UNIVERSITY OF SOUTHAMPTON

Deep Optimisation: Learning and Searching in Deep Representations of Combinatorial Optimisation Problems

by

Jamie R. Caldwell

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

June 2022

Research Thesis: Declaration of Authorship

Print name: Jamie Robert Caldwell

Title of thesis: Deep Optimisation: Learning and Searching in Deep Representations of Combinatorial Optimisation Problems

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:
    - JR Caldwell and Richard A Watson. How to get more from your model: the role of constructive selection in estimation of distribution algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 101–102, 2017
    - JR Caldwell, Richard A Watson, C Thies, and Joshua D Knowles. Deep optimisation: Solving combinatorial optimisation problems using deep neural networks. *arXiv preprint arXiv:1811.00784*, 2018
    - Jamie Caldwell, Joshua Knowles, Christoph Thies, Filip Kubacki, and Richard Watson. Deep optimisation: Multi-scale evolution by inducing and searching in deep representations. In Pedro A. Castillo and Juan Luis Jiménez Laredo, editors, *Applications of Evolutionary Computation*, pages 506–521, Cham, 2021. Springer International Publishing. ISBN 978-3-030-72699-7

Date: 21st June 2022

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Jamie R. Caldwell

Evolutionary algorithms are a class of optimisation techniques used to solve problems by emulating evolutionary processes (variation and selection) to search the solution space. In this thesis, we focus on the evolutionary process of Evolutionary Transitions in Individuality (ETI). In this case, evolutionary processes are scaled up via a multi-scale process whereby individuality (and hence variation and selection) is continually revised by forming associations between formerly independent entities. This thesis develops a novel Model-Building Optimisation Algorithm (MBOA) called Deep Optimisation (DO) that exploits deep learning methods to enable multi-scale optimisation. DO uses an autoencoder model to induce a multi-level representation of solutions. Variation and selection are then performed within the induced representations, allowing search to continue in a new and reorganised space. By using a class of configurable problems, we find and understand more precisely the distinct problem characteristics that DO can solve that other MBOAs cannot. Specifically, we observe a polynomial vs exponential scaling distinction where DO is the only algorithm to show polynomial scaling for all problems. We also demonstrate that some problem characteristics need a deep network in DO. Further, for the first time, we show that overlap differentiates the performance between current MBOAs that are considered state-of-the-art. DO is then applied to different optimisation problem domains to demonstrate its potential for exploiting unknown problem structure and overcoming infeasible solution spaces. Here, DO shows impressive performance and does so without using a domain-specific operator. This thesis provides a connection between deep learning models and MBOAs, showing results that outperform existing algorithms can be achieved by utilising the tools available in deep learning. This suggests numerous avenues for further investigation, transferring deep learning methods into the domain of MBOAs.

Contents

List of Figures

List of Tables

Nomenclature

GA	Genetic Algorithms
MBOA	Model-Buidling Optimisation Algorithms
EDA	Estimation of Distribution Algorithms
MIG	Model-Informed Generation
MIC	Model-Informed Crossover
MIV	Model-Informed Variation
DO	Deep Optimisation
LTGA	Linkage-Tree Genetic Algorithm
P3	Parameterless Population Pyramid
DSMGA-II	Dependency-Structure Matrix Genetic Algorithm
BOA	Bayesian Optimisation Algorithm
hBOA	Hierarchical Bayesian Optimisation Algorithm
rHN-G	Restart Hopfield Network with Generative Associations
nOV	non-Overlapping Variation
OV	Overlapping Variation
nDOV	non-Distinguishable Overlapping Variation
nPOV	non-Pairwise Overlapping Variation
GC	Generating Combinations
HGC	Hierarchically Generating Combinations
RS	Rescaling Search
MKP	Multi-dimensional Knapsack Problem
TSP	Travelling Salesman Problem

Acknowledgements

Thank you to all those who I have learned from, received feedback and direction from, and to those who have spent countless hours discussing the ideas presented in this dissertation. It has been an incredible journey considering where we first started to where we have ended up, and this could not have been achieved without the help and support from others. Thank you to Josh, Miguel, Filip and in particular Dave, Christoph, and Freddie for taking the time to really understand my work and help develop and shape the thesis throughout. A huge thank you to Richard for all your support and time; I have learned a lot both in and outside this thesis that I will be forever grateful for. Finally, thank you for the support from those outside the academic lifestyle, to my family Julie, Steven, and Rebecca and partner Jessica; thank you.

To Mother

Chapter 1

Introduction

Evolutionary computing is the study of computational algorithms that emulate evolutionary processes to find good solutions to an optimisation problem. Specifically, an organism (solution) is adapted via variation and selection (search operators) to maximise (optimise) survival in an environment (problem). The dynamical process of evolution provides the mechanism of adaptation to a solution to the environment - finding a high-quality solution. Evolution has proven to provide extraordinary adaptations to an organism in a variety of challenging environments. It is with this observation that evolutionary computing has been extensively studied for solving optimisation problems (De Jong, 2006; Eiben and Smith, 2015).

Understanding the mechanisms that provide adaptation via an evolutionary process remains an extensive area of research both in evolutionary biology (Laland et al., 2015) and evolutionary computation (Eiben and Smith, 2015). In Genetic Algorithms (GA), it is hypothesised that higher-order solutions are found by conserving low-order adapted variable combinations whilst maintaining variability between them. This is formally known as the Building-Block Hypothesis (Goldberg, 1989b; Holland et al., 1992a). The emergence of building blocks reduces the dimensionality of the search space so that variation between individual units is rescaled to variation between combinations of units.

The Building-Block Hypothesis provides a thought-provoking mechanism for rescaling the search process from searching combinations of individual variables to searching combinations of variables (building block). This idea of decomposition and using sub-solutions to help solve the global problem is familiar in many areas of engineering. Specifically, the idea of decomposing a complex problem into easier to solve sub-problems, finding solutions to these sub-problems, and then combining these solutions to solve the original problem. The building block ideas in genetic algorithms were inspired by the biology of crossover, which exchanges material between solutions. In this case, variable combinations that contributed to the quality of a solution early in the search are used to guide future variation. In a genetic algorithm, the Building-Block

Hypothesis has not been proven beyond the first generation and is further limited to low-order (short) building blocks, where the likelihood of disruption is small (O'Reilly and Oppacher, 1995; Beyer, 1997). Additionally, in the case of hierarchical building-block problems, where lower-level building blocks combine to construct a higher-level building block, genetic operators that perform recombination using two parents are only successful if variables within a building block are located physically close at the solution representation. (Thierens and Goldberg, 1993; Thierens, 1995, 1999; Watson et al., 1998).

Consequently, significant attention has been given to improving evolutionary computation by using machine learning to explicitly capture and exploit the building block information and overcome the challenge of positional bias during recombination (Hauschild and Pelikan, 2011). In particular, Estimation of Distribution Algorithms (EDAs) use a probabilistic model to replace crossover and mutation operators in a GA (Pelikan and Goldberg, 2006), Linkage Learning methods use a model to capture dependencies between variables to inform crossover masks used during recombination (Thierens, 2010; Goldman and Punch, 2014; Hsu and Yu, 2015) and Multi-Scale Search Algorithms use machine learning models to bias the variability of an individual solution (Iclanzan and Dumitrescu, 2007; Mills et al., 2014; Cox, 2015; Watson et al., 2011b). As we explore in this thesis, all methods use a machine learning model to induce a compressed representation of the search space and then use this space to find higher-order solutions. As we explore, the interpretation of what the model represents and how the model is used differs between these algorithms. We refer to these algorithms as Model-Building Optimisation Algorithms (MBOAs)[1].

The inspiration used in this thesis for developing the algorithm of Deep Optimisation, and specifically the process of how to learn a deep model representation of the problem structure and use this information to inform search, comes from the ideas associated with the theory of Evolutionary Transitions in Individuality (West et al., 2015). The Evolutionary Transitions in Individuality process describes transforming the representation and scale that natural selection can act on, where individual evolutionary units at one level of organisation form associations that result in a new evolutionary unit at a higher level of organisation (Maynard Smith and Szathmáry, 1997). As a result, this allows for natural selection to act on the higher-level units - natural selection can decide which higher-order evolutionary unit is fitter than other higher-order evolutionary units. Further, the process of Evolutionary Transitions in Individuality is recursive, producing successive hierarchical transitions in individuality and allowing natural selection to apply to multiple scales of organisation (Watson and Szathmáry, 2016).

[1]In this thesis, the term MBOA refers only to evolutionary algorithms that use models to replace the recombination operator. Specifically, using a model to learn relationships in the solution space and then using this information to search the space. In this thesis, the term does not include other methods that can also be considered model building, such as surrogate methods.

Deep Neural networks provide a model space to explicitly capture and represent a multi-level representation, with each layer inducing higher-order representations from the layer below. Further, recent success in deep learning has provided a scalable method for constructing deep representations of a data set (Hinton and Salakhutdinov, 2006). In doing so, they have produced state-of-the-art results for many tasks (Krizhevsky et al., 2012; Hinton et al., 2012a; Cho et al., 2014; Silver et al., 2016), but not including optimisation. Inspired by the connections between developmental processes and associative learning (Watson et al., 2014; Watson and Szathmáry, 2016), in this thesis, we connect deep neural network models and evolutionary computing. We call this approach Deep Optimisation (DO). A significant difference between DO and other MBOAs is the ability to represent a solution at multiple organisation levels and perform variation and selection at any of these levels. In DO, solution variables are encoded and compressed into a collapsed space which enforces relationships between the units in the layers below. The relationships required for this encoding contribute to the solution's quality and are identified by previous experience. Variation and selection are performed at this induced representation by performing a local variation at the hidden layer and decoding back into the solution space, producing a large but organised variation that respects lower-level constraints. In an iterative selection process (local search in the latent space), DO represents the process of variation and selection acting on high-order evolutionary units. A deep model is constructed by the recursive process of inducing a higher-level representation of solutions found by variation and selection performed at the representation level below. In this thesis, we develop an algorithm that performs the described process, and we look to answer the following three questions:

1. How can a deep learning model be used to induce a higher-order representation of a solution space, and what information does this capture?
2. How can information captured by a deep neural network model be used to inform search?
3. What can search in a deep representation of a solution do that other algorithms limited to a shallow representation cannot?

In this thesis, we select the Linkage-Tree Genetic Algorithm Thierens (2010), The Hierarchical Bayesian Optimisation Algorithm Pelikan et al. (2003), The Parameterless Population Pyramid Goldman and Punch (2014), and the Dependency Structure Matrix Genetic Algorithm - II Hsu and Yu (2015) from the literature for comparison with the performance of Deep Optimisation. These algorithms are selected because they are considered state-of-the-art within the class of evolutionary algorithms that build models from a population of solutions and then use this model to find new solutions. Specifically, they have produced the best performance on a range of synthetic problems Goldman and Punch (2015); Hsu and Yu (2015); Pelikan and Goldberg (2006) in comparison to other algorithms available in the same class. Note that other algorithms exist outside the class of algorithms considered in this thesis that are more suitable for the problems investigated in this thesis. However, this is not the claim of the thesis. Instead, we focus on if and how a deep neural network can improve this class of algorithms. Therefore, we specifically mean within this class of algorithms when referring to state-of-the-art performance.

