Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Scalability of using Restricted Boltzmann Machines for combinatorial optimization

Malte Probst[a,*], Franz Rothlauf[a], Jörn Grahl[b]

[a] Department of Information Systems and Business Administration, Johannes Gutenberg-Universität Mainz, Jakob-Welder-Weg 9, Mainz 55128, Germany
[b] Department of Management, Economics and Social Sciences, Universität zu Köln, Universitätsstr. 91, Cologne 50969, Germany

ARTICLE INFO

Article history:
Received 30 October 2014
Accepted 28 June 2016
Available online 1 July 2016

Keywords:
Combinatorial optimization
Heuristics
Evolutionary computation
Estimation of Distribution Algorithms
Neural Networks

ABSTRACT

Estimation of Distribution Algorithms (EDAs) require flexible probability models that can be efficiently learned and sampled. Restricted Boltzmann Machines (RBMs) are generative neural networks with these desired properties. We integrate an RBM into an EDA and evaluate the performance of this system in solving combinatorial optimization problems with a single objective. We assess how the number of fitness evaluations and the CPU time scale with problem size and complexity. The results are compared to the Bayesian Optimization Algorithm (BOA), a state-of-the-art multivariate EDA, and the Dependency Tree Algorithm (DTA), which uses a simpler probability model requiring less computational effort for training the model. Although RBM–EDA requires larger population sizes and a larger number of fitness evaluations than BOA, it outperforms BOA in terms of CPU times, in particular if the problem is large or complex. This is because RBM–EDA requires less time for model building than BOA. DTA with its restricted model is a good choice for small problems but fails for larger and more difficult problems. These results highlight the potential of using generative neural networks for combinatorial optimization.

1. Introduction

Estimation of Distribution Algorithms (EDA, Larrañaga & Lozano, 2002; Mühlenbein & Paaß, 1996) are metaheuristics for combinatorial and continuous non-linear optimization. They maintain a population of solutions which they improve over consecutive generations. Unlike other heuristic methods, EDAs do not improve solutions with mutation, recombination, or local search. Instead, they estimate how likely it is that decisions are part of an optimal solution, and try to uncover the dependency structure between the decisions. This information is obtained from the population by the estimation of a probabilistic model. If a probabilistic model generalizes the population well, random samples drawn from the model have a structure and solution quality that is similar to the population itself. Repeated model estimation, sampling, and selection steps can solve difficult optimization problems in theory (Mühlenbein & Mahnig, 1999) and in practice (Lozano, Larrañaga, Inza, & Bengoetxea, 2006). It is important to empirically assess the efficiency of using probability models in EDAs. Simple models, such as factorizations of univariate frequencies, can be quickly estimated from a population, but they cannot represent interactions between the decision variables. As a consequence, EDAs using univariate frequencies cannot efficiently solve complex problems. Using flexible probability models such as Bayesian networks allows complex problems to be solved, but fitting the model to a population and sampling new solutions can be very time-consuming.

Thus, a central goal of EDA research is the identification of probabilistic models that are flexible and can quickly be estimated and sampled. This is also a central topic in the field of machine learning. A recent focus in machine learning is the development of feasible unsupervised learning algorithms for generative neural networks. These algorithms and models can learn complex patterns from high-dimensional data sets. Moreover, generative neural networks can sample *new data* based on the associations that they have learned so far, and they can be fit to data (e.g., to a population of solutions) in an unsupervised manner. This makes them potentially useful for EDAs. Some of these models can also be “stacked” on several layers and be used as building blocks for “deep learning”.

In this paper, we focus on Restricted Boltzmann Machines (RBM, Hinton, 2002; Smolensky, Rumelhart, McClelland, & Group, 1986). RBMs are a basic, yet powerful, type of generative neural network where the connections between the neurons form a bipartite graph. Hence, they can model multivariate dependencies in

* Corresponding author. Tel.: +49 61313922734.
E-mail addresses: probst@uni-mainz.de (M. Probst), rothlauf@uni-mainz.de (F. Rothlauf), grahl@wiso.uni-koeln.de (J. Grahl).
URL: http://wi.bwl.uni-mainz.de (M. Probst), http://wiso.uni-koeln.de (J. Grahl)

http://dx.doi.org/10.1016/j.ejor.2016.06.066
0377-2217/© 2016 Elsevier B.V. All rights reserved.

the input variables. Due to recent work (Hinton, Osindero, & Teh, 2006), training RBMs is computationally tractable. They show impressive performance in classic machine learning tasks such as image or voice recognition (Dahl, Yu, Deng, & Acero, 2012).

Given these successes, it is not surprising that researchers have integrated RBMs and similar models using latent variables and distributed representations into EDAs and studied how these systems perform in optimization tasks. Zhang and Shin (2000) used a Helmholtz Machine in an EDA. Helmholtz Machines are predecessors of RBMs. Due to their limited performance, they are nowadays widely discarded. Zhang and Shin evaluated their EDA by comparing it to a simple Genetic Algorithm (Goldberg, 1989). They did not study the scaling behavior for problems of different sizes and complexity. In a series of recent papers, Tang, Shim, Tan, and Chia (2010), Shim et al. (2010), Shim and Tan (2012) and Shim, Tan, Chia, and Al Mamun (2013) studied EDAs that use RBMs. These works are similar to ours in that an RBM is used inside an EDA. An important difference is that they considered problems with multiple objectives. Also, they hybridized the EDA with clustering and particle swarm optimization. Thus, it is unknown if using an RBM in an EDA leads to competitive performance in single-objective combinatorial optimization.

Therefore, in this paper, we raise the following questions:

1. How efficient are EDAs that use RBMs for single-objective combinatorial optimization?
2. How does the run time scale with problem size and problem difficulty?
3. Is the performance competitive with the state-of-the-art?

To answer these questions, we integrated an RBM into an EDA (RBM–EDA) and conducted a thorough experimental scalability study. We systematically varied the difficulty of two tunably complex single-objective test problems (concatenated trap functions and NK landscapes), and we computed how the run time and the number of fitness evaluations scaled with problem size. Furthermore, we performed instance-based testing on the maximum satisfiability problem (MAX-SAT). We then compared the results to those obtained for the Dependency Tree Algorithm (DTA, Baluja & Davies, 1997) and the Bayesian Optimization Algorithm (BOA, Pelikan, 2005; Pelikan et al., 1999). DTA is an EDA assuming tree-structured dependencies between decision variables, whereas BOA is a state-of-the-art multivariate EDA.

In the scalability analysis, RBM–EDA solved the test problems in polynomial time depending on the problem size. Indeed, concatenated trap functions can only be solved in polynomial time by decomposing the overall problem into smaller parts and then solving the parts independently. RBM–EDA's polynomial scalability suggests that RBM–EDA recognized the structure of the problem correctly and that it solved the sub-problems independently from one another. The hidden neurons of the RBM (its latent variables) captured the building blocks of the optimization problem. The run time of RBM–EDA scaled better than that of BOA on trap functions of high order and NK landscapes with large k. RBM–EDA hence appears to be useful for complex problems. It was mostly faster than BOA if instances were large. DTA was fast for easy or small problems. However, due to its limited modeling power, it was unable to compete on the larger instances of the difficult problems. On the tested set of MAX-SAT problems, RBM–EDA performed better than DTA and BOA, on both the number of fitness evaluations, and run time.

The paper is structured as follows: in Section 2, we introduce EDAs in general, and DTA and BOA in particular. In Section 3, we introduce Restricted Boltzmann Machines, show how an RBM samples new data, and describe how an RBM is fit to given data. Section 3.4 describes RBM–EDA. The test functions, the experimental design and the results are presented in Section 4. Section 5 concludes the paper.

2. Estimation of Distribution Algorithms

We introduce Estimation of Distribution Algorithms (Section 2.1) and discuss the relevance of the used probabilistic model (Section 2.2). We present the two EDAs used for comparison: the Dependency Tree Algorithm (Section 2.3), as well as the Bayesian Optimization Algorithm (Section 2.4).

2.1. Functionality and applications

EDAs are population-based metaheuristics (Hauschild & Pelikan, 2011; Larrañaga, Etxeberria, Lozano, & Peña, 1999; Mühlenbein & Mahnig, 1999; Mühlenbein & Paaß, 1996; Pelikan, Goldberg, & Lobo, 1999). Similar to genetic algorithms (GA, Goldberg, 1989; Holland, 2001), they evolve a population of solutions over a number of generations by means of selection and variation. Unlike other evolutionary algorithms like GA, differential evolution (Storn & Price, 1997), or particle swarm optimization (Kennedy & Eberhart, 1995), EDAs do not improve individual solutions using mutation, recombination or local search. Instead, they use a probabilistic model to capture properties of the population, and use the model to sample new solutions. For an overview how EDAs relate to other types of heuristic search methods see e.g. Blum and Roli (2003) or Rothlauf (2011).

Algorithm 1 outlines the basic functionality of an EDA. After initializing a population P of solutions, the EDA runs for multiple generations. In each generation, a selection operator selects a subset $P_{parents}$ of high-quality solutions from P. $P_{parents}$ is then used as input for the variation step. In contrast to a GA, which creates new individuals using recombination and mutation, an EDA builds a probabilistic model M from $P_{parents}$, often by estimating their (joint) probability distribution. Then, the EDA draws samples from M to obtain new candidate solutions. Together with $P_{parents}$, these candidate solutions constitute P for the next generation. The algorithm stops after the population has converged or another termination criterion is met.

EDAs have demonstrated their performance in various domains. For example, single-objective EDAs have been used for flow shop scheduling (Ceberio, Irurozki, Mendiburu, & Lozano, 2014), resource scheduling with resource-dependent processing times (Liang, Chen, & Lozano, 2015), problems from statistical physics such as finding ground states for spin glasses (Pelikan & Hartmann, 2006), automated alphabet reduction for protein structures (Bacardit et al., 2007), protein folding (Santana, Larrañaga, & Lozano, 2008), side chain placement (Santana, Larrañaga, & Lozano, 2007), gene expression data clustering (Peña, Lozano, & Larranaga, 2004), optimizing cancer chemotherapy (Petrovski, Shakya, & McCall, 2006), or economic dispatch for power systems (Chen & Chen, 2007). In many of these domains, exhaustive algorithms, or approaches based on local search often struggle: as the problem size

Algorithm 1 Estimation of Distribution Algorithm.

```
1: Initialize Population $P$
2: while not converged do
3:     $P_{parents} \leftarrow$ Select high-quality solutions from $P$ based on their fitness
4:     $M \leftarrow$ Build a model estimating the (joint) probability distribution of $P_{parents}$
5:     $P_{candidates} \leftarrow$ Sample new candidate solutions from $M$
6:     $P \leftarrow P_{parents} \cup P_{candidates}$
7: end while
```

gets larger, they often cannot find optimal solutions, or even provide high-quality ones with reasonable computational effort. In the examples above, EDAs provide a viable way of finding high-quality solutions with limited computational resources.

We briefly discuss two examples where EDAs are able to beat the state-of-the-art not only in the field of evolutionary computation but in the field of optimisation.

Ceberio et al. (2014) use an EDA to solve the permutation flow shop problem, where a set of jobs has to be processed by a given number of machines. To model the distribution of high-fitness solutions, they use a probabilistic model which explicitly approximates distributions in the permutation domain. They augment their EDA with a variable neighborhood search (Mladenović & Hansen, 1997). Their approach obtains new best solutions for 152 of 220 instances of a well-known set of benchmark instances.

Liang et al. (2015) use an EDA based on the Boltzmann estimated distribution algorithm (BEDA, Mühlenbein & Mahnig, 2002a; 2002b; Mühlenbein, Mahnig, & Ais, 2003) to find solutions for an $\mathcal{NP}$-complete, discrete resource allocation problem with resource-dependent processing times. They compare their approach to a branch-and-bound algorithm, two local-search-based approaches, two EDA variants (UMDA and M_k-EDA), and ant colony optimization (for details see Liang et al., 2015). For a set of discrete benchmark problems, their approach yields the best performance for both the small- and large-scale problems.

For optimization problems where the fitness of an individual cannot be determined analytically, but is derived in a simulation, it is particularly hard to build a problem-specific heuristic. However, evolutionary approaches like EDAs can be used, as the basic selection mechanisms only need to know which of two solutions is the better one. For an illustrative example from military antenna design, see Santarelli et al. (2006).

Evolutionary algorithms in general and EDAs in particular are state-of-the art for many multi-objective problems. Here, the goal is not to find a single, best solution, but to approximate a Pareto front of solutions. As EDAs operate on populations of solutions, they can efficiently find multiple Pareto-optimal solutions in a single run of the algorithm (see e.g. Deb, 2014; Pelikan et al., 2006). For example, EDAs have been successfully applied to multi-objective knapsack problems (Shah & Reed, 2011), multi-objective monitoring network design problems (Kollat, Reed, & Kasprzyk, 2008), or the multi-objective travelling salesman problem (Shim & Tan, 2012). Also, probabilistic approaches closely resembling EDAs can be sucessfully applied for this kind of problems (Martí, García, Berlanga, & Molina, 2008, 2009).

