A Study on Fuzzy Cognitive Map Optimization Using Metaheuristics

Aleksander Cisłak[1(✉)], Władysław Homenda[2], and Agnieszka Jastrzębska[1]

[1] Faculty of Mathematics and Information Science,
Warsaw University of Technology,
ul. Koszykowa 75, 00-662 Warsaw, Poland
{a.cislak,a.jastrzebska}@mini.pw.edu.pl
[2] Faculty of Economics and Informatics in Vilnius, University of Bialystok,
Kalvariju g. 135, LT-08221 Vilnius, Lithuania
homenda@mini.pw.edu.pl

Abstract. Fuzzy Cognitive Maps (FCMs) are a framework based on weighted directed graphs which can be used for system modeling. The relationships between the concepts are stored in graph edges and they are expressed as real numbers from the $[-1, 1]$ interval (called weights). Our goal was to evaluate the effectiveness of non-deterministic optimization algorithms which can calculate weight matrices (i.e. collections of all weights) of FCMs for synthetic and real-world time series data sets. The best results were reported for Differential Evolution (DE) with recombination based on 3 random individuals, as well as Particle Swarm Optimization (PSO) where each particle is guided by its neighbors and the best particle. The choice of the algorithm was not crucial for maps of size roughly up to 10 nodes, however, the difference in performance was substantial (in the orders of magnitude) for bigger matrices.

1 Introduction

Real-world phenomena modeling requires a framework that would not be hindered a by variety and diversity of relevant information. Standard methods, for instance for time series modeling, are predominantly numerical and are not well-fitted to process data in a form different than a sequence of numbers. An impressive range of fuzzy and granular models has emerged as a remedy for such issues. Fuzzy Cognitive Maps (FCMs) have been proposed by Kosko [11] in 1986 as an alternative framework for phenomena modeling.

FCMs represent knowledge in the form of a directed graph. Phenomena are stored in vertices, while edges represent their relationships. These relationships are expressed as real numbers from the $[-1, 1]$ interval. Weight matrix (or connection matrix) is a formal representation of each FCM as it gathers all weights in the map. In our research we focus on the application of FCMs to time series modeling, a domain relatively new as it has emerged in the 2000s [20].

In this paper, we present a study on a very important aspect of modeling with FCMs, namely on weight matrix learning procedures. In general, the core of each FCM, the weight matrix, can be constructed in three ways: (a) manually,

K. Saeed and W. Homenda (Eds.): CISIM 2016, LNCS 9842, pp. 577–588, 2016.
DOI: 10.1007/978-3-319-45378-1_51

by human experts; (b) automatically, using optimization algorithms; (c) with the combination of the two aforementioned options.

The first and the last option often turn out to be inapplicable, as they require human expert knowledge. A convenient alternative is offered by automated approaches that are able to determine the shape of the weight matrix. In particular, the literature of the topic recommends the application of various metaheuristic optimization procedures [15,19]. Inspired by the current developments in the studies on FCM optimization, we present a comparative study on various algorithms, which are presented later in this paper.

The study is supported by a series of empirical experiments that let us investigate and compare the quality of obtained maps. The experiments were conducted for a few time series data sets (containing both real-world and synthetic data) and for maps of various sizes. In contrast to the studies on FCM optimization reported in the literature, in our experiments we also process very large maps, with up to 27 nodes (which turned out to be crucial in terms of algorithm performance). The key novel element introduced in this paper is a thorough comparison of the effectiveness of various optimization methods. Also, to the best of our knowledge, this is the first paper where we apply Harmony Search and Improved Harmony Search to FCM optimization.

2 Related Works

The need for automated methods for FCM weight matrix construction coincided with a rapid development of various search metaheuristics. Researchers recognized opportunities offered by this family of algorithms and applied them to FCM optimization.

Indeed, there are several important arguments speaking in favor of this approach. First, FCMs are fuzzy models and the key limitation of search metaheuristics, which is a lack of guarantee for the global optimal solution, is not a major concern. Hence, when it comes to FCMs, it is common to model phenomena with an "acceptably" good accuracy. In other words, there is no need for the best solution as long as we can arrive at a one which is "good enough". Secondly, weight matrix optimization is a difficult problem and standard optimization procedures are inadequate to handle it. Last but not least, metaheuristics are attractive: they are easy to use, they make few or no assumptions about the problem being optimized, and they provide a relatively good performance. Having the above in mind, more and more heuristic search algorithms have been adapted to FCM optimization. However, many of the latest contributions mostly build on the first methodologies that have been proposed.

