Learning to Generate Genotypes with Neural Networks

Alexander W. Churchill alexanderchurchill@gmail.com
Queen Mary, University of London

Siddarth Sigtia sg@qmul.ac.uk
Queen Mary, University of London

Chrisantha Fernando c.fernando@qmul.ac.uk
Queen Mary, University of London

Abstract

Neural networks and evolutionary computation have a rich intertwined history. They most commonly appear together when an evolutionary algorithm is used to optimise the parameters and topology of a neural network for reinforcement learning problems or when a neural network is applied as a surrogate fitness function to aid the optimisation of expensive fitness functions. In this paper we take a different approach, asking the question of whether a neural network can be used to provide a mutation distribution for an evolutionary algorithm, and what advantages this approach may offer? Two modern neural network models are investigated, a denoising autoencoder modified to produce stochastic outputs and a generative model that learns a Bayesian network, the Neural Autoregressive Distribution Estimator. Results show that the neural network approach to learning genotypes is able to solve many difficult discrete problems, such as MaxSat and HIFF, and regularly outperforms other evolutionary techniques.

Keywords

Genetic algorithms, Estimation of Distribution algorithms, Autoencoder, Neural Autoregressive Distribution Estimator, Neural Networks.

1 Introduction

Evolutionary Algorithms typically employ static exploration methods, such as recombination and mutation, in order to traverse a search space. A problem with this approach is that "building blocks", by which we mean structural features of the search space, can be easily broken, discarding important information acquired during the search process. Addressing this problem are Estimation of distribution algorithms (EDAs), which attempt to statistically model sections of a search space in order to uncover underlying structure and guide search towards optimal solutions in an efficient manner (Pelikan et al. (2006)).

At the heart of any EDA lies a model-building process. Examples include Bayesian Networks, Markov Networks and K-Means clustering (Pelikan et al. (2002)). In this paper we introduce and investigate two neural-based methods for modelling: a denoising autoencoder (dA) and the Neural Autoregressive Distribution Estimator (NADE). An autoencoder is a feed forward neural network, consisting of at least one hidden layer, which is trained to reproduce its inputs from its outputs. Over the course of training, the hidden layer forms a, typically compressed, representation of the inputs, which has been used in various complex machine learning tasks(Hinton and Salakhutdinov (2006)). We explore two special types of autoencoder for learning effective genotypes. The dA is an autoencoder trained using inputs that are stochastically corrupted, which acts as a strong regulariser and also widens the basins of attraction. Although the dA is not a generative model, the encoding process can capture relationships between variables. When combined with stochastic outputs the dA can provide mutations to seen solutions, guided by the learnt encoded structure of the search space. We also investigate the NADE, an autoencoder instance that explicitly learns a generative model from data. The NADE has been shown to rival a Restricted Boltzmann Machine (RBM) in its performance in addition to being tractable unlike the RBM (Larochelle and Murray (2011)). Results show that an evolutionary algorithm incorporating the autoencoder models is able to outperform a canonical genetic algorithm (GA) across a range of combinatorial optimisation problems, as well as the PBIL and BOA EDA methods in several cases.

2 Background

Since their introduction in Holland (1975), evolutionary algorithms have themselves evolved, with extensive work exploring representation schemes, operators, parallelisation, co-evolution and the simultaneous optmisation of multiple objectives to name but a few areas. A key concern has been to reduce the sensitivity to initial parameters, e.g. self adaptive algorithms (Bäck and Schwefel (1993)), and automate the perceived dark art of designing genetic operators. Tackling the latter are Estimation of Distribution Algorithms (EDAs), also known as Probabilistic-Model-Building Genetic Algorithms, which are population based optimisers that typically replace genetic operators with a statistical model. The rationale behind the model building and related linkage learning approach (e.g. Thierens (2010); Harik (1999)) is that dependencies between variables can be captured and preserved. Search can be biased by sampling from the learnt structure of promising areas of the search space. Early work on EDAs concentrated on methods that explicitly modelled the probabilities of features occurring independently in a population of genotypes. These include the Compact Genetic Algorithm (Harik et al. (1999)), Baluja (1994)'s Population-Based Incremental Learner (PBIL) and Univariate Marginal Probability methods (Pelikan et al. (2002)). Improved success was found by modelling multivariate dependencies using clustering algorithms, e.g. the Extended Compact Genetic Algorithm (Harik (1999)); Bayesian networks, e.g **?**'s Bayesian Optimisation Algorithm (BOA); Markov networks and tree structures, among many others (Pelikan et al. (2002)). The popular and successful CMA-ES optimiser can be considered both a type of EDA and self-adaptive algorithm (Hansen et al. (2003)).

Artificial neural networks and evolutionary computation have a rich intertwined history. They most commonly appear together when an evolutionary algorithm is used to optimise the parameters and topology of a neural network for reinforcement learning problems or when a neural network is applied as a surrogate fitness function to aid the optimisation of expensive evaluation functions. In the first case, early research directly encoded the synaptic weights of a neural network, such as in Montana and Davis (1989). This has been found to have serious limitations and modern work, such as Hyper-NEAT (Stanley et al. (2009)), employs indirect encodings that allow both the weights and topologies of large networks to be successful optimised simultaneously. Other than using evolutionary techniques to effectively train neural networks, there has also been interest in employing neural networks directly in the search process. Early work has shown that including learning in the evolutionary process can improve the effectiveness and efficiency of optimisation, e.g. (Hinton and Nowlan (1987); Nolfi et al. (1994)). This is the standpoint taken in this paper but coming from a different perspective. The evolutionary algorithms in (Hinton and Nowlan (1987); Nolfi et al. (1994)) are applied to optimising a neural network and integrate learning through backpropagation (Rumelhart et al. (1988)) as a Baldwinian or Lamarckian process. Here, we use the unsupervised learning capabilities of neural networks to learn to generate incrementally more effective genotypes, either through a guided mutation (with the denoising autoencoder) or a learnt distribution (with the NADE).

With the 'neural networks revival' of the last decade, which has seen neural network methods frequently outperforming other machine learning techniques, there has been interest in neural-based methods in an EDA context, particularly for multi-objective optimisation. A Growing Neural Gas (GNG) was used as a model in Martí et al. (2008), employing a competitive Hebbian learning rule to cluster data without having to pre specify the number of groups. A shallow Restricted Boltzmann Machine (RBM) was used to model high dimensional data in (Tang et al. (2010); **?**), beating the state-of-the-art on several multi-objective continuous benchmark problems. Helmholtz machines have also been applied to this task with promising results (Zhang and Shin (2000)). Autoencoders (often referred to in the literature as auto-associators) are also neural-based method for unsupervised learning and have hitherto not been applied to combinatorial optimisation problems. As we will see in the next section, autoencoders have several interesting properties which can be favourably exploited to learn to generate genotypes in an evolutionary algorithm.

Another motivation for using an autoencoder approach is to investigate methods in which evolutionary algorithms can be implemented using neural structures. The *Neural Replicator Hypothesis* (Fernando et al. (2010)) proposes that evolutionary dynamics could operate in the brain at ontogenetic timescales. In Fernando et al. (2010), a (1+1)-ES is implemented in a network of spiking neurons. Adding Hebbian learning enabled linkages to be found between features, and the 128-bit HIFF problem to be solved significantly faster than by a genetic algorithm. However, Hebbian learning is restricted to learning only pairwise relationships between variables, while autoencoders have the potential to learn complex multivariate interactions. Although the training method employed in this paper is not thought to be biologically plausible, success with these models would suggest that unsupervised learning in neural structures can provide an effective means of evolutionary search.

3 Methods

3.1 Motivation

The main motivation for this paper is to assess whether neural network unsupervised learning methods can be used to learn genotypes. In the evolutionary computation literature this topic has mainly been investigated in the context EDAs. Typically, EDAs use unsupervised machine learning techniques to estimate distributions, i.e. build generative models, over the space of good quality solutions. Novel solutions are produced at each generation by updating and sampling from these distributions. An alternative approach is to use models that learn an encoding or compress the solution space, e.g. the generative grammar of **?**. Our proposed use of the denoising autoencoder deviates from the more standard generative model approach in the following ways. Instead of learning a distribution over the entire solution space, we directly learn a mapping from a given input to a distribution over potential solutions. By carefully choosing the inputs to the autoencoder and the degree of input corruption or perturbation, we can generate better solutions at each iteration. Therefore, model building with the autoencoder gives us a greater degree of control during sampling, by allowing the user to carefully select the inputs to the autoencoder used for sampling. The autoencoder can also be finetuned during training by varying the degree of corruption of the inputs. The autoencoder can be trained online with mini-batch gradient descent which allows us to update the model at each iteration without having to rebuild the model every time. This implies the network has a slowly decaying memory over good solutions and that interesting features learnt at one generation can be carried forward to future generations.

