Automated Equation Formulation for Causal Loop Diagrams

Marc Drobek[1,2(✉)], Wasif Gilani[1], Thomas Molka[1], and Danielle Soban[2]

[1] SAP UK Ltd., Belfast, UK
{marc.drobek,wasif.gilani,thomas.molka}@sap.com
[2] Department of Mechanical and Aerospace Engineering,
Queens University Belfast, Belfast, UK
d.soban@qub.ac.uk

Abstract. The annotation of Business Dynamics models with parameters and equations, to simulate the system under study and further evaluate its simulation output, typically involves a lot of manual work. In this paper we present an approach for automated equation formulation of a given Causal Loop Diagram (CLD) and a set of associated time series with the help of neural network evolution (NEvo). NEvo enables the automated retrieval of surrogate equations for each quantity in the given CLD, hence it produces a fully annotated CLD that can be used for later simulations to predict future KPI development. In the end of the paper, we provide a detailed evaluation of NEvo on a business use-case to demonstrate its single step prediction capabilities.

Keywords: Business dynamics · Causal loop diagrams · Neural networks · Evolutionary algorithms · Big data · Predictive analyses

1 Introduction

The prediction of Key Performance Indicators (KPIs) in large enterprises is one of the major assets for business analysts and decision makers to drive company success. Traditional approaches, such as time series analyses are most common and yield quick results [1]. However, their restriction on small-dimensional dependencies limits the capability to identify actual root causes and main drivers for the particular KPI under study. Especially nowadays, in the Big-Data era, the massive amount of readily available business data raises the question, whether it can be incorporated in the prediction process to pinpoint root causes. The Business Dynamics (BD) domain tries to overcome this small-dimensionality by establishing broader scopes of causal chains and feedback loops [2,3]. In the field of BD, enterprise KPIs are modelled with the help of Causal Loop Diagrams (CLDs) and State & Flow Diagrams (SFDs) that are used to identify causality and feedback between the KPIs, as well as the material/resources flowing through the system. Once the modeller has arrived at a reasonable CLD and SFD, she needs to annotate the model with equations and parameters to create a

W. Abramowicz (Ed.): BIS 2015, LNBIP 208, pp. 38–49, 2015.
DOI: 10.1007/978-3-319-19027-3_4

final simulation model that will eventually produce prediction results. We have already summarised traditional approaches for the process of parameter estimation and equation formulation (PEEF) and concluded that these traditional concepts are cumbersome, resource-intensive and are in general only manually applicable [4]. In this paper, we will demonstrate the automated equation formulation and annotation of a given CLD based on a given historical data set. This is done by training neural networks in combination with evolutionary algorithms. The trained neural network is then annotated to the associated CLD element as a function surrogate (FS) that can replay the historical data or predict its future development. However, the main difference to traditional time series analyses is the incorporation of all target KPI dependencies given in the CLD. Rather than analysing the historical information of an isolated variable (silo-mode), we are exploiting the dependency relationships provided by a CLD and incorporate the historical information of all influencing elements to predict its future development. In the following section, we briefly describe the traditional annotation and simulation process in BD as well as provide a background of neural networks and evolutionary algorithms, and how they can be employed together for time series predictions. Afterwards, we define the research question that is derived from the traditional BD annotation approach and provide a novel approach (NEvo) to tackle this question. We then show the application of NEvo on a simple business use-case and evaluate its results. The paper ends with a conclusion.

2 Background

As has been stated earlier, it is most common in the field of BD to annotate SFDs (for instance their flows and variables) and finally simulate those to produce prediction output [2,5]. The retrieved mathematical equations are usually of a simply nature (as is intended to improve model understanding) and are rarely based on actual system data, but rather are manually derived [4]. Outside the BD domain, approaches have been developed that simplify the creation of function surrogates. For instance, Neural networks (NNs) are well known to reproduce and/or predict time series behaviour, given that the modeller has access to historical training data, knows how to train the NN efficiently, is capable of determining the correct network topology and can identify an adequate number of inputs for the NN [6,7]. Good examples for such applications are, for instance, traditional KPI time series analyses or the prediction of the stock market [8,9]. In both cases, the historical information of the one target KPI is used as training data for the NN, before the NN produces predictions of the target KPI. However, the two latter properties (network topology and NN input) are quite restrictive, since an expert-level background is required to pinpoint the correct topology (feed-forward, recurrent, partial-recurrent, Elman, etc.), the topology configuration (number of input neurons, number of hidden layers, number of hidden layer neurons, hidden layer activation functions) and input variables (number of historical input information) for the problem at hand. Dependent on the target KPI to predict, different network topologies and input variables will yield diverse prediction results. The design and parameterization