The objective is to form a weight matrix by an iterative search procedure. The initial weight matrix is filled in randomly, which eliminates the necessity for involving human experts in the learning process. Algorithms explore the search space in order to find a weight matrix that satisfies to the greatest extent given fitness criterion. Usually, such fitness criterion is expressed as an error between map's responses and target values:

$$\min_{\mathbf{W}=[w_{ij}],w_{ij}\in[-1,1],i,j=1,\ldots,c} error\,(\mathbf{Y},\mathbf{T}) \tag{1}$$

where $\mathbf{Y} = [y_{ij}], y_{ij} \in [0,1], \quad i = 1,\ldots,c, \quad j = 1,\ldots,N$ is map's output. *error* is a measure of discrepancy between FCM's outputs ($\mathbf{Y}$) and desired target values, $\mathbf{T} = [t_{ij}], t_{ij} \in [0,1], \quad i = 1,\ldots,c, \quad j = 1,\ldots,N$. N is the number of observations, and c is the number of nodes in the map. The literature-based approaches often use Mean Squared Error (MSE) [15,19]:

$$MSE = \frac{1}{N \cdot c} \cdot \sum_{j=1}^{N} \sum_{i=1}^{c} \left(y_{ij} - t_{ij}\right)^2, \tag{2}$$

where y_{ij} denotes map's ij-th response and t_{ij} is a corresponding target value.

Koulouriotis et al. [12] proposed the first methodology in 2001 that joined search heuristics with FCM learning. The authors applied Evolution Strategies to FCM learning in a case study of two FCMs, both with 6 nodes. Stach et al. [21] published in 2005 another important attempt at employing search heuristics for the benefits of FCM learning. Cited authors applied Real-coded Genetic Algorithm (RCGA) to develop FCM weight matrix. Presented results for maps with up to 10 nodes confirm the soundness of this approach.

Briefly after the appearance of FCM learning algorithms employing genetic approaches, Parsopoulos et al. [17] applied Particle Swarm Optimization (PSO) to automated FCM learning. The authors modeled an exemplary industrial control problem: a tank and three valves that guard the level of a liquid. The fitness function was determined accordingly to fit the application domain and it represented a combination of conditions for the tank and the valves that stabilize the industrial process. Further applications of PSO and related swarm intelligence algorithms to FCM learning have been documented [13,15]. Also, a few other successful methods for FCM learning using heuristics other than ones already discussed have been proposed, e.g., Big Bang-Big Crunch [16].

3 Approach

Our goal was to determine the performance of various non-deterministic algorithms in the context of optimizing FCM weight matrix. Regular approaches turn out to be rather ill-suited to this problem. Let us clarify that we used a popular classical algorithm Limited-memory BFGS-B [4]. The limitations concern first and foremost large maps, with 17 nodes (289 weights) and more. In our experiments we have observed that as we add more nodes, a classical optimization algorithm starts to set more and more weights to its pre-defined upper and lower bounds (-1 and 1). Such a result is not acceptable for the model assumptions of FCMs and it produces high error values. Hence, a more suited approach is required.

The challenge is caused by the lack of a feasible exact algorithm that could ensure reaching a *global* minimum (using, e.g., an exhaustive combinatorial search) owing to the complexity of the problem. However, with metaheuristic

approaches there still exists a risk of reaching a *local* minimum, and there is no a priori approach to determine which evolutionary approach is better suited, which is the motivation behind our experimental evaluation.

All algorithms which were used for the optimization fall into the category of *population-based* algorithms, and most of them can be also described as *evolutionary*, since they feature operations such as breeding, mutation, and selection based on utility. Popular terms include a population, which describes a set of all individuals (candidate solutions) that are considered during a current iteration of the algorithm, and fitness, which describes the utility of an individual. An individual is essentially a multidimensional vector $V = [v_1, \ldots, v_d]$ which consists of a fixed number of features. The number of dimensions, d, usually corresponds to certain properties of real-world objects, or, in our case, the size of the FCM.

Since in our case the fitness describes an average error, we deal with a minimization problem. When we mention a better or worse fitness value, it means that it is associated with a lower or a higher fitness value, respectively (the terms error and fitness are used interchangeably). Similarly, a better individual is the one associated with a lower fitness.

