International Journal of Neural Systems, Vol. 20, No. 6 (2010) 481–500

DOI: 10.1142/S0129065710002565

World Scientific
www.worldscientific.com

ON THE PROBABILISTIC OPTIMIZATION OF SPIKING NEURAL NETWORKS

STEFAN SCHLIEBS* and NIKOLA KASABOV†
Knowledge Engineering and Discovery Research Institute
Auckland University of Technology, New Zealand
**sschlieb@aut.ac.nz*
†*nkasabov@aut.ac.nz*

MICHAËL DEFOIN-PLATEL
Department of Biomathematics and Bioinformatics
Rothamsted Research, United Kingdom
michael.defoinplatel@gmail.com

The construction of a Spiking Neural Network (SNN), i.e., the choice of an appropriate topology and the configuration of its internal parameters, represents a great challenge for SNN based applications. Evolutionary Algorithms (EAs) offer an elegant solution for these challenges and methods capable of exploring both types of search spaces simultaneously appear to be the most promising ones. A variety of such heterogeneous optimization algorithms have emerged recently, in particular in the field of probabilistic optimization. In this paper, a literature review on heterogeneous optimization algorithms is presented and an example of probabilistic optimization of SNN is discussed in detail. The paper provides an experimental analysis of a novel Heterogeneous Multi-Model Estimation of Distribution Algorithm (hMM-EDA). First, practical guidelines for configuring the method are derived and then the performance of hMM-EDA is compared to state-of-the-art optimization algorithms. Results show hMM-EDA as a light-weight, fast and reliable optimization method that requires the configuration of only very few parameters. Its performance on a synthetic heterogeneous benchmark problem is highly competitive and suggests its suitability for the optimization of SNN.

Keywords: Spiking Neural Network; heterogeneous optimization algorithms; Estimation of Distribution Algorithms; Multi-Model EDA; evolutionary algorithms.

1. Introduction

The desire to better understand the impressive information processing capabilities of the mammalian brain has recently led to the development of more complex and more biologically plausible connectionist models, the spiking neural networks (SNN). See e.g., Ref. 1 for an introduction and Ref. 2 for a comprehensive standard text on the material. Recent reviews on SNN can be found in Refs. 3 and 4. These models use trains of spikes as internal information representation rather than continuous variables. In Ref. 1, the author argues that SNN have at least similar computational power to the traditional artificial neural networks (ANN), such as the multi-layer perceptron (MLP) derivates. Nowadays, many studies use SNN for practical applications, some of them demonstrating very promising results in solving complex real world problems. For example, substantial progress has been made in areas such as speech recognition,[5] learning rules,[6,7] associative memory,[8] and function approximation.[9] Other studies have investigated the simulated growth of SNN and the emergence of firing sequences in large networks,[10] but also hardware implementations using CMOS technology[11] and the use of SNN for EEG data classification and seizure detection.[12,13]

However, these novel models have also introduced new complex problems. The learning of a desired

network behavior, i.e., the synaptic plasticity, is particularly hard to address in SNN. In a general way, the overall construction of a network, including the choice of an appropriate topology and the setting of its internal parameters, remains a great challenge for SNN based applications, because a variety of difficulties impair the development of learning procedures for SNN. The explicit time dependence results in asynchronous information processing that commonly requires complex software and/or hardware implementations to simulate these neural networks. Additional difficulties are added by the fact that recurrent network topologies are commonly used in SNN and thus the formulation of a straightforward learning method, such as back-propagation for MLP, is not possible. Finally, the fine-tuning of the learning algorithm itself appears to be another complex but critical aspect of the construction of SNN and therefore specific methods need to be designed and evaluated.

Evolutionary Algorithms (EAs) are stochastic optimization techniques offering an elegant solution for constructing and optimizing SNN. Numerous studies have discussed such schemes, especially in the context of the evolution of the weight matrix and the topology of neural networks. See for example the study presented in Ref. 14 where a Differential Evolution algorithm trains the weights of a SNN to solve a classification task and the work by Ref. 15 where the topology of a neural network is optimized using a Genetic Algorithm (GA).

Methods capable of exploring different search spaces simultaneously appear to be the most promising. Recently, such an *heterogeneous* evolutionary algorithm was proposed in the context of a SNN based feature selection and classification problem.[16] This algorithm employs separate probabilistic models to represent a combined solution consisting of a binary and a continuous sub-component. The binary sub-components encodes the feature space (presence/absence of features) and the continuous sub-component encodes the parameter space of the SNN. Due to the metaphor of the optimization algorithm the SNN based classifier was named the Quantum-inspired SNN (QiSNN) framework. It was shown in Ref. 16 that employing a heterogeneous optimizer is very beneficial for the classification performance of the system. Exchanging the optimizer for a more traditional EA resulted in a significant performance decrease.

In this paper, we argue that the heterogeneous EA proposed in Ref. 16 may be a general tool for the optimization of other SNN based applications. Consequently, this paper has the following goals. First, we present a literature review on heterogeneous EA that might be potentially useful for optimizing SNN. Second, recent developments on QiSNN are reviewed, since this approach represents an example for the successful application of heterogeneous EAs. Third, we experimentally analyze the novel heterogeneous EA introduced in QiSNN and integrate it into the family of Estimation of Distribution Algorithms (EDA). We provide practical guidelines for configuring the method in order promote the application of the algorithm to other problems. Furthermore, we compare its performance to numerous optimization algorithms and discuss its computational costs.

2. EA for Optimizing SNN

The idea of exploring heterogeneous search spaces is not new. Among the earliest contributions to this research area is the work by Ref. 17 where a GA was used to optimize binary chromosomes structured in sub-components to allow the encoding of connectivity and connection weights in a single bit string. Promising results have been reported on a 9×9 bit character recognition problem. A similar approach was investigated in Ref. 18, where an algorithm called ANNA ELEONORA[a] was presented. Here the presence or absence of a connection between two neurons was encoded by a connectivity bit, followed by a number of additional bits representing the corresponding connection weight. Due to the binary representation of the weights, a conversion from bit strings into real values was required. The granularity of the weights, i.e., the number of bits used for encoding a single weight, was adapted as part of the evolutionary process. As a consequence, since the interpretation of the bits in the chromosome was not homogeneous, a set of complex crossover and mutation operators has been defined. The method was later developed further in Ref. 19, in which the binary chromosome was replaced by a continuous

[a]Abbreviation for **A**rtificial **N**eural **N**etworks **A**daptation: **E**volutionary **Le**arning **o**f **N**eural **O**ptimal **R**unning **A**bilities.

one. Although the real value representation seems appropriate for the evolution of connection weights, it is less suitable for the representation of the connectivity bit. Similar genetic approaches were discussed in Refs. 20–23.

All of the above studies employ EAs to explore a heterogeneous search space using either a binary or a continuous representation of the chromosome. As a consequence these methods are not optimally adapted for the exploration of either the continuous or the binary sub-component of a candidate solution. In Ref. 24 this issue was explicitly addressed by proposing a method called FeaSANNT (Feature Selection and Artificial Neural Network Training). FeaSANNT is a GA using a binary representation for the evolution of appropriate feature subsets and a continuous representation for the optimization of the weight matrix of the neural net. The method is discussed in greater detail in Ref. 25. For each representation individual genetic operators are implemented. A standard two-point crossover along with bit-flip mutation is used for the binary landscape, while uniform random mutations and Lamarckian learning using back-propagation are applied to the variables of the continuous solution part. A practical application of FeaSANNT on a wood veneer classification problem can be found in Ref. 26. Further efforts have been made to also evolve the network topology, cf. Ref. 27, which required the definition of additional operators, such as node deletion and insertion according to some user-specified probability parameters.

Very recent studies follow similar trends. The chromosome in Ref. 28 consists of the concatenation of three parts: A connectivity bit encoding the presence or absence of a connection, a real-valued weight, and another bit representing the presence or absence of a particular hidden neuron. A GA is used but the actual genetic operators are unfortunately not reported in the article. A modified version of a Particle Swarm Optimizer is proposed in Ref. 29 that also evolves the neural transfer function in addition to topology and connection weights.

Many heterogeneous optimizers are specialized for the evolution of the topology and the weight matrix of neural networks. Only very few *general* mixed-variable algorithms exist. Arguably among the most promising algorithms on heterogeneous optimization is the Mixed Bayesian Optimization Algorithm (MBOA) introduced in Ref. 30. In MBOA, a set of decision trees that are iteratively constructed and adapted during the evolutionary process, explore the search space in a probabilistic fashion. New solution candidates are sampled according to the current state of the trees. Although MBOA was not extensively investigated on heterogeneous problems, promising results have been obtained on binary benchmark problems. The continuous optimization performance of MBOA, on the other hand, is less competitive as experimentally demonstrated in Ref. 31. Furthermore, the method involves a significant computational overhead, which has motivated a multi-threaded implementation on parallel hardware.[32]

Other directions have suggested the use of different EA variants. A heterogeneous version of an Ant Colony Optimization (ACO) algorithm was proposed in Ref. 33. Due to the lack of comparison algorithms, the authors have experimentally investigated the performance of the method using a number of continuous benchmark functions. Thus, the suitability of this ACO on mixed-variable problems is less clear. However, it is interesting to note that the principle idea of ACO is also based on a probabilistic exploration of the search space, as shown in Refs. 34 and 35. This is very similar to the above-mentioned MBOA, despite the very different metaphor employed in ACO. While ACO assumes a population of "ants", each of them iteratively constructing a solution according to discrete or continuous probability distributions, MBOA emphasizes on an entirely mathematical description of its working.

When summarizing the presented survey on mixed-variable optimization methods, several conclusions can be made. First of all, the exploration of heterogeneous search spaces is feasible and was implemented in numerous algorithms. However, few of them are suitable for an application to general heterogeneous optimization problems. Either the representation of the search space is non-optimal, i.e., binary-only or continuous-only representations, or the optimization algorithm is too problem specific, e.g., its application aims explicitly towards topology and weight optimization of neural networks. Furthermore, although some general purpose mixed-variable optimizers have been developed recently, none of them was studied thoroughly on heterogeneous problems. We have also noted that the most

promising mixed-variable algorithms employ a probabilistic model to explore the search space.

3. QiSNN

With encouraging results SNN were presented in the context of a feature subset selection (FSS) problem.[36] In this work, a binary state-of-the-art optimization algorithm, namely the Versatile Quantum-inspired Evolutionary Algorithm (vQEA),[37] was combined with an evolving Spiking Neural Network (eSNN).[38] Through implementing quantum principles, vQEA evolves in parallel a number of independent probability vectors, that may interact at certain intervals with each other, forming a multi-model Estimation of Distribution Algorithm (EDA).[39] Following the wrapper approach,[40] vQEA was used to identify relevant feature subsets and simultaneously evolve an optimal eSNN parameter setting. Due to the quantum metaphor employed in vQEA, the architecture was referred to as the Quantum-inspired SNN (QiSNN) framework. Applied to carefully designed benchmark data, containing irrelevant and redundant features of varying information quality, QiSNN reported excellent classification results and an accurate detection of relevant information in the data set.[36]

The QiSNN framework was also used in a case study on ecological modeling[41] in which relevant features for predicting the presence/absence of insect species at different geographical sites were identified. Recently, QiSNN was improved by introducing a novel hybrid optimization method based on vQEA, which allows the exploration of heterogeneous search spaces.[16] In the context of QiSNN, this algorithm uses a binary representation for optimizing feature subsets and a continuous representation for evolving appropriate real-valued configurations of the eSNN classifier. The novel method was shown to be highly beneficial on two benchmark problems and one real-world data set.