of NNs is an optimisation problem whose search space is based on the different configurations and its desired optimum is such a configuration that, given a well trained matrix, yields very accurate replay and prediction results for its target KPI. A concept that is particularly well suited to tackle such optimisation problems is that of Evolutionary Algorithms (EAs) [10,11]. Holland, Rechenberg, Goldberg and Schwefel among others, transferred the idea of the biological evolution concept to the field of computer science and mathematical optimisation, thus introducing *evolution strategy*, *evolutionary programming* and *genetic & evolutionary algorithms* [10–13]. Exploiting EAs to improve NNs in all different dimensions (input data, architecture design, weight matrix improvements, learning rule adaptation, etc.) has been extensively researched [14,15].

3 Network Evolution

The scope of this paper is to answer the question, if we can employ the massive available historical business data, which is tracked on an hourly/daily/monthly basis and resides in the companies databases, to support the BD modeller with the annotation process. Furthermore, we are challenging the traditional BD annotation process, by completely avoiding any SFDs, which are so far the pillars of any BD modelling process. We will show, that a given CLD can be automatically annotated, as long as each vertex in the CLD (each KPI) is associated with a historical time series data set. These data sets need to be available in the same time format, e.g., hourly, daily or monthly and are expected to be timely ordered. Ideally, the entire CLD creation was based on those time series data sets (see [16]). However, in the business domain, every KPI depends on a variety of variables that drive the behaviour and future development of the KPI. These causal dependencies (and their associated time series) are reflected within a CLD and are the input for NEvo. The goal is to create an FS for each target KPI which is reflected by an NN, because they are well proven to replay and/or predict time series. In our implementation, we have decided to start with a Feed-Forward network topology, because it can be easily created in an automated fashion. We are handling the challenge of parameterizing the NN topology properties as an optimisation problem that we tackle with the help of an EA. The EA is fed with a parent population that consists of multiple NN individuals. Each individual contains a genotype that represents all topology properties of one NN. Equation 1 defines such a genotype.

$$\begin{aligned} G &= \{\underbrace{g_0, .., g_i}, \ \underbrace{g_{i+1}, .., g_j}, \ \underbrace{g_{j+1}, .., g_k}\} \\ G &= \{\quad G^{IL}, \qquad G^{HL}, \qquad G^{AF} \quad\} \end{aligned} \tag{1}$$

A genotype G embodies $k+1$ genes, each reflecting one particular structural property of an NN topology. In our genotype model, the following structural NN properties are represented:

- **Input Layer:** Subsequence $G^{IL} = \{g_0, .., g_i\}$ represents the number of historical input data points for each dependent variable used to predict the target KPI.

- **Hidden Layers:** Sequence $G^{HL} = \{g_{i+1}, .., g_j\}$ contains all genes that reflect the hidden layers. Gene g_{i+1} defines the number of hidden layers in the neural network and the subsequence $\{g_{j+2}, .., g_j\}$ represents the number of neurons per hidden layer, respectively.
- **Activation Functions:** Subsequence $G^{AF} = \{g_{j+1}, .., g_k\}$ defines the activation function starting from the input layer to the first hidden layer, then from the first hidden layer to the second and so on to the last hidden layer.

