Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 917–927
Neural Networks
www.elsevier.com/locate/neunet

Modeling of gene regulatory networks with hybrid differential evolution and particle swarm optimization

Rui Xu[a,*], Ganesh K. Venayagamoorthy[b], Donald C. Wunsch II[a,1]

[a] Applied Computational Intelligence Laboratory, Department of Electrical and Computer Engineering, University of Missouri – Rolla, MO 65409, USA
[b] Real-Time Power and Intelligent Systems Laboratory, Department of Electrical and Computer Engineering, University of Missouri – Rolla, MO 65409, USA

Received 24 October 2006; received in revised form 12 July 2007; accepted 12 July 2007

Abstract

In the last decade, recurrent neural networks (RNNs) have attracted more efforts in inferring genetic regulatory networks (GRNs), using time series gene expression data from microarray experiments. This is critically important for revealing fundamental cellular processes, investigating gene functions, and understanding their relations. However, RNNs are well known for training difficulty. Traditional gradient descent-based methods are easily stuck in local minima and the computation of the derivatives is also not always possible. Here, the performance of three evolutionary–swarm computation technology-based methods, known as differential evolution (DE), particle swarm optimization (PSO), and the hybrid of DE and PSO (DEPSO), in training RNNs is investigated. Furthermore, the gene networks are reconstructed via the identification of the gene interactions, which are explained through corresponding connection weight matrices. The experimental results on two data sets studied in this paper demonstrate that the DEPSO algorithm performs better in RNN training. Also, the RNN-based model can provide meaningful insight in capturing the nonlinear dynamics of genetic networks and revealing genetic regulatory interactions.

Keywords: Differential evolution; Particle swarm optimization; Genetic regulatory networks; Recurrent neural networks; Time series gene expression data

1. Introduction

A major advancement in genetic experimental technologies, DNA microarray analysis provides a systematic method to characterize genetic functions, discriminate disease types, and test drug effects (McLachlan, Do, & Ambroise, 2004; Xu & Wunsch II, 2005). Inference of genetic regulatory networks from time series gene expression data has attracted attention, due to its importance in revealing fundamental cellular processes, investigating functions of genes, and understanding complex relations and interactions between genes (De Jong, 2002; D'haeseleer, Liang, & Somogyi, 2000; Xu, Hu, & Wunsch II, 2004a). In the context of the data generated from microarray technologies, i.e. transcriptional regulation of protein-coding genes, a genetic regulatory network consists of a set of DNA, RNA, proteins, and other molecules, and it describes regulatory mechanisms among these components. Genetic information that determines structures, functions, and properties of living cells is stored in DNA, whose coding regions, known as genes, encode proteins. According to the central dogma of molecular biology, genes are first transcribed into mRNA molecules, which are then translated to proteins. During the sophisticated process, the mechanism of gene regulation, which can occur at any step along the cellular information flow, determines which subset of genes is expressed, to what level, and in response to what conditions of the cellular environment (Latchman, 2005; Perdew, Vanden Heuvel, & Peters, 2006). For example, genes encoding digestive enzymes are expressed in the gut but not the skin, and their level of expression increases in the presence of food. One of the most common and well-studied levels on gene expression regulation is the initiation of transcription. In this context,

* Corresponding address: 116 Emerson Electric Co. Hall, 1870 Miner Circle, Rolla, MO 65409, USA. Tel.: +1 573 341 6811; fax: +1 573 341 4532.
E-mail address: rxu@umr.edu (R. Xu).

[1] Portions of this paper were previously presented in conference format at the Third International Symposium on Neural Networks (Xu, R., Venayagamoorthy, G., & Wunsch, D., II (2006). A study of particle swarm optimization in gene regulatory networks inference. In J. Wang, Z. Yi, J. Zurada, B. Lu, & H. Yin (Eds.), Advances in neural networks — ISNN 2006: Third international symposium on neural networks (pp. 648–653). Berlin, Heidelberg: Springer).

0893-6080/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.07.002

the control is achieved through the actions of regulatory proteins, called transcription factors, which activate or inhibit the transcription rate of certain genes by binding to their transcriptional regulatory sites. Therefore, the transcription of a specific gene, or the control of its expression, can be regarded as a combinatorial effect of a set of other genes. In other words, when gene g_1 is said to regulate gene g_2, it actually means that the transcription factors encoded by g_1, translated from its mRNA products, control the transcription rate of g_2. Other activities, such as RNA splicing and post-translational modification of proteins, are also constituents of the entire regulatory system. However, due to limited data availability, the focus can only be put on the transcription (mRNA) level at the current stage, instead of the protein level. Despite such restrictions, simplification has the advantage of measurements over a very large scale, and the use of mRNA expression-based microarrays has led to many interesting and important results (Bar-Joseph, 2004; De Jong, 2002; D'haeseleer et al., 2000).

Classical molecular methods such as Northern blotting, reporter genes, and DNA footprinting, have provided great insight into the regulatory relationships between a pair of genes or among a few pre-selected genes, which is far from sufficient for exploring their complicated regulatory mechanisms. DNA microarray technologies provide an effective and efficient way to measure the gene expression levels of up to tens of thousands of genes simultaneously under many different conditions, which makes it possible to investigate gene relations and interactions when taking the entire genome into consideration (De Jong, 2002; D'haeseleer et al., 2000). Currently, major microarray technologies belong to two categories, based on the nature of the attached DNA and the manufacturing technologies (Eisen & Brown, 1999; Lipshutz, Fodor, Gingeras, & Lockhart, 1999; McLachlan et al., 2004; Schena, Shalon, Davis, & Brown, 1995). For instance, in the context of cDNA technologies, cDNA clones or expressed sequence tag (EST) (EST is a unique short subsequence for a certain gene) libraries, with length varying from several hundred to a few thousand bases are robotically spotted and placed on a solid substrate (Eisen & Brown, 1999; McLachlan et al., 2004). The robot spotter touches and then places the droplets of the solution containing the probes synthesized in advance on the array. Fluorescently labeled cDNA, obtained from RNA samples of interest through the process of reverse transcription, is then hybridized with the array. A reference sample with a different fluorescent label is also required for the purpose of comparison. Generally, the test sample and reference sample are labeled with Cyanine 5 (Cy5) in red and Cyanine 3 (Cy3) in green dyes, respectively. Therefore, the red spots in the hybridized array indicate higher expression of the genes in the test sample, and the green spots indicate the higher expression in the reference sample. If the gene transcripts are similar in both samples, the spots are displayed in yellow. The fluorescence of each dye after the genes are washed off can be measured using image analysis techniques. The final data are the ratios of the intensity of test sample and the reference sample and reflect relative levels of gene expression. On the other hand, for the high-density oligonucleotide microarray, oligonucleotides, containing 20–70 bases, are fixed on a chip through techniques like photolithography and solid-phase DNA synthesis (Lipshutz et al., 1999). In this case, absolute gene expression levels are obtained. Gene expression data can be used for investigating the activities of a single gene, clustering genes of similar functions, providing evidence for cancer diagnosis, and reconstructing genetic regulatory networks, to name a few (Baldi & Long, 2001).

