neural network considers time as an internal representation [7] and [9]. The dynamic aspect is obtained by the use of an additional self-connection on the input neurons with a sigmoid activation function. These looped neurons are a special case of Locally Recurrent Globally Feed-forward architecture, called local output feed-back [20]. The RRBF network can thus take into account a certain past of the input signal.

Each neuron of the input layer gives a summation at the instant t between its input x_i and its previous output weighted by a self-connection w_{ii}: The output of its activation function is:

$$\begin{aligned} a_i(t) &= w_{ii}\xi_i(t-1) + x_i(t) \\ \xi_i(t) &= f\left(a_i(t)\right) \end{aligned} \tag{1}$$

where $a_i(t)$ and $\xi_i(t)$ represent respectively the neuron activation and its output at the instant t, f is the sigmoid activation function:

$$f\left(x\right) = \frac{1-\exp(-kx)}{1+\exp(-kx)} \tag{2}$$

To highlight the influence of this self-connection, we let evolve the neuron without an external influence. The initial conditions are: the input $x_i(t) = 0 \quad \forall t$ and $\xi_i(0) = \pm 1$. Neuron output evolves according to the following expression:

$$\xi_i(t) = f\left(a_i(t)\right) = \frac{1-\exp\left(-kw_{ii}\xi_i(t-1)\right)}{1+\exp\left(-kw_{ii}\xi_i(t-1)\right)} \tag{3}$$

Fig.3 shows the temporal evolution of the neuron output.

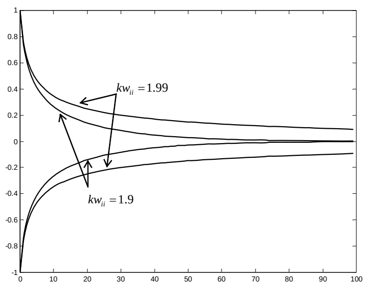

Fig. 3. The output of the looped neuron for two initial conditions: $\xi_i(0) = +1$ and $\xi_i(0) = -1$.

This evolution depends on two parameters: the self connection weight w_{ii} and the k value of the activation function parameter. The equilibrium points δ of the looped neuron satisfy the following equation:

$$\delta = w_{ii}\,f(\delta)$$

According to kw_{ii}, the looped neuron has one or more equilibrium points:

If $kw_{ii} \le 2$, the neuron has only one equilibrium point $\delta = 0$.

If $kw_{ii} > 2$, the neuron has three equilibrium points $\delta = 0$, $\delta^{+} > 0$, $\delta^{-} < 0$.

The looped neuron can thus exhibit two behaviors according to kw_{ii}: the forgetting behavior $kw_{ii} \le 2$ and the temporal memory behavior $kw_{ii} > 2$ [23], [24].

Self-connection gives to the neuron the capacity to memorize a certain past of the input data. The weight of this self-connection can be obtained by training, but the easier way to do it is to fix it a priori. The best performance is obtained with $kw_{ii} = 2$.

IV. Data validation

The performance of our proposed approach has been tested using Software Reliability Dataset application. The objective of this dataset is to collect failure interval data to assist software managers in monitoring test status and predicting schedules and to assist software researchers in validating software reliability models. The dataset consists of software failure data on 16 projects. Careful controls were employed during data collection to ensure that the data would be of high quality. The data was collected throughout the mid 1970's. It represents projects from a variety of applications including real time command and control, word processing, commercial, and military applications. The following table displays the size of each sample of failure data for each project in the dataset. Fig. 4 shows the cumulative failure time in hour of the second data set. This dataset, from table 1, is available on: *https://www.thedacs.com/databases/sled/swrel.php*.)

TABLE I
DATA SETS WITH THE NUMBER OF FAILURES USED IN THE EXPERIMENTS

System Code	***Number of Failures***
data_1	136
data_2	54
data_3	38
data_4	53
data_5	831
data_6	73
data_14C	36
data_17	38
data_27	41
data_40	101
data_SS1A	112
data_SS1B	375
data_SS1C	277
data_SS2	192
data_SS3	278
data_SS4	196

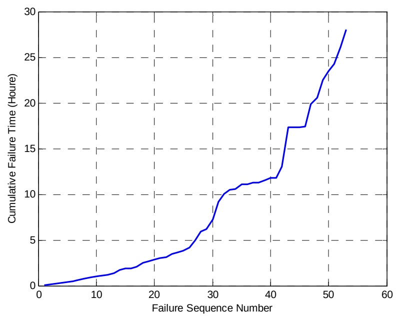

Fig. 4. Cumulative failure time for the data_2

To determine the next-step-predictability, we iteratively present the failure time data one at a time to the dynamically learned and optimized network. $x_i, x_{i+1}, \ldots\ldots x_{i+k-1}$ are used to predict the value of x_{i+k}.

$$RE = \left| \frac{\hat{x}_i - x_i}{x_i} \right| \tag{4}$$

where $\hat{x}_i$ is the predicted value of cumulative failure time, and x_i is the actual value of cumulative failure time. For example, using DATA-1, 90 of the next-step predicted values fall within 1% of their actual observed values. The results show that our proposed evolutionary neural networks approach provides highly accurate online prediction capability. The proposed evolutionary connectionist model dynamically learns and optimizes the neural network architecture whenever a new failure time data arrives, and is easily implemented to predict failures in real-time.