The gene sequence G^{IL} reveals two facts about the NN: Firstly, the number of genes $(i+1)$ represents the number of variables that impact the target KPI, including the target KPI itself. Secondly, the length of the historical information used for each dependency: For example, the sequence $\{5, 3, 7\}$ aggregates to an overall input layer length of 15 neurons, which is split into 5 historical values of the first dependency, 3 historical values of the second dependency and 7 historical values of the actual target KPI. However, one of the main questions in creating a well suited NN is the question of how much historical data has to be incorporated when training the network? In order to answer this question we perform a *lag window computation* via *autocorrelation* [17]. Lag windows describe the length of a repeating pattern in a given stationary function that is produced by any type of periodicity. The NN is then used to train the 'pattern' that spans the lag window and supposed to apply it for prediction values it hasn't been trained on. In the business domain, it is common that all KPIs follow a particularly repeating pattern (seasonality), respectively, which is caused by many impacting factors, e.g., customer behaviour, product demand and so on. However, finding a specific pattern in a given time series can be very challenging, since multiple periodicities are usually overlapping one another. Hence, it seems reasonable to expect a diverse set of multiple lag windows for each KPI that can be fed into the creation of an input layer. Because of the large number of different NNs that undergo the evolution process, it is not of high importance which lag window is chosen for which individual, but rather, that all possible lag windows are used at some point. We have therefore decided to randomly choose lag windows, which is depicted via the $U\{X\}$ operator, that represents the uniform random selection of an element in the given set X. The creation of sequence G^{IL} is shown in Algorithm 1, which requires the target KPI, a list of lag windows for all KPIs and a CLD as input. We have shown earlier how to arrive at such CLDs based on a given time series data set [16].

The overall size of the NN mainly depends on the number of hidden layers and neurons per hidden layer stored in G^{HL}. All layers are connected in a feed-forward fashion. We have decided to restrict the overall NN hidden layer structure in its form to a sideways rotated triangle, whose base is the first hidden layer and whose top is the last hidden layer, i.e., each consecutive layer consists of equal or less neurons compared to its predecessor. This is mainly because we find such a structure to beneficially impact the overall prediction results. However, other structures, such as a square or diamond, are also promising and are subject to further research. Algorithm 2 provides the pseudo-code to produce G^{HL} with the required inputs G^{IL}, an interval for the minimum and maximum number of hidden layers and a global minimum of neurons for any hidden layer, to avoid empty hidden layers. Similarly to the random selection in a set $U\{X\}$ in

Algorithm 1. Create an input layer sequence for a target KPI $\mathbb{T}$

Procedure: *buildILSequence* : $G^{IL} = \{g_0, \ldots, g_i\}$
Require: target KPI $\mathbb{T}$; list of lag window sets Ω;
causal loop diagram CLD
$G^{IL} \leftarrow [\ ]$
$dependencies \leftarrow$ find dependencies of $\mathbb{T}$ in CLD
$dependencies \leftarrow$ add $\mathbb{T}$ to $dependencies$
$index \leftarrow 0$
for (dependency $\mathbb{D} \in dependencies$) **do**
$\Omega^{\mathbb{D}} \leftarrow$ find lag window set for $\mathbb{D}$ in Ω
$G^{IL}[index_{++}] \leftarrow U\{\Omega^{\mathbb{D}}\}$
end for
return G^{IL}

Algorithm 1, we are using $U\{[min, max]\}$ as a discrete uniform random selection of a given integral number interval $[min, max]$.

Once the NN structure has been defined, one needs to determine the activation functions for each layer transition, represented in G^{AF}. We have created a set of commonly used activation functions, which includes the following functions: Linear, Sigmoid, Elliot, Tanh, Sin, Log, Gaussian. For each transition, one of these activation functions is randomly choosen. Algorithm 3 shows this procedure. The genotyope G is then simply a union of the three subsequences created with the Algorithms 1, 2 and 3.

