Accepted for publication in *Evolutionary Computation* published by MIT Press. The final published version is available at: https://doi.org/10.1162/evco_a_00257

Learning and Searching Pseudo-Boolean Surrogate Functions from Small Samples

Dr. Kevin Swingler kms@cs.stir.ac.uk
Computing Science and Mathematics, University of Stirling, Stirling, FK9 4LA, Scotland

Abstract

When searching for input configurations that optimise the output of a system, it can be useful to build a statistical model of the system being optimised. This is done in approaches such as surrogate model-based optimisation, estimation of distribution algorithms and linkage learning algorithms. This paper presents a method for modelling pseudo-Boolean fitness functions using Walsh bases and an algorithm designed to discover the non-zero coefficients while attempting to minimise the number of fitness function evaluations required. The resulting models reveal linkage structure that can be used to guide a search of the model efficiently. It presents experimental results solving benchmark problems in fewer fitness function evaluations than those reported in the literature for other search methods such as EDAs and linkage learners.

Keywords

Fitness Function Modelling, Estimation of Distribution Algorithms, Pseudo-Boolean Functions, Linkage Learning, Walsh Decomposition, Mixed Order Hyper Networks, Statistical Machine Learning.

1 Introduction

Black box optimisation is the challenge of choosing a solution to a given problem based purely on the relative scores of different candidate solutions. Search methods involve evaluating candidate solutions, either one at a time or in groups, and using the results of those evaluations to attempt to generate new, better solutions by a mixture of exploiting what can be learned from good solutions that have gone before and exploring new possible solutions. Such processes are known as metaheuristics and include local search algorithms like iterated local search (Lourenço et al., 2003) and simulated annealing (S. Kirkpatrick, 1983); as well as population based methods such as genetic algorithms (Goldberg, 1989b) and particle swarm optimisation (Kennedy, 2001). The function that takes a candidate solution as input and produces a score (or cost) as output is known as the fitness function and a common goal when performing such searches is to minimise the number of fitness function evaluations required to find a suitable solution.

The fitness function is generally a computer simulation of the real world process to be optimised. In cases where this simulation takes a long time to run or is expensive to evaluate for other reasons, it is particularly important to be able to find a good solution in as few evaluations as possible. One approach to minimising the number of times an expensive fitness function needs to be evaluated during a search involves sampling $(input, output)$ pairs from the function and using the resulting data set to build a predictive model. Some functions can be modelled in fewer evaluations than are needed to perform a search. When the resulting model is faster to evaluate than the full simulation, a longer search can take place in a reduced time span.

Speeding up the evaluation of expensive fitness functions is the most common reason for using a fitness function model, but it is not the only one. Once the model is built, it still needs to be searched and some models are easier to search than others. A model's representation can

expose facts about the fitness function that can be exploited during a search. The most useful of these are facts about the way in which input variables interact in their effect on the function output. This is known as *linkage* and knowing the linkage structure of a function can provide valuable insights for metaheuristic search algorithms. For example, it can guide crossover in a genetic algorithm to avoid splitting blocks of values that work together. For additively decomposable problems, it can reveal subsets of variables that can be optimised independently from the others. Local search algorithms rely on evaluations of the fitness function within a small neighbourhood of the current search point and knowing the linkage structure can allow incremental partial evaluations to be made very efficiently.

Much of the work on surrogate models has concerned continuous optimisation problems. This paper will focus on combinatorial optimisation problems, which can be encoded as pseudo-Boolean inputs to a fitness function. The main contribution of the paper is to demonstrate an approach that explicitly aims to minimise the number of fitness function evaluations required to build a surrogate model. This is of particular importance when fitness evaluations are expensive. Section 2 describes some existing approaches to fitness function modelling and motivates the current work. Section 3 describes Mixed Order Hyper Networks and the qualities they have that make them suitable as surrogate models. Some experiments are documented in Section 4 and Section 5 suggests further work that needs to be done.

2 Existing Work

Approaches to improving optimisation using statistical modelling can be roughly split into three different types. **Surrogate model-based optimisation** (SMBO) attempts to reproduce the fitness function to speed up the search as described above. The models may fully replace the original fitness function or be used in conjunction with it to guide the search. **Estimation of Distribution Algorithms** (EDAs) build models of only the more promising areas of the search space and are used in an evolutionary paradigm where only the better examples from each generation are used to build the model, which is then sampled to produce further candidate solutions. **Linkage learning algorithms** (LLAs) use statistical learning of the relationships among input variables as a way to inform other search techniques such as genetic algorithms. This may involve building statistical models or make use of a linkage analysis such as that provided by performing a Walsh decomposition of the fitness function.

2.1 Surrogate Model-Based Optimisation

Surrogate model-based optimisation (SMBO) attempts to replicate a fitness function in the form of a predictive model as part of a search process. Models are built by sampling and evaluating candidate solutions to generate a training data set. Once built, the model must be searched to find an optimal input pattern. Learning the model has been addressed using a variety of methods. Neural networks are popular for surrogate models (Holeňa et al., 2010) and have the advantage of allowing partial derivatives of the output to be calculated at each input variable, which is very useful for gradient based searches. The representation of multilayer perceptrons makes it difficult to explicitly fix or infer linkage structure, however, so that information is not available to guide linkage based search. Radial basis functions have also been proposed as surrogate models (Gutmann, 2001), as have Gaussian processes (Frean and Boyle, 2008), and the closely related Kriging technique (Willmes et al., 2003).

Fitness functions with high dimensional inputs have been addressed using dimensionality reduction techniques such as proper orthogonal decomposition (also known as principal components analysis). This process transforms the training data into a new set of orthogonal variables ordered by the amount of variance they contain. Lower variance variables can be discarded to simplify a model and attempt to solve the optimisation problem in a new, smaller search space.

Such methods are popular in the optimisation of aerodynamic design (Iuliano and Quagliarella, 2013).

There has been little work published on pseudo-Boolean fitness function models, possibly because many of the benchmark problems are not expensive to evaluate. However, other benefits of modelling such as linkage discovery and fast partial function evaluation mean that modelling pseudo-Boolean fitness functions is still worthwhile. Recent examples include the use of radial basis functions to solve combinatorial problems (Kim et al., 2014) and a low order Walsh basis to model pseudo-Boolean problems (Verel et al., 2018), as described in Section 2.3. Reviews of the use of surrogate fitness models can be found in (Jin, 2005) and (Jin, 2011).

2.2 Estimation of Distribution Algorithms

Estimation of distribution algorithms (EDAs) use a statistical model of the distribution of promising regions of search space, combining model bias with sample bias in an iterative process that alternates between model building and the use of the model to generate a new population of candidate solutions. In some cases, the models can be simpler than would be needed by a surrogate model as only a subspace of the problem is modelled at each generation. The cost of this is that more samples from the original fitness function are required to allow the selection of better solutions to take place. For a comprehensive introduction to EDAs, see (Pelikan et al., 2015).

Many different approaches to modelling the distribution in an EDA have been proposed, covering the full range of levels of model bias. In contrast to the literature on SMBO, much of the EDA literature addresses pseudo-Boolean problems. Many of the EDA models that have been proposed use some form of graphical model to represent the probability distribution of promising candidate solutions. The simplest were first order models like the Univariate Marginal Distribution Algorithm (UMDA) (Mühlenbein, 1997) and Population Based Incremental Learning (PBIL) (Baluja and Caruana, 1995). Algorithms like the Bivariate Marginal Distribution Algorithm (BMDA) (Pelikan and Mühlenbein, 1999) have an explicit level of model bias, modelling only bivariate interactions, for example. Markov Random Fields can be given any level of model bias and have been used in EDAs such as DEUM (Shakya et al., 2009) and MARLEDA (Alden and Miikkulainen, 2013). Bayesian networks have also been used to model distributions in EDAs, for example the Bayesian Optimisation Algorithm (BOA) (Pelikan et al., 2000) and a hierarchical version (hBOA) (Pelikan, 2005) have been proposed. Both deep Boltzmann machines (Probst and Rothlauf, 2015) and restricted Boltzmann machines (Probst et al., 2014) have also been used as EDAs.