The alternative algorithm that we explore uses a NADE for model building, which is a distribution estimator for high-dimensional binary data. As briefly mentioned before, the NADE was initially proposed as a tractable alternative to the RBM, but has since been shown to be very effective in its own right. The RBM has been successfully applied to multi-objective optimisation problems in the past and therefore it would be interesting to investigate how to the NADE performs in an EDA. Theoretically, the NADE has several advantages over the RBM. Unlike the RBM, the gradients of the objective function for training the NADE can be exactly computed. The NADE can also be trained in mini-batches using stochastic gradient descent, excluding the need for model-rebuilding at each generation. Stochastic sampling from the NADE is an easier process compared to Gibbs sampling from the RBM, where the optimal length of the Gibbs chain needs to be empirically determined during evaluation. Another advantage of using the NADE is that it can be extended to modelling real-valued data (cite RNAD), unlike the BOA and hBOA.

3.2 The Autoencoder

A standard autoencoder consists of an encoder and a decoder network. The encoder performs an affine transformation followed by an element-wise non-linear operation. The mapping performed by the encoder is deterministic and can be described as:

$$h_\theta(\mathbf{x}) = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

The decoder is also a deterministic network and operates on the encoded input $h_\theta(x)$ to produce the reconstructed input:

$$r_{\theta'}(\mathbf{h}) = g(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

The outputs of the encoder network can be interpreted as the encoding or representation that the autoencoder learns. The encoder network can be viewed as a non-linear generalisation of PCA Hinton and Salakhutdinov (2006). The autoencoder learns interactions between the input attributes and maps it to the hidden layer via a non-linear transformation, making it a powerful model for learning and exploiting structure present in the best individuals.

Rather than using the outputs of the decoder network directly, we use them as parameters for a conditional distribution $p(X|Z = \mathbf{z})$ over the outputs of the network given an input $\mathbf{z}$. For binary optimization problems, the outputs $\mathbf{z}$ are considered to be parameters for the distribution $X|\mathbf{z} \sim \mathcal{B}(\mathbf{z})$ where $\mathcal{B}$ is the bernouilli distribution. For continuous parameter-optimisation problems, the outputs $\mathbf{z}$ parameterise a multi-variate normal distribution $X \sim \mathcal{N}(\mathbf{z}, \sigma^2)$, where the covariance matrix is assumed to be diagonal and the standard deviation along the diagonal is a tuneable parameter.

The autoencoder learns an encoding or representation of the input space in order to recreate the inputs with high accuracy. In order to force the autoencoder to learn

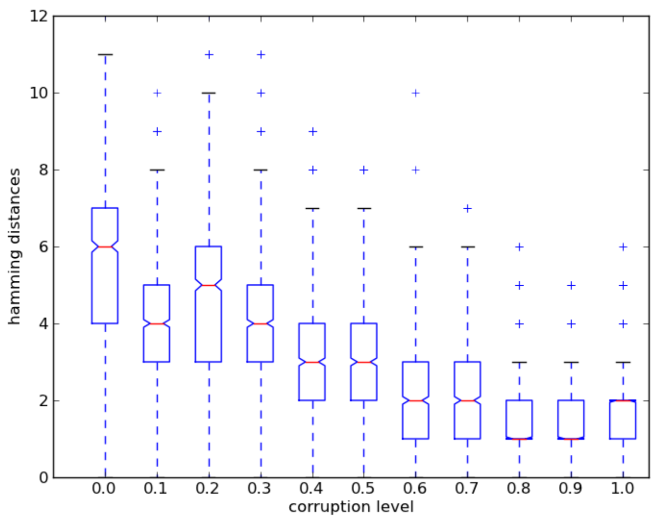


Figure 1: Hamming distances to optimal solution on the MaxOnes problem when training the dA with different corruption levels.

interesting representations, typically some kind of information bottleneck is applied to the representations. This bottleneck can either be enforced by reducing the dimensionality of the representation as compared to the input, or enforcing the hidden units to be sparse. A geometric interpretation of the representations learnt by an autoencoder is that it identifies the non-linear manifold on which the data is concentrated. The motivation for using an autoencoder is that it can discover structure, or alternatively the manifold on which the good solutions lie. Novel solutions can then be generated by exploring the manifold or sampling from the distribution learnt by an autoencoder. Due to the non-linear nature of the mapping learnt, it is possible to discover complex dependencies and structure in the data and therefore learn a more effective mutation distribution.

In our algorithm we use a denoising autoencoder which is a variant of the standard autoencoder Vincent et al. (2008). The dA tries to recreate the input $\mathbf{x}$ from 'corrupted' or 'noisy' versions of the input, which are generated by a stochastic corruption process $\tilde{\mathbf{x}} = \mathbf{q}(\tilde{\mathbf{x}}|\mathbf{x})$. Adding the denoising criterion to the model forces the autoencoder to learn more robust representations which are invariant to small perturbations of the input. Alternatively, the denoising criterion can be seen as learning an operator that maps points that are far from the manifold towards points on or near the manifold on which the good solutions are concentrated. Although denoising was introduced in order to encourage the learning of more robust representations, in our system we employ the denoising criterion to widen the basins of attraction around the best individuals in every generation. During training, we train the autoencoder on the most promising solutions, and by increasing corruption we encourage individuals that are far away from the training set to move towards the nearest high quality solution. By making use of the corruption noise as a tuneable parameter, the extent of the basins of attraction can be controlled, with high corruption giving rise to large basins of attraction.

We perform a simple experiment to test the effect of corruption noise on the outputs of the dA. We argue that increasing corruption noise widens the basin of attraction around the training examples. In this experiment, we train the GA on a 20-bit MaxOnes problem. The choice of the problem was motivated by the fact that MaxOnes has a unique optimum solution and the hamming distances from the optimal solution are easy to interpret. We allow the GA to run for 10 iterations and train the autoencoder on the best solutions at every iteration. The inputs to the dA are corrupted by a binary corruption process that stochastically flips a fixed proportion of bits for each example. The autoencoder is then trained to reconstruct the true targets from the corrupted inputs. After 10 iterations we present a set of randomly generated input vectors and sample a new population from the dA. We repeat this experiment with different corruption rates and the same set of input vectors and plot the hamming distances of the new population from the optimum solution. From figure 1 we observe that increasing the corruption rate leads to a decrease in the hamming distances. This shows that as corruption noise is increased, the outputs of the dA are more likely to be closer to an example in the training set.

3.3 Neural Autoregressive Distribution Estimator

Although the outputs of the decoder network can be viewed as the parameters of a probability distribution, the autoencoder is inherently a deterministic unsupervised learning algorithm. The Neural Autoregressive Distribution Estimator (NADE) is a Bayesian network for density estimation of high dimensional binary data. The NADE can be viewed as an autoencoder architecture devised such that the output units represent valid probabilities. The initial inspiration for the NADE was as a tractable approximation to the Restricted Boltzmann Machine (RBM). Restricted Boltzmann Machines have been used in EDAs for multi-objective optimisation in the past (reference). However, the NADE has been demonstrated to be an accurate density estimator on its own and has been used for various applications (refs).

The NADE belongs to the family of fully visible Bayesian networks that factorise the joint distribution of some D dimensional data point x as a product of one-dimensional conditional distributions:

$$P(x) = \prod_{i=1}^{D} P(x_i | x_{<i})$$

where $x_{<i} \equiv \{x_k, \forall k < i\}$ and D is the dimensionality of the input space. The main advantage of the factorisation above is that instead of trying to model a complex high-dimensional multi-modal distribution for x, the problem can be simplified to estimating the one-dimensional conditionals $P(x_i|x_{<i})$. A complex distribution can be estimated accurately if each conditional is learnt correctly. For the NADE:

$$P(v_i = 1 | v_{i<i}) = \sigma(b_i + W_{i,.} h_i)$$

$$h_i = \sigma(c + W_{.,<i} x_{<i})$$

where $W_{.,<i}$ denotes a sub-matrix of W with columns 1 to $i-1$. The above network is equivalent to a feed-forward neural network for each conditional, with weight sharing between the neural networks. The tied connections distinguish the NADE from other fully visible Bayesian networks and make it computationally tractable to train and obtain probabilities at test time. Another advantage while using the NADE is that the

log-likelihood and therefore the gradients can be computed exactly. This implies that the model can be trained with powerful gradient based optimisers without having to resort to sampling to estimate the log-likelihood like in training RBMs.

