Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007

Parallel Learning of Large Fuzzy Cognitive Maps

Wojciech Stach, Lukasz Kurgan, and Witold Pedrycz

Abstract— **Fuzzy Cognitive Maps (FCMs) are a class of discrete-time Artificial Neural Networks that are used to model dynamic systems. A recently introduced supervised learning method, which is based on real-coded genetic algorithm (RCGA), allows learning high-quality FCMs from historical data. The current bottleneck of this learning method is its scalability, which originates from large continuous search space (of quadratic size with respect to the size of the FCM) and computational complexity of genetic optimization. To this end, the goal of this paper is to explore parallel nature of genetic algorithms to alleviate the scalability problem. We use the global single-population master-slave parallelization method to speed up the FCMs learning method. We investigate the influence of different hardware architectures on the computational time of the learning method by executing a wide range of synthetic and real-life benchmarking tests. We analyze the quality of the proposed parallel learning method in application to both dense and sparse large FCMs, i.e. maps that consist of several dozens of concepts. The parallelization is shown to provide substantial speed-ups, allowing doubling the size of the FCM that can be learned by parallelization with 8 processors.**

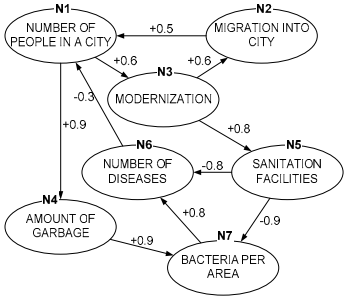
I. Introduction

A. Fuzzy Cognitive Maps

1) Overview

Proposed in 1986 by Bart Kosko [14], Fuzzy Cognitive Maps (FCMs) form a class of discrete-time Artificial Neural Networks. They represent knowledge in a symbolic manner and relate states, variables, events, outputs and inputs using a cause and effect approach. FCMs, when compared with neural networks, have several important advantages such as relative easiness to represent structured knowledge, and simplicity of the inference that is computed by numeric matrix operations [17].

The FCM structure is similar to a recurrent artificial neural network (RNN), where concepts are represented by neurons and causal relationships by weighted edges connecting the neurons. However, in contrast to FCMs, RNN neurons have external inputs, whereas in FCMs, the nodes (neurons) are only internally interconnected. Each of FCM's edges is associated with a weight value that reflects the strength of the corresponding relation. This value is usually normalized to the interval [–1,1]. Positive values reflect promoting effect, while negative ones describe inhibiting effect. The value of –1 represents full negative, +1 full positive and 0 denotes neutral relation. Other values correspond to different intermediate levels of causal effect. The graph representation is equivalent to a square matrix, called *connection matrix*, which stores all weight values of edges between corresponding concepts. Figure 1 shows an example of FCM that models city health issues [16].



	N1	N2	N3	N4	N5	N6	N7
N1	0	0	0.6	0.9	0	0	0
N2	0.5	0	0	0	0	0	0
N3	0	0.6	0	0	0.8	0	0
N4	0	0	0	0	0	0	0.9
N5	0	0	0	0	0	-0.8	-0.9
N6	-0.3	0	0	0	0	0	0
N7	0	0	0	0	0	0.8	0

Fig. 1. Example of FCM model and its equivalent connection matrix

In FCMs, each node has a value that reflects the degree to which the corresponding concept in the system is active at a particular iteration. This value, called *activation level*, is a floating-point number between 0 (inactive) and 1 (active). For a given concept, this value is calculated by taking into account the activation levels at the previous iteration of all the concepts that exert influence on it:

$$\forall j \in \{1,...,N\}, C_j(t+1) = f\left(\sum_{i=1}^{N} e_{ij} C_i(t)\right) \quad (1)$$

where: $C_j(t)$ – activation level of concept j^{th} at iteration t
e_{ij} – strength of relation from concept C_i to concept C_j
f – transformation function

The transformation function is used to reduce unbounded weighted sum to a certain range. The normalization hinders quantitative analysis, but, at the same time, it allows for comparisons between activation levels of different concepts.