2.2. Probabilistic EDA models

EDA variants mainly differ in their probabilistic models M. The models describe dependency structures between the decision variables with different types of probability distributions. Consider a binary solution space with n decision variables. A naive EDA could attempt to store a probability for each solution. Then, M would contain 2^n probabilities. This could be required if all variables depended on each other. However, computing and storing 2^n probabilities is computationally intractable for large n. If some decision variables are, however, independent from other variables, then the joint distribution could be factorized into products of marginal distributions and the effort and space required for calculating and storing M shrinks. If all variables are independent, only n probabilities have to be considered. In most problems, some variables are independent of other variables but the structure of the dependencies is unknown to those who want to solve the problem. Hence, model building consists of finding a network structure that matches the problem structure and estimating the model's parameters.

Simple models like the Breeder Genetic Algorithm (Mühlenbein & Paaß, 1996) or population-based incremental learning (Baluja, 1994) use univariate, fully factorized probability distributions with a vector of activation probabilities for the variables and choose to ignore dependencies between decision variables. Slightly more complex approaches like the bivariate marginal distribution algorithm or the Dependency Tree Algorithm use bivariate probability distributions which model pairwise dependencies between variables as trees or forests (Baluja & Davies, 1997; Pelikan & Mühlenbein, 1999). More complex dependencies between variables can be captured by models with multivariate interactions, like the Bayesian Optimization Algorithm (BOA, Pelikan et al., 1999), the extended compact GA (ECGA, Harik et al., 2006), Distribution Estimation using Markov random fields (DEUM, Shakya & McCall, 2007), or the Markov network EDA (MN-EDA, Santana, 2005).

Univariate models can lead to an exponential growth of the required number of fitness evaluations (Pelikan, 2005; Pelikan et al., 1999). Hence, multivariate models are often better suited for complex optimization problems. However, the computational effort to build a model M increases with its complexity and representational power. Many algorithms use probabilistic graphical models with directed edges, i.e., Bayesian networks, or undirected edges, i.e., Markov random fields (Larrañaga, Karshenas, Bielza, & Santana, 2012; Shakya & Santana, 2012).

In addition to the search for suitable models, much progress has been made on other EDA aspects. For example, there are EDAs for problems in the continuous or multi-objective domain, and for designing programs using variable-length representations in genetic programming. EDAs have been parallelized, hybridized with other methods, incorporated into mixture models, adapted to work on permutation-based representations, and used to predict the fitness of new solutions (for an overview, see Larrañaga et al., 2012; Lozano et al., 2006). The used model M plays a central role for EDA performance as it defines how interactions between variables can be considered. Hence, we limit the scope of this paper to analyzing different models, knowing that many of the above ideas and techniques are often applicable to EDAs with different model types.

2.3. Dependency Tree Algorithm

The Dependency Tree Algorithm (DTA) assumes that the probability distribution of a population of solutions can be modeled using a factorized, tree-like dependency structure (Baluja & Davies, 1997). Its model $P'(X_1, \ldots, X_n)$ has the form:

$$P'(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_{m(i)} | X_{m(p(i))}),$$

where $m = (m_1, \ldots, m_n)$ is an unknown permutation of $(1, \ldots, n)$, $p(i)$ expresses the tree structure by mapping the integers $0 < i \le n$ to integers $0 \le p(i) < i$, and $P(X_i|X_0)$ is by definition equal to $P(X_i)$ (see Baluja & Davies, 1997). Thus, each variable only depends on a single parent variable in a dependency tree.

Within this restricted domain, it is possible to find the optimal model yielding the best approximation using a variant of the Chow–Liu algorithm (Chow & Liu, 1968). DTA implements the Chow–Liu algorithm in the following way: it maintains an array A which keeps track of all counts $A[X_i = a, X_j = b]$ for every pair of variables X_i and X_j and every combination of binary assignments to a and b. Before starting with the first generation, all values of A are initialized to a constant c_{init}.

In each generation t, DTA decays all values in A by a constant factor $\eta < 1$. Subsequently, DTA selects the best ψ individuals from the population and increments $A[X_i = a, X_j = b]$ by one each time that $X_i = a$ and $X_j = b$ in the selected individuals. Based on A, DTA then calculates the marginal and conditional probabilities $P(X_i)$ and

$P(X_i|X_j)$ for all i and j, as well as the mutual information $I(X_i, X_j)$ as

$$I(X_i, X_j) = \sum_{a,b} P(X_i = a, X_j = b) * \log \frac{P(X_i = a, X_j = b)}{P(X_i = a) \cdot P(X_j = b)}.$$

A random X_i is then selected as the root of the dependency tree. Subsequently, among the variables which are not yet part of the tree, the variable with the highest I to any of the leaves of the tree is selected, and added as a child of this leaf. This step is repeated until all variables have been added to the tree. After the tree has been constructed, the appropriate conditional probability tables are added to the model.

To sample new individuals from the model's probability distribution, the root variable is sampled using its marginal distribution; all other nodes are then sampled in topological order using their conditional probabilities.

2.4. Bayesian Optimization Algorithm

The Bayesian Optimization Algorithm is the state-of-the-art EDA optimization algorithm for discrete multivariate optimization problems. It was been proposed by Pelikan et al. (1999) and has been heavily used and researched since then (Abdollahzadeh et al., 2012; Pelikan & Goldberg, 2003; Pelikan, 2008).

BOA uses a Bayesian network for modeling dependencies between variables. Decision variables correspond to nodes and dependencies between variables correspond to directed edges. BOA does not impose any assumptions on the structure of the dependencies. 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 to model 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 selected individuals, and repeats this process until no more edges can be added. The fit between selected individuals and the model is measured by the Bayesian Information Criterion (BIC, Schwarz, 1978). BIC is based on the conditional entropy of nodes given their parent nodes and correction terms penalizing complex models. BIC assigns each network G a scalar score

$$BIC(G) = \sum_{i=1}^{n} \left(-H(X_i|\Pi_i)N - 2^{|\Pi_i|} \frac{\log_2(N)}{2} \right), \quad (1)$$

where n is the number of decision variables, N is the sample size (i.e., the number of selected individuals), Π_i are the predecessors of node i in the Bayesian network (i's parents), and $|\Pi_i|$ is the number of parents of node i. The term $H(X_i|\Pi_i)$ is the conditional entropy of the ith decision variable X_i given its parental nodes, Π_i, defined as

$$H(X_i|\Pi_i) = -\sum_{x_i, \pi_i} p(x_i, \pi_i) \log_2 p(x_i|\pi_i), \quad (2)$$

where $p(x_i, \pi_i)$ is the observed probability of instances where $X_i = x_i$ and $\Pi_i = \pi_i$, and $p(x_i|\pi_i)$ is the conditional probability of instances where $X_i = x_i$ given that $\Pi_i = \pi_i$. The sum in (2) runs over all possible configurations of X_i and Π_i. The BIC score depends only on the conditional entropy of a node and its parents. Therefore, 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. The term $-2^{|\pi_i|} \ldots$ in (1) penalizes model complexity. BOA's greedy network construction algorithm adds the edge with the largest gain in BIC(G) until no more edges can be added. Edge additions resulting in cycles are not considered.

After the network structure has been learned, BOA calculates the conditional activation probability tables for each node. Once the model structure and conditional activation probabilities are available, BOA can produce new candidate solutions by drawing random values for all nodes in topological order.

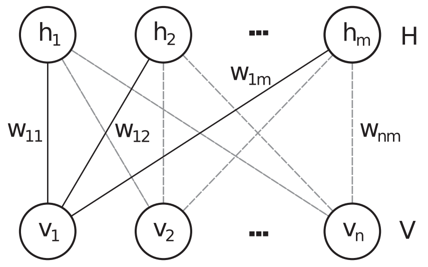


Fig. 1. A Restricted Boltzmann Machine as a graph. The visible neurons v_i ($i \in 1, \ldots, n$) can hold a data vector of length n from the training data. In the EDA context, V represents decision variables. The hidden neurons h_j ($j \in 1, \ldots, m$) represent m features. Weight w_{ij} connects v_i to h_j.

3. Restricted Boltzmann Machines and RBM–EDA

Restricted Boltzmann Machines (Smolensky, 1986) are stochastic neural networks that are successful in areas such as image classification, natural language processing, or collaborative filtering (Dahl, Ranzato, Mohamed, & Hinton, 2010; Hinton et al., 2006; Salakhutdinov, Mnih, Hinton, & Ghahramani, 2007). In this section, we describe the structure of Restricted Boltzmann Machines (Section 3.1), show how an RBM can sample new data (Section 3.2), and how contrastive divergence learning is used to model the probability distribution of given data (Section 3.3). Finally, we describe how an RBM can be used as probabilistic model of an EDA in the proposed RBM–EDA (Section 3.4).

3.1. Structure of RBMs

Fig. 1 illustrates the structure of an RBM. We denote V as the input (or "visible") layer. V holds the input data represented by n binary variables v_i, ($i = 1, 2, \ldots, n$). The m binary neurons h_j ($j = 1, 2, \ldots, m$) of the hidden layer H are called feature detectors as they are able to model patterns in the data. A weight matrix W holds weights $w_{i,j} \in \mathbb{R}$ between all neurons v_i and h_j. Together, V, H, and W form a bipartite graph. The weights W are undirected. Hence, an RBM forms a Markov random field.

In the sampling and training phase, each neuron in V and H makes stochastic decisions about whether it is active (its value then becomes 1) or not (its value then becomes 0). Therefore, it collects inputs from all neurons to which it is directly connected. W determines the strengths of the inputs.

An RBM encodes a joint probability distribution $P(V, H)$. In the sampling phase, a configuration of V and H is thus sampled with probability $P(V, H)$ (Smolensky, 1986). In the training phase, the weights W are adapted such that the marginal probability $P(V)$ approximates the probability distribution of the training data. With its latent variables, the RBM uses a distributed representation to model $P(V)$ (Hinton, 1986). Hence, the modeling capacity grows exponentially in the number of hidden neurons. That is, when a single hidden neuron is added, the number of possible configurations doubles (see e.g. Bengio, Ducharme, Vincent, & Janvin, 2003; Hinton, 1986). Nevertheless, training and sampling remain tractable because of the bipartite structure of the RBM. Hence, it is not necessary to define the problem structure beforehand.

3.2. Sampling

Assume that the RBM has already approximated the probability distribution of the training data. In other words, the weights W have been fixed during the training phase.

Algorithm 2 Pseudo code for sampling an RBM.

```
1: initialize $V$ (randomly, or with a $V$ close to the stationary distribution)
2: for a number of Gibbs sampling steps do
3:     $H \leftarrow$ sample $H|V$, i.e., set $h_j = 1$ with $P(h_j = 1|V)$ from (4)
4:     $V \leftarrow$ sample $V|H$, i.e., set $v_i = 1$ with $P(v_i = 1|H)$ from (3)
5: end for
6: Use $V$ as a sample from the marginal distribution $P(V)$
```

Now, the goal of the sampling phase is to generate new values for the neurons in the visible layer V according to $P(V, H)$. This is straightforward if the activations of the neurons in the hidden layer H are known. In this case, all v_i are independent of each other and the conditional probability that v_i is active is simple to compute. The conditional probability $P(v_i = 1|H)$ that the visible neuron v_i is active, given the hidden layer H is calculated as

$$P(v_i = 1|H) = \text{sigm}\left(\sum_j w_{ij} h_j\right), \tag{3}$$

where $\text{sigm}(x) = \frac{1}{1+e^{-x}}$ is the logistic function.[1] Analogously, given the activations of the visible layer V, the conditional probability $P(h_j = 1|V)$ for the hidden neurons H is calculated as

$$P(h_j = 1|V) = \text{sigm}\left(\sum_i w_{ij} v_i\right). \tag{4}$$

Although the two conditional distributions $P(V|H)$ and $P(H|V)$ are simple to compute, sampling from the joint probability distribution $P(V, H)$ is much more difficult, as it usually requires integrating over one of the conditional distributions. An alternative is to use Gibbs sampling, which approximates the joint distribution $P(V, H)$ from the conditional distributions. Gibbs sampling starts by assigning random values to the visible neurons. Then, it iteratively samples from $P(H|V)$ and $P(V|H)$, respectively, while assigning the result of the previous sampling step to the non-sampled variable (see Algorithm 2). Sampling in the order of $V \to H \to V \to H \to V \to \ldots$ forms a Markov chain. Its stationary distribution is identical to the joint probability distribution $P(V, H)$ (Geman & Geman, 1984). The quality of the approximation increases with the number of sampling steps. If Gibbs sampling is started with a V that has a high probability under the stationary distribution, only a few sampling steps are necessary to obtain a good approximation.