The QiSNN is illustrated in Fig. 1. Given a specific data set, samples are selected and for each of them a feature subset is extracted using a bit mask in which each bit represents a single feature. The quality of this feature subset is then evaluated by the eSNN classification method which is configured using a specific parameter set, i.e., a vector of real values. The quality measure for both the bit mask and the parameter configuration is used as the fitness criterion for an evolutionary algorithm, which in turn proposes a new candidate solution. This solution consists of a binary and a continuous sub-component, that represent a bit mask and a parameter set respectively. The process iterates until a termination criterion is met.

In Ref. 42, some detailed experimental analysis of the behavior and functioning of QiSNN were presented. Especially interesting is the process of the simultaneous optimization of features subsets and eSNN parameters and their interaction and mutual influences. Furthermore, the role of the

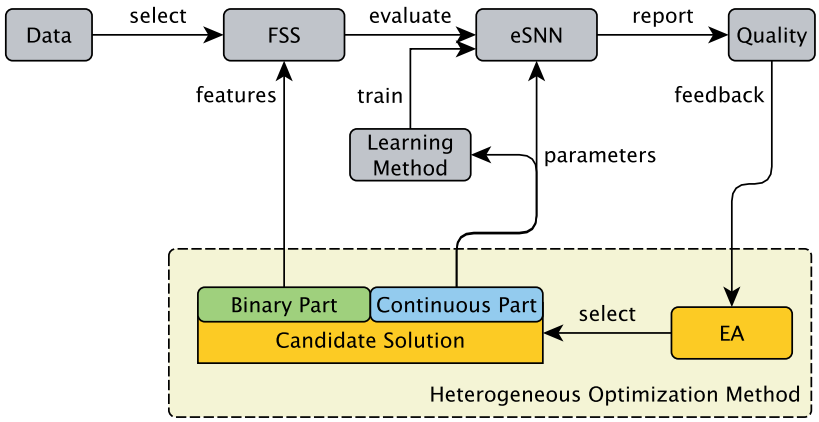


Fig. 1. Structure of QiSNN. A specialized evolutionary algorithm evolves a combined solution consisting of a binary and a real-valued sub-component, which represent a feature subset (FSS) for a data sample and a parameter configuration for an eSNN classifier respectively. The quality of this combined solution is evaluated by determining the classification accuracy of eSNN on a set of test samples.

neural encoding and its impact on the classification characteristics of QiSNN was investigated. As a result of this analysis the experimenter is provided with a comprehensive understanding of all parameters involved in QiSNN and recommendations are given for configuring parameters that are not included in the evolutionary optimization process.

A comprehensive description and analysis of QiSNN can be found in Ref. 43.

4. Heterogeneous Optimization

QiSNN employs a heterogeneous optimization algorithm belonging to the family of Estimation of Distribution Algorithms (EDA).[44] It uses specialized representations to explore a binary and a continuous search space simultaneously. Consequently, the method was named Heterogeneous Multi-Model EDA (hMM-EDA). In the context of QiSNN, hMM-EDA was shown to perform well especially when compared to a binary-only optimizer.

In this section we present a comprehensive description of hMM-EDA. On a synthetic benchmark problem, we derive some practical guidelines to configure the method, compare its performance to a number of related EAs and discuss the computational costs of the method.

4.1. *Description of the algorithm*

The overall structure of hMM-EDA can be decomposed in three different interacting levels, see Fig. 2.

Individuals The lowest level corresponds to *individuals*. An individual i at generation t contains a heterogeneous probabilistic model $\mathcal{H}_i(t)$ and two compound solutions $S_i(t)$ and $A_i(t)$. More precisely, $\mathcal{H}_i$ corresponds to a string of N pairs of models $(Q_i^{(j)}, P_i^{(j)})$:

$$\mathcal{H}_i = \mathcal{H}_i^{\infty} \ldots \mathcal{H}_i^{\mathcal{N}} = \begin{bmatrix} Q_i^{(1)} & \ldots & Q_i^{(N)} \\ P_i^{(1)} & \ldots & P_i^{(N)} \end{bmatrix} \quad (1)$$

where P_i denotes the continuous representation of the search space in the form of a string of Gaussian distributions:

$$P_i = P_i^1 \ldots P_i^N = \begin{bmatrix} \mu_i^{(1)} & \ldots & \mu_i^{(N)} \\ \sigma_i^{(1)} & \ldots & \sigma_i^{(N)} \end{bmatrix} \quad (2)$$

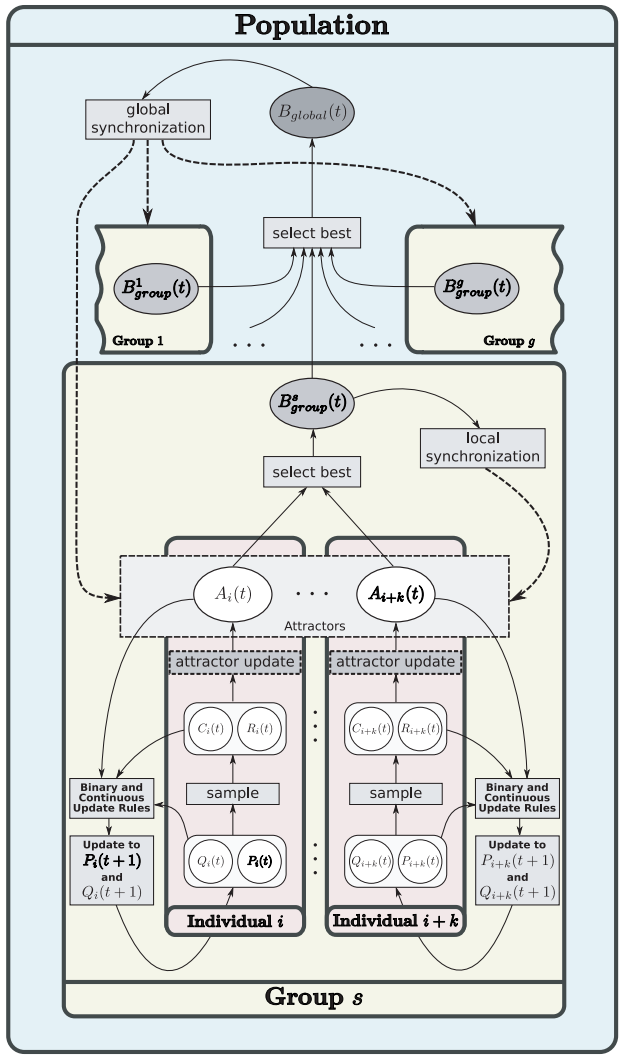


Fig. 2. Three interacting levels can be distinguished in hMM-EDA: The individual, group and population level.

and Q_i the binary representation in form of a concatenation of $\mathcal{Q}$bits:

$$Q_i = Q_i^1 \ldots Q_i^N = \begin{bmatrix} \alpha_i^1 & \ldots & \alpha_i^N \\ \beta_i^1 & \ldots & \beta_i^N \end{bmatrix} \quad (3)$$

The pair $(\mu_i^{(j)}, \sigma_i^{(j)})$ corresponds to the parameters of the distribution of the jth variable of the ith probabilistic model, and $(\alpha_i^{(j)}, \beta_i^{(j)})$ correspond to the probability amplitudes of the jth $\mathcal{Q}$bit of the ith probabilistic model. We refer to Refs. 37 and 39 for a comprehensive discussion on the probabilistic model represented by a $\mathcal{Q}$bit.

Each variable in Q_i and P_i is sampled according to $(\alpha_i^{(j)}, \beta_i^{(j)})$ and $(\mu_i^{(j)}, \sigma_i^{(j)})$ respectively, forming

a compound solution $S_i = (C_i, R_i)$, where C_i is a bit vector and R_i a real-valued vector of size N. Hence, $S_i(t)$ represents a configuration in the search space whose quality can be determined using a fitness function f.

Without loss of generality, we assume each $r_i^{(j)} \in R_i$ to be defined in the range $[-1, 1]$. As a consequence, each $r_i^{(j)} \in R_i$ follows a *truncated normal distribution* in the range $[-1, 1]$. Truncated normals can be sampled using a simple numerical procedure and the technique is widely adopted in pseudo-random number generation, see e.g., Ref. 45 for an efficient implementation.

To each individual i, a solution A_i consisting of a binary and a continuous sub-component is attached acting as an attractor for $\mathcal{H}_i$. Every generation $S_i(t)$ and A_i are compared in terms of their fitness. If A_i is better than $S_i(t)$, an update operation is applied on the corresponding model $\mathcal{H}_i$. Each representation uses its corresponding update operator to drive the probabilistic model. The binary probabilistic model Q_i is updated using the rotation gate as employed in vQEA. The jth $\mathcal{Q}$bit at generation t of Q_i is updated as follows:

$$\begin{bmatrix} \alpha_i^j(t+1) \\ \beta_i^j(t+1) \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix} \begin{bmatrix} \alpha_i^j(t) \\ \beta_i^j(t) \end{bmatrix} \tag{4}$$

where the constant $\Delta\theta$ is a rotation angle designed in compliance with the application problem.[46] We note that the sign of $\Delta\theta$ determines the direction of rotation (clockwise for negative values). In this study the application of the rotation gate operator is limited in order to keep θ in the range $[0, \pi/2]$.

The continuous model P_i is modified by a mean and standard deviation shift. The mean $\mu^{(j)}$ is shifted towards the value of the current attractor $a^{(j)}$ at location j. Depending on the distance $d^{(j)}(t) = a^{(j)}(t) - \mu^{(j)}(t)$, a shift $\Delta\mu^{(j)}(t)$ at generation t is defined as a sigmoid function:

$$\Delta\mu^{(j)}(t) = \frac{2}{1 + e^{-5d^{(j)}(t)}} - 1 \tag{5}$$

which is then used to perform the update:

$$\mu^{(j)}(t+1) = \mu^{(j)}(t) + \theta_\mu \Delta\mu^{(j)}(t) \tag{6}$$

In Eq. (6) a parameter θ_μ is introduced which we will refer to as the learning rate of the mean. We note that θ_μ corresponds to the maximum mean shift in a single generation. The standard deviation at generation t is updated using

$$\sigma^{(j)}(t+1) = \begin{cases} \sigma^{(j)}(t) \times (1 - \theta_\sigma) & \text{if } |d^{(j)}(t)| < \sigma^{(j)}(t) \\ \sigma^{(j)}(t) \times (1 - \theta_\sigma)^{-1} & \text{otherwise} \end{cases} \tag{7}$$

In Eq. (7) a parameter θ_σ is introduced which we will refer to as the learning rate of the standard deviation. In order to avoid divergent behavior of the algorithm, i.e., $\sigma^{(j)}(t)$ increases indefinitely, the domain of $\sigma^{(j)}(t)$ is restricted by defining upper and lower bounds, such that $\sigma_{\min} \leq \sigma^{(j)}(t) \leq \sigma_{\max}$.