A snapshot of activation levels of all nodes at a particular iteration defines the system state. It can be conveniently represented by a vector, called *state vector*, which consists of the nodes' activation values. *Initial state vector* refers to the system state at the first iteration. Successive states are calculated by iterative application of the formula (1).

Fuzzy Cognitive Maps have been successfully applied in various domains, including engineering [21][23], medicine [10], political science [12], economics [11], Earth and environmental sciences [6], etc.

This work was supported in part by the Walter Karplus Summer Research Grant, by the Alberta Ingenuity, and by the Natural Sciences & Engineering Research Council of Canada (NSERC)

W. Stach, L. Kurgan, and W. Pedrycz are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada (e-mail: {wstach, lkurgan, pedrycz}@ece.ualberta.ca)

2) Development

There are two main groups of approaches to develop Fuzzy Cognitive Maps: (1) deductive modeling (i.e., it uses an expert knowledge from the domain of application); and (2) inductive modeling (i.e., it uses learning algorithms to establish FCMs from historical data) A comprehensive overview of these methods can be found in [22].

Figure 2 shows a high-level diagram of the FCMs learning method based on real-coded genetic algorithms (RCGA), which is used in this paper. This method has been recently introduced and thoroughly tested [19][20]. RCGA is a floating-point extension to generic genetic algorithms. Section B.2 gives more information on both genetic algorithms and RCGA.

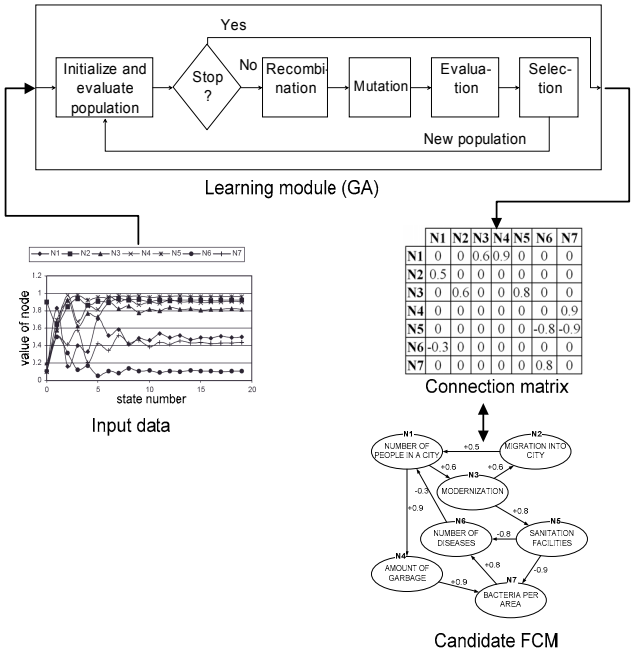


Fig. 2. High-level diagram of RCGA learning method for FCMs

The RCGA learning method uses input data to develop an FCM (*candidate FCM*), which is capable to mimic the historical data [19]. The input data is given as time series and consist of a sequence of state vectors that describe a given system at different time points. Since FCM can be fully represented by its connection matrix, the learning goal is to establish N^2 parameters, where N denotes the number of concepts. They correspond to the weighted values of mutual relations between the concepts. The RCGA algorithm exploits the input data to find these values. The learning objective is to generate the same state vector sequence from the candidate FCM for the same initial state vector, as it is defined in the input data. At the same time, the candidate FCM generalizes the inter-relations between concept nodes, which are inferred from the input data. Therefore, it is suitable to perform simulations from different initial state vectors and, based on their results, to draw conclusions about the modeled system.

B. Parallelization of Genetic Algorithms

1) Introduction to parallel computing

Parallel computing is one of the popular techniques to speed up the process of solving complex computational problems [8]. It assumes simultaneous execution of the same task on multiple processors to obtain the results faster. The underlying assumption is that a problem being solved can be divided into smaller tasks, which can be executed simultaneously with a certain level of coordination.

In parallel computing, it is crucial to use parallel algorithms to take advantage of hardware systems [25]. *Parallel algorithms*, in contrary to *sequential algorithms*, can be divided into parts performed in parallel. Subsequently, the partial results are put back together to obtain the final result. The task of finding an efficient parallel algorithm to solve a given problem can be very challenging as we often deal with sequential constrains. For instance, recursive solutions that require a result from previous iteration to calculate result in next iteration are very difficult to parallelize.