A theoretical limitation when using the NADE is the fact that the structure of the Bayesian graph is chosen *a priori*. The NADE joint distribution is explicitly ordered from left to right. The NADE also assumes a dependence between a visible unit and all units preceding the unit. In theory, the ordering of the NADE should not affect density estimation if we assume that each conditional can be learnt accurately. However, we observed that estimating the conditions exactly is hard and sometimes the model can learn erroneous dependencies that are not actually present in the data. These observations and limitations are discussed in detail in Section K.

3.4 Optimisation Algorithm

The pipeline of the optimisation algorithm is similar to canonical EDAs and is inspired by methods in HBOA (Pelikan (2005)). Pseudocode is provided in Algorithm 1 below. A population of P solutions is maintained, pop_t, and updated at each iteration, t. An initial population of solutions, pop_0, is drawn from a uniform distribution. These solutions are evaluated and the fittest unique x% are selected (i.e. truncation selection) to be in the training set, $trainingData_t$. The model, $model_t$, is then trained with $trainingData_t$ for E epochs. Following training, P solutions are stochastically sampled from the model and stored in $samples_t$. Solutions from $samples_t$ are then incorporated into pop_{t+1}.

There are slight differences in the algorithm based on the neural network model. For the denoising autoencoder (GA-dA), $trainingData_t$ consists of the fittest unique x% of solutions from the population are selected. For generating samples in GA-dA, a new set of solutions, $input_t$, is selected from P using tournament selection (with replacement). Each member of $input_t$ is inputted to the dA model, and the output vector, y_t, is sampled from using a binomial distribution, to produce a new solution. For the NADE model (GA-NADE), solutions are sampled from the model until $samples_t$ contains a set of solutions not present in pop_t, i.e. $samples_t \cap pop_t$. Both GA-dA and GA-NADE can use a niching technique. Although any method could be used, here we employ Restricted Tournament Selection (Pelikan (2005)). A solution from $samples_t$ is included in P_{t+1} if it is better than its closest neighbour in a randomly chosen subset of the population of size, W. Whether to use niching or not, as well as other parameters are determined through a grid search on a problem by problem basis.

4 Experiments

The neural network-based optimisation algorithms (GA-dA and GA-NADE) are tested on several difficult discrete problems and their performance is compared to a GA, PBIL and BOA. Below we describe the problems and problem instances tested and the comparison algorithms.

4.1 Problems

4.1.1 Multi-dimensional Knapsack Problem

Here we wish to choose of subset of N items to maximise the total value of items, z, while satisfying m constraints. We wish to maximise:

$z = \sum_{j=1}^{N} v_j x_j$, subject to $\sum_{j=1}^{N} w_{ij} x_i \leq c_i, i = 1, ..., m$

$x_i \in \{0, 1\}, j = 1, ..., N.$

Algorithm 1 Evolutionary optimisation with the NADE or denoising autoencoder models

```
$P \Leftarrow$ population size
$T \Leftarrow$ % population used for training
$E \Leftarrow$ no. epochs of backprop
$LR \Leftarrow$ learning rate for backprop
$H \Leftarrow$ no. hidden neurons
$NICHING \Leftarrow$ [True False]                    # Use niching?
$W \Leftarrow$ window size for niching
$EVALS \Leftarrow$ Max no. evaluations
$evalsCount \Leftarrow 0$
$model \Leftarrow$ dA NADE                           # Choose neural network model
$pop \Leftarrow$ Initialise population
while $evalsCount < EVALS$ do
  $trainingData = getTrainingData(model, pop, T)$
  $model = trainModel(model, trainingData)$
  if $model$ is dA then
    $input = tournamentSelection(pop, P)$
    $samples = sampleModel(model, P, input)$
  else
    $samples = sampleModel(model, P)$
  end if
  $evaluate(samples) = i$
  $evalsCount = evalsCount + P$
  if $NICHING$ is True then
    $pop = newPopulationNiching(pop, samples)$
  else
    $pop = samples$
  end if
end while
return $pop$
```

Two different instances are used in the results presented below. The first is the Weing8 instance Weingartner and Ness (1967), which has 105 items and two constraints (optimal solution is 602,319), and the second is a randomly generated instance with 500 items and one constraint (optimal solution is 100,104). Both instances are available in the supplementary material. If any of the m constraints are violated, the total value of chosen items is multiplied by -1.

4.1.2 Hierarchical If and only If

The Hierarchical If and only If (HIFF) problem was created by Watson et al., as an example of a function which is not separable (Watson and Pollack (1999)). It is a pathological function for a hill climber because of a very large number of local optima. The HIFF is structured as a balanced binary tree (Pelikan (2005)), where at each lower level a transfer function is applied to consecutive non-overlapping 2-bit partitions, $00 \rightarrow 0$, $11 \rightarrow 1$, anything else $\rightarrow$ null, to define the string in the layer above. At each level, 00 and 11 blocks contribute to the fitness by 2^{level}. Two global optima exist, a string of all 1s and all 0s. In the results section, the algorithms are applied to the 128-bit and 256-bit problem instances. In order to remove any possible bias towards an all 1s solution, solutions from the neural network models have a random bit mask (fixed before each trial) applied to them before evaluating fitness.

4.1.3 Royal Road

The Royal Road function as defined by Mitchell et al. (1992), divides a bit string, x, into a number of equally sized, non-overlapping partitions, $z_i \in [x_i, ..., x_{i+n}])$. If all of the bits in the partition match a predefined pattern, s_i , then the partition's fitness contribution is added to the overall fitness. The existence of these "Building blocks" was meant to act as a "royal road" for GAs compared to hill-climbers but bad alleles hitchhiking to good partial solutions slows down convergence speed. The 128-bit problem with 16 8-bit partitions is defined as,

$$f(x) = \sum_{i=1}^{16} \delta_i(x) o(s_i), \;\; \text{where } \delta_i(x) \begin{cases} 1, & \text{if } x \in s_i \\ 0, & \text{otherwise} \end{cases}$$

If a string contains two instances of s, a fitness of 16 is achieved. If all instances are found, fitness = 128. s_i instances are blocks of all 1s but as with the HIFF, a random mask is applied to solutions from the neural network models before fitness evaluation.

4.1.4 Maximum Satisfaction

The maximum satisfaction problem (MaxSat) is defined as finding an interpretation of predicates that satisfies the maximum number of clauses in a given predicate logical formula in conjunctive-normal form (CNF). This is known to be a difficult problem for GAs due to partial solutions leading away from the optimal solution (Rana and Whitley (1998); Pelikan and Goldberg (2003)). Here, algorithms are tested on a 3-CNF, 100-bit MaxSat problem obtained from SATLIB. The problem consists of 430 clauses and belongs to the phase transition region, which is the point at which the problem transitions from generally solvable to generally unsolvable. This instance is available in the supplementary material.

4.2 Comparison Algorithms

The neural networks are compared to a Genetic Algorithm (GA), Population-based Incremental Learning Algorithm (PBIL) and the Bayesian Optimisation Algorithm

(BOA). The GA was chosen to see whether GA-dA and GA-NADE outperform a canonical evolutionary algorithm and thus tests whether the method has potential. The GA uses two-point crossover and bit-flip mutation.

PBIL is a univariate estimation of distribution algorithm which operates on a single genetic string, a probability vector that represents the likelihood of each variable being present in the population. Each element in the probability vector, p^n, is initially set to 0.5. The algorithm learns online, updating p using,

$$p_i = (1-\alpha)p_i + \alpha b_i \quad \forall i \in [1...n]$$

where b is the best solution found in a set of samples from p at each iteration. At each iteration there is also a small chance of a mutation to each component of p. The algorithm was chosen to see how a univariate EDA compares to GA-dA and GA-NADE. Additionally, the two neural network methods are similar to PBIL as they also learn online.

BOA probabilistically models solutions present in a population by building a Bayesian network. At each iteration, a new Bayesian network, BN, is constructed from a selection of solutions from the population selected by truncation selection. The joint distribution encoded in the network is then sampled from, and the samples are incorporated into the population. BOA was selected for comparison because the model captures multi-variate dependencies and the NADE model approximates a Bayesian network. In BOA the Bayesian network is constructed using a greedy algorithm, that restricts each node to having two parents at most, to limit computational cost.

5 Results

The neural network methods and the three comparison algorithms described in Section 4.2 were tested on the discrete problems described above. Before comparison, a grid search was performed on the parameters of all five algorithms, on a problem by problem basis. The parameter configuration that achieved the best fitness (averaged over three trials) was selected for comparison. On each problem, the algorithms were assessed over 10 independent trials.