Consequently, hMM-EDA requires the setting of three learning rates for the model update: the learning rate $\Delta\theta$ used in the rotation gate to update a $\mathcal{Q}$bit, and the two learning rates θ_μ and θ_σ to update the Gaussian mean and standard deviation respectively.

Groups The second level corresponds to *groups*. The population is divided into g groups each containing k individuals having the ability of synchronizing their attractors. For that purpose, the best attractor (in terms of fitness) of a group, denoted B_{group}, is stored at every generation and is periodically distributed to the group attractors. This phase of local synchronization is controlled by the parameter S_{local}.

Population The set of all $p = g \times k$ individuals forms the *population* and defines the topmost level of the multi-model approach. As for the groups, the individuals of the population can synchronize their attractors, too. For that purpose, the best attractor (in terms of fitness) among all groups, denoted B_{global}, is stored every generation and is periodically distributed to the group attractors. This phase of global synchronization is controlled by the parameter S_{global}.

4.2. *Benchmark problem*

In order to analyze hMM-EDA, a simple benchmark is proposed here. We consider a minimization problem containing two equally sized search landscapes, i.e., a binary and a continuous one. The dimensionality (number of variables) of each landscape is denoted by N. Target vectors representing the global optimum of the problem are specified for each landscape:

a binary vector $B^* = (b_1^*, \ldots, b_N^*)$ and a continuous vector $R^* = (r_1^*, \ldots, r_N^*)$. A solution for this problem is denoted as $S = (B, R)$, where $B = (b_1, \ldots, b_N)$ and $R = (r_1, \ldots, r_N)$ represent the binary and the real part of the problem respectively. The goal is to evolve a solution S, such that it becomes equivalent to the target solution $S^* = (B^*, R^*)$. More specifically, the fitness function in this problem is defined as the Euclidean distance between the real part R of a solution S to the real part R^* of the target solution S^*. The binary part B of the solution acts as a mask in the computation of the distance: only if bit $b_i = 1$, does the corresponding real value r_i contribute to the computation of the difference. Furthermore, if $b_i \neq b_i^*$ a penalty is added to the overall fitness of the solution. The complete fitness function is described in detail in Fig. 3. The global optimum is reached if the fitness becomes $f = 0$.

The problem is designed to resemble a typical wrapper-based feature selection scenario. The feature space is represented by the binary solution sub-component while the parameter space of the classification method is reflected by the real-valued sub-component. If a certain bit (feature) $b_i \in B$ is wrongly selected, i.e., $b_i \neq b_i^*$, the solution $S = (B, R)$ receives a penalty $(r_i^*)^2$. Thus, different bits (features) may have a different significance, since different fitness penalties are associated with them. On the other hand, if the bit (feature) is correctly selected, i.e., $b_i = b_i^* = 1$, the size of the fitness penalty depends on the quality of the variable (parameter of the classifier) r_i of the real solution part R. Thus, even if the optimization method correctly selects a certain feature, the fitness penalty may be large if the classification method is poorly parametrized. Both solution sub-components need to *co-operate* in order to minimize the fitness penalties.

Require: $B = (b_1, \ldots, b_N)$ and $R = (r_1, \ldots, r_N)$
$f \Leftarrow 0$
for $i = 1$ to N **do**
  if $b_i = 1$ and $b_i^* = 1$ **then**
    $d \Leftarrow (r_i^* - r_i)^2$
  else if $b_i \neq b_i^*$ **then**
    $d \Leftarrow (r_i^*)^2$
  else
    $d \Leftarrow 0$
  end if
  $f \Leftarrow f + d$
end for

Fig. 3. Algorithm to evaluate the fitness f of a heterogeneous solution $S = (B, R)$.

In the following experiment, the target solution $S^* = (B^*, R^*)$ was chosen in dependence of the problem size N:

$$B^* = (\overbrace{1, \ldots, 1}^{\times \frac{N}{2}}, \overbrace{0, \ldots, 0}^{\times \frac{N}{2}})$$
$$R^* = (\overbrace{p_{\max}, \ldots, p_{\min}}^{\times \frac{N}{2},\ \text{equi-distant}}, \underbrace{p_{\max}, \ldots, p_{\min}}_{\times \frac{N}{2},\ \text{equi-distant}}) \quad (8)$$

The parameters $p_{\min}$ and $p_{\max}$ denote the minimum and maximum fitness penalty assigned to a certain bit. Penalties are equi-distantly distributed over the first $\frac{N}{2}$ and last $\frac{N}{2}$ elements of the real-valued solution sub-component. In the experiments discussed later in this paper, $p_{\min} = 0.5$ and $p_{\max} = 1$ are chosen.

It is noteworthy that, using this configuration, only the first $\frac{N}{2}$ real-valued elements $r_i \in R$ have to be optimized by the algorithm. The other $\frac{N}{2}$ elements become irrelevant in the fitness computation, if the algorithm has evolved zeroes at the last $\frac{N}{2}$ positions of the binary vector.

Since different fitness penalties are assigned to each binary element, all bits correspond to a different marginal fitness contribution. In the GA domain, such a situation is also referred to as *salient* building blocks.[47] Due to the difference of significance, the convergence behavior of the binary probabilistic model is directly affected. More specifically, a sequential convergence of variables is expected, starting with the ones with the highest salience and finishing with the ones with the lowest salience. This sequential convergence phenomenon is called *domino convergence* and was first mentioned in Ref. 48.

4.3. *Configuring hMM-EDA*

The population structure consisting of ten individuals that are fully synchronized in every generation, i.e., $S_{\text{global}} = S_{\text{local}} = 1$, is directly adopted from previous experiments on vQEA. Although this setting has generally reported good optimization performance, it is noted that this structure might not be necessarily optimal for hMM-EDA. Nevertheless, we restrict the analysis here to this simple configuration only and leave the exploration of more complex

population hierarchies for future research. We derive practical guidelines on the setting of the three learning rates θ_μ, θ_σ and $\Delta\theta$ in the next paragraphs.

Learning rates θ_μ and θ_σ The first series of experiments investigates the impact of the learning rates θ_μ and θ_σ on the performance of hMM-EDA. For this analysis, the learning rate of the rotation gate is fixed to specific values $\Delta\theta \in \{0.0005\pi, 0.001\pi, 0.005\pi\}$. For each $\Delta\theta$, the parameters θ_μ and θ_σ are varied and the success rate of hMM-EDA is computed based on 25 independent runs on the proposed heterogeneous benchmark problem. A run is considered successful if the final achieved fitness value is lower than 10^{-5}. The success rate is defined as the ratio between the successful and total number of runs. Different problem sizes N are investigated and a maximum number of $N \times 4 \times 10^3$ fitness evaluations (FES) is allowed for the optimizer.

Figure 4 presents the success rate of hMM-EDA in dependence of the two learning rates θ_μ and θ_σ for the problem sizes $N = 25$, $N = 50$, $N = 100$ and $\Delta\theta = 0.001\pi$. The darker the color in these diagrams, the higher the success rate of the particular parameter setting. It is clearly demonstrated that a variety of settings are suitable for solving the problem. We also note that the setting of the mean shift θ_μ has only a low impact on the performance of the algorithm. Additionally, it is only slightly affected by the increase of the problem size N. The learning rate θ_σ, on the other hand, is a critical parameter that strongly depends on the problem size. The larger the size N, the smaller θ_σ has to be set in order to achieve optimal performance. Very similar results are reported for $\Delta\theta = 0.0005\pi$ and $\Delta\theta = 0.005\pi$. Due to the similarities of the figures, the results for $\Delta\theta = 0.0005\pi$ and $\Delta\theta = 0.005\pi$ are not presented here.

A general observation of the presented results above is the comparably low importance of the mean shift rate θ_μ. An appropriate default value seems to be $\hat{\theta}_\mu = 0.05$. The standard deviation rate θ_σ is a critical parameter in hMM-EDA. It should always be adjusted according to the dimensionality N of the problem. A reasonable choice for a default value seems $\hat{\theta}_\sigma = \frac{1}{10\times N}$.

Learning rate $\Delta\theta$ Since the mean shift rate θ_μ has only a low impact on the performance of hMM-EDA, we now focus on the relationship between the learning rate $\Delta\theta$ of the binary model and the standard deviation shift θ_σ. For this analysis θ_μ is fixed to $\hat{\theta}_\mu = 0.05$, which was earlier introduced as the default value for this parameter. Due to the explicit linkage between the binary and continuous search variables, several local optima exist in the fitness landscape of the heterogeneous benchmark problem. A known remedy against premature convergence of QEA and vQEA towards local optima in multi-modal landscapes is the use of a modified rotation gate operator, which was introduced as the H_ϵ gate in Ref. 49. For vQEA the H_ϵ gate was already utilized in the performance and noise analysis presented in Refs. 37 and 39. In the following experiments, the two configurations $\epsilon = 0$ and $\epsilon = \sin^2(0.02\pi)$ are investigated, where for $\epsilon = 0$ the H_ϵ gate equals to the standard rotation gate, while $\epsilon = \sin^2(0.02\pi)$ was introduced as an appropriate default configuration for H_ϵ in Ref. 39.

Figure 5 presents the average success rate showing the interdependence of $\Delta\theta$ and θ_σ obtained from 25 independent runs of hMM-EDA on the benchmark problem for a problem size $N = 100$. The darker the color in these diagrams, the higher the success rate of the particular parameter setting.

In the case of the standard rotation gate, cf. Fig. 5 (top), a certain correlation between $\Delta\theta$ and θ_σ is observed. Clearly the best performance is reported when small values for both learning rates are used. If $\Delta\theta$ is increased, θ_σ also needs to increase (and vice versa) in order to maintain a non-zero success rate. Particularly a combination of a small (large) $\Delta\theta$ and a large (small) θ_σ is not suitable for the algorithm.