While implementing parallel algorithms, communication among the processors to coordinate the execution of subtasks must be considered. Hence, in practice, linear time speed-up vs. the number of processors is very difficult to obtain. The communication is usually realized either using shared memory or message passing approach [15].

2) Overview of genetic algorithms

Genetic algorithms (GAs) are a search technique to find solutions to optimization and search problems [5][7]. They have many advantages, which include broad applicability, ease of use, and global perspective, to name a few. GAs originate from evolutionary biology and use genetically inspired mechanisms, such as mutation, selection, and crossover. These operators are applied to a population of chromosomes that is maintained throughout the entire searching process. Each chromosome represents a solution to the problem, and its quality is quantified by a fitness value that is calculated from the fitness function. The GAs usually start from a randomly generated population and modify it in subsequent generations. In each generation, a new population is formed by using genetic operators, and by exploiting fitness values of the chromosomes. The idea is to produce better solutions to the problem over the evolving generations. GAs can be successfully applied to large, complex, and poorly understood search spaces, in which classical optimization tools are often inappropriate. The detailed information about GAs can be found in [5][7][24].

Several extensions to the generic GAs have been introduced. In this paper, a real-coded genetic algorithms (RCGA) approach is used as a part of the method for FCMs learning. RCGA is a floating-point extension to GAs. In RCGA chromosomes are represented as floating-point vectors in contrast to GAs that use binary vectors. This extended chromosome representation makes RCGA more

effective to tackle optimization problems with continuous variables. Although the genetic operators in RCGA are revised in order to deal with floating-point values, the main principles of both GA and RCGA are the same. Comprehensive description of the RCGA is given in [9].

The genetic algorithms can be parallelized in several different ways that are compatible with the RCGA learning. Parallelization can be implemented based on a single population, or by splitting the population into subpopulations. In this paper, a single population method (global single-population master-slave) is chosen. This approach parallelizes the evaluation of fitness function, which is usually the most time-consuming part of the GA's optimization. The implementation is usually done as a master-slave structure, in which the master process stores the population and the slaves evaluate the fitness. With this feature in mind, the entire population is split up into subsets, which are then assigned to different available processors. Communication between the processes occurs only when slaves receive a subset of individuals to evaluate and when they return the fitness values to the master process. This parallelization architecture does not affect the behavior of the genetic algorithm, since no additional restrictions are imposed on the genetic operators such as crossover or selection. This simplicity motivated our selection of this parallelization approach. Additional information on different approaches to parallelization of genetic algorithms can be found in [1][3][4][13].

II. Motivation and Methods

The bottleneck of the RCGA learning method for FCMs is its scalability, as the number of parameters to be established grows quadratically with the map size (number of concepts). In addition, genetic optimization is time consuming when employed to problems with large number of variables. As a result, RCGA method has been applied to FCMs which consist of up to 10 concepts. This restriction substantially limits the applicability of this learning approach. Since there are no other quality inductive techniques, learning of larger size FCMs from data was not possible. At the same time, in some areas such as systems biology, the underlying networks that could be modeled with FCMs are large and consist of at least several dozens of nodes. Moreover, recently proposed extension to generic FCMs, called *higher-order fuzzy cognitive maps* [18], requires even more parameters to be established during the learning process. These issues call for development of learning approaches for FCMs that would be fast enough to be applied to larger systems.

Therefore, this paper has two main goals:

- to propose a time-efficient learning method for FCMs based on RCGA approach. To satisfy this goal, we use a parallelized approach to real-coded genetic algorithm in order to achieve boost in execution time. In this paper, the global single-population master-slave method for parallelization of GAs is used due to its simplicity.
- to test the proposed parallelized RCGA learning method on a set of large and diverse FCMs to assess accuracy of the developed FCMs and the amount of obtained speed-up (when compared with sequential learning). This paper includes results of experiments with synthetic FCMs that consist of 10, 20, 40, and 80 concepts. Additionally, the tests are carried out with a large real-life map.