3.3. Training

In the training phase, the RBM adapts the weights W such that $P(V)$ approximates the distribution of the training data. An effective approach for adjusting the weights of an RBM is contrastive divergence (CD) learning (Hinton, 2002). The main element of CD is Gibbs sampling. CD minimizes the negative log-likelihood of the training data under the model. That is, it tries to find weights W which minimize the loss function $l(V;W) = -\log(P(V;W))$, for all examples $V^e, e \in (1, \ldots, \tau)$ in the training set:

$$W := \underset{W}{\text{argmin}} \frac{1}{\tau} \sum_{e=1}^{\tau} -\log(P(V^e; W)) \tag{5}$$

CD minimizes Eq. (5) by performing stochastic gradient descent. For each point V^e in the training data, CD learning updates w_{ij} in the direction of $-\frac{\partial \log(P(V^e))}{\partial w_{ij}}$. This partial derivative is the difference of two terms usually referred to as positive and negative gradient, Δ_{ij}^{pos} and Δ_{ij}^{neg} (for details, see Hinton, 2002). The total gradient Δw_{ij} is

$$\begin{aligned}\Delta w_{ij} &= \Delta_{ij}^{\text{pos}} - \Delta_{ij}^{\text{neg}} \\ &= <v_i \cdot h_j>^{\text{data}} - <v_i \cdot h_j>^{\text{model}},\end{aligned} \tag{6}$$

where $<x>$ is the expected value of x. Δ_{ij}^{pos} is the expected value of $v_i h_j$ when setting the visible layer V to a data vector from the training set and sampling H according to (4). Δ_{ij}^{pos} increases the marginal probability $P(V)$ of the data point V. In practice, the expectation is replaced by the point estimate $v_i h_j$. In contrast, Δ_{ij}^{neg} is the expected value of a configuration sampled from the joint probability distribution $P(V, H)$, which is approximated by Gibbs sampling. If $P(V)$ is equal to the distribution of the training data, in the expectation, the positive and negative gradient equal each other and the total gradient becomes zero.

Calculating Δ_{ij}^{neg} exactly is infeasible since a high number of Gibbs sampling steps is required until the RBM is sampling from its stationary distribution. Therefore, CD estimates Δ_{ij}^{neg} by using two approximations. First, CD initializes the Markov chain with a data vector from the training set, rather than using an unbiased, random starting point. Second, only a small number of sampling steps is used. We denote CD using N sampling steps as CD-N. CD-N approximates the negative gradient Δ_{ij}^{neg} by initializing the sampling chain to the same data point V which is used for the calculation of Δ_{ij}^{pos}. Subsequently, it performs N Gibbs sampling steps. Note that the first half-step $V \to H$ has, in practice, already been performed during the calculation of Δ_{ij}^{pos}. Despite using these two approximations, CD-N usually works well (Hinton et al., 2006).

Algorithm 3 describes contrastive divergence for $N = 1$ (CD-1). For each training vector, V is initialized with the training vector and H is sampled according to (4). This allows the calculation of Δ_{ij}^{pos} as $v_i h_j$. Following this, two additional sampling steps are carried out: first, we calculate the "reconstruction" $\hat{V}$ of the training vector as in (3). Subsequently, we calculate the corresponding hidden probabilities $P(\hat{h}_j = 1|\hat{V})$. Now, we can approximate Δ_{ij}^{neg} as $\hat{v}_i \cdot P(\hat{h}_j|\hat{V})$ and obtain Δw_{ij}. Finally, we update the weights as

$$w_{ij} := w_{ij} + \alpha \cdot \Delta w_{ij} \tag{7}$$

where $\alpha \in (0, \ldots, 1)$ is a learning rate defined by the user. This procedure repeats for several epochs, i.e., passes through the training set. Usually, CD is implemented in a mini-batch fashion. That is, we calculate Δw_{ij} in (7) for multiple training examples at the same time, and subsequently use the average gradient to update w_{ij}. This reduces sampling noise and makes the gradient more stable (Bishop, 2006; Hinton et al., 2006).

Algorithm 3 Pseudo code for a training epoch using CD-1.

```
1: for all training examples do
2:     $V \leftarrow$ set $V$ to the current training example
3:     $H \leftarrow$ sample $H|V$, i.e., set $h_j = 1$ with $P(h_j = 1|V)$ from (4)
4:     $\Delta_{ij}^{pos} = v_i h_j$
5:     $\hat{V} \leftarrow$ sample "reconstruction" $\hat{V}|H$, using (3)
6:     $\hat{H} \leftarrow$ calculate $P(\hat{H}|\hat{V})$ as in (4)
7:     $\Delta_{ij}^{neg} = \hat{v}_i \cdot P(\hat{h}_j|\hat{V})$
8:     $\Delta w_{ij} \leftarrow$ calculate all $\Delta w_{ij}$ as in (6)
9:     $w_{ij} \leftarrow$ update all weights according to (7)
10: end for
```

[1] In addition, all neurons are connected to a special "bias" neuron, which is always active and works like an offset to the neuron's input. Bias weights are treated like normal weights during learning. Due to brevity, we omitted the bias terms throughout the paper. For details, see Hinton et al. (2006).

Table 1
Overview of all RBM parameter settings.

Param	Description	Value	Comment
n	Number of visible neurons	= problem size	
m	Number of hidden neurons	$n/2$	
α	Learning rate (weights)	0.05	$\alpha = 0.025$ if $\gamma < 0.05$
α'	Learning rate (biases)	0.5	$\alpha' = 0.25$ if $\gamma < 0.05$
β	Momentum	0.5	$\beta = 0.8$ if $\gamma < 0.1$
γ	Relative decrease of error	–	Calculated each epoch
λ	Weight cost	0.0001	
	Mini batch size	100	
	Contrastive divergence steps	1	When training the RBM (CD-1)
	Gibbs sampling steps	25	When sampling new individuals

3.4. RBM–EDA: an RBM as EDA model

This section describes how we used an RBM in an EDA. Recall that an EDA uses a probabilistic model to capture the properties of promising solutions and sample new candidate solutions.

In each generation, this model approximates the probability distribution of the selected subpopulation $P_{parents}$ (line 4 of Algorithm 1). When using an RBM as EDA model, the model building phase corresponds to the training phase of the RBM (see Algorithm 3). $P_{parents}$ is hereby used as training data for the RBM. Note that, in contrast to other EDAs using Markov random fields like DEUM (Shakya & McCall, 2007), it is not necessary to specify a network structure in advance.

Subsequently, the EDA uses the trained model to generate candidate solutions $P_{candidates}$ for the next generation, as in line 5 of Algorithm 1. In RBM–EDA, this is done by sampling the trained RBM, and using the samples as $P_{candidates}$ (see Algorithm 2). Like a standard EDA, RBM–EDA then evaluates the fitness of the candidate solutions, and iterates to the next generation.

For the RBM–EDA, we have to specify a number of RBM-specific parameters (for the EDA-specific settings, please refer to Section 4.2). We chose the following parameter settings (see Table 1): the number m of hidden neurons was set to be half the number n of visible neurons (i.e. half the problem size). Standard values were used for the parameters that control the training and sampling phase (Hinton, 2012).

Sampling parameters. When sampling new candidate solutions from the RBM, we used the individuals in $P_{parents}$ to initialize the visible neurons V close to the stationary distribution. Subsequently, we performed 25 full Gibbs sampling steps.

Training parameters. The learning rate was set to $\alpha = 0.05$ and $\alpha' = 0.5$ for the weights and biases, respectively. We applied standard momentum (Qian, 1999), which adds a fraction β of the last parameter update to the current update, making the update less prone to fluctuations caused by noise, and dampening oscillations. We started with $\beta = 0.5$ and increased it during an EDA run to $\beta = 0.8$ (see below).

We used weight decay (L2 regularization) to improve generalization. Weight decay adds the term $-0.5 \cdot \lambda W^2$ to the RBM's optimization objective in Eq. (5). The partial gradient Δw_{ij} in Eq. (6) thus includes the term $-\lambda w_{ij}$, which decays large weights and thereby reduces overfitting. The weight cost λ determines the strength of the decay. We chose $\lambda = 0.0001$.

We used contrastive divergence learning with one Gibbs sampling step (CD-1) and a mini-batch size of 100. Furthermore, we initialized the visible biases with $\log\left(\frac{P(v_i=1)}{1-P(v_i=1)}\right)$, where $P(v_i = 1)$ is the probability that the visible neuron v_i is set to one in the training set (see Hinton, 2012). This initialization speeds up the training process.

Therefore, the overall update $\overline{w_{ijt}}$ for the RBM's weights at time t is calculated as

$$\overline{w_{ijt}} = \alpha(\Delta w_{ijt} - \lambda w_{ijt}) + \beta \cdot \overline{w_{ij(t-1)}}.$$

Parameter control. We applied a simple parameter control scheme for the learning rate α, the momentum β, and the number of epochs. The scheme was based on the reconstruction error e. The reconstruction error is the difference between a training vector V and its reconstruction $\hat{V}$ after a single step of Gibbs sampling (see lines 2 and 7 in Algorithm 3).[2] e usually decreases with the number of epochs. Every second epoch $t \in 1, \ldots, T$, we calculated for a fixed subset s of the training set S the relative difference

$$e_t^s = 1/|s| \sum_{j \in s} \sum_i |v_i - \hat{v}_i|/n$$

between v and $\hat{v}$. We measured the decrease γ of the reconstruction error in the last 25 percent of all epochs as $\gamma = \frac{e^S_{0.75t} - e^S_t}{e^S_0 - e^S_t}$. γ was then used to adjust the learning parameters. The learning rate α was initialized in the first epoch with $\alpha = 0.05$. As soon as $\gamma <$ .05, α was decreased to 0.025. In the first epoch we set the momentum β to 0.5. As soon as $\gamma < 0.1$ we increased β to $\beta = 0.8$.

We used two self-adaptive termination criteria and stopped the training phase as soon as one of the two criteria held. First, we stopped training if $\gamma < 0.01$. The rationale behind this was that the RBM was likely to have learned the most relevant dependencies between the variables. Continuing the training was therefore unlikely to improve the model considerably. Second, we stopped the training if the RBM was overfitting, i.e., learning noise instead of problem structure. Therefore, we split the original training set into a training set S containing 90 percent of all samples and a validation set S' containing the remaining 10 percent. We trained the RBM only for the solutions in S and, after each epoch, calculated the reconstruction error e^S and $e^{S'}$ for the training and validation set S and S', respectively. We stopped the training phase as soon as $\frac{|e^S - e^{S'}|}{e^{S'}} \geq 0.02$ (i.e., the absolute difference between the reconstruction errors was larger than 2 percent).

4. Experiments

We describe the test problems (Section 4.1) and our experimental design (Section 4.2). Sections 4.3.1–4.3.4 present results for the scalability of the algorithms, their performance for instances of the MAX-SAT problem, and results for RBM–EDAs using larger RBMs. We discuss the results in Section 4.3.5.

[2] CD learning does not minimize the reconstruction error e but maximizes $P(V)$, which cannot be calculated exactly tractably. As an alternative, the reconstruction error e is usually a good approximation for how good the model can (re-)produce the training data.

4.1. Test problems

We evaluated the performance of RBM–EDA on onemax, concatenated deceptive traps (Ackley, 1987), NK landscapes (Kauffman & Weinberger, 1989), and the maximum satisfiability problem (MAX-SAT, Du, Gu, & Pardalos, 1997). All four are standard benchmark problems, defined on binary strings of fixed length. Their difficulty depends on the problem size, i.e., problems with more decision variables are usually more difficult. Furthermore, the difficulty of concatenated deceptive trap functions and NK landscapes is tunable by a parameter. All four problems are additively decomposable (Mühlenbein & Mahnig, 1999).

The onemax problem assigns a binary solution x of length l a fitness value $f = \sum_{i=1}^{l} x_i$, which means the fitness of x is the number of ones in x. The onemax function is rather simple. It is unimodal and can be solved by a deterministic hill climber.

A trap function is defined on a binary solution x of length k. It assigns a solution x a fitness

$$f^k(x) = \begin{cases} k & \text{if } \sum_i x_i = k, \text{ and} \\ k - (\sum_i x_i + 1) & \text{otherwise.} \end{cases}$$

The optimal solution consists of all ones. Trap functions are difficult to solve because the second-best solution with fitness $k-1$ consists of all zeros and the structure of the function is deceptive. It leads guided search methods away from the global optimum towards the second-best solution.

A *concatenated* trap function places o trap functions of order k on a single bit string x of length $o \cdot k$. Its fitness is calculated as $f(x) = \sum_{i=1}^{o} f_i^k$ (Ackley, 1987). The problem difficulty increases with k as well as with o. Dependencies exist between the k variables of a trap function, but not between variables in the o different trap functions (Deb & Goldberg, 1993).

NK landscapes are defined by two parameters N and k and N fitness components f_i^N (Kauffman & Weinberger, 1989). A solution vector x consists of $l = N$ bits. The bits are assigned to N overlapping subsets, each of size $k+1$. The fitness of a solution is the sum of N fitness components. Each component f_i^N depends on the value of the corresponding variable x_i as well as k other variables. Each f_i^N maps each possible configurations of its $k+1$ variables to a fitness value. Thus, the overall fitness function is

$$f(x) = 1/N \sum_{i=1}^{N} f_i^N(x_i, x_{i1}, \ldots, x_{iK}).$$

Each decision variable usually influences several f_i^N. These dependencies between subsets make NK landscapes non-separable. 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. We used instances of NK landscapes with known optima from Pelikan (2008).