2.3 Linkage Learning

If the inputs to a fitness function do not interact in their effect on the output score, optimisation simply involves finding the best value for each variable independently. For all interesting problems, however, there are interactions among the inputs and their nature determines the difficulty with which an optimal solution may be found. Consequently, much effort has been expended in studying such interactions, which are often referred to as the linkage problem. It has been widely studied in terms of the proximity of genes in a chromosome in genetic algorithms (Harik and Goldberg, 1997) and many algorithms have been proposed that attempt to direct crossover in GAs based on linkage learning. For example, the Linkage Tree Genetic Algorithm (LTGA) (Thierens, 2010) mixes model building and linkage discovery by building a linkage tree that is used to generate new solutions in which groups of linked variables are undisturbed from the previous generation. More recently, the term linkage has come to refer to the study of the interactions among input variables across all metaheuristic approaches (Chen, 2008). Methods for detecting linkage among variables include simple pairwise tests such as probing (Heckendorn

and Wright, 2004) or entropy based tests used in EDAs such as DEUM (Shakya et al., 2012) and the ECGA (Harik et al., 2006).

The problem with testing pairs of variables for linkage is that it ignores higher order interactions. It is not generally possible to infer anything about higher order interactions from the presence or absence of pairwise interactions and evaluating groups of more than two variables incurs an exponentially increasing computational cost. (Coffin and Smith, 2008), for example point out that most EDA searches are greedy and start from a search for pairwise interactions that can fail to find higher order interactions that are not signalled by similar ones at lower orders. They suggest that researchers 'bite the bullet' and search for higher order linkages or employ a hybrid method involving both an EDA and linkage detection such as D^5 (Tsuji et al., 2004).

2.4 Walsh Decomposition

Another useful way of investigating linkage is to use a Walsh decomposition of the fitness function. Walsh functions (Walsh, 1923) form a basis for functions in $f : \{-1,1\}^p \rightarrow \mathbb{R}$ and have been widely used as a tool for assessing linkage, see (Goldberg, 1989a) and (Davidor, 1990) for some early work. A Walsh decomposition consists of a weighted sum of Walsh functions, which each acts on a unique subset of the input variables. Each Walsh function, ψ_j selects an associated subset of inputs and calculates the parity across those variables in the current input pattern being evaluated. This parity, in $\{-1,1\}$ is multiplied by the associated Walsh coefficient, ω_j and added to the sum. Any function with $\omega_j = 0$ has no effect on the sum and so indicates that the variables it selects do not interact. Discovering which Walsh functions have a non-zero coefficient is equivalent to detecting all the linkages in a fitness function. This is very useful, but a full decomposition requires every possible input pattern to be evaluated, which makes it interesting for analysing problems of few variables, but impractical as an optimisation tool in general so a method for discovering the zero valued Walsh coefficients in fewer fitness function evaluations is required. The Walsh decomposition of a function is represented as the sum:

$$f(X) = \sum_{j=0}^{2^n-1} \omega_j \psi_j(X) \tag{1}$$

where

$$\psi_j(X) = \oplus(X \wedge j_{bin}) \tag{2}$$

where $\oplus(X)$ is a parity count function that returns 1 if the number of values set to 1 in X is even and -1 otherwise and j_{bin} is the binary vector representation of the index j. The $X \wedge j_{bin}$ uses the binary vector representation of j to select the subset of variables in X operated on by $\psi_j(X)$.

Specific fitness functions have been analysed using a Walsh decomposition, for example (Heckendorn and Whitley, 1997) present a Walsh based analysis of NK landscapes. Walsh functions have recently been used as a basis for surrogate fitness functions (Verel et al., 2018) but without an attempt to address the question of finding the correct model bias, other than by using existing knowledge of the fitness function's linkage order. The number of coefficients to be estimated grows exponentially with linkage order and the required number of data samples grows linearly with the number of coefficients to be estimated so stricter control of that number is important when modelling expensive functions. (Verel et al., 2018) limit the number of coefficients they estimate by only modelling up to a low linkage order. The number of coefficients at order k in a model of p variables is $\binom{p}{k}$, so the number of fitness function evaluations required to build models with larger k grows quickly.

2.5 Motivation

A Walsh decomposition of a pseudo-Boolean function will reveal its linkage structure but for every unique coefficient you wish to calculate, you need a single unique fitness function evaluation. Many of the coefficients evaluate to zero, but data are still required for the decomposition, so discovering a sparse model over p variables still requires $\binom{p}{k}$ fitness evaluations for each linkage order k included in the model. For a full decomposition, these sum to 2^p. If the final sparse model has only w coefficients, then that model could be built from only w fitness evaluations if only the correct structure were known. The correct structure is not generally known (the point of doing the decomposition is to discover it!) so the challenge is to discover the correct structure from a reduced sample of fitness evaluations.

This paper presents Mixed Order Hyper Networks (MOHNS), which use a Walsh-like decomposition, a hypergraph representation and linear parameter estimation to model arbitrary fitness functions. An algorithm is presented that needs a limited number, n fitness function evaluations to search for non-zero coefficients over $m \gg n$ possible linkage combinations and that is often observed in experiments to be successful. For example, 5,000 fitness evaluations are used in one experiment to find the correct Walsh coefficients from among over 2 million possibilities. Searching the MOHN, once it is built, is made more efficient than searching a black box fitness function (regardless of how expensive that function is) by the linkage structure it reveals. Examples of local search and variable neighbourhood search that exploit the MOHN structure are presented.

3 Mixed Order Hyper Networks

Binary Mixed Order Hyper Networks (MOHNs) represent any function in $f : \{-1,1\}^p \rightarrow \mathbb{R}$ as a weighted sum of products of (possibly overlapping) subsets of the input variables. For example, any function with three inputs, $f(X_1,X_2,X_3)$ and a continuous valued output can be represented as a weighted sum of products from the set $\{c,(X_1),(X_2),(X_3),(X_1X_2),(X_1X_3),(X_2X_3),(X_1X_2X_3)\}$ where the c represents an offset constant. More generally, any function $f(X)$ with $X \in \{1,-1\}^p$ can be represented by a set of weights, $\mathbf{W}$ where each weight, W_j consists of a value, $\omega_j \in \mathbb{R}$ and a set of indices of the connected variables, $\mathbf{I}_j$. The pseudo-Boolean function is represented as

$$\hat{f}(X) = \sum_j \omega_j \prod_{i \in \mathbf{I}_j} X_i \tag{3}$$

Equation 3 is linear in the parameters, ω_j and represents a multivariate power series. A full description of the structure, training and use of MOHNs can be found in (Swingler, 2016a).

3.1 Learning the MOHN Coefficients

This section describes methods for estimating the coefficients of a fixed structure MOHN, in which the linkage structure defined in the weight index set $\mathbf{I}$ does not change during learning. The coefficients of a fixed structure MOHN can be learned from a sample of fitness function evaluations. With $X = X_1 \ldots X_p$ representing the p inputs to the fitness function, let $\mathbf{X}$ be the $n \times p$ matrix that represents n candidate solutions and let $\mathbf{Y} = y_1 \ldots y_n$ be the corresponding n fitness scores. The design matrix, $\mathbf{Z}$ used to estimate the coefficients is built by expanding each row in $\mathbf{X}$ by the products defined in $\mathbf{I}$. A single row, $Z = Z_1 \ldots Z_j$ is calculated with

$$Z_j = \prod_{i \in \mathbf{I}_j} X_i \tag{4}$$

The full design matrix, $\mathbf{Z}$ has n rows and j columns. To include a constant in the model,

a first column, Z_0 in the design matrix is used with all of its values equal to one. The ordinary least squares (OLS) coefficients can be estimated using

$$\omega = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{Y} \tag{5}$$

or, for a more numerically stable estimate, use the Moore-Penrose pseudo-inverse:

$$\omega = \mathbf{Z}^+\mathbf{Y} \tag{6}$$

where ω is the resulting vector of coefficients, one for each column in $\mathbf{Z}$. Alternatively, $\mathbf{Z}$ can be used as the inputs and $\mathbf{Y}$ as the target outputs to a gradient descent approach to estimation. Equation 3 is linear in its parameters, which means that the least squares cost function is convex and has a single global minimum.