III. Evaluation

A. Data Sets

Similarly to experiments reported in [19], we used both synthetic and real-life data in our experiments.

1) Synthetic data

The synthetic data for our experiments were obtained by simulating randomly generated FCMs from random initial vectors. Four groups of experiments have been performed with maps that consist of 10, 20, 40, and 80 concepts, respectively. Additionally, for each group we generated two series of data with two different map densities (defined as the ratio of the non-zero relations weights to the total number of weights), 40% and 80%, respectively. Thus, as a result, we applied 8 different experimental setups. Additionally, for each setup, 5 independent maps were generated to assure statistical validity of the results.

2) Real-life data

We selected one of the largest FCMs found in literature to prepare the input data. In this case, the FCM was predefined by the domain experts and concerned factors that affected slurry rheology [2]. The input map included 13 concepts: gravity, mechanical properties of particles, physiochemical interaction, hydrodynamic interaction, effective particle concentration, particle-particle contact, liquid viscosity, effective particle shape, effective particle size, temperature, inter-particle attraction, floc/structure, and shear rate. The actual FCM can be found in [2]. Its density is 38.5%. We generated the input data from the initial vector, which was used to analyze the map in the original paper.

B. Evaluation Criteria

The evaluation measures of the proposed method are threefold:

- *execution time* – time needed to complete the learning
- *in-sample error* – difference between the input data, and data generated by simulating the candidate FCM from the same initial state vector as for the input data. The criterion is defined as a normalized average error between corresponding concept values at each iteration between the two state vector sequences [19].

$$error\_initial = \frac{1}{(K-1)\cdot N}\sum_{t=1}^{K-1}\sum_{n=1}^{N}\left|C_n(t)-\hat{C}_n(t)\right| \quad (2)$$

where $C_n(t)$ is the value of a node n at iteration t in the

input data, $\hat{C}_n(t)$ is the value of a node n at iteration t from simulation of the candidate FCM, K is the input data length, and N is the number of nodes

- *out-of-sample error* – evaluation of the generalization capabilities of the candidate FCM. To compute this criterion, both the input model and the candidate FCMs are simulated from ten randomly chosen initial state vectors. Subsequently, the error_initial value is computed for each of the simulations to compare state vector sequences generated by the input and the candidate FCM, and an average of these values is computed [19].

$$error\_behavior = \frac{1}{P \cdot (K-1) \cdot N} \sum_{p=1}^{P} \sum_{t=1}^{K-1} \sum_{n=1}^{N} \left| C_n^p(t) - \hat{C}_n^p(t) \right| \quad (3)$$

where $C_n^p(t)$ is the value of a node n at iteration t for data generated by input FCM started from p^{th} initial state vector, $\hat{C}_n^p(t)$ is the value of a node n at iteration t for data generated by candidate FCM started from p^{th} initial state vector, K is the input data length, and N is the number of nodes, and P is the number of different initial state vectors.

The first criterion is used to test the speed-up in execution time, whereas the other two are used to evaluate quality of the developed map. They are consistent with the criteria reported in [19].

C. Experimental Setup

The hardware used to execute the experiments was a state-of-the-art 12-way IBM p570 server with POWER5 processors. The code has been written in C++ using *OpenMP*[1], which is an application programming interface that supports multi-platform shared memory multiprocessing.

We repeated each experiment with different configuration of our hardware using the following scenario. We started by performing simulation on a single processor, and then, we doubled the number of processors in the subsequent experiments, up to eight processors. As a result, we carried out four independent simulations for each experiment.

IV. Results

Table I summarizes the experimental results for the synthetic data. The reported values have been calculated as averages obtained from 5 independent experiments (with different models) for each setup. The columns correspond to different experimental setups in terms of maps' sizes (10, 20, 40, and 80) and densities (40% and 80%), the rows correspond to different hardware configurations (the first column defines the number of processors), and the value in each cell expresses the average value of a corresponding criterion followed by the standard deviation. The criteria are labeled on the right hand side of the table.