In the case of the H_ϵ gate, a similar correlation is noted, but additionally another effect impacts the performance of the method, cf. Fig. 5 (bottom). Very surprising is the low sensitivity of the algorithm to the learning rate of the binary model. Almost any $\Delta\theta$ is suitable, as long as the standard deviation shift θ_σ is small enough. The H_ϵ operator prevents the convergence of the binary probabilistic model towards 1 or 0, and instead defines for the two values $|\alpha|^2$ and $|\beta|^2$ of a $\mathcal{Q}$bit a minimal and a maximal probability, i.e., ϵ and $1 - \epsilon$ respectively. Due to the residual probabilities ϵ and $1 - \epsilon$ a certain mechanism is employed by the algorithm that is similar to the bit-flip mutations used in a GA. With low probabilities, a certain $\mathcal{Q}$bit may collapse towards 1 (or 0), although its amplitudes have evolved close towards 0 (or 1). Thus, at least for some problems, premature

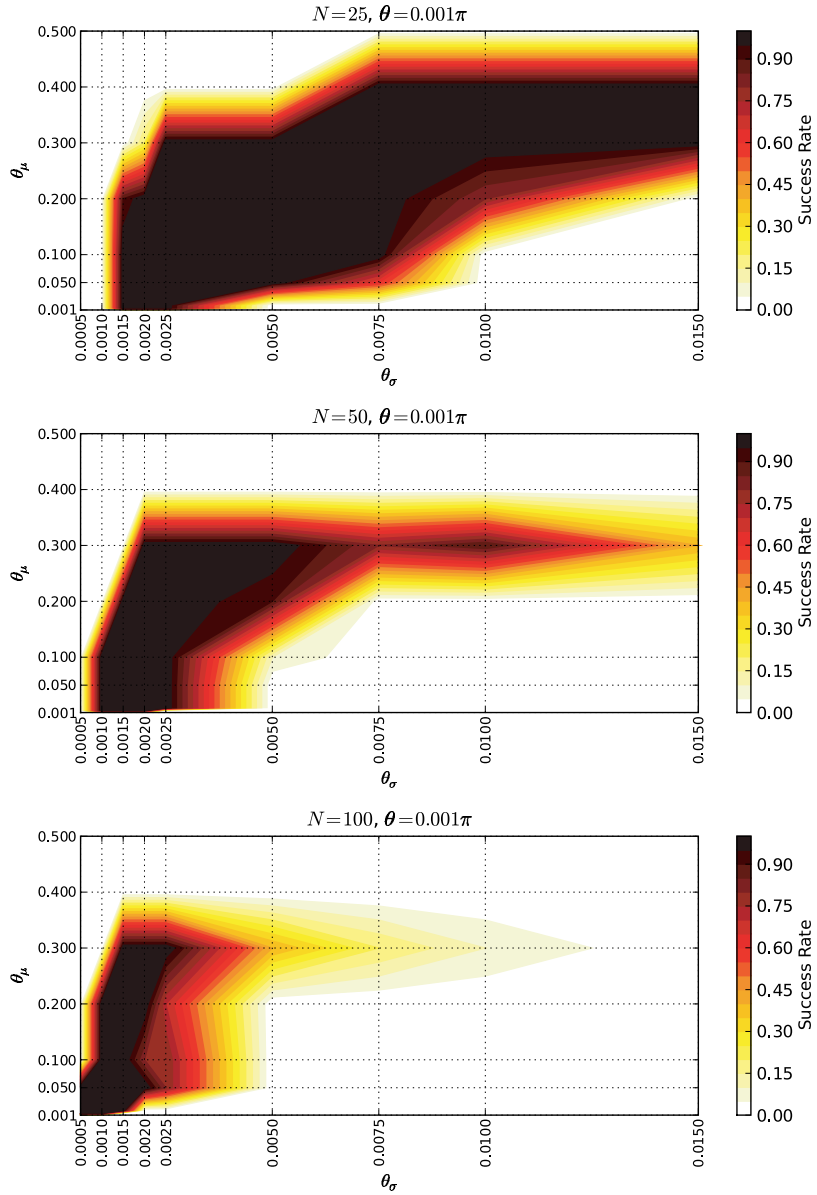


Fig. 4. The success rate of hMM-EDA in dependence of the two learning rates θ_μ and θ_σ. The parameter $\Delta\theta$ of the rotation gate was fixed to 0.001π. Different problem sizes of the benchmark are presented. The diagrams show the average of 25 independent runs. The low impact of the learning rate θ_μ of the mean shift is clearly demonstrated. The learning rate θ_μ is dependent on the problem size N.

convergence of a specific bit due to hitch-hiking phenomena may be compensated through the use of the H_ϵ gate.

In the context of the heterogeneous benchmark problem, the H_ϵ gate is highly advantageous and counteracts hitch-hiking efficiently, since larger learning rates $\Delta\theta$ not only increase the risk of hitch-hiking effects, but at the same time also increase the impact of the mutations on the probabilistic model. If a certain bit-flip mutation is evaluated to be positive, i.e., the fitness of the mutated solution improves, the corresponding $\mathcal{Q}$bit is updated towards the mutated bit value. Larger learning rates result in larger model shifts, which in turn increase the probability of mutations for the $\mathcal{Q}$bit in the next generation. Thus, in succeeding generations the state of a $\mathcal{Q}$bit may completely invert due to the impact of earlier mutations.

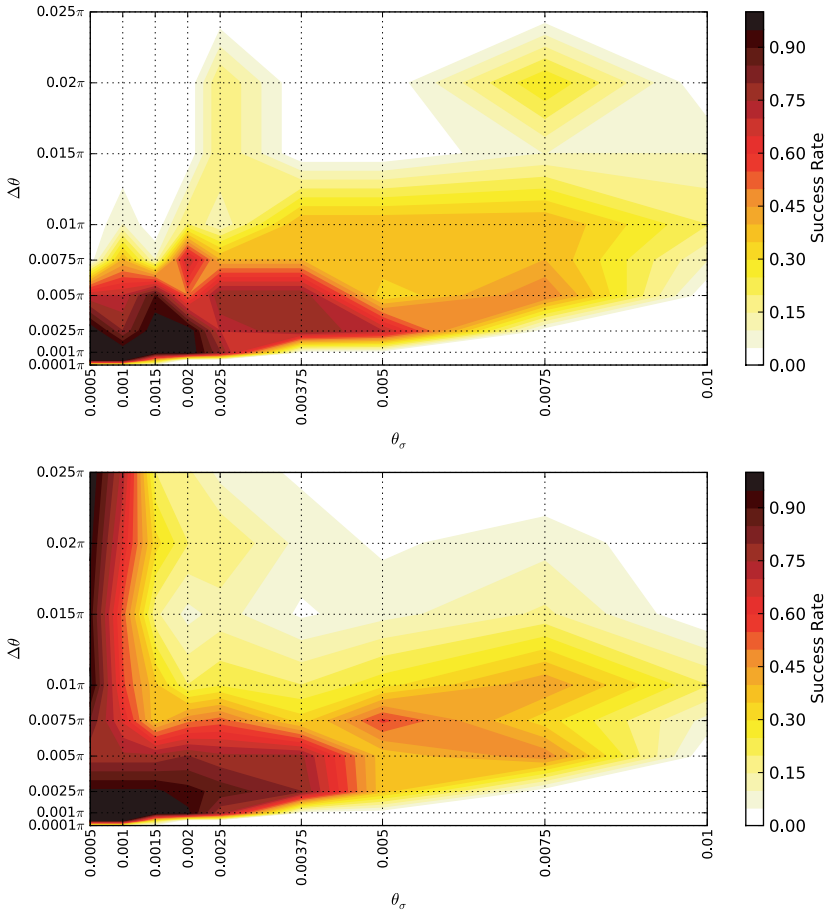


Fig. 5. The success rate of hMM-EDA in dependence of the two learning rates $\Delta\theta$ and θ_σ. The learning rate θ_μ was fixed to the default value $\hat{\theta}_\mu = 0.05$. In the **top** figure the standard rotation gate was used, which allows the convergence of the probability amplitudes α and β to 0 or 1. Using the H_ϵ gate (**bottom** figure) prevents the complete convergence of the amplitudes, which decreases the sensitivity of hMM-EDA to the parameter $\Delta\theta$. Almost any $\Delta\theta$ is suitable, as long as the standard deviation shift θ_σ is small enough.

Since the mutations occur with low probabilities only and are entirely random for each bit, many generations are required to mutate the non-optimal bits in the binary sub-component of a solution. If a certain $\mathcal{Q}$bit $Q_i^{(j)}$ is non-optimally converged, the corresponding continuous model $P_i^{(j)}$ has to maintain enough diversity, i.e., the standard deviations $\sigma_i^{(j)}$ need to stay reasonably large, until the desired mutation occurs, in order to be able to optimize the continuous search variable r_j after the bit b_j is mutated. This is due to the fact that the continuous variable r_j only contributes to the fitness computation if the corresponding bit $b_j = b_j^* = 1$. In any other case r_j is irrelevant in the fitness evaluation and its value is subject to genetic drift, since no selective pressure is provided by the fitness function. Thus, the described mutation mechanism works well only for small learning rates θ_σ, which prevents the premature convergence of $\sigma_i^{(j)}$ due to drift before a positive mutation at bit b_j occurs.

From the presented experimental analysis we conclude that the most critical parameter in hMM-EDA is the learning rate θ_σ of the standard deviation shift. It should be adjusted according to the number of variables in the problem to solve. The mean shift θ_μ is of low importance and can be fixed to standard values for most problems. Configuring the learning rate $\Delta\theta$ for updating the binary probabilistic model is straightforward if the H_ϵ gate is used.

An appropriate value for the parameter ϵ was presented in the experiments.

4.4. *Performance analysis*

In this section, the performance of hMM-EDA is evaluated. Results are compared to a selection of contemporary continuous-only and binary-only optimization methods, along with the already mentioned MBOA. For the binary-only optimizers three first-level binary EDA are considered, namely UMDA,[50] PBIL[51] and cGA.[52] Using binary representations to explore continuous search spaces is a typical scenario in the context of traditional genetic algorithms, cf. e.g., the early work in Ref. 53 and also Ref. 18, in which a binary GA was applied on a heterogeneous optimization problem. Bit strings of predefined length are mapped into real values by a Gray encoding.

Using a continuous representation to explore a binary landscape, on the other hand, is less common. An example can be found in Ref. 19 where a real-coded GA evolves the topology and the weight matrix of a neural network. Since a continuous representation is used, real values $x \in \mathbb{R}$ of the chromosome are converted into bits using a simple mapping:

$$\delta(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{else} \end{cases} \tag{9}$$

This mapping enables a numerical optimizer to explore a binary search space. Due to the excellent performance reported in Ref. 54, the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) is used for the performance analysis presented here.

Furthermore, we investigate the performance of the continuous-only version of hMM-EDA. In this version, hMM-EDA utilizes the continuous probabilistic model exclusively, i.e., the binary model is removed. Similar to CMA-ES, Eq. (9) is used to explore the binary part of the search space. We refer to the continuous version of hMM-EDA as the continuous Multi-Model EDA (cMM-EDA). This scenario allows us to directly compare the benefits of the heterogeneous optimization of hMM-EDA over the continuous-only optimization of cMM-EDA and the binary-only optimization of vQEA.