Several computational models have been proposed to infer regulatory networks through the analysis of gene expression data (De Jong, 2002; D'haeseleer et al., 2000; Xu et al., 2004a). Boolean networks are binary models, which consider that a gene has only two states: 1 for active and 0 for inactive (Hallinan & Jackway, 2005; Kauffman, 1993; Liang, Fuhrman, & Somogyi, 1998; Shmulevich, Dougherty, & Zhang, 2002). The effect of other genes on the state change of a given gene is described through a Boolean function. Although Boolean networks make it possible to explore the dynamics of a genetic regulatory system, they ignore the effect of genes at intermediate levels and inevitably, cause information loss during the discretization process. Bayesian networks are graph models that estimate complicated multivariate joint probability distributions through local probabilities (Friedman, Linial, Nachman, & Pe'er, 2000; Smith, Jarvis, & Hartemink, 2002; Werhli, Grzegorczyk, & Husmeier, 2006). Under this framework, a genetic regulatory network is described as a directed acyclic graph that includes a set of vertices and edges. The vertices are related to random variables and are regarded as genes or other components while the edges capture the conditional dependence relation and represent the interactions between genes. Bayesian networks are effective in dealing with noise, incompleteness, and stochastic aspects of gene expression data. However, they do not consider dynamical aspects of gene regulation and leave temporal information unhandled. Recently, dynamic Bayesian networks (DBN) have attracted more attention (Husmeier, 2003; Murphy & Mian, 1999; Perrin et al., 2003; Tamada et al., 2003; Yu, Smith, Wang, Hartemink, & Jarvis, 2004). DBN can model behaviors emerging temporally and can effectively handle problems like hidden variables, prior knowledge, and missing data. The disadvantage of DBN is that they may not scale very well to large-scale data sets. For linear additive regulation models (D'haeseleer, 2000; D'haeseleer, Wen, Fuhrman, & Somogyi, 1999; van Someren, Wessels, & Reinders, 2000, 2001), the expression level of a gene at a certain time point can be calculated by the weighted sum of the expression levels of all genes in the network at a previous time point. Although linear additive regulation can reveal certain linear relations in the regulatory systems, it lacks the capability to capture the nonlinear dynamics between gene regulations.

Considering the limitations of the above methods, in the paper, genetic regulatory networks are inferred from time series gene expression in the framework of recurrent neural networks (Kolen & Kremer, 2001). In using RNNs for genetic network inference, their ability to interpret complex temporal behavior becomes particularly important, since that is an inherent characteristic of times series gene expression data and makes

them different from static expression data (Bar-Joseph, 2004). Generalized RNNs can be considered as signal processing units forming a global regulatory network. The recurrent structure of RNNs effectively reflects the existence of feedback, which is essential for gene regulatory systems. RNN models have already been proposed and used for genetic regulatory networks inference (D'haeseleer, 2000; Mjolsness, Mann, Castaño, & Wold, 2000; Vohradský, 2001; Weaver, Workman, & Stormo, 1999; Xu, Hu, & Wunsch II, 2004b). For instance, D'haeseleer (2000) discussed a realization of RNNs in modeling gene networks using synthetic data. Vohradský (2001) investigated the dynamic behaviors of a 3-gene network in the framework of RNNs.

A commonly known problem in the neural network community is the training difficulty of RNNs (Jaeger, 2002). Typically, back-propagation through time (BPTT) (Werbos, 1990) and evolutionary algorithms (EAs) (Fogel, 1994; Yao, 1999) are used for RNN training. BPTT may be derived by unfolding the temporal operation of the network into a layered feedforward network, the topology of which grows by one layer at every time step (Haykin, 1999). The derivatives of a cost function with respect to the individual weight of the network are calculated, which can then be used to do gradient descent on the weights, updating them in the direction that minimizes the error. However, the requirement of the computation of the derivatives limits its application because the error functions are not always differentiable. More seriously, the deep architecture effect causes the gradient descent computation more and more problematic. As a consequence, the procedure is easy to get stuck to some local minima.

EAs are inspired by the process and principles of natural evolution and refer to a class of population-based stochastic optimization search algorithms (Fogel, 1994). The major technologies of EAs include genetic algorithms (GAs), genetic programming (GPs), evolution strategies (ESs), and evolutionary programming (EP), each of which focuses on a different facet of natural evolution (Fogel, 1994; Yao, 1999). Evolutionary algorithms have no requirement for the gradient information and the objective or prior information can be explicitly incorporated into the learning. Particularly, gene regulatory network inference with GAs has already been reported by Wahde and Hertz (2000, 2001) and Keedwell and Narayanan (2005). As an example, Wahde and Hertz (2001) used GA to identify the network parameters in inferring gene regulatory interactions during the development of the central nervous system of rats. These parameters are encoded based on a decimal scheme and the fitness function is defined based on the errors between the measurements and the estimated values.

This paper presents the application of a hybrid evolutionary–swarm algorithm called differential evolution particle swarm optimization (DEPSO) (Zhang & Xie, 2003) for training RNNs. Its performance is compared with differential evolution (Storn & Price, 1997) and particle swarm optimization (Kennedy, Eberhart, & Shi, 2001), and the experimental results demonstrate that the hybrid differential evolution particle swarm optimization algorithm can achieve superior performance in RNN training. DEPSO achieves convergence all the time and is faster than differential evolution (DE) or particle swarm optimization (PSO) individually. Furthermore, the paper demonstrates that the RNN-based model can provide meaningful insight in capturing the nonlinear dynamics of genes networks and revealing genes regulatory interactions, explained through corresponding connection weight matrices of RNNs. Such information can then be used to reconstruct genetic regulatory networks and help understand the regulatory pathways.

The paper is organized as follows. Section 2 describes the RNN model for regulatory network inference. In Section 3, the three evolutionary/swarm computational technologies, used for RNNs training, are described and discussed. Section 4 introduces the artificial data and the real data — the SOS DNA repair system used in the analysis. Section 5 illustrates experimental results to both data sets. The paper is concluded in Section 6.

2. Recurrent neural networks

For a continuous time system, the genetic regulation model can be represented through a recurrent neural network formulation (D'haeseleer, 2000; Mjolsness et al., 2000; van Someren, Wessels, & Reinders, 2001; Weaver et al., 1999),

$$\tau_i \frac{\mathrm{d}e_i}{\mathrm{d}t} = f\left(\sum_{j=1}^{N} w_{ij} e_j + \sum_{k=1}^{K} v_{ik} u_k + \beta_i\right) - \lambda_i e_i, \tag{1}$$

where e_i is the gene expression level for the ith gene ($1 \le i \le N$, N is the number of genes in the system), $f(\cdot)$ is a nonlinear function (usually, a sigmoid function is used $f(z) = 1/(1+e^{-z})$), w_{ij} represents the effect of the jth gene on the ith gene ($1 \le i, j \le N$), u_k is the kth ($1 \le k \le K$, K is the number of external variables) external variable, which could represent the externally added chemicals, nutrients, or other exogenous inputs, v_{ik} represents the effect of the kth external variable on the ith gene, τ is the time constant, β is the bias term, and λ is the decay rate parameter. A negative value of w_{ij} represents the inhibition of the jth gene on the ith gene, while a positive value indicates the activation controls. When w_{ij} is zero, there is no influence of the jth gene on the expression change of the ith gene. The effects of other factors can be added into the formula based on the specific situation. Note that this model is a natural extension of the linear additive model in D'haeseleer et al. (1999) and van Someren et al. (2000), in order to explicitly take into account the nonlinear dynamics of the networks. Several applications based on the model in Eq. (1) have been reported in the literature (D'haeseleer, 2000; Mjolsness et al., 2000; van Someren, Wessels, & Reinders, 2001; Weaver et al., 1999).

This model can also be described in a discrete form for computational convenience, since measure is only made at certain time points:

$$\frac{e_i(t+\Delta t) - e_i(t)}{\Delta t} = \frac{1}{\tau_i}\left(f\left(\sum_{j=1}^{N} w_{ij} e_j(t) + \sum_{k=1}^{K} v_{ik} u_k(t) + \beta_i\right) - \lambda_i e_i(t)\right), \tag{2}$$

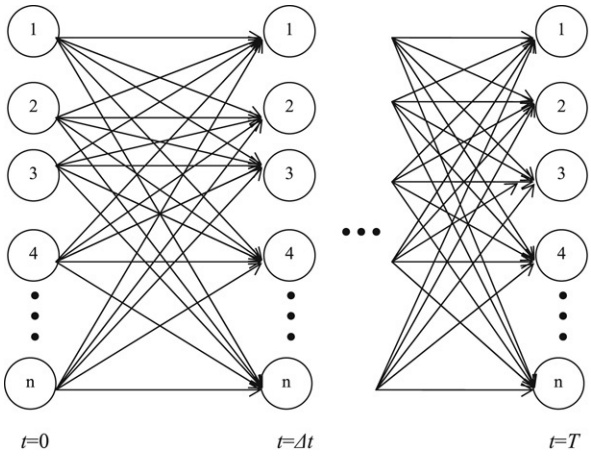

Fig. 1. The description of a genetic network through a recurrent neural network model. This network is unfolded in time from $t = 0$ to T with an interval Δt. Here, the regulatory network is shown in a fully connected form, although, in practice, the network is usually sparsely connected.