Typical operation of a population-based algorithm proceeds as follows:

1. The initial population containing n random individuals is created.
2. The fitness of each individual is calculated.
3. A predefined number (e) of individuals with the best fitness is preserved (so-called elite). Afterwards, the population is refreshed, i.e. typically $n - e$ individuals are replaced with new (random) ones, although this might depend on fitness values (e.g., new individuals might enter the population only if they are better than the previous ones).
4. The population is updated using algorithm-specific tools – this can consist in, e.g., mutating the features, that is introducing random changes.
5. The search continues until a desired number of iterations is performed. In the end, the individual with the overall best fitness is returned.

Specific algorithms which were evaluated are listed below:

– **Artificial Bee Colony (ABC)** [10]
ABC is based on the behavior of bees in their natural habitat. Each individual is called a bee, and a current state of the feature vector is referred to as a position. There exist 3 kinds of bees:
  - Employed bees: they modify their positions by mutating single features.
  - Unemployed bees: they randomly select one of the positions of the employed bees (the better the fitness of the position, the higher the probability that this position is selected) and mutate it.
  - Scout bees: they try to occupy a new random position.
  Each new position in the ABC algorithm is retained only if it turns out to be better than the original one. Main parameters include the ratios of the number of bees of each kind in the population.

- **Differential Evolution (DE)** [22]
  DE is closely related to the Genetic Algorithm, however, it uses various strategies for crossing individuals rather than the standard crossover operation, and explicit mutation is usually omitted. These differ mostly in the number of selected individuals, sampling strategies, as well as the method for combining the features. After testing selected strategies, we have decided to opt for the one which combines 3 individuals (V_1, V_2, V_3) sampled randomly from the current population. With a given probability, each feature is replaced with a new value according the following formula: $V[i] = V_1[i] + \delta(V_2[i] - V_3[i])$, where δ is a fixed parameter.
- **Genetic Algorithm (GA)**
  GA is based on two basic operations – crossover, which takes two individuals and combines them into one by mixing their features together, and mutation, which assigns a random value to one of the features. These operations correspond directly to changes in the genetic material which happen in the real world. The parameters consist of a probability of a crossover between an individual and another randomly selected individual, and a probability that an element is mutated.
- **Harmony Search (HS)** [8]
  HS was inspired by a group of musicians who use improvisation (that is, essentially, mutations of musical notes) in order to find the optimal sound combination. During a single iteration, each harmony (individual) is improvised (mutated) and it replaces the current worst harmony if it produces a better fitness score. Each feature $V[i]$ is then either replaced with a random value, or it is replaced with another feature $V_x[i]$ from a randomly selected harmony V_x from the current generation. Moreover, if one of the existing features is selected, its value is changed (it is "pitch-adjusted") by a random value from a given interval of size $\delta(max - min)$ (in our case $min = -1$ and $max = 1$). The parameters describe the probabilities of mutations.
- **Improved Harmony Search (IHS)** [14]
  IHS is based directly on harmony search and it strives to fine-tune its parameters. According to authors, effectiveness of the original algorithm might be improved when, for each consecutive iteration, the probability that a mutation occurs increases, however, the rate of change decreases. This means that at the beginning the algorithm causes the features to take fewer longer leaps, and at the end there are more frequent but smaller changes.
- **Particle Swarm Optimization (PSO)** [5]
  PSO consists of multiple particles (individuals), which change the position in the search space. The maximum value of this change is determined by the speed of a particle in question, and it can be influenced by other particles (e.g., k of its neighbors or the best particle found so far). We have investigated a simple PSO variant, where the particles explore the search space on their own, that is they are not influenced by other particles, as well as a more complicated variant from the `cran` library [3] (we refer to it as informed PSO), which takes into account both the position of the best particle as well as k neighbors (where $k = \lfloor 1 - (1 - (1 - \frac{1}{n}))^3 \rfloor$ for the population of size n).

4 Time Series Modeling with Fuzzy Cognitive Maps

In this section we briefly present the necessary formalisms related to FCM construction in order to present a self-contained experimental study on FCM optimization.

In a nutshell, FCM is represented by its weight matrix $\mathbf{W}$ which is used to iteratively model the behavior of phenomena. On the input to the map we pass current activation values, and the map responds with an output. Ideally, map's response is as close to the actual state of phenomena (the target) as possible. The input corresponds to the i-th discrete time point, while the output corresponds to the $i+1$-th time point.

