Journal of Machine Learning Research 21 (2020) 1-31 Submitted 10/16; Revised 4/20; Published 4/20

Harmless Overfitting: Using Denoising Autoencoders in Estimation of Distribution Algorithms

Malte Probst* MALTE.PROBST@HONDA-RI.DE
Honda Research Institute EU
Carl-Legien-Str. 30, 63073 Offenbach, Germany

Franz Rothlauf ROTHLAUF@UNI-MAINZ.DE
Information Systems and Business Administration
Johannes Gutenberg-Universität Mainz
Jakob-Welder-Weg 9, 55128 Mainz, Germany

Editor: Francis Bach

Abstract

Estimation of Distribution Algorithms (EDAs) are metaheuristics where learning a model and sampling new solutions replaces the variation operators recombination and mutation used in standard Genetic Algorithms. The choice of these models as well as the corresponding training processes are subject to the bias/variance tradeoff, also known as under- and overfitting: simple models cannot capture complex interactions between problem variables, whereas complex models are susceptible to modeling random noise. This paper suggests using Denoising Autoencoders (DAEs) as generative models within EDAs (DAE-EDA). The resulting DAE-EDA is able to model complex probability distributions. Furthermore, overfitting is less harmful, since DAEs overfit by learning the identity function. This overfitting behavior introduces unbiased random noise into the samples, which is no major problem for the EDA but just leads to higher population diversity. As a result, DAE-EDA runs for more generations before convergence and searches promising parts of the solution space more thoroughly. We study the performance of DAE-EDA on several combinatorial single-objective optimization problems. In comparison to the Bayesian Optimization Algorithm, DAE-EDA requires a similar number of evaluations of the objective function but is much faster and can be parallelized efficiently, making it the preferred choice especially for large and difficult optimization problems.

Keywords: denoising autoencoder; estimation of distribution algorithm; overfitting; combinatorial optimization; neural networks

1. Introduction

Estimation of Distribution Algorithms (EDAs) are metaheuristics for combinatorial optimization which evolve a distribution over potential solutions (Mühlenbein and Paaß, 1996; Larrañaga and Lozano, 2002). As in Genetic Algorithms (GAs, Holland, 2001; Goldberg, 1989), a single solution is often referred to as an individual, the current sample of individuals is called a population, and one iterative step of evolution is considered to be a generation. In each generation, an EDA selects a subset of high-quality solutions from the population. It updates a probabilistic model to approximate the probability distribution of the variables in

∗. This work was done while the author was a member of the University of Mainz

this subset of the population. In the next generation, this model is then used to sample new candidate solutions. The underlying idea is that these sampled candidate solutions have properties similar to those of the selected high-quality solutions. Hence, it is important that the model's approximation of the dependency structure is precise[1].

A central challenge when learning a model is the bias/variance tradeoff, also referred to as under- and overfitting (see, e.g., Geman et al., 1992). This tradeoff requires matching the capacity and complexity of the model to the structure of the training data, both when selecting a model, and, subsequently, when the approximation process estimates the parameters of the model. Simple models often suffer from a strong bias since they may lack the capacity to capture the dependencies between variables properly. In contrast, complex models can show high variance, that is, they can overfit: Indeed, for a finite number of training examples, there is always sampling noise and if the approximation process captures the structure of this sampling noise in the model, then the model is overfitting. Both underfitting and overfitting can deteriorate the quality of a probabilistic model (Cawley and Talbot, 2010).

EDAs require models that accurately capture the dependencies between problem variables within the selected population and allow them to sample new solutions with similar properties and quality (Radetic and Pelikan, 2010). Simple, univariate models often suffer from underfitting, i.e., they are unable to capture the relevant dependencies. This can lead to an exponential number of evaluations of the objective function for growing problem sizes (Pelikan et al., 1999; Pelikan, 2005a). Hence, for more relevant and complex problems, multivariate models are usually used. Indeed, such models have the capacity to approximate more complex interactions between variables. However, they are not only susceptible to underfitting (in the case that the approximation process terminates before all relevant dependencies are captured); they are also affected by overfitting, that is, they learn sampling noise. There are a number of mechanisms for limiting the amount of overfitting in complex models, such as adding penalties for overly complex models, stopping the approximation process early, or using Bayesian statistics (see, e.g., Bishop, 2006, Chapter 1.1.; Murphy, 2012, Chapter 5). However, even these techniques often cannot avoid overfitting entirely, which decreases the model quality and, in turn, the EDA's optimization performance (Wu and Shapiro, 2006; Santana et al., 2008; Chen et al., 2010).

This paper suggests using Denoising Autoencoders (DAE, Vincent et al., 2008) as probabilistic EDA models and studies the properties and performance of the resulting DAE-EDA. DAEs are neural networks which map n input variables to n output variables via a hidden representation, which captures the dependency structure of the input data. Since DAEs implicitly model the probability distribution of the input data, sampling from this distribution becomes possible (Bengio et al., 2013). In this paper, we assess the performance of DAE-EDA on multiple combinatorial optimization benchmark problems, i.e., problems with binary decision variables. We report the number of evaluations and required CPU

1. Throughout the paper, we use the term "dependency structure" when referring to dependencies (correlations) between decision variables. For example, such a dependency structure could be modeled as a graph where the nodes correspond to problem variables, and the connections correspond to conditional dependencies between them. The structure or topology of such a graph could take various forms, e.g. a single tree, multiple trees, a chain, or others, depending on the problem.

times. For comparison, we include results for the Bayesian Optimization Algorithm (BOA, Pelikan et al., 1999; Pelikan, 2005a), which is still a state-of-the-art EDA approach.

We find that DAE-EDA has a model quality similar to BOA, however, it is faster and can be parallelized more easily. The experiments reveal a major reason for the good performance on the set of tested combinatorial optimization problems, specifically on the difficult deceptive problems, which guide local search optimizers away from the global optimum: A slightly overfitted DAE is beneficial for the diversity of the population since it does not repeat sampling noise but is biased towards random solutions. When overfitting, a DAE tends to learn the trivial identity function as its hidden representation, gradually overlaying and replacing the learned, non-trivial hidden representation capturing problem structure. Therefore, an overfitted DAE does not approximate the structure of the sampling noise of the training data, but tends to simply replicate inputs as outputs. Sampling from a DAE is done by initializing the DAE's input with random values, which have high diversity with respect to the solution space (meaning that solutions are sampled randomly from all solutions of the solution space). Since an overfitted DAE partly learned the identify function, which replicates inputs as outputs, it will carry over some of this diversity to the generated samples, instead of replicating sampling noise specific to the training set. This particular way of DAE overfitting in combination with the randomly initialized sampling process leads to higher population diversity throughout an EDA run: Like other population-based metaheuristics, EDAs usually reduce the population diversity continuously, due to the selection step in each generation. Using a DAE as EDA model leads to a lower loss of diversity since sampling from the DAE is biased towards random solutions. As a result, populations sampled from a DAE are slightly more diverse than the training population, allowing an EDA to keep population diversity high, to run for more generations, and to find better solutions.

Section 2.1 introduces EDAs, describes the bias/variance tradeoff in EDAs, and reviews the reference algorithm BOA. In Section 2.2, we describe DAEs, and propose DAE-EDA. Section 2.3 gives a brief literature review on DAEs in combinatorial optimization. Section 3 presents test problems, the experimental setup, and results. Furthermore, it provides an in-depth analysis of the overfitting behavior of a DAE and its influence on the EDA. We discuss the findings in Section 4 and finish in Section 5 with concluding remarks.

2. Preliminaries

2.1. Estimation of Distribution Algorithms

We introduce EDAs, discuss the bias/variance tradeoff in EDAs, and briefly introduce BOA.

2.1.1. Principles

EDAs (Mühlenbein and Paaß, 1996; Larrañaga and Lozano, 2002) are well-established black-box optimization methods for combinatorial optimization problems (Lozano et al., 2006; Larrañaga et al., 2012). Algorithm 1 outlines their basic functionality. EDAs "evolve" a distribution D over the search space of all possible solutions, over the course of multiple generations. In each generation, an EDA samples a set $P_{candidates}$ from D. It then calculates the objective value of all individuals in $P_{candidates}$ and selects a subset of high-quality solutions P_{t+1}. Subsequently, the distribution D_t is updated using P_{t+1}. The updated dis-

tribution D_{t+1} can, in turn, be used to sample new candidates in the next iteration. This loop continues until the distribution D_t does not change any more (population P is converged), or some other termination criterion holds.[2] The underlying assumption of an EDA is that with each update, the distribution D models the hidden properties of the objective function f better, and that sampling D yields previously unknown high-quality individuals.

EDAs can also be viewed as instances of Information-Geometric Optimization algorithms (IGO, Ollivier et al., 2017). In a nutshell, IGOs strive to follow the natural gradient of the objective function in the parameter space θ of the distribution D while strictly adhering to invariance principles (for details, see Ollivier et al. 2017).

Algorithm 1 Estimation of Distribution Algorithm

1: **Initialize** Distribution D_0, $t = 0$
2: **while** not converged **do**
3: $P_{candidates} \leftarrow$ **Sample** new candidate solutions from D_t
4: $P_{t+1} \leftarrow$ **Select** high-quality solutions from $P_{candidates}$ based on their objective value
5: $D_{t+1} \leftarrow$ **Update** D_t using P_{t+1}
6: $t \leftarrow t + 1$
7: **end while**

EDAs differ in their choice of the model to approximate D. Simple models like UMDA (Mühlenbein and Paaß, 1996) or PBIL (Baluja, 1994) use a probabilistic model that corresponds to the product of univariate marginal distributions. They use vectors with activation probabilities for each decision variable but neglect dependencies between variables. Slightly more complex models like the Dependency Tree Algorithm (Baluja and Davies, 1997) use pairwise dependencies between variables modeled as trees or forests (Pelikan and Mühlenbein, 1999). More complex dependencies can be captured by models that use multivariate interactions, like ECGA (Harik et al., 2006) or BOA (Pelikan et al., 1999). Common models for capturing complex dependencies are probabilistic graphical models with directed edges (like Bayesian networks) or undirected edges (like Markov random fields) (Larrañaga et al., 2012). Hence, model building consists of, first, finding a network structure that matches the problem structure well and, second, estimating the model's parameters. Usually, the computational effort to build the model increases with its complexity and representational power.

2.1.2. Bias/Variance Tradeoff in EDAs

As indicated earlier, the modeling/distribution update step in EDAs is subject to the bias/variance tradeoff. Univariate EDA models assume that the probability distribution is completely factorized. Hence, such models are unable to properly capture the dependencies between the variables of more complicated problems with interdependencies between variables (see, e.g., Radetic and Pelikan, 2010). Due to this underfitting, the use of univariate models can cause exponential growth of the required number of fitness evaluations for

2. In the Evolutionary Algorithms community, EDAs are often defined by an "orthogonal" description, that emphasizes the population-based nature: Here, the EDA's main loop iterates over successive populations P, from which it selects high-quality solutions, builds a new model M, and samples new candidates $P_{candidates}$ from the model.

growing problem sizes (Pelikan et al., 1999; Pelikan, 2005a). Hence, multivariate models are often a better choice for complex optimization problems.