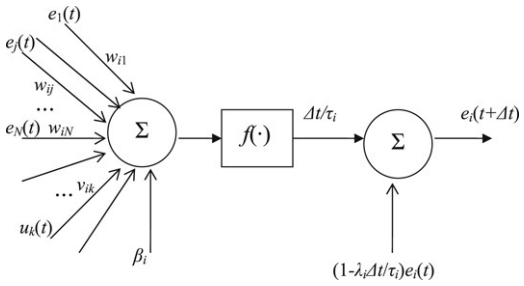

Fig. 2. A node (neuron) in the recurrent neural network model, based on Eq. (3).

or,

$$e_i(t+\Delta t) = \frac{\Delta t}{\tau_i} f\left(\sum_{j=1}^{N} w_{ij} e_j(t) + \sum_{k=1}^{K} v_{ik} u_k(t) + \beta_i\right) + \left(1 - \frac{\lambda_i \Delta t}{\tau_i}\right) e_i(t). \quad (3)$$

Fig. 1 depicts a recurrent neural network, which is unfolded in time from $t = 0$ to T with an interval Δt, for modeling a genetic network. Here, each node corresponds to a gene, and a connection between two nodes defines their interaction. The weight values can be either positive, negative, or zero, as mentioned above. Fig. 2 illustrates a node in the recurrent neural network, which realizes Eq. (3).

The role that external variables play (affectors of expression that originate and act downstream from transcription) will vary from system to system. However, ignoring these variables at this stage will not invalidate the inferred regulatory network since the networks being studied, by definition, are largely controlled via transcription. From the following section, it can be seen that the inclusion of these exogenous inputs does not affect the derivation of the learning algorithm. It is also assumed that the decay rate parameter λ is 1 for computational simplicity. The final model processed in the paper is represented as

$$e_i(t+\Delta t) = \frac{\Delta t}{\tau_i} \times f\left(\sum_{j=1}^{N} w_{ij} e_j(t) + \beta_i\right) + \left(1 - \frac{\Delta t}{\tau_i}\right) e_i(t). \quad (4)$$

3. Training algorithms

Although the RNN-based model has already been reported in the literature, the difficulty of RNN training limits its further application for gene network inference, as aforementioned. Here, three different evolutionary/swarm computational technology-based training approaches, i.e., DE, PSO, and DEPSO, are used to determine the unknown network parameters.

3.1. Particle swarm optimization

PSO is a population-based search strategy, consisting of a swarm of particles, each of which represents a candidate solution (delValle, Venayagamoorthy, Mohagheghi, Hernandez, & Harley, in press; Eberhart & Shi, 2001; Kennedy et al., 2001). Each particle i with a position represented as $\boldsymbol{x}_i$ moves in the multidimensional problem space with a corresponding velocity $\boldsymbol{v}_i$. The basic idea of PSO is that each particle randomly searches through the problem space by updating itself with its own memory and the social information gathered from other particles. These components are represented in terms of two best locations during the evolution process: one is the particle's own previous best position, recorded as vector $\boldsymbol{p}_i$, according to the calculated fitness value, and the other is the best position in the whole swarm, represented as $\boldsymbol{p}_g$. Also, $\boldsymbol{p}_g$ can be replaced with a local best solution obtained within a certain local topological neighborhood. In the application of PSO in RNNs training, the fitness function is defined to measure the deviation of network output $e(t)$ from the real measurement (target) $d(t)$, written as

$$\mathrm{Fit}(\boldsymbol{x}_i) = \frac{1}{TN} \sum_{t=0}^{T} \sum_{i=1}^{N} (e_i(t) - d_i(t))^2. \quad (5)$$

More elaborate error terms can be added easily based on the specific requirement of the problem at hand.

Fig. 3 depicts the vector representation of the PSO search space. The corresponding canonical PSO velocity and position equations at iteration t are written as,

$$\boldsymbol{v}_i(t) = w \times \boldsymbol{v}_i(t-1) + c_1 \times \phi_1 \times (\boldsymbol{p}_i - \boldsymbol{x}_i(t-1)) + c_2 \times \phi_2 \times (\boldsymbol{p}_g - \boldsymbol{x}_i(t-1)), \quad (6)$$

$$\boldsymbol{x}_i(t) = \boldsymbol{x}_i(t-1) + \boldsymbol{v}_i(t), \quad (7)$$

where w is the inertia weight, c_1 and c_2 are the cognitive and social acceleration constants respectively, and ϕ_1 and ϕ_2 are uniform random functions in the range of [0, 1].

PSO has many desirable characteristics, such as flexibility in balancing global and local searches, computational efficiency

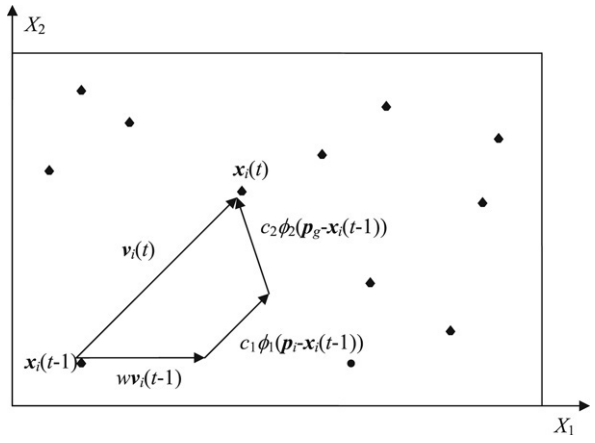

Fig. 3. Vector representation of the PSO operation in a two dimensional space. $\boldsymbol{x}_i(t-1)$ and $\boldsymbol{v}_i(t-1)$ denote the particle's position and the associated velocity vector in the searching space at generation $t-1$, respectively. Vector $c_1\phi_1(\boldsymbol{p}_i - \boldsymbol{x}_i(t-1))$ and $c_2\phi_2(\boldsymbol{p}_g - \boldsymbol{x}_i(t-1))$ describe the particle's cognitive and social activities, respectively. The new velocity $\boldsymbol{v}_i(t)$ is determined by the momentum part, cognitive part, and social part, given in Eq. (6). The particle's position at generation t is updated with $\boldsymbol{x}_i(t-1)$ and $\boldsymbol{v}_i(t)$, given in Eq. (7).

for both time and memory, no need for encoding, and ease of implementation. Also, the memory mechanism implemented in PSO can retain the information of previous best solutions that may get lost during the population evolution. It has been shown that PSO requires less computational cost and can achieve faster convergence than conventional back-propagation in training feedforward neural networks for nonlinear function approximation (Gudise & Venayagamoorthy, 2003). Juang (2004) and Cai, Venayagamoorthy, and Wunsch II (2006) combined PSO with EAs in training RNNs for dynamic plant control and engine data classification, respectively. A comparison of PSO and GA in evolving RNNs was also given by Settles, Rodebaugh, and Soule (2003).

In the context of RNN training, each PSO particle, from a set of M particles $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_M)$, is referred to as a candidate solution, represented as a D $(=N(N+2))$-dimensional vector $\boldsymbol{x}_i = (w_{i,11}, \ldots, w_{i,N1}, w_{i,12}, \ldots, w_{i,1N}, \ldots, w_{i,NN}, \beta_{i,1}, \ldots, \beta_{i,N}, \tau_{i,1}, \ldots, \tau_{i,N})$, $i = 1, \ldots, M$. The velocity associated with particle i is then denoted as $\boldsymbol{v}_i = (v_{i1}, v_{i2}, \ldots v_{iD})$. Here, a batch mode is used for training, which means the parameter updates are performed after all input data points are presented to the model (Haykin, 1999). The procedure for the implementation of PSO involves the following basic steps:

(i) Initialize a population of particles with random positions and velocities of D dimensions. Specifically, the connection weights, biases, and time constants are randomly generated with uniform probabilities over the range $[w_{\min}, w_{\max}]$, $[\beta_{\min}, \beta_{\max}]$, and $[\tau_{\min}, \tau_{\max}]$, respectively. Similarly, the velocities are randomly generated with uniform probabilities in the range $[-V_{\max}, V_{\max}]$, where $V_{\max}$ is the maximum value of the velocity allowed.