A single run of an FCM (single input-output) is described with an input activation vector $\mathbf{x}$, $\mathbf{x} = [x_1, x_2, \ldots, x_c]^T$, $x_i \in [0, 1]$. Map's response is a vector $\mathbf{y} = [y_1, y_2, \ldots, y_c]^T$, $y_i \in [0, 1]$. In order to calculate the i-th element of the output vector we apply the following formula:

$$y_i = f\Big(\sum_{j=1}^{c} w_{ij} \cdot x_j\Big) \tag{3}$$

for $i = 1, 2, \ldots, c$. f is a sigmoid function endowed with a steepness parameter $\tau > 0$:

$$f(u) = \frac{1}{1 + e^{-\tau u}} \tag{4}$$

The greater the τ, the more the shape of f resembles the unit step function. Here, $\tau = 5$ was assumed, which was based on experimental studies and on the literature review, where the majority of researchers assume the same settings [19, 20].

When the processing concerns a sequence of N activation vectors (N observations), we gather them in an activation matrix that is denoted as $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$, where $\mathbf{x}_i$ is an i-th activation vector. Consistently, the FCM responds with matrix of outputs ($\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]$), which is of size $c \times N$.

Modeling with an FCM consists of the following steps:

1. FCM design,
2. FCM optimization,
3. FCM interpretation.

The first step revolves around the process of node extraction for the map, and it is a crucial point which determines the quality of the map. A general concern is that the more nodes we have in a map, the more specific model we obtain. A corollary of this is that when we add a node to the map, we should expect that its numerical accuracy will increase. The downside of an increased map size is that it affects not only the ease of interpretation, but also a computational effort needed to optimize such a map. Let us stress that number of edges (elements in the weight matrix) grows quadratically, but an average time needed to optimize such map using any metaheuristic algorithm grows faster than quadratically.

There exists a key parameter of a time series representation for modeling with FCMs: concept's dimensionality. Dimension corresponds to the number of consecutive time points represented by each concept. For instance, if the dimensionality is equal to 2, then each concept represents a pair: current time series value ($z_i \in \mathcal{R}$, $i = 2, \ldots$) and change ($\delta z_i = z_i - z_{i-1}$). If dimensionality equals 3, then each concept represents current value (z_i), change (δz_i), and change of change ($\delta\delta z_i = \delta z_i - \delta z_{i-1} = z_i - 2z_{i-1} + z_{i-2}$). The same representation has been applied by Stach et al. [20], who published a fundamental paper for the time series modeling method analyzed in this study.

Our previous research shows that the larger the dimension, the better numerical accuracy of predictions. However, the gain diminishes as the dimensionality grows. Having in mind that FCMs are models constructed to be interpreted and applied by human beings, concept dimensionality equal to 2 or 3 is a very reasonable choice, especially since we may easily represent such a space visually.

A detailed elaboration on the issues of FCM design has been presented in our previous paper [9]. At this point let us assume that we are proceeding with a task of time series modeling and we are equipped with the following: (a) a set of c concepts and (b) a set of training and testing data sets consisting of input activations $\mathbf{X}$ and targets $\mathbf{T}$.

In this paper we do not dwell further on this topic, as the focus in on the second step: FCM optimization. The aim is to construct a weight matrix $\mathbf{W}$ of size $c \times c$ that provides the smallest possible MSE (as defined in Formula 2). FCM exploration is as described in Formula 3. The time series that we model was elevated to the level of concepts and hence the modeling procedure operates on the level of concepts. In a typical scenario (as in this paper), concepts are realized with fuzzy sets, and we predict membership degrees to the extracted concepts in each discrete time point. With such assumptions we move towards the empirical section of this article, where we employ and compare a suite of different heuristic search algorithms in order to construct FCM weight matrix.

5 Experimental Results

Let us note that the problems in question are very demanding from the computational point of view, as a single run on all 4 of our data sets requires at least several days even on a modern multithreaded machine. Still, the experiments were conducted a few times and the results were consistent with one another. In fact, the relative differences between consecutive runs were surprisingly small, namely less than a few percent, and they were often negligible.

Our implementation is mostly based on the tools provided by the DEAP library [7] for Python. Moreover, we have also implemented the Genetic Algorithm from scratch in order to rule out a possible dependency on the library, and the results turned out to be similar. We have investigated 4 data sets, two of which represent real-world data (Bicup, Rainfall), and 2 synthetic ones (Synth3, Synth10); see Appendix A for more information. As regards the parameters, we have used similar values to those suggested in the literature [8,10,14,18]; consult Appendix B for a complete list.