However, complex models are subject to overfitting (see, e.g., Santana et al., 2008; Chen et al., 2010; Bishop, 2006; Murphy, 2012). Due to limited number of samples drawn by the EDA in each generation, model building is always affected by sampling noise. When updating the distribution—that is, fitting the model to the selected subset of sampled solutions—it is impossible to decide whether observed correlations are due to the inherent problem structure or due to sampling noise. Hence, the learned model will include both kinds of dependencies: those due to the problem structure and those due to sampling noise. This can be harmful for optimization, since variables which are independent in the problem may be modeled as correlated variables due to sampling noise.

There are multiple ways to control overfitting. A straightforward one is to increase the population size. On increasing the population size, correlations due to sampling noise will become weaker (see, e.g., Goldberg et al., 1992).
Alternatively, early stopping can be applied (Bishop, 2006, Chapter 5.2). When using early stopping, the quality of a model is not only evaluated on the training set, but also on an independent validation set. The rationale is that both sets are sampled from the same distribution, however the sampling noise is different. As soon as a model stops improving on the validation set, but keeps improving on the training set, there is a risk of overfitting to the training data. However, the reverse is not necessarily true: a model may already overfit the training data even if it is still improving on the validation data (Hinton, 2012). Therefore, there is no unique, optimal time point for stopping early.

Another way of limiting overfitting is regularization, which usually includes a term in the approximation objective that penalizes overly complex models. This can be done directly, via a scoring function that prefers simple models to complicated ones (see, e.g., Schwarz, 1978), or indirectly, by imposing constraints on the values of the learned parameters, thereby limiting the model complexity (see, e.g., Murphy, 2012). In both cases, it is necessary to specify the strength of regularization. This is not a straightforward task, given that there is usually no previous information on problem complexity (Bishop, 2006, Chapter 1.1). Even when applying approaches like early stopping or regularization in EDAs to avoid overfitting, overfitting can still be harmful to the optimization process (Wu and Shapiro, 2006).

A different approach to keep overfitting at bay is to use Bayesian statistics, where the goal is to estimate a distribution over the parameters, rather than a point estimate (see, e.g, Murphy, 2012, Chapter 5).[3] However, it is often computationally infeasible to compute the full posterior distribution over the parameters for larger models involving many parameters, and instead a point estimate (e.g., the maximum a posteriori (MAP) estimate) has to be used, which is, again, prone to overfitting.

2.1.3. Bayesian Optimization Algorithm

The Bayesian Optimization Algorithm (Pelikan et al., 1999) is a state-of-the-art EDA for combinatorial optimization problems (Pelikan and Goldberg, 2003; Pelikan, 2005a, 2008;

3. Note that the Bayesian Optimization Algorithm mentioned in the next Section is *not* using Bayesian statistics in the above sense.

Abdollahzadeh et al., 2012)[4]. BOA uses a Bayesian network for modeling dependencies between decision variables. Decision variables correspond to nodes, and dependencies between variables correspond to directed edges. As the number of possible network topologies grows exponentially with the number of nodes, BOA uses a greedy construction heuristic to find a network structure G that models the training data. Starting from an unconnected (empty) network, BOA evaluates all possible additional edges, adds the one that maximally increases the fit between the model and the selected individuals, and repeats this process until no more edges can be added. The fit between the model and the selected individuals is measured using the Bayesian Information Criterion (BIC, Schwarz, 1978). The resulting BIC score guides the structural learning of the probabilistic model. BIC calculates the conditional entropy of nodes given their parent nodes and includes a term which reduces overfitting by penalizing complex models. It can be calculated independently for all nodes. If an edge is added to the Bayesian network, the change of the BIC can be computed quickly. BOA's greedy network construction algorithm adds the edge with the largest BIC gain until no more edges can be added. Edge additions resulting in cycles are not allowed.

After the network structure has been learned (that is, the Bayesian model has been fitted to a population of solutions), the parameters of the model can be determined by calculating the conditional distributions from the data. Once the model structure and conditional distributions are available, BOA can produce new candidate solutions by drawing random values for all nodes in topological order.

BOA has been succeeded by the Hierarchical Bayesian Optimization Algorithm (hBOA), which outperforms BOA on complex, hierarchical problems (Pelikan, 2005b). hBOA extends BOA by using a diversity-preserving selection mechanism, as well as exploiting local structures in Bayesian networks. In this work, we decided to use the non-extended version of BOA, for the following reasons: First, both extensions could be applied to DAE-EDA as well (directly or in principle, see discussion in Section 4). Second, in our analysis, we want to isolate effects of the respective probabilistic models of the EDAs, which is not possible if the analyzed system (the EDA) gets overly complex.

2.2. Using Denoising Autoencoders as EDA Models: DAE-EDA

We describe how to train an autoencoder (AE), introduce the Denoising AE, and illustrate how to sample new solutions. Section 2.2.4 summarizes the proposed DAE-EDA.

2.2.1. AE Structure and Training Procedure

An AE is a multi-layer perceptron, which is one of the basic types of neural networks (see, e.g., Murphy, 2012, Chapter 16.5). AEs have been used for dimensionality reduction and are one of the building blocks for deep learning (Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Bengio, 2009).

An AE consists of one visible layer $\mathbf{x}$, at least one hidden layer $\mathbf{h}$, and one output layer $\mathbf{z}$ (see Figure 1 for the case of one hidden layer, which we will consider throughout this

4. There are a number of BOA variants, each of which has its specific up- and downsides for specific problem types. However, all of them follow the basic BOA principles laid out in this section, which, in this sense, is still "state-of-the-art" (also see Section 4).

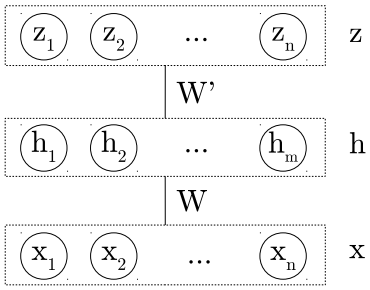


Figure 1: Autoencoder with a single hidden layer $\mathbf{h}$.

paper)[5]. The visible neurons x_i, $i \in \{1, \ldots, n\}$ can hold a data vector of length n from the training data. For example, in the EDA context, each x_i represents a decision variable. The hidden neurons h_j, $j \in \{1, \ldots, m\}$ are a non-linear representation of the input. The output neurons z_i, $i \in \{1, \ldots, n\}$ hold the AE's *reconstruction* of the input. The weights $\mathbf{W}$ and $\mathbf{W}'$ fully connect $\mathbf{x}$ to $\mathbf{h}$ and $\mathbf{h}$ to $\mathbf{z}$, respectively.

An AE defines two deterministic functions: First, the encoding function $\mathbf{h} = c(\mathbf{x}; \theta)$, which maps a given input $\mathbf{x} \in \{0,1\}^n$, to the hidden layer $\mathbf{h} \in \{0,1\}^m$, with parameters θ and $n, m \in \mathbb{N}$. Second, the decoding function $\mathbf{z} = f(\mathbf{h}; \theta')$, which maps $\mathbf{h}$ back to the reconstruction $\mathbf{z} \in \{0,1\}^n$ in the input space. The training objective of an AE is to find parameters θ, θ' that minimize the *reconstruction error* $\mathrm{Err}_{\theta,\theta'}(\mathbf{x}, \mathbf{z})$, which is calculated based on the differences between $\mathbf{x}$ and $\mathbf{z}$ for all examples $\mathbf{x}^i, i \in \{1, \ldots, \tau\}$ in the training set.

$$\theta, \theta' := \underset{\theta,\theta'}{\operatorname{argmin}} \frac{1}{\tau} \sum_{i=1}^{\tau} \mathrm{Err}_{\theta,\theta'}(\mathbf{x}^i, \mathbf{z}^i). \tag{1}$$

Common choices for $\mathrm{Err}_{\theta,\theta'}(\mathbf{x}, \mathbf{z})$ are the mean squared error function $\mathrm{Err}_{\theta,\theta'}(\mathbf{x}, \mathbf{z}) = ||\mathbf{x} - \mathbf{z}||^2$ or the cross entropy function $\mathrm{Err}_{\theta,\theta'}(\mathbf{x}, \mathbf{z}) = -\sum_{k=1}^{n} [x_k * \log(z_k) + (1 - x_k) * \log(1 - z_k)]$.

Encoding and decoding functions are usually chosen as $c(\mathbf{x}) = \mathrm{sigm}(\mathbf{x} * \mathbf{W} + \mathbf{b}^h)$ and $f(\mathbf{h}) = \mathrm{sigm}(\mathbf{h} * \mathbf{W}' + \mathbf{b}^z)$, where $\mathrm{sigm}(x) = \frac{1}{1+e^{-x}}$ is the logistic function, $\mathbf{W}$ and $\mathbf{W}'$ are weight matrices of size $(n \times m)$ and $(m \times n)$, respectively, and $\mathbf{b}^h \in \mathbb{R}^m$, $\mathbf{b}^z \in \mathbb{R}^n$ are biases which work as offsets. Often, $\mathbf{W}$ and $\mathbf{W}'$ are *tied*, that is, $\mathbf{W}' = \mathbf{W}^\top$. Then, the AEs configurable parameters are $\theta = \{\mathbf{W}, \mathbf{b}^h, \mathbf{b}^z\}$, which can be trained by gradient descent.

2.2.2. Denoising AE

If the representational power of the hidden layer $\mathbf{h}$ is large enough (that is, if m is not too small), a trivial way to solve Equation 1 is to learn the identity function where each x_i is directly mapped to the corresponding z_i (Alain and Bengio, 2014). Regularization can be used to force the model to learn a more useful representation (Bengio, 2009; Alain and Bengio, 2014). One example of a regularized AE is the Denoising Autoencoder (DAE, Vincent et al., 2008). In a DAE, each training example $\mathbf{x}$ is corrupted by a stochastic

5. We use the following notation: x denotes a scalar value, $\mathbf{x}$ denotes a vector of scalars, $\mathbf{X}$ denotes a matrix of scalars.

Algorithm 2 Pseudo code for training an AE (when training a DAE, replace $\mathbf{x}^i$ with $q(\mathbf{x}^i)$ in line 5)

1: **Initialize** $\theta = \{\mathbf{W}, \mathbf{b}^h, \mathbf{b}^z\}$ randomly
2: **Set** $0 < \alpha < 1$, e.g., $\alpha = 0.1$
3: **while** not converged **do**
4: **for each** example i in the training set **do**
5: $\mathbf{h} = c(\mathbf{x}^i; \theta)$
6: $\mathbf{z} = f(\mathbf{h}; \theta)$
7: $\theta := \theta - \alpha * \frac{\partial \text{Err}_\theta(\mathbf{x}^i, \mathbf{z})}{\partial \theta}$
8: **end for**
9: **end while**

mapping $\hat{\mathbf{x}} = q(\mathbf{x})$, that is, we add random noise. Subsequently, a DAE calculates the reconstruction of the corrupted input using the encoding and decoding function as $\mathbf{z} = f(c(\hat{\mathbf{x}}))$. As with the original AE, the parameters are updated in the direction of $\frac{\partial \text{Err}(\mathbf{x}, \mathbf{z})}{\partial \theta}$ (see Algorithm 2). Hence, the DAE tries to reconstruct $\mathbf{x}$ rather than $\hat{\mathbf{x}}$.[6] DAEs have the additional property that the noise introduced by the corruption process $q(\cdot)$ makes the model more robust to partially destroyed inputs (Vincent et al., 2008).