Performance comparisons previously performed between these MBOAs show that less sophisticated models, such as using a linkage tree, can sometimes outperform more sophisticated models, such as using a Bayesian network, with regards to the number of function evaluations performed to find a global optimum (Thierens and Bosman, 2013; Goldman and Punch, 2015; Hsu and Yu, 2015). Importantly, results do not show an exponential vs polynomial differentiation (what one model can do that another cannot). As we explore in this thesis, MBOAs differ significantly in the model capacity and the method for informing search using the model. Further, using a neural network model within the class of MBOAs has not yielded competitive performance. Nevertheless, neural network models have provided state-of-the-art results in the domains of generation, classification, regression and dimensional reduction tasks. Consequently, in our aim to understand Deep Optimisation, we also aim to understand how different models and different methods for exploiting information from the model affect the ability of an algorithm to explore the solution space. This dissertation aims to connect neural network models and MBOAs via the algorithm of Deep Optimisation, leading to the main research question:

Can a deep neural network
improve evolutionary optimisation?

In this thesis, we show how the algorithm of DO can overcome problem challenges that other MBOAs explored in this thesis cannot. We do this by first exploring the mechanisms of informing search by comparing the performance between an example of an Estimation of Distribution Algorithm and a Multi-Scale Search Algorithm. We find that even though an Estimation of Distribution Algorithm uses a more sophisticated model than the Multi-Scale Search Algorithm, it can better exploit the information. We then develop the Deep Optimisation algorithm and discuss how DO contributes to the class of MBOAs and also unpick differences between the existing methods. We then evaluate our findings by constructing a synthetic problem that includes complexities of overlapping dependencies, non-pairwise dependencies and higher-order local search. We find that DO is the only algorithm to overcome all problem complexities explored, and we can attribute these differences due to either the model capacity (deep vs shallow) or the method used to inform search using the model. Further, for the first time, we categorically differentiate the performance between the existing MBOAs explored in this thesis. Finally, we explore how DO can be applied to more applied optimisation problems such as the Multi-Dimensional Knapsack problem, the Travelling Salesman Problem and a continuous optimisation problem. We find that DO shows impressive results, especially considering that no domain-specific operator is required to repair solutions (in the case of problems containing infeasible solutions).

The claims of this thesis are:

1. Deep Optimisation can induce and search in deep representations of a solution space. Consequently, DO can find solutions in polynomial time that, for the other MBOAs take exponential time with respect to the number of function evaluations.
2. Using the model to inform higher-order substitutions can find solutions that other methods, such as using the model to inform the generation of a complete solution or using the model to inform a crossover mask during recombination, cannot.
3. Searching in deep representations can exploit problem structures that searching in shallow representations cannot.
4. Overlapping dependencies between higher-order units categorically differentiate the performance between existing MBOAs.

In addition, the contributions of this thesis are:

1. In Chapter 6, DO is used for applied optimisation problems and shows it is capable of learning and exploiting structure in these problems to find fitter solutions. This provides a promising direction for DO, in that it can be applied to a broad range of problems and is not just limited to a particular optimisation problem (such as the one used in Chapter 5).
2. This thesis contributes the class of algorithms called deep optimisation that uses a deep neural network to repeatedly transform the neighbourhood of a solution by learning from a population of fit solutions. The implementation presented in this thesis is the first algorithm within the class of MBOAs to use a deep neural network model and find solutions that other MBOAs cannot.
3. The categorisation of Model-Informed Generation, Model-Informed Crossover and Model-Informed Variation. These categorisations distinguish the different methods used by the MBOAs explored in this thesis to exploit higher-order units learned by a model to inform the search.

1.1 Thesis Overview

The thesis is organised as follows:

Chapter 2: The foundations of this thesis are discussed, and we review the relevant literature regarding the development of evolutionary computing, with specific attention to the use of machine learning methods.

Chapter 3: The functionality of multi-scale search algorithm and estimation of distribution algorithm are compared using the algorithms restart Hopfield network with generative associations algorithm and the Bayesian optimisation algorithm as examples. We show that restart Hopfield network with generative associations algorithm can overcome problem challenges that the Bayesian optimisation algorithm cannot, even though the model used in the Bayesian optimisation algorithm is of higher sophistication than that used by the restart Hopfield network with generative associations algorithm. We then develop the restart Hopfield network with generative associations algorithm to use an autoencoder model, which we call the restart Autoencoder with generative associations algorithm, and demonstrate a comparable performance with restart Hopfield network with generative associations algorithm. We show that restart Autoencoder with generative associations algorithm can represent problem structure that the restart Hopfield network with generative associations algorithm cannot and that synthetic hierarchical problems, previously used as a benchmark in the literature for MBOAs, can be represented by a single-layered autoencoder model.

Chapter 4: Introduces the Deep Optimisation (DO) Algorithm. We review the idea of deep optimisation and relevant literature that allows for inducing a multi-level representation. The DO algorithm is then described in detail. Finally, we compare the functionality of DO with other MBOAs. We discuss the differences between their functionalities and develop hypotheses for problem characteristics that categorically separate the performance between algorithms.

Chapter 5: We first review the relevant literature for constructing and using synthetic benchmark optimisation problems. We develop and detail the theoretical problem construction that contains problem characteristics identified in Chapter 4 into a single problem. We explore the interaction between these characteristics to understand the challenges the problem presents to an optimisation algorithm. Then, we perform a comprehensive scalability study using the theoretical problem. The results show a categorical differentiation between the best performing algorithms for the first time. Further, we show that DO is the only algorithm capable of efficiently overcoming all problem challenges.

Chapter 6: In the last chapter, we apply DO to the Multi-Dimensional Knapsack Problem, Travelling Salesman Problem and Griewank Continuous Optimisation problem and show promising results and future directions to explore.

1.2 Publications from thesis

1. Poster and 2-page paper (Caldwell and Watson, 2017). Title: How to get more from your model: the role of constructive selection in EDAs. Conference: Genetic and Evolutionary Computation Conference. Chapter 3 expands on the work presented at this conference.

2. Archive Paper published on ArXiv (Caldwell et al., 2018). Title: Deep Optimisation: Solving Combinatorial Optimisation Problems using Deep Neural Networks. Chapter 4 and Chapter 6 expand on the work presented in this paper.

3. Conference paper. Accepted and presented at the EvoStar 2021 conference (Caldwell et al., 2021). Title: Deep Optimisation: Multi-scale Evolution by Inducing and Searching in Deep Representations. Chapter 4 and chapter 5 expand on the work presented in this paper.

4. Journal Paper (in submission). Extension of the conference paper submitted to the EvoStar 2021 conference. Title: Deep Optimisation: Multi-scale Evolution by Inducing and Searching in Deep Representations.

of evolutionary computation in comparison to alternative optimisation methods is that it makes few assumptions about the problem structure. This makes the class of algorithms successful in finding solutions across a wide range of problem domains (De Jong, 2006). Therefore, by understanding and emulating the mechanisms that contribute to adaptation, we can further develop the success of evolutionary algorithms. That is not to say understanding the algorithm of evolution can solve all our problems. Wolpert and MacReady put this succinctly with the no free lunch theorem (Wolpert and Macready, 1997). It states that the performance of a search process, averaged over all optimisation problems, will be equal to the performance of random/exhaustive search. The no free lunch theorem is not a limiting factor when designing optimisation methods; rather, it is an important caveat to consider. Generally, we are interested in designing search methods that perform well on problems with particular characteristics. The performance of a search process is therefore determined by evaluating the performance of the search on this distribution of solutions, or as Wolpert describes how well ‘aligned’ a search algorithm is with the distribution of problems (Wolpert, 2013). In the the case of evolutionary computing, how well variability is aligned with the problem structure - the evolvability of the system.

Chapter 2

Foundations

The focus of this thesis is on search methods that are aligned with the Building-Block Hypothesis (Holland et al., 1992a; Goldberg, 1989a). Specifically, algorithms that automatically decompose and exploit regular occurring relationships that improve solution quality by biasing future search. The Building-Block Hypothesis inspired the use of machine learning methods to explicitly capture and represent building-block structures from a distribution of solutions. This thesis discusses the models and methods used to exploit the information contained in the problem structure to inform future search. In this chapter, we review the literature that led to the development of using machine learning models in the context of evolutionary computing. We introduce the algorithms that show best performance in the literature within this class of algorithms (which are subsequently used to compare with Deep Optimisation's performance) and set the foundations from which this thesis extends from.

This chapter provides the main literature review. For readability, Chapters 4 and 5 include additional reviews for deep learning methodology used to induce multi-level representations from a data set and synthetic benchmark problems, based on the Building-Block Hypothesis, used to evaluate the performance of evolutionary algorithms, respectively.

2.1 Decomposing Optimisation Problems

The no-free-lunch theorem states that the performance of a search process averaged over all optimisation problems will be equal to the performance of random/exhaustive search (Wolpert and Macready, 1997). The no-free-lunch theorem is not a limiting factor when designing optimisation methods; instead, it is an important caveat to consider. We are interested in creating search methods that perform well on problems with particular characteristics. The search performance is therefore determined by the performance on the distribution of problems that contain these characteristics. Or, as Wolpert describes, how well ‘aligned’ a search algorithm is with the distribution of problems that the algorithm has been designed to work on (Wolpert, 2013).

Therefore, to search more efficiently than exhaustive exploration in a particular class of problems, the process must contain some bias toward that class of problems. For instance, linear programming exploits the characteristic of linear relationships that allows for solving linear problems in polynomial time (Karmarkar, 1984). For problems that are not represented by mathematical models with particular properties, heuristic methods are a promising alternative. A notable example is the Lin-Kernighan heuristic used for the symmetric travelling salesman problem (Lin and Kernighan, 1973; Applegate et al., 2006). Therefore, by understanding and exploiting information about the problem, the bias of the search can be aligned with the problem characteristics, allowing for an efficient search.

Often, it is not practical or even possible (in the case of black-box optimisation problems) to directly examine the problem structure. In this case, information about the problem’s characteristics can only be inferred by ‘probing’ the problem — information gathered by observing how the quality of a solution changes. Simple but effective methods such as Simulated Annealing (Kirkpatrick et al., 1983) and Tabu-Search (Glover, 1989) rely on using previously explored solutions to bias future search - a memory-based bias. In this thesis, we are interested in the decomposition bias - a popular reductionist technique used for optimisation (Papadimitriou and Steiglitz, 1998). Specifically, the idea of taking a complex high-dimensional function and reducing it to many smaller functions that interact to construct the original function. This idea is intuitive to an engineer. Generally, when an engineer is presented with a challenging optimisation problem, they decompose the global problem into smaller, easier to solve problems at their natural joints. Solutions are found for these easier to solve problems and combined to construct the global solution. The decomposition structure can be separated into multiple scales of organisation, creating a hierarchical decomposition of relationships that build the global function.