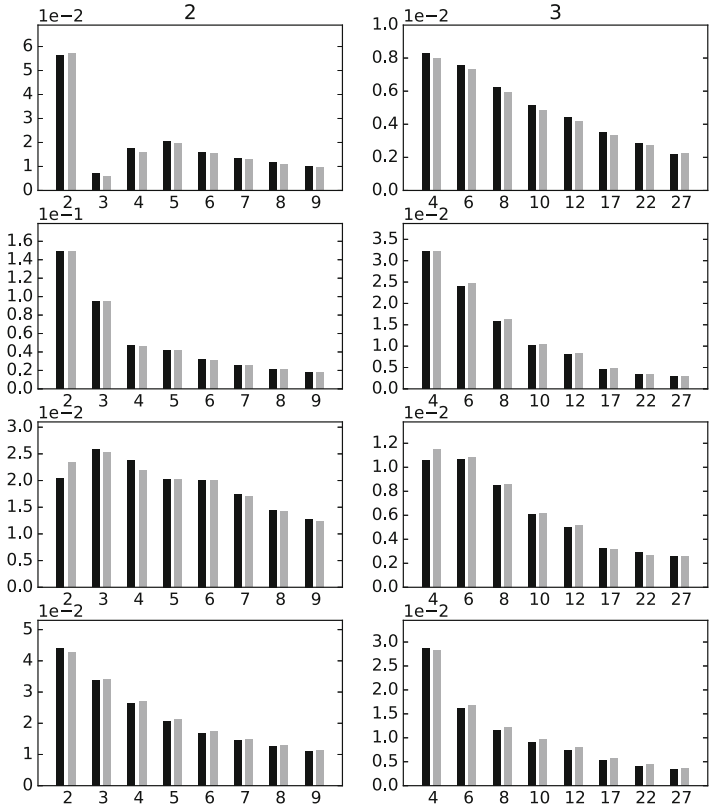

Fig. 1. Results for Differential Evolution (DE) which managed to minimize the error as the matrix size increased. Four consecutive rows correspond to each of the data sets: Synth3, Synth10, Bicup, and Rainfall (top to bottom), whereas the columns correspond to the dimensionality of data samples (consult Sect. 4 for more information). Values on the x-axis of each subplot refer to matrix sizes, and values on the y-axes describe the error (i.e. fitness in our case). Training set errors are indicated with black bars, and test set errors are indicated with grey bars.

Our results can be summarized as follows: for this particular problem class, the best effectiveness was achieved by *Differential Evolution* (DE, consult Fig. 1) and *informed PSO*. We also present the results for Improved Harmony Search (IHS) in Fig. 2. Results for other algorithms are omitted, since they demonstrate the same tendencies as the ones presented in the figures, but with higher error values (informed PSO with respect to DE and the remaining algorithms with respect to IHS).

It is worth noticing that the algorithms which managed to continue decreasing the error value as the matrix size increased were also the ones which reported overall the smallest error values. Moreover, let us note that almost all algorithms (with the exception of ABC) managed to decrease the error up to a certain point,