2.2.3. Sampling a DAE

Standard AEs do not include (and do not need) a sampling process to generate new solutions. However, recent work has shown that some variants of AEs, including the DAE, implicitly capture the probability distribution of the training data. Consequently, multiple sampling processes have been suggested and empirically validated (for an overview, see Bengio et al., 2013).

For the current study, we adopted the sampling process proposed by Bengio et al. (2013), which is a general approach and comes with a theoretical justification. Given a data-generating distribution, $P(\mathbf{x})$, a corruption process $\hat{\mathbf{x}} = q(\mathbf{x})$ and a DAE that has been trained to reconstruct x from $\hat{\mathbf{x}}$, the sampling process is as follows (see Algorithm 3): First, we randomly initialize a sample $\mathbf{x} \in \{0, 1\}^n$. Then, for s sampling steps, we iteratively corrupt the sample using the corruption process $\hat{\mathbf{x}} = q(\mathbf{x})$, use the trained DAE to reconstruct the input $\mathbf{z} = f(c(\hat{\mathbf{x}}))$, and set $\mathbf{x} := \mathbf{z}$. After s sampling steps, we use $\mathbf{x}$ as a sample from the DAE.

Bengio et al. (2013) showed that this sampling process returns samples from the DAE's approximation of the data-generating distribution, that is, the training data.

6. On first glance, the DAE's sample corruption looks similar to the mutation operator of Genetic Algorithms. However, in a DAE, the corrupted data point $\hat{\mathbf{x}}$ is never directly used as a sample, but is first reconstructed/denoised by f and c. This is in contrast to the genetic mutation operator, where the random mutations are carried over to the next generation. Second, the level of corruption in a DAE is typically on a much higher absolute level, i.e., more bits are distorted.

Algorithm 3 Pseudo code for sampling a DAE

1: **Given** the trained DAE's $\theta = \{\mathbf{W}, \mathbf{b}^c, \mathbf{b}^f\}$, its reconstruction function $f(c(\hat{\mathbf{x}}))$, and the corruption process $q(\mathbf{x})$
2: **Initialize** $\mathbf{x} \in [0, 1]^n$ randomly
3: **for a fixed number** s **of sampling steps do**
4: $\quad \hat{\mathbf{x}} = q(\mathbf{x})$
5: $\quad \mathbf{z} = f(c(\hat{\mathbf{x}}))$
6: $\quad \mathbf{x} := \mathbf{z}$
7: **end for**
8: Use $\mathbf{x}$ as a sample from the DAE

2.2.4. DAE-EDA

We describe how we use a DAE within an EDA, and how our practical implementation slightly diverges from the generalized description in Algorithm 1 in two aspects. Recall that an EDA uses a probabilistic model D to capture the properties of high-quality solutions and sample new candidate solutions $P_{candidates}$. Since we assume no prior knowledge, D_0 is initialized to the uniform distribution over the search space.

In each EDA generation the EDA uses the current model D_t to generate candidate solutions $P_{candidates}$, as in line 3 of Algorithm 1. In DAE-EDA, this is done by sampling from the DAE, and using the samples as $P_{candidates}$ (see Algorithm 3). DAE-EDA then evaluates the objective value of the candidate solutions, and selects a subset of high-quality solutions. Note that in our implementation of DAE-EDA, the selection step resembles that of a steady-state Genetic Algorithm (see, e.g., Syswerda, 1991). Here, previously selected high-quality solutions are not simply discarded, but have a chance to be carried over into the next generation. That is, we select P_{t+1} not only from $P_{candidates}$ but from $P_{candidates} \cup P_t$.

Subsequently, DAE-EDA approximates the probability distribution D_{t+1} by training a DAE, using P_{t+1} as training data (see Algorithm 2). In principle, it would be possible to create D_{t+1} by *updating* D_t using P_{t+1}, as stated by Algorithm 1. However, in this case, the training data distribution constantly changes over the number of generations. Therefore, the problem becomes an online learning task, which is much harder than training on *iid* data (see, e.g., Murphy, 2012). Therefore, we train a new DAE on P_{t+1} in each generation.[7] After the DAE has been trained, DAE-EDA iterates to the next generation.

In order to train and sample the DAE, a number of parameters must be set in advance. In a set of preliminary experiments, we coarsely tested various parameter settings, without extensive fine tuning to specific problem types. Throughout our experiments, we used the following parameters for DAE-EDA[8]: since we want to make as few assumptions about

7. In earlier experiments we studied training RBM-EDA (which uses a Restricted Boltzmann Machine instead of an DAE, see Probst et al. 2017) in this fashion. However, the resulting "online-learning" EDA had large problems finding optimal solutions for more difficult problems, as the model was not able to adapt to the necessary changes during an EDA run.
8. The chosen values for the parameters specific to the DAE can be considered standard. The parameters are identical for all benchmark problems. Additional experiments showed that the performance of DAE-EDA is robust to minor changes of single parameters (parameters should remain in the same order of magnitude). The complete source code including configuration files containing all parameter settings for the experiments is available at `https://github.com/wohnjayne/eda-suite`.

the problem structure as possible, we set the number m of hidden neurons equal to the problem size n. The corruption process $q(\mathbf{x})$ randomly selects $c = 10\%$ of the inputs. Subsequently, each of the selected bits is randomly set to 0 or 1 (salt+pepper noise). We draw the initial weights $\mathbf{W}$ from a normal distribution with zero mean and variance 0.005, as $\mathbf{W} \sim N(0, 0.005)$. During training, the learning rates are set to $\alpha = 0.1$ and $\alpha' = 0.5$ for weights and biases, respectively. We use a batch size of $b = 100$, and a cross-entropy loss. To improve generalization, we use weight decay (L2 regularization)with a weight cost of $\lambda = 0.001$. Furthermore, we use a *momentum* of $\beta = 0.3$ (Qian, 1999).

When sampling new candidate solutions from the DAE, a proper choice of the number of sampling steps s is important. Preliminary experiments revealed that s should be relatively small. We chose $s = 10$, which allows the DAE to introduce correlations between the hidden neurons in the samples. Nevertheless, s is still low enough to be influenced by the random noise used for initializing the sampling process (see also Sections 3.4.3 and 4). Note that each sample is a vector $\mathbf{x} \in \{0, 1\}^n$. To turn this vector of real-valued elements into a candidate solution for the EDA, that is, a binary string, we sample each variable x_i from a Bernoulli distribution with $p = x_i$.

When training the DAE, we have to decide how many epochs the learning process shall be allowed to run. We apply a rather simple parameter control scheme to terminate DAE training (Probst et al., 2017; Probst, 2015a). The scheme uses the reconstruction error $e_t = Err(\mathbf{x}, \mathbf{z})$, which usually decreases with the number of epochs $t \in 1, \ldots, T$. Every second epoch, we calculate for a fixed subset u of the training set U the relative difference $e_t = 1/|u| \sum_{j \in u} Err(\mathbf{x}^j, \mathbf{z}^j)$. We run the learning process for a minimum of 50 training epochs. After the initial 50 epochs, we continue training, unless e_t rises. We measure the decrease γ of the reconstruction error in the last 33% of all epochs as

$$\gamma = (e_{0.67t} - e_t)/(e_0 - e_t).$$

We use γ to automatically check for convergence of the training. We stop training as soon as $\gamma < 0.05$, as we assume that the DAE has, by then, learned the most relevant dependencies between the variables. Further training is unlikely to improve the model considerably. When sampling from the DAE (after training is terminated), we use the weights $\mathbf{W}$ of the epoch with the lowest e_t.

2.3. Previous use of DAEs for Combinatorial Optimization

Recently, a technical report suggested using a DAE in an EDA-like optimization process (Churchill et al., 2014). In that work, the DAE is not used as a multivariate EDA model to sample new solutions, but is trained on the best 10-20% of the population. The trained DAE model is then used to improve a second set of selected individuals from a population as a variant of local search. Those individuals are first corrupted by the DAE's corruption process, and then reconstructed by the DAE, using the encoding and decoding functions. The underlying idea is that the DAE is capable of improving the second set of individuals by using the hidden representation of the problem structure learned from the best 10-20% of the population. Therefore, it resembles a local search where the variation operator is defined by the DAE's learned representation. Churchill et al. report that the approach works better than a simple genetic algorithm on a number of benchmark problems.

We recently published the basic idea of DAE-EDA in a technical report and presented preliminary results on the performance of DAE-EDA for combinatorial optimization (Probst, 2015a,b). The early results showed that DAE-EDA is capable of solving a set of benchmark problems very quickly, although it showed lower performance in comparison to EDAs considering multivariate dependencies like BOA. However, its model quality was already better than that of a simple univariate model. The refined DAE-EDA presented in this paper differs in a number of aspects from the variant already published on arXiv (Probst, 2015a). In contrast to the previous version, we now initialize the weights to smaller values and also use lower values for the learning rate α and weight cost λ. Furthermore, we use momentum, train the DAE for at least 50 epochs (instead of 20), and do not use early stopping using a held-out validation set. In sum, the DAE with these modified settings makes more steps in the direction of the gradient, but each individual step is smaller. This usually yields a better approximation to the training data, while accepting more overfitting and a longer run time.

3. Experiments and Results

We introduce the test problems (Section 3.1) and the experimental setup (Section 3.2), and present results for DAE-EDA and BOA on solving the test problems to optimality (Section 3.3). In Section 3.4, we study overfitting in DAE-EDA. We find that overfitting is no major problem for DAE-EDA.

3.1. Test Problems

We evaluate DAE-EDA on four standard benchmark problems using binary decision variables. Their difficulty depends on the problem size, that is, problems with more decision variables are more difficult. Additionally, for some of the problems, the difficulty is tunable by a parameter. Apart from the simplest problem, all problems are composed of concatenated subproblems. Those subproblems make the optimization objectives multimodal, since they are either deceptive, i.e., any local search using a small neighborhood would be first guided away from the global optimum, overlapping, i.e., variables of different subproblems are correlated, or even hierarchical, i.e., the optimization objective is composed of multiple hierarchical steps taking into account more and more problem variables. All problems are maximization problems.

The simple **onemax** problem assigns a binary solution $\mathbf{x}$ of length l an objective value $f(\mathbf{x}) = \sum_{i=1}^{l} x_i$, that is, it is equal to the number of ones in $\mathbf{x}$. The onemax function is unimodal and can be solved by a deterministic hill climber.