2.1.1 Building-Block Problem Structure

Problem structure in an optimisation problem is defined as the relationship between variables that provide some form of regularity and pattern to solutions for a problem. For example, in a travelling salesman problem, it is possible that the optimal route between two regions (travelling from Southampton to London) can be similar, regardless of the routes taken within these two regions. In this thesis, we are particularly interested in problems that demonstrate a modular structure (building-blocks). That is regularities between variables that contribute to the quality of a solution. This regularity can be observed as variable combinations or relationships between variables that frequently contribute to the fitness of a solution (we can refer to these as partial-solutions). This structure, for example, can emerge when there exists a weighting of interactions between variables and the strong interactions favouring the particular variable combinations in comparison to the weaker interactions. These modules of problem structure contribute to the quality of a solution in many contexts (provides quality to the solution relatively independent of the state of the remainder of the solution). As such, the effect of a change to a module is mostly confined to the module.

When the modules do not have a dependency on other modules in the problem structure, the problem is said to be separable or additive. E.g., adding the solution to each module together produces the globally optimal solution. Examples of this type of problem structure are the deceptive trap function Goldberg (1987). More interesting problems arise when dependencies exist between modules. This type of problem structure is called non-separable, E.g., the solution for a module is dependent on the solution to other modules. An example of this type of problem structure is the Modular Constraint Problem Watson et al. (2011b). And finally, a particular dependency between modules that we explore deeply in this thesis is hierarchical structure which is a recursive formation of modular structures. Specifically, a higher-order module is constructed via a dependency between two or more lower-order modules and consequently creating a dependency between modules at the lower level.

In the context of this thesis, learning and exploiting the problem structure refers to using the problem structure to reduce the dimensionality of the search space. For example, in identifying a module (building-block), future search is rescaled to search in the space of modules (combinations of variables) rather than the original solution level. It is assumed that combining lower-order modules found early in the search process that were identified to provide good fitness benefits in many contexts are likely to be a good solution when combined together.

It is not clear if the presence of decomposable structure is due to the problem structure or the search process (Lipson, 2007). For instance, a function constructed from multiple sub-functions may not be the best function possible (the globally optimal solution). Therefore, exploiting low-order sub-functions to construct a high-order function may not yield the relationships present in the global optimum solution. Such can be the case for deceptive (Goldberg, 1987; Whitley, 1991) or random (Heckendorn et al., 1999) problem structure. In these cases, constructing a global solution from low-level partial solutions is unlikely to lead to a global solution. Nonetheless, the problem can still be decomposed at its natural joints and be exploited. However, there is no guarantee that a global solution will be found. We refer to this structure as natural structure - the decomposable structure that is identifiable observing regularities in variable relationships whether these identifiable regularities are globally optimal or not.

Simon (1969) identifies that naturally occurring functional systems – be they biological or human-designed – e.g., multi-cellular organisms, software, engineered solutions and social organisation's, share a nearly-decomposable structure. Specifically, the organisation of the global function is a nesting of lower-order components. The lower-order components are functionally independent entities, i.e., they provide a function that is (to some degree) independent of external functions such as those of other components. Therefore, the internal relationships of a sub-function (within-component relationships) are stronger than the relationships with external sub-functions (between-component relationships) and are subsequently prioritised. The lower-order functions are combined to construct higher-order functions. This is recursively applied to construct the global function containing a hierarchical organisation. Therefore, in the case of nearly decomposable functions, the system relies on representations at multiple scales of complexity and organisation.

Simon (1969) explains that we can induce the natural decomposition of functions by observing their reaction to stimulus. By stimulating (or 'probing') the system, we can observe regularities that occur in the response that provides a signal to the type of decomposition or the features of a complex system (units that show a regular relationship when stimulated). Regularities that occur often can be inferred as contributions to the functionality of the complex system. In the context of optimisation, the stimulus results in a change to the solution, providing a comparison between two or more solutions. Therefore, the response is the difference in the quality of the solution, i.e., did it improve the solution? Thus, in optimisation, by observing regularities that emerge in response to stimulus, i.e., regular occurring relationships that contribute to the quality of a solution, it can be possible to induce the natural decomposition of the problem. The question arises in how to identify, represent, and exploit this information?

The idea of identifying and exploiting regularities in promising solutions relates to the Building-Block hypothesis that is proposed as the explanation for the success of Genetic Algorithms (GAs) (Holland et al., 1992a). GAs use a population of solutions and genetic operators to evolve the population towards higher quality regions in the search space. It is hypothesised that during the evolution of a population, low-order regular occurring building blocks are conserved and recombined during future search (Goldberg, 1989a; Holland et al., 1992a).

A building block is identified by the frequency of a partial solution in the distribution of solutions: the more solutions that contain a partial solution, the more likely it will be exploited during recombination. Thus, the Building-Block Hypothesis provides an efficient mechanism for exploiting information about the problem structure without knowledge of the problem structure 'a priori'. However, the quality of the solutions found will be dependent on the alignment of the Building-Block Hypothesis to the problem structure. Therefore, the performance of algorithms based on the Building-Block Hypothesis are often differentiated by evaluating their scaling performance on synthetic optimisation problems that have clear and controllable decomposable structure (Deb and Goldberg, 1994; Mitchell et al., 1991; Watson et al., 1998; Pelikan and Goldberg, 2001; Watson et al., 2011b), i.e., the performance is measured as the ability to search the solution space using building blocks and not whether the optimisation problem contains building blocks. This enables an understanding of how the algorithm searches the solution space and the types of building block interactions that cause challenging features in the solution space. In Chapter 5 we explore the types of building block characteristics that differentiate the performance between the best performing algorithms and the Deep Optimisation algorithm.

2.2 Genetic Algorithms

Natural evolutionary processes provide inspirational mechanisms for identifying and exploiting the structure of a black-box optimisation problem. Evolution does not have direct access to relationships and functional dependencies in an environment that determine the fitness of an organism – the environment is in that sense a black-box – yet evolution constructs complex solutions. Genetic Algorithms (GAs) are a class of optimisation algorithms that emulate the dynamical process of evolution by natural selection to solve optimisation problems (Holland et al., 1992a). GAs have been extensively applied and are successful across multiple problem domains (De Jong, 2006). GAs use a population of solutions (that maintains solution diversity), genetic operators to exchange material between solutions, and a selection pressure that favours solutions with higher fitness to search the solution space. At each generation, new solutions are generated by recombination of material from prioritised solutions (decided by selection). These solutions replace other solutions of lower fitness to complete a single generation. Multiple generations are performed to evolve the population towards regions of higher fitness.

The adaptive variation, and consequently exploitation, is implicitly controlled by the solution representation, crossover operator, mutation rate, and selection criteria (Eiben and Schippers, 1998; Črepinšek et al., 2013). The operators combine to exploit information in the current state of the population to search in neighbouring regions of a solution. For example, crossover controls the exchange of partial solutions between parents. The change to the value of a variable depends on the differences between the parent solutions (or other factors reflecting variability available in the population at that time). Consequently, the control of exploitation is implicit in the combination of genetic operators (Črepinšek et al., 2013). Solution representations and genetic operators (mutation and crossover) that are mis-aligned with the problem structure can induce additional local optima into an optimisation problem (Rothlauf, 2006; Weise et al., 2012). Therefore, improving the performance of a GA requires understanding the interaction between the representation and genetic operators and how this transfers to overcoming problem challenges (Weise et al., 2012). Some research directions have focused on improving the recombination of solutions, either via improving the representation of a solution — so that genetic operators are more effective on a specific type of problem (Salomon, 1996; Rothlauf, 2006) — or by improving the genetic operators to improve the alignment of search to the problem structure (Potvin, 1996; Herrera et al., 2003; Pavai and Geetha, 2016). A significant area of research has focused on the maintenance of alternative solutions in the population – methods referred to as diversity maintenance – and therefore variability. Crossover and selection act to reduce variation in a population, as material from the parents replaces material in alternative solutions contained in the population. When selection acts at the population level, a generated solution can replace an alternative solution in different regions of the solution space, i.e., replace solutions containing weak or no relationship with the new solution. Solutions that are of low fitness in the current context can nevertheless contain useful information. Replacing this information before the algorithm has exploited it can cause the GA to converge to a sub-optimal solution, even though the information was available during the search. Therefore, population sizing, selection methods, replacement methods, crowding (De Jong, 1975), and niching (Goldberg, 1989b) are techniques developed to overcome this issue.

It has long been hypothesised that the method of exploitation is explained by the GAs implicit ability to transform the neighbourhood of a solution by conserving adapted variable combinations (building-blocks) whilst maintaining variability between them (Goldberg, 1989b; Holland et al., 1992a). Specifically, selection prioritises common features shared in high-quality solutions, and crossover provides a mechanism for recombining building-block material to construct good solutions as combinations of good building blocks. The Building-Block Hypothesis describes a method for variation and selection to act at the group level as if a combination of variables was a selective individual unit. This process is applied recursively, constructing higher-order building blocks and eventually the complete solution. The process aligns with evolutionary thinking, such as the evolution of evolvability (Wagner and Altenberg, 1996; Altenberg et al., 1994) where the emergence of building blocks during evolution provide adaptive variation that was not available originally and the evolutionary transitions in individuality (West et al., 2015) where natural selection acts at multiple scales of organisation.

The idea of exploiting building-block information during recombination requires minimising the disruption to partial solutions during such recombination. For GAs, the ability to exploit building block information is limited to short (low-order) building blocks, where the likelihood of disruption is small (O'Reilly and Oppacher, 1995; Beyer, 1997) unless specific operators that are aligned with the building-block structure are used for recombination (Holland et al., 1992b; Goldberg et al., 1993). For instance, positional biased crossover operators, such as 1-point crossover, allow for complete recombination of large building blocks with tight-linkage (Goldberg et al., 1993; Thierens, 1995; Deb, 2000; Watson, 2002). In this case, building-blocks can be exploited to construct higher-order building-blocks (Thierens and Goldberg, 1993; Thierens, 1995, 1999) and consequently overcome hierarchical building-block problems (Watson et al., 1998; Watson, 2002). However, developing positional unbiased variation operators, such as uniform crossover, cause significant disruption to high-order building blocks and therefore fail to efficiently recombine the partial-solutions (Thierens, 1995; Harik, 1997; Thierens, 1999; Watson, 2002). This leads to using explicit machine learning algorithms to capture the building-block structure and then exploit this information to perform recombination in removing the necessary positional bias in crossover operators to exploit building-blocks.