From Table 1 and Table 2 one has some data used in Fig. 5 (a - g) to represent graphically. One can realize that all the figures, from below, used data represented in both tables.

TABLE II
PERFORMANCES RESULTS OF NEXT-STEP-PREDICTABILITY

Next-step-predictability (RE ≤1%)							
data_1	data_2	data_3	data_4	data_5	data_6	data_14C	data_17
90%	*100%*	*85%*	*87%*	*97%*	*90%*	*91%*	*95%*
data_27	Data_40	data_SS1A	data_SS1B	data_SS1C	data_SS2	data_SS3	Data_SS4
95%	*77%*	*95%*	*89%*	*92%*	*90%*	*82%*	*87%*

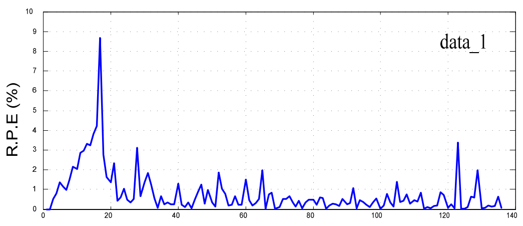

Fig. 5(a).

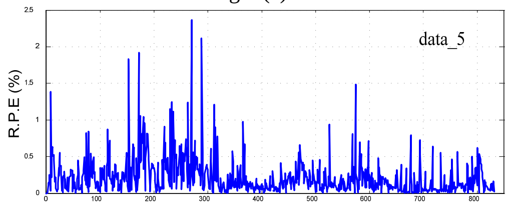

Fig. 5(b).

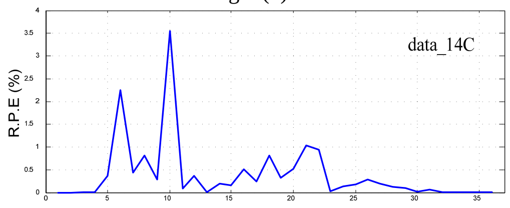

Fig. 5(c).

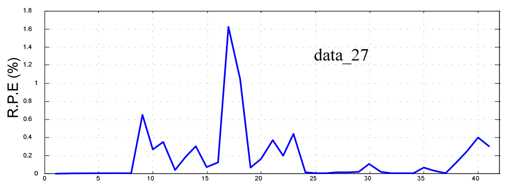

Fig. 5(d).

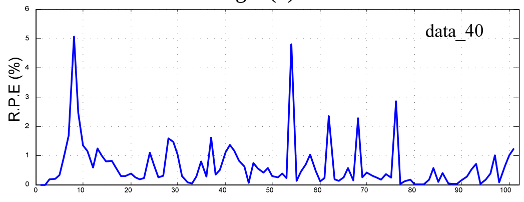

Fig. 5(e).

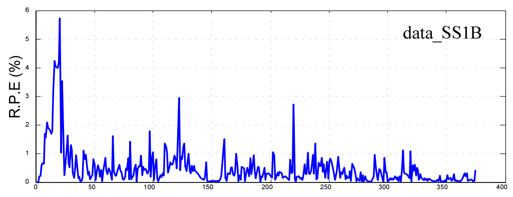

Fig. 5(f).

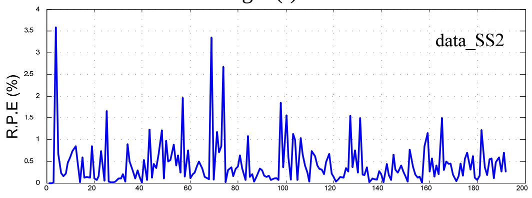

Fig. 5(g).

Fig. 5(a, b, c, d, e, f, g). Deviation between the next-step predicted value and the real value

V. Conclusion

In this paper, an on-line adaptive software reliability prediction model using evolutionary connectionist approach based on multiple-delayed-input single-output architecture is proposed.

Based on the currently available software failure time data, Fuzzy Min-Max algorithm is used to globally optimize the number of the k Gaussian nodes. This technique allows determining and initializing the k-centers of the neural network architecture in an iterative way.

The corresponding optimized neural network architecture is iteratively and dynamically reconfigured in real-time as new failure time data arrives.

Experimental results show that our proposed approach is robust across different software projects, and has a good performance with respect to next-step-predictability.