Concatenated deceptive traps (Deb and Goldberg, 1993) are tunably hard problems composed of separable subproblems. Here, a solution vector $\mathbf{x}$ is divided into l subsets of size k, with each one being a *deceptive* trap, which guides a local optimizer towards a local optimum: Within a trap, all bits are dependent on each other but independent of all other bits in $\mathbf{x}$. Thus, the contribution F_l to the optimization objective of each of the l traps can be evaluated separately and the total objective of the solution vector is the sum of these expressions. In particular, the assignment $\mathbf{a} = \mathbf{x}_{i:i+k-1}$ (that is, the k bits from x_i to

$x_{i+k-1})$[9] leads to a contribution of

$$F_l(\mathbf{a}) = \begin{cases} k & \text{if } \sum_i a_i = k, \\ k - (\sum_i a_i + 1) & \text{otherwise.} \end{cases}$$

In other words, the objective value of a single trap increases with the number of zeros, except for the vector of all ones, which is the global optimum for this particular trap. Consequently, the objective value of a solution vector $\mathbf{x}$ is calculated as $f(\mathbf{x}) = \sum_l F_l(\mathbf{a})$.

NK landscapes (Kauffman and Weinberger, 1989) are defined by the two parameters n and $k \in \{0, n-1\}$ as well as n components $f_i(x), i \in \{1 \ldots, n\}$. A solution vector $\mathbf{x}$ consists of n bits and each bit belongs to $k+1$ subsets. Consequently, each subset (component) has $k+1$ bits. Thus, each component f_i depends on the value of the corresponding variable x_i as well as k other variables and maps each possible configuration of its $k+1$ variables to a scalar contribution to the objective value. The total objective value f of a solution is the sum of the n components and calculated as

$$f(\mathbf{x}) = 1/n \sum_{i=1}^{n} f_i(x).$$

Each decision variable usually influences several f_i. These dependencies between subsets make NK landscapes also non-separable in the subproblems, that is, in general, we cannot solve the subproblems independently. The problem difficulty increases with k. $k = 0$ is a special case where all decision variables are independent and the problem reduces to a unimodal onemax. For our experiments, we use instances of NK landscapes with known optima provided by Pelikan (2008).

The Hierarchical If-and-only-if (HIFF) function (Watson et al., 1998) is defined for solutions vectors of length $n = 2^l$ where $l \in \mathbb{N}$ is the number of layers of the hierarchy. It uses a mapping function M and a contribution function C, both of which take two inputs. The mapping function takes each of the $n/2$ blocks of two neighboring variables of level $l = 1$, and maps them onto a single symbol each. In particular, an assignment of 00 is mapped to 0, 11 is mapped to 1 and everything else is mapped to the null symbol '-'. The concatenation of M's outputs on level l is used as M's input for the next level $l+1$ of the hierarchy, that is, if level $l = 1$ has n variables, level $l = 2$ has $n/2$ variables. On each level, C calculates a contribution to the objective value for each block of two variables. In particular, the assignments 00 and 11 contribute a value of 2^l to the overall objective value, all other assignments contribute nothing. The overall objective value is the sum of all blocks' contributions on all levels. In other words, a block contributes to the value of the current level if both variables in the block have the same assignment. However, only if neighboring blocks agree on the assignment, they will contribute to the value of the *next* level, which is why HIFF is a difficult problem. HIFF has two global optima: the string of all ones, and the string of all zeros.

3.2. Experimental Setup

First, we wanted to empirically determine the minimal population size that is necessary to optimally solve a problem according to a predefined percentage of all runs (for example,

9. The k variables assigned to trap l do not have to be adjacent, but can be at any position in x.

90%).[10] Consequently, for each instance and algorithm we tested multiple population sizes between 50 and 16,000[11]. For each population size, we ran 20 instances. We terminated each EDA run after either the (known) optimal solution was found, or after a maximum of 150 generations, or if there was no improvement in the best solution for more than 50 generations. Both types of EDAs (DAE-EDA and BOA) used tournament selection without replacement of size two, which is the tournament size that leads to the smallest possible selection pressure, i.e., which is most diversity-preserving (Miller and Goldberg, 1995).

All test problems (with the exception of NK landscapes) have the string of all ones as their global optimum, for any problem size. To avoid any possible model-induced bias towards solutions with ones or zeros, we generated a random matrix $\mathbf{R} \in \{0,1\}^{n*m}$ of ones and zeros for each run. In each generation, we applied the following operations: before training, we modified the training data to be $P_{t+1} \oplus \mathbf{R}$, with $\oplus$ being a logical XOR. After sampling from the DAE model, we obtained the resulting candidate solutions as $P_{\text{candidates}} = P_{\text{modelSamples}} \oplus \mathbf{R}$. Both operations are transparent to correlations between variables.

All algorithms were implemented in Matlab/Octave and executed using Octave V3.2.4 on a single core of an AMD Opteron 6272 processor with 2,100 MHz.

3.3. Performance Results

Table 1 lists the performance of DAE-EDA and BOA for onemax with $n \in \{50, 75, 100, 150\}$ bits, concatenated deceptive traps with $k = 4, n \in \{40, 60, 80, 120, 160\}$ bits and $k = 5, n \in \{50, 75, 100, 150\}$ bits, NK landscapes with $k \in \{4, 5\}$ and $n \in \{30, 34\}$ bits (two instances i each) as well as the HIFF function with $n \in \{64, 128\}$. For each instance and algorithm, we report the minimal population size N that allows the EDA to find the optimal solution in at least 18 of 20 runs, the corresponding average number of unique evaluations, i.e., the average number of times the objective function was called in these 20 runs, as well as the corresponding average CPU time. Note that we only count *unique* evaluations—if a particular configuration $\mathbf{x}$ occurs multiple times, we only count the first evalution of $f(\mathbf{x})$, store the objective value, and load it on subsequent calls[12]. Bold numbers indicate a significant difference between BOA and DAE-EDA according to a Wilcoxon signed-rank test ($p < 0.01$, data is not normally distributed), highlighting the better approach.

First, we study the number of required evaluations. Overall, neither of the two approaches (BOA versus DAE-EDA) clearly dominates. For the easy onemax problems, BOA needs slightly fewer evaluations in comparison to DAE-EDA. This is also true for the smaller $k = 4$ trap problems up to $n = 80$. For larger 4-trap problems, and all 5-trap problems, DAE-EDA needs fewer evaluations. For the NK-landscapes, neither of the two approaches dominates; each approach beats the other one in about the same number of test instances.

10. For a fixed problem size n, the probability α that population-based metaheuristics like EDAs fail to find the optimal solution is $O(e^{-N})$, where N is the used population size (Harik et al., 1999). While larger populations increase the probability of finding the optimal solution, calculating the objective values of all solutions may become prohibitively large.
11. $N \in \{50; 125; 250; 500; 1{,}000; 1{,}500;$ and 2,000 to 16,000 (increment 1,000)$\}$
12. The computational cost for querying and maintaining such a memory is small, compared to the cost of re-evaluating the fitness for our benchmark problems, and very likely to be negligible for real-world problems.

Problem	**Algorithm**	**Average results** Population size N such that optimum is found in $\geq$90% of runs		
		N	Unique Evaluations	Time (sec)
ONEMAX50	BOA	125	**2,119**±125	685±101
	DAE-EDA	125	2,324±121	**204**±30
ONEMAX75	BOA	125	2,787±158	2,182±321
	DAE-EDA	125	2,944±137	**336**±36
ONEMAX100	BOA	250	**6,259**±153	8,967±1,016
	DAE-EDA	250	7,005±214	**693**±58
ONEMAX150	BOA	250	**7,698**±270	26,867±3,380
	DAE-EDA	250	8,132±253	**1,361**±140
4-Traps 40 bit	BOA	1,000	**13,673**±758	2,728±297
	DAE-EDA	1,000	20,309±2,690	**237**±38
4-Traps 60 bit	BOA	1,000	**20,236**±1,362	10,604±1,707
	DAE-EDA	1,000	33,794±1,205	**492**±35
4-Traps 80 bit	BOA	2,000	**43,777**±1,695	43,935±4,994
	DAE-EDA	1,000	46,918±1,816	**910**±34
4-Traps 120 bit	BOA	3,000	82,068±2,120	205,244±16,962
	DAE-EDA	1,000	**74,998**±2,670	**2,633**±100
4-Traps 160 bit	BOA	-	execution time	limit exceeded
	DAE-EDA	1,000	**105,528**±3,749	**6,287**±323
5-Traps 25 bit	BOA	1,500	14,924±1,028	1,384±211
	DAE-EDA	1,000	**10,833**±992	**101**±17
5-Traps 50 bit	BOA	3,000	47,904±3,120	20,199±2,704
	DAE-EDA	1,000	**28,017**±3,414	**380**±52
5-Traps 75 bit	BOA	6,000	119,044±4,353	119,275±15,826
	DAE-EDA	1,000	**45,438**±2,313	**925**±84
5-Traps 100 bit	BOA	8,000	190,011±4,664	355,140±25,659
	DAE-EDA	1,000	**62,599**±2,852	**1,897**±161
5-Traps 150 bit	BOA	-	execution time	limit exceeded
	DAE-EDA	1,500	**125,238**±5,440	**7,671**±535
NK $n = 30$, $k = 4$, $i = 1$	BOA	2,000	32,015±3,094	4,590±1,044
	DAE-EDA	1,000	**17,472**±2,130	**202**±37
NK $n = 30$, $k = 4$, $i = 2$	BOA	4,000	**67,939**±6,649	13,360±3,445
	DAE-EDA	10,000	189,613±15,223	**1,091**±319
NK $n = 34$, $k = 4$, $i = 1$	BOA	1,000	**21,546**±1,860	3,603±625
	DAE-EDA	2,000	44,860±7,456	**368**±119
NK $n = 34$, $k = 4$, $i = 2$	BOA	5,000	88,321±9,272	20,377±5,123
	DAE-EDA	1,500	**34,879**±4,686	**344**±99
NK $n = 30$, $k = 5$, $i = 1$	BOA	500	**11,221**±644	1,565±260
	DAE-EDA	3,000	66,820±10,883	**504**±152
NK $n = 30$, $k = 5$, $i = 2$	BOA	2,000	**41,641**±5,157	6,831±1,541
	DAE-EDA	4,000	83,832±13,303	**463**±157
NK $n = 34$, $k = 5$, $i = 1$	BOA	16,000	307,171±24,227	86,846±15,431
	DAE-EDA	5,000	**117,065**±10,868	**631**±97
NK $n = 34$, $k = 5$, $i = 2$	BOA	16,000	300,058±42,914	84,266±22,365
	DAE-EDA	8,000	**195,202**±24,422	**1,011**±226
HIFF64	BOA	500	**11,991**±731	7,480±1,059
	DAE-EDA	1,000	22,066±2,234	**369**±32
HIFF128	BOA	1,500	51,008±2,387	137,617±12,151
	DAE-EDA	1,000	50,447±1,214	**2,542**±161

Table 1: Number of evaluations and CPU time for DAE-EDA and BOA. Bold numbers indicate better performance.