G is a representation of an NN and as such easily transformable into any specific NN runtime representation (a real NN). This NN is then trained with the given historical time series and a training algorithm. We have implemented three well established training algorithms for NEvo: Backpropagation, Resilient-Backpropagation and Scaled-Conjugate-Gradient [18,19]. After a particular training error has been reached, the training is stopped. In some cases, the training error

Algorithm 2. Create a hidden layer sequence for a target KPI

Procedure: *buildHLSequence* : $G^{HL} = \{g_{i+1}, \ldots, g_j\}$
Require: $G^{IL} = \{g_0, \ldots, g_i\}$; hidden layer interval $[hli_{min}, hli_{max}]$;
minimum number of neurons per layer $LOWER\_BOUND$
$G^{HL} \leftarrow [\ ]$
$hiddenLayers \leftarrow U\{[hli_{min}, hli_{max}]\}$
$maxNeurons \leftarrow \sum_{a=0}^{a=i} G^{IL}[a]$
$idealNeurons \leftarrow \lceil (maxNeurons + 1)/hiddenLayers \rceil$
$G^{HL}[0] \leftarrow hiddenLayers$
for ($layer \leftarrow 1$; $layer \leq hiddenLayers$; $layer_{++}$) **do**
$minNeurons \leftarrow max(maxNeurons - idealNeurons, LOWER\_BOUND)$
$G^{HL}[layer] \leftarrow U\{[minNeurons, maxNeurons]\}$
$maxNeurons \leftarrow minNeurons$
end for
return G^{HL}

Algorithm 3. Create an activation function sequence

Procedure: *buildAFSequence* : $G^{AF} = \{g_{j+1}, \ldots, g_k\}$
Require: G^{HL}; set of activation functions Φ
  $G^{AF} \leftarrow [\ ]$
  $activationFunctionSize \leftarrow \text{size}(G^{HL})$
  for ($index \leftarrow 0$; $index < activationFunctionSize$; $index{+}{+}$) **do**
    $G^{AF}[index] \leftarrow U\{\Phi\}$
  end for
  return G^{AF}

might be stuck in a local optima that it can't escape from. This effect can be counteracted by introducing an iteration limit, which automatically stops the training after reaching the number of iterations. Once the network has been trained, it has to be evaluated to compute a measure of its fitness.

Fitness Function. The fitness function is a measurement of the quality for an individual in a population. It determines, whether a given individual i_1 is a better solution for the optimisation problem than individual i_2. The definition of an adequate fitness function for the problem at hand is therefore critical. In our case, the fitness function incorporates the following two major metrics:

- The neural network training error f_{TE}.
- The prediction error of the individual f_{PE}.

The training error f_{TE} of an NN is a direct measure of how much the network aligns to the given input data it has been trained on. This value is a direct output of the training function itself and therefore available without any further computation cost. However, since these values are usually very small f_{TE} needs to be scaled appropriately in the fitness function. The prediction error f_{PE} is a measure for evaluating how well a trained NN performs on data it hasn't seen before. In this paper, we have decided to use 80 % of the available historical data (training data) to train the network and the remaining 20 % (prediction data) to compute the prediction error. The prediction error is computed from the deviation of the given timeseries data and the output of the NN for that time period by applying an error function. Literature provides various different error functions that can be applied, e.g., the mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE) or mean magnitude of relative error (MMRE). As we will see later in the evaluation section, we have tested NEvo with all those error functions. The fitness function is then a simple aggregation of both, the scaled network training error f_{TE} and the weighted prediction error f_{PE} (2). This function is used to evaluate each individual and compare it with other individuals in a given population, therefore creating evlutionary pressure.

$$f = w_1 * f_{TE} + w_2 * f_{PE} \tag{2}$$

Evolutionary Operators. To improve the overall fitness of a given population according to the above defined fitness function f, we have implemented several evolutionary operators. Each generation is started with a uniform random

selection on a given parent population $Pop = \{G_{P_0}, G_{P_1}, \ldots, G_{P_n}\}$. The selection creates pairs of parent individuals that are then recombined with a uniform crossover at gene g_j (number of hidden layers) in order to create two child individuals. Equation (3) shows the signature and definition of this recombination ρ. Hence, the first created child individual contains the input layer subsequence from the first parent and the hidden layer and activation function subsequence from the second parent. The same holds vice versa for the second individual.