(ii) Calculate the estimated gene expression time series based on the RNN model, and evaluate the optimization fitness function defined in Eq. (5) for each particle.

(iii) Compare the fitness value of each particle $\text{Fit}(\boldsymbol{x}_i)$ with $\text{Fit}(\boldsymbol{p}_i)$. If the current value is better, reset both $\text{Fit}(\boldsymbol{p}_i)$ and $\boldsymbol{p}_i$ to the current value and location.

(iv) Compare the fitness value of each particle $\text{Fit}(\boldsymbol{x}_i)$ with $\text{Fit}(\boldsymbol{p}_g)$. If the current value is better, reset $\text{Fit}(\boldsymbol{p}_g)$ and $\boldsymbol{p}_g$ to the current value and location.

(v) Update the velocity and position of the particles with Eqs. (6) and (7).

(vi) Return to step (ii) until a stopping criterion is met, which, usually, occurs upon reaching the maximum number of iterations or discovering high-quality solutions.

PSO has only four major user-dependent parameters. The inertia weight w is designed as a tradeoff between the global and local search. Larger values of w facilitate global exploration while lower values encourage a local search. w can be fixed to some certain value or vary with a random component, such as

$$w = w_{\max} - \frac{\text{rand}}{2}, \tag{8}$$

where rand is a uniform random function in the range of [0, 1]. As an example, if $w_{\max}$ is set as 1, Eq. (8) makes w vary between 0.5 and 1, with a mean of 0.75. In this paper, these two strategies are referred to as PSO-FIXEW and PSO-RADW, respectively. c_1 and c_2, the cognitive and social components, respectively, are used to adjust the velocity of a particle towards $\boldsymbol{p}_i$ and $\boldsymbol{p}_g$. Typically, these are set to 2.0 based on past experience (Eberhart & Shi, 2001; Kennedy et al., 2001). During the evolutionary/swarming procedure, the velocity for each particle is restricted to a limit $V_{\max}$, like in velocity initialization. When the velocity exceeds $V_{\max}$, it is reassigned to $V_{\max}$. If $V_{\max}$ is too small, particles may become trapped into local optima, while if $V_{\max}$ is too large, particles may miss some good solutions. $V_{\max}$ is usually set to around 10%–20% of the dynamic range of the variable on each dimension (Kennedy et al., 2001).

3.2. Differential evolution

DE is a simple stochastic function minimizer and evolves individuals in order to increase the convergence to optimal solutions. The main concept of DE is generating trial parameter vectors by adding a weighted difference of two parameters to a third one. In this way, no separate probability distribution has to be used which makes the scheme completely self organizing (Feoktistov & Janaqi, 2004; Price, Storn, & Lampinen, 2005; Storn & Price, 1997).

Fig. 4 shows the vector diagram of the DE mutation process. A population of individuals, shown by the stars, is initialized. In every generation, for each individual in the population, three other distinct individuals are randomly selected. As depicted in Fig. 4, for the individual $\boldsymbol{x}_1$, three other individuals $\boldsymbol{x}_2$, $\boldsymbol{x}_3$, and $\boldsymbol{x}_4$ are randomly selected. The difference vector between $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ is added to $\boldsymbol{x}_4$ in order to generate an offspring $\boldsymbol{y}_1$ for $\boldsymbol{x}_1$, shown by the small circular dot. Of the parent $(\boldsymbol{x}_1)$ and the offspring $(\boldsymbol{y}_1)$, the one with greater fitness is selected to be part of the next generation of individuals.

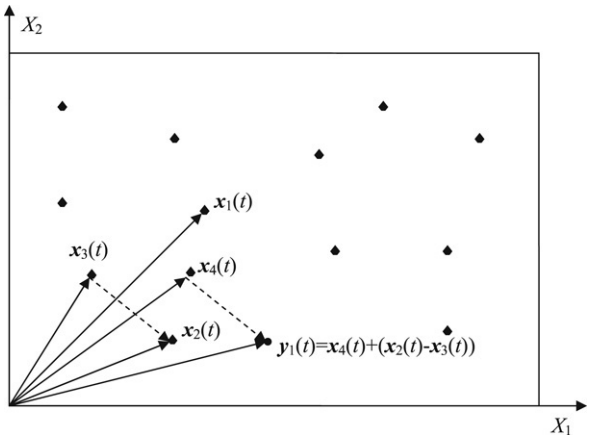

Fig. 4. Vector representation of the DE operation in a two dimensional space. $\boldsymbol{y}_1$ is the offspring reproduced for the parent $\boldsymbol{x}_1$. Other individuals $\boldsymbol{x}_2$, $\boldsymbol{x}_3$, and $\boldsymbol{x}_4$ are randomly selected from the population.

Eqs. (9) and (10) describe the DE operation shown in Fig. 4,

$$y_{1j} = x_{4j} + \gamma \left(x_{2j} - x_{3j}\right), \tag{9}$$

$$y_{1j} = x_{1j}, \tag{10}$$

where j corresponds to the dimension of the individual that is to be mutated and γ is a scaling factor chosen based on the application. The mutation procedure takes place for each dimension in the individual depending on the value of the mutation probability, p_r.

Again, let $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_M)$ be a set of M DE individuals, where each individual is a D-dimensional vector as described in the previous section. The stepwise operation of the DE algorithm is then described as follows (Engelbrecht, 2003):

(i) Initialize a population of M individuals. Set the values of p_r and γ.
(ii) In every generation, for each individual select three distinct individuals randomly from the remaining population.
(iii) For every dimension/parameter of the individual, if mutation is to take place based on the probability p_r, Eq. (9) is used; otherwise Eq. (10) is used to update the parameter values of the offspring.
(iv) For each parent and its offspring, the individual with the higher fitness is passed on to the next generation.
(v) Repeat steps (ii)–(iv) until all the individuals have satisfied some convergence criterion.

3.3. Hybrid differential evolution and particle swarm optimization

In PSO, the search process is based on the social and cognitive components, $\boldsymbol{p}_g$ and $\boldsymbol{p}_i$. The entire swarm tries to follow the $\boldsymbol{p}_g$, thus improving its own position. But for the particular particle that is the $\boldsymbol{p}_g$, it can be seen from Eq. (6) that the new velocity depends solely on the weighted old velocity. To add diversity to the PSO, a hybrid DE and PSO, i.e., DEPSO, is proposed (Zhang & Xie, 2003), thus eliminating the particles from falling into a local minimum.

The DEPSO algorithm involves a two step process. In the first step, the canonical PSO as described in Section 3.1 is implemented. In the second step, the differential evolution mutation operator is applied to the particles. The mutation probability for this study is taken to be one (Zhang & Xie, 2003). Therefore, for every odd iteration, the canonical PSO algorithm is carried out and for every even iteration the DEPSO algorithm is carried out.