HIFF problem size	Average number of evaluations to solve all runs	
	DAE guided local search (10 runs)	DAE-EDA (20 runs)
128bit	231,000±23,537	86,700±6,867
256bit	1,355,500±190,793	220,650±13,348

Table 2: Number of evaluations to solve two HIFF problem instances to optimality. Results for the DAE guided local search are from Churchill et al. (2014).

For the 64 bit HIFF problem, BOA needs fewer evaluations. For $n = 128$ bit, there is no significant difference between both approaches.

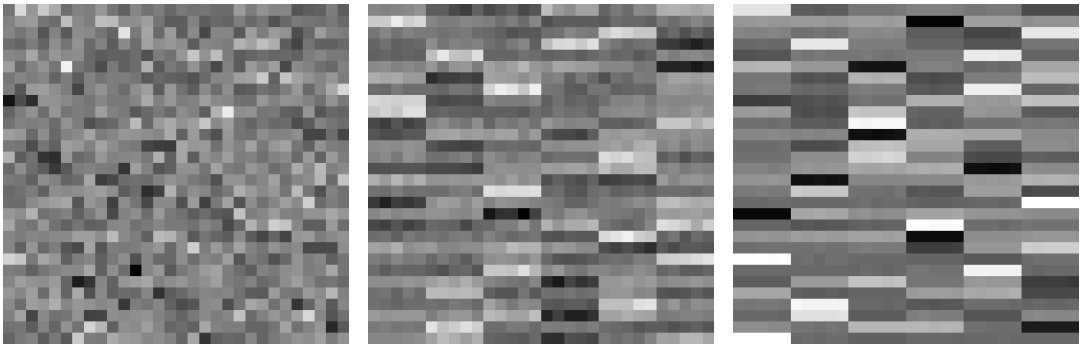
When comparing the performance of DAE-EDA to the results of its previous variant reported in a technical report (Probst, 2015a, not included in the table), the number of evaluations is significantly smaller, often by a large margin (except one single instance). Thus, we do not further examine performance differences between DAE-EDA and its previous variant.

Second, we examine the average time the two algorithms require to solve the problems. Here, the situation is different: For all problem instances, DAE-EDA is significantly faster than BOA, sometimes by multiple orders of magnitude. We want to emphasize that the direct comparison of CPU times is not entirely fair for BOA, due to the choice of programming language (see discussion in Section 4).

Table 2 compares DAE-EDA to the DAE-inspired local search proposed by Churchill et al. (2014). We report the number of evaluations required to optimally solve two instances of the HIFF problem (128bit and 256bit) in all runs. The results for the DAE-guided local search are taken from Churchill et al. (2014) and are the average number of evaluations necessary to solve 10 out of 10 optimization runs. For DAE-EDA, we report the average number of evaluations necessary to solve 20 out of 20 runs[13]. We observe that DAE-EDA needs fewer evaluations for both instances of the HIFF problem. For the instance with 128 bit, DAE-EDA needs only approx. 38% of the number of evaluations; for the 256 bit problem, it only needs 17%. We attribute this large performance difference to the sampling process, which samples from the trained model's distribution directly instead of using the DAE as a tool for local search modifying only selected individuals.

The results indicate that DAE-EDA is able to properly decompose the problems by approximating the probability distribution of the training data. Figure 2 exemplarily visualizes the DAE's problem decomposition when solving a concatenated 5-trap problem. For three different generations (7, 13, and 17), the plots visualize the DAE's weight matrix; white pixels correspond to large positive weights, black pixels correspond to large negative weights. Each of the 30 rows (each row corresponds to one of the 30 hidden neurons) contains 30 weights connecting the hidden neuron with the 30 problem variables. In the deceptive 5-trap problem, blocks of five adjacent variables have a strong contribution to the objective value if *all five* variables are equal to one or equal to zero. Furthermore, each block

13. Note that solving 20 out of 20 runs is more difficult than solving 10 out of 10 runs. Also note that the number of evaluations reported for DAE-EDA for the 128bit problem in Table 2 differs from the one in Table 1, which reports *unique* evaluations only.



(a) EDA generation 7 (b) EDA generation 13 (c) EDA generation 17

Figure 2: Example of weight matrices $\mathbf{W} \in \mathbb{R}^{[m \times n]}$ of DAE-EDA optimizing a 30 bit concatenated 5-trap problem.

of five variables is independent of all other blocks. In early generations (Figure 2a), problem structure is strongly masked by noise. Rows with a single bright or dark pixel correspond to hidden neurons which have partly learned the identity function, due to overfitting. In later generations (Figures 2b and 2c), many of the hidden neurons strongly influence a single block of problem variables (bright/dark blocks of five adjacent pixels), and are indifferent to most other variables (mid gray values). Therefore, the learned representation of the model matches the problem structure well.

3.4. Overfitting in DAE-EDA

We study overfitting of DAEs and how it affects EDA behavior and performance. We find that, in a single EDA generation, the DAE overfits by learning the identity function (IF) (Section 3.4.1). This situation is especially relevant in the early EDA generations, where strong noise is masking the problem structure (Section 3.4.2). We show that samples from a DAE-EDA which has learned the IF are very diverse, sometimes even more diverse than the training data (Section 3.4.3). This diversity can help DAE-EDA to avoid premature convergence, search the solution space more thoroughly by escaping local optimal, and yield better overall solutions (Section 3.4.4).

Throughout this section, we report results for a population size of $N = 200$, which are averaged over 50 runs. For the DAE corruption noise, we use three different values $c \in \{0.05, 0.1, 0.2\}$. The corruption noise acts as a regularizer and, therefore, has a direct influence on overfitting.

3.4.1. An Overfitting DAE Learns the Identity Function

We study the overfitting behavior of the DAE when approximating the population P_{t+1} in the very first EDA generation. Recall that an AE tends to learn the IF as its hidden representation. To measure how strongly the AE replicates given input in its output, we

define the degree $\tau \in [0, 1]$ to which the DAE has learned the IF as

$$\tau = \frac{1}{u * n} \sum_{u,n} \mathbb{I}(A_{un}, f(c(A_{un}))),$$

where $\mathbf{A} \in [0, 1]^{un}$ is a matrix containing u random individuals $\in \{0, 1\}^n$, A_{un} is the nth bit in the uth row of $\mathbf{A}$, and

$$\mathbb{I}(a, b) = \begin{cases} 1 & \text{if } a = b, \\ 0 & otherwise. \end{cases}$$

τ measures the percentage of output bits that are identical to the corresponding input bits after performing a single encoding/decoding step with $\mathbf{z} = f(c(\mathbf{x}))$ (see Algorithm 2, lines 5-6). For an untrained DAE, $\tau \approx 0.5$. If the learned hidden representation is equal to the IF, the output is identical to the input, and, hence, $\tau = 1$.

To study how τ changes during DAE training, we focus on the first EDA generation. If the initial distribution is uniform over the search space, the diversity of the population will be the highest possible. Each subsequent selection step consecutively reduces diversity throughout an EDA run.

Figure 3 plots the degree τ to which the IF has been learned by the partially trained DAE over the number of training epochs for a 50 bit 5-Trap problem.[14] We show results for different amounts of corruption $c \in \{0.05, 0.1, 0.2\}$.

14. Results for other problem instances are qualitatively similar. Hence, for brevity reasons, Sections 3.4.1 to 3.4.3 only display results for 50 bit 5-Traps.

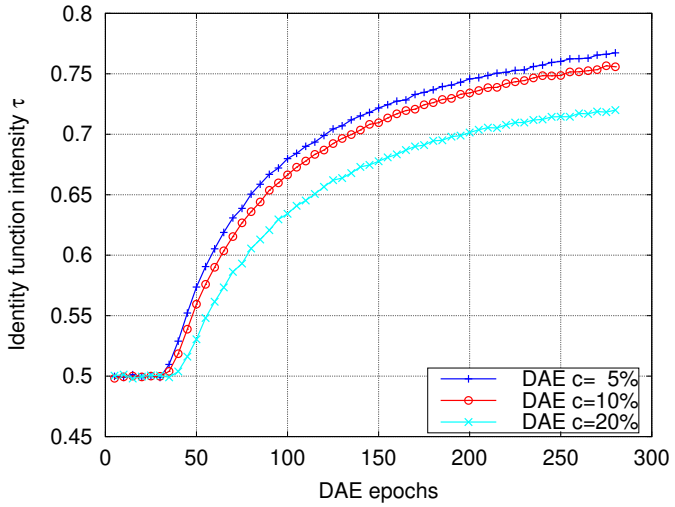


Figure 3: Degree τ to which a DAE learns the identity function over the training epochs of the first generation. We plot results for different levels of regularization noise c.

We find that in the first epochs, $\tau \approx 0.5$, for all c. This means, the DAE has not (yet) learned the identify function. After approximately 30 to 40 training epochs, τ starts increasing. With a lower amount of corruption noise c, that is, with less regularization, τ starts to increase earlier and grows stronger. Hence, after an initial period of a few epochs where the DAE is not overfitting, a DAE that is trained for more epochs gradually overfits by learning the univariate IF. Lower values for c lead to an earlier and stronger overfitting behavior.

Learning the IF has a strong impact on the samples obtained from the DAE, due to the way the sampling process works: New samples are initialized by random values, followed by multiple encoding/decoding and corruption steps (see Section 2.2.3). These initial random values are uniformly distributed in the EDA solution space. Since the overfitting DAEs learns the identity function (which just replicates the random input values as output values), the DAEs samples become more similar to the random initialization noise, which avoids replicating sampling noise.

3.4.2. DAE Overfitting over the Number of EDA Generations

We extend our analysis and study how τ develops over the number of generations of an EDA run. Figure 4 plots the degree τ to which the trained DAE has learned the identity function (after training is terminated) over the generations of the EDA. As before, we show results for 50 bit 5-traps and average results over 50 runs. Results for other problem instances are analogous. We find that τ decreases during an EDA run, and that, again, lower values of corruption noise c lead to larger values of τ. This means that with lower corruption noise, the DAE tends to overfit more strongly by learning the identity function.

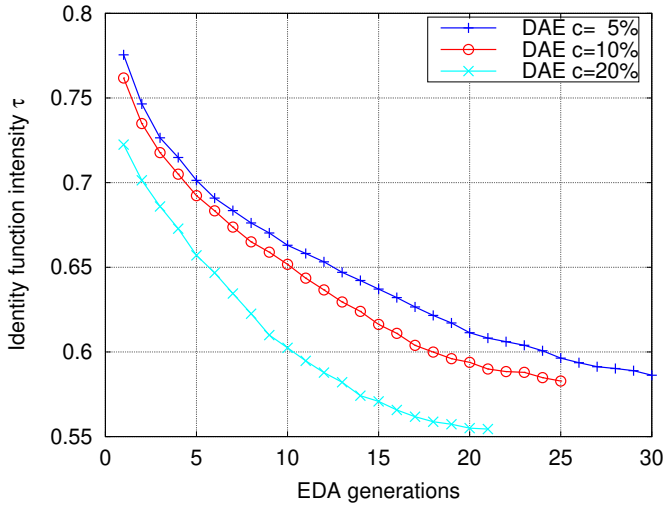


Figure 4: Degree τ to which DAE-EDA has learned the identity function over the number of generations. We plot results for different levels of regularization noise c.