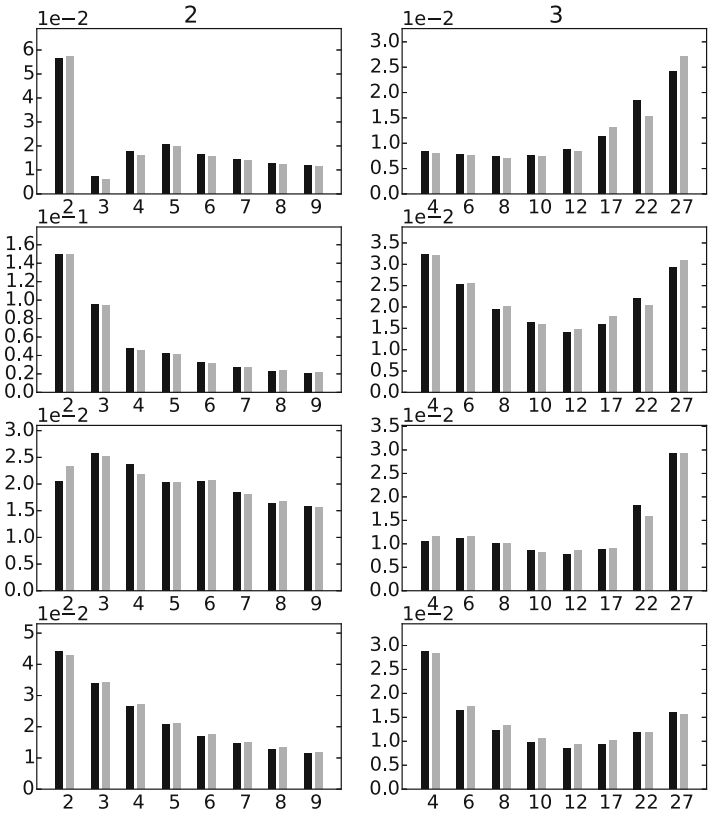

Fig. 2. Results for Improved Harmony Search (IHS) which was mostly successful only up to the matrix size of 10. Four consecutive rows correspond to each of the data sets: Synth3, Synth10, Bicup, and Rainfall (top to bottom), whereas the columns correspond to the dimensionality of data samples (consult Sect. 4 for more information). Values on the x-axis of each subplot refer to matrix sizes, and values on y-axes describe the error (i.e. fitness in our case). Training set errors are indicated with black bars, and test set errors are indicated with grey bars.

roughly matrix size 10, and in these cases the relative differences between error values were rather insignificant. This observation is consistent with the results reported by other authors, referred to in Sect. 2. Beyond this point, and especially for the largest matrices, the error value actually increased, despite a larger (i.e. more precise, at least in theory) amount of information describing the model. This leads us to the conclusion that the choice of the optimizing algorithm is not significant for smaller matrix sizes (in which case even the aforementioned L-BFGS-B approach was successful, while also being faster than population-based algorithms), however, it can be a true game changer for bigger matrices. The biggest relative differences in error values were observed for the 27-node matrix, and for some algorithms they ranged up to two orders of magnitude.

For some algorithms (e.g., simple PSO) it could be inferred from the data that a certain local minimum was being approached. This was due to the increasing appearance of −1 s and 1 s in the weight matrix, and this behavior was similar to one observed for the L-BFGS-B algorithm (as mentioned in Sect. 3). In that case, we observed a clear correlation between an increase in the error value and an increasing number of −1 s and 1 s. Still, in other cases the situation was not so clear, for instance the IHS did not manage to minimize the error value as the matrix size increased, even though the aforementioned −1 s and 1 s did not appear in the weight matrix at all.

We believe that the better performance of DE could be possibly explained by the fact that it is mostly based on the recombination of the existing population members, augmented with only a limited number of mutations. This can be partially supported by the fact that increasing the crossover probability and lowering the number of mutations for the Genetic Algorithm produced relatively better results (although still worse than DE, in particular it was unsuccessful for maps of sizes 22 and 27). Conversely, increasing the mutation probability and decreasing the crossover probability had a negative effect on the error value. Similar behavior was observed for HS, which was more effective when the mutation was almost disabled (with the probability of 1 % for each attribute). Surprisingly, this leads us to the conclusion that it is the limiting rather than the widening of the search radius which yields better results in this particular case.

6 Future Works

We would like to recognize a certain threat to validity, namely the dependence on parameter values. It is not feasible to perform exhaustive parameter tuning for this particular problem due to computational constraints. Moreover, there exist certain drawbacks associated with such tuning, explored in detail by, e.g., Eiben et al. [6]. For this reason, we plan to investigate certain parameter control methods [6] (one of the algorithms that we have evaluated, namely IHS, is an example of such a method) as future work. It would be also advantageous to extend the scope of the study onto other real-world as well as synthetic time series data samples.

Appendix A

The following data sets were used in the empirical study:

- synthetic time series (**Synth3**) based on sequence (2,6,8),
- synthetic time series (**Synth10**) based on sequence (1,5,7,3,9,9,3,7,5,1),
- real-world time series **Bicup**,
- real-world time series **Rainfall**.

Synthetic time series were constructed by replication of a base sequence so that the entire set had 3,000 elements. Then, a random distortion taken from

the normal distribution with mean 0 and standard deviation 0.7 was added to each value.

Bicup time series describes the number of passenger arrivals at a subway bus terminal. Rainfall time series contains information on daily precipitation [1,2].