2.3 Comparing Algorithm Performance

In this thesis, we refer to the performance of an algorithm as the number of function evaluations to find a globally optimal solution. This is because this thesis aims to understand what problem structures the as Model-Building Optimisation Algorithms (MBOAs) investigated in this thesis can and cannot solve and specifically how the design decisions made for the algorithms affect their performance. The approach used to determine this is by evaluating an algorithm's scaling with regard to the number of function evaluations required to find a globally optimal solution. If the algorithm shows polynomial scaling, the algorithm is said to be successful. If the algorithm shows exponential scaling, the algorithm is said to fail.

In addition to the number of function evaluations, there are other measures that one can use to evaluate the performance of an algorithm. However, in this thesis, these performance differences are only discussed briefly where appropriate as they do not add to supporting the claims of the thesis. Specifically, the thesis's claims only refer to algorithms showing polynomial or exponential scaling in the number of function evaluations. Given that no algorithm has exponential time complexity for other processes during optimisation, such as the learning process or search process, then showing an exponential vs polynomial scaling in the number of function evaluations is sufficient to separate the algorithms.

This is not to say other performance measures are not important. Instead, the thesis does not claim one algorithm is better than another when both algorithms show the same scaling behaviour in function evaluations. The only claim made in this thesis is that one algorithm cannot overcome problem structure that others can because we show exponential scaling for one algorithm and polynomial scaling for another algorithm.

Consequently, when referring to the performance of algorithms in this thesis and comparing the performance, unless otherwise stated, one is referring to the number of function evaluations required to find the globally optimal solution.

2.4 Using Machine Learning to Exploit Problem Structure

The challenges of identifying, representing and exploiting building block information lead to the use of centralised models to explicitly capture this information from the population of solutions. Machine learning provides a powerful and automated way of extracting relationships from data. The types of algorithms that use a machine learning model to capture and exploit building-block structures are: Estimation of Distribution Algorithms (Mühlenbein and Paass, 1996), Linkage-learning Algorithms (Harik, 1997; Thierens, 2010), and Multi-scale Search Algorithms (Watson, 2006; Watson et al., 2011b). The methods for identifying promising candidate solutions (selection pressure and model construction), the type of model used to capture this information (model capacity), and the method used to exploit this information to inform future search (model informed search) differ between these algorithms. However, the principal idea of using a machine learning model to explicitly capture relationships that contributed to the quality of previously explored solutions and then subsequently exploit this information to bias future search is consistent. We, therefore, refer to all these algorithms as Model-Building Optimisation Algorithms (MBOAs). In this section, we review these algorithms and detail their functionality. A detailed comparison between the algorithms is provided in Chapter 4.

2.4.1 Estimation of Distribution Algorithms

Estimation of Distribution algorithm's (EDA's) use machine learning methods to model a distribution of promising candidate solutions and then sample the model to generate new solutions (Mühlenbein and Paass, 1996; Hauschild and Pelikan, 2011). EDA's use a probabilistic model to replace the crossover and mutation operators in a genetic algorithm. At each generation, a probabilistic model is constructed that represents the joint probability distribution of the selected for sub-set of solutions (distribution of promising candidate solutions) from the population. Specifically, selection is applied to the group level i.e., the fitness of a solution is relative to others in the population. The model captures emergent variable relationships that contribute to the quality of a solution. The model is then sampled to generate new solutions, which in-turn replace solutions in the population; completing a single generation. Sampling of the model provides a method for exploiting these relationships during future search, via random recombination of these relationship - a process we refer to later as Model-Informed Generation (MIG). The strength of the relationship, or likelihood of conserving the relationships is proportional to the frequency of the relationship in the distribution of promising candidate solutions. The algorithm performs many generations, each time constructing a new model of the distribution of solutions. EDA's have been shown to be capable of overcoming challenges that GA's could not (Harik, 1997; Pelikan and Goldberg, 2001, 2006) and have been successfully applied to a wide range of problem domains (Pelikan and Goldberg, 2003a; Aickelin et al., 2007; Ceberio et al., 2013; Santana et al., 2008). The general Estimation of Distribution algorithm is presented in Algorithm 1.

Algorithm 1: The Estimation of Distribution Algorithm

```
Initialize: Population of Solutions;
while Termination Criteria not met do
    Evaluate the fitness of each solution;
    Select a set of promising solutions from the population;
    Construct a probabilistic model that estimates the distribution of the promising
      solution set;
    for N in Number Solutions to Generate do
        Sample the probabilistic model to generate a solution;
        Replace a solution in the original population with the generated solution;
```

The earliest example of an EDA used simple uni-variate models to represent the joint probability distribution of solutions (Baluja, 1994; Mühlenbein and Paass, 1996; Harik et al., 1999b) and were an important step toward understanding how machine learning models can be used in the framework of genetic algorithms. However, these EDA's were outperformed by GA's on problems that contained epistatic fitness contributions (the fitness contribution depends on if the relationship between two or more variables is satisfied) (Pelikan and Mühlenbein, 1999; Pelikan, 2008). This therefore drove the development of more sophisticated models. At first, bi-variate models were introduced. Bi-variate models form dependencies between variables in the form of a chain (De Bonet et al., 1997), tree structure (Baluja and Davies, 1997) or a forest (multiple trees) (Pelikan and Mühlenbein, 1999). By sampling the model to generate new solutions, the relationship information between a pair of variables is conserved, allowing for an improvement when performing building-block recombination. These methods were successful in outperforming uni-variate models and GA's (De Bonet et al., 1997; Baluja and Davies, 1997; Pelikan and Mühlenbein, 1999).

The most sophisticated models used by EDA's are capable of multi-variate factorisation of the joint-probability distribution. This allows for high-order variable relationships to be captured by the model. However, the cost of constructing the models is more complex and computationally expensive than univariate and bi-variate modals. The Factorised Distribution Algorithm (FDA) (Mühlenbein and Mahnig, 1999) uses a fixed factorised distribution as the model: the structure is not learnt during optimisation, and must instead be provided. Therefore, its use requires prior knowledge of the problem structure. The extended Compact Genetic Algorithm (eCGA) (Harik, 1997) constructs a marginal product model that groups variables into independent sub-sets (variables within a subset are assumed to be dependent). A greedy algorithm combines two groups together that decrease the Minimum Description Length for the distribution model. Initially, each variable is considered an independent group. When no grouping can decrease the Minimum Description Length metric, the model construction is terminated. The probability for each group is calculated from the statistics measured in the promising candidate solution for the group. The probabilities are then sampled to generate new candidate solutions. Therefore, the groupings can represent building-blocks and the random sampling allows for recombination between building-blocks. However, with this model, overlapping building-blocks can not be accurately represented.

The Bayesian Optimisation Algorithm (Pelikan et al., 1999) uses a Bayesian network as the model. A node represents a variable and an edge represents the conditional dependency between variables. The Bayesian network model allows for a variable to be conditioned on the value of multiple parents. The Bayesian Optimisation Algorithm constructs a Bayesian network using a greedy hill-climber and the K2 metric (Cooper and Herskovits, 1992). There is a significantly cost for the model construction algorithm as for each modification made to the Bayesian Matrix (addition, removal or reverse of an edge) the metric needs to be recalculated. BOA reduces the computation cost by only considering edge additions to the network. Never-the-less the time-complexity for constructing the Bayesian network scales exponentially with the maximum number of incoming edges k - a tunable parameter in BOA with higher k allowing for higher-order dependencies to be captured by the model.

Here we detail The Bayesian Optimisation Algorithm's operation as it is used in chapter 3. BOA works by initialising a population of candidate solutions from a uniform distribution of possible solution states. The fitness of the individual solutions is evaluated. Truncation selection is then applied to the population to filter out a set of promising solutions (generally a 50% separation for model-building and the other 50% used for replacement). The promising candidate solutions contain variable combinations that contribute to the quality of the solution. The model is constructed to represent the distribution of promising solutions. The model, is then sampled using forward sampling to generate new solution. The solutions generated replace the sub-set of solutions from the population that have the lowest fitness. This process is repeated until the termination criterion is met. The Bayesian Optimisation Algorithm is capable of learning overlapping dependencies, unlike the extended Compact Genetic Algorithm, and the Bayesian Optimisation Algorithm does not require the prior definition of the model structure, unlike the Factorised Distribution Algorithm (FDA).

2.4.1.1 Hierarchical Bayesian Optimisation Algorithm

The most sophisticated example of an EDA is the Hierarchical Bayesian Optimisation Algorithm (hBOA) (Pelikan et al., 2003; Pelikan and Goldberg, 2006). It has been shown to be effective at solving a wide range of optimisation problems (Pelikan and Goldberg, 2003a; Pelikan, 2010) and is often used as a baseline algorithm when evaluating the performance of new algorithms (Thierens and Bosman, 2013; Goldman and Punch, 2015; Hsu and Yu, 2015). We detail its operation here as it is used throughout this dissertation.

hBOA is an extension of the Bayesian Optimisation Algorithm. The first extension made is the use of a decision trees to represent the Bayesian network model. In BOA, the number of conditional probabilities grows exponentially as the number of interactions between variables increases. hBOA overcomes this by exploiting regularities in conditional probabilities and encoding these using decision trees. This representation allows for the model to include high-order interactions more efficiently. hBOA uses a forest of decision trees to achieve this, with each variable containing a decision tree that represents the variables that condition the probability for the variable.

hBOA maintains a population of candidate solutions and applies binary tournament selection to produce a distribution of promising candidate solutions used for model construction. Like BOA, a greedy algorithm is used to add edges between nodes that improves the performance score. When no edge addition improves the metric, model construction is terminated. Details of the model construction process are provided by Pelikan and Goldberg (2006). Note that the metric used to evaluate the performance of the model for representing the distribution contains an implicit inductive bias towards simpler models. Later, we see that DO is capable of explicitly controlling the parsimony pressure. After model construction, new solutions are generated using forward-sampling. These solutions are introduced back into the population using restricted tournament replacement (Harik et al., 1995). Generally, half the population is used for constructing a model, and the same number of solutions are generated from the model. The number of replacements is determined by restricted tournament replacement. restricted tournament replacement causes competition between the newly generated solution and a similar solution that exists in a sub-population of the entire population, determined by the hamming distance (rather than a random competition). Further, this competition is restricted to sub-populations of a tunable size. An important note is that a new Bayesian model is created with each generation: no information in a Bayesian model from earlier generations is retained.

The Bayesian Optimisation Algorithm and hBOA are both used in this thesis. hBOA is used as the primary algorithm to compare with the performance of the Deep Optimisation algorithm (chapter 5. The Bayesian Optimisation Algorithm is used as for evaluating the difference in exploiting information in the model (chapter 3.