Minimising the least squares cost gives an unbiased estimate of the MOHN coefficients. Regularisation can be introduced using the lasso (Tibshirani, 1996), which adds a term to the cost function designed to reduce the L_1 norm of the weights vector, ω. This shrinks the coefficient values and forces some of them to zero. Lasso is used in the algorithm for discovering the correct MOHN structure, described below. (Swingler, 2015) gives a full description of the MOHN learning rules.

3.1.1 MOHN Structure Discovery Algorithm

When the correct linkage structure is unknown, it must be discovered as part of the function learning process. Let the index set corresponding to the correct structure for a given fitness function be $\mathbf{I}^*$ and let $\mathbf{I}^* \subset \mathbf{I}$ mean that $\mathbf{I}$ contains all the weights in $\mathbf{I}^*$ plus some additional spurious connections. This section discusses a method for finding the correct linkage structure, $\mathbf{I}^*$, given samples from a fitness function. In the trivial case, we can choose a connection set $\mathbf{I}$ that is big enough to guarantee that $\mathbf{I}^* \subset \mathbf{I}$ and then build an unbiased model using least squares as described above. When the fitness samples are noise free, an unbiased estimate of the coefficients on the weights connected by $\mathbf{I}$ can be made from a sample of size $n = |\mathbf{I}|$. That is to say, one noise free sample per coefficient is required. In such cases, the spurious coefficients will all have estimated coefficients of zero. Removing those connections leaves the remaining correct structure, $\mathbf{I}^*$.

The problem is that evaluating the fitness function $|\mathbf{I}|$ times to train the model is expensive. Training a smaller model reduces the required number of fitness function evaluations but also reduces the chance of the model containing all the weights in $\mathbf{I}^*$. The solution is to iteratively add and remove coefficients from a model, retraining at each iteration with the same sample of data. This moves the expense away from evaluating the fitness model and onto building multiple models.

(Swingler, 2016b) presents a method for discovering the right structure (i.e., which weights to include) for a MOHN from a set of training examples. In the MOHN Structure Discovery Algorithm (MSDA), weights are added and removed in an iterative process designed to control the number of parameters in the model at any one time, which controls the number of data samples required. Weights are added by selecting from a distribution over the weight orders. The choice of this distribution is guided by assumptions about the linkage order of the function being modelled and the number of inputs to the model. For small models where no preference for linkage order exists, a discrete uniform distribution could be used but in all of the work reported here, a discrete Laplace distribution was used with a mode set to be the lowest order at which there are still untried weights. The Laplace distribution is defined as

$$f_L(x) = \frac{1}{2\lambda}e^{-\frac{|c-x|}{\lambda}} \tag{7}$$

where λ controls the width of the distribution and c defines the mode. The mode, c increases as the search progresses, so that the more time there is available to perform the search, the higher the order of weights that are considered becomes. Domain knowledge, if available, can be used to bias this distribution further. The distribution is also updated in response to the orders of the weights that are kept or discarded from the model at each iteration. Orders at which previously added weights have been kept in the model are more likely to be sampled than those from which weights have been deleted. This attempts to exploit regularities in the linkage order of the function being learned.

Rather than using least squares to estimate the coefficients during structure discovery, lasso is used. In a model that contains some of the correct connections and some spurious connections, but which also lacks some required connections, the lasso regularisation can be used to force the coefficients on the spurious connections to zero. These weights are then removed and replaced with new ones selected as described above. Once removed, the same weight cannot be added again for a number of iterations. This avoids the same weight being added and removed repeatedly. Once the new weights have been added, the coefficients are re-estimated. The coordinate descent search used to estimate the coefficients with the lasso cost function can start with remaining coefficients already set to their current values, which speeds up the error minimisation process.

Algorithm 1 presents a simplified version of the MOHN Structure Discovery Algorithm. For a full description, see (Swingler, 2016b).

Algorithm 1 MOHN Structure Discovery Algorithm

Let $\mathbf{W} \leftarrow \emptyset$ be the set of weights in the current model
Let $\mathbf{H}$ be the full set of possible weights
Initialise a discrete distribution, $P(\mathbf{I})$ for $\mathbf{I} \in \mathbf{H}$
repeat
  Sample some weights, $\mathbf{C}$ from $\mathbf{H}$, each with probability $P(\mathbf{I})$ without replacement
  Add $\mathbf{C}$ to $\mathbf{W}$
  Remove $\mathbf{C}$ from $\mathbf{H}$
  Estimate the weight values for the resulting network, $\mathbf{W}$ using lasso
  Remove the weights in $\mathbf{W}$ with zero valued coefficients
  Update the weights distribution, $P(\mathbf{I})$ towards those orders that are useful
until Stopping criteria are met

The structure discovery algorithm has a number of hyperparameters, which are described here. The size, n of the sample of fitness function evaluations effects the speed with which the algorithm can discover the correct structure. Larger samples lead to faster learning times as they allow more weights to be included at each model iteration. The trade-off is with the cost of each evaluation. The choice of cost functions for learning the weights is restricted here to squared error or lasso, though others could be used. Lasso is preferred as it automatically sets some weight values to zero and simplifies the decision of which weights to remove. The number of weights maintained in the network should be kept so that it is always less than the number of data samples being learned so the number added at each iteration depends on how many have been removed.

3.2 Interpreting the MOHN Structure

The linkage structure of a MOHN function is explicitly represented in the edges of the hypergraph. Any input with no weight attached to it has no effect on the fitness score and can be removed from the model. Functions that produce a disconnected graph (one in which there are

some variables that cannot be reached by a path from some others) are additively decomposable into subfunctions defined by the subgraphs. Each subgraph can be optimised separately and the global optimum of the whole function is found at the point where each subgraph is at its global optimum.

A MOHN can also be considered as a set of weighted constraints among subsets of variables. W_j defines a constraint with strength ω_j across the variables in the index set $\mathbf{I}_j$ such that the sign of ω_j dictates whether the product of the values across the variables indexed by $\mathbf{I}_j$ should be positive or negative and the magnitude of ω_j dictates the relative importance of the constraint. Maximising the output of the MOHN function is equivalent to maximising the sum of the satisfied weight values. This observation is the basis of the weight satisfaction search described in Section 3.5.

Partial function evaluations are possible from any point in input space. The effect on the output of moving from input point x to a new point, x' where x' differs from x in only a subset of variables, z can be evaluated by considering only the weights and other variables that are connected to the members of z. Consider a function with p inputs and a simple hill climb algorithm in which a single variable is flipped (undergoing a change of sign) if the result improves the fitness score. Let us define the sparsity of a MOHN, s as the average number of weights connected to a variable. A full evaluation of the MOHN requires p variables to be considered, but an incremental update can be done by considering only s variables, a reduction to s/p. In large problems (say, $p > 1000$) and sparse networks ($s < 10$), this represents a significant gain in efficiency.

Evaluations of the average output from the modelled function given a partial input are also simple. Given an input in $\{-1,1,0\}^p$ where 0 indicates a missing or unknown value, a MOHN will output the average value across all input patterns that are made by keeping the $\{-1,1\}$ values fixed and replacing the 0 values with every possible combination across those values. In the field of metaheuristics, such subspaces of the input are known as schemata. This averaging is done in a single evaluation by the MOHN and is a consequence of the way it represents the function as a sum of schema averages.