$$\begin{aligned} \rho: \ & \langle G_{P_i}, G_{P_j} \rangle \rightarrow \langle G_{C_k}, G_{C_l} \rangle \\ \rho(\langle G_{P_i}, G_{P_j} \rangle) &= \rho(\langle \{G_{P_i}^{IL}, G_{P_i}^{HL}, G_{P_i}^{AF}\}, \{G_{P_j}^{IL}, G_{P_j}^{HL}, G_{P_j}^{AF}\} \rangle) \\ &= \langle \{G_{P_i}^{IL}, G_{P_j}^{HL}, G_{P_j}^{AF}\}, \{G_{P_j}^{IL}, G_{P_i}^{HL}, G_{P_i}^{AF}\} \rangle \\ &= \langle G_{C_k}, G_{C_l} \rangle \end{aligned} \tag{3}$$

This procedure is repeated until a sufficiently dense child population has been created. Afterwards, a subset population is selected based on a mutation rate. All child individuals in this subset undergo a mutation operation, which alters one gene or a small gene subsequence. We have implemented four different mutations:

- Input neuron mutation: One gene $g_n \in G^{IL}$ is randomly chosen and replaced with a new lag size from the precomputed pool of different lag sizes for the associated input variable.
- Neuron mutation: The number of neurons in one particular hidden layer $g_n \in G^{HL}$ is reset with a new random value in a given interval.
- Hidden layer mutation: The entire hidden layer gene sequence G^{HL} is rebuilt.
- Activation function mutation: One activation function gene $g_n \in G^{AF}$ is replaced with a randomly chosen element from the given pool of activation functions.

The actual mutation, which is applied to a given individual, is randomly selected from the above defined mutation set. Finally, the recombined and mutated child population is filtered with an environment selection, which in our case is a best selection based on the fitness of each child. This selection reduces the size of the child population to the size of the parent population for the next generation. Since the selection is based on the fitness of the child individuals, it also helps to achieve a convergence of the population fitness towards an optimal solution.

Configuration Parameters. NEvo is a complex FS computation algorithm, and as such, subject to a configuration that guides the algorithm and yield different FS results. The following parameters are required for each NEvo run:

- Generations: The number of generations the evolutionary cycle is executed to improve the overall fitness.
- Number of parent individuals: The number of parent individuals each generation is started with.
- Number of child individuals: The number of children created from the parent population with the recombination operator.
- Mutation rate: The percentage of child individuals in a given child population that are subject to the mutation operators.

- Prediction error function: The error function to compute the deviation of the prediction output and the orginal time series. As explained earlier, multiple functions are available for f_{PE}, e.g., RMSE, MSE, MAE, MMRE.
- Training function: The neural network training function, e.g., Backpropagation, Resilient-Backpropagation, Quick-Propagation or Scaled-Conjugate-Gradient.

4 Usecase

The evaluation of NEvo is based on the business use-case of the company Akron Heating (AH), which operates in the highly competitive retail sector [20]. AH's business model is based on selling goods via an online store and supported by various Business Processes (BPs), e.g., the Order process, Consignment fill-up process, Return-Item process and so on. For managing and controlling their business, AH employs various software solutions (ERP, CRM and HRM). These solutions are thoroughly tracking and storing the generated operational and business data, such as event data and aggregated high level data (revenue, sales, market share, cost of goods sold, cashflow and so forth). We have shown in our previous research work how one could automatically create CLDs based on this data, to evaluate causality between these variables and find root causes for bottlenecks [21]. Such automatically created CLDs are created with the help of Granger-Causality and domain specific ontologies, thus aiming to visualize causation rather than correlation [22]. In this paper, we are using an extended CLD that incorporates more strategic KPIs than created in [21]. It has been automatically generated with the same algorithm. An image section of the CLD is shown in Fig. 1. This image section does not contain loops, since it has been simplified from a larger more complex view. However, the entire complex CLD with all loops is the basis for the evaluation and as such, input for NEvo (including all loops). The image section of the CLD shows 18 high-level KPIs, for instance, profit, sales, expenses, number of orders, etc., as well as their causal relationships. Every KPI in the CLD has to be annotated with an accurate FS computed via NEvo. A created FS is 'accurate' if its respective fitness computed via f is minimal. A minimised fitness is a strong indicator, that the FS is capable of replaying the historical time series data, as well as, predict its future development. As we have explained earlier, the usage of NEvo requires time series data for each KPI and its dependencies. In the AH use-case, such time series data is provided for a range of 10 years (2000–2010) and was monitored on a monthly basis. Each time series therefore provides 120 entries, of which 90 entries are used for NEvo (training and prediction data) and 30 entries are used to evaluate NEvo's prediction capabilites (evaluation data). The 90 entries used for NEvo are split in the above stated 80/20 fashion, which means that 72 data points are used as training data and 18 points are used as prediction data. To create a reference frame for later comparison, we have set a default parameter pool for each NEvo run, which looks as follows: 200 generations, 25 parent individuals, 125 child individuals and a mutation rate of 0.7. These parameters have