References

[1] Adnan, W.A., Yaacob, M.H., 1994. An integrated neural-fuzzy system of software reliability prediction. In: Proceedings of the 1st International Conference on Software Testing, Reliability and Quality Assurance. pp. 154–158.

[2] Adnan, W.A., Yaacob, M.H., Anas, R., Tamjis, M.R., 2000. Artificial neural network for software reliability assessment. In: 2000 TENCON Proceedings of Intelligent Systems and Technologies for the New Millennium. pp. 446–451.

[3] Aljahdali, S.H., Sheta, A., Rine, D., 2001. Prediction of software reliability: a comparison between regression and neural network non-parametric models. In: Proceedings of ACS/IEEE International Conference on Computer Systems and Applications. pp. 470–473.

[4] Aljahdali, S.H., Sheta, A., Rine, D., 2002. Predicting accumulated faults in software testing process using radial basis function network models. In: Proceedings of the 17th International Conference on Computers and their Applications. pp. 26–29.

[5] Cai, K.Y., Cai, L., Wang, W.D., Yu, Z.Y., Zhang, D., 2001. On the neural network approach in software reliability modeling. The Journal of Systems and Software 58 (1), 47–62.

[6] Cai, K.Y., Wen, C.Y., Zhang, M.L., 1991. A critical review on software reliability modeling. Reliability Engineering and System Safety 32 (3), 357–371.

[7] Chappelier J.C., Grumbach A., «*A Kohonen Map for Temporal Sequences*», Proceeding of neural Networks and Their Application, NEURAP'96, IUSPIM, Marseille, mars 1996, p. 104-110.

[8] Chua, C.G., Goh, A.T.C., 2003. A hybrid Bayesian back-propagation neural network approach to multivariate modeling. International Journal for Numerical and Analytical Methods in Geomechanics 27(8),651–667.

[9] Elman J.L., « *Finding Structure in Time* », Cognitive Science, vol. 14, juin 1990, p. 179-211.

[10] Fahlman, S.E., Lebiere, C., 1990. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University.

[11] Ho, S.L., Xie, M., Goh, T.N., 2003. A study of the connectionist models for software reliability prediction. Computers and Mathematics with Applications 46 (7), 1037–1045.

[12] Hochman, R., Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J.P., 1996. Using the genetic algorithm to build optimal neural networks for fault-prone module detection. In: Proceedings of the 7th International Symposium on Software Reliability Engineering. pp. 152–162.

[13] Hochman, R., Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J.P., 1997. Evolutionary neural networks: a robust approach to software reliability problems. In: Proceedings of the 8th International Symposium on Software Reliability Engineering. pp. 13–26.

[14] Karunanithi, N., Whitley, D., Malaiya, Y.K., 1992a. Prediction of software reliability using connectionist models. IEEE Transactions on Software Engineering 18 (7), 563–574.

[15] Karunanithi, N., Whitley, D., Malaiya, Y.K., 1992b. Using neural networks in reliability prediction. IEEE Software 9 (4), 53–59.

[16] Kohonen T., *Self-organised formation of topologically correct feature maps*, Biol. Cybern. 43 (1982) 59–69 (reprinted in Anderson and Rosen.eld, 1988).

[17] Leung, F.H.F., Lam, H.K., Ling, S.H., Tam, P.K.S., 2003. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. IEEE Transactions on Neural Networks 14 (1), 79–88.

[18] Park, J.Y., Lee, S.U., Park, J.H., 1999. Neural network modeling for software reliability prediction from failure time data. Journal of Electrical Engineering and Information Science 4 (4), 533–538.

[19] Sitte, R., 1999. Comparison of software-reliability-growth predictions: neural networks vs. parametric-recalibration. IEEE Transactions on Reliability 48 (3), 285–291.

[20] Tsoi C.T., Back A.D., « *Locally Recurrent Globally Feedforward Networks : A Critical Review of Architectures* », IEEE Transaction on Neural Networks Vol.05, pp. 229-239, 1994.

[21] Tsoukalas, L.H., Uhrig, R.E., 1996. Fuzzy and Neural Approaches in Engineering. Practical Aspects of Using Neural Networks. John Wiley & Sons, New York, Chapter 11, pp. 385–405.

[22] Utkin, L.V., Gurov, S.V., Shubinsky, M.I., 2002. A fuzzy software reliability model with multiple-error introduction and removal. International Journal of Reliability, Quality and Safety Engineering 9 (3), 215–227.

[23] Zemouri, R., Patic P.C., The effect of different basis functions for system output prediction, 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'2010, September 13-16, 2010, Bilbao Spain (Submitted for publication).

[24] Zemouri, R., Patic P.C., Prediction Error Feedback for Time Series Prediction: a way to improve the accuracy of predictions, Proceedings of the 4th EUROPEAN COMPUTING CONFERENCE (ECC '10), April 20-22, 2010, Bucharest, Romania, p. 58-62, ISSN 1790-5117, ISBN 978-960-474-178-6.