TABLE I
Experimental Results for Synthetic Data

#	10 nodes 40%	10 nodes 80%	20 nodes 40%	20 nodes 80%	40 nodes 40%	40 nodes 80%	80 nodes 40%	80 nodes 80%	
1	608.4 ±0.55	608 ±0.00	1738.6 ±1.52	1737.8 ±1.10	5598.8 ±2.04	5602.6 ±1.14	19638.4 ±6.16	19837.6 ±7.16	Time [s]
2	316.8 ±0.45	316.8 ±0.45	988.8 ±9.28	992.2 ±8.46	2940.8 ±1.92	2946.6 ±2.07	10416.6 ±4.16	10476.6 ±9.16	
4	205 ±8.90	204.2 ±8.55	673 ±3.67	633.2 ±6.42	2502.8 ±11.36	2494.8 ±13.82	9195.6 ±24.05	9185.6 ±26.16	
8	191.8 ±13.95	163.2 ±18.55	453.4 ±19.35	451.0 ±20.83	1878.8 ±14.15	1862.8 ±21.04	6845.8 ±58.14	6865.0 ±66.16	
1	0.0027 ±0.0009	0.0045 ±0.0039	0.0055 ±0.0045	0.0091 ±0.0087	0.0230 ±0.0104	0.0147 ±0.0085	0.0337 ±0.061	0.0355 ±0.013	In-sample error
2	0.0030 ±0.0010	0.0043 ±0.0037	0.0056 ±0.0038	0.0077 ±0.0082	0.0207 ±0.0106	0.0081 ±0.0036	0.0373 ±0.0061	0.0364 ±0.013	
4	0.0029 ±0.0011	0.0044 ±0.0042	0.0063 ±0.0057	0.0084 ±0.0091	0.0224 ±0.0096	0.0077 ±0.0034	0.0378 ±0.0092	0.0366 ±0.015	
8	0.0031 ±0.0011	0.0045 ±0.0048	0.0057 ±0.0042	0.0099 ±0.0111	0.0232 ±0.0125	0.0086 ±0.0060	0.0379 ±0.0085	0.0361 ±0.013	
1	0.2012 ±0.114	0.1389 ±0.123	0.2584 ±0.121	0.1169 ±0.093	0.1782 ±0.077	0.1178 ±0.051	0.1482 ±0.023	0.1553 ±0.039	Out-of-sample error
2	0.1201 ±0.091	0.0341 ±0.041	0.1521 ±0.108	0.0963 ±0.106	0.1755 ±0.089	0.1177 ±0.110	0.1738 ±0.036	0.1600 ±0.033	
4	0.1303 ±0.072	0.0803 ±0.096	0.1447 ±0.078	0.1152 ±0.121	0.1656 ±0.039	0.1226 ±0.101	0.1632 ±0.030	0.1488 ±0.056	
8	0.0776 ±0.081	0.0288 ±0.023	0.1458 ±0.134	0.1058 ±0.109	0.1843 ±0.081	0.0946 ±0.086	0.1517 ±0.033	0.1494 ±0.041	

Table II summarizes the experimental results for the real-life FCM. The reported values have been calculated as averages obtained from 5 independent experiments for each setup.

TABLE II
Experimental Results for Real-life Data

#	Time [s]	In-sample error	Out-of-sample error
1	891.6 ±0.54	0.0048 ±0.0016	0.1166 ±0.038
2	464.2 ±0.45	0.0049 ±0.0018	0.1284 ±0.055
4	317.4 ±0.55	0.0053 ±0.0024	0.1424 ±0.101
8	269.6 ±18.62	0.0048 ±0.0019	0.1259 ±0.059

Analysis of the results for both synthetic and real-life data is presented in the two following subsections.

A. Execution time

Results in Table I show no significant influence of density on the execution time. The influence of the map's size and the number of processors on the execution time is shown in Figure 3. The execution time for the two map densities has been averaged out for each map size. Additionally, the columns are linearly normalized to the maximum execution time for each setup (which corresponds to the sequential learning). A number above each bar shows the execution time (in seconds).