2.4.1.2 Neural Network Models

Neural Network models offer advantages over other machine learning models used by EDA's. They are flexible, high capacity models that can use efficient learning algorithms to approximate a function. They use input, hidden and output nodes connected by a set of weights that are updated by a learning algorithm such as back-propagation. The hidden variables represent relationships in the training data that are used to produce the desired output. The idea of using a NN as the model in an EDA is not novel. Santana

(2017) provides a brief but excellent overview of algorithms that use a NN within the framework of EDA's. In general, EDA's that use a NN have been limited to a single layer network (Probst, 2015; Churchill et al., 2014; Santana, 2017). More notable, is that evaluation of their performance has failed to outperform existing methods (Churchill et al., 2014). Probst (2015) was able to demonstrate that an autoencoder model could use less computational resources than BOA to construct a model, however, the algorithms still required more function evaluations to solve a problem. However, other models (explored later) have also shown to be less computational demanding than the Bayesian Optimisation Algorithm in constructing a model but they also show an improvement in the number of function evaluations and population sizing used (Thierens and Bosman, 2013). Thus we do not include the performance of a neural network based EDA in comparisons performed in this thesis.

It has been hypothesised that performance has not been achieved using a neural network model thus far because of the limitation of the depth of the neural network. A deeper neural network provides additional capacity in comparison to a single-layered (or shallow) network and allows for abstraction of higher-level features from the dataset. However, little research has focused on using a deep neural network for optimisation. Examples of algorithms that use a deep neural network are the Deep Boltzmann Machine Estimation of Distribution Algorithm (DBM-EDA) (Probst and Rothlauf, 2015) and the Deep-Opt Genetic Algorithm (Baluja, 2017). For DBM-EDA, a two-layered Boltzmann machine is used as the model. The model is trained to learn the probability distribution of the selected candidate solutions and then sampled to generate new solutions. In Deep-Opt GA, the model is used to approximate the fitness function of a problem. New solutions are then generated using a technique called network-inversion where the back-propagation algorithm is used to modify the inputs into the network rather than the weights (as done during training). Whilst not technically in line with an EDA framework, as the model captures relationships between input variables and the fitness values (rather than just the input variables), it is never-the-less a rare example of using a deep neural network to improve evolutionary search. In both cases, performance does not outperform existing methods. Further, understanding why and whether a deep network provided a performance improvement is also not presented (i.e., comparison to using a shallow (single-layered) model). Therefore, an important question remains: what type of problem structure can a deep neural network model capture that a single-layered (shallow) neural network cannot? But maybe more importantly, how to use deep neural network model within the framework of evolutionary search?

Missing from the literature is the understanding of why the performance of an EDA using a neural network is outperformed by other, less sophisticated, models. Performance in non-optimisation tasks, such as classification, regression, and generative tasks, has shown that deep models can learn meaningful structure that provides state-of-the-art performance (Krizhevsky et al., 2012; Hinton et al., 2012a; Cho et al., 2014; Silver et al., 2016). The role of the model in EDA's is the same as that used in these other domains - extracting and representing salient features from a dataset. Further, given the relative simplicity of the problem structure in the benchmarks used to evaluate the performance of EDA's, it is expected that a neural network model will be sufficiently capable of learning and representing this structure. Yet, evidence to date suggests that neural network models are not a suitable model space to apply to evolutionary optimisation.

2.4.2 Multi-Scale Search Algorithms

Multi-Scale Search Algorithms are an alternative approach to providing a source of adaptive variation. The main difference between a Multi-Scale Search Algorithm and an EDA is that, for a Multi-Scale Search Algorithm, the selection is applied directly to the solution — the individual level. In an EDA, the selection is applied to the group level i.e., the fitness of a solution is relative to others in the population. In a Multi-Scale Search Algorithm, the improvement of the solution is relative to its neighbours (independent of other solutions previously explored, or contained in the distribution of solutions). In a Multi-Scale Search Algorithm, hill-climbing is used as the primary evolutionary process that applies selection at the individual level. Specifically, a solution is replaced by a neighbouring solution with higher-fitness. However, hill-climbing can easily become trapped at suboptimal solutions due to its inability to explore beyond the neighbourhood of a solution. Consequently, a machine learning model is used to transform the neighbourhood of a solution by informing the type of substitution to make to a solution during hill-climbing. The model is learned using a distribution of solutions that have each been updated using hill-climbing. Consequently, the model will capture regular relationships between variables that emerge in the distribution of solutions. In the next iteration, the model is used to inform the substitutions to make during hill-climbing i.e., to substitute a building-block instead of a single variable. In doing so, this rescales the organisation of the substitutions to a higher-order and allows for hill-climbing to explore solutions that were inaccessible in the previous neighbourhood. New solutions found will contain information about how building-blocks can combine. Therefore, in the next iteration, the model transforms the substitution from single building-blocks to building-block combinations. The process iterates, repeatedly transforming the scale of substitutions that can be made to a solution by capturing relationships that emerge during hill-climbing. Therefore, in a Multi-Scale Search Algorithm, a fitter solution is found by making partial substitutions to a solution and using selection to determine if a substitution improved the individual solution. In an EDA, a fitter solution is found by generating a new complete solution and using selection to determine if the new solution is fitter than the groups average.

The general Multi-Scale Search Algorithm is presented in Algorithm 2. Multi-Scale Search Algorithms can be interpreted as a repeated local search algorithm that adapts

Algorithm 2: The General Multi-Scale Search Algorithm

```
Initialize: Model;
while Termination Criteria not met do
    Initialize: Solutions;
    while Optimising Solution do
        Make partial change to solution, defined by model;
        if Partial Change improves solution quality then
            Accept partial change to solution;
        else
            Reject partial change to solution;
        Update the model using optimised solution;
```

the search neighbourhood according to observed relationships from previous searches (its own dynamical experience). The algorithm can be separated into two cycles, an inner cycle that performs local search and an outer cycle that updates a model that defines a solutions neighbourhood. In the outer loop, the model can either be updated using a population of solutions (population updates - batch learning) (Iclanzan and Dumitrescu, 2007) or after each solution has been optimised (incremental model updates - online learning) (Mills, 2010; Watson et al., 2011b; Cox, 2015). The inner cycle updates and optimizes every solution. Therefore all solutions in the distribution are considered promising candidate solutions due to prior local optimisation (unlike in an EDA) and can be used to update or construct the model. The model captures relationships between variables that are relevant to the solution's quality. These relationships are then exploited by enforcing them at the lower level. This reduces the dimension of the search space and reorganises the neighbourhood of a solution. Thus, at initialisation, the search operator is naive, and allows for an 'unbiased' exploration of the search space, e.g. for a binary solution a single bit substitution is suitable. At this stage, the performance of an Multi-Scale Search Algorithm is equivalent to a restart local search algorithm. As the model learns relationships between variables, this is used to inform the substitution to make during search, i.e., transforming a single bit substitution to a multi-bit substitution (a building-block). In doing so, hill-climbing is able to explore a reorganised neighbourhood where multi-bit substitutions can 'jump' across fitness valleys. A solution is either reinitialised (Watson et al., 2011b) or updated (Iclanzan and Dumitrescu, 2007). The algorithm repeats, continuing local search in the redefined solution neighbourhood. This produces a new distribution of solutions that are locally optimal in the redefined neighbourhood. The model is then updated again to provide a further adaptation to the solution neighborhood. The process iterates, repeatedly adapting the neighbourhood based on information found contained in locally optimal solutions in the previous neighbourhood.

In Multi-Scale Search Algorithm's, search is explicitly performed at the scale of building-blocks, providing a method of combining and evaluating each building-block separately rather than randomly recombining building-blocks (Iclanzan and Dumitrescu, 2007; Watson et al., 2011b; Watson, 2006; Mills and Watson, 2011; Mills et al., 2014). As such, in Multi-Scale Search Algorithm's, selection prioritises the performance of individuals (updates are made explicitly to an individual solution) wherein an EDA selection prioritises the performance of the population (although this is not strictly true for the hierarchical Bayesian Optimisation Algorithm (hBOA)). Prioritising the performance of an individual provides implicit diversity maintenance in a population (Thierens and Bosman, 2013) as a solution is only replaced with a neighbouring solution. In an EDA, a solution can be updated by any generated solution. These differences are further explored and discussed in chapter 4.

Biologically, Multi-Scale Search Algorithm's emulates the process of the evolution of the developmental process. Watson and Szathmáry (2016) connects how evolutionary process operating on an individual can learn from its own experiences to shape future variability. Specifically, a separation of timescales between the effects of development and evolution on an organism facilitates a type of learning that can exploit information from previous experiences and bias future exploration (Watson et al., 2014). Algorithmically, development is a process that applies variation and selection to find a good solution by hill-climbing. The evolutionary process adapts the developmental process, specifically the variation, based on information provided by the developmental process, i.e., the solution found.

The Building-Block Hill-Climbing algorithm (Iclanzan and Dumitrescu, 2007) is one of the first examples of a Multi-Scale Search Algorithm. It performs local search on a distribution of solutions that are locally-optimal before then using these solutions to construct a model. The model represents a mapping from building-block to variable i.e., which variable belongs to which building-block. The configuration of a building-block (the values of the variables in a building-block) are extracted from the distribution of solutions. Hill-climbing in the space of building-blocks is then performed on each solution in the population by substituting in a building-block into a solution and evaluating the change in solution quality for each possible configuration of that building-block. The configuration that provides the greatest increase in a solutions quality is then kept. This is repeated on all solutions before a new model is then constructed. The Building-Block Hill-Climbing algorithm provides an efficiency improvement compared to an EDA (Iclanzan and Dumitrescu, 2007). The model used by Building-Block Hill-Climbing algorithm constructs a building-block representation by making discrete and explicit joins between variables such that a higher-order substitution is a discrete representation of the lower-level variables. Mills et al. (2014) described this approach as making hard-joints between variables. Specifically, the variables are either connected or not connected.

An alternative approach, developed by Watson and colleagues (Watson et al., 2011b;

Cox, 2015; Cox and Watson, 2014b; Mills, 2010; Mills et al., 2014) constructs building-blocks using soft-joints interpretation. Specifically, a building-block is constructed probabilistically, conditioned on the frequency of the relationship between variables observed in previous successful solutions (Watson et al., 2011b; Mills et al., 2014). In this case, the adaptation to a solutions neighbourhood is performed continuously: a solution is optimised and then used to update the model before a new solution is optimised (on-line learning). The likelihood of a relationship being exploited to update a solution increases as the strength of a relationship captured by the model increases. This, therefore, removes the need for a population to update the model, unlike in Building-Block Hill-Climbing algorithm. Further, the differences between the relationship strengths allow for representing difference scales of building-blocks, which is further explored in chapter 3. Consistent with the results provided by the Building-Block Hill-Climbing algorithm, results show an efficiency improvement when compared to an EDA (Cox and Watson, 2014b; Mills et al., 2014). Indeed Mills et al. (2014) analytically shows that a Multi-Scale Search Algorithm can solve optimisation problems that an EDA cannot. This thesis empirically supports this result in chapter 3 by conducting experiments using a synthetic optimisation problem. Finally, Multi-Scale Search Algorithms fit naturally with the idea of representing multi-level representation of building-blocks. However, this representation is captured implicitly in the model, as differences in the connection strengths. Deep neural network models provide a model space capable of representing multi-levels of representation. Thus, we hypothesis that deep neural networks are a natural model space within the algorithm of Multi-Scale Search Algorithm.