The MAX-SAT problem operates on binary strings x of length l. A problem instance is defined by a Boolean formula in conjunctive normal form (CNF), i.e., r conjunctive clauses consisting of u_r disjunctive literals (Du et al., 1997):

$$\text{CNF}: (l_{11} \vee l_{12} \vee \cdots \vee l_{1u_1}) \wedge \cdots \wedge (l_{r1} \vee l_{r2} \vee \cdots \vee l_{ru_r})$$

Each literal l_{ru_r} can map to any x_i, meaning that l_{ru_r} is TRUE, if $x_i = 1$. Furthermore, each of the literals in the CNF can be negated, such that $\neg l_{ru_r}$ is TRUE, if $x_i = 0$. The fitness of a solution string x is equal to the number of clauses which are satisfied[3]. Many real-world problems are in, or can be transformed into a MAX-SAT formulation (see Du et al., 1997). We used MAX-SAT instances with known optima from Competition (2014).

[3] There are other versions of the satisfiability problem where clauses can be given weights.

Table 2
Overview of EDA parameter settings (BOA and RBM–EDA).

Parameter/description	Value/comment
Max. number of generations	100
Max. generations w/o improvement	20
Tournament selection: tourn. size	2 (one parent, one offspring)

Table 3
Overview of DTA parameter settings.

Parameter/description	Value
Max. number EDA generations	1000
Max. gens. w/o improvement	200
Best selection: #best individuals ψ	4
Decay rate η	0.99
Initialization constant c_{init}	1000

4.2. Experimental setup

We studied the performance of RBM–EDA on the onemax function, concatenated deceptive traps with trap size $k \in \{4, 5, 6\}$, NK landscapes with $k \in \{2, 3, 4, 5\}$ and 100 instances of the MAX-SAT problem of size $l \in (70, 80)$, $r \in [600, 1200]$ clauses, with each clause consisting of $u = 3$ literals. We performed a scalability analysis for onemax, deceptive traps, and NK landscapes. We also report results for DTA and BOA so that RBM–EDA can be compared to other EDAs with different model complexities.

BOA and RBM–EDA shared the same settings for the EDA, i.e., they only differed in the choice of probabilistic model. They used tournament selection without replacement of size two (Miller & Goldberg, 1995), where each tournament contained one individual of the parents, and one individual of the candidate solutions. Hence, the selection scheme implicitly introduced some elitism (see e.g. Lozano et al., 2006).

BOA and RBM–EDA runs were stopped after 100 generations or 20 generations without improvement. For DTA, these numbers were increased tenfold. With these settings it was very unlikely that any EDA run was terminated prematurely, i.e, before convergence.

For DTA, we chose the same parameters as the original DTA (Baluja & Davies, 1997). Each entry in A is initialized to $c_{\text{init}} = 1000$, the decay rate was set to $\eta = 0.99$. In each EDA generation, DTA selects the best $\psi = 4$ individuals to update A. For an overview on all EDA and DTA parameter settings, see Tables 2 and 3.

We adopted an experimental setup similar to Pelikan (2008). We used bisection to approximate the smallest population size for which a method solved a problem to optimality. That is, for each algorithm and instance, the population size was systematically increased (or decreased) until (or as long as) the optimal solution was found, until lower and upper bound were within 10 percent of each other. For onemax and deceptive traps, we required each method to find the optimal solution in 30 out of 30 independent runs. For NK landscapes, we randomly selected 25 problem instances for each size N and k. For all NK problem instances as well as the 100 MAX-SAT instances, we determined the population size that solved five out of five independent runs of each instance.

We report the average number of fitness evaluations that were necessary to solve the problems to optimality, using the population sizes found by bisection. In addition, we report average CPU running times. CPU running time includes the time required for fitness calculation, the time required for model building (either building the Bayesian network, the dependency tree, or training the RBM), the time required for sampling new solutions, and the time required for selection. We also report CPU times even though

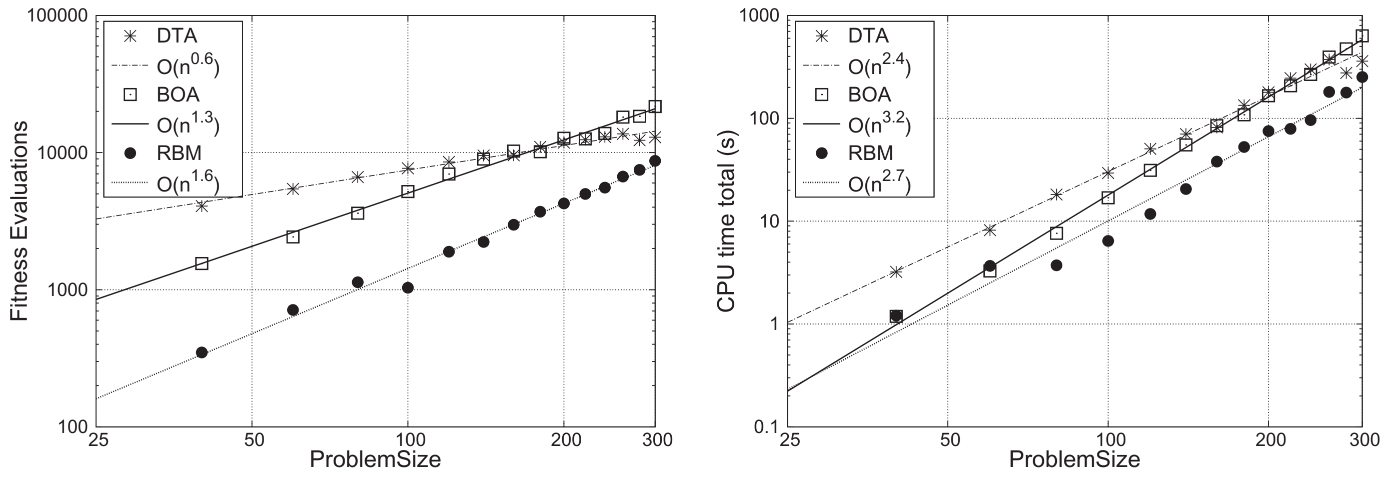


Fig. 2. Number of evaluations (left) and CPU time (right) over problem size l for the onemax problem.

Table 4
Approximated scaling behavior between the number of fitness evaluations and problem size, as well as CPU times and problem size.

	Fitness evaluations		CPU time	
	BOA	RBM	BOA	RBM
ONEMAX	$O(n^{1.3})$	$O(n^{1.6})$	$O(n^{3.2})$	$O(n^{2.7})$
4-TRAPS	$O(n^{1.8})$	$O(n^{2.8})$	$O(n^{3.9})$	$O(n^{3.0})$
5-TRAPS	$O(n^{1.8})$	$O(n^{2.4})$	$O(n^{4.0})$	$O(n^{2.8})$
6-TRAPS	$O(n^{1.9})$	$O(n^{2.3})$	$O(n^{4.2})$	$O(n^{2.8})$
NK $k = 2$	$O(n^{2.8})$	$O(n^{3.9})$	$O(n^{3.9})$	$O(n^{3.9})$
NK $k = 3$	$O(n^{3.6})$	$O(n^{4.3})$	$O(n^{6.4})$	$O(n^{3.5})$
NK $k = 4$	$O(n^{5.3})$	$O(n^{7.7})$	$O(n^{9.2})$	$O(n^{5.4})$
NK $k = 5$	$O(n^{4.5})$	$O(n^{5.2})$	$O(n^{7.9})$	$O(n^{4.9})$

previous EDA research mostly ignored the time required for model building and sampling.

We implemented RBM–EDA, DTA, and BOA in Java, using the same surrounding EDA code. The experiments were conducted on a single core of an AMD Opteron 6272 processor with 2100 MHz. The JBLAS library was used for the linear algebra operations of the RBM.

4.3. Results

We present results for the scalability analysis of all algorithms on onemax, concatenated deceptive traps, and NK landscapes (Section 4.3.1) as well as a detailed analysis on the CPU time for these problems (Section 4.3.2). Furthermore, we report results on several instances of the MAX-SAT problem (Section 4.3.3), and show for RBM–EDA that there is a trade off between model quality and execution time which can be controlled by the number of hidden neurons of the RBM (Section 4.3.4). We discuss the results in Section 4.3.5.

4.3.1. Scalability analysis

We report the performance of RBM–EDA, BOA, and DTA on onemax (Fig. 2), concatenated traps (Fig. 3), and NK landscapes (Fig. 4).

The figures have a log–log scale. Straight lines indicate polynomial scalability. Each figure shows the average number of fitness evaluations (left-hand side) and the overall CPU time (right-hand side) required until the optimal solution was found. The figures also show regression lines obtained from fitting a polynomial to the raw results for BOA and RBM–EDA (Table 4 shows a summary)[4].

First, we study the number of fitness evaluations required until the optimal solution was found (Figs. 2–4, left-hand side). For the onemax problem, RBM–EDA needed fewer fitness evaluations than BOA (Fig. 2, left-hand side) but had a slightly higher complexity ($O(n^{1.6})$ vs. $O(n^{1.3})$). DTA had a lower complexity ($O(n^{0.6})$), but needed more fitness evaluations than both BOA and RBM–EDA for small problems. For concatenated traps, BOA needed fewer fitness evaluations (Fig. 3, left-hand side). As the problem difficulty increased (larger k), the performance of BOA and RBM–EDA became more similar. The complexity of BOA was slightly lower than the one of RBM–EDA (around $O(n^{1.8})$ versus $O(n^{2.3})$ for larger problems). For DTA, the complexity of the number of fitness evaluations was not polynomial for the deceptive trap problems. For the larger problems, the number of fitness evaluations required by DTA was orders of magnitude higher than those required by both BOA and RBM–EDA. Hence, DTA was unable to solve deceptive trap problems with more than 200 decision variables for $k = 5$ and more than 120 decision variables for $k = 6$ within the given computational limits.

The situation was similar for NK landscapes, where BOA needed fewer fitness evaluations and scaled better than RBM–EDA, whereas DTA needed much more fitness evaluations on large problem instances (Fig. 4, left-hand side). Note that the results for DTA on the NK landscapes represent lower bounds (indicated in the plot by the shaded region above the DTA results). Again, due to computational restrictions, DTA was only able to solve a subset of the 25 NK landscape instances per problem size[5]. For the unsolved instances, we used the last result of bisection which completed, even if it did not solve the problem to optimality.

We discuss average total CPU times (Figs. 2–4, right-hand side). Besides the time required for fitness evaluation, this includes time spent for model building, sampling, and selection. For the onemax problem, RBM–EDA was faster than BOA and had a lower time complexity ($O(n^{2.7})$ vs. $O(n^{3.2})$, see Fig. 2, right-hand side). DTA needed more CPU time than RBM–EDA, but had a slightly lower complexity ($O(n^{2.4})$). For deceptive trap functions, BOA was faster on small problem instances, but its time complexity was larger that of RBM–EDA (Fig. 3, right-hand side). Hence, the

[4] We performed a linear regression of the form $y \approx a \cdot x + b$ in the log domain, i.e., $x = log(n)$, with n being the problem size, and $y = log(r)$, with $r =$ (num evals|CPU time). We then plot the line $\hat{y} = exp(a \cdot log(n) + b)$.

[5] DTA was unable to solve 2 of 425 instances for $k = 2$, 15 of 375 instances for $k = 3$, 9 of 275 instances for $k = 4$ and 28 of 250 instances for $k = 5$.