3.3 MOHNs as Universal Function Models

Model bias in a MOHN is controlled by the choice of which weights to include or exclude. A MOHN with only first order weights and a constant is equivalent to a multiple regression model, taking the form

$$\hat{f}(X) = \sum_{i=0}^{p} \omega_i X_i \tag{8}$$

which is the simplest form the model can take without removing variables. Bias can be reduced by introducing higher order weights. A fully connected MOHN has 2^p weights, and forms a basis for all functions in $f : \{-1,1\}^p \rightarrow \mathbb{R}$. Each weight, ω_j contributes to a single basis function, $\omega_j \prod_{i \in \mathbf{I}_j} X_i$. The MOHN basis is equivalent to the Walsh basis with a change of sign in the parameters on the odd ordered weights. See (Swingler, 2015) for a proof of this equivalence.

3.4 MOHN Search Heuristics

Once a fitness function has been modelled, the model must be searched to find the optimal input configuration. Two possible types of efficiency can be gained from a knowledge of the network structure. Partial evaluations allow local searches to be carried out more quickly as each step does not require a full evaluation of the function and other search techniques may be guided by the linkage patterns the network reveals. This section proposes three methods for searching a

MOHN. The first two make use of the partial evaluation efficiency and the third demonstrates a variable neighbourhood search guided by the MOHN structure.

3.4.1 Local Search

Let us first consider how local search methods can be efficiently implemented in a MOHN. A simple hill climb can be performed by evaluating the impact on the function output of flipping a single input variable at a time. Consider a variable, X_i selected uniformly at random during a local search from a current location, x. Changing the sign of X_i is a candidate local search step to a new point, x', which should be taken if $f(x) < f(x')$. A partial evaluation of the nodes connected to X_i in the MOHN hypergraph for $X_i = 1$ and $X_i = -1$ will reveal which of the two values produces the higher score. Alternatively, a high order version of the node update rule used in a Hopfield neural network (Hopfield, 1982) will determine the maximising value for X_i given the values on the nodes to which it is connected. First we calculate the activation on X_i, called a_i:

$$a_i = \sum_{j:i\in\mathbf{I}_j} \left(\omega_j \prod_{k\in\mathbf{I}_j\setminus i} X_k \right) \tag{9}$$

where $j : i \in \mathbf{I}_j$ makes j iterate over each weight connected to X_i, ω_j is the weight value associated with W_j and $k \in \mathbf{I}_j \setminus i$ iterates over the indices of every node connected to W_j, except X_i itself. A variable's value is then calculated using the threshold function in Equation 10.

$$X_i = \begin{cases} 1 & \text{if } a_i > 0 \\ -1 & \text{otherwise} \end{cases} \tag{10}$$

Setting the values of X to an initial pattern and then repeatedly applying equations 9 and 10 to nodes selected uniformly at random without replacement causes the MOHN to move to an attractor state, from which those equations cause no further change to the node values. This point represents a local minimum in the energy function and a local optimum in the MOHN function. Each calculation in Equation 9 uses only those weights connected to the candidate variable, X_i and so introduces efficiency gains over evaluating the full fitness function proportionate to the sparsity of the variable's connectivity. Repeating this process from randomly selected start points represents a random restart hill climb (RRHC).

For functions with many local optima, RRHC is an inefficient search method. An improvement can be made by setting values with a probability related to the improvement any change will make using simulated annealing (SA) (S. Kirkpatrick, 1983). SA allows local steps to a new point, with a score of y' that is lower than the current point's score, y with a probability dependent on the difference between y' and y and a temperature control, T, which makes large steps more likely earlier in the search. The probability of making such a move is defined as

$$P(y, y', T) = \frac{1}{1 + exp((y' - y)/T)} \tag{11}$$

and as with a local search, the computational cost of calculating the difference $y' - y$ depends on the number of weights and variables connected to the variable under consideration, not the size of the entire function as we can replace $y' - y$ with the activation a_i from Equation 9.

3.5 Weight Satisfaction Search

The limitation of a local search that changes a single variable at each step is that it can fall into local minima in the MOHN energy space (equivalent to local optima in the fitness function). Random restart hill climbing attempts to solve this problem by repeatedly climbing from random

start points. In functions where the basin of attraction of the global optimum is small, the process is reduced to a random search for a starting point in that basin of attraction, which can be very inefficient. Local minima in a single variable neighbourhood hill climb occur when two or more variables interact in their effect on the output. Knowing which variables interact allows an algorithm to become less local in its search by enumerating all the combinations over a small set of connected variables. This approach is known as variable neighbourhood search (Mladenović and Hansen, 1997) and in a MOHN, the membership of each search neighbourhood can be defined by the connectivity pattern of the network. This is particularly useful in functions that are additively decomposable because the independent subgraphs revealed by the MOHN structure can be optimised independently.

High order weights encode weak constraints among several input variables, offering an insight into candidate moves in a variable sized neighbourhood. Any variables that are not connected (i.e., there is no path between them in the hypergraph) may be optimised separately and those that are connected will often form smaller subsets of variables over which it may be possible to find optimal values. Algorithm 2 describes a variable neighbourhood search algorithm that defines the current neighbourhood as all the points that can be reached by changing the variables connected to a single weight. Weights are chosen uniformly randomly without replacement until all have been tried once in an iteration.

Algorithm 2 High Order Weight Satisfaction Search

$X_i = \text{rand}\{-1,1\} \forall i$ ▷ Choose a random starting point
repeat
  $ch = FALSE$ ▷ Keep track of whether or not a change has been made
  $visited = \emptyset$ ▷ Keep track of which weights have been visited
  repeat
    $W_j = rand(W_j : j \notin visited)$ ▷ Pick a random unvisited weight
    $temp = \{X_i : i \in \mathbf{I}_j\}$ ▷ Make a note of its connected values for later comparison
    $\{X_i : i \in \mathbf{I}_j\} = \underset{X_i : i \in \mathbf{I}_j}{\text{argmax}}(f(X))$ ▷ Find the pattern across the connected variables that maximises network function output
    if $X_i : i \in \mathbf{I}_j \neq temp$ **then**
      $ch = TRUE$
    end if ▷ If a change was made to any variable's value, note the fact
    $visited = visited \cup W_j$ ▷ Add the weight to the visited set
  until $\|visited\| = |\mathbf{W}|$ ▷ Loop until all weights have been visited
until $ch = FALSE$ ▷ Loop if any variable has changed

The number of patterns tried when finding the optimum for a given weight is 2^o where o is the order of the weight, so networks with high order weights can produce slow searches. The search can lead to local optima so may need to be repeated. The simplest approach that we propose is a random restart weight satisfaction search, (RRWSS), which repeats Algorithm 2 from random starting points.

4 Experimental Results

This section compares the use of a MOHN as a surrogate fitness function to several approaches and benchmark fitness functions from the literature. In each case, a full MOHN model is built and then searched until a solution is found. Each benchmark fitness function has a known global optimum with a known score and the task in all cases is for the algorithm to discover that solution in the fewest evaluations of the fitness function. For comparison, the number of fitness function

evaluations required by other methods are reported directly from the literature.

For each experiment, the hyperparameters for the structure discovery algorithm were set as follows. The number of weights added at each training iteration was a third of the number of training examples, with a limit to prevent the number of parameters exceeding the number of samples. Lasso was used to estimate the coefficients, which were removed if their value went to zero. The initial probability distribution from which weight orders were picked was a discrete Laplace with a mode of 1 and $\lambda = 1$, which causes the probability of weights at orders over 5 to be almost zero (but does not rule out the distribution mode moving up past that point). No hard limit on the weight orders that the algorithm might consider was set. The weight picking distribution simply favours those with lower order. Discarded weights are stored in a list that prevents them from being tried again. The list is emptied every 15 training iterations. The algorithm stops when the error on an independent validation sample from the fitness function reaches zero. In each experiment, the number of fitness evaluations was fixed and no additional samples were made during the learning process. Trials were repeated to verify that the algorithm completed successfully every time, so the reported number of fitness evaluations is a fixed value, not an average over trials.