Figures 2 and 3 show plots of the best found fitness as the number of evaluations increase, averaged across ten independent trials, as well as boxplots showing the distribution of the best found solutions at the end of the optimisation process, for each of the six test problems. Results are also summarised in table 1. We observe that on four of the six problems, either GA-dA or GA-NADE performs the best, with BOA outperforming both on the two HIFF problems.

On the MaxSat problem, GA-NADE is the only algorithm that reaches the optimal solution on all 10 trials. Although both GA-NADE and GA-dA progress in a similar manner earlier on in the search process, the NADE method continues to improve and finds the optimal solution of 430 satisfied clauses much more quickly, within around 140,000 evaluations. Interestingly BOA is unable to find the optimal solution within the evaluation limit, although it performs much better than PBIL and the GA.

GA-dA is the clear winner on both knapsack problems. On the 500-item instance it is the only algorithm that locates the optimal solution and on the Weing8 instance (which has two constraints) it has a much greater success rate than the next best performer. On both problems it also improves its best solution much more quickly than the other algorithms. Both GA-NADE and BOA have much slower rates of improvement than GA-dA, although GA-NADE is considerably faster than BOA. BOA finds slightly

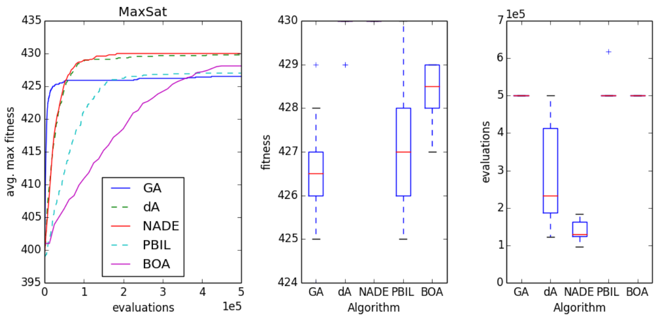
Experiment	Algorithm	Min	Max	Mean	Mean Evals	Success %
Royal Road 128	GA	120.0	128.0	127.20 ±2	42600.00 ±27034	90%
	dA	128.0	128.0	128.00 ±0	26500.00 ±3721	100%
	NADE	128.0	128.0	128.00 ±0	50900.00 ±9771	100%
	PBIL	112.0	128.0	123.20 ±6	74110.00 ±23463	60%
	BOA	128.0	128.0	128.00 ±0	38175.00 ±3118	100%
MaxSat	GA	424.0	429.0	426.50 ±1	500000.00 ±0	0%
	dA	429.0	430.0	429.78 ±0	291944.44 ±134967	70%
	NADE	430.0	430.0	430.00 ±0	141600.00 ±28182	100%
	PBIL	425.0	430.0	427.10 ±1	511800.00 ±35400	10%
	BOA	427.0	429.0	428.30 ±0	500000.00 ±0	0%
HIFF 128	GA	832.0	1024.0	940.80 ±86	416000.00 ±87429	50%
	dA	1024.0	1024.0	1024.00 ±0	231000.00 ±23537	100%
	NADE	1024.0	1024.0	1024.00 ±0	65500.00 ±17492	100%
	PBIL	560.0	772.0	652.00 ±76	500000.00 ±0	0%
	BOA	1024.0	1024.0	1024.00 ±0	29400.00 ±1854	100%
HIFF 256	GA	1664.0	2304.0	1984.00 ±225	1590500.00 ±626719	30%
	dA	2304.0	2304.0	2304.00 ±0	1355500.00 ±190793	100%
	NADE	2304.0	2304.0	2304.00 ±0	318333.33 ±82663	90%
	PBIL	1354.0	1544.0	1457.40 ±48	2000000.00 ±0	0%
	BOA	2304.0	2304.0	2304.00 ±0	98800.00 ±7493	100%
Knapsack 500	GA	10047.0	10093.0	10074.20 ±13	200000.00 ±0	0%
	dA	10096.0	10104.0	10102.70 ±2	121620.00 ±52114	70%
	NADE	10051.0	10075.0	10060.33 ±10	200000.00 ±0	0%
	PBIL	9984.0	10083.0	10039.00 ±31	200000.00 ±0	0%
	BOA	10039.0	10099.0	10079.40 ±17	200000.00 ±0	0%
Weing8 Knapsack	GA	619568.0	621086.0	620137.40 ±511	100000.00 ±0	0%
	dA	621086.0	624319.0	623615.40 ±1270	85400.00 ±19172	60%
	NADE	619886.0	624319.0	621483.50 ±1479	91550.00 ±17632	20%
	PBIL	620060.0	624319.0	621304.80 ±1566	97220.00 ±5600	20%
	BOA	578923.0	620821.0	599630.20 ±17700	100000.00 ±0	0%

Table 1: Results for the different search methods on 6 discrete problems. Showing the value of the minimum and maximum solution returned at the end of search, the mean best solution in the population after the end of search, the mean number of evaluations required to reach the optimum solution (or until the maximum number of evaluations) and the success rate for reaching the optimum, averaged across 10 trials.

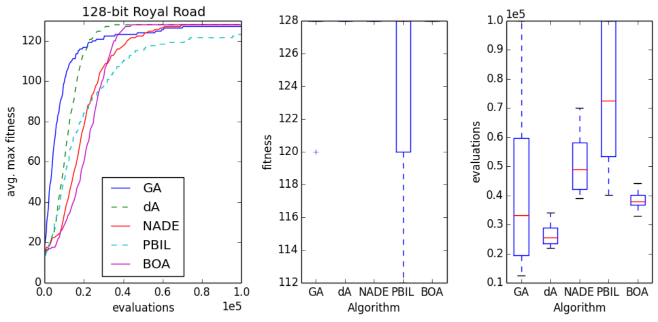
better solutions than GA-NADE on the 500-item knapsack but performs considerably worse on the Weing8 instance.

On both tested HIFF instances BOA is the far better performer, solving the 128-bit problem twice as fast and the 256-bit instance three times faster than GA-NADE. In turn, GA-NADE is able to solve the 128-bit instance in half the evaluations required from GA-dA and a quarter as many on the 256-bit instance. GA-dA is better than the standard GA on this problem, solving it faster and more consistently. The GA is not able to solve the HIFF instances on every trial, although it can locate the optimal solution while PBIL cannot.

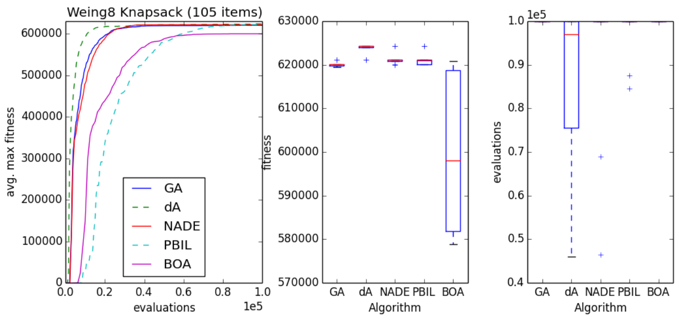
GA-dA displays the best performance on the Royal Road problem, consistently solving it within a small number of evaluations. The next best is the GA which can solve it more quickly but produces a much greater variance in discovery time. BOA is able to solve the problem much faster on average than GA-NADE, with low variance in terms of the number of evaluations taken to solve the problem.



(a) MaxSat



(b) Knapsack (500 items, 1 constraint)



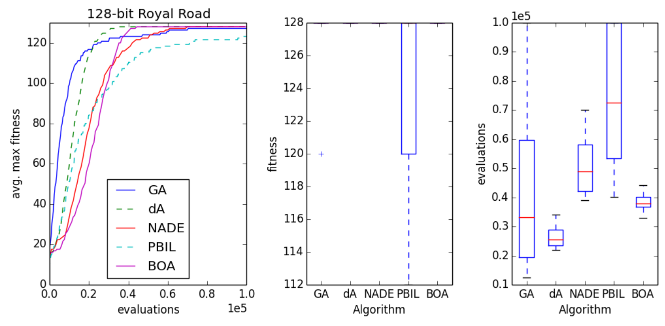
(c) Weing Knapsack (105 items, 2 constraints)

Figure 2: Showing results obtained by the different algorithms on MaxSat and Knapsack problems.