Recall that, in each generation, DAE training stops when the improvement on the reconstruction error has slowed down (see Section 2.2.4). In the early EDA generations, the problem structure is usually strongly masked by random noise. During training, the DAE can only learn problem structure which is not masked. After doing so, it will capture the IF to further decrease the reconstruction error, especially in the early EDA generations. This is different in later generations, when the amount of noise present in the population is lower. In this case, the DAE can reduce the reconstruction error by capturing problem structure, before starting to overfit by learning the IF. As a result, the high uncertainty about the problem structure in the first EDA generations leads to a DAE with a hidden representation strongly determined by the IF. Such a model will tend to reproduce inputs as outputs.

3.4.3. Population Diversity in DAE-EDA and BOA

We study how the diversity of the individuals in a population develops over an EDA run and how DAE corruption noise c affects population diversity.

The diversity of a population measures how diverse the individuals in a population are. Diversity is high if there are large differences between the solutions in a population, which is the case when sampling uniformly from the solution space. In contrast, a completely converged population has minimal diversity since all individuals are identical. We measure the diversity $\delta \in [0, 1]$ of a population as the (normalized) *inertia* I (Morrison and De Jong, 2002) as

$$\delta = \frac{4}{nN} * I = \frac{4}{nN} \sum_{i=1}^{n} \sum_{u=1}^{N} (x_i^u - \bar{x}_i)^2,$$

where n is the length of a solution (number of bits), N is the population size, x_i^u is the value of individual $u \in \{1, \ldots, N\}$ at position $i \in \{1, \ldots, n\}$, and $\bar{x}_i = 1/N * \sum_{u=1}^{N} x_i^u$ is the mean of the ith variable across all individuals. Thus, the inertia I is equal to the sum of the variances of the individual variables. The factor $4/(nN)$ normalizes δ to be $\in [0, 1]$. Besides inertia, there are other ways to measure the diversity of a population such as the population diversity index (Smit et al., 2011), which counts the number of duplicates in a population. Unfortunately, such approaches are not meaningful in the first EDA generations since the number of duplicates is usually close to zero. In contrast, using the inertia I has the drawback that it is univariate, that is, it does not consider interactions between variables. Hence, it tends to overestimate population diversity when there are multiple copies of pairwise diverse individuals in the population (see discussion in Smit et al., 2011), which can be the case in later EDA generations (especially for trap functions).

The search behavior and performance of EDAs strongly depend on the diversity of its population. Often, EDAs start with an initial population of solutions that are uniformly sampled from the solution space. In such an initial population, diversity is maximal and subsequent EDA generations usually continuously reduce diversity (Rothlauf, 2011). The main mechanism that reduces diversity in an EDA is selection. In each selection step, low-quality solutions are discarded from the population. Since the selected solutions form a subset of the original population, population diversity is reduced with every selection

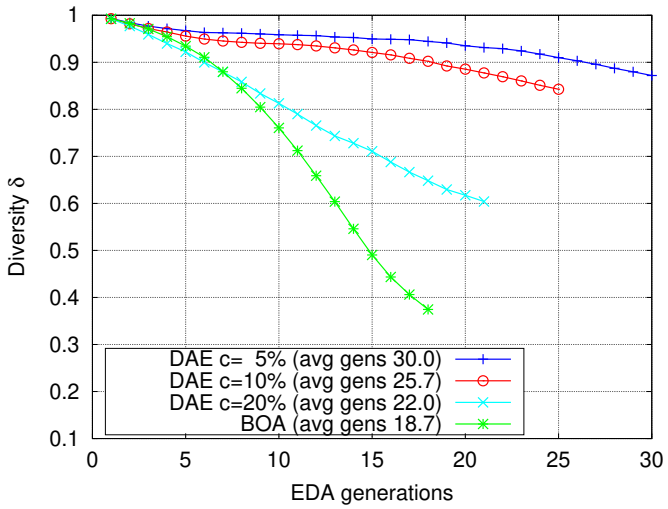


Figure 5: Diversity δ of the population over number of generations. We plot results for DAE-EDA with different levels of regularization noise c and BOA.

step.[15] Often, an EDA run stops when the population is converged and diversification is low.

Usually, building an appropriate model for a population and sampling from the model does not affect a population's diversity. For example, when using large population sizes N and simple univariate models like UMDA or PBIL, which assume factorized probability distributions, univariate diversity measures like δ find that the diversity of the sampled population is identical to its parent population which was used to train the model. However, the situation is different if the model does not learn the properties of the parent population, that is, it is underfitting. In this case, model building and sampling can also reduce population diversity since candidate solutions sampled from an underfitting model of the parent population have lower diversity (Branke et al., 2007).

We study how the diversity δ of an EDA population changes over the number of generations. Figure 5 plots the diversity δ of the individuals selected by the EDA for model building over the number of EDA generations. We show results for the three DAE configurations with different corruption noise ($c \in \{0.05, 0.1, 0.2\}$) and for BOA. Again, results are for 50 bit 5-traps and averaged over 50 runs.[16]

As expected, the initial diversity is close to $\delta \approx 1$, which indicates a randomly sampled population uniformly distributed over the search space. Furthermore, diversity decreases in subsequent generations. For DAE-EDA, low values of c (which means low regularization)

15. Note that there are specific mechanisms that can be used in GAs and EDAs to limit the loss of diversity in the selection step—see discussion in Section 4.

16. We only plot results up to the average generation in which each of the individual runs found its best solution (which may also be a local optimum). As outlined in Section 3.2, EDAs that do not find the optimum solution may continue for up to 50 more generations before converging or terminating. These "tails" of the runs are omitted in the plots.

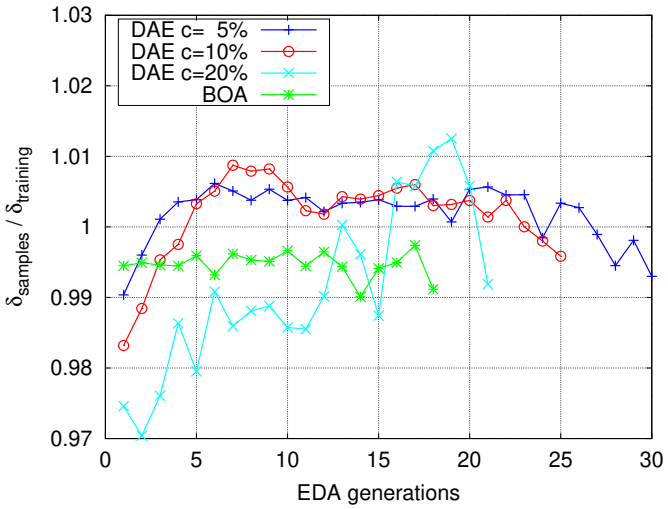


Figure 6: $\delta_{\text{samples}}/\delta_{\text{training}}$ over number of generations. We plot results for DAE-EDA with different levels of regularization noise c and BOA.

result in a slower decrease of diversity. Accordingly, DAE-EDA with weak regularization runs for more EDA generations than DAE-EDA with strong regularization. For example, with $c = 0.05$, DAE-EDA finds the best solutions after an average number of 30.0 generations; with $c = 0.2$, DAE-EDA finds the best solutions already after an average of 22.0 generations. BOA runs find their best solutions after only 18.7 generations. We see that for this specific problem instance, the population diversity in DAE-EDA is significantly higher than in BOA, where we observe the strongest loss of diversity and the lowest average number of generations.

The results obtained are surprising since all four EDA variants use the same selection mechanism. Hence, the differences in the population diversity cannot be a result of selection but must be a result of the model building and sampling step. Therefore, we analyze the diversity of the individuals sampled from a model and compare it to the individuals selected for training a model. Figure 6 shows the ratio $r_\delta = \frac{\delta_{\text{samples}}}{\delta_{\text{training}}}$ for DAE-EDA with different regularization noise c and BOA over the number of generations. Again, we only plot results up to the average number of generations that are necessary to find the best solutions. $r_\delta = 1$ indicates that the diversity of the training and sampled population is identical. Lower values of r_δ indicate a diversity loss due to model building and sampling. Higher values of r_δ indicate that model building and sampling introduce additional diversity into the search process.

For BOA, we observe $r_\delta \approx 0.995$. Thus, model building and sampling leads to a small, but constant loss of diversity over the number of generations. A reason for this loss in diversity could be that BOA uses BIC, which "tends to favor overly simply models" (Bishop, 2006, p. 33), thereby introducing a bias towards simpler, less diverse solutions.

The behavior of DAE-EDA varies, depending on the setting of c. For weak regularization ($c \in \{0.05, 0.1\}$), model building and sampling reduces diversity only in the first few generations. After ≈ 5 generations, $r_\delta \gtrsim 1$ which means that model building and sampling slightly increases the diversity. This constant increase of diversity enables the weakly regularized DAE-EDAs to keep the population diversity at a relatively high level for many generations (compare Figure 5). When increasing regularization noise to $c = 0.2$, the situation is similar; however, it takes more generations until the model building and sampling increases diversity. Since the loss of diversity is stronger in the early generations, the diversity of the population is reduced more quickly in comparison to lower values of c.

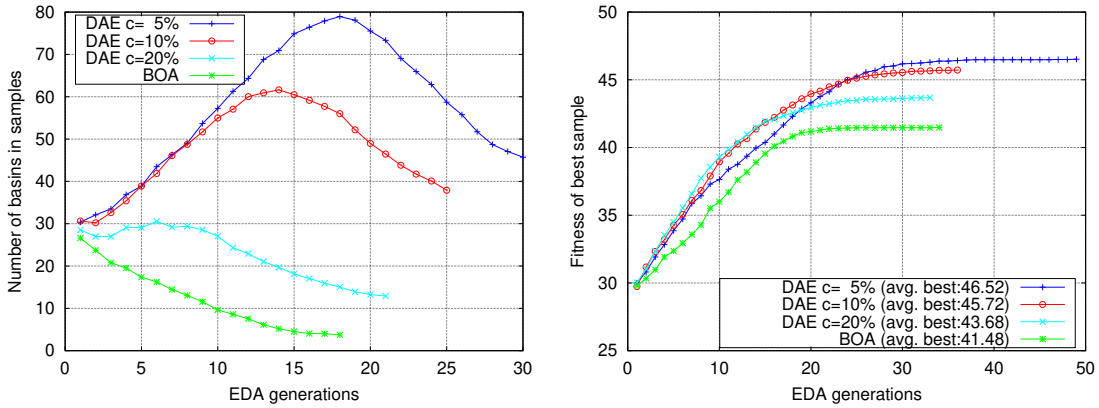
3.4.4. Diverse Populations Lead to Better Solutions

We found that DAE-EDA with weak regularization tends to increase the diversity of the samples, compared to the training data. This keeps the population diversity high, enabling the EDA to run for more generations. However, diverse solutions alone do not necessarily entail better optimization results: if the additional diversity is merely caused by noise overlaying the same prototype solution, it may cause the EDA to converge more slowly, but not by default to a better solution. We are more interested in solutions that are diverse in the sense that they belong to different, promising areas of the solution space.