4.1 Comparing MOHNs and BMDA

The Bivariate Marginal Distribution Algorithm (BMDA) (Pelikan and Mühlenbein, 1999) is a second order EDA that models conditional probabilities with a pairwise variable interaction graph. Pelikan and Mühlenbein present results that measure the number of fitness function evaluations required to find a global optimum for three different fitness functions using BMDA. This section considers one of them: the quadratic fitness function:

$$f_q(X) = \sum_{i=1}^{\frac{p}{2}} f_2(X_{2i-1}, X_{2i}) \tag{12}$$

where $f_2(u,v)$ is

$$f_2(u,v) = 0.9 - 0.9(u+v) + 1.9uv \tag{13}$$

where $u, v \in \{0,1\}$. For the sake of comparison, this domain is used in the following experiments and inputs of -1 to the MOHN are simply replaced with a value of 0 before evaluation with Equation 13. Pelikan and Mühlenbein pair variables from randomised locations, so there is no prior knowledge about which input interacts with which. No matter how the variables are paired, there are always $p/2$ pairs in a function of p variables.

(Pelikan and Mühlenbein, 1999) report results for a genetic algorithm (GA) and the BMDA algorithm. Functions of between 20 and 120 variables were tested and the number of function evaluations required by BMDA for a model of 120 inputs was around 16,000 and the average number required by the GA was 140,000.

4.1.1 Experiment 1: Fixed Model Structure

If we assume that we know the level of model bias required, in other words, we know that the function has second order interactions only, but that we do not know which variables interact, we can use the OLS learning rule to discover the interacting pairs from a fixed structure second order MOHN. There are $p(p-1)/2$ possible second order interactions among p variables, so we know that a sample of $p(p-1)/2$ fitness function evaluations is required to learn the function perfectly. Any weights that have a coefficient of zero after learning can be discarded, leaving only the $p/2$ non-zero weights that correspond to the pairs in the fitness function. As the function is additively decomposable (variable pairs do not overlap), a weight satisfaction search can trivially find the optimum solution by independently optimising the variable pairs defined by the weights.

This simple solution makes the same assumption about model bias that BMDA makes (all interactions are of second order only) and can provably solve the 120 input problem in $120(120-1)/2 = 7140$ fitness evaluations, which is less than half the number required by the BMDA. If the second order assumption is relaxed, the problem becomes more difficult because interactions of any order must be considered and there are 2^p of those in a model with p inputs. This also provides an opportunity to reduce the number of samples required as the structure discovery algorithm can be used. The next experiment addresses this.

4.1.2 Experiment 2: Structure Discovery

The MOHN structure discover algorithm was run on each model from size 20 to 120 in steps of 10. A sample size of $p(p-1)/4$ was used for each model of size p, making a rather arbitrary halving of the number of samples used compared to training a full second order model. No maximum weight order was set, however, leaving the MSDA to discover for itself that all connections were of second order.

The MOHN Structure Discovery Algorithm was able to find the correct MOHN structure and weights in every trial from size 20 to 120, achieving an error of zero and finding the optimal input pattern. A summary of the results achieved in (Pelikan and Mühlenbein, 1999), a fully connected second order MOHN and MSDA are shown in Figure 1. The number of fitness evaluations required to optimise the function are plotted against the number of variables in the function input. In the original paper, Pelikan and Mühlenbein also showed the results for a genetic algorithm but they were so much higher than the figures presented here they have been excluded from the chart. The GA required 140,000 fitness evaluations for the 120 variable version of the function.

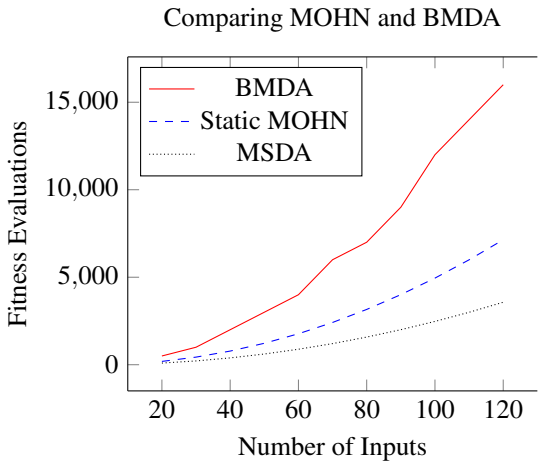

Figure 1: Number of fitness function evaluations required to optimise the quadratic fitness function using BMDA and a MOHN. The top line shows the figures for BMDA taken from (Pelikan and Mühlenbein, 1999), and the other two are different approaches to training a MOHN. Note that the MSDA requires fewer evaluations, even though it is searching a much larger space than the other two, fixed, methods.

4.2 Comparing MOHNs and Markov Random Field EDAs

A Markov Random Field (MRF) can be used to model any distribution over a set of binary random variables. The model can then be sampled using Gibbs sampling. This has led several

researchers to investigate the use of MRFs for modelling distributions of promising solutions in EDAs. This section compares two MRF based EDAs from the literature with a MOHN fitness model given the task of optimising different Ising models. The EDAs are Density Estimation using Markov Random Fields (DEUM) (Shakya et al., 2009) and Markovian Learning Estimation of Distribution Algorithm (MARLEDA) (Alden and Miikkulainen, 2013).

4.2.1 Ising Spin Glass Learning

An Ising spin glass model defines an energy function over a lattice of neighbouring variables. It calculates a weighted sum of the products of the values on each set of variables in a neighbourhood.

$$H(X) = -\sum_{\langle i,j \rangle} J_{i,j} X_i X_j \tag{14}$$

where $J_{i,j}$ represents the weight of the connection between variables X_i and X_j. The notation $\langle i, j \rangle$ defines X_i and X_j as neighbours. In a 2D Ising model, each variable has four neighbours (above, below, left and right) and in a 3D lattice there are six neighbours. In all cases, the field is toroidal so that there are no edge variables with fewer neighbours.

The Ising model has been a popular choice of fitness function in the EDA literature as finding the input values that maximise the output of Equation 14 is a challenging problem. DEUM (Shakya et al., 2009), was tested on 2D Ising models of 100, 200 and 400 variables. MARLEDA (Alden and Miikkulainen, 2013) was tested on 2D Ising models of 400 variables and sDEUM (Valentini et al., 2012) reported results from a 125 variable 3D Ising model. Both attempt to discover the correct structure for the MRF using tests on pairwise interactions among the variables. MARLEDA uses a chi-square test to identify pairs of variables that interact and DEUM calculates mutual information to the same end. DEUM infers higher order connections from cliques in the second order graph.

A 10×10 Ising function in the form of Equation 14 is defined by 200 parameters so if the structure is known but the coefficients are unknown, the coefficients can be estimated exactly by a MOHN from 200 fitness function evaluations. If the structure is not known, but the assumption that variables are connected only in pairs is made, then the situation is the same as it was for the quadratic function described above. There are $p(p-1)/2$ possible connections. The structure can be discovered from that many fitness function evaluations. In the case of a 100 variable Ising, that is 4,950 samples. That figure can be improved upon further using the MOHN Structure Discovery Algorithm, which adds and removes weights in an attempt to reduce the number of samples required. In (Shakya et al., 2009), the 100 variable MRF was built using a population of 30,000 fitness evaluations from which 5,000 were selected to estimate the structure and just 250 were used for the parameter estimation. These quantities were fixed by design and not discovered as part of the learning process.