4.4.1. *Benchmark analysis*

We apply hMM-EDA to the proposed heterogeneous benchmark problem. In all experiments a problem size $N = 100$ is used which should present a certain challenge for the tested algorithms. Each method is allowed to perform a maximum number of $N \times 4 \times 10^3 = 4 \times 10^5$ FES. The search space was limited to the range $[-1, 1]$ for each search variable.

Two configurations of hMM-EDA are considered that are directly adopted from the configuration analysis discussed above. The first setting is $\theta_\mu = \hat{\theta}_\mu = 0.05$ and $\theta_\sigma = \hat{\theta}_\sigma = \frac{1}{10 \times N} = 0.001$. The only difference of the second setting is a slightly faster rate $\theta_\sigma = 0.0015$. Both configurations use a small value, $\Delta\theta = 0.001\pi$, for the H_ϵ gate to update the binary probabilistic model, that was shown to be efficient in the previous analysis.

Optimal configurations for all tested methods were obtained through a comprehensive parameter analysis. In the case of UMDA, the choice of an appropriate population size n is critical. Different sizes in the range $[200, 2500]$ were investigated. The default ratio of 50% for the truncation selection is used. PBIL also requires the setting of a population size which was varied $n \in [50, 300]$. Additional parameters are the learning rate R_l and the mutation shift R_s. We assume $R_l = R_s$; values were varied $R_l, R_s \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.2\}$. In total, 36 different parameter configurations were investigated for PBIL.

The only parameter of cGA is the virtual population size n that was optimized in the range $[150, 1250]$. For vQEA, a single group of ten fully synchronized individuals is tested while the learning rate of a H_ϵ gate is varied $\Delta\theta \in \{0.001, 0.0025, 0.005, 0.0075, 0.01\}$. Default $\epsilon = \sin^2(0.02\pi)$ was used. All binary methods use 12 bits to encode a single real value. A Gray encoding was used for the conversion of bit strings into a continuous value.

The CMA-ES employs special mechanisms that adapt most of its parameters automatically. According to Ref. 54, only the initial starting points and the initial standard deviation of the method needs to be specified for a given problem. We adopt the strategy given in[54] and set the initial standard deviation to $10^{-2}(B - A)/2$, with $[A, B]^N = [-1, 1]^N$ being the search interval of the benchmark. The

initial starting points were uniformly sampled in the range $[-0.1, 0.1]^N$, which is slightly different from Ref. 54, but in favor for the method. In Ref. 54 the initial starting points were uniformly drawn from $[A, B]^N$. No further parameter fine-tuning was attempted for this method. The Java implementation provided by Nikolaus Hansen[b] was used in the experiments.

For cMM-EDA, the default value for $\theta_\mu = \hat{\theta}_\mu$ is used and only $\theta_\sigma \in \{0.00025, \ldots, 0.0015\}$ is varied which allows a direct comparison to hMM-EDA. The only difference between cMM-EDA and hMM-EDA is the different probabilistic model for the binary solution sub-component of the latter one. All continuous methods use Eq. (9) to explore the binary solution sub-component.

MBOA only requires the proper setting of its population size n. Sizes are varied $n \in \{50, 100, 125, 150, 200, 250, 300\}$. These values correspond to the size of the base population in MBOA. Every generation, $\tau \times N$ new offspring are generated and evaluated, thus each generation requires the computation of $\tau \times N$ FES instead of N. Parameter $\tau \in \mathbb{R}$ was set to 0.5 as recommended as the default in.[31] An official implementation of the method in the programming language C++ is provided by Jiri Ocenasek.[c]

4.4.2. *Results*

The results of the parameter analysis can be found in Tables 1 and 2. For each setting of a method, the best, median, worst and mean performance along with the standard deviation obtained from 25 independent runs is presented in the columns. Additionally, the success rate as defined in the previous section is given. The most suitable configuration in terms of success rate is highlighted. In the cases where the success rate is not discriminative enough, the mean fitness and number of required FES are considered, in order to determine the most suitable setting.

hMM-EDA, vQEA, UMDA, cMM-EDA and MBOA all report a success rate of 100%. With cGA, 76% of the runs were successful, while not a single run reached the required fitness threshold using PBIL. It is also noted that the binary methods require a rather large population size due to the mapping of 100×12 bits into 100 real values. Because of this mapping, the overall precision of the optimization is also affected. In the case of cMM-EDA, hMM-EDA, CMA-ES and MBOA, the optimization was stopped when the fitness value dropped below 10^{-10}. In the tables, this situation is indicated by the value $0.00e + 00$.

In Fig. 6, the fitness evolution of the median run is presented. We note the logarithmic scale of the fitness axis. hMM-EDA is clearly the fastest optimizer among the tested algorithms on this benchmark, requiring only 12300 FES to achieve the desired solution accuracy of $\epsilon = 10^{-5}$ and 21700 FES to drop below a precision of 10^{-10}. The fitness is exponentially minimized resulting in a linear curve on the logarithmic scale of the ordinate.

Particularly interesting is the fitness evolution of PBIL, since a number of stepwise fitness improvements are observed. This behavior is caused by mutations having a positive impact on the fitness of a solution. Mutations become very important in the later stages of the optimization process when the probabilistic model has almost converged towards a specific solution candidate in the search space. Mutating a wrongly evolved bit in the binary solution sub-component can result in an especially significant fitness improvement of the overall solution. Since a comparably large mutation shift $R_s = 0.1$ is used, an improvement due to mutation can be efficiently exploited by PBIL.

The step-wise fitness evolution of CMA-ES, on the other hand, has an entirely different reason. It reflects the local restarts of the method after getting stuck on some non-optimal solution during the evolutionary process. In the presented median run, CMA-ES performed four independent restarts, the first finishing after 88,483 FES, the second after 190,799 FES, the third after 327,562 FES, while the fourth restart exhausted the maximum number of FES and achieved the best results. That means, if the initial population of CMA-ES represents a solution close to the optimum, the method can converge towards it very quickly. Indeed, the fastest run of CMA-ES required only three restarts and a total of 294,065 FES to achieve the precision of 10^{-10}.

[b]Available at http://www.lri.fr/~hansen
[c]Available at http://jiri.ocenasek.com

Table 1. Results of the parameter analysis for hMM-EDA, vQEA, UMDA, cGA, cMM-EDA, CMA-ES and MBOA. Shown is the best, median and worst run obtained from 25 independent runs. Additionally the mean and standard deviation of the runs, along with the success rate is presented (see text for a definition of the success rate). The most suitable setting in terms of success rate for each method is highlighted.

Method	Setting	Best	Med	Worst	Mean	Stdev	Success rate (%)
hMM-EDA	$\Delta\theta = 0.001\pi$, $\theta_\sigma = 0.001$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	100
	$\Delta\theta = 0.001\pi$, $\theta_\sigma = 0.0015$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	100
vQEA	$\Delta\theta = 0.001\pi$	$1.41e-04$	$2.38e-04$	$2.82e-04$	$2.30e-04$	$3.42e-05$	0
	$\Delta\theta = 0.0025\pi$	$9.84e-06$	$1.72e-05$	$2.91e-05$	$1.73e-05$	$5.49e-06$	8
	$\Delta\theta = 0.005\pi$	$2.89e-06$	$5.41e-06$	$9.08e-06$	$5.52e-06$	$1.50e-06$	100
	$\Delta\theta = 0.0075\pi$	$2.41e-06$	$3.40e-06$	$5.30e-06$	$3.71e-06$	$9.42e-07$	100
	$\Delta\theta = 0.01\pi$	$2.07e-06$	$3.83e-06$	$5.15e-06$	$3.67e-06$	$7.78e-07$	100
UMDA	$n = 200$	$1.17e+00$	$1.55e+00$	$3.85e+00$	$1.84e+00$	$6.82e-01$	0
	$n = 300$	$5.84e-02$	$6.98e-01$	$2.02e+00$	$7.93e-01$	$5.23e-01$	0
	$n = 400$	$1.84e-02$	$8.69e-02$	$1.06e+00$	$2.66e-01$	$3.05e-01$	0
	$n = 500$	$7.66e-04$	$1.64e-02$	$7.59e-01$	$1.46e-01$	$2.30e-01$	0
	$n = 600$	$9.22e-05$	$5.72e-03$	$7.81e-01$	$4.37e-02$	$1.51e-01$	0
	$n = 700$	$6.04e-04$	$5.08e-03$	$4.28e-01$	$3.89e-02$	$1.03e-01$	0
	$n = 800$	$3.28e-05$	$1.14e-03$	$2.50e-01$	$1.32e-02$	$4.87e-02$	0
	$n = 900$	$1.15e-05$	$1.79e-04$	$5.64e-03$	$8.31e-04$	$1.28e-03$	0
	$n = 1500$	$1.21e-06$	$2.76e-06$	$2.54e-04$	$2.50e-05$	$5.46e-05$	72
	$n = 2000$	$1.21e-06$	$1.30e-06$	$3.33e-05$	$3.72e-06$	$6.84e-06$	92
	$n = 2500$	$1.21e-06$	$1.21e-06$	$4.68e-06$	$1.50e-06$	$7.49e-07$	100
cGA	$n = 150$	$3.72e-01$	$1.30e+00$	$3.12e+00$	$1.41e+00$	$6.70e-01$	0
	$n = 250$	$1.79e-02$	$3.33e-01$	$1.43e+00$	$4.27e-01$	$4.22e-01$	0
	$n = 350$	$2.82e-04$	$1.33e-02$	$5.32e-01$	$1.13e-01$	$1.56e-01$	0
	$n = 450$	$2.06e-04$	$3.83e-03$	$3.82e-01$	$4.12e-02$	$1.02e-01$	0
	$n = 550$	$1.94e-05$	$4.62e-04$	$5.74e-01$	$3.46e-02$	$1.20e-01$	0
	$n = 750$	$1.91e-06$	$5.94e-05$	$1.01e-03$	$2.09e-04$	$2.92e-04$	20
	$n = 850$	$1.42e-06$	$5.53e-06$	$2.92e-01$	$1.17e-02$	$5.73e-02$	72
	$n = 900$	$1.21e-06$	$4.15e-06$	$3.09e-05$	$7.26e-06$	$7.79e-06$	76
	$n = 950$	$1.64e-06$	$6.63e-06$	$4.11e-05$	$1.02e-05$	$9.14e-06$	60
	$n = 1000$	$5.70e-06$	$1.10e-05$	$5.56e-05$	$1.43e-05$	$1.03e-05$	36
	$n = 1250$	$1.33e-04$	$1.81e-04$	$3.34e-04$	$1.88e-04$	$4.27e-05$	0
cMM-EDA	$\theta_\sigma = 0.00025$, $\theta_\mu = \hat{\theta}$	$2.02e-03$	$2.37e-03$	$2.72e-03$	$2.35e-03$	$1.79e-04$	0
	$\theta_\sigma = 0.0005$, $\theta_\mu = \hat{\theta}$	$2.44e-07$	$3.13e-07$	$3.45e-07$	$3.07e-07$	$2.61e-08$	100
	$\theta_\sigma = 0.00075$, $\theta_\mu = \hat{\theta}$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	100
	$\theta_\sigma = 0.001$, $\theta_\mu = \hat{\theta}$	$0.00e+00$	$0.00e+00$	$6.83e-01$	$2.73e-02$	$1.34e-01$	96
	$\theta_\sigma = 0.0015$, $\theta_\mu = \hat{\theta}$	$0.00e+00$	$0.00e+00$	$5.55e-01$	$4.56e-02$	$1.31e-01$	88
CMA-ES		$0.00e+00$	$7.34e-06$	$4.40e-01$	$2.89e-02$	$1.00e-01$	52
MBOA	$N = 50$	$4.66e+00$	$8.11e+00$	$1.32e+01$	$8.20e+00$	$1.83e+00$	0
	$N = 100$	$0.00e+00$	$0.00e+00$	$1.38e+00$	$2.61e-01$	$3.59e-01$	56
	$N = 125$	$0.00e+00$	$0.00e+00$	$5.70e-01$	$6.22e-02$	$1.50e-01$	84
	$N = 150$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	100
	$N = 200$	$3.93e-07$	$9.61e-06$	$2.24e-04$	$2.80e-05$	$4.91e-05$	52
	$N = 250$	$3.56e-04$	$1.37e-03$	$4.47e-03$	$1.43e-03$	$9.20e-04$	0
	$N = 300$	$2.74e-03$	$1.08e-02$	$1.64e-02$	$1.05e-02$	$3.71e-03$	0