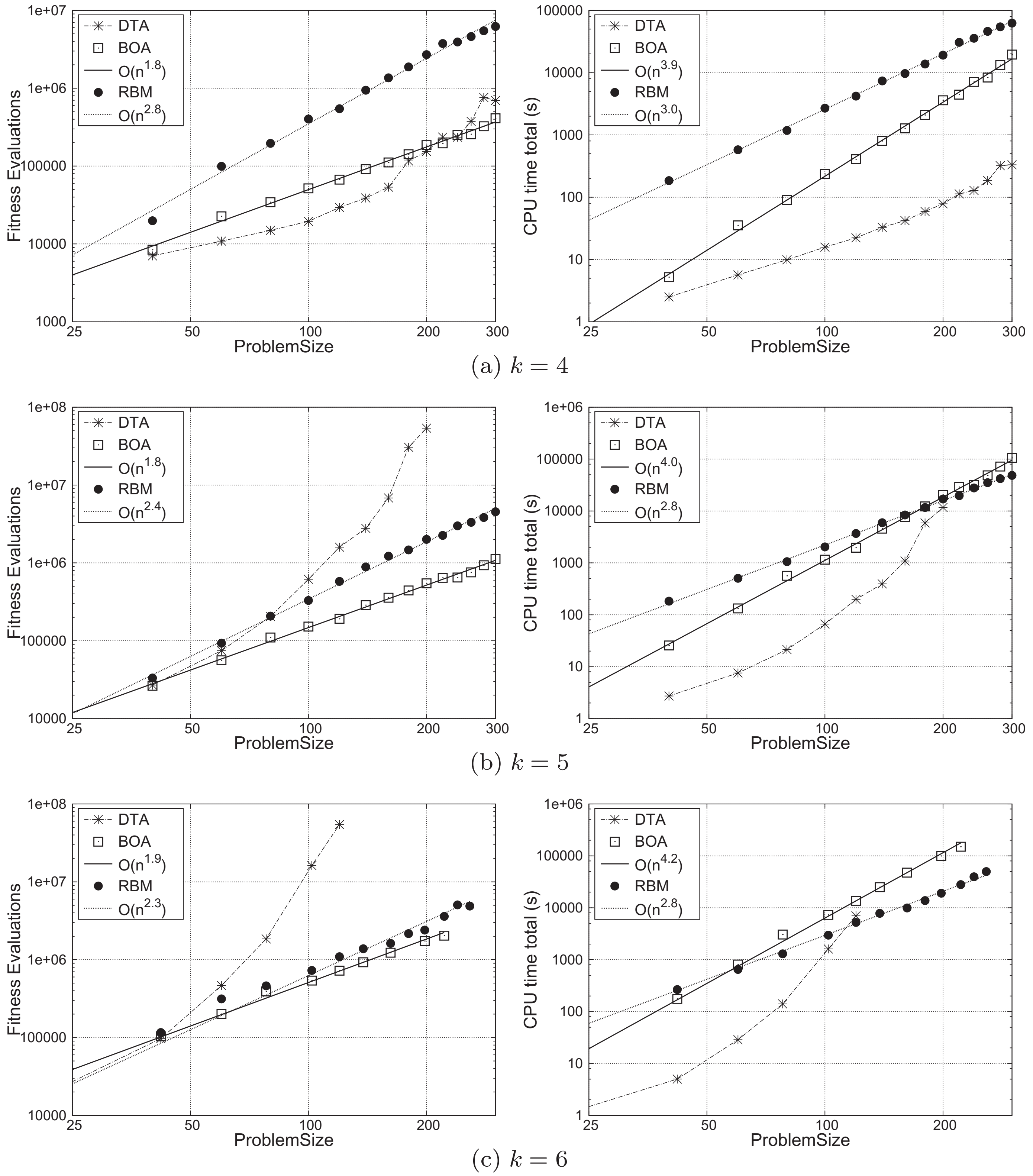
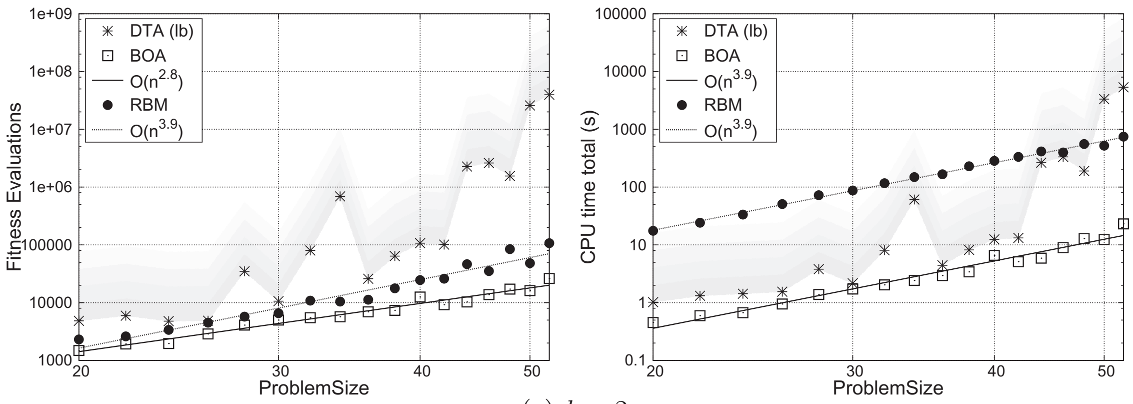


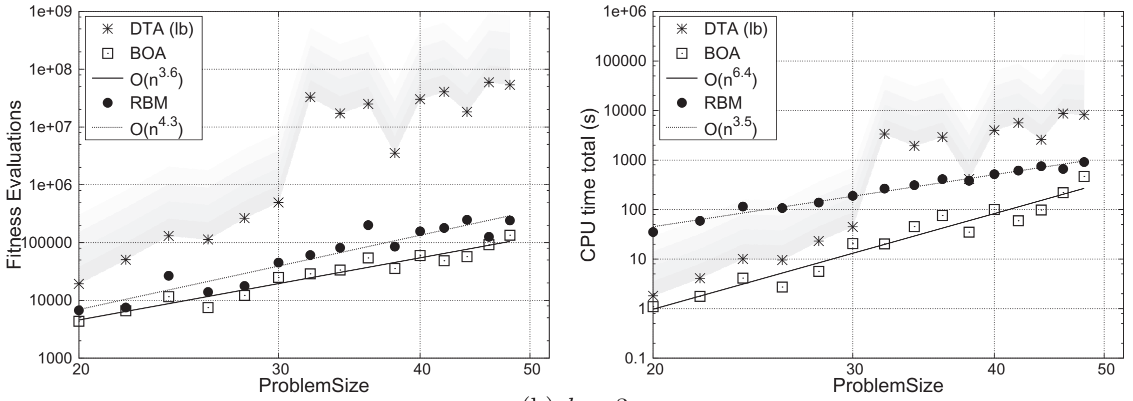
Fig. 3. Number of fitness evaluations (left) and CPU time (right) over problem size l for deceptive traps with (a) $k = 4$, (b) $k = 5$, and (c) $k = 6$. Lines connecting the points for DTA are for visualization purposes only.

absolute difference in computation time became smaller for larger and more difficult problems. For traps of size $k = 5$, RBM–EDA was faster than BOA for problems with more than 180 decision variables (36 concatenated traps). For traps of size $k = 6$, RBM–EDA was already faster for problems with more than 60 decision variables (10 concatenated traps). BOA's time complexity increased slightly with higher k (from $O(n^{3.9})$ for $k = 4$ to $O(n^{4.2})$ for $k = 6$). In contrast, the time complexity of RBM–EDA remained about constant (between $O(n^{3.0})$ and $O(n^{2.8})$).[6] DTA was able to solve small

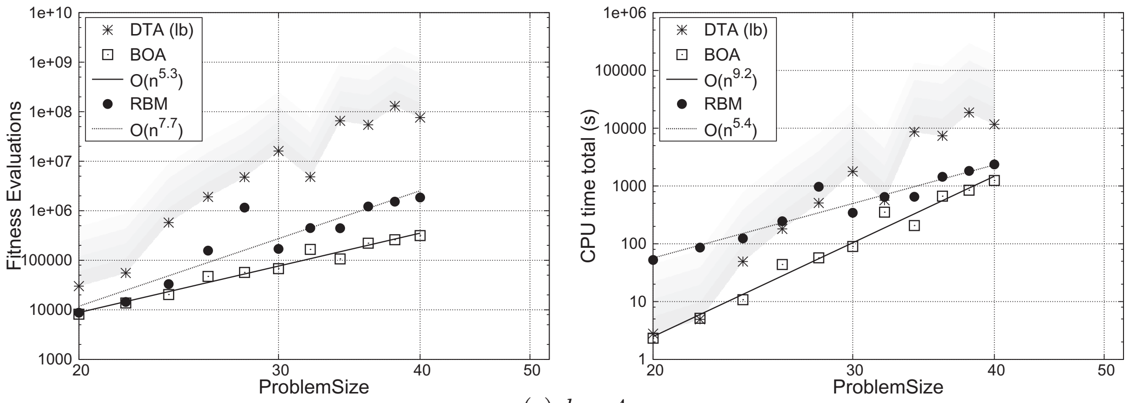
[6] Note that BOA was unable to solve deceptive 6-traps of sizes larger that 222 bits, due to the given computational limits.



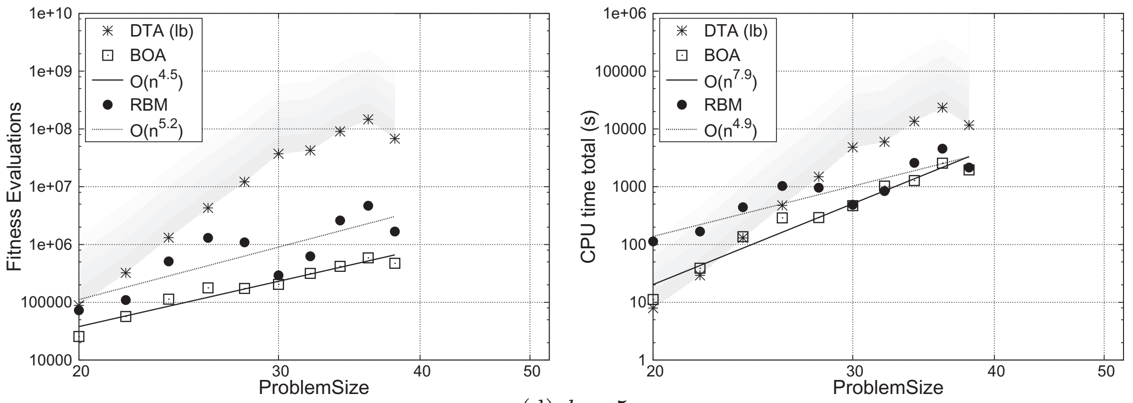
(a) $k = 2$



(b) $k = 3$



(c) $k = 4$



(d) $k = 5$

Fig. 4. Number of evaluations (left) and CPU time (right) over problem size $l = N$ for NK landscapes with (a) $k = 2$, (b) $k = 3$, (c) $k = 4$, and (d) $k = 5$. Note that all results for DTA are lower bounds (lb), indicated by the shaded area (DTA was not always able to solve all of the 25NK landscape instances per problem size, see text).

problems faster than both BOA and RBM–EDA. However, the exponential growth in the number of fitness evaluations could be observed for the CPU time as well.

The results for NK landscapes were qualitatively similar (Fig. 4, right-hand side). BOA was faster than EDA–RBM, but its computational effort increased stronger with n. Therefore the computation times became similar for difficult and large problems (cf. results for $k = 5, n \in 30, 32, 38$). For DTA, the lower bounds on CPU time required to solve larger problem instances were higher than the results for both BOA and RBM–EDA, for all k.

4.3.2. CPU time, model building, and fitness evaluation costs

The results of the previous section indicate that DTA was competitive only for the simple onemax problem, and for smaller instances of deceptive traps and NK landscapes. We therefore focus on BOA and RBM–EDA. Although RBM–EDA needed a higher number of fitness evaluations, its overall computational effort was about similar or even lower than BOA, especially for difficult problems. For this reason, we analyze in greater detail the differences in computations effort (CPU time) between BOA and RBM–EDA. Fig. 5 exemplarily shows the absolute CPU times for model building (left-hand side) and relative CPU times necessary for model building, sampling, and fitness evaluation (right-hand side) for onemax, concatenated 5-traps and NK landscapes with $k = 5$ for various problem sizes[7],[8].

First, we study the absolute CPU times for model building (Fig. 5, left-hand side). For all three problem types, RBM–EDA's model building had a lower CPU time complexity (Onemax: $O(n^{2.7})$ vs. $O(n^{3.2})$, 5-traps: $O(n^{2.7})$ vs. $O(n^{4.1})$, NK landscapes $k = 5$: $O(n^{4.3})$ vs. $O(n^{8.1})$). Model building was faster for RBM–EDA than for BOA for all onemax problems, for concatenated 5-traps of size $>= 140$ and for NK-$k = 5$ problems of size $\in (30, 32, 36, 38)$. Note that this was despite the fact that the population size, i.e., the training set, was usually bigger for RBM–EDA.

Second, we analyze the relative amount of CPU time that each EDA part used (Fig. 5, right-hand side). For BOA, model building dominated the CPU usage with more than 95 percent of the total CPU time for all problems. Furthermore, with growing problem size, the share monotonously increased (95.2–99.5 percent for onemax, 97.4–99.8 percent for 5-traps, 85.6–97.2 percent for NK with $k = 5$). The relative times for sampling and fitness evaluations were very low. Furthermore, their shares of the total time decreased with growing problem size, despite the growing population sizes. In contrast, the RBM spent a significantly smaller fraction of the total time for model building. Also, with growing problem size, this fraction decreased or stayed relatively constant (85.7–88.3 percent for onemax, 92.2–64.3 percent for 5-traps, 78.8–49.2 percent for NK with $k = 5$). Correspondingly, the CPU time fractions for sampling and fitness evaluations increased with growing problem size (except onemax). Hence, in RBM–EDA the CPU time spent for model building is lower in comparison to BOA[9].

Third, we study the influence of the time for fitness evaluations on the total run times of BOA and RBM–EDA. Recall that RBM–EDA needed a higher number of fitness evaluations than BOA, but was faster for concatenated traps of size $k = 5$ and $k = 6$, for problems larger than a particular size (c.f. Fig. 3). For those instances, we calculate by which factor M the cost of a single fitness evaluation has to be multiplied to result in the same total running time for BOA and RBM–EDA. In other words, if an individual fitness evaluation is more than $M \times$ more expensive, using BOA will be faster than RBM–EDA on this problem size.

Fig. 6 plots M for the concatenated traps problem with $k = 5$ (Fig. 6(a)) and $k = 6$ (Fig. 6(b)). For $k = 5$, M is in between 40 and 770, for $k = 6$, M ranges between 540 and 14,000. With growing problem size, M tends to increase. Hence, for a problem with a similar complexity as concatenated traps, RBM–EDA would be faster than BOA for large problem sizes even if the fitness evaluation cost were hundreds ($k = 5$) or thousands ($k = 6$) of times higher. Moreover, with growing problem size, the impact of the cost of fitness evaluations on RBM–EDAs's advantage of total computation time tends to decrease.