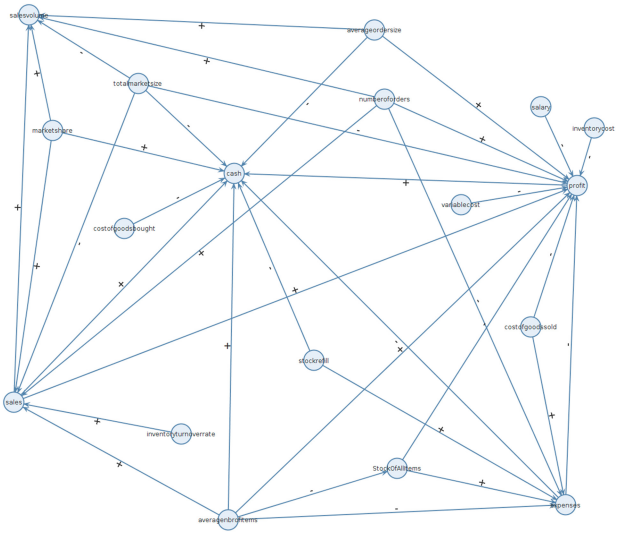
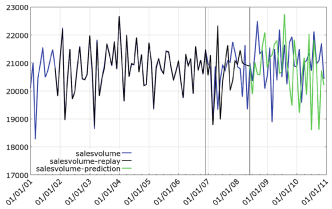


Fig. 1. An image section of a greater CLD that represents interesting dependencies and KPIs in the AH use-case.

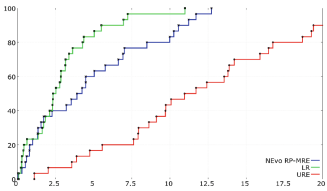
been identified to work well throughout several simulation runs. The prediction error function for f_{PE} and the NN training function for f_{TE} are subject to the specific run.

5 Results and Evaluation

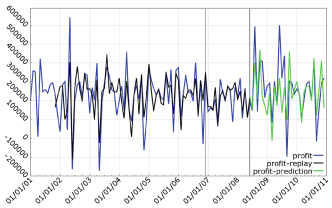
The evaluation of NEvo is a comparison of its prediction capabilities with both, a linear regression (*LR*) and a uniform random estimator (*URE*). *LR* produces a linear mean function over the given time series, whereas *URE* is a uniform random distribution in the interval [*min*, *max*], with *min* and *max* being the minimum and maximum value of the given time series. We compare the quality of the results for NEvo, *LR* and *URE* by using a *cumulative distribution function* (*CDF*) of the relative error. It shows how many y percent of the prediction results are within a relative error range of x percent of the original time series. These results are produced on the evaluation data set, which consists of 30 entries. The NEvo prediction results are created as single step predictions with the evaluation data set as input (rather than previously predicted values). However, none of the evaluation data points have been used to train the NN. Figure 2 shows results for the replay and prediction of the KPIs salesvolume and profit. As we can see in the first picture 2(a), the winning NN nearly reproduces the original training data with an accuracy of a 100 %, but continues less accurate on the remaining prediction data (which was not fed into the NN, but rather used to internally evaluate the NN with the EA). Investigating the evaluation output (green line) clearly shows that only few trends are recognized by NEvo. This observation can be confirmed by analyzing the CDF graph 2(b). The simple *LR* algorithm