The procedure for the implementation of DEPSO involves the following basic steps:

(i) For every odd iteration, carry out the canonical PSO operation on each individual of the population by implementing steps (i)–(vi) from Section 3.2.
(ii) For every even iteration, carry out the following steps:
  (a) For every particle $\boldsymbol{x}_i$, $\boldsymbol{\Delta}_1$ and $\boldsymbol{\Delta}_2$ are calculated using Eqs. (11) and (12). $\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c$, and $\boldsymbol{x}_d$ are different from $\boldsymbol{x}_i$ and randomly chosen,

$$\boldsymbol{\Delta}_1 = \boldsymbol{x}_a - \boldsymbol{x}_b, \quad a \neq b, \tag{11}$$

$$\boldsymbol{\Delta}_2 = \boldsymbol{x}_c - \boldsymbol{x}_d, \quad c \neq d; \tag{12}$$

  (b) The mutation value $\boldsymbol{\delta}_i$ is calculated by Eq. (13) and added in Eq. (14) to create the offspring $\boldsymbol{y}_i$, depending on a mutation probability,

$$\boldsymbol{\delta}_i = (\boldsymbol{\Delta}_1 + \boldsymbol{\Delta}_2)/2; \tag{13}$$

$$\boldsymbol{y}_i = \boldsymbol{p}_i + \boldsymbol{\delta}_i. \tag{14}$$

  The mutation value applied to each dimension of the particle i may be different.
  (c) Once the new population of offspring is created using steps (a) and (b), their fitness is evaluated against that of the parent. The one with the higher fitness is selected to participate in the next generation.
(iii) The $\boldsymbol{p}_g$ and $\boldsymbol{p}_i$ of the new population are recalculated.
(iv) Repeat steps (i)–(iii) until convergence.

4. Data sets

The proposed method is applied to both a synthetic data set and a real data set — SOS data set, as described below.

4.1. Synthetic data set

The synthetic genetic network is a simplified network consisting of 8 genes. Based on the biological assumption that the gene networks are usually sparsely connected (D'haeseleer, 2000; Perrin et al., 2003; van Someren, Wessels, & Reinders, 2001), the number of connections for each gene is limited to no more than 4, which leads to 21 non-zero weights. The network is simulated from a random initial state for each gene. Three curves with 300 time points for each curve are generated, based on Eq. (4), at a time resolution of $\Delta t = 0.1$, since multiple series are more effective as demonstrated before (Wahde & Hertz, 2000, 2001; Xu, Venayagamoorthy, & Wunsch II, 2006). The typical behaviors of some simulated genes are depicted in Fig. 5. It is clear that the expression levels for these genes quickly get saturated, since we do not consider stimuli from the external environment. Since the data points in real data are generally not sufficient; only 30 points from each curve are used to train the regulatory systems, mostly taken from the early stage of the process.

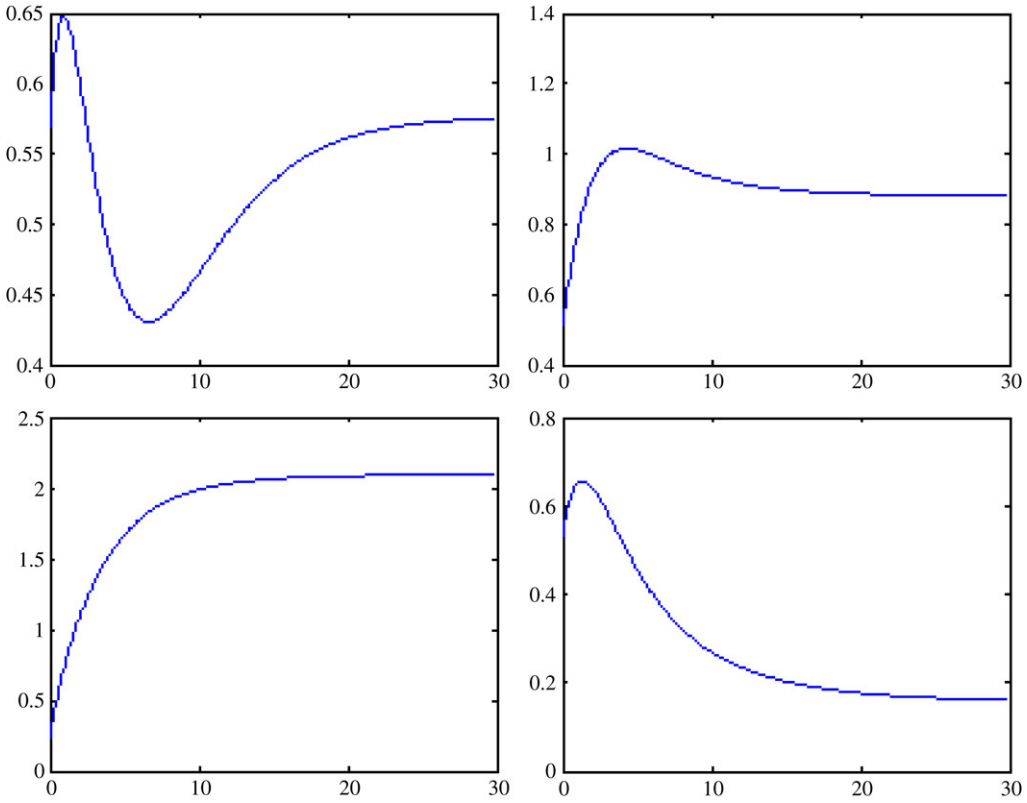

Fig. 5. Typical gene behaviors of the synthetic data in terms of expression level over time. X-axis: time, Y-axis: gene expression level.

4.2. SOS data set

The SOS DNA Repair network in bacterium *Escherichia coli* is depicted in Fig. 6, which consisting of around 30 genes regulated at the transcriptional level (Ronen, Rosenberg, Shraiman, & Alon, 2002). When damage occurs, the protein *Rec*A, which functions as a sensor of DNA damage, becomes activated and mediates cleavage of the *Lex*A protein. The *Lex*A protein is a repressor that blocks transcription of the SOS repair genes. The drop in *Lex*A protein levels causes the activation of the SOS genes. After the damage is repaired or bypassed, the cleavage activity of *Rec*A drops, which causes the accumulation of *Lex*A protein. Then, *Lex*A binds sites in the promoter regions of these SOS genes and represses their expression. The cells return to their original states. For this data set, four experiments have been conducted with different light intensities (Experiments 1 & 2: UV $= 5$ J m^{-2}, Experiments 3 & 4: UV $= 20$ J m^{-2}) (Ronen et al., 2002). Each experiment consists of the expression measurements for 8 major genes (*uvr*D, *lex*A, *umu*D, *rec*A, *uvr*A, *uvr*Y, *ruv*A, and *pol*B) across 50 time points, sampled every 6 minutes. In the study, only the time series from experiment 2 is used, since other series display similar behaviors.

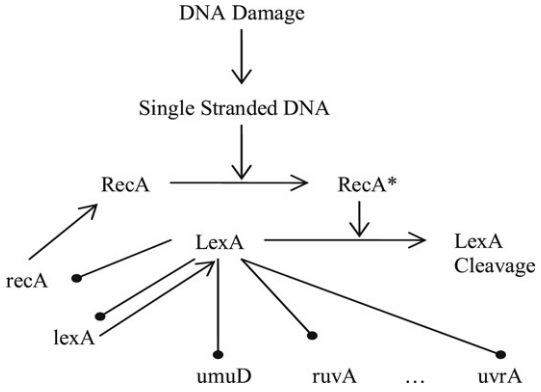

Fig. 6. The bacterial *E. coli* SOS DNA Repair network. Inhibitions are represented by -•, while activations are represented by → (Ronen et al., 2002).

5. Results

5.1. Synthetic data set

The RNN model, together with the three training algorithms, is first applied to a simplified synthetic genetic network consisting of 8 genes. The objective here is to investigate and compare the performance of the three training algorithms on RNNs' parameters learning and the capability of the model in unveiling the potential relations between genes, using the generated time series gene expression data.

All three training algorithms are written in C++ and the difference of their performance time is not significant. Generally, for all three training algorithms, it takes less than 5 s to run 1000 training epochs with 90 time points on a 2.4 GHz Intel Pentium 4 processor with 512 M of DDR RAM. Moreover, the codes are not specifically optimized. For a large-scale data analysis with hundreds or thousands of genes, parallel technology can be used to implement a population-based training algorithm, thus, speeding up the training of the RNN.