For all problems, DTA's time and complexity for model building (not plotted in the figures) were lower than those of both BOA and RBM–EDA. Recall that, for DTA, the effort for model building is independent of the population size, as it always updates its statistics using the best four individuals. Hence, the relative time for fitness evaluations and sampling grew much stronger, due to the exponential growth in the number of fitness evaluations for concatenated traps and NK landscapes.

4.3.3. Results for MAX-SAT

For the MAX-SAT problem, we did not perform a scalability analysis, but determined for each of the 100 problem instances the optimal population size that allowed each algorithm to find the optimal solution. Fig. 7 shows the results for DTA, BOA, and RBM–EDA[10]. The left-hand side shows the cumulative number of fitness evaluations, the right-hand side the cumulative CPU time. Under the given computational limits, DTA, BOA and RBM–EDA were able to find the optimum solution for 63, 66, and 93 instances, respectively. DTA was able to solve some of the instances with fewer fitness evaluations than both BOA and RBM–EDA, but was quickly outperformed by the other algorithms (left-hand side of Fig. 7). On the particular set of instances, RBM–EDA needed approximately the same number of fitness evaluations to solve 93 instances as BOA needed to solve 66 instances. Looking at the CPU times (right-hand side of Fig. 7), we observe similar behavior. DTA, and, to a lesser extent, BOA, were able to solve some of the instances very quickly, but were outperformed by RBM–EDA on the long run considering all (solved) instances.

4.3.4. RBM–EDA with increased RBM capacity

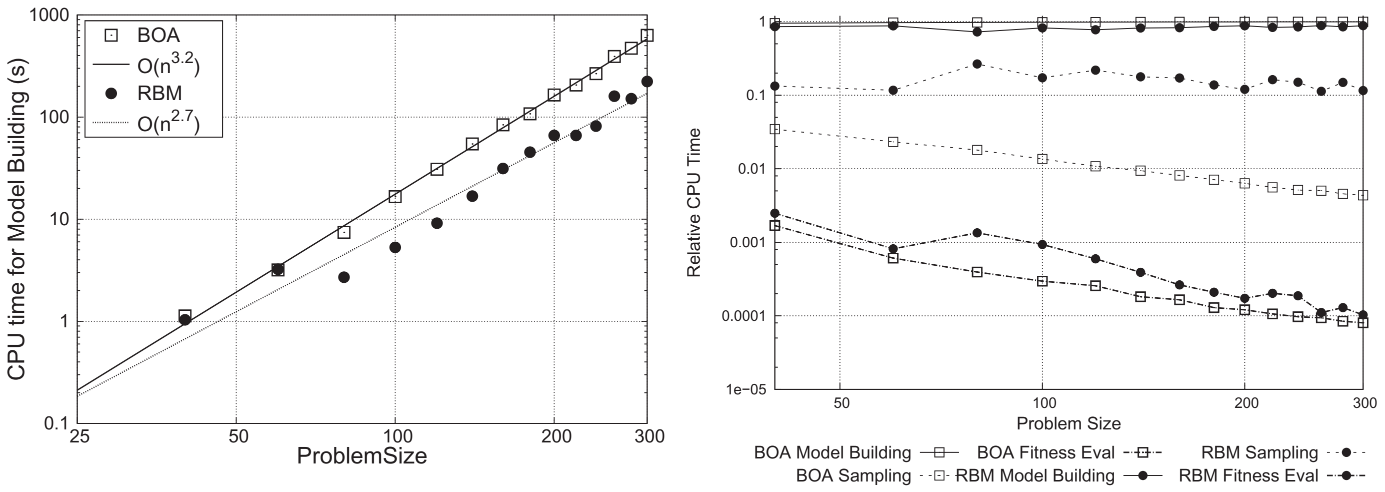
Recall that the number of hidden neurons m is a configurable parameter of the RBM. The larger m, the higher the capacity of the RBM. It is well known that RBMs with more hidden neurons can yield a better approximation of complex probability distributions (see e.g. Hinton, 1986) but lead to higher computational effort. For all of the results reported above, we used $m = n/2$, i.e., the number of hidden neurons was half the problem size. Fig. 8 exemplarily shows the results for the scalability analysis of concatenated deceptive traps with $k = 6$ for RBM–EDA where $m = n$, i.e., the number of hidden neurons was doubled. The results show that RBM–EDA parameterized with a high-capacity RBM needed fewer fitness evaluations than the RBM–EDA with $m = n/2$ (c.f. Fig. 3(c)). That is, the model quality was better with an increased number of hidden neurons. For the 6-trap problem, the high-capacity RBM–EDA even needed fewer fitness evaluations than BOA, but the complexity remains higher. The improved model quality came at a cost: the right-hand side of Fig. 8 indicates that the overall computational effort also slightly increased in comparison to RBM–EDA

[7] We omitted the relative CPU times for selection in order to increase the readability of Fig. 5. By definition, they are proportional to the CPU time for fitness evaluations, and, in absolute numbers, negligible.

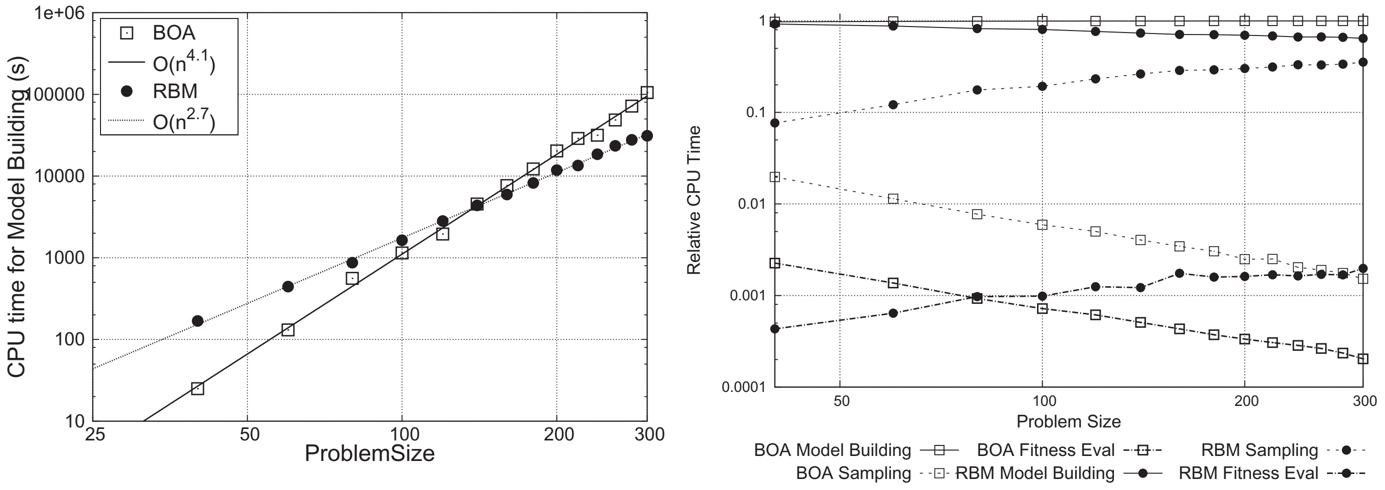
[8] The results for concatenated traps of size $k \in (4, 6)$ and NK landscapes of size $k \in (3, 4, 6)$ were qualitatively similar.

[9] Note that in this comparison, each algorithm used the population sizes from its own bisection run, i.e., the population size for BOA was usually smaller. If we used the same population sizes, we expect the time dominance of model building in BOA to be even greater.

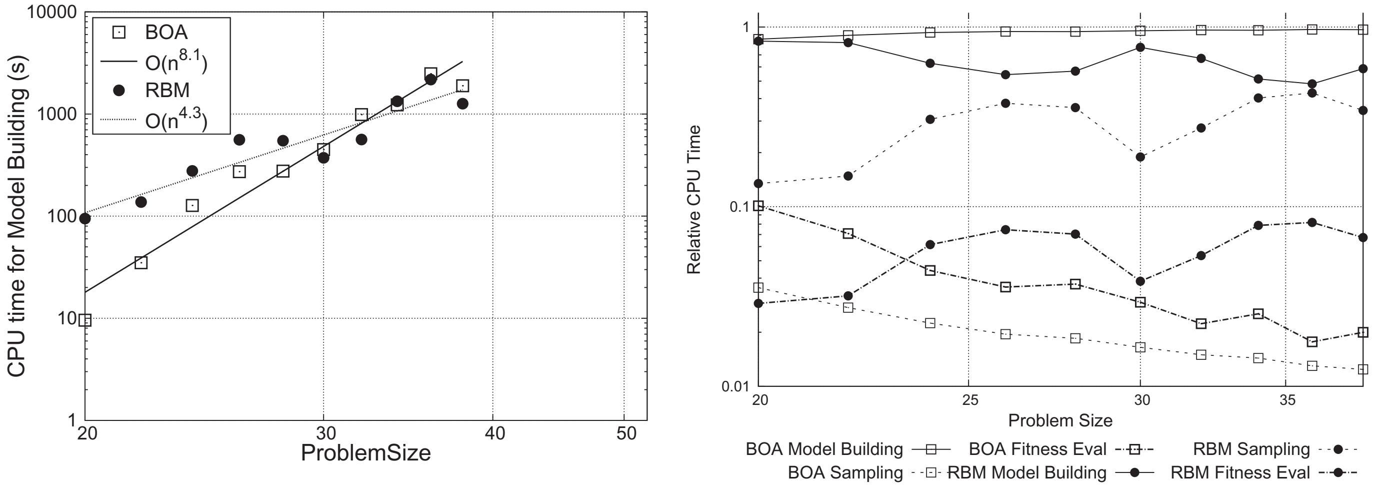
[10] The type of the graph is similar to the ones used in the yearly MAX-SAT competition, see http://maxsat.ia.udl.cat. Results for all instances are ordered in ascending order and cumulated.



(a) Onemax

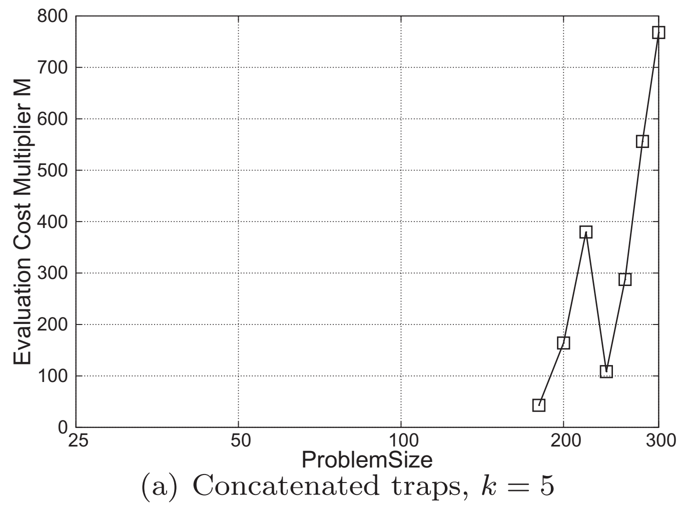


(b) Concatenated traps, $k = 5$



(c) NK landscapes, $k = 5$

Fig. 5. Details on CPU times for selected problems. Left: scalability of CPU time for model building. Right: relative CPU times for model building, sampling, and fitness evaluation.



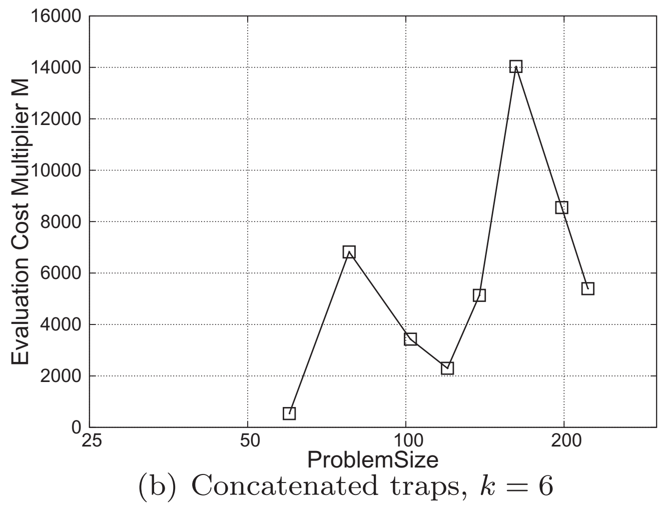
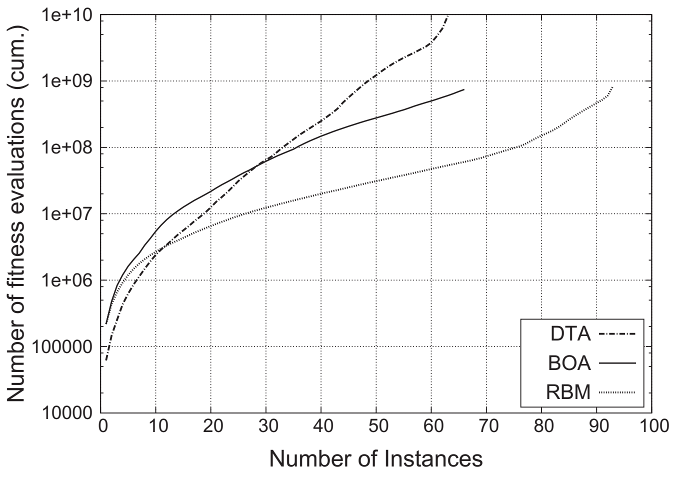


Fig. 6. Simulated multiplier M for fitness evaluation costs, such that BOA and RBM–EDA need the same total CPU time. Plots only show points where $M > 0$. Lines connecting the points are for visualization purposes only.