Table 2. Results of the parameter analysis for PBIL. Shown is the best, median and worst run obtained from 25 independent runs. Additionally the mean and standard deviation of the runs, along with the success rate is presented (see text for a definition of the success rate). The most suitable setting in terms of success rate for PBIL is highlighted.

Method	Setting	Best	Med	Worst	Mean	Stdev	Success rate (%)
PBIL	$n = 50,\ R_l = R_s = 0.001$	$6.46e+00$	$7.53e+00$	$8.58e+00$	$7.52e+00$	$5.29e-01$	0
	$n = 50,\ R_l = R_s = 0.005$	$1.17e-02$	$1.81e-02$	$2.56e-02$	$1.81e-02$	$3.17e-03$	0
	$n = 50,\ R_l = R_s = 0.01$	$2.64e-03$	$3.45e-03$	$5.62e-03$	$3.61e-03$	$7.13e-04$	0
	$n = 50,\ R_l = R_s = 0.05$	$1.80e-03$	$2.45e-03$	$3.29e-03$	$2.50e-03$	$4.21e-04$	0
	$n = 50,\ R_l = R_s = 0.1$	$2.37e-03$	$3.93e-03$	$5.09e-03$	$3.89e-03$	$6.35e-04$	0
	$n = 50,\ R_l = R_s = 0.2$	$5.06e-03$	$9.15e-03$	$1.33e-02$	$9.07e-03$	$1.64e-03$	0
	$n = 100,\ R_l = R_s = 0.001$	$1.43e+01$	$1.63e+01$	$1.74e+01$	$1.62e+01$	$8.38e-01$	0
	$n = 100,\ R_l = R_s = 0.005$	$1.14e-01$	$1.49e-01$	$1.66e-01$	$1.46e-01$	$1.25e-02$	0
	$n = 100,\ R_l = R_s = 0.01$	$3.90e-03$	$5.84e-03$	$7.59e-03$	$5.87e-03$	$9.03e-04$	0
	$n = 100,\ R_l = R_s = 0.05$	$3.57e-04$	$5.30e-04$	$7.53e-04$	$5.28e-04$	$9.72e-05$	0
	$n = 100,\ R_l = R_s = 0.1$	$5.43e-04$	$7.14e-04$	$1.06e-03$	$7.44e-04$	$1.50e-04$	0
	$n = 100,\ R_l = R_s = 0.2$	$8.41e-04$	$1.46e-03$	$2.30e-03$	$1.54e-03$	$3.36e-04$	0
	$n = 150,\ R_l = R_s = 0.001$	$1.90e+01$	$2.06e+01$	$2.23e+01$	$2.07e+01$	$7.96e-01$	0
	$n = 150,\ R_l = R_s = 0.005$	$4.76e-01$	$6.14e-01$	$7.37e-01$	$6.09e-01$	$6.72e-02$	0
	$n = 150,\ R_l = R_s = 0.01$	$1.75e-02$	$2.52e-02$	$2.94e-02$	$2.39e-02$	$3.64e-03$	0
	$n = 150,\ R_l = R_s = 0.05$	$2.06e-04$	$2.99e-04$	$4.70e-04$	$3.19e-04$	$6.61e-05$	0
	$n = 150,\ R_l = R_s = 0.1$	$2.26e-04$	$3.47e-04$	$4.52e-04$	$3.42e-04$	$5.27e-05$	0
	$n = 150,\ R_l = R_s = 0.2$	$3.96e-04$	$7.32e-04$	$1.07e-03$	$7.29e-04$	$1.72e-04$	0
	$n = 200,\ R_l = R_s = 0.001$	$1.86e+01$	$2.30e+01$	$2.40e+01$	$2.25e+01$	$1.23e+00$	0
	$n = 200,\ R_l = R_s = 0.005$	$1.50e+00$	$1.80e+00$	$2.08e+00$	$1.80e+00$	$1.64e-01$	0
	$n = 200,\ R_l = R_s = 0.01$	$5.61e-02$	$7.43e-02$	$9.62e-02$	$7.51e-02$	$1.12e-02$	0
	$n = 200,\ R_l = R_s = 0.05$	$2.07e-04$	$3.02e-04$	$6.24e-04$	$3.23e-04$	$1.08e-04$	0
	$n = 200,\ R_l = R_s = 0.1$	$1.44e-04$	$2.60e-04$	$1.11e-02$	$6.80e-04$	$2.13e-03$	0
	$n = 200,\ R_l = R_s = 0.2$	$1.81e-04$	$4.16e-04$	$4.14e-01$	$1.70e-02$	$8.09e-02$	0
	$n = 250,\ R_l = R_s = 0.001$	$2.06e+01$	$2.41e+01$	$2.57e+01$	$2.41e+01$	$1.11e+00$	0
	$n = 250,\ R_l = R_s = 0.005$	$3.01e+00$	$3.51e+00$	$4.37e+00$	$3.56e+00$	$3.08e-01$	0
	$n = 250,\ R_l = R_s = 0.01$	$1.61e-01$	$1.90e-01$	$2.39e-01$	$1.90e-01$	$1.97e-02$	0
	$n = 250,\ R_l = R_s = 0.05$	$1.74e-04$	$3.82e-04$	$1.16e-03$	$4.22e-04$	$1.82e-04$	0
	$n = 250,\ R_l = R_s = 0.1$	$9.19e-05$	$2.52e-04$	$6.13e-04$	$2.68e-04$	$1.19e-04$	0
	$n = 250,\ R_l = R_s = 0.2$	$1.24e-04$	$3.23e-04$	$3.75e-01$	$1.55e-02$	$7.34e-02$	0
	$n = 300,\ R_l = R_s = 0.001$	$2.39e+01$	$2.54e+01$	$2.67e+01$	$2.54e+01$	$6.77e-01$	0
	$n = 300,\ R_l = R_s = 0.005$	$5.10e+00$	$5.61e+00$	$6.77e+00$	$5.72e+00$	$3.98e-01$	0
	$n = 300,\ R_l = R_s = 0.01$	$3.19e-01$	$4.16e-01$	$5.03e-01$	$4.13e-01$	$4.53e-02$	0
	$n = 300,\ R_l = R_s = 0.05$	$2.73e-04$	$8.23e-04$	$2.46e-02$	$1.79e-03$	$4.67e-03$	0
	$n = 300,\ R_l = R_s = 0.1$	$1.17e-04$	$3.25e-04$	$2.56e-02$	$1.73e-03$	$5.19e-03$	0
	$n = 300,\ R_l = R_s = 0.2$	$9.08e-05$	$3.68e-04$	$9.24e-01$	$3.75e-02$	$1.81e-01$	0

All 25 runs of cMM-EDA solved the problem reliably in the given maximum number of FES. Since the learning rate $\theta_\sigma = 0.00075$ for cMM-EDA is two times smaller than in hMM-EDA, the latter is also significantly faster. The overall fitness evolution of the method is very similar to hMM-EDA.

MBOA, on the other hand, reports a very different convergence behavior. The optimization performance is comparatively fast in early stages of the run, but slows down significantly after $\approx 0.5 \times 10^5$ FES, increases again after $\approx 1.5 \times 10^5$ FES and finally converges towards the optimum at an exponential rate. It was also noted that MBOA is able to explore the binary search space very efficiently. The binary model of the presented median run, for example, converged after only $17,100$ FES, while the remaining $232,800$ FES were used to optimize the continuous model.

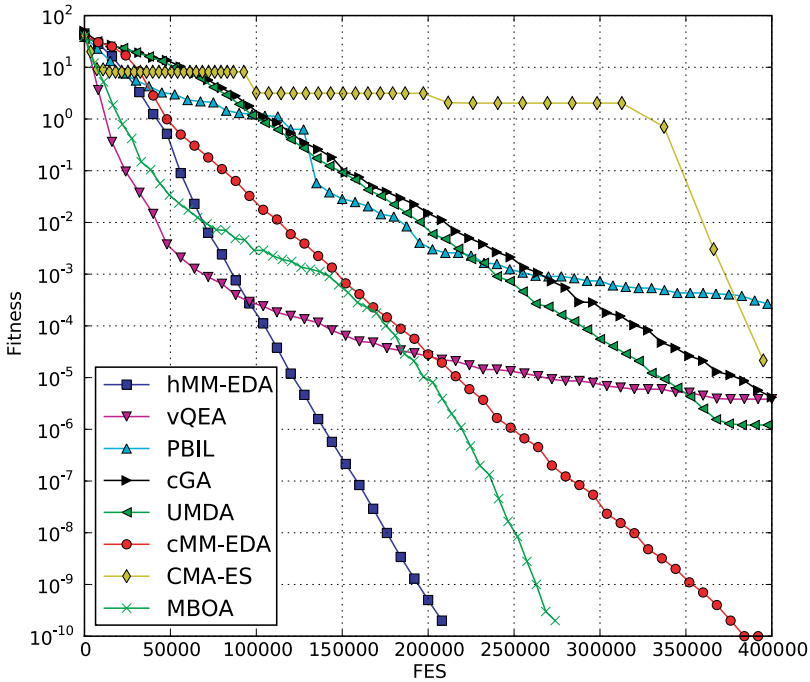


Fig. 6. Evolution of the median fitness for all tested algorithms on the heterogeneous benchmark problem. Results are obtained from 25 independent runs. Due to the mapping from bit values to the continuous domain, the binary methods allow a minimal solution quality of $\approx 10^{-6}$ only. As the continuous optimizers are more precise, the evolution was stopped when the fitness value dropped below 10^{-10}.