In this thesis, we use the restart Hopfield Network with Generative associations algorithm (rHN-G) (Watson et al., 2011b) as an example of a Multi-Scale Search Algorithm due to its efficient learning and exploitation method. Further, the restart Hopfield Network with Generative associations algorithm forms the basis from which Deep Optimisation is developed from. For readability, the details of the restart Hopfield Network with Generative associations algorithm, summarised from Watson et al. (2011b), are provided in chapter 3 where the restart Hopfield Network with Generative associations algorithm is developed to use a single-layered autoencoder as the model (replacing the correlation matrix).

Other methods based on local search that use information about the solutions found thus far are Iterated Local search (ILS), Tabu Search (TS) (Glover, 1989) and Simulated Annealing (Kirkpatrick et al., 1983). However, these methods rely on memorisation of solutions found either the best solution found so far, or which solutions have been already explored. Multi-Scale Search Algorithms differ by using machine learning models to learn and represent the relationships between variables that contribute to the quality of a solution. Also, Multi-Scale Search Algorithms share similarities with the well-known Variable Neighbourhood Search (VNS) method (Hansen et al., 2010). VNS works by performing a local search in a fixed set of neighbourhoods that are manually defined prior to searching in them. However, in a Multi-Scale Search Algorithm, the idea of searching in a new neighbourhood is shared with VNS, but a Multi-Scale Search Algorithm instead adaptively reorganises a solution neighbourhood by capturing relationships from a distribution of locally optimal solutions found in the current neighbourhood. The model is used to induce a new and intelligent space to search in based on the information of the current or previous neighbourhood spaces.

2.4.2.1 Local Search for Estimation of Distribution Algorithms

Local search has been used to initialise a population of solutions to improve the performance of an EDA (Pelikan and Goldberg, 2006; Bosman and Thierens, 2011). In doing so, it has been shown to significantly reduce the population size and also the number of function evaluations required for an EDA to find the global optimum solution (Bosman and Thierens, 2011). The performance improvement has been attributed to local search providing a clearer signal for the relationships between variables that improve the quality of a solution (Radetic and Pelikan, 2010). This is consistent with the results observed when comparing Multi-Scale Search Algorithm's and to an EDA. In this case, the performance improvement is attributed to the efficiency of local search. However, the use of local search is not consistent with the process of an EDA, unlike the case for Multi-Scale Search Algorithm's: selection in an EDA is used to move the average fitness of the population in the direction of greater fitness whereas local search moves the fitness of an individual solution in the direction of greater fitness (selection does not depend on the fitness of other solutions in the population). Subsequently, an EDA utilising a local search to initialise the population is often referred to as a hybrid method (Hauschild and Pelikan, 2011; Hsu and Yu, 2015).

The performance improvement identified by using local search inspired the idea of incorporating local search in a more consistent way. Sastry and Goldberg (2004) exploited the linkage information contained within the probabilistic model to adapt the mutation operator applied to a solution. Lima et al. (2006) exploited relationships in the Bayesian network of BOA to adapt the search operator for local search, searching in a substructure neighbourhood and subsequently improving the performance of BOA. Therefore, in an EDA, the efficiency of using the model to inform partial substitutions to a solution has been identified.

2.4.3 Linkage Learning

In the algorithms explored thus far, the models are used to capture the relationships between the variables and the assignments made to a variable. A simpler modelling method is to ignore the value of a variable and only model variable dependencies, referred to as the linkage information. Constructing and learning a good probabilistic model of a distribution of solutions requires identifying the linkage information - the structure of a probabilistic model (Harik et al., 1999a). Thus linkage-learning only refers to which variables are involved in a building-block and not the solution or assignment of a building-block. This can be advantageous for high-order relationships as learning the conditional probabilities can become intractable in the number of parameters required.

The earliest example of linkage learning only algorithms mGA (Goldberg et al., 1989), GEMGA (Kargupta, 1996) and LLGA (Harik, 1997). These methods are based on the idea of reordering the representation of a solution allowing positionally biased recombination operators to perform effectively. By doing so, these algorithms are capable of optimising building-block optimisation problems containing random structure. More recently, and considered state-of-the-art with this class (including a comparison with a Multi-Scale Search Algorithm and an EDA), are algorithms that construct a model of the linkage information to provide crossover masks between solutions (we refer to this type of method later as Model-Informed Crossover (MIC)). These algorithms are: the Linkage-Tree Genetic Algorithm (LTGA) (Thierens, 2010), Parameter-less Population Pyramid (P3) (Goldman and Punch, 2014), and the Dependency Structure Matrix Genetic Algorithm (DSMGA-II) (Hsu and Yu, 2015). We detail these algorithm here as they are used throughout this thesis. A key difference compared to a Multi-Scale Search Algorithm is that the model is used to only inform which variables to exchange between two solutions. The model is not used to inform the substitution (the values of variable substitution as well) to make to a solution. Here, the model only represents the structure of dependencies between variables, not the values assigned to variables. Values are constructed from a random solution drawn from the population. As such, the variation applied to a solution is dependent on the population and linkage-set.

2.4.3.1 LTGA

The Linkage Tree Genetic Algorithm (LTGA) (Thierens, 2010; Thierens and Bosman, 2013) uses an incremental tree linkage-set as the model. LTGA maintains a population of candidate solutions, and selection is used to filter out which solutions to use for constructing the model, as in EDAs. In LTGA, binary tournament selection is used. The incremental-linkage set is generated by first constructing a dependency structure matrix (DSM). The dependency structure matrix represents the pairwise dependencies between variables. The strength of the dependencies represents the measurement of mutual information between the two variables. That is, by observing one of the variables, how much information does this provide about the other variable. The matrix of pairwise dependencies, therefore, represents the partial-solution (building-block) structure, namely which variables should be exchanged together. Agglomerate hierarchical clustering is used to construct a hierarchical tree linkage-set. This clustering technique recursively combines clusters based on their dependency strength until only a single cluster remains.

Each variable is initially considered a separate cluster. After each clustering step, the dependency structure matrix is updated to include the new clustering. The outcome is a tree data-structure of linkage-sets, with each set representing a compression of lower-order linkage-sets.

New solutions are generated by optimal-mixing: as a generalised analogue of crossover in sexual recombination, the constructed linkage-set determines which variables to exchange between solutions (Thierens and Bosman, 2011). Optimal mixing updates an individual solution from the population using the linkage-sets to inform crossover. A random solution from the population is drawn for each crossover operation. A child solution replaces the parent solution if it has greater fitness, i.e., only beneficial exchanges are kept. All linkage sets are used before updating another solution. The linkage-sets are often traversed bottom-up (exchanging smallest groups first). This process is similar to that of a Multi-Scale Search Algorithm, where a solution is repeatedly updated and selection prioritising the fitness of an individual, rather than the group. This is applied to all solutions in a population to complete a generation. Selection is then reapplied to the population and a new model is constructed using the new distribution of solutions.

2.4.3.2 DSMGA-II

The Dependency Structure Matrix Genetic Algorithm Version 2 (DSMGA-2) (Hsu and Yu, 2015) uses a incremental graph linkage-set as the model. DSMGA-2 maintains a population of candidate solutions. The model is constructed from the distribution of solutions selected using binary tournament selection. New solutions are generated restricted-mixing and back-mixing. These methods are an extension of the ideas of optimal mixing. However, the methods of exchange information between solutions remain the same; the linkage-set is used to determine which variables exchange states between solutions. The subtle difference comes from deciding which solutions to use for the exchange. Like LTGA, the linkage set is constructed using a dependency structure matrix. However, instead of using agglomerate clustering, DSMGA-II constructs a linkage set by searching for a specific sub-graph called the approximation maximum-weight connected sub-graph. The linkage set is, therefore, a graph structure: it is possible for a cluster to have multiple parents. However, despite these differences, results fail to demonstrate a categorical difference between DSMGA-II and LTGA.

2.4.3.3 P3

The Parameter-less Population Pyramid (P3) (Goldman and Punch, 2014) uses multiple incremental tree linkage-sets as the model. P3 maintains multiple populations arranged in a hierarchy. Each level of the hierarchy can be summarised as an LTGA instance, and thus each population has its own linkage-tree model that exploits information contained only at the corresponding population level. What differs significantly from the other methods is how the populations are managed. A solution is generated one at a time. A local search is applied to the solution to provide a solution containing variable combinations that contribute to the solution quality. This solution is then added to the lowest level population that does not already include this solution. When a new solution is added to a population, the linkage-tree model for that population, is reconstructed. The solution is then update via optimal mixing using the population and model only at the current level in the pyramid. If the solution is improved, the solution is added to the population at the next level in the hierarchy. If no improvement is found, then the solution is left in the population, and the algorithm restarts with a new solution. The significant advantage of P3 is that it requires no tuning of the population size. However, the algorithm exploits a type of incremental learning - an online learning method - to update the model, which can become costly as many models need to be reconstructed at each iteration.

LTGA, P3 and DSMGA-II have all outperformed hBOA on numerous synthetic problems (Goldman and Punch, 2015; Thierens and Bosman, 2013; Hsu and Yu, 2015). Further, the modelling building process used by LTGA and P3 has a lower computational cost than hBOA (which uses a more complex model) and yet show equal performance on numerous synthetic problems when comparing the number of function evaluations required to find a globally optimal solution. Interestingly, results fail to show a categorical difference between these three algorithms. Specifically what one algorithm can do and another algorithm cannot in the formal sense of polynomial vs exponential scaling. We discuss these differences in more detail in chapter 4 and find problem structure in chapter 5 that does provide a categorical differentiation between what one algorithm can do that another algorithm cannot.

2.5 Alternative Machine Learning Methods

In this thesis, we concentrate on how machine learning, and specifically deep neural networks, can provide additional functionality to optimisation in the framework of MBOAs. In other classes of optimisation methods, machine learning is also being used to improve the optimisation performance. However, what the model is being used for and how the information is being used to improve search differs from MBOAs, and therefore also differs from the DO algorithm presented in this thesis. Of particular note is the Learnable Evolution Model (LEM) Michalski (2000) that shares the concepts of learning from a population of solutions and then using this information to guide the search for new solutions. The key difference between DO, and the MBOAs studied in this thesis is that LEM learns relationships by comparing the difference between fit and unfit individuals, whereas for the MBOAs studied in this thesis, relationships are induced from fit individuals only (which is more aligned with evolutionary processes). LEM also generates complete solutions from the learnt model. This is in line with EDAs and the methods termed Model-Informed Generation in this thesis. It is found that Model-Informed Generation fails for some problem structures in this thesis.