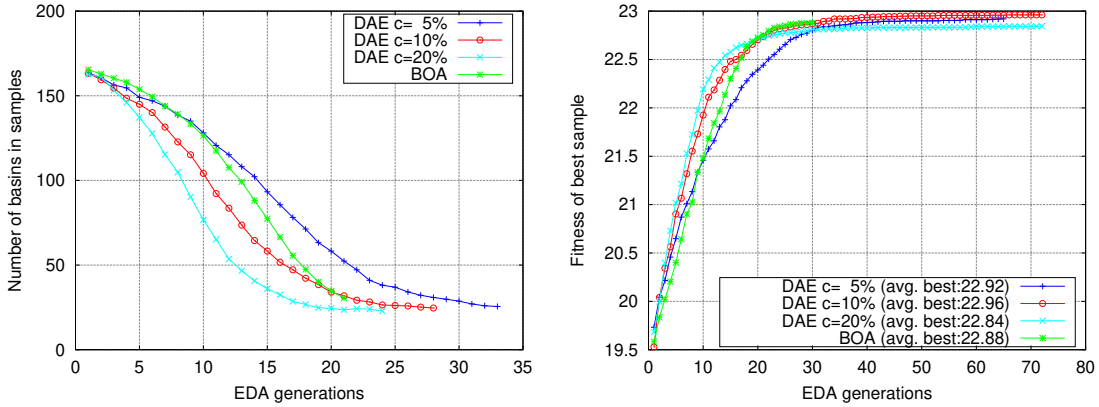
Hence, we analyze how many of the promising parts of the solution space are searched. All test problems are multimodal, that is, they have multiple local optima. One of these local optima is the global optimum. Therefore, during the search, multiple areas of the search space can be deemed promising. As a rule of thumb, a good heuristic should not focus on very few of these promising areas too quickly, since it may discard the part of the solution space containing the global optimum. Instead, it should keep the coverage of the promising areas high. A proxy for this coverage is the number of local optima, that the algorithm is likely to find. A higher number of local optima means a better coverage of the promising areas of the solution space. We can determine this number by counting the number of different basins of attraction of local optima in the population. A basin of attraction for a local optimum consists of all solutions which take a deterministic hill climber to this local optimum. Therefore, for a population of solutions, we determine the number of basins γ by counting how many different local optima a simple hill climber finds on the candidate solutions.

The left-hand side of Figure 7 shows the number of basins of attraction γ for DAE-EDA and BOA on the 50 bit 5-Traps (7a), NK with $n = 30$, $k = 4$, and $i = 1$ (7b), and 64 bit HIFF instances (7c). For all three problem instances, DAE-EDAs coverage of the solution space γ is higher for low values of c, that is, using a weakly regularized DAE. That is, throughout a run, DAE-EDA is able to keep a higher number of different basins of attraction in the population, and focus the search on specific basins only in later EDA generations.

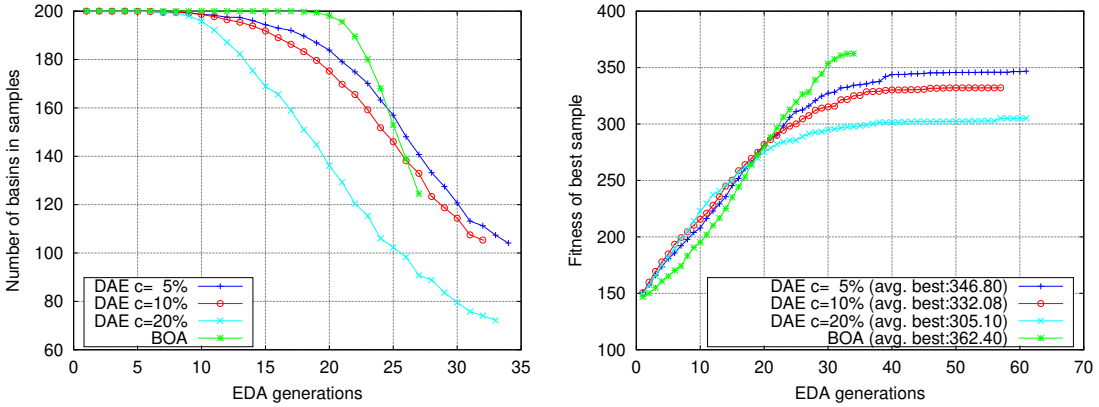
Consequently, the objective values of the model samples develops differently, depending on the regularization. The right-hand side of Figure 7 shows the average value of the best model sample in each EDA generation. For all three problems, DAE-EDA with strong regularization finds good solutions faster by quickly focusing the search on a smaller number of basins of attraction. On the other hand, DAE-EDA with weak regularization explores



(a) 50 bit 5-Traps



(b) NK landscape $n = 30, k = 4, i = 1$



(c) 64 bit HIFF

Figure 7: Number of basins of attraction γ covered in the model samples (left) and the fitness of the best model sample (right) over the number of generations for the 50 bit 5-Trap problem (top), the NK $n = 30, k = 4, i = 1$ instance (middle), and the 64 bit HIFF problem (bottom). The left-hand plots show γ up to the average number of generations until the best solution is found, whereas the right-hand plots show the fitness until the EDA run is stopped.

more basins, and needs a higher number of EDA generations to find high-quality solutions. With increasing number of generations the higher coverage of basins pays off for the weakly regularized DAE-EDAs, allowing them to find better solutions. For all test problems, the quality of the final solution is higher for the weakly regularized DAE-EDAs.[17]

These results, which are obtained with a lower population size $N = 200$, confirm the findings from Table 1: DAE-EDA performs particularly well on the deceptive trap problem. Here, DAE-EDA's high coverage of the solution space, which is higher than BOA, leads to a performance difference between DAE-EDA and BOA. For the NK instance, the coverage γ of the solution space is similar for DAE-EDA and BOA, which leads to similar performance (even when low-regularized DAE-EDAs slightly outperform BOA). For the 64 bit HIFF problem, we observe a higher performance of BOA, which goes along with a higher coverage γ of the solution space. For the larger 128 bit HIFF problem (not shown in the figure), the situation would slightly change since weakly-regularized DAE-EDAs better cover the solution space, allowing them to perform similar to BOA (compare Table 1).

In sum, we find that DAEs overfit by learning the IF, especially in the first EDA generations, where there is a lot of uncertainty about the problem structure. The strength of this effect can be influenced by the regularization parameter c. The overfitting of the DAE leads to more diverse samples, which enables DAE-EDAs to run longer before converging. Furthermore, higher diversity leads to a higher coverage of the promising parts of the solution space, and hence, to a better solution quality.

4. Discussion

Our results indicate that DAE-EDA is able to solve difficult combinatorial optimization problems. Regarding the number of evaluations of the objective functions, its performance is similar to BOA. Especially for larger, difficult concatenated trap problems, DAE-EDA outperforms BOA. This indicates that the representation learned by the DAE in each EDA generation matches the underlying problem structure well, allowing DAE-EDA to properly decompose difficult problems.

The way the DAE handles overfitting has a major impact on the performance of DAE-EDA. Usually, when fitting complex models to data, overfitting has to be carefully controlled by adjusting the amount of regularization. DAE overfits by learning the identity function (IF), gradually overlaying the learned, useful representation. Although this type of overfitting has a negative effect on the quality of the DAE's approximation of the training data, learning the univariate IF is less harmful for the EDA in comparison to learning multivariate sampling noise, since the IF introduces unbiased random noise into the samples. This unbiased noise leads to a higher level of population diversity, which allows DAE-EDA to run for more generations before convergence, results in a better coverage of promising solutions in the solution space, and enables DAE-EDA to search the promising parts of the solution space more thoroughly.

Since overfitting leads to unbiased random noise in the samples, DAE-EDA is robust with respect to the amount of regularization. Thus, an accurate adjustment of the amount of regularization is usually not necessary. This is particularly useful since the probability

17. For the NK landscape problem, the overall pattern is consistent but the average best solution of DAE-EDA with $c = 0.1$ is slightly better than with $c = 0.05$ (see Figure 7b).

distributions of populations in different EDA generations change strongly: in early EDA generations, populations are very noisy and population diversity is high. With an increasing number of generations, there is less noise and population diversity gets lower as the search focuses on promising areas of the search space. At the end of a run, population diversity is very low and the population has converged. Since DAE-EDA is robust with respect to the amount of regularization, no larger effort to control overfitting throughout an DAE-EDA run is necessary.

All algorithms presented in this paper are implemented in Matlab/Octave. We found that the execution times of DAE-EDA are much lower than BOA, sometimes by multiple orders of magnitude (see Table 1). However, the CPU times are biased since the implementation of DAE-EDA usually makes extensive use of matrix operations, which are relatively fast in Matlab/Octave. In contrast, BOA mainly uses nested loops for calculating the model, which are relatively slow in Matlab/Octave. When switching to other programming languages like C, we expect the running time differences between BOA and DAE-EDA to be smaller since such languages better support nested loops. On the other hand, of course, DAE-EDA can strongly benefit from parallelization using GPUs (see e.g. Probst et al. (2014)). In contrast, parallelizing multivariate EDAs like BOA is possible but difficult, and typically yields lower speedups, even when using GPUs (Očenášek and Schwarz, 2000; Munawar et al., 2009). In summary, moving from Octave/Matlab to more efficient implementations is likely to reduce the gap in running times between DAE-EDA and BOA. However, since model building and sampling in DAEs can strongly benefit from parallelization, we presume that the use of parallel architectures like GPUs would, again, result in DAE-EDA being the faster algorithm.

In this work, we compare a DAE-EDA with a single hidden layer to the basic version of BOA rather than its successor hBOA (compare Section 2.1.3). To tackle challenging, hierarchical problems more efficiently, hBOA extends BOA with two mechanisms: First, it uses Restricted Tournament Replacement (RTR), a modified selection operator designed to retain higher levels of diversity (Harik, 1995). RTR could be used directly in DAE-EDA as well. Second, hBOA uses decision trees to capture local structures better, and reduce the number of probabilities that need to be calculated and stored when a node has many parents, i.e., hierarchical dependencies are present. A similar effect could be achieved in a DAE by increasing the number of hidden layers, i.e., making it *deeper*. Deep latent layers could then model patterns spanning many decision variables by building upon patterns spanning fewer decision variables modeled by shallow layers.

Consequently, it would be interesting to investigate in greater detail how the performance of DAE-EDA varies when modifying the number and size of the hidden layer(s). A single, smaller hidden layer can not properly implement the identity function required for unbiased sampling noise, but would allow for faster training and automatic regularization. Using a deeper network with more than one hidden layer and skip connections (see, e.g., He et al. (2016)) accounting for the identity function could be a promising compromise.

Lastly, it would be worthwhile to investigate how the discovered diversity-preserving mechanism of DAE-EDA relates to IGO algorithms (Ollivier et al., 2017). For IGO algorithms (like PBIL or cGA), it has been shown that the loss of diversity during an optimization run is minimal. DAE-EDA also follows this idea and keeps the diversity loss resulting from model building and sampling low. We encourage future work that studies

the properties of the Fisher information matrix (Ollivier et al., 2017) when using DAE models.