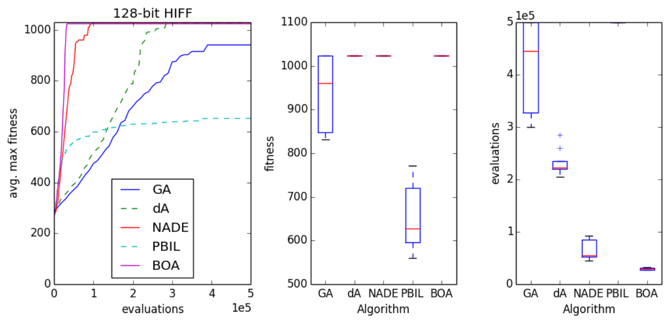
6 Discussion

6.1 Algorithmic Comparison

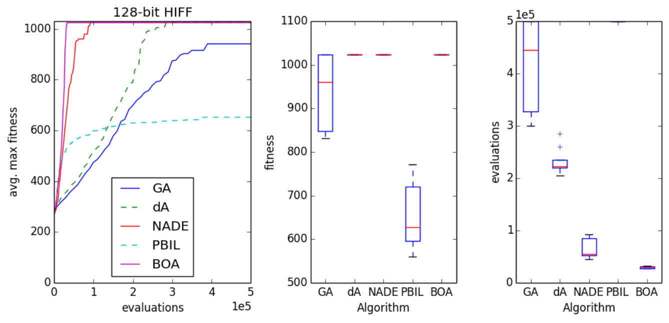
The results presented in the preceding section demonstrate that the denoising autoencoder (GA-dA) and NADE (GA-NADE) methods perform well across a number of difficult discrete problems. GA-dA outperforms the other methods on both knapsack problems and the Royal Road and GA-NADE is the only algorithm that is able to consistently solve the MaxSat problem instance. GA-dA outperforms a GA on all test problems in terms of the quality of the final solution and/or the number of evaluations re-



(a) 128-bit Royal Road



(b) 128-bit HIFF



(c) 256-bit HIFF

Figure 3: Showing results obtained by the different algorithms on Royal Road and HIFF problems.

quired. GA-NADE outperforms a GA on all of the problems apart from the Royal Road and the 500-item knapsack where they are not significantly different. BOA outperforms both neural network methods on the HIFF problems. It also outperforms GA-NADE on the Royal Road and Knapsack-500 problems but not on the others. This suggests that BOA is better able to discover the type of dependancy present in the HIFF problem. GA-NADE performs considerably better than GA-dA, which suggests that the Bayesian network-based generative model is well-suited to discovering the structure in the search space present in this problem. The poor performance of PBIL on the HIFF problem indicates that a model capable of learning complex multivariate dependencies is needed. The structure learnt by GA-dA allows it to significantly outperform the GA but it is much slower than BOA or GA-NADE on this problem.

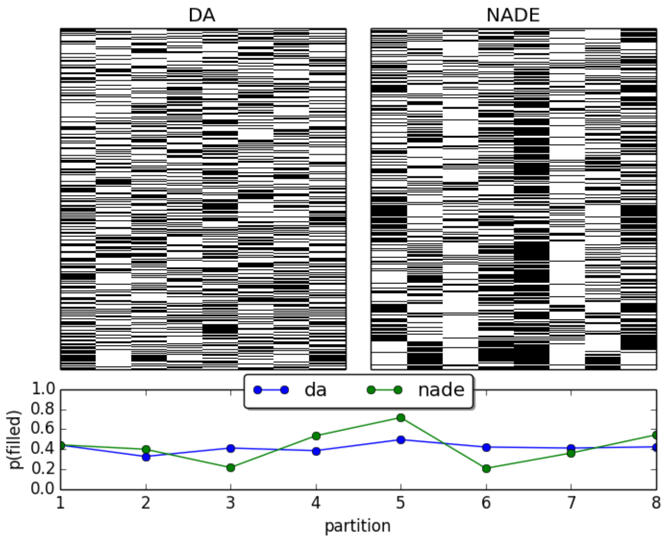
A MaxSat problem is difficult because there are a large number of interdependencies between variables and solving subproblems in the form of individual clauses can lead you away from the optimal solution (Rana and Whitley, 1998). GA-NADE is able to solve this problem consistently while BOA is not. This problem could be well suited to the NADE model which assumes a dependency between every variable - each variable is either a parent or a child of another. Thus a complex joint distribution of the variables is learnt. The dA is also able to learn multi-variate dependency with the potential advantage of having no precedence constraints. The tested implementation of BOA restricts each node to have at most 2 parents, which may adversely affect its ability to find solutions to this problem. Increasing this limit is prohibitively expensive.

The knapsack problems are interesting for two reasons, they have have real-world uses in resource management and they do not have obvious linkages. A 3-CNF MaxSat problem is complex because individual variables are present in multiple clauses and changing one variable can have a large impact on the fitness of the sequence. Due to the nature of the constraint handling of these knapsack problems, here a small change in the genotype can have a huge impact on fitness. If a solution changes from being under to over capacity, its value will swing from positive to negative. If there were no constraints, a knapsack problem would reduce to a MaxOnes problem and all variables would be fully independent. Variables can still be treated independently, which effectively will assign a higher probability to high value or denser items. Given more constraints the dependencies between groups of variables become more complex as certain combinations of items are required in order to satisfy the constraints. In terms of the rate of improvement, PBIL performs much better on the single constraint knapsack problem but takes much longer to get close to the optimal solution compared to the other algorithms. BOA has a much slower improvement rate on the Weing8 instance and does not find as good solutions. The GA uses a high crossover rate of 0.5 and thus swaps a large number of partial solutions. GA-dA has the fastest improvement rate and the highest mean score on both knapsack problems. This may be the type of problem for which it is best suited, as the dA has no ordering constraints, can capture dependencies between large groups of variables and, importantly, mutate around these solutions.

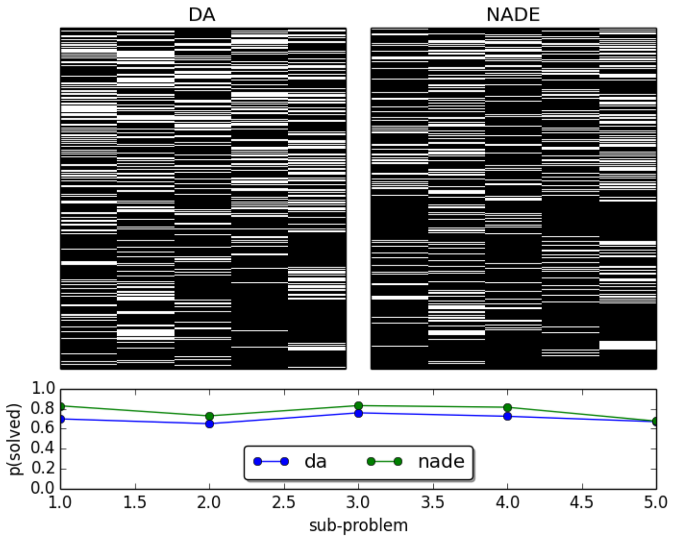
GA-dA is also the fastest consistent solver of the 128-bit Royal Road Problem. This problem is famous because it was used to identify the hitch-hiking phenomenon that can adversely affect the performance of GAs. For an EDA to perform well on this problem, it has to build a model that learns that either the partitions are independent or that every variable in the string is independent. PBIL suffers on this problem by falling into a local optima where certain variables have a very low probability of outputting a one. At this point it becomes very unlikely that certain subproblems will be solved. At the point of solving, BOA has learnt a model where every variable is independent. We will look more closely at the models produced by GA-dA and GA-NADE in the next section but GA-NADE clearly learns a complex relationship between variables that does not exist in the real problem. GA-dA hones in a very small part of the subspace, which allows it to quickly find the optimal solution.