Experimental Results Experiments were performed with randomly generated 10×10 Ising models. In each experiment, a MOHN was used to model the fitness function and the model was then searched to reveal the solution that minimised the Ising energy function using simulated annealing, as described in Section 3.4.1. The learning process used the MSDA to discover the correct structure and estimate the parameter values at the same time. The optimal state for the Ising model was obtained using the on line resource from the University of Cologne [1].

A sample of 3,000 fitness function evaluations was generated to train the model, which is a slightly arbitrary amount, chosen to be significantly less than the 4,950 required to try every second order weight. As Ising models have only second order relationships, the algorithm

[1] http://www.informatik.uni-koeln.de/spinglass/

quickly learned that only second order parameters were needed and converged on the correct structure. At the point where the model parameters included all of the 200 required parameters for the model, the training error dropped to zero and the algorithm terminated. In all cases, 3,000 samples were sufficient for the model to learn correctly and the simulated annealing process produced the correct energy minimisation state.

Figure 2 shows the training and validation RMSE of the MOHN by epoch as it learned one of the Ising models. The error trace reflects a number of properties of the algorithm. The reduction in error during coordinate descent is visible between peaks that show the points where weights are discarded and new weights are added. It is also clear from the trace that although these changes in weights cause the error to spike, they do not take the error back up to the point it was at when the network was new. This is evidence of the efficiency of the error descent approach over learning each network from scratch at each iteration. Validation error is greater than training error while the MSDA is searching for the correct weights, but as more of the required weights are found, the gap between the training and validation error closes until, at the point where the model is correct, they converge.

Figure 3 shows the number of weights at each order from 1 to 5 in the network as it learned, with the top line showing the total number of weights. Note the spikes at iterations 15 and 30 where the history of weights to avoid was emptied and the way the number of second order weights grows as the rest reduce in number in the second half of the training. At the final step, the number of second order weights reaches 200 (which is the correct number) and the rest all drop to zero.

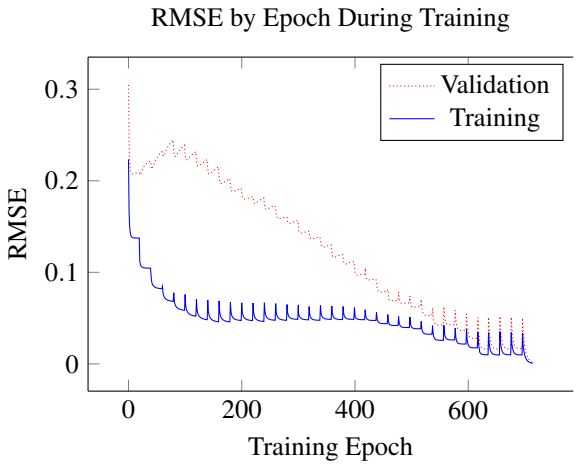

Figure 2: Training and Validation RMSE during MSDA learning of a 100-node Ising model from 3,000 training samples.

4.3 3D Ising Models, Simulated Annealing and hBOA

An approach to choosing which weights to include in an MRF EDA was proposed by (Valentini et al., 2012), who used the lasso to set unused weights to zero in an approach they called Sparsified DEUM (sDEUM). They presented results comparing standard DEUM, sDEUM, simulated annealing and hBOA given the task of finding the global optimum in a 3D Ising model. A 3D model extends the neighbourhood of each node to those other nodes that are its neighbours in a cube. The largest model analysed in (Valentini et al., 2012) contained $5 \times 5 \times 5 = 125$ nodes and it is that size of network that is used to compare the performance of a MOHN. sDEUM required

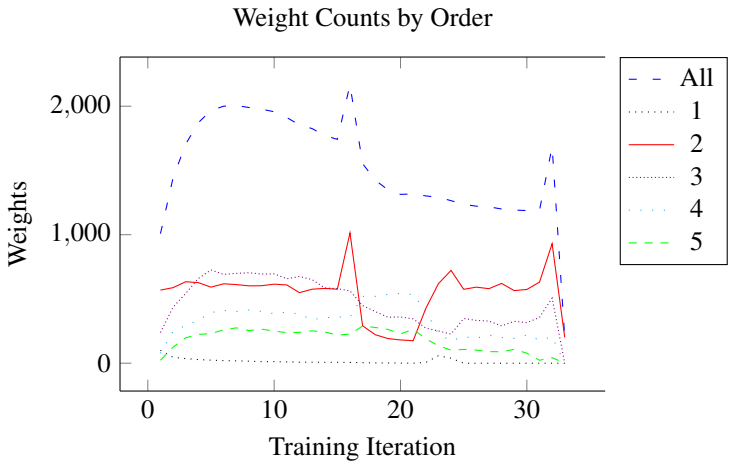

Figure 3: Weight counts at different orders during MSDA learning a 100 node Ising model.

an average of around 20,000 evaluations of the 125 node Ising energy function to find an optimal input pattern. The next experiment attempts to solve the same problem in 5,000 evaluations.

4.3.1 Experimental Results

The MSDA was used to discover the correct structure of randomly generated 3D Ising models of 125 nodes. For each trial, a training set of 5,000 examples was generated, consisting of uniformly random input patterns and their associated output from the Ising energy function. In each of the 20 trials, the MOHN was able to learn the correct structure and weights of the target 3D Ising model from a sample of 5,000 fitness function evaluations (i.e., the sum of errors on validation data was zero). Figure 4 reproduces part of Figure 3 from (Valentini et al., 2012) showing the results reported in that paper for a 125 node Ising model with an additional entry for the MOHN.

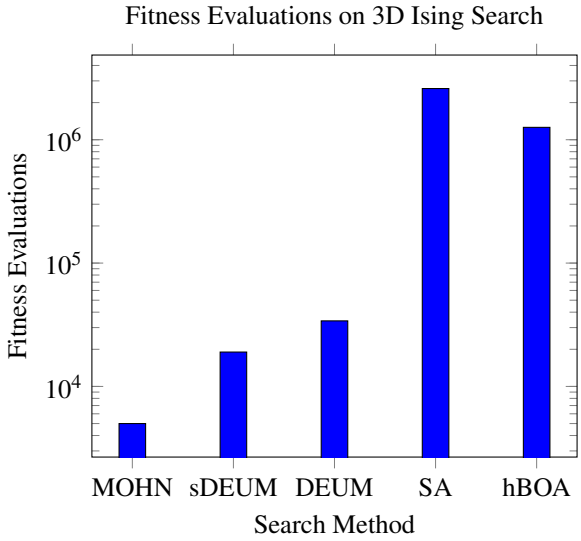

Figure 4: Average number of fitness evaluations (log scale) required to find the first optimal solution to a 3D Ising model by different algorithms.

4.4 Comparing Performance on *mk*-Trap Problems

Trap functions are common in optimisation research because they can lead heuristic search algorithms into local optima from which it is very difficult to escape. *mk*-trap functions are counting functions based on m concatenated subvectors of k variables. An input vector is split into non-overlapping subvectors, $C \subset X$ of size k and each subvector is scored separately. The function output is the sum of the subvector scores. Each subvector is scored by counting the number of bits set to 1 and letting patterns with k 1s (all of them) score k, but letting patterns with $< k$ 1s score $k-1-b$ where b is the number of bits set to 1:

$$f(C) = \begin{cases} k, & \text{if } b = k \\ k-1-b, & \text{otherwise} \end{cases} \tag{15}$$

and the sum is calculated as

$$f(X) = \sum_{C \subset X} f(C) \tag{16}$$

This section compares MOHN performance on *mk*-trap problems to the Bayesian Optimisation Algorithm (Pelikan et al., 2000), the Linkage Tree Genetic Algorithm (Thierens, 2010) and two Boltzmann machine based EDAs. Boltzmann machines are a type of neural network that use a stochastic activation function that enables them to model probability distributions. They represent dynamic systems that can be used to generate data in a Boltzmann distribution using Gibbs sampling. Both deep Boltzmann machines (Probst and Rothlauf, 2015) and restricted Boltzmann machines (Probst et al., 2014) have been used to build EDAs for combinatorial optimisation.