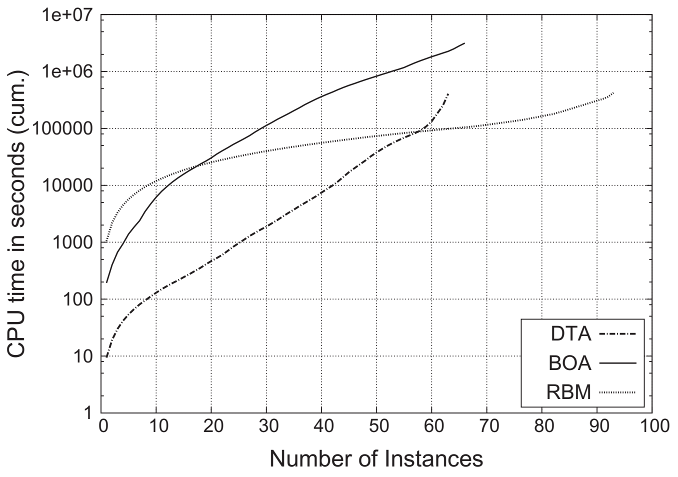
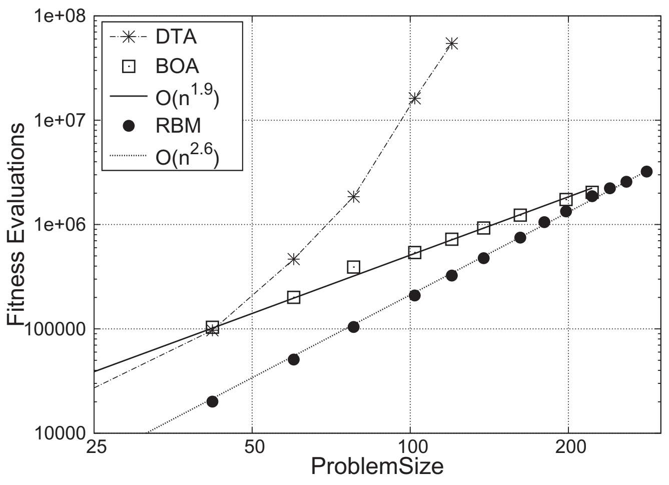


Fig. 7. Cumulative number of evaluations (left) and CPU time (right) over the number of problem instances for the MAX-SAT problem.



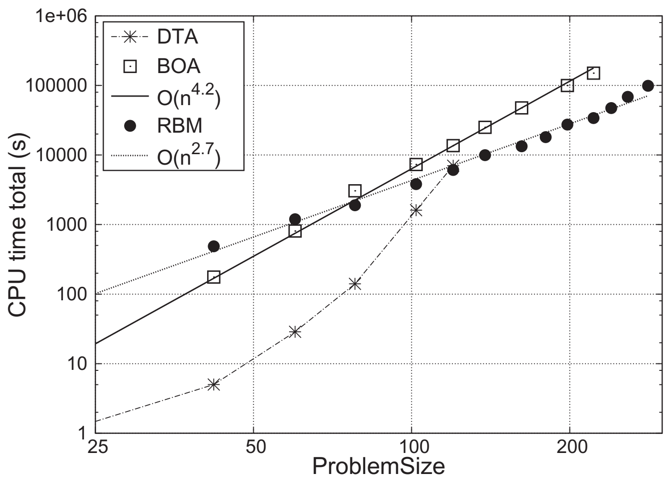


Fig. 8. Number of fitness evaluations (left) and CPU time (right) over problem size l for deceptive traps with $k = 6$. RBM–EDA with $m = n$, i.e., the number of hidden neurons m is equal to the problem size n (c.f. Fig. 3(c) where $m = n/2$). Lines connecting the points for DTA are for visualization purposes only.

with $m = n/2$. The results for the high-capacity RBM–EDA for onemax, concatenated traps of sizes $k \in (3, 4, 5)$, and NK landscapes were qualitatively similar, although less pronounced in the case of NK landscapes. For MAX-SAT, the high-capacity RBM–EDA was able to solve 98 of the 100 instances. It required 37 percent less fitness evaluations than the original version needed for solving 93 instances. At the same time, it required 50 percent more computational time.

4.3.5. *Discussion of results*

Due to its restricted model, DTA only worked well on simple problems, or smaller instances of difficult problems. For large and difficult problems (in particular for MAX-SAT), RBM–EDA's performance was higher than the state-of-the-art BOA although RBM–EDA usually needed a higher number of fitness evaluations to solve a problem. Even more, the scalability analysis for concatenated traps and NK landscapes showed that the computational effort in terms of fitness evaluations grew faster with the problem size n. The higher number of fitness evaluations used by RBM–EDA indicates that the statistical model created during training in RBM–EDA was less accurate than the statistical model learned in BOA. However, from a computational perspective, building this more accurate model in BOA was often much more expensive than learning a more inaccurate model in RBM–EDA combined with a higher number of sampling steps to compensate the lower model quality. Furthermore, the time necessary for building the model increased slower with increasing problem size than for BOA. Thus, the lower time effort necessary for learning a less accurate statistical model in RBM–EDA overcompensated for the higher number of fitness evaluations that were necessary to find optimal solutions. As a result, the CPU times for RBM–EDA were not only lower than BOA, they also increased more slowly with growing problem sizes. This effect was especially strong for difficult and large problems.

We hypothesize that the lower complexity of model building is due to the distributed representation of the RBM (see Section 3.1). The training algorithm (contrastive divergence) operates on this distributed representation. When CD modifies a weight between a single pair of visible and hidden neurons, an exponential number of configurations of the hidden neurons is affected. Hence, in this way, CD may be able to search the parameter space more efficiently than BOA's greedy search, which only consider single pairwise dependencies between variables at a time.

An encouraging argument for the use of RBM–EDA is its ability to scale well on parallel hardware. For example, Probst, Rothlauf, and Grahl (2014) report a speedup of 200 × and more when RBM–EDA is implemented on a modern Graphics Processing Unit (GPU), compared to optimized CPU code. The RBM's underlying architecture makes a parallel implementation very easy and the algorithm scales very efficiently even on massively parallel systems, which matches the current trend in hardware towards many cores and GPU usage (see e.g. Brodtkorb, Hagen, & Sætra, 2013). Parallelizing other multivariate EDAs such as BOA is possible, but much more difficult and the resulting speedups are often only single- or double-digit, even on GPUs (e.g. Munawar, Wahib, Munetomo, and Akama, 2009; Očenášek, & Schwarz, 2000 – for an overview of parallel EDAs, see Mendiburu, Lozano, & Miguel-Alonso, 2005; Mendiburu, Miguel-Alonso, & Lozano, 2010 or Probst et al., 2014).

When using RBM–EDA, it is possible to explicitly trade off model quality (i.e., fitness evaluations) and computational time. By using a larger number of hidden neurons in the RBM, the model quality improved significantly. This, in turn, reduced the number of fitness evaluations required by RBM–EDA, at the cost of slightly increased CPU times.

In summary, we can give the following recommendations for the selection of black-box algorithms: if the problem at hand is relatively easy, we suggest using approaches with restricted models like DTA. Such methods can solve easy problems pretty fast. With more difficult problems, or if the performance of approaches using non-multivariate models is low, we recommend using multivariate models like those used in BOA or RBM–EDA. Such algorithms use models that are able to capture more complex dependencies between decision variables. However, the higher performance comes with higher effort for model building. When using BOA, the computational effort for model building can be prohibitively high if the problems become either more difficult, or larger. In this case, we recommend switching to RBM–EDA. Although the RBM-based model used in RBM–EDA is less powerful and needs more fitness evaluations, the overall computational effort is lower due to the lower effort for model building.

In case there is parallel hardware available, we strongly recommend using a parallelized version of RBM–EDA, with a large number of hidden neurons in the RBM. Parallelizing the fitness evaluations is possible in any standard EDA. When using RBM–EDA, learning and sampling the model can be parallelized easily and efficiently as well. This dramatically reduces the effective times required, and, consequently, leads to large speedups on parallel hardware.

Finally, if the effort for calculating the fitness of solutions is very high (e.g. when running expensive simulations), the time for model building has a only a low impact on overall running time. In this case, we recommend to also try out a serial or parallel implementation of BOA, as BOA was often able to build a more accurate model, which allows for a lower number of fitness evaluations.

5. Summary and conclusions

We carried out an in-depth experimental analysis of using a Restricted Boltzmann Machine within an Estimation of Distribution Algorithm for combinatorial optimization. We tested RBM–EDA on standard binary benchmark problems: onemax, concatenated deceptive traps and NK landscapes of different sizes, as well as a set of MAX-SAT instances. We carried out a scalability analysis for the number of fitness evaluations and the computation times required to solve the problems to optimality. We compared our results to those obtained from the Bayesian Optimization Algorithm, a multivariate state-of-the-art method, and the Dependency Tree Algorithm, an algorithm with assumes a tree-like dependency structure.

Our experimental results suggest that RBM–EDA is competitive with the multivariate BOA, and superior to DTA on larger problem instances of the non-trivial problems. In the scalability analysis, we observed that, in many cases, it was less efficient than BOA in terms of fitness evaluations, both in absolute numbers and in complexity. However, the estimated complexity of probabilistic model building in RBM–EDA was lower than in BOA. Especially for large problems, RBM–EDA was able to build probabilistic models much faster than BOA. This led to lower total run times for RBM–EDA than for BOA, especially if the problem instances were large and difficult. RBM–EDA performed best on the tested MAX-SAT instances, both in terms of fitness evaluations and CPU time. RBM–EDA has the nice property that the power of the used model can be controlled by modifying the number of hidden neurons of the RBM. Increasing the number of hidden neurons yields a more accurate model, leading to a lower number of fitness evaluations. However, the effort for building the model increases, leading to slightly higher overall computational effort.

In sum, RBM–EDA is a powerful black-box optimization method, especially if problems are large and difficult, and the computational effort for fitness evaluations is low. This highlights the potential of using generative neural networks for combinatorial optimization.

A major advantage of using neural networks such as RBMs in EDAs is that – due to the structure of RBMs – they can be parallelized much easier, and more efficiently, in comparison to other types of EDAs. A parallel version of RBM–EDA on a Graphics Processing Unit yields massive speedups by factors of 200 and more, compared to optimized CPU code. The RBM's architecture makes it suitable for deployment on massively parallel hardware. This is a particularly useful feature, given current hardware trends.

There are multiple directions for further research. We demonstrated that RBMs are useful in EDAs and outperform BOA on difficult problems. However, carefully adjusting the RBM's parameters (such as the number of hidden neurons, the learning rate, or the number of CD sampling steps), or including recent optimization techniques (e.g. different regularizers or momentum variants) could improve the RBM's model quality, possibly leading to further performance improvements. The same could be true if the simple control scheme regulating the RBM's parameters during training was to be replaced by a more sophisticated method. It might also be beneficial to stack RBMs on multiple layers and to use such deep systems, e.g. Deep Belief Nets (DBN, Hinton et al., 2006) or Deep Boltzmann Machines (DBM, Salakhutdinov & Hinton, 2009), for solving hierarchical problems. However, preliminary results on DBMs shows using the more complex model comes at a very high computational cost, which is not necessarily justified by the solution quality.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions on earlier versions of this paper.