Other alternative methods that are less linked to the algorithms studied in this this, but yet incorporate machine learning models include: learning a heuristic for a set of problem instances (Zhang and Dietterich, 2000; Khalil et al., 2017; Bello et al., 2016); using a surrogate model to approximate the fitness function (Queipo et al., 2005; Vu et al., 2017); adapting the learning function to bias future search (Hopfield and Tank, 1985; Boyan and Moore, 2000); embedding a machine learning model within the model of a combinatorial problem (Lombardi et al., 2017). The use of deep reinforcement learning algorithms for combinatorial optimisation is a popular approach (Bengio et al., 2020; Mazyavkina et al., 2020). Deep reinforcement learning is used to learn a policy that performs an action on a given state to improve the solution. This policy can then be used on multiple instances from the same problem class. Unlike DO, and MBOAs in general, these methods don't use the model to re-code the neighbourhood of a search space.

Alternative methods also include using machine learning outside the optimisation process, for example, using a machine learning method to select a suitable solver for a problem instance (Volpato and Song, 2019). For these types of methods, the machine learning method does not improve/adapt the performance of the optimisation process. Instead, it decides which solver is best suited for the problem. Although outside the scope of this thesis, we include it here to signify the broad range of how machine learning can be applied to improve optimisation.

In this thesis, we focus on algorithms that use machine learning methods to improve the process of recombination in evolutionary algorithms.

2.6 Summary

There exist multiple algorithms that use centralised models to capture and exploit building-blocks. At a detailed level of description, there appears significant differences between the algorithms. However, as we discuss further in chapter 4, the principal idea of using the model to adapt the variation of future search is consistent in all algorithms. Therefore, the algorithms are comparable. We, therefore, class these algorithms as Model-Building Optimisation Algorithm's (MBOA's). However, results in the literature that evaluate the performance between the algorithms fail to show a categorical differentiation of what one algorithm can do that another cannot. Therefore, there is a lack of understanding regarding how the model affects the performance of the algorithm. In this thesis, we aim to provide this understanding by exploring the differences in model construction, model capacity, and model exploitation. Further, the main topic of interest for this thesis is the limited work that includes a deep neural network. Machine learning models have been used to improve the performance of evolutionary computing. However, competitive results are not achieved using what are considered state-of-the-art machine learning models. Specifically, it has not been shown that using sophisticated machine learning models, such as a Bayesian network or neural network model, can solve problems that methods using a simpler model, such as a linkage tree, cannot. However, in other domains, it has been shown that these more sophisticated models can produce state-of-the-art results, generally due to their ability to learn and represent more complex relationships. It is, therefore, interesting that this same behaviour has not been observed for this class of algorithms. As a result, this thesis explores this deeper by discussing what the model is tasked with learning during the optimisation process (Chapter 4) and then developing a synthetic problem containing complex relationships that a model must learn to solve the problem. The performance of each MBOA is evaluated using this problem in Chapter 5. The empirical results show that there do exist problems that contain complexity that is sufficient to clearly differentiate the performance between methods that use complex and simple machine learning models (Chapter 5). The Building-Block Hypothesis has provided a direction for development of these algorithms. So much so, evaluating the performance of an algorithm is generally performed on optimisation problems containing explicit building-block structures. However, the Building-Block Hypothesis describes a method of constructing higher-order building-blocks from low-order building-blocks, i.e., a multi-level representation of building-block structures. Deep neural networks are known for their ability to extract and represent multi-level representation, with each layer extracting higher-order features from the layer below, i.e., higher-level features are constructed from the low-layer features in the layer below. Therefore, deep neural networks provide a natural model space to capture multi-level building block structures. However, using a deep neural network in the framework of a MBOA has not been successfully achieved. As such, in this thesis, we introduce the Deep optimization (DO) algorithm that uses a deep neural network as the model within the framework of an MBOA. We hypothesise that competitive and outperforming results have not been achieved using a deep neural network due to the way information is being exploited from the model, specifically a Multi-Scale Search Algorithm approach. As such in the next chapter, we explore the differences between a Multi-Scale Search Algorithm and an EDA more closely and explore how a neural network model can be used in a Multi-Scale Search Algorithm.

Therefore, the contribution that this thesis makes to the current literature is the development of an evolutionary algorithm that uses a deep neural network as the model, which we name ‘Deep Optimisation’. The class of Multi-Scale Search Algorithm’s will be used as the foundation for DO as the algorithm provides a mechanism that naturally fits with a deep neural network model space (a multi-level search process) that is also connected with evolutionary thinking, such as the evolution of development and the evolutionary transitions in individuality. Further, it is evident from the literature review that the understanding of what one algorithm can do and what another cannot is limited. Therefore, in this thesis, we spend considerable effort providing an understanding of what DO adds to a community that already has multiple algorithms that all claim to be the best performing. Further, in developing DO, we generate a new direction for research around the question of how a multi-level representation affects the adaptive search of an evolutionary algorithm.

Chapter 3

Inducing and Searching in a Compressed Representation of a Solution

This thesis aims to understand how a deep neural network model can be used within evolution computing. In this chapter, we examine how the information learned by a neural network model can be exploited to inform adaptive search. As identified in Chapter 2, using a neural network model to capture the joint probability distribution of solutions and then sampling the model to generate new solutions failed to outperform other EDA's with respect to the number of function evaluations (Churchill et al., 2014; Probst, 2015). Further, the best results have been achieved using algorithms using models with lower capacity but with a different method for exploiting information from the model to further search (Thierens, 2010; Thierens and Bosman, 2011). Specifically, using the information in the model to inform a crossover mask, which in turn allows for multi-variable substitutions to be performed. Consequently, new solutions are found by making partial substitutions to existing solutions rather than replacing solutions with solutions generated by sampling the model. In this chapter, we explore the performance difference between Model-Informed Generation and Model-Informed Variation by comparing the Bayesian Optimisation Algorithm and the Restart Hopfield Network with Generative Associations algorithm (rHN-G). We find that Model-Informed Variation can find solutions to a non-additive modular problem that Model-Informed Generation cannot.

We then explore how a neural network model can be used to perform Model-Informed Variation. Specifically, we replace the pairwise correlation matrix used by rHN-G with a single-layered autoencoder model, creating the Restart Autoencoder with Generative Associations (rA-G) algorithm. We find that rA-G successfully uses the autoencoder model to represent modular structure and exploit this information using Model-Informed

Variation. Importantly, we show that the capability of rA-G (using an autoencoder model) is similar to rHN-G (using a Hopfield network model) on synthetic optimisation problems. We then perform preliminary investigations into how the autoencoder model enhances rA-G in comparison to rHN-G. We first study the performance of rA-G on a synthetic problem containing a hierarchical organisation. We find that a shallow (single-layer) neural network model is sufficient to overcome the hierarchical problem structure and conclude that this problem structure alone is insufficient to require a multi-level representation to capture and exploit the problem structure efficiently. We then investigate the types of problem characteristics a correlation matrix cannot represent, but an autoencoder model can. We find that a problem structure inspired by the 424 encoding problem (Ackley et al., 1985) is a suitable example to demonstrate the types of relationships that distinguish the capability of rA-G from rHN-G. However, we find that rA-G fails to exploit this information using a method in line with rHN-G and requires development. Thus, this chapter provides sufficient motivation to develop the Deep Optimisation algorithm, detailed in the following chapter (chapter 4). Additionally, this chapter identifies preliminary problem characteristics that are challenging for some algorithms, which we further explore in chapter 5).

3.1 Searching and Generating using the Model

In suitable domains, Model-Building Optimisation Algorithms (MBOAs) can learn how to decompose an optimisation problem into nearly separable subproblems (modules) without *a priori* knowledge of the underlying problem structure. Identifying such modularity aims to exploit the familiar idea of separating a problem into smaller, simpler sub-problems, solving these sub-problems and then re-combining their solutions to solve the original problem. This forms the basis of the building block hypothesis where strongly correlated variables in the solution space are linked during a future search, referred to as a building block (Holland et al., 1992a; Goldberg, 1989b). The idea is that building blocks are exchanged and combined, maintaining the linkage information at the lower level (within modules), to search for solutions to constraints imposed at the higher level (between modules). In this chapter, we are interested in the two different methods of using building blocks to bias future search. That is Model-Informed Generation and Model-Informed Variation. We do not include Model-Informed Crossover in this chapter due to its similarity to Model-Informed Variation. Specifically, for Model-Informed Variation, selection acts on an individual solution that has been updated by making a partial substitution. The mechanism of making this partial substitution differs and is discussed further in chapter 4.

Model-Informed Generation is the more familiar method used by Estimation of Distribution Algorithm's (EDA's). These algorithms build explicit probabilistic models of fit individuals in a population, selected by comparing the quality of a solution with other solutions in the population. The model is then sampled to generate new candidate solutions that may replace a solution in the population. Concretely, the algorithms use the learned model to bias the sampling of solutions. In addition, there is no apparent connection between Model-Informed Generation and biological processes.

Model-Informed Variation, on the other hand, is an idea developed in recent work by Watson and colleagues in the development of Multi-Scale Search Algorithms (Watson et al., 2011b; Mills et al., 2014; Cox and Watson, 2014a). The idea has connections to new perspectives in biological evolution. Specifically, the notion of 'internal selection', a key idea in the extended evolutionary synthesis (Pigliucci and Müller, 2010; Laland et al., 2015), or 'developmental selection' (Snell-Rood, 2012) recognising the observation that the process of development is not a deterministic transformation of genotype into phenotype but rather has elements of trial and error and feedback. For example, the wiring of synapses or blood vessels' paths involves internal context-sensitive feedback that adapts the construction process to the environmental conditions. The developmental process biases the variability of a phenotype, allowing selection to operate at a lower-dimensional representation of the original representation. Thus the action of selection is not to filter good phenotypes from bad, nor is the action of development to decode a genotype. The two processes are connected.

Model-Informed Variation emulates developmental selection. Specifically, random variation to an individual unit is constrained by the associative relationships between other variables captured by the model. Thus, a change to an individual unit causes a simultaneous change to other units that have a strong associative relationship with the unit being changed. Consequently, variation is transformed from a change to an individual unit (when no relationships exist between units) to a simultaneous change to multiple units (when relationships exist between units). Selection then provides immediate feedback to determine whether this change is kept. The process is repeated to update a solution to fit the environment, resembling a hill-climbing process. The relationships between variables are captured by the evolution of the developmental process. Specifically, the developmental process that biases variability can itself be subject to natural selection (Altenberg, 1995; Watson et al., 2016) and consequently facilitate a learning mechanism from previous experience (Watson and Szathmáry, 2016; Watson et al., 2016). However, in the models explored in this thesis, this connection is simplified by using explicit learning models to emulate the evolution of the developmental bias.