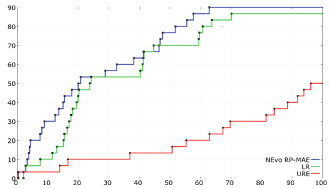
(a)Salesvolume (MRE, RP): Original (blue), NN replay (black) and predictions (green)



(b)CDF for Salesvolume with NEvo (blue), LR (green) and URE (red)



(c)Profit (MAE, RP): Original (blue), NN replay (black) and predictions (green)

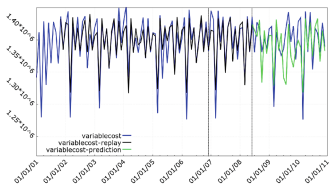


(d)CDF for Profit with NEvo (blue), LR (green) and URE (red)

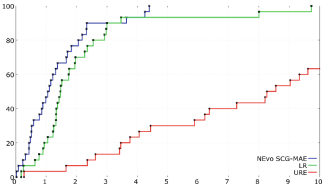
Fig. 2. NEvo results: The left side shows the original time series (blue), the training replay of the winning NN (black) and the predictions of this NN (green). The right side shows the CDF for NEvo (blue), *LR* (green) and *URE* (red) (Color figure online).

outperforms NEvo in this scenario, since 80 % of all prediction results produced by *LR* are better than 5 % of relative error of the original salesvolume time series. This is due to the small overall fluctuation of the salesvolume time series (all values are between a 15 % relative error). However, if we focus more on the results of a KPI with a higher fluctuation, e.g., profit, evidence points to more reliable results predicted with NEvo (see Fig. 2(c) and (d)). The graph 2(c) on the left shows a couple of prediction intervals that are picked up correctly by NEvo, e.g., 03/2009 until 06/2009 or 02/2010 until 06/2010. This is also reflected in the respective *CDF* graph 2(d) on the right, which shows a better performance of NEvo compared to *LR* and *URE*. These sort of prediction results can be found along all given KPIs, if all previously explained error functions and training functions are included (see Fig. 3(a) and (b) for another example of the monthly variable costs of Akron Heating).

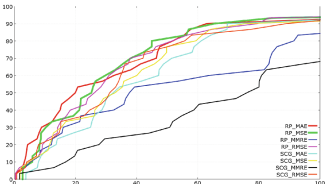
An interesting question at this point is that of the 'best' NEvo configuration to guarantee results that are at least of the quality as shown earlier in the profit example. As we can see in the CDF graphs 3(c) and (d), such a configuration might not exist. Both figures are clearly showing that different configurations for different KPIs yield the respective 'best' result. For instance, in case of the profit KPI, *RP_MAE* and *RP_MSE* produce the best results, whereas in case of expenses *RP_RMSE* and *SCG_MSE* are better. Nevertheless, the CDF itself can be employed to automatically determine the 'best' winning NN accurately.



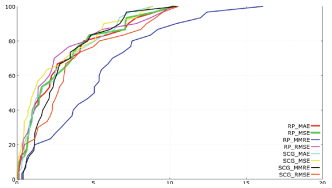
(a)Variable costs (MAE, SCG): Original (blue), NN replay (black) and predictions (green)



(b)CDF for variable costs with NEvo (blue), LR (green) and URE (red)



(c)CDF of profit for all 8 winning NEvo NNs from all different configurations



(d)CDF of expenses for all 8 winning NEvo NNs from all different configurations

Fig. 3. The prediction output for the variable cost and its CDF (3(a) and 3(b)). 3(c) and 3(d) show a comparison of all CDFs for the profit and expenses KPI (Color figure online).