Table 1
Performance of PSO in RNN-based gene networks training

c_1	c_2	w	Performance	
			>500 iterations	Average number of iterations
2	2	Fixed at 0.8	12	51.16
		1 − rand/2	8	55.11
0.5	2	Fixed at 0.8	39	60.16
		1 − rand/2	41	47.54
2	0.5	Fixed at 0.8	7	99.02
		1 − rand/2	20	122.45
0.5	0.5	Fixed at 0.8	91	83.11
		1 − rand/2	93	46.29
1.5	2.5	Fixed at 0.8	18	69.11
		1 − rand/2	21	59.44
2.5	1.5	Fixed at 0.8	4	57.73
		1 − rand/2	4	54.63

Table 2
Performance of DE in RNN-based gene networks training

p_r	>500 iterations	Average number of iterations
0.1	0	72.16
0.2	0	53.94
0.3	0	50.82
0.4	0	54.36
0.5	0	57.41
0.6	0	85.12
0.7	0	105.06

Table 3
Performance of DEPSO in RNN-based gene networks training

c_1	c_2	w	Performance	
			>500 iterations	Average number of iterations
2	2	Fixed at 0.8	0	47.39
		1 − rand/2	0	42.91
1.5	2.5	Fixed at 0.8	0	44.83
		1 − rand/2	0	47.90
2.5	1.5	Fixed at 0.8	0	47.62
		1 − rand/2	0	41.49

The performance of PSO, DE, and DEPSO in training the RNN-based 8-gene regulatory network is illustrated in Tables 1, 2, 3, respectively. The performance is compared in terms of the number of iterations required to reach a pre-specified error. The maximum number of iterations allowed is further set as 500. If the training algorithm reaches the expected error threshold within 500 iterations, it is regarded to have achieved convergence. The results summarized in the tables are based on 100 runs, which consist of both the number of times that the iteration exceeds the allowed maximum and the average number of epochs if the algorithm has converged. The number of particles or individuals in each run is chosen at 30.

The parameter selection, such as the inertia weight w and the cognitive and social components c_1 and c_2, are important to the performance of PSO. An optimization strategy has been proposed to use another PSO to explore optimal parameter values (Doctor, Venayagamoorthy, & Gudise, 2004). However, the computational price may become quite expensive in this situation. Here, the PSO performance is examined with the commonly used values of c_1 and c_2 (Doctor et al., 2004; Eberhart & Shi, 2001), together with both PSO-FIXEW and PSO-RADW strategy for w For PSO-FIXEW, w is fixed at 0.8, and for PSO-RADW, $w_{\max}$ is chosen to equal to 1 so that w varies in the range of [0.5, 1]. As indicated in Table 1, the best performance is achieved when PSO-RADW is used, and c_1 and c_2 are set as 2.5 and 1.5, respectively, where the expected error can be reached averagely within 54.63 iterations, except for 4 runs that failed to converge. PSO-FIXEW with the same values of c_1 and c_2 requires 3 more iterations on average. The results when c_1 and c_2 are set as 2 and 2 show that a few more runs did not reach the error within 500 iterations. For all other parameter combinations, the convergence depends more on the initialization and is not consistently stable. For instance, when the values of c_1 and c_2 are 0.5 PSO-RADW is used, the percentage of convergence is only 59%, although the average number iterations is as few as 47.54.

Compared with PSO, DE achieves more stable performance, which means that all 100 runs have converged within 500 epochs. When DE is used, the scaling factor γ is set as 0.5 and the values of the mutation probability p_r are changed from 0.1 to 0.7. It is clear from Table 2 that neither large nor small values of p_r could improve the performance of DE. The smallest number of epochs required on average is 50.82, as p_r takes the value of 0.3. In Table 3, the effectiveness of adding the DE operator to the original PSO is demonstrated. For all six different parameter combinations of w, c_1, and c_2 shown in Table 3, DEPSO has reached the identified error in all 100 runs, with the average number of epochs less than 50. In particular, the best performance at $c_1 = 2.5$ and $c_2 = 1.5$, with PSO-RADW used, only requires averagely 41.49 epochs to achieve convergence, which shows a significant improvement of the results when only either PSO or DE is used.

As discussed before, the weights in the RNN model have their own biological meaning, i.e., a positive weight indicates the activation, a negative weight represents the inhibition, and a zero weight means there is no regulation. Thus, in order to unravel the potential relations among genes or reconstruct the gene networks, it is important to identify the associated weight matrix $\boldsymbol{W} = \{w_{ij}\}$. However, one of the major obstacles for genetic network inference is the "curse of dimensionality" (D'haeseleer, 2000; van Someren, Wessels, Reinders, & Backer, 2001), which describes the exponential growth in computational complexity and the demand for more time points as a result of high dimensionality in the feature space (Haykin, 1999). Typically, the gene expression data currently available contain measurements of thousands of genes, but only with a limited number of time points (less than 50). Fortunately, biological knowledge of genetic regulatory networks assumes that a gene is only regulated by a limited number of genes, which implies that the regulatory networks are sparsely connected, rather than fully connected, and most weight values are zeros (D'haeseleer, 2000; Perrin et al., 2003; van Someren, Wessels, & Reinders, 2001). Therefore, it is

still reasonable to identify the non-zero weights from the time series expression data, with extra caution the effect of the curse of dimensionality. For instance, Wahde and Hertz (2000, 2001) proposed a two-step procedure for genetic regulatory network inference. The goal of the first step is to unravel the possible interactions between genes by iteratively searching non-significant network parameters. With the results of the first stage, the non-zero weights can be further fine-tuned, while the non-significant weights are clamped to zero. This procedure is repeated for different values of the maximum allowed weight.

In the following analysis, a method similar to what was used in Perrin et al. (2003) is adopted to discretize the weights into three categories using the following criteria:

- Category + representing activation: $\mu_{ij} > \mu + \sigma$ and $|\mu_{ij}| > \sigma_{ij}$;
- Category — representing inhibition: $\mu_{ij} < \mu - \sigma$ and $|\mu_{ij}| > \sigma_{ij}$;
- Category 0 representing absence of regulation: otherwise,

where μ_{ij} and σ_{ij}^2 are the mean and variance for the element w_{ij} in the weight connection matrix, and μ and σ^2 are the mean and variance of the means of all the 64 weights.

The following results are based on 500 different runs with random initial values and the networks are trained by DEPSO for 1000 generations, with parameters set as: number of particles = 30, $c_1 = 2.5$, $c_2 = 1.5$ and PSO-RADW used. For each run, the global best $\boldsymbol{p}_g$ is taken as the solution, which leads to 500 networks, on which the inferred weight connection matrix shown in Table 4 are based. Compared with the original weight matrix, also shown in Table 4, the model can identify 14 out of 21 possible connections existed in the network. Among the 14 identified connections in the inferred matrix, 13 are correctly labeled as either activation or inhibition, while the one between node 5 and 6 is indicated as inhibition relation instead of activation. The results also include 5 false positives, which are not existent in the original network and are wrongly identified as activation or inhibition between nodes. It is also observed that the attained standard deviations are generally large, compared with the corresponding mean values, which causes the missing of some real connections in the network. An algorithm is further developed by the authors to evolve not only the network weights, but also the network structures, with a statistical heuristic used to estimate the number of the potential connections in a given network. The results on both synthetic and real data sets show an evident improvement (Xu, Wunsch II, & Frank, in press).

Table 4
The generated connection matrix (upper panel) and the learned connection matrix (lower panel)

w_{ij}							
+	0	0	0	0	0	0	0
+	+	0	0	0	0	0	0
0	−	+	0	0	0	0	0
0	−	−	+	0	0	0	0
0	+	+	−	+	−	0	0
0	0	0	−	+	+	+	−
0	0	0	0	0	0	+	−
0	0	0	0	0	0	0	−
0	0	0	0	+	0	0	0
0	+	0	0	0	0	0	0
0	0	+	0	0	0	0	+
+	−	−	0	0	0	0	0
0	+	+	−	+	+	0	0
0	0	0	−	+	+	0	0
0	0	0	0	0	0	+	0
0	0	0	0	0	+	+	−

Each element in the matrix represents the activation (+), inhibition (−), and absence of regulation (0), between a pair of genes.