References

Abdollahzadeh, A., Reynolds, A., Christie, M., Corne, D. W., Davies, B. J., & Williams, G. J. (2012). Bayesian optimization algorithm applied to uncertainty quantification. *SPE Journal, 17*(03), 865–873.
Ackley, D. H. (1987). *A connectionist machine for genetic hill climbing*. Norwell, MA, USA: Kluwer Academic Publishers.
Bacardit, J., Stout, M., Hirst, J. D., Sastry, K., Llorà, X., & Krasnogor, N. (2007). Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. In *Proceedings of the ninth annual conference on genetic and evolutionary computation (GECCO 2007)* (pp. 346–353). ACM.
Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search-based function optimization and competitive learning. *Technical report cmu-cs-94-163*. Pittsburgh, PA.: Carnegie Mellon University.
Baluja, S., & Davies, S. (1997). 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)* (pp. 30–38). Morgan Kaufmann.
Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research, 3*, 1137–1155.
Bishop, C. M. (2006). Pattern recognition and machine learning. In *Information science and statistics*. Springer.
Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, 35*(3), 268–308.
Brodtkorb, A. R., Hagen, T. R., & Sætra, M. L. (2013). Graphics processing unit (GPU) programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing, 73*(1), 4–13.
Ceberio, J., Irurozki, E., Mendiburu, A., & Lozano, J. A. (2014). A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Transactions on Evolutionary Computation, 18*(2), 286–300.
Chen, C.-H., & Chen, Y.-P. (2007). Real-coded ECGA for economic dispatch. In *Proceedings of the ninth annual conference on genetic and evolutionary computation (GECCO 2007)* (pp. 1920–1927). New York, NY, USA: ACM.
Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, 14*(3), 462–467.
Competition, M.-S. (2014). Benchmark problems for the maxiumum satisfiability problem (random, unweighted). We used all 100 instances in folder /*ms_random*/*max*3*sat*/ of the compressed container. http://www.maxsat.udl.cat/14/benchmarks/ms_random.tgz.
Dahl, G. E., Ranzato, M. A., Mohamed, A., & Hinton, G. (2010). Phone recognition with the mean-covariance restricted boltzmann machine. In *Proceedings of the advances in neural information processing (NIPS 2010)* (p. 23).
Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing, 20*(1), 30–42.
Deb, K. (2014). Multi-objective optimization. In E. K. Burke, & G. Kendall (Eds.), *Search methodologies* (pp. 403–449). US: Springer.
Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In D. L. Whitley (Ed.), *Foundations of genetic algorithms 2* (pp. 93–108). Morgan Kaufmann.
(1997). Satisfiability problem: Theory and applications: DIMACS workshop, March 11–13, 1996. In Du, D., Gu, J., & Pardalos, P. (Eds.). In Center for discrete mathematics and theoretical computer science (DIMACS)*: Vol. 35*. New Brunswick, NY: American Mathematical Society.
Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI, 6*(6), 721–741.
Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. MA, USA: Addison-Wesley Professional, Reading.
Harik, G. R., Lobo, F. G., & Sastry, K. (2006). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In M. Pelikan, K. Sastry, & E. Cantú-Paz (Eds.), *Scalable optimization via probabilistic modeling*. In Studies in computational intelligence*: Vol. 33* (pp. 39–61). Berlin, Heidelberg: Springer.
Hauschild, M., & Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation, 1*(3), 111–128.
Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation, 14*(8), 1771–1800.
Hinton, G. E. (2012). 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, M. Y. Vardi, G. Weikum, G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade*. In Lecture notes in computer science*: Vol. 7700* (pp. 599–619). Berlin, Heidelberg: Springer Berlin Heidelberg.
Hinton, G. E. (1986). Distributed representations. In J. L. McClelland, & D. E. Rumelhart (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 1* (pp. 77–109). Cambridge, MA: MIT Press. Foundations.
Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7), 1527–1554.
Holland, J. H. (2001). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence. Complex adaptive systems (6th). Cambridge: MIT Press.
Kauffman, S. A., & Weinberger, E. D. (1989). The NK model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology, 141*(2), 211–245.
Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedingd of the IEEE international conference on neural networks: Vol. 4* (pp. 1942–1948).
Kollat, J., Reed, P., & Kasprzyk, J. (2008). A new epsilon-dominance hierarchical Bayesian optimization algorithm for large multiobjective monitoring network design problems. *Advances in Water Resources, 31*(5), 828–845.
Larrañaga, P., Etxeberria, R., Lozano, J. A., & Peña, J. M. (1999). Optimization by learning and simulation of Bayesian and gaussian networks. *Technical report ehu-kzaa-ik-4/99*. Intelligent Systems Group, Dept. of Computer Science and Artificial Intelligence, University of the Basque Country.
Larrañaga, P., Karshenas, H., Bielza, C., & Santana, R. (2012). A review on probabilistic graphical models in evolutionary computation. *Journal of Heuristics, 18*(5), 795–819.
Larrañaga, P., & Lozano, J. A. (2002). Estimation of distribution algorithms: A new tool for evolutionary computation. Genetic algorithms and evolutionary computation: Vol. 2. US, Boston, MA, USA: Springer.
Liang, X., Chen, H., & Lozano, J. (2015). A Boltzmann-based estimation of distribution algorithm for a general resource scheduling model. *IEEE Transactions on Evolutionary Computation, 19*(6), 793–806.
(2006). Towards a new evolutionary computation: Advances on estimation of distribution algorithms. In Lozano, J. A., Larrañaga, P., Inza, I., & Bengoetxea, E. (Eds.). In Studies in fuzziness and soft computing*: Vol. 192*. Berlin, Heidelberg: Springer.
Martí, L., García, J., Berlanga, A., & Molina, J. M. (2008). Introducing MONEDA: Scalable multiobjective optimization with a neural estimation of distribution algorithm. In *Proceedings of the tenth annual conference on genetic and evolutionary computation (GECCO 2008)* (pp. 689–696). ACM.
Martí, L., García, J., Berlanga, A., & Molina, J. M. (2009). Solving complex high-dimensional problems with the multi-objective neural estimation of distribution algorithm. In *Proceedings of the eleventh annual conference on genetic and evolutionary computation (GECCO 2009)* (pp. 619–626). ACM.
Mendiburu, A., Lozano, J. A., & Miguel-Alonso, J. (2005). Parallel implementation of EDAs based on probabilistic graphical models. *IEEE Transactions on Evolutionary Computation, 9*(4), 406–423.
Mendiburu, A., Miguel-Alonso, J., & Lozano, J. A. (2010). A review on parallel estimation of distribution algorithms. In *Parallel and distributed computational intelligence* (pp. 143–163). Springer.
Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems, 9*, 193–212.
Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*(11), 1097–1100.
Mühlenbein, H., & Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation, 7*(4), 353–376.
Mühlenbein, H., & Mahnig, T. (2002a). Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal of Approximate Reasoning, 31*(3), 157–192.

Mühlenbein, H., & Mahnig, T. (2002b). Mathematical analysis of evolutionary algorithms. In *Essays and surveys in metaheuristics* (pp. 525–556). Springer.

Mühlenbein, H., Mahnig, T., & Ais, F. (2003). Evolutionary algorithms and the boltzmann distribution. In *Foundations of genetic algorithms 7*. San Francisco: Morgan Kaufmann Publishers.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions i. binary parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature (PPSN IV)*. In Lecture notes in computer science*: Vol. 1141* (pp. 178–187). Berlin, Heidelberg: Springer.

Munawar, A., Wahib, M., Munetomo, M., & Akama, K. (2009). 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)* (pp. 457–462). IEEE.

Očenášek, J., & Schwarz, J. (2000). The parallel Bayesian optimization algorithm. In P. Sinčák, J. Vaščák, V. Kvasnička, & R. Mesiar (Eds.), *The state of the art in computational intelligence*. In Advances in soft computing*: Vol. 5* (pp. 61–67). Heidelberg: Physica-Verlag.

Peña, J. M., Lozano, J. A., & Larranaga, P. (2004). Unsupervised learning of Bayesian networks via estimation of distribution algorithms: an application to gene expression data clustering. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 12*(12), 63–82.

Pelikan, M. (2005). Bayesian optimization algorithm. In *Hierarchical Bayesian optimization algorithm*. In Studies in fuzziness and soft computing*: Vol. 170* (pp. 31–48). Springer.

Pelikan, M., & Goldberg, D. E. (2003). Hierarchical BOA solves ising spin glasses and MAXSAT. In *Proceedings of the genetic and evolutionary computation conference (GECCO 2003)*. In Lecture notes in computer science*: Vol. 2724* (pp. 1271–1282). Berlin, Heidelberg: Springer.

Pelikan, M., Goldberg, D. E., & Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, & R. E. Smith (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO 1999): Vol. 1* (pp. 525–532). San Francisco, CA, USA: Morgan Kaufmann.

Pelikan, M., Goldberg, D. E., & Lobo, F. (1999). A survey of optimization by building and using probabilistic models. *Illigal report no. 99018*. Urbana, IL.: University of Illinois at Urbana-Champaign.

Pelikan, M., & Hartmann, A. K. (2006). Searching for ground states of using spin glasses with hierarchical boa and cluster exact approximation. In *Scalable optimization via probabilistic modeling* (pp. 333–349). Springer.

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, & P. K. Chawdhry (Eds.), *Advances in soft computing* (pp. 521–535). London: Springer.

Pelikan, M. (2008). Analysis of estimation of distribution algorithms and genetic algorithms on NK landscapes. In C. Ryan, & M. Keijzer (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO 2008): vol. 10* (pp. 1033–1040). New York, NY, USA: ACM. We used the first 25 instances in each subfolder of $/nk-instances/$ of the compressed container. http://medal-lab.org/files/nk-instances.tar.gz

Pelikan, M., Sastry, K., & Goldberg, D. E. (2006). Multiobjective estimation of distribution algorithms. In M. Pelikan, K. Sastry, & E. Cantú-Paz (Eds.), *Scalable optimization via probabilistic modeling*. In Studies in computational intelligence*: Vol. 33* (pp. 223–248). Springer Berlin Heidelberg.

Petrovski, A., Shakya, S., & McCall, J. (2006). Optimising cancer chemotherapy using an estimation of distribution algorithm and genetic algorithms. In *Proceedings of the eighth annual conference on genetic and evolutionary computation (GECCO 2006)* (pp. 413–418). ACM.

Probst, M., Rothlauf, F., & Grahl, J. (2014). An implicitly parallel EDA based on restricted Boltzmann machines. *Proceedings of the genetic and evolutionary computation conference (GECCO 2014)* (pp. 1055–1062). New York, NY, USA: ACM.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks, 12*(1), 145–151.

Rothlauf, F. (2011). *Design of modern heuristics*. Heidelberg: Springer.

Salakhutdinov, R., & Hinton, G. E. (2009). Deep Boltzmann machines. In D. van Dyk, & M. Welling (Eds.), *Proceedings of the international conference on artificial intelligence and statistics (AISTATS 2009): vol. 5* (pp. 448–455).

Salakhutdinov, R., Mnih, A., & Hinton, G. E. (2007). Restricted Boltzmann machines for collaborative filtering. In Z. Ghahramani (Ed.), *Proceedings of the twenty–fourth international conference on machine learning (ICML 2004)* (pp. 791–798).

Santana, R. (2005). Estimation of distribution algorithms with kikuchi approximations. *Evolutionary Computation, 13*(1), 67–97.

Santana, R., Larrañaga, P., & Lozano, J. A. (2007). Side chain placement using estimation of distribution algorithms. *Artificial Intelligence in Medicine, 39*(1), 49–63.

Santana, R., Larrañaga, P., & Lozano, J. A. (2008). Protein folding in simplified models with estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation, 12*(4), 418–438.

Santarelli, S., Yu, T.-L., Goldberg, D. E., Altshuler, E., O'Donnell, T., Southall, H., & Mailloux, R. (2006). Military antenna design using simple and competent genetic algorithms. *Mathematical and Computer Modelling, 43*(9), 990–1022.

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

Shah, R., & Reed, P. (2011). Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems. *European Journal of Operational Research, 211*(3), 466–479.

Shakya, S., & McCall, J. (2007). Optimization by estimation of distribution with DEUM framework based on Markov random fields. *International Journal of Automation and Computing, 4*(3), 262–272.

(2012). *Markov networks in evolutionary computation*. In Shakya, S., & Santana, R. (Eds.). Springer.

Shim, V., Tan, K., & Chia, J. (2010). Probabilistic based evolutionary optimizers in bi-objective travelling salesman problem. In K. Deb, A. Bhattacharya, N. Chakraborti, P. Chakroborty, S. Das, J. Dutta, S. K. Gupta, A. Jain, V. Aggarwal, J. Branke, S. J. Louis, & K. C. Tan (Eds.), *Simulated evolution and learning*. In Lecture notes in computer science*: Vol. 6457* (pp. 588–592). Springer.

Shim, V., & Tan, K. (2012). Probabilistic graphical approaches for learning, modeling, and sampling in evolutionary multi-objective optimization. In J. Liu, C. Alippi, B. Bouchon-Meunier, G. Greenwood, & H. Abbass (Eds.), *Advances in Computational Intelligence*. In Lecture notes in computer science*: Vol. 7311* (pp. 122–144). Springer.

Shim, V. A., Tan, K. C., Chia, J. Y., & Al Mamun, A. (2013). Multi-objective optimization with estimation of distribution algorithm in a noisy environment. *Evolutionary Computation, 21*(1), 149–177.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, & P. R. Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition*. In Computational models of cognition and perception*: Vol. 1* (pp. 194–281). Cambridge, MA, USA: MIT Press.

Storn, R., & Price, K. (1997). Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization, 11*(4), 341–359.

Tang, H., Shim, V. A., Tan, K. C., & Chia, J. Y. (2010). Restricted Boltzmann machine based algorithm for multi-objective optimization. In *Proceedings of the IEEE congress on evolutionary computation (CEC 2010)* (pp. 1–8).

Zhang, B.-T., Shin, S.-Y., Schoenauer, M., Deb, K., Rudolph, G., Yao, X., … Schwefel, H.-P. (2000). Bayesian evolutionary optimization using helmholtz machines. In *Parallel problem solving from nature (PPSN VI)*. In Lecture notes in computer science*: Vol. 1917* (pp. 827–836). Berlin, Heidelberg: Springer.