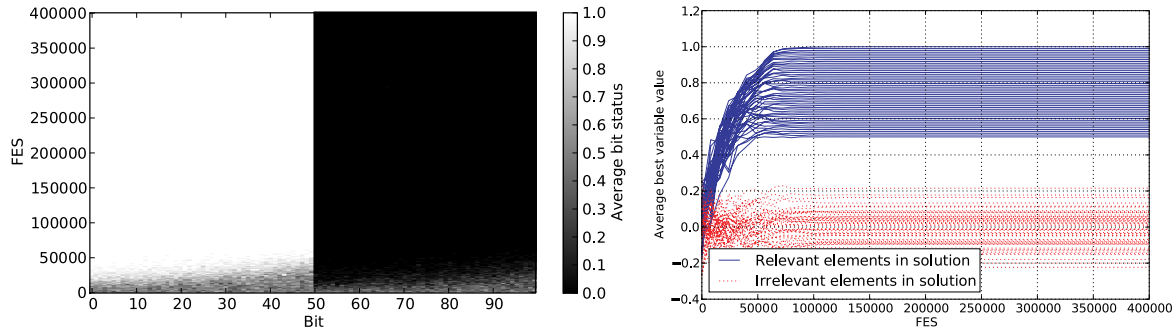
This observation suggests a very competitive performance of MBOA on binary optimization problems, but a comparably slow convergence rate on numerical problems. In Ref. 31, very similar results are reported. Here, several continuous EA, i.e., the Cumulative Step Size Adaptation Evolutionary Strategy (CSA-ES), CMA-ES, the Iterated Density Estimation Evolutionary Algorithm (IDEA) and MBOA were experimentally compared to each other using well-known numerical benchmark problems. Especially on simple uni-modal, separable problems, MBOA was shown to be less competitive than the considered ES. Furthermore, it has been demonstrated in Ref. 31, that although good results could be obtained on separable multi-modal functions, MBOA was not able to optimize any of the tested non-separable functions at all.

Similar to MBOA, also hMM-EDA follows a stepwise optimization strategy of its two models. The optimal binary solution sub-component is discovered after 47,000 FES and the optimization of the continuous component was finished after 170,000 additional FES. The evolution of the mean generational best solutions of the binary and the real solution sub-components are presented in Fig. 7. Results are averaged from the 25 runs of hMM-EDA. The color in Fig. 7a reflects the average bit status of each of the 100 bits at a specific generation, where dark colors denote a status of 0, and white colors a status of 1. The domino convergence due to the different salience of the bits is clearly visible in the figure. Bits corresponding to larger fitness penalties converge earlier during the evolutionary process.

Simultaneously the continuous search space is explored, cf. Fig. 7b. If the binary sub-component was successfully optimized, only the first $\frac{N}{2} = 50$ real-valued elements $r_i \in R$ are considered for further optimization. The last 50 variables are subject to genetic drift and converge randomly.

4.4.3. *Computational cost*

The tested methods are also compared according to their computational cost. The binary-only algorithms are generally fast, since the computational overhead for managing the simple probabilistic



(a) Evolution of binary solution sub-component

(b) Evolution of continuous solution sub-component

Fig. 7. Evolution of binary and continuous solution sub-component using hMM-EDA. Results are averaged from 25 independent runs. Dark colors in (a) correspond to an average bit status of 0, white colors a status of 1. The domino convergence effect due to different salience of the bits is clearly visible in the figure. Simultaneously the continuous search space is optimized, cf. (b). Only the first 50 variables are subject to optimization, if the binary solution was identified correctly. The irrelevant variables are subject to genetic drift and converge randomly.

model is low. For vQEA, a multi-model has to be maintained and updated which slightly increases the computational requirements compared to PBIL, UMDA and cGA. Also, cMM-EDA and hMM-EDA are fast, since their algorithmic structure and the employed models are very similar to vQEA.

The more costly methods are clearly CMA-ES and MBOA. In terms of CMA-ES, a covariance matrix is generated based on the population of the current generation. Also, the sampling of new solutions according to this covariance matrix adds complexity to the algorithm. MBOA is the most costly among the tested methods here. As discussed earlier, its computational overhead is large and it requires significantly more resources than any of the other methods.

In order to demonstrate the computational cost of all the methods, the execution time for each of them is recorded. It is explicitly noted that the execution time is not a very reliable metric to compare algorithms to each other since it has a number of problems. The results depend not only on the used hardware, but also on the used programming language, the programmer's capabilities to optimize the code and the included software libraries. For example, MBOA is based on a C++ implementation, while all other methods are implemented in Java. Nevertheless, such a comparison can be very informative, if the limitations are known and discussed properly.

All methods apply the same configurations as used in the benchmark analysis. Only the stopping criterion was slightly modified: the algorithms perform the maximum number of FES and are not allowed to stop earlier, even if the success criterion is reached. Thus, all methods evaluate the fitness function $N \times 4 \times 10^3 = 4 \times 10^5$ times. The execution time was averaged over five runs. All experiments are performed on the same machine, which is an Intel Core2 Duo CPU, 3.00 GHz, 4 GB RAM, running a 64Bit Ubuntu Linux. The C++ code of MBOA was compiled using GCC 4.3.3 and the highest optimization level.

Table 3 presents the measured CPU time for each method required to finish a single run. As expected, all binary methods are approximately equal in their computational demands, vQEA being slightly slower

Table 3. Execution time of the tested methods when applied on the heterogeneous benchmark problem of size $N = 100$. In brackets the standard deviation is given. The third column presents the required time in relation to the execution time of hMM-EDA. For example, CMA-ES required ≈18.5 more time than hMM-EDA.

Method	Time in sec.	Relative to hMM-EDA
hMM-EDA	8.5 (0.0)	1.0
cMM-EDA	12.0 (0.0)	1.4
vQEA	38.6 (0.9)	4.5
PBIL	18.4 (0.1)	2.2
cGA	26.5 (0.1)	3.1
UMDA	19.8 (0.0)	2.3
CMA-ES	157.5 (5.1)	18.5
MBOA	2740.3 (12.0)	322.6

due to the additional probabilistic models. Also cMM-EDA and hMM-EDA report a fast execution time. The very good results of hMM-EDA are attributed to the conditional model update. Only if the sampled solution is worse than the current attractor does an update occur. Since the algorithm converges before the maximum number of FES is reached, no model update occurs in later stages of the run since the attractor and sampled solution are always identical. This situation results in an impressive execution time. If hMM-EDA is configured with a slower learning rate θ_σ in order to prevent the early convergence of the method, the execution time of the algorithm is close to the one for cMM-EDA.

CMA-ES and MBOA require on average ≈157 and ≈2740 seconds, respectively, to finish the run. Compared to hMM-EDA, these methods are approximately 18 and 322 times slower than hMM-EDA.

Considering the obtained results on the proposed benchmark, hMM-EDA is clearly a highly competitive algorithm among the presented methods. However, a more detailed analysis on a wider range of test functions will have to be performed to provide further statistical evidence for this claim. Nevertheless, the obtained results have demonstrated a promising proof of concept. The hMM-EDA is a light-weight, fast and reliable optimizer with a negligible computational overhead. Practical guidelines have been presented that allow an easy and intuitive configuration of the method.

5. Conclusion and Future Directions

In this paper, it was argued that the simultaneous optimization of different search spaces can be highly beneficial in the context of optimizing SNN. We have presented a literature review on heterogeneous optimization algorithms and provided an example for the successful application of such a method for a SNN based feature selection and classification method. The heterogeneous evolutionary algorithm proposed by the authors in Ref. 16 was named Heterogeneous Multi-Model Estimation of Distributed Algorithm (hMM-EDA).

This paper provides an experimental analysis of hMM-EDA using a synthetic test problem. The benchmark shares similarities with a typical wrapper-based feature selection scenario as observed in QiSNN. It resembles a classical sphere function, see e.g.,[55] for a definition, which was investigated in many EA related studies. We argue that in order to address complex problems in the context of SNN, it is essential for any heterogeneous optimizer to solve this simple test function efficiently.

Eight different optimization techniques were tested and discussed. In comparison to binary-only and continuous-only optimization algorithms, hMM-EDA is highly competitive. Even the much more complex continuous-discrete optimizer MBOA required slightly more FES than hMM-EDA to solve the benchmark reliably. However, the analysis of more test functions is required to provide strong statistical evidence to this claim.

In terms of the computational cost of these methods, it was shown that hMM-EDA requires very little algorithmic overhead, especially in comparison to MBOA and CMA-ES. hMM-EDA is a light-weight, fast and reliable optimization method that requires the configuration of only very few parameters. Practical guidelines for configuring the method were experimentally derived.

Future research might consider the optimization of other benchmark functions. A generalized benchmark suite providing a variety of heterogeneous test functions with known characteristics would allow a more rigorous experimental analysis of the performance of hMM-EDA. Similar suites have been proposed for the testing of numerical optimization algorithms, cf. e.g., the excellent CEC'05 benchmark suite.[55] Other directions might elaborate on the differences between hMM-EDA and co-evolutionary approaches such as presented in Ref. 56. An early discussion of these differences can be found in Ref. 43.

A concrete example for an application of hMM-EDA is the optimization of the specific neural models, such as probabilistic SNN[57] or the computational neuro-genetic modeling (CNGM) presented in Ref. 58. In the latter a gene regulatory network (GRN) affects the spike activity of a SNN. Both the GRN and the SNN have parameters that need to be optimized in order to fit the CNGM to a given data set. A manually fine-tuned CNGM that is capable of reproducing experimental data on long-term potentiation (LTP) occurring in the rat hippocampal dentate gyros was presented in Ref. 59. Using hMM-EDA the optimization process could be automated and the model accuracy increased.