Both (Probst and Rothlauf, 2015) and (Probst et al., 2014) present results for searching *mk*-trap problems of various sizes with Boltzmann EDAs, reporting the number of fitness evaluations and the CPU time taken to find a solution. They used an AMD Opteron 6272 processor with 2.1 GHz. For comparison, the MOHN process was run on a single core Intel i7 processor

running at 2.5 GHz. (Probst and Rothlauf, 2015) used deep Boltzmann machines in a method called DBM-EDA and (Probst et al., 2014) used restricted Boltzmann machines in a method called RBM-EDA. Both papers compared the performance of the EDAs to that of a Bayesian Optimisation Algorithm (BOA) on a number of problems including the *mk*-trap.

Unlike the method reported above for MRF EDAs, the Boltzmann EDAs make use of a number of generations of a cycle of data generation, selection and modelling to perform an evolutionary search. The principle behind an evolutionary EDA is that the model can be simpler as only the space of promising (and eventually, very good) solutions is modelled. The risks associated with the evolutionary approach are that larger samples from the fitness function may be needed to build multiple populations. This is in contrast to the approach of building and then sampling an accurate model reported by (Malago et al., 2011) and (Shakya et al., 2009) and employed by the MOHN model-and-search approach.

4.4.1 Experimental Results

Four different *mk*-trap problems were modelled and searched using a MOHN. They were four bit traps over 40 and 80 bits, and five bit traps over 25 and 50 bits. The MSDA was used with the same hyperparameter settings for every trial. Lasso was used to estimate the network parameters at each structure discovery iteration and the number of fitness function samples used for learning was fixed for each experiment based on the size of the problem and the number of samples reported by (Probst and Rothlauf, 2015). Each fitness function was modelled 10 times, each with a new fixed sized sample of fitness evaluations. The purpose of the repeated trials is not to calculate an average number of evaluations required (that value is fixed) but to verify that a solution can be found reliably.

In all cases, the MOHN was able to model and successfully search the fitness function in far fewer evaluations and in much less time than the results reported for LTGA, RBM-EDA, DBM-EDA and BOA. Table 1 summarises the results, taking data from (Thierens, 2010), (Probst and Rothlauf, 2015) and (Probst et al., 2014). Note that the results from the DBM-EDA are for trials where the global optimum was found 90% of the time or more. All other results provide numbers where all searches found the global optimum. The BOA and DBM-EDA figures were taken from Table 1 in (Probst and Rothlauf, 2015). The figures for RBM-EDA are approximate as they were read from the graphs in Figure 3 in (Probst et al., 2014).

Problem	Algorithm	Evaluations	Time
4-trap	BOA	13,673	2,728
40 bits	DBM-EDA	47,231	2,201
	RBM-EDA	16,000	150
	MOHN	**2,000**	**22**
4-trap	BOA	43,777	43,935
80 bits	DBM-EDA	153,278	13,271
	RBM-EDA	160,000	1,100
	MOHN	**10,000**	**170**
5-trap	BOA	14,924	1,384
25 bits	DBM-EDA	13,291	566
	LTGA	15,069	-
	MOHN	**1,000**	**8**
5-trap	BOA	47,904	20,199
50 bits	DBM-EDA	49,886	3060
	RBM-EDA	63,000	300
	LTGA	43,933	-
	MOHN	**20,000**	**295**

Table 1: Number of unique fitness function evaluations and CPU time in seconds required to find the global optimum in different *mk*-trap functions using a MOHN and the figures presented for DBM-EDA and BOA (Probst and Rothlauf, 2015), RBM-EDA (Probst et al., 2014) and LTGA (Thierens, 2010).

The models built by the MOHN were searched using weight satisfaction search (see Section 3.5), which was able to find the global optimum in a single pass of the algorithm. This is very fast, adding only one or two seconds to the total search time because the function is perfectly suited to the WSS as each trap is small and can be solved independently. These *mk*-trap problems are easy to search once the structure is known but the structure is not trivial to discover so almost all of the time taken to arrive at a solution is taken up by the modelling process.

The reader might ask whether the time gained by making fewer fitness function evaluations is lost to the model building process. Table 1 shows some evidence that the MOHN approach is faster than some other methods but more generally the answer will depend on how expensive the fitness function evaluations are. There is a trade-off between the number of fitness function evaluations available and the number of iterations required by the structure discovery algorithm. Efficiency will also depend on the number of parameters required to model the fitness function. Some functions are expensive to model but easy to search and such problems will never be solved quickly by a surrogate model approach. The model building process for a MOHN has not been found to be as time consuming as some other methods as the combination of a convex cost function and the use of warm starts from one iteration to the next of the MSDA speeds up the learning process. Evidence for this can be seen in Figure 2, in which the error does not increase appreciably with each addition of new weights during the MSDA search.

4.4.2 Visualising the MOHN

The connection structure of a MOHN can reveal useful insights into the fitness function it represents. We have already noted that the *mk*-trap MOHN is easy to search as it reveals the additively decomposable nature of the function. This can be seen by visualising the MOHN in a heat map. Figure 5 shows an example for a correctly fitted 5-trap problem over two repeated traps. The interactions among the variables in each trap are plain to see, as is the lack of inter-trap connections. It is also clear that any extra or missing weights would be identifiable to the human eye

from the pattern formed in Figure 5, which allows a human element to be included in the search for the correct weights to include in a model.

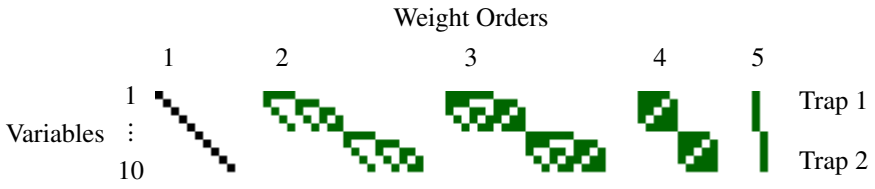

Figure 5: Weights of a MOHN trained on two concatenations of a 5 bit trap. Each column represents one weight, a visible pixel indicates a connected variable, and weights increase in order (the number of variables connected) from left to right. Each row represents a single variable so the number of pixels in a column indicates the order of the weight and the index of the variables it connects. For example, there are two order 5 weights, one in each trap. The blocks that make up each trap are easy to identify and the fact that the function is additively decomposable is clear from the diagram as there are no connections from within one block to another. We can see that the maximum weight order is 5 and that variables within blocks are fully connected.

5 Conclusions

This paper has presented MOHNs as universal fitness function models for pseudo-Boolean problems. An incremental model building approach is used to discover linkage structure from a sample of fitness function evaluations that is much smaller than the number of potential variable interactions being searched. This reduces the number of fitness function evaluations required to build an accurate model. MOHNs expose the linkage structure of the function they are trained on, which can be used to guide a search of the model for the optimal input configuration. MOHNs have been shown to be able to model and optimise fitness functions including quadratics, Ising models and *mk*-traps in far fewer evaluations than reported in the literature for EDAs and linkage learning algorithms.

There are many similarities among some of the other methods described in this paper and the MOHN approach. A fully connected MOHN represents a Walsh-like basis and a partially connected MOHN learns the non-zero coefficients in a similar manner to that used by (Verel et al., 2018) to build surrogate models. The MOHN Structure Discovery Algorithm attempts to reduce the number of fitness samples required, unlike that of (Verel et al., 2018). The graph structure of Markov Random Field models such as DEUM represents non-zero Walsh coefficients in its connectivity pattern. DEUM uses OLS to learn the parameters and sDEUM uses lasso. The two differences are that DEUM learns the log of the fitness values to allow samples from the model to follow a Boltzmann distribution and chooses the connectivity pattern by fully connecting all cliques in the graph produced by testing for paired dependencies. The MOHN model is presented here in a surrogate model-based optimisation framework, but the MSDA is suitable for learning the structure of an MRF with very little modification.