[1] http://www.openmp.org

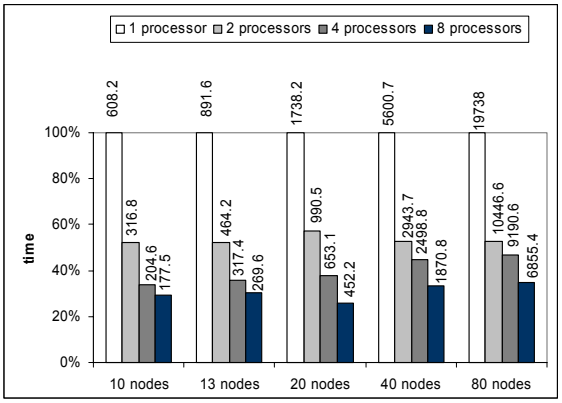


Fig. 3. Execution time vs. experimental setup

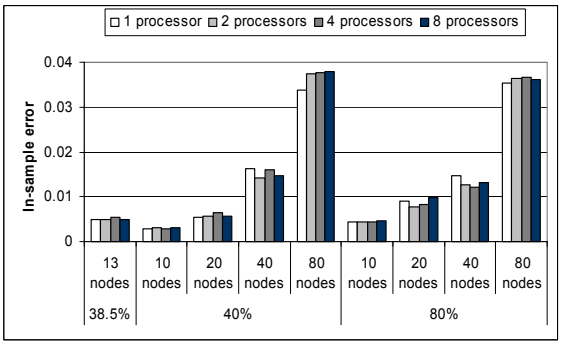


Fig. 4. In-sample error vs. experimental setup

In general, parallel learning on eight processors allows doubling the size of the FCM within virtually the same time that is required for the sequential learning. Comparison of the results obtained from simulations on a single processor (traditional, sequential learning) to those on two processors shows almost two-fold time-reduction for each setup (time gain varies between 43% and 47%). However, doubling the number of processors again does not result in the corresponding two-fold time reduction. When compared with sequential learning with a single processor, the time gain is, on average, 60% and 69% for four and eight processors, respectively. This is due to the fact that the execution time of the sequential constraints of performing genetic operations on population of chromosomes becomes a more significant part of the total execution time. This happens as the fitness function evaluations performed in parallel are completed faster on larger number of processors. Therefore, in this RCGA parallelization approach we observe decreasing returns along with increasing the number of processors.

B. Learning quality

Figure 4 shows relation between in-sample error and the size and density of FCMs, as well as the number of processors. A few interesting conclusions can be drawn from this figure. Firstly, the quality of learning gradually decreases along with the increasing input map size. The in-sample error value changes from 0.004 for FCMs with 10 nodes to 0.036 for FCMs with 80 nodes. This is due to complexity of the optimization problem, which drastically increases with the increase of the size of maps. However, even for the largest investigated maps the results are still significantly better from the baseline of this problem, which has been experimentally found at the value of 0.391 (see [19] for details). Secondly, the experiments with different hardware setups, i.e. different number of processors for a certain map size and density, are consistent in terms of the solution quality. The standard deviation of the in-sample error measured for different number of processors for a given setup is very small and varies from 0.00012 (10 nodes 80%) to 0.0020 (80 nodes 40%). This is in spite of the fact that the error values differ, which suggests that the learning method finds sub-optimal solutions of similar quality. The last observation is also consistent with the conclusions in [19]. Finally, the in-sample error does not depend on the input map's density. For sparser maps, it is only slightly lower for 10 and 20-nodes FCMs, and slightly higher for 40 and 80-nodes FCMs.

Figure 5 shows the relationship between out-of-sample error and the experimental setup, i.e. the map size and density, and the number of processors.