5.2. SOS data set

The RNN-based model is then employed to analyze the SOS DNA Repair network in bacterium *E. coli*. Fig. 7 shows the real gene expression profiles and the learned mean expression profiles for Exp. 2. It can be seen that the proposed model provides a promising way in capturing the dynamics of genes from the given time series. Specifically, the dynamic behaviors of gene *lex*A, *rec*A, *uvr*A, *uvr*D, and *umu*D are well learned, with the major change trends of the gene expression levels reflected in the learning curves. The expression profiles for genes *uvr*Y, *ruv*A, and *pol*B oscillate dramatically between the maximum value and zero, and the obtained models generally use their means to represent the profiles because of the definition of the fitness function. One possible strategy is to add an additional item in the fitness function in order to track the difference between two time points. However, when the criteria aforementioned are used to infer the genes interactions, only 2 out of 9 potential connections can be correctly identified from the data, which are the inhibition of *lex*A on *uvr*D and the activation of *rec*A on *lex*A. Furthermore, even two or more time series are used, there still no improvement for the results. As discussed before, large standard deviations are observed for many weight values, which partially explains the reason of many zero weights in the inferred network. Therefore, the reader should note that reconstructing the correct dynamics is easier than reconstructing the correct regulatory network structure. The reason for this is that genetic regulatory network inference is an ill-posed problem — there is no unique solution that will satisfy the data upon which the inference is based. This inherent difficulty is a limitation of our approach, as well as of any other approach we are aware of.

6. Conclusions

Recurrent neural network-based models, using time series gene expression data from microarray experiments, provide a promising way to investigate gene regulatory mechanisms and understand gene interactions. A new hybrid evolutionary–swarm algorithm DEPSO, based on the combined concepts of PSO and DE, is presented to address the challenge of training these RNNs. The performance of DEPSO is also compared with PSO and DE and the simulation results demonstrate the effectiveness of the hybridization of different evolutionary/swarm technologies in improving their search capability in network parameters learning. Although in a general case, two data sets are not

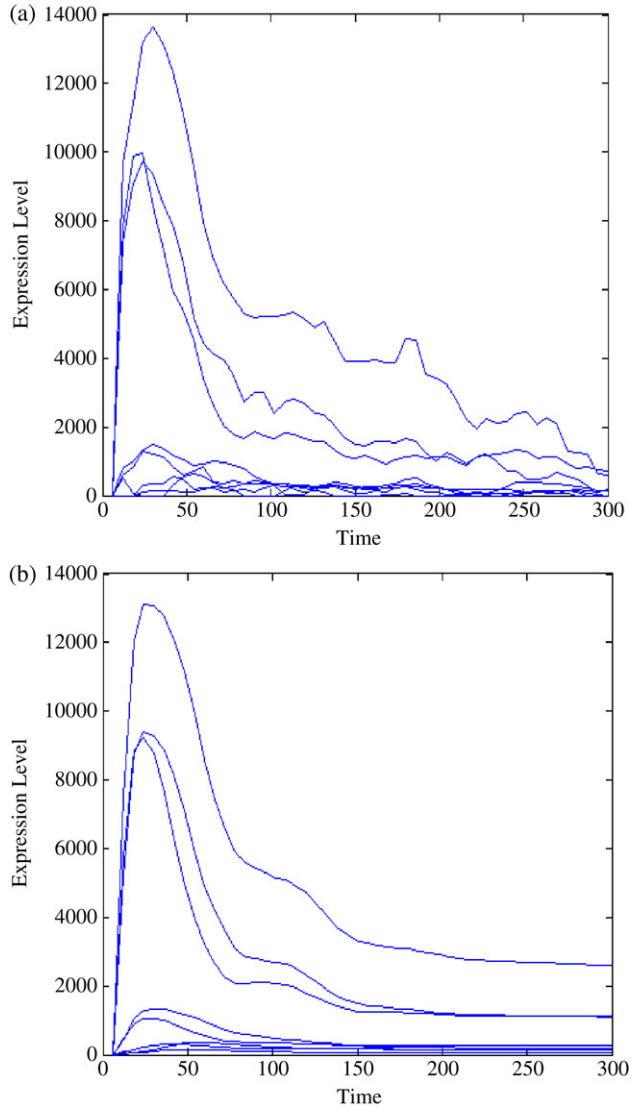

Fig. 7. The measured time series gene expression profiles (a) and the learned mean expression profiles based on an RNN-based model (b).

sufficient to establish the superiority of one optimization algorithm over another, as more gene expression data become available, the performance comparison of these algorithms can be affirmed. Also, the experimental results show that the RNN-based model is very useful in capturing the nonlinear dynamics of genetic regulatory system, and further revealing the interactions between genes. Given the similarity between RNNs and gene networks, the authors believe that RNNs will play a more important role in exploring the mystery of gene regulation relationships. However, currently, one of the major limiting factors for genetic regulatory network inference is the paucity of reliable gene expression time series data, which restricts the applications of current computational methods to only synthetic data, or small-scale real networks, with only several genes or gene clusters. To construct a synthetic system that can simulate some well-known gene networks may provide a way for allowing preliminary investigation and comparison of the proposed methods. On the other hand, more advanced models are critically needed to integrate gene expression data with data from other sources, such as proteomic and metabolomic data, in order to provide more reasonable and accurate modeling of the behaviors of gene regulatory systems. Also, it is equally important to combine prior information about the regulatory networks of study into the model so as to remove some biologically impossible connections. This strategy further simplifies computational requirements and save computational time. For example, genes that are co-regulated may share similar expression patterns in their sequences and have common motifs. Such information could be used to examine the inferred relations and eliminate false positives. As to the RNNs model, it can be further extended to include factors like time delay, which is an important property of genetic regulatory networks, but unfortunately, not well addressed yet. Also, RNNs can also take into account more complex interactions, such as interactions between triplets of variables rather than pairs. RNNs are well known for their capability in dealing with temporal information and are well suited to handle this type of problems. To increase the robustness and redundancy of current models and further improve the search capability of the training algorithms are also important and interesting directions for further research.

Acknowledgements

Partial support for this research from the National Science Foundation, and from the M.K. Finley Missouri endowment, is gratefully acknowledged.

References

Baldi, P., & Long, A. (2001). A Bayesian framework for the analysis of microarray expression data: Regularized *t*-test and statistical inferences of gene changes. *Bioinformatics*, *17*, 509–519.

Bar-Joseph, Z. (2004). Analyzing time series gene expression data. *Bioinformatics*, *20*(16), 2493–2503.

Cai, X., Venayagamoorthy, G., & Wunsch, D., II (2006). Hybrid PSO-EA algorithm for training feedforward and recurrent neural networks for challenging problems. In F. Wang, & D. Liu (Eds.), *Computational intelligence: Theory and application*. Hackensack, NJ: World Scientific Publishing.

De Jong, H. (2002). Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, *9*, 67–103.

delValle, Y., Venayagamoorthy, G., Mohagheghi, S., Hernandez, J., & Harley, R. (2007). Particle swarm optimization: Basic concepts, variants and applications in power systems. *IEEE Transactions on Evolutionary Computation*, in press (doi:10.1109/TEVC.2007.896686).

D'haeseleer, P. (2000). Reconstructing gene network from large scale gene expression data. *Dissertation*. University of New Mexico.

D'haeseleer, P., Liang, S., & Somogyi, R. (2000). Genetic network inference: From co-expression clustering to reverse engineering. *Bioinformatics*, *16*(8), 707–726.

D'haeseleer, P., Wen, X., Fuhrman, S., & Somogyi, R. (1999). Linear modeling of mRNA expression levels during CNS development and injury. In *Proceedings of the pacific symposium biocomputing* (pp. 41–52).

Doctor, S., Venayagamoorthy, G., & Gudise, V. (2004). Optimal PSO for collective robotic search applications. In *Proceedings of congress on evolutionary computation 2004* (pp. 1390–1395).

Eberhart, R., & Shi, Y. (2001). Particle swarm optimization: Developments, applications and recourses. In *Proceedings of the 2001 congress on evolutionary computation* (pp. 81–86).