References

1. W. Maass, Computing with spiking neurons, in *Pulsed Neural Networks* (MIT Press, Cambridge, MA, USA, 1999), pp. 55–85.
2. W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity* (Cambridge University Press, Cambridge, MA, 2002).
3. S. Ghosh-Dastidar and H. Adeli, Third generation neural networks: Spiking neural networks, in W. Yu and E. Sanchez (eds.), *Advances in Computational Intelligence*, Vol. 116 of *Advances in Soft Computing*, Springer Berlin/Heidelberg (2009), pp. 167–178.
4. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *Int. J. Neural Syst.* **19**(4) (2009) 295–308.
5. D. Verstraeten, B. Schrauwen and D. Stroobandt, Isolated word recognition using a liquid state machine, in *ESANN* (2005), pp. 435–440.
6. S. M. Bohte, J. N. Kok and J. A. L. Poutré, Error-backpropagation in temporally encoded networks of spiking neurons, *Neurocomputing* **48**(1–4) (2002) 17–37.
7. F. Ponulak, ReSuMe–new supervised learning method for spiking neural networks, *Tech. Rep.*, Institute of Control and Information Engineering, Poznań University of Technology, Poznań, Poland (2005).
8. A. Knoblauch, Neural associative memory for brain modeling and information retrieval. *Inf. Process. Lett.* **95**(6) (2005) 537–544.
9. N. Iannella and L. Kindermann, Finding iterative roots with a spiking neural network, *Information Processing Letters* **95**(6) (2005) 545–551.
10. J. Iglesias and A. E. Villa, Emergence of preferred firing sequences in large spiking neural networks during simulated neuronal development, *Int. J. Neural Syst.* **18**(4) (2008) 267–277.
11. J. L. Rosselló, V. Canals, A. Morro and J. Verd, Chaos-based mixed signal implementation of spiking neurons. *Int. J. Neural Syst.* **19**(6) (2009) 465–471.
12. S. Ghosh-Dastidar and H. Adeli, Improved spiking neural networks for EEG classification and epilepsy and seizure detection, *Integr. Comput.-Aided Eng.* **14**(3) (2007) 187–212.
13. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural Networks* **22**(10) (2009) 1419–1431.
14. N. Pavlidis, D. K. Tasoulis, V. P. Plagianakos, G. Nikiforidis and M. N. Vrahatis, Spiking neural network training using evolutionary algorithms, in *Proc. of International Joint Conference on Neural Networks (IJCNN'05)* (2005), pp. 2190–2194.
15. C. Taylor and A. Agah, Evolving neural network topologies for object recognition, in *Automation Congress, 2006. WAC'06*, IEEE, Budapest, Hungary (2006), pp. 1–6.
16. S. Schliebs, M. Defoin-Platel, S. Worner and N. Kasabov, Integrated feature and parameter optimization for an evolving spiking neural network: Exploring heterogeneous probabilistic models, *Neural Networks* **22**(5–6) (2009) 623–632.
17. K. Hintz and J. Spofford, Evolving a neural network, in *5th IEEE International Symposium on Intelligent Control*, Vol. 1 (1990), pp. 479–484.
18. V. Maniezzo, Genetic evolution of the topology and weight distribution of neural networks, *IEEE Transactions on Neural Networks* **5**(1) (1994) 39–53.
19. F. Leung, H. Lam, S. Ling and P. Tam, Tuning of the structure and parameters of a neural network using an improved genetic algorithm, *IEEE Transactions on Neural Networks* **14**(1) (2003) 79–88.
20. D. White and P. A. Ligomenides, GANNet: A genetic algorithm for optimizing topology and weights in neural network design, in *IWANN'93: Proceedings of the International Workshop on Artificial Neural Networks*, Springer-Verlag, London, UK (1993), pp. 322–327.
21. E. Alba, J. F. A. Montes and J. M. Troya, Full automatic ANN design: A genetic approach, in *IWANN'93: Proceedings of the International Workshop on Artificial Neural Networks*, Springer-Verlag, London, UK (1993), pp. 399–404.
22. S. Oliker, M. Furst and O. Maimon, Design architectures and training of neural networks with a distributed genetic algorithm, in *IEEE International Conference on Neural Networks*, Vol. 1 (1993), pp. 199–202.
23. S. Hung and H. Adeli, A parallel genetic/neural network learning algorithm for MIMD shared memory machines, *Neural Networks, IEEE Transactions* **5**(6) (1994) 900–909.
24. M. Valko, N. C. Marques and M. Castelani, Evolutionary feature selection for spiking neural network pattern classifiers, in B. *et al.* (ed.), *Proceedings of 2005 Portuguese Conference on Artificial Intelligence*, IEEE Press (2005), pp. 24–32.
25. M. Castellani and N. Marques, FeaSANNT — An embedded evolutionary feature selection approach for neural network classifiers, *VIMation Journal* **1** (2008) 46–53.
26. M. Castellani and H. Rowlands, Evolutionary artificial neural network design and training for wood veneer classification, *Engineering Applications of Artificial Intelligence* **22**(4–5) (2009) 732–741.
27. M. Castellani, ANNE — A new algorithm for evolution of artificial neural network classifier systems, in *IEEE Congress on Evolutionary Computation, CEC'06* (2006), pp. 3294–3301.
28. D. Rivero, J. Dorado, E. Fernández-Blanco and A. Pazos, A genetic algorithm for ANN design, training and simplification, in *IWANN'09: Proceedings of the 10th International Work-Conference on Artificial*

Neural Networks, Springer-Verlag, Berlin, Heidelberg (2009), pp. 391–398.
29. B. A. Garro, H. Sossa and R. A. Vazquez, Design of artificial neural networks using a modified particle swarm optimization algorithm, *International Joint Conference on Neural Networks, IEEE - INNS - ENNS* (2009), pp. 938–945.
30. J. Ocenasek and J. Schwarz, Estimation of distribution algorithm for mixed continuous-discrete optimization problems, in *2nd Euro-International Symposium on Computational Intelligence*, IOS Press, Kosice, Slovakia (2002), pp. 227–232.
31. S. Kern, S. Müller, N. Hansen, D. Büche, J. Ocenasek and P. Koumoutsakos, Learning probability distributions in continuous evolutionary algorithms — a comparative review, *Natural Computing* **3**(1) (2004) 77–112.
32. J. Ocenasek, Parallel estimation of distribution algorithms. Ph.D. thesis, Faculty of Information Technology, Brno University of Technology, Brno, Czech Rep. (2002).
33. K. Socha, ACO for continuous and mixed-variable optimization, *Ant Colony, Optimization and Swarm Intelligence* (2004), pp. 25–36.
34. O. Cordón, I. Fernández de Viana, F. Herrera and L. Moreno, A new ACO model integrating evolutionary computation concepts: The best-worst ant system, in M. Dorigo, M. Middendoff and T. Stützle (eds.), *From Ant Colonies to Artificial Ants: Proceedings of the Second International Workshop on Ant Algorithms*, Springer, Brussels, Belgium (2000), pp. 22–29.
35. N. Monmarché, E. Ramat, G. Dromel, M. Slimane and G. Venturini, On the similarities between AS, BSC and PBIL: Toward the birth of a new meta-heuristic, *Rapport Interne 215*, Laboratoire d'Informatique de l'Université de Tours, E3i Tours (1999).
36. S. Schliebs, M. Defoin-Platel and N. Kasabov, Integrated feature and parameter optimization for an evolving spiking neural network, in M. Köppen, N. K. Kasabov and G. G. Coghill (eds.), *Advances in Neuro-Information Processing, 15th International Conference*, Vol. 5506 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany (2009), pp. 1229–1236.
37. M. Defoin-Platel, S. Schliebs and N. Kasabov, A versatile quantum-inspired evolutionary algorithm, in *IEEE Congress on Evolutionary Computation, CEC'07*, IEEE Press, Singapore (2007), pp. 423–430.
38. S. G. Wysoski, L. Benuskova and N. Kasabov, Online learning with structural adaptation in a network of spiking neurons for visual pattern recognition, in *Artificial Neural Networks ICANN 2006*, Springer, Berlin/Heidelberg (2006), pp. 61–70.
39. M. Defoin-Platel, S. Schliebs and N. Kasabov, Quantum-inspired evolutionary algorithm: A multimodel EDA, *Evolutionary Computation, IEEE Transactions* **13**(6) (2009) 1218–1232.
40. R. Kohavi and G. H. John, Wrappers for feature subset selection, *Artificial Intelligence* **97**(1–2) (1997) 273–324.
41. S. Schliebs, M. Defoin-Platel, S. Worner and N. Kasabov, Quantum-inspired feature and parameter optimisation of evolving spiking neural networks with a case study from ecological modeling, in *International Joint Conference on Neural Networks, IEEE - INNS - ENNS*, IEEE Computer Society, Los Alamitos, CA, USA (2009), pp. 2833–2840.
42. S. Schliebs, M. Defoin-Platel and N. Kasabov, Analyzing the dynamics of the simultaneous feature and parameter optimization of an evolving spiking neural network, in *International Joint Conference on Neural Networks, IEEE - INNS - ENNS*, IEEE Computer Society, Barcelona, Spain (2010).
43. S. Schliebs, *Heterogeneous Probabilistic Models for Optimisation and Modelling of Evolving Spiking Neural Networks*. Ph.D. thesis, Auckland University of Technology (2010). http://hdl.handle.net/10292/963.
44. P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, Boston (2002).
45. J. Geweke, Efficient simulation from the multivariate normal and student-t distributions subject to linear constraints and the evaluation of constraint probabilities, in *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, American Statistical Association, New York, Seattle, Washington (1991), pp. 571–578.
46. K.-H. Han and J.-H. Kim, On setting the parameters of quantum-inspired evolutionary algorithm for practical application, in *Congress on Evolutionary Computation, CEC'03*, Vol. 1, IEEE Press, Canberra, Australia (2003), pp. 178–194.
47. D. Thierens, D. Goldberg and A. Pereira, Domino convergence, drift, and the temporal-salience structure of problems, in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence*, IEEE Press (1998), pp. 535–540.
48. W. M. Rudnick, *Genetic Algorithms and Fitness Variance with an Application to the Automated Design of Artificial Neural Networks*. Ph.D. thesis, Oregon Graduate Institute of Science & Technology, Beaverton, OR, USA (1992).
49. K.-H. Han and J.-H. Kim, Quantum-inspired evolutionary algorithms with a new termination criterion, H_ϵ gate, and two phase scheme, *IEEE Transactions on Evolutionary Computation* **8**(2) (2004) 156–169.
50. H. Mühlenbein and G. Paass, From recombination of genes to the estimation of distributions I. binary parameters, in *PPSN* (1996), pp. 178–187.

51. S. Baluja, Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning, *Tech. Rep. CMU-CS-94-163*, Carnegie Mellon University, Pittsburgh, PA (1994).
52. G. R. Harik, F. G. Lobo and D. E. Goldberg, The compact genetic algorithm, *IEEE Transactions on Evolutionary Computation* **3**(4) (1999) 287–297.
53. Z. Michalewicz and C. Z. Janikow, Genetic algorithms for numerical optimization, *Statistics and Computing* **1**(2) (1991) 75–91.
54. A. Auger and N. Hansen, Performance evaluation of an advanced local search evolutionary algorithm, in *IEEE Congress on Evolutionary Computation*, IEEE Press, Vol. 2 (2005), pp. 1777–1784.
55. P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger and S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real parameter optimization, *Tech. Rep.*, Nanyang Technological University, Singapore (2005).
56. M. A. Potter and K. A. D. Jong, Cooperative coevolution, An architecture for evolving coadapted subcomponents, *Evolutionary Computation* **8** (2000) 1–29.
57. N. Kasabov, To spike or not to spike: A probabilistic spiking neuron model, *Neural Networks* **23**(1) (2010) 16–19.
58. L. Benuskova and N. Kasabov, *Computational Neurogenetic Modeling*, Springer, NY (2007).
59. L. Benuskova, V. Jain, S. G. Wysoski and N. Kasabov, Computational neurogenetic modeling: A pathway to new discoveries in genetic neuroscience, *Intl. Journal of Neural Systems* **16**(3) (2006) 215–227.