5. Conclusions

We introduced DAE-EDA, an Estimation of Distribution Algorithm which uses a Denoising Autoencoder as its probabilistic model and tested its performance on a set of standard benchmark problems for combinatorial optimization with a single objective. DAE-EDA required a similar number of evaluations of the objective function to solve the problems to optimality, compared to the Bayesian Optimization Algorithm (BOA). For larger instances of difficult concatenated deceptive trap problems, DAE-EDA needed a lower number of evaluations than BOA. In our Octave implementation, the running times of DAE-EDA are much lower than BOA since model training and sampling is much faster for a DAE than for a Bayes network as used in BOA. Therefore, DAE-EDA can be a very useful optimization tool for problems where the computational cost of evaluating the objective function is low.

The major reason for the high performance of DAE-EDA is its specific way of overfitting to limited amounts of data. Overfitting usually leads to learning multivariate sampling noise, where the model learns spurious correlations between decision variables. In contrast to other models, DAEs overfit by gradually replacing a learned, useful latent problem representation by the univariate identity function. Although this has a detrimental effect on the overall model quality, its effect on heuristic search is low since candidate solutions sampled from a DAE tend to be diverse and resemble white noise when combined with the stochastic sampling process. This enables DAE-EDA to run for more generations, and search the solution space more thoroughly.

There are multiple paths for future research. First, it could be beneficial to implement *walkback* training, an extension of the training algorithm often used with DAEs (Bengio et al., 2013). This could yield an improved model quality, at the cost of a higher computational effort. Furthermore, there are other sampling techniques that could be applied to the DAE to create new candidate solutions (Bengio et al., 2013). Another area of research could be to use other generative variants of autoencoders, such as the Contractive Autoencoder (Rifai et al., 2011) or the Variational Autoencoder (Kingma and Welling, 2013). Lastly, it would be very interesting to investigate the effect of modifying the size of the DAE's hidden layer, or increasing the number of hidden layers.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable and insightful comments.

References

Asaad Abdollahzadeh, Alan Reynolds, Michael Christie, David W Corne, Brian J Davies, and Glyn JJ Williams. Bayesian optimization algorithm applied to uncertainty quantification. *SPE Journal*, 17(03):865–873, 2012.

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *Journal of Machine Learning Research*, 15:3563–3593, 2014.

Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search-based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.

Shumeet Baluja and Scott Davies. Using optimal dependency-trees for combinational optimization: Learning the structure of the search space. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997)*, pages 30–38. Morgan Kaufmann, 1997.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends Machine Learning*, 2(1):1–127, January 2009. ISSN 1935-8237. doi: 10.1561/2200000006. URL `http://dx.doi.org/10.1561/2200000006`.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J.C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising Auto-encoders as generative models. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*. NIPS Foundation (http://books.nips.cc), 2013.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN 9780387310732.

Juergen Branke, Clemens Lode, and Jonathan L. Shapiro. Addressing sampling errors and diversity loss in umda. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 508–515, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: 10.1145/1276958.1277068. URL `http://doi.acm.org/10.1145/1276958.1277068`.

Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(Jul): 2079–2107, 2010.

Shih-Hsin Chen, Min-Chih Chen, Pei-Chann Chang, Qingfu Zhang, and Yuh-Min Chen. Guidelines for developing effective estimation of distribution algorithms in solving single machine scheduling problems. *Expert Systems with Applications*, 37(9):6441–6451, 2010.

Alexander W. Churchill, Siddharth Sigtia, and Chrisantha Fernando. A denoising autoencoder that guides stochastic search. Technical Report arXiv:1404.1614, Queen Mary, University of London, 2014.

Kalyanmoy Deb and David E Goldberg. Analyzing deception in trap functions. In D. L. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 93–108. Morgan Kaufmann, 1993. doi: 10.1016/B978-0-08-094832-4.50012-X.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

David E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, Reading, MA, USA, 1989.

David. E. Goldberg, Kalyanmoy Deb, and James H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

George Harik, Erick Cantú-Paz, David E. Goldberg, and Brad L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7 (3):231–253, 1999.

Georges R. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 15-19, 1995*, pages 24–31, 1995.

Georges R Harik, Fernando G Lobo, and Kumara Sastry. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In Martin Pelikan, Kumara Sastry, and Erick Cantú-Paz, editors, *Scalable optimization via probabilistic modeling*, volume 33 of *Studies in Computational Intelligence*, pages 39–61. Springer, Berlin, Heidelberg, 2006.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Geoffrey E. Hinton. A practical guide to training restricted Boltzmann machines. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Ragan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, Moshe Y. Vardi, G. Weikum, G. Montavon, G. B. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35288-1. doi: 10.1007/978-3-642-35289-8{\textunderscore}32.

Geoffrey E Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

John H Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Complex Adaptive Systems. MIT Press, Cambridge, 6 edition, 2001. ISBN 9780262082136.

Stuart A Kauffman and Edward D Weinberger. The NK model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology*, 141(2):211–245, 1989. ISSN 00225193. doi: 10.1016/S0022-5193(89)80019-0.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2014 International Conference on Learning Representations (ICLR)*, 2013.

Pedro Larrañaga and Jose A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2 of *Genetic Algorithms and Evolutionary Computation*. Springer US, Boston, MA, USA, 2002. ISBN 978-1-4613-5604-2. doi: 10.1007/978-1-4615-1539-5.

Pedro Larrañaga, Hossein Karshenas, Concha Bielza, and Roberto Santana. A review on probabilistic graphical models in evolutionary computation. *Journal of Heuristics*, 18(5): 795–819, 2012. doi: 10.1007/s10732-012-9208-4.

Jose A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*, volume 192 of *Studies in Fuzziness and Soft Computing*. Springer, Berlin, Heidelberg, 2006. ISBN 9783540324942. doi: 10.1007/3-540-32494-1.

Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

Ronald W. Morrison and KennethA. De Jong. Measurement of population diversity. In Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, and Marc Schoenauer, editors, *Artificial Evolution*, volume 2310 of *Lecture Notes in Computer Science*, pages 31–41. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43544-0. doi: 10.1007/3-540-46033-0_3. URL `http://dx.doi.org/10.1007/3-540-46033-0_3`.

Heinz Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer, Berlin, Heidelberg, 1996. ISBN 978-3-540-61723-5.

Asim Munawar, Mohamed Wahib, Masaharu Munetomo, and Kiyoshi Akama. Theoretical and empirical analysis of a GPU-based parallel Bayesian optimization algorithm. In *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2009)*, pages 457–462. IEEE, 2009. doi: 10.1109/PDCAT.2009.32.

Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

Jiří Očenášek and Josef Schwarz. The parallel Bayesian optimization algorithm. In P. Sinčák, Ján Vaščák, V. K., and R. Mesiar, editors, *The State of the Art in Computational Intelligence*, volume 5 of *Advances in Soft Computing*, pages 61–67. Physica-Verlag, Heidelberg, 2000. ISBN 978-3-7908-1322-7. doi: 10.1007/978-3-7908-1844-4{\textunderscore}11.

Yann Ollivier, Ludovic Arnold, Anne Auger, and Nikolaus Hansen. Information-geometric optimization algorithms: A unifying picture via invariance principles. *The Journal of Machine Learning Research*, 18(1):564–628, 2017.

Martin Pelikan. Bayesian optimization algorithm. In *Hierarchical Bayesian Optimization Algorithm*, volume 170 of *Studies in Fuzziness and Soft Computing*, pages 31–48. Springer, 2005a.

Martin Pelikan. Hierarchical Bayesian optimization algorithm. In *Hierarchical Bayesian Optimization Algorithm*, volume 170 of *Studies in Fuzziness and Soft Computing*, pages 105–129. Springer, 2005b.

Martin Pelikan. Analysis of estimation of distribution algorithms and genetic algorithms on NK landscapes. In C. Ryan and M. Keijzer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, volume 10, pages 1033–1040, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-130-9. doi: 10.1145/1389095.1389287. URL `http://medal-lab.org/files/nk-instances.tar.gz`.

Martin Pelikan and David E Goldberg. Hierarchical BOA solves ising spin glasses and MAXSAT. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, volume 2724 of *Lecture Notes in Computer Science*, pages 1271–1282, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-40603-7. doi: 10.1007/3-540-45110-2{\textunderscore}3.

Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry, editors, *Advances in Soft Computing*, pages 521–535. Springer, London, 1999. ISBN 978-1-85233-062-0. doi: 10.1007/978-1-4471-0819-1{\textunderscore}39.

Martin Pelikan, David E Goldberg, and Erick Cantu-Paz. BOA: the Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, volume 1, pages 525–532, San Francisco, CA, USA, 1999. Morgan Kaufmann.

Malte Probst. Denoising autoencoders for fast combinatorial black box optimization. Technical Report arXiv:1503.01954, University of Mainz, 2015a.

Malte Probst. Denoising autoencoders for fast combinatorial black box optimization. In *Proceedings of the Companion Publication of the 2015 Genetic and Evolutionary Computation Conference (GECCO)*, GECCO Companion '15, pages 1459–1460, New York, NY, USA, 2015b. ACM.

Malte Probst, Franz Rothlauf, and Jörn Grahl. An implicitly parallel EDA based on restricted Boltzmann machines. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014)*, pages 1055–1062, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2662-9. doi: 10.1145/2576768.2598273.

Malte Probst, Franz Rothlauf, and Jörn Grahl. Scalability of using restricted Boltzmann machines for combinatorial optimization. *European Journal of Operational Research*, 256 (2):368–383, 2017.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. ISSN 08936080. doi: 10.1016/S0893-6080(98)00116-6.

Elizabeth Radetic and Martin Pelikan. Spurious dependencies and EDA scalability. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 303–310. ACM, 2010.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840, 2011.

Franz Rothlauf. *Design of Modern Heuristics*. Springer, Heidelberg, 2011.

Roberto Santana, Pedro Larrañaga, and José A Lozano. Combining variable neighborhood search and estimation of distribution algorithms in the protein side chain placement problem. *Journal of Heuristics*, 14(5):519–547, 2008.

Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978.

Selmar K Smit, Zoltan Szláavik, and Agoston E Eiben. Population diversity index: a new measure for population diversity. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 269–270. ACM, 2011.

Gilbert Syswerda. A study of reproduction in generational and steady-state genetic algorithms. In *Foundations of Genetic Algorithms*, volume 1, pages 94–101. Elsevier, 1991.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine learning*, pages 1096–1103. ACM, 2008.

Richard A Watson, Gregory S Hornby, and Jordan B Pollack. Modeling building-block interdependency. In *Parallel Problem Solving from Nature - PPSN V*, pages 97–106. Springer, 1998.

Hao Wu and Jonathan L. Shapiro. Does overfitting affect performance in estimation of distribution algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 433–434, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: 10.1145/1143997.1144078. URL `http://doi.acm.org/10.1145/1143997.1144078`.