6.2 Neural Network Methods Comparison

There are clear differences in the performance of the two Neural Network based algorithms, with GA-NADE performing much better than GA-dA on the HIFF and MaxSat



(a) A reduced view of each sub-partition for each sample



(b) A reduced view of each sub-problem for each sample

Figure 4: Showing samples from the dA and NADE on the Royal Road with Linkages problem.

problems but significantly worse on the others. The two methods discover structure in solution space in considerably different ways, with GA-NADE creating a generative model in the form of a Bayesian network and GA-dA through a compressed encoding of promising solutions. In this section we will first investigate the differences between the NADE and Autoencoder models on a specially designed function and then provide

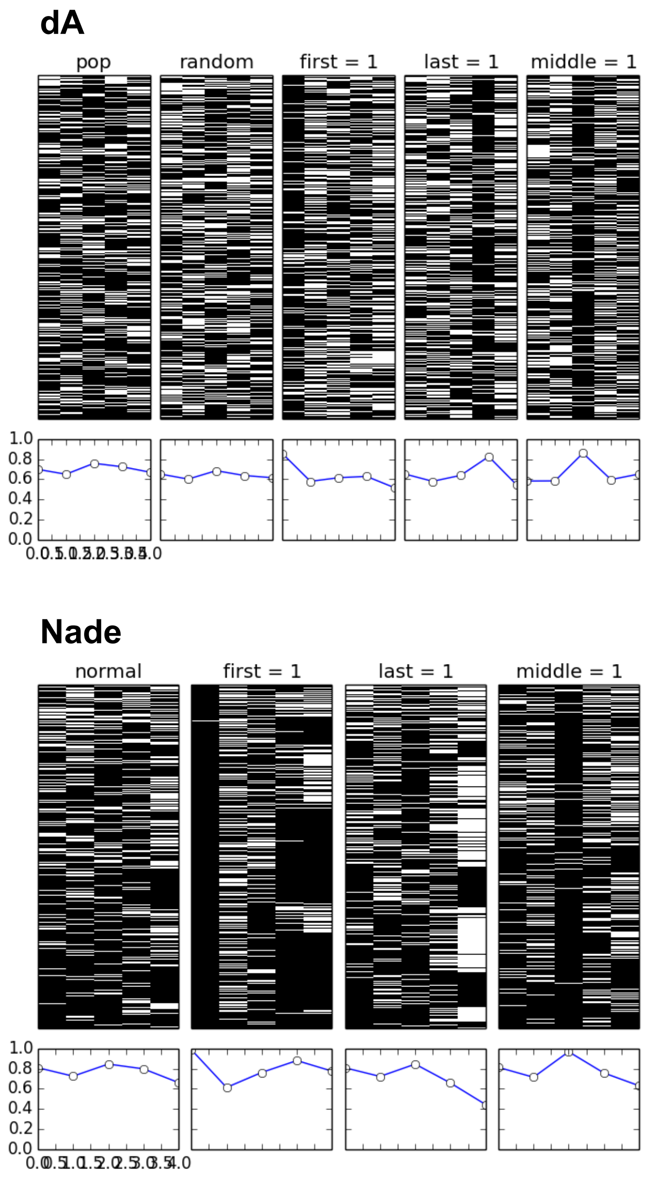
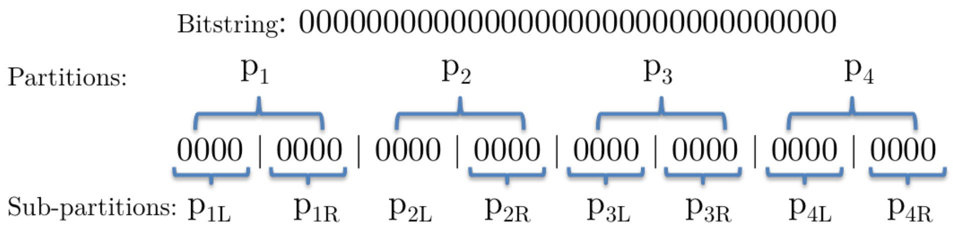


Figure 5: Showing sub-problem solutions from 50,000 samples from the dA and NADE given different conditions.

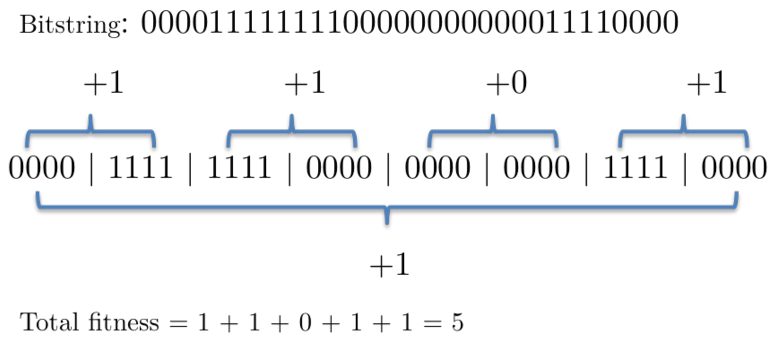
further analysis on a selection of the problems from the results section above.

6.2.1 A Royal Road with Linkages

A new objective function has been devised based on Mitchell et al.'s Royal Road (Mitchell et al., 1992). A solution consists of n non-overlapping partitions, $p_i i \in [1 \dots n]$, of size $2k$. p_i is further divided into two non-overlapping partitions of size k, $\hat{p}_{iL}$ on the



(a) Showing how a bit string is partitioned on the Royal Road with Linkages


Bitstring: 00001111111100000000000011110000
+1
+1
+0
+1
0000 | 1111 | 1111 | 0000 | 0000 | 0000 | 1111 | 0000
+1
Total fitness = 1 + 1 + 0 + 1 + 1 = 5

(b) An example optimal solution on the Royal Road with Linkages

Figure 6: Showing samples from the dA and NADE on the Royal Road with Linkages problem.

left side and $\hat{p}_{iR}$ on the right. If each bit in $\hat{p}_{iL}$ is equal to 1 and each bit in $\hat{p}_{iR}$ is equal to 0, 1 is added to fitness. Alternatively, a 1 is added to fitness if each bit in $\hat{p}_{iL}$ is equal to 0 and each bit in $\hat{p}_{iR}$ is equal to 1. Finally, an additional 1 is added to fitness if every bit in $\hat{p}_{1L}$ and $\hat{p}_{nR}$ are equal to 1, or if every bit in $\hat{p}_{1L}$ and $\hat{p}_{nR}$ is equal to 0, where $i = 1$ signifies the first partition and $i = n$ the last. An example is given for a 32-bit sequence in figure 6. This fitness function has been invented to explore how the algorithms deal with dependency between blocks of variables. There is linkage between the first and second halves of each partition, which is referred to below as 'local-linkage', and between the first k and the last k bits in the string, referred to as the 'global-linkage'.

GA-NADE and GA-dA are applied to a 32-bit version of this problem ($k = 4, n = 4$), both using a population size of 500. Samples produced by the models at the point that the optimal solution is found are explored in figures 4 and 5. In figure 4(a), samples have been reduced from 32 dimensions to 8, by displaying only the sub-partitions. If more than 2 bits in a sub-partition are equal to 1, the corresponding element in the string in the figure is assigned a 1, with a 0 otherwise. Underneath each figure a plot shows the overall bias, the probability of a sub-partition consisting of all 1s or all 0s, averaged across the samples. Figure 4(b) shows, for each sample, which of the four partitions have been solved (sub-problems 1 - 4), and whether every bit in $\hat{p}_{1L}$ and $\hat{p}_{nR}$ is equal to 1, or if every bit in $\hat{p}_{1L}$ and $\hat{p}_{nR}$ is equal to 0 (sub-problem 5). Therefore it shows which local-linkage sub-problems have been solved and whether the global-linkage problem has also been solved.

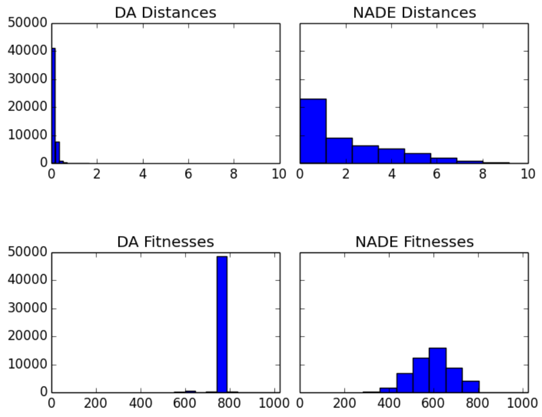
Figure 4 shows 500 samples from the dA and NADE, with the dA using the final population as input. In terms of sub-partition behaviour, the NADE exhibits a much clearer bias, with sub-partition 3 biased towards 0s and 4 biased towards 1s, and sub-partition 5 biased towards 1s and 6 biased towards 0s. This shows that the NADE has produced a peaked distribution in certain parts of solution space. Looking at the solution of sub-problems in figure 4(b), the samples generated by both algorithms solve the problems in all of the partitions with high probability. NADE displays a slightly higher probability of solving the local-linkage problems in partitions but both have an equal probability of solving the global-linkage problem in sub-problem 5.

We have seen in Figure 4 that although the algorithms produce samples from different distributions they have similar probabilities of solving the sub-problems. In figure 5 we now probe the models to investigate the structure that has been learnt. Figure 5(a) shows sub-problems solved by 50,000 samples from the dA given different input conditions. The first condition is the same as in figure 5, using the final population as input to the model. The second condition uses input strings sampled from a uniform random distribution. The third, fourth and fifth conditions also use samples from a uniform random distribution for input but fix the sub-partitions $\hat{p}_{1L}$ (referred to as c_{first}), $\hat{p}_{3L}$ (c_{middle}), and $\hat{p}_{4R}$ (c_{last}) to equal all ones, respectively.