Appendix B

This appendix describes parameter values for each evaluated algorithm; for detailed information regarding these parameters we refer the reader to original articles. For the description of the algorithms, consult Sect. 3. The number of iterations was set to 200 for all algorithms (a larger number was unnecessary, since in most cases the improvements in fitness values were non-existent or negligible beyond this point), and the population size was equal to 100 individuals (with the exception of informed PSO, which uses its fine-tuned, custom parameters).

- **ABC**: employed bee ratio = 0.5, abandon limit = 3.
- **DE**: $\delta = 0.25$, mutation probability = 0.5.
- **GA**: two-point crossover probability = 0.2, attribute mutation probability = 0.05, tournament size = 10, elitism rate = 0.25.
- **HS**: random value probability = 0.1, pitch adjustment probability = 0.3, $\delta = 0.01$.
- **IHS**: pitch adjustment probability $\in$ [0.1, 0.9], delta $\in$ [0.0001, 0.75].
- **PSO (simple)**: minimum speed = $min/4$, maximum speed = $max/4$, (where $min = -1$ and $max = 1$), $\phi = 1.5$.

References

1. Bicup time series. http://robjhyndman.com/tsdldata/data/bicup2006.dat. Accessed 11 Jan 2016
2. Rainfall time series. http://robjhyndman.com/tsdldata/data/rainfall.dat. Accessed 11 Jan 2016
3. Bendtsen, C.: Package 'PSO'. https://cran.r-project.org/web/packages/pso/pso.pdf. Accessed 1 Apr 2016
4. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**, 1190–1208 (1995)
5. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, New York, NY, vol. 1, pp. 39–43. (1995)
6. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. **3**(2), 124–141 (1999)
7. Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., Gagné, C.: DEAP: evolutionary algorithms made easy. J. Mach. Learn. Res. **13**, 2171–2175 (2012)
8. Geem, Z.W., Kim, J., Loganathan, G.V.: A new heuristic optimization algorithm: harmony search. Simulation **76**(2), 60–68 (2001)

9. Homenda, W., Jastrzebska, A., Pedrycz, W.: Design of fuzzy cognitive maps for modeling time series. IEEE Trans. Fuzzy Syst. **24**(1), 120–130 (2016)
10. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Global Optim. **39**(3), 459–471 (2007)
11. Kosko, B.: Fuzzy cognitive maps. Int. J. Man-Mach. Stud. **24**, 65–75 (1986)
12. Koulouriotis, D., Diakoulakis, I., Emiris, D.: Learning fuzzy cognitive maps using evolution strategies: a novel schema for modeling and simulating high-level behavior. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2001), pp. 364–371 (2001)
13. León, M., Mkrtchyan, L., Depaire, B., Ruan, D., Bello, R., Vanhoof, K.: Learning method inspired on swarm intelligence for fuzzy cognitive maps: travel behaviour modelling. In: Villa, A.E.P., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012, Part I. LNCS, vol. 7552, pp. 718–725. Springer, Heidelberg (2012)
14. Mahdavi, M., Fesanghary, M., Damangir, E.: An improved harmony search algorithm for solving optimization problems. Appl. Math. Comput. **188**(2), 1567–1579 (2007)
15. Papageorgiou, E.: Learning algorithms for fuzzy cognitive maps - a review study. IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. **42**(2), 150–163 (2012)
16. Papageorgiou, E.: Maps, Fuzzy Cognitive Maps for Applied Sciences and Engineering: From Fundamentals to Extensions and Learning Algorithms. Springer Science & Business Media, Heidelberg (2013)
17. Papageorgiou, E., Parsopoulos, K., Stylios, C., Groumpos, P., Vrahatis, M.: Fuzzy cognitive maps learning using particle swarm optimization. J. Intell. Inf. Syst. **25**(1), 95–121 (2005)
18. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. Swarm Intell. **1**(1), 33–57 (2007)
19. Stach, W., Kurgan, L., Pedrycz, W.: A survey of fuzzy cognitive map learning methods. Issues Soft Comput.: Theor. Appl., 71–84 (2005)
20. Stach, W., Kurgan, L., Pedrycz, W.: Numerical and linguistic prediction of time series. IEEE Trans. Fuzzy Syst. **16**(1), 61–72 (2008)
21. Stach, W., Kurgan, L., Pedrycz, W., Reformat, M.: Genetic learning of fuzzy cognitive maps. Fuzzy Sets Syst. **153**, 371–401 (2005)
22. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)