Eisen, M., & Brown, P. (1999). DNA arrays for analysis of gene expression. *Methods Enzymol*, *303*, 179–205.

Engelbrecht, A. (2003). *Computational intelligence: An introduction*. Chichester: John Wiley.

Feoktistov, V., & Janaqi, S. (2004). Generalization of the strategies in differential evolution. In *Proceedings of 18th international parallel and distributed processing symposium* (pp. 165–170).

Fogel, D. (1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, *5*(1), 3–14.

Friedman, N., Linial, M., Nachman, I., & Pe'er, D. (2000). Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, *7*, 601–620.

Gudise, V., & Venayagamoorthy, G. (2003). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the 2003 IEEE swarm intelligence symposium* (pp. 110–117).

Hallinan, J., & Jackway, P. (2005). Network motifs, feedback loops and the dynamics of genetic regulatory networks. In *Proceedings of the IEEE symposium on computational intelligence in bioinformatics and computational biology* (pp. 1–7).

Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Husmeier, D. (2003). Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks. *Bioinformatics*, *19*(17), 2271–2282.

Jaeger, H. (2002). A tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the echo state network approach. *GMD report 159*. German National Research Center for Information Technology.

Juang, C. (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transaction on Systems, Man, and Cybernetics, Part B*, *34*(2), 997–1006.

Kauffman, S. (1993). *The origins of order: Self-organization and selection in evolution*. New York, NY: Oxford University Press.

Keedwell, E., & Narayanan, A. (2005). Discovering gene networks with a neural–genetic hybrid. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *2*(3), 231–242.

Kennedy, J., Eberhart, R., & Shi, Y. (2001). *Swarm intelligence*. San Diego, CA: Academic Press.

Kolen, J., & Kremer, S. (2001). *A field guide to dynamical recurrent networks*. New York, NY: IEEE Press.

Latchman, D. (2005). *Gene regulation: A eukaryotic perspective* (5th ed.). New York, NY: Taylor & Francis Ltd.

Liang, S., Fuhrman, S., & Somogyi, R. (1998). Reveal: A general reverse engineering algorithm for inference of genetic network architectures. In *Proceedings of the pacific symposium biocomputing* (pp. 18–29).

Lipshutz, R., Fodor, S., Gingeras, T., & Lockhart, D. (1999). High density synthetic oligonucleotide arrays. *Nature Genetics*, *21*, 20–24.

McLachlan, G., Do, K., & Ambroise, C. (2004). *Analyzing microarray gene expression data*. Hoboken, NJ: John Wiley & Sons.

Mjolsness, E., Mann, T., Castaño, R., & Wold, B. (2000). From co-expression to co-regulation: An approach to inferring transcriptional regulation among gene classes from large-scale expression data. In S. Solla, T. Leen, & K. Müller (Eds.), *Advances in neural information processing systems*: *Vol. 12* (pp. 928–934). Cambridge, MA: MIT Press.

Murphy, K., & Mian, S. (1999). Modeling gene expression data using dynamic Bayesian networks. *Technical report*. Berkeley: Computer Science Division, University of California.

Perdew, G., Vanden Heuvel, J., & Peters, J. (2006). *Regulation of gene expression*. Totowa, NJ: Human Press.

Perrin, B., Ralaivola, L., Mazurie, A., Battani, S., Mallet, J., & d'Alchė-Buc, F. (2003). Gene networks inference using dynamic Bayesian networks. *Bioinformatics*, *19*(Suppl. 2), ii138–ii148.

Price, K., Storn, R., & Lampinen, J. (2005). *Differential evolution: A practical approach to global optimization*. Berlin: Springer-Verlag.

Ronen, M., Rosenberg, R., Shraiman, B., & Alon, U. (2002). Assigning numbers to the arrows: Parameterizing a gene regulation network by using accurate expression kinetics. *Proceedings of the National Academic Sciences USA*, *99*(16), 10555–10560.

Schena, M., Shalon, D., Davis, R., & Brown, P. (1995). Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, *270*(5235), 467–470.

Settles, M., Rodebaugh, B., & Soule, T. (2003). Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network. In *Proceedings of the genetic and evolutionary computation conference 2003* (pp. 148–149).

Shmulevich, I., Dougherty, E., & Zhang, W. (2002). From Boolean to probabilistic Boolean networks as models of genetic regulatory networks. *Proceedings of the IEEE*, *90*(11), 1778–1792.

Smith, A., Jarvis, E., & Hartemink, A. (2002). Evaluating functional network inference using simulations of complex biological systems. *Bioinformatics*, *18*(Suppl. 1), S216–S224.

Storn, R., & Price, K. (1997). Differential evolution — A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*, 341–359.

Tamada, Y., Kim, S., Bannai, H., Imoto, S., Tashiro, K., Kuhara, S., et al. (2003). Estimating gene networks from gene expression data by combining Bayesian network model with promoter element detection. *Bioinformatics*, *19*(Suppl. 2), ii227–ii236.

van Someren, E., Wessels, L., & Reinders, M. (2000). Linear modeling of genetic networks from experimental data. In *Proceedings of the 8th international conference on intelligent systems for molecular biology* (pp. 355–366).

van Someren, E., Wessels, L., & Reinders, M. (2001). Genetic network models: A comparative study. In *Proceedings of SPIE, micro-arrays: Optical technologies and informatics* (pp. 236–247).

van Someren, E., Wessels, L., Reinders, M., & Backer, E. (2001). Robust genetic network modeling by adding noisy data. In *Proceedings of the 2001 IEEE-EURASIP workshop on nonlinear signal and image processing*.

Vohradský, J. (2001). Neural network model of gene expression. *FASEB Journal*, *15*, 846–854.

Wahde, M., & Hertz, J. (2000). Coarse–grained reverse engineering of genetic regulatory networks. *Biosystems*, *55*, 129–136.

Wahde, M., & Hertz, J. (2001). Modeling genetic regulatory dynamics in neural development. *Journal of Computational Biology*, *8*, 429–442.

Weaver, D., Workman, C., & Stormo, G. (1999). Modeling regulatory networks with weight matrices. In *Proceedings of the pacific symposium on biocomputing* (pp. 112–123).

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of IEEE*, *78*(10), 1550–1560.

Werhli, A., Grzegorczyk, M., & Husmeier, D. (2006). Comparative evaluation of reverse engineering gene regulatory networks with relevance networks, graphical Gaussian models and Bayesian networks. *Bioinformatics*, *22*(20), 2523–2531.

Xu, R., Hu, X., & Wunsch, D., II (2004a). Inference of genetic regulatory networks from time series gene expression data. In *Proceedings of international joint conference on neural networks* (pp. 1215–1220).

Xu, R., Hu, X., & Wunsch, D., II (2004b). Inference of genetic regulatory networks with recurrent neural network models. In *Proceedings of the 26th annual international conference of IEEE engineering in medicine and biology society* (pp. 2905–2908).

Xu, R., Venayagamoorthy, G., & Wunsch, D., II (2006). A study of particle swarm optimization in gene regulatory networks inference. In J. Wang, Z. Yi, J. Zurada, B. Lu, & H. Yin (Eds.), *Advances in neural networks — ISNN 2006: Third international symposium on neural networks* (pp. 648–653). Berlin, Heidelberg: Springer.

Xu, R., & Wunsch, D., II (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, *16*(3), 645–678.

Xu, R., Wunsch, D., II, & Frank, R. (2007). Inference of genetic regulatory networks with recurrent neural network models using particle swarm optimization. IEEE Transactions on Computational Biology and Bioinformatics, in press (doi:10.1109/TCBB.2007.1056).

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, *87*(9), 1423–1447.

Yu, J., Smith, V., Wang, P., Hartemink, A., & Jarvis, E. (2004). Advances to Bayesian network inference for generating causal networks from observational biological data. *Bioinformatics*, *20*(18), 3594–3603.

Zhang, W., & Xie, X. (2003). DEPSO: Hybrid particle swarm with differential evolution operator. In *Proceedings of IEEE international conference on systems, man, and cybernetics* (pp. 3816–3821).