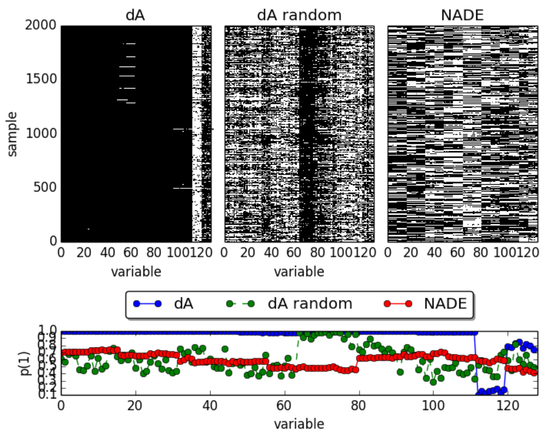
Using input from a uniform random distribution we see that sub-problem solutions follow a similar distribution to using the final population as input, but the probability of solving a sub-problem is slightly lower. With condition c_{first} there is an increase in the frequency of solutions to the first sub-problem, showing that the algorithm increases the probability of setting the sub-partition on the right to all zeroes, and suggesting that it has learnt the local-linkage in this partition. However, the probability of solving the global-linkage between the first and last sub-partitions, $\hat{p}_{1L}$ and $\hat{p}_{4R}$, has decreased, although not drastically. Similar behaviour is seen with c_{last}, where the probability of solving the sub-problem in the last partition is increased, however this leads to a small decrease in the probability of solving the global-linkage. For c_{middle} we see an increase in solving the third subproblem (which is independent from the other partitions) without any noticeable change in the probability of solving the other sub-problems.

In figure 5(b), we perform a similar probe on the NADE. Here, the first plot shows samples from the model with no bias applied to probabilities. The second, third and fourth plots sample from the same NADE model but fix the probability of a 1 in the sub-partitions $\hat{p}_{1L}$ (c_{first}), $\hat{p}_{3L}$ (c_{middle}), and $\hat{p}_{4R}$ (c_{last}) to be equal to 1, respectively. In constrast to the dA, with the NADE the c_{first} condition increases the probability of solving the first subproblem to almost 1, while also greatly increasing the probability of solving the long range dependency of the last sub-problem. The condition c_{middle} greatly increases the probability of solving the middle sub-problem while not affecting the distribution of solutions to the other sub-problems. Finally, the condition c_{last} has a negative effect, decreasing the probability of solving the last two subproblems, and affecting the long-range dependency particularly badly.

The samples displayed in figure 5 provide an interesting insight into the differences between the two models. The condition c_{first} provides a much greater performance increase with the NADE than with the dA. Setting the left half of the first partition to all ones means that the right half of the first partition must contain all zeros, the right half of the last partition must also contain all zeros and the left half of the last partition must contain all ones. This means that a complex sequence of linkages must be learnt. Figure 5 clearly demonstrates that the NADE is able to completely capture both the local and global linkages, while the dA only captures the local. However, the dA has certainly learnt complex dependencies, as in the c_{last} condition, while there is a small



(a) Showing the distribution of the mean distances from their 5 nearest neighbours and the fitnesses of 50,000 samples from the dA and NADE at the point that the optimal solution is found



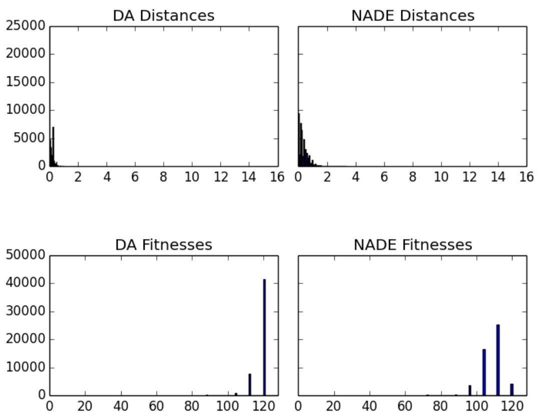
(b) Showing 2,000 random samples (first row) and the probability of a 1 at each locus averaged over 50,000 samples (second row).

Figure 7: Showing statistics from samples from the dA and NADE models on the 128-bit HIFF.

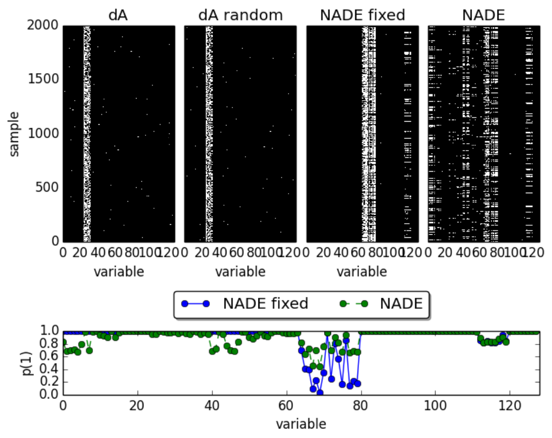
decrease in solving the long range dependency, it is not anywhere near as drastic as that displayed by the NADE, which means that it is influencing the first sub-partition. The dA does not suffer from ordering constraints, while the NADE does, therefore fixing the last sub-partition cannot influence the probabilities of the variables in earlier partitions. This implies that ordering could have a large influence on the performance of the NADE, which is a topic for further investigation. There is more flexibility in biasing the variables in the dA, which could be utilised to further guide the search process.

6.2.2 Tested Problems

We will now take a closer look at a selection of the test problems where there was a marked difference between GA-dA and GA-NADE: HIFF-128, the Royal Road and



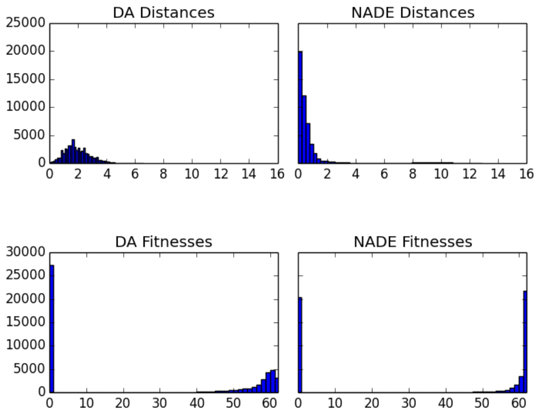
(a) Showing the distribution of the mean distances from their 5 nearest neighbours and the fitnesses of 50,000 samples from the dA and NADE at the point that the optimal solution is found



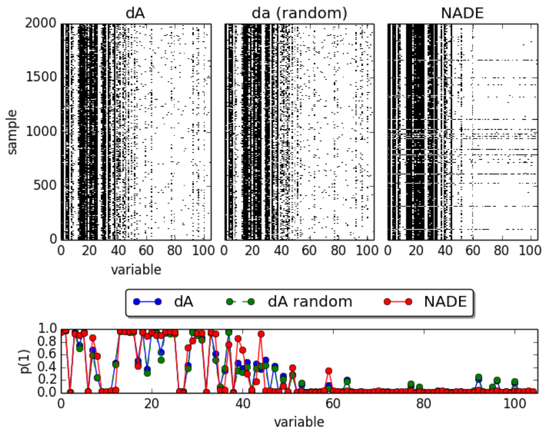
(b) Showing 2,000 random samples (first row) and the probability of a 1 at each locus averaged over 50,000 samples (second row).

Figure 8: Showing statistics from samples from the dA and NADE models on the Royal Road.

Weing8. Figure 7(a) shows the distribution of the mean distances from their 5 nearest neighbours and the fitnesses of 50,000 samples from the dA and NADE at the point that the optimal solution is found on HIFF-128. Here we see that there is much greater diversity in the samples produced by the NADE, while the dA has a much more peaked distribution. This can be partly explained by the fact that the dA does not use niching on this problem (as a result of the grid search). On this problem, most of the dA's solutions are small mutations around the local optima of 784, while the NADE produces a much more diverse set of samples. Looking at a selection of samples in figure 7(b), we see that the dA model using the population as input is extremely peaked, with almost all variables in the string set to 1 apart from the last 16 bits. Using input sampled from a uniform random distribution produces a more varied distribution of solutions,



(a) Showing the distribution of the mean distances from their 5 nearest neighbours and the fitnesses of 50,000 samples from the dA and NADE at the point that the optimal solution is found



(b) Showing 2,000 random samples (first row) and the probability of a 1 at each locus averaged over 50,000 samples (second row).