5.1 Research Questions

This work motivates a number of new research questions. Although the results presented in this paper show that MOHNs can model certain benchmark problems efficiently, it is clear that these functions are well suited to the modelling approach. They have low order linkage among variables and in some cases are additively decomposable. It is easy to imagine functions with linkage at many different orders or even with full linkage at all orders.

Three strands of further work are ongoing. The structure discovery algorithm currently works from a fixed sample of fitness evaluations. In future work, the algorithm will iteratively take further samples as they are needed. Other methods for searching models once they are built are also under development, including the use of the linkage structure to guide a GA and variable interaction graph searching methods (Chicano et al., 2014). Finally, a hybrid model-and-search approach is needed in which the data generated by a metaheuristic search are modelled concurrently with the search, making use of each evaluation once for the search and once in the model. It is hoped that this may provide an effective method for making double use of each fitness function evaluation.

References

Alden, M. and Miikkulainen, R. (2013). Marleda: Effective distribution estimation through Markov random fields. Technical Report TR-13-18, Department of Computer Science, The University of Texas at Austin, Austin, TX.

Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In *International Conference on Machine Learning*, pages 38–46. Morgan Kaufmann.

Chen, Y.-p. (2008). *Linkage in evolutionary computation*, volume 157. Springer.

Chicano, F., Whitley, D., and Sutton, A. M. (2014). Efficient identification of improving moves in a ball for pseudo-boolean problems. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 437–444. ACM.

Coffin, D. and Smith, R. E. (2008). Linkage Learning in Estimation of Distribution Algorithms. In *Linkage in evolutionary computation*, pages 141–156. Springer.

Davidor, Y. (1990). Epistasis variance: A viewpoint on GA-hardness. In *Foundations of Genetic Algorithms*, pages 23–35, San Francisco. Morgan Kaufmann.

Frean, M. and Boyle, P. (2008). Using gaussian processes to optimize expensive functions. In *Australasian Joint Conference on Artificial Intelligence*, pages 258–267. Springer.

Goldberg (1989a). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems*, pages 3–153.

Goldberg, D. E. (1989b). *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley.

Gutmann, H.-M. (2001). A radial basis function method for global optimization. *Journal of global optimization*, 19(3):201–227.

Harik, G. R. and Goldberg, D. E. (1997). Learning linkage. In *Foundations of Genetic Algorithms 4*, pages 247–262. Morgan Kaufmann.

Harik, G. R., Lobo, F. G., and Sastry, K. (2006). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). In *Scalable Optimization via Probabilistic Modeling*, pages 39–61. Springer, Berlin, Heidelberg.

Heckendorn, R. B. and Whitley, D. (1997). A Walsh analysis of nk-landscapes. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 41–48. Morgan Kaufmann.

Heckendorn, R. B. and Wright, A. H. (2004). Efficient linkage discovery by limited probing. *Evolutionary computation*, 12(4):517–545.

Holeňa, M., Linke, D., Rodemerck, U., and Bajer, L. (2010). Neural networks as surrogate models for measurements in optimization algorithms. In Al-Begain, K., Fiems, D., and Knottenbelt, W. J., editors, *Analytical and Stochastic Modeling Techniques and Applications*, pages 351–366, Berlin, Heidelberg. Springer Berlin Heidelberg.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 79(8):2554–2558.

Iuliano, E. and Quagliarella, D. (2013). Proper orthogonal decomposition, surrogate modelling and evolutionary optimization in aerodynamic design. *Computers & Fluids*, 84:327–350.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9:3–12.

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70.

Kennedy, J. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco.

Kim, Y.-H., Moraglio, A., Kattan, A., and Yoon, Y. (2014). Geometric generalisation of surrogate model-based optimisation to combinatorial and program spaces. *Mathematical Problems in Engineering*, 2014.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). *Iterated Local Search*. Springer.

Malago, L., Matteucci, M., and Valentini, G. (2011). Introducing the L1-regularized logistic regression in Markov networks based EDAs. In *The 2011 Congress on Evolutionary Computation*, pages 1581–1588. IEEE.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, 24(11):1097 – 1100.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346.

Pelikan, M. (2005). Hierarchical Bayesian Optimization Algorithm. In *Hierarchical Bayesian Optimization Algorithm*, pages 105–129. Springer.

Pelikan, M., Goldberg, D., and Cantú-Paz, E. (2000). Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary computation*, 8(3):311–340.

Pelikan, M., Hauschild, M. W., and Lobo, F. G. (2015). Estimation of distribution algorithms. In *Springer Handbook of Computational Intelligence*, pages 899–928. Springer.

Pelikan, M. and Mühlenbein, H. (1999). The Bivariate Marginal Distribution Algorithm. In Roy, R., Furuhashi, T., and Chawdhry, P. K., editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London. Springer-Verlag.

Probst, M. and Rothlauf, F. (2015). Deep Boltzmann machines in estimation of distribution algorithms for combinatorial optimization. *arXiv preprint arXiv:1509.06535*.

Probst, M., Rothlauf, F., and Grahl, J. (2014). Scalability of using restricted Boltzmann machines for combinatorial optimization. *arXiv preprint arXiv:1411.7542*.

S. Kirkpatrick, C. D. Gelatt, M. P. V. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Shakya, S., Brownlee, A., McCall, J., Fournier, F., and Owusu, G. (2009). A fully multivariate DEUM algorithm. In *The 2009 Congress on Evolutionary Computation*, pages 479–486.

Shakya, S., McCall, J., Brownlee, A., and Owusu, G. (2012). DEUM - distribution estimation using Markov networks. In Shakya, S. and Santana, R., editors, *Markov Networks in Evolutionary Computation*, volume 14 of *Adaptation, Learning, and Optimization*, pages 55–71. Springer Berlin Heidelberg.

Swingler, K. (2015). A comparison of learning rules for mixed order hyper networks. In *Proceedings of the 7th International Joint Conference on Computational Intelligence*, pages 17–27, Setubal. ScITePress.

Swingler, K. (2016a). *Mixed Order Hyper Networks for Function Approximation and Optimisation*. PhD thesis, Stirling University.

Swingler, K. (2016b). Structure discovery in mixed order hyper networks. *Big Data Analytics*, 1(1):8.

Thierens, D. (2010). The linkage tree genetic algorithm. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*, PPSN'10, pages 264–273, Berlin, Heidelberg. Springer-Verlag.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58:267–288.

Tsuji, M., Munetomo, M., and Akama, K. (2004). Modeling dependencies of loci with string classification according to fitness differences. In *Genetic and Evolutionary Computation*, pages 246–257. Springer.

Valentini, G., Malagò, L., and Matteucci, M. (2012). Optimization by L1-constrained Markov fitness modelling. In *Learning and Intelligent Optimization*, pages 250–264. Springer.

Verel, S., Derbel, B., Liefooghe, A., Aguirre, H., and Tanaka, K. (2018). A surrogate model based on Walsh decomposition for pseudo-boolean functions. In *International Conference on Parallel Problem Solving from Nature*, volume 11102 of *Lecture Notes in Computer Science*, pages 181–193, Coimbra, Portugal.

Walsh, J. (1923). A closed set of normal orthogonal functions. *American Journal of Mathematics*, 45:5–24.

Willmes, L., Back, T., Jin, Y., and Sendhoff, B. (2003). Comparing neural networks and Kriging for fitness approximation in evolutionary optimization. In *The 2003 Congress on Evolutionary Computation*, volume 1, pages 663 – 670 Vol.1.