6 Conclusion and Future Work

In this paper we have demonstrated an EA-guided NN function retrieval algorithm that is capable of annotating all variables in a given CLD, thus preparing the CLD itself for a later multi step prediction (simulation). Our future research will be focussed on evaluating if the simulation of CLDs annotated with NEvo can accurately predict the future development of the modelled KPIs for longer time frames (multi step). We have shown that some of the NEvo results are not capable of matching the quality of a simple *LR* algorithm that produces a mean function over the given time series. Some of these results can be explained with a timely shift in the prediction results that has a strong impact on the CDF. However, this clearly shows room for improvement. As stated in the "Network Evolution" section, the NN topology is currently limited to simple feed-forward networks. One possible improvement is the implementation of different NN topologies, such as partial-recurrent networks that are capable of maintaining a state. We have also explained earlier that different hidden layer structures are under investigation (square and diamond structures). Furthermore, after analysing the CDF results, one could think of an extension of the EA fitness function by enriching it with a CDF threshold: Whenever y % of the prediction results are above a relative error of x %, an additional penalty is applied. This would surely favour all those individuals that are below such a threshold. It remains to be investigated, whether such improvements yield more accurate prediction results.

References

1. Brockwell, P.J., Davis, R.A.: Time Series: Theory and Methods, 2nd edn. Springer, Heidelberg (2006)
2. Forrester, J.W.: Industrial Dynamics. Currently Available from Pegasus Communications. MIT Press, Cambridge (1961)
3. Sterman, J.D.: Business Dynamics: Systems Thinking and Modeling for a Complex World. McGraw-Hill, New York (2000)
4. Drobek, M., Gilani, W., Soban, D.: Parameter estimation and equation formulation in business dynamics. In: 3rd International Symposium on Business Modeling and Software Design. ScitePress, Noordwijkerhout (2013)
5. Burns, J.R.: Converting signed digraphs to forrester schematics and converting forrester schematics to differential equations. IEEE Trans. Syst. Man Cybern. B Cybern. **10**, 695–707 (1977)
6. Fausett, L.V.: Fundamentals of Neural Networks: Architectures, Algorithms, and Application, 1st edn. Pearson, London (1993)
7. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
8. McNelis, P.D.: Neural Networks in Finance: Gaining Predictive Edge in the Market. Academic Press, Orlando (2005)
9. Tudor, N.L.: Intelligent system for time series prediction in stock exchange markets. In: Abramowicz, W., Kokkinaki, A. (eds.) BIS 2014. LNBIP, vol. 176, pp. 122–133. Springer, Heidelberg (2014)
10. Holland, J.H.: Genetic algorithms and the optimal allocations of trials. SIAM J. Comput. **2**, 88–105 (1973)
11. Goldberg, D.E.: Genetic Algorithms in Search. Optimization and Machine Learning. Addison-Wesley Longman Publishing Inc., Boston (1989)
12. Rechenberg, I.: Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Ph.D. thesis (1971)
13. Schwefel, H.P.: Numerische Optimierung von Computer-Modellen, vol. 26th. Birkhaeuser, Basel (1977)
14. Schaffer, J., Whitley, D., Eshelman, L.: Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: COGANN 1992, pp. 1–37. IEEE, Baltimore, MD (1992)
15. Yao, X.: Evolving artificial neural networks. Proc. IEEE **87**, 1423–1447 (1999)
16. Drobek, M., Gilani, W., Soban, D.: A data driven and tool supported CLD creation approach. In: The 32nd International Conference of the System Dynamics Society, pp. 1–20, Delft (2014)
17. Wei, W.W.S.: Time Series Analysis: Univariate and Multivariate Methods, 2nd edn. Pearson, London (2005)
18. Riedmiller, M., Braun, H.: RPROP - a fast adaptive learning algorithm. In: Proceedings of ISCIS VII (1992)
19. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Nat. Bur. Stan. **49**(6), 409–436 (1952)
20. Fritzsche, M., Picht, M., Gilani, W., Spence, I., Brown, J., Kilpatrick, P.: Extending BPM environments of your choice with performance related decision support. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 97–112. Springer, Heidelberg (2009)
21. Drobek, M., Gilani, W., Redlich, D., Molka, T., Soban, D.: On advanced business simulations - converging operational and strategic levels. In: 4th International Symposium on Business Modeling and Software Design. ScitePress, Luxembourg (2014)
22. Granger, C.W.J.: Investigating causal relations by econometric models and cross-spectral methods. Econometrica **37**, 424–438 (1969)