Figure 9: Showing statistics from samples from the dA and NADE models on the Weing8 Knapsack instance.

although the fitness of the samples is not as high. The NADE produces a much less biased distribution of samples, which explains its better performance on this problem. Of particular interest is that samples from the NADE contain partial solutions from both the 'all ones' and 'all zeroes' global optima, while the dA concentrates on the 'all ones' optima. This implies that NADE is better suited towards capturing the distribution of multi-modal search spaces.

Figure 8(a) shows a similar distribution of samples is produced by the dA on the Royal Road problem, with 15 of the 16 8-bit partitions almost always being all ones and mutations around a single unfilled partition. Although the NADE appears to produce more diverse samples, the distribution is peaked with a very low probability of a one on bits 64 - 80 (partitions 8 and 9). Fixing $p(x_i) = 1$ for the first 64 bits ($i \in [1 \ldots 64]$) in the

NADE modelling when sampling increases $p(x_i) = 0$ for $i \in [64 ... 80]$. This shows that a false dependency has been learnt linking earlier parts of the bit string to partitions 15 and 16, and it makes it very unlikely that the optimal solution will occur in a sample. This could be due to the topology of the dependency graph that the NADE is tied to. For the dA model, we see in this case that there is very little difference between using input from a random distribution or from the population, which is due to the high level of corruption ($p(c) = 0.9$).

For the Weing8 knapsack problem instance, the histograms in figure 9(a) show that the NADE produces slightly less diversity in its samples. The fitnesses of the samples are also heavily skewed towards very high scoring. Both models produce a large number of infeasible (and thus low scoring) samples. The raster plots in figure 9(a) indicate that both models concentrate on similar solutions, with a peaked response to many variables on the left side of the string (bits 50 and under). As indicated in the aggregate plot in figure 9(b), the NADE has a more peaked response to many variables than the dA. We see in the raster plot that there are a large number of samples from the NADE that use fewer items from the left side of the string and more from the right side (leading to the appearance of lines in the plot). This suggests that, similarly to the HIFF example, the NADE can model a multi-modal distribution, which could be useful for multi-modal search spaces or multi-objective optimisation. The corruption level is relatively high in the dA in this example ($p(c) = 0.25$), and there is only small differences between using the population and a random sample as input to the model. However, using the population as input does increase the probability of certain variables in a number of positions.

6.3 Areas for Further Investigation

In this paper we have demonstrated how the dA and NADE models can be successfully incorporated into an evolutionary search. Both models outperform a GA on number of problems (the dA is never beaten by a GA) as well as PBIL and BOA on certain problems. There are a number of areas that are currently being investigated that could potentially improve performance. Both models are trained online, and we have seen in the previous section that a bias creeps into the models. For example, in the Royal Road problem the NADE has a strong bias towards the bits in two partitions being set to 0, keeping it in a local optima for a long time. Training online allows search space structure discovered earlier in the search process to be reused later. It also helps to keep down the computational cost of model building. However, rebuilding the models either at certain predefined intervals, at every iteration or when probabilities begin to peak could reduce sample bias and improve the performance of the algorithm. As an aside, there could be another benefit to training online, which is the potential for transfer of the models between different related instances of the same class of problem. The question of transfer learning - can building a model to solve one task help improve performance on another? - has not been answered by the EDA community, although recent work has made a start (e.g. Hauschild et al. (2012)). Preliminary investigations suggest that using a dA, information learned in one task can be retained and recombined with that learnt in a second task to produce an across the board speed up on the third task in the sequence. Careful experimentation is needed to discover which types of problem can use machine learning techniques to transfer knowledge between unique instances and speed up optimisation as more examples are encountered.

In the last decade there has been a resurgence of interest in neural network-based methods, spurred by so called ‘deep architectures’. Both the dA and NADE could potentially benefit from the hierarchical features supported by deep networks and this needs to be investigated further. A potential problem with the NADE model is the precedence ordering of variables, which could have a great effect on the quality of the learnt distribution, as well as the training time. There are several paths that could be taken in this direction, for example having multiple models being learnt in parallel or using alternate methods to learn Bayesian networks at certain points in the search process and using the discovered ordering for the NADE model.

With both models, although especially with the NADE, the addition of local search could improve performance by providing more exploitation and fine-tuning of solutions, as seen in (Pelikan and Goldberg, 2003). This would allow the dA or NADE to power a more global search and a share of the responsibility for exploitation to the local search procedure. In the same vein, an interesting extension of the algorithms would be to hybridise them with a selecto-recombinative evolutionary algorithm (EA). Here, the EA could maintain the population of solutions, recombining and mutating them, while the neural network models are trained on the population. The EA could draw upon the models to generate genotypes, to guide search in a direction learnt from the structure of the solution space.

6.4 Real Valued, Mixed Integer and Multi-objective Problems

Both the NADE and dA can be modified to work with continuous or mixed-integer domains, providing an advantage over other discrete only EDA methods. As briefly mentioned earlier, the outputs of the dA can be interpreted as parameters of a distribution from which new solutions are sampled. Therefore, for real-valued problems, the outputs of the dA can be interpreted as means of a Gaussian distribution. Similarly, instead of predicting the means of a Bernouilli distribution, the NADE can be set up to predict the means and variances of a one-dimensional mixture of Gaussians. This real-valued variant of the NADE is called the RNADE (**?**). It is also be possible to mix Bernouilli and Guassian output units, to optimise Mixed Integer Problems, an under-represented area in EDA optimisation. The fact that the proposed algorithms can be generalised to real-valued problems is very useful since other popular EDAs like BOA and hBOA work only for discrete-valued data.

7 Conclusion

We have presented a novel stochastic evolutionary algorithm based on two neural network models that iteratively learn to produce more effective genotype. Training online, using the best found genotypes, the models are able to adaptively learn an exploration distribution, which guides search towards optimal solutions on difficult problems such as the 256-bit HIFF and a 430 3-CNF MaxSat problem, regularly outperforming a Genetic Algorithm and other EDA methods on a number of problems.

References

Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23.

Baluja, S. (1994). Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, DTIC Document.

Fernando, C., Goldstein, R., and Szathmáry, E. (2010). The neuronal replicator hypothesis. *Neural Computation*, 22(11):2809–2857.

Hansen, N., Müller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ecga. *Urbana*, 51(61):801.

Harik, G. R., Lobo, F. G., and Goldberg, D. E. (1999). The compact genetic algorithm. *Evolutionary Computation, IEEE Transactions on*, 3(4):287–297.

Hauschild, M. W., Pelikan, M., Sastry, K., and Goldberg, D. E. (2012). Using previous models to bias structural learning in the hierarchical boa. *Evolutionary Computation*, 20(1):135–160.

Hinton, G. E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex systems*, 1(3):495–502.

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

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. *Journal of Machine Learning Research*, 15:29–37.

Martí, L., García, J., Berlanga, A., and Molina, J. M. (2008). Introducing moneda: Scalable multi-objective optimization with a neural estimation of distribution algorithm. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 689–696. ACM.

Mitchell, M., Forrest, S., and Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and ga performance. In *Proceedings of the first european conference on artificial life*, pages 245–254. Cambridge: The MIT Press.

Montana, D. J. and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767.

Nolfi, S., Parisi, D., and Elman, J. L. (1994). Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28.

Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm*. Springer.

Pelikan, M. and Goldberg, D. E. (2003). Hierarchical boa solves ising spin glasses and maxsat. In *Genetic and Evolutionary ComputationGECCO 2003*, pages 1271–1282. Springer.

Pelikan, M., Goldberg, D. E., and Lobo, F. G. (2002). A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, 21(1):5–20.

Pelikan, M., Sastry, K., and Cantú-Paz, E. (2006). *Scalable optimization via probabilistic modeling*. Springer.

Rana, S. and Whitley, D. (1998). Genetic algorithm behavior in the maxsat domain. In *Parallel Problem Solving from NaturePPSN V*, pages 785–794. Springer.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

Tang, H., Shim, V. A., Tan, K. C., and Chia, J. Y. (2010). Restricted boltzmann machine based algorithm for multi-objective optimization. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

Thierens, D. (2010). The linkage tree genetic algorithm. In *Parallel Problem Solving from Nature, PPSN XI*, pages 264–273. Springer.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

Watson, R. A. and Pollack, J. B. (1999). Hierarchically consistent test problems for genetic algorithms. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2. IEEE.

Weingartner, H. M. and Ness, D. N. (1967). Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103.

Zhang, B.-T. and Shin, S.-Y. (2000). Bayesian evolutionary optimization using helmholtz machines. In *Parallel Problem Solving from Nature PPSN VI*, pages 827–836. Springer.