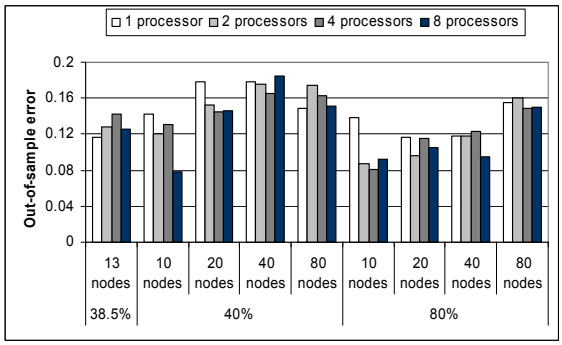


Fig. 5. Out-of-sample error vs. experimental setup

Figure 5 shows that the out-of-sample error slightly increases as the size of maps increases. However, it does not increase as rapidly for the larger maps as in case of the in-sample error. The out-of-sample error is, on average, equal to 0.109 for 10 nodes, 0.132 for 20 nodes, 0.144 for 40 nodes, and 0.156 for 80 nodes. The error values for maps with 10 nodes are consistent with values reported in [19]. Again, we stress that the learning results are still of high quality with respect to the baseline value (0.391). The out-of-sample error has larger standard deviation for experiments with the same map size and density performed on different number of processors when compared with corresponding values for the in-sample error, i.e., they differs between 0.0053 (80 nodes 80%) and 0.028 (10 nodes 40%). This is due to the fact that sub-optimal solutions (FCM models) for the out-of-sample experiments may differ from each other. Thus, even if all of them provide high quality results on previously seen data (in-sample error); these models can give different simulations for new initial vectors (out-of-sample error). Lastly, the relationship between the input map density and the out-of-sample error is more consistent here than in case of the in-sample error. On average, better quality is obtained for denser maps in each case, i.e. for 10, 20, 40, and 80 nodes.

We also analyze relation between the learning quality and the number of processors. The difference in the in-sample error between experiments repeated with the same setups (on 2, 4, and 8 processors, respectively) with respect to the sequential learning is very small (2-3% on average). In case of the out-of-sample error, these values are larger (6-12%), which is due to sub-optimal solutions found by the RCGA (see the above paragraph). However, when compared with the standard deviations obtained for the sequential setup, the difference in errors between the non-parallelized and the parallelized implementations is not significant.

V. Conclusion

This paper proposed a novel parallel approach to learning of FCMs. The method is based on RCGA learning, which has been previously reported as being able to develop high-quality FCMs from input data. Our motivation was to eliminate the main drawback of the RCGA based method, which is incapability of dealing with larger maps due to high computational complexity.

Our focus was to propose a solution that would combine the quality of the RCGA method with a substantial decrease of the execution time. We have proposed, implemented and tested a solution that parallelizes the genetic algorithm, which is the core of the RCGA approach.

The experimental results show that parallelization gives substantial improvement in the execution time. The parallelized learning of FCMs on eight processors was reported to be up to four times faster than the sequential learning. The proposed method allows learning maps that include several dozens of concepts in matter of few hours when using eight processors.

The parallelized RCGA method for learning FCMs has been tested on both synthetic and real-life data. The experimental results show that this method is able to provide high-quality solutions for large FCMs. Both in-sample and out-of-sample errors increased with the increasing map size; however the error values were still substantially smaller than the baseline error.

VI. Future Work

There are several future research directions based on this project. First, it would be interesting to measure how much of the computational time is devoted to do the fitness evaluation. Second, a comparison between different methods of RCGA parallelization to select the best approach will be investigated. Third, we plan to propose an alternative approach to speed up the learning process by exploiting inherent characteristics of FCMs, e.g. by dividing the input data into subsets, performing independent learning on each subset, and, finally, merging the sub-models. Last but not least, we plan to apply one of these learning methods to real-life problems, e.g. in the systems biology field.

References

[1] E. Alba, F. Luna, A. J. Nebro, and J. M. Troya, "Parallel heterogeneous genetic algorithms for continuous optimization," *Parallel Computing*, vol. 30, no. 5–6, pp. 699–719, 2004

[2] G. A. Banini, and R. A. Bearman, "Application of fuzzy cognitive maps to factors affecting slurry rheology," *Int. Journal of Mineral Processing*, vol. 52, no. 4, 1998

[3] E. Cantú–Paz, "A survey of parallel genetic algorithms," *Calculateurs Parallèles*, vol. 10, no. 2, pp. 141–171, 1998

[4] E. Cantú–Paz, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publishers, 2000

[5] K. Deb, *An Introduction to Genetic Algorithms*, SADHANA, 1999

[6] R. Giordano, G. Passarella, V. F. Uricchio, and M. Vurro, "Fuzzy cognitive maps for issue identification in a water resources conflict resolution system," *Physics and Chemistry of the Earth*, vol. 30, no. 6–7 (Special Issue), pp. 463–469, 2005

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison–Wesley, 1989

[8] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, Addison Wesley, 2003

[9] F. Herrera , M. Lozano , and J. L. Verdegay, "Tackling real–coded genetic algorithms: operators and tools for behavioural analysis," *Artificial Intelligence Review*, vol. 12, no. 4, pp. 265–319, 1998

[10] P. R. Innocent, and R. I. John, "Computer aided fuzzy medical diagnosis," *Information Sciences*, vol. 162, no. 2, pp. 81–104, 2004

[11] D. Kardaras, and G. Mentzas, "Using fuzzy cognitive maps to model and analyse business performance assessment," in *Advances in Industrial Engineering Applications and Practice II*, J. Chen, and A. Mital, (Eds), pp. 63–68, 1997

[12] M. Khan, and M. Quaddus, "Group decision support using fuzzy cognitive maps for causal reasoning," *Group Decision and Negotiation Journal*, vol. 13, no. 5, pp. 463–480, 2004

[13] Z. Konfrsť, "Parallel genetic algorithms: advances, computing trends, applications and perspectives," *Int. Parallel and Distributed Processing Symposium*, vol. 18, pp. 2303–2310, 2004

[14] B. Kosko, "Fuzzy cognitive maps," *Int. Journal of Man–Machine Studies*, vol. 24, pp. 65–75, 1986

[15] T. J. LeBlanc, and E. P. Markatos, "Shared memory vs. message passing in shared–memory multiprocessors," *IEEE Symposium on Parallel and Distributed Processing*, pp. 254–263, 1992

[16] K. C. Lee, W. J. Lee, O. B. Kwon, J. H. Han, and P. I. Yu, "Strategic planning simulation based on fuzzy cognitive map knowledge and differential game," *Simulation*, vol. 71, no. 5, pp. 316–327, 1998

[17] E. I. Papageorgiou, C. D. Stylios, and P. P. Groumpos, "Fuzzy cognitive map learning based on nonlinear Hebbian rule," In: T. D. Gedeon, and L. C. C. Fung, (Eds.), *Lecture Notes in Artificial Intelligence*, Springer–Verlag, vol. 2903, pp. 254–266, 2003.

[18] W. Stach, L. Kurgan, and W. Pedrycz, "Higher–order fuzzy cognitive maps," *North American Fuzzy Information Processing Society Conference (NAFIPS'06)*, 2006

[19] W. Stach, L. Kurgan, W. Pedrycz, and M. Reformat, "Genetic learning of fuzzy cognitive maps," *Fuzzy Sets and Systems*, vol. 153, no. 3, pp. 371–401, 2005

[20] W. Stach, L. Kurgan, W. Pedrycz, and M. Reformat, "Learning fuzzy cognitive maps with required precision using genetic algorithm approach," *Electronics Letters*, vol. 40, no. 24, pp. 1519–1520, 2004

[21] W. Stach, L. Kurgan, W. Pedrycz, and M. Reformat, "Parallel fuzzy cognitive maps as a tool for modeling software development project," *North American Fuzzy Information Processing Society Conference (NAFIPS'04)*, pp. 28–33, 2004

[22] W. Stach, L. A. Kurgan, and W. Pedrycz, "A survey of fuzzy cognitive map learning methods," In: P. Grzegorzewski, M. Krawczak, and S. Zadrozny, (Eds.), *Issues in Soft Computing: Theory and Applications*, Exit, pp. 71–84, 2005

[23] M. A. Styblinski, and B. D. Meyer, "Signal flow graphs versus fuzzy cognitive maps in application to qualitative circuit analysis," *Int. Journal of Man–Machine Studies*, vol. 35, pp. 175–186, 1991

[24] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994

[25] C. Xavier, and S. S. Iyengar, *Introduction to Parallel Algorithms*, Wiley–Interscience, 1998