The first section aims to show that using the model to inform variation (or equivalently, adapting the neighbourhood of a solution) produces a significant efficiency gain compared to using the model to inform the generation of solutions. We evaluate the performance difference of the Bayesian Optimisation Algorithm (an example of Model-Informed Generation) and the restart Hopfield Network with Generative Associations algorithm (rHN-G) (an example of Model-Informed Variation) in solving an idealised nearly-decomposable constraint optimisation problem: the Modular Constraint problem (Watson et al., 2011b). Learning the structure of this problem is straightforward; if an algorithm exploits this structure correctly, then the algorithm can find the globally optimal solution in polynomial time (Mills et al., 2014). However, methods that sample the model can take up to exponential time. Mills et al. (2014) shows this analytically, and in this section, we provide empirical evidence to support the analytical result. This section does not claim that rHN-G is a better Model-Building Optimisation Algorithm (MBOA) than the Bayesian Optimisation Algorithm (BOA); the Bayesian Optimisation Algorithm is a significantly more robust and general method than rHN-G. Additionally, the Bayesian Optimisation Algorithm's model is of greater capacity, capable of capturing higher-order relationships than rHN-G could not. Rather, and more specifically, we aim to assess the performance of Model-Informed Variation in contrast to Model-Informed Generation. The modular structure of the Modulear Constraint problem is easy to represent for both algorithms, thus performances differences are due to how the model informs search. We show that rHN-G is capable of using the model to find globally optimal solutions in polynomial time that takes the Bayesian Optimisation Algorithm exponential time, consequently demonstrating the value of Model-Informed Variation. We verify this further by removing all the model building complexity from the algorithms (by giving them identical models of the modularity 'for free') and evaluating their ability to exploit the model's information to adapt the search.

3.1.1 A Constraint Optimisation Problem with Simple Modularity

The Modular Constraint problem (Watson et al., 2011b) is used to evaluate the algorithms' ability to search in the space of building blocks. The Modular Constraint problem is an example of a simple 'nearly separable' modular problem with one level of hierarchy. It is analogous to the natural energy minimisation of a dynamical system (Watson et al., 2011a).

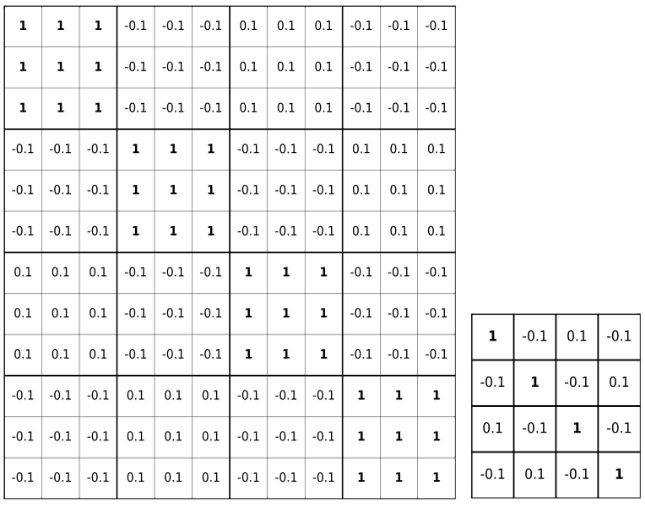
The solution state to the problem is $S = \{S_1, \ldots, S_i, \ldots, S_N\}$, where $S_i \in \{-1, 1\}$ and N is the size of the problem. An energy function encodes the problem as a set of constraints between variables S_i and S_j, ω_{ij}. The fitness of a solution state is given by:

$$F(S) = \sum_{ij}^{N} \omega_{ij} S_i S_j \tag{3.1}$$

Modularity is incorporated into the constraint matrix by selecting stronger weights for within module connections and weaker weights for between module connections. Figure 3.1.a presents an example of a constraint matrix imposed on the solution state for size $N = 12$. It contains $n = 4$ modules, each containing $k = 3$ variable, with symmetric connectivity. Given that the between module constraints are small in contrast to the within module constraints creates local optima that correspond to $S = \{S^k, -S^k\}^n$.

For example, the state configuration $S = (1^3, 1^3, 1^3, 1^3)$ is a local optimum on the fitness landscape for the Modular Constraint problem in Fig. 3.1. The global optima would be $S = (1^3, -1^3, 1^3, -1^3)$ or its complement.

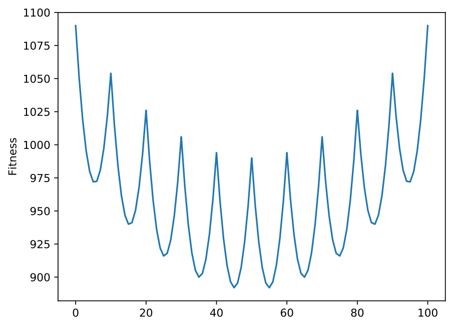
Figure 3.1.b illustrates a dimensional compression of the problem, representing the modules as individual units. Thus, at the scale of modules, the problem contains no higher-order structure. Therefore, it is trivial for algorithms to find a globally optimal solution if it can search in the compressed representation. Figure 3.2 illustrates the fitness cross-section between the two global optima relative to the original solution representation (Figure 3.2(a)) and at the module representation level (Figure 3.2(b)). Single variable substitution at the solution representation level will easily become trapped at locally optimal solutions due to the large (stronger) within-module constraints. Attempts to satisfy between-module constraints will violate the within-module constraints, reducing the solution's fitness and is therefore rejected by a hill-climber. Given that the number of locally optimal solutions increases exponentially with respect to the number of modules, a restart local search algorithm operating in the original solution space will take exponential time to find a globally optimal solution, i.e., requires exhaustive (random) search at the scale of modules. In contrast, at the representation level of modules, the fitness landscape is smooth (see Figure 3.2(b)). This is because the within module constraints remain satisfied when one simultaneously substitutes all variables within a module - thus the fitness difference observed is due to constraints only between modules. Finding a globally optimal solution is, therefore, straightforward when searching in the neighbourhood of modules. Therefore, if an algorithm can represent the modules and search in the space of modules, the global solution can be easy to find. In contrast, without exploiting the modular structure, the Modular Constraint problem is challenging to solve.



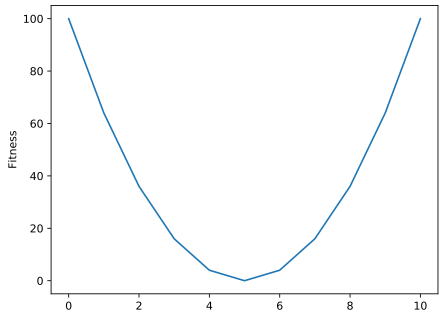
1	**1**	**1**	-0.1	-0.1	-0.1	0.1	0.1	0.1	-0.1	-0.1	-0.1
1	**1**	**1**	-0.1	-0.1	-0.1	0.1	0.1	0.1	-0.1	-0.1	-0.1
1	**1**	**1**	-0.1	-0.1	-0.1	0.1	0.1	0.1	-0.1	-0.1	-0.1
-0.1	-0.1	-0.1	**1**	**1**	**1**	-0.1	-0.1	-0.1	0.1	0.1	0.1
-0.1	-0.1	-0.1	**1**	**1**	**1**	-0.1	-0.1	-0.1	0.1	0.1	0.1
-0.1	-0.1	-0.1	**1**	**1**	**1**	-0.1	-0.1	-0.1	0.1	0.1	0.1
0.1	0.1	0.1	-0.1	-0.1	-0.1	**1**	**1**	**1**	-0.1	-0.1	-0.1
0.1	0.1	0.1	-0.1	-0.1	-0.1	**1**	**1**	**1**	-0.1	-0.1	-0.1
0.1	0.1	0.1	-0.1	-0.1	-0.1	**1**	**1**	**1**	-0.1	-0.1	-0.1
-0.1	-0.1	-0.1	0.1	0.1	0.1	-0.1	-0.1	-0.1	**1**	**1**	**1**
-0.1	-0.1	-0.1	0.1	0.1	0.1	-0.1	-0.1	-0.1	**1**	**1**	**1**
-0.1	-0.1	-0.1	0.1	0.1	0.1	-0.1	-0.1	-0.1	**1**	**1**	**1**

1	-0.1	0.1	-0.1
-0.1	**1**	-0.1	0.1
0.1	-0.1	**1**	-0.1
-0.1	0.1	-0.1	**1**

FIGURE 3.1: a) An example of the modular constraint problem b) building-block scale representation of the same problem (Watson et al., 2011b)



(a) Fitness cross-section at the scale of solution variables.



(b) Fitness cross-section at the scale of modules.

FIGURE 3.2: Fitness landscape cross-section taken between the two global optima in the Modular Constraint problem. The y-axis represents the fitness of a solution. Each step along the x-axis represents a change to unit x_i from 1 to -1. The extremes of the x-axis represent the complementary global optima. The within module constraints create many local optima. The between module constraints create a relatively weak fitness differential between the modules. At the scale of modules, the fitness landscape is convex, and consequently finding a global optimum is easy when searching at the representation scale of modules

For all experiments, the within-module constraint strength is set to 1, and the between-module constraint strength (p) is set to 1×10^{-6} (this is a known value that creates an Modular Constraint problem that is pathologically difficult for a hill-climber (Mills, 2010)). The inter-module weights between different modules vary only by their sign, acting to attract (same sign) or repel (opposite sign) each other. These are specifically chosen to produce a global optimum with alternating module solutions, e.g. $S = (1^k, -1^k, 1^k, -1^k, \dots)$ or its complement. The extreme imbalance between the within module and the between module weights means that the basins of attraction for each local optimum in the high-dimensional problem are almost equal and so reveal almost no between module correlation. A single run with a hill-climber will not be significantly more likely to find a global optimum than any other local optimum. However, in principle, an algorithm that can learn and properly exploit knowledge of the modular structure has the potential to hill-climb in the space of module combinations, following the relatively weak between-module fitness gradients to locate a global optimum. Note, the linkage information could also be shuffled for this problem to make it more challenging for some algorithms that do not use a model (e.g., Genetic Algorithms using crossover). However, as the models learn the linkage information, shuffled linkage is not a challenge for the algorithms considered in this experiment and would not affect the performance outcome. Consequently, the linkage is not shuffled during the experiments.

3.1.2 Restart Hopfield Network with Generative Associations (rHN-G)

The restart Hopfield Network with Generative associations (rHN-G) algorithm is used as the example of Model-Informed Variation (Watson et al., 2011b). rHN-G is a restart local search algorithm that adaptively modifies a solution's neighbourhood by learning relationships in solutions previously found. Learning is provided by a correlation matrix that is updated using Hebb's rule (Hebbian learning) applied to solutions found at the termination of the local search process - thus locally optimal in the neighbourhood defined by the current model. The model, therefore, learns associative relationships between variables that contribute to the quality of a solution. The model is then used to constrain variability such that the associated relations are maintained during future variation. Thus, the model restructures the neighbourhood of a solution from variability in the original problem variables to variability in multi-variable combinations (modules). Consequently, the model transforms local search to a lower-dimensional representation of the solution space. Note, rHN-G forms the basis of the Deep Optimisation. As such, we review the rHN-G algorithm that is detailed in Watson et al. (2011b):

rHN-G can be described as an algorithm containing an inner-loop and outer-loop. The computation performed by the inner loop interacts with the outer loop to produce the desired dynamics. The inner loop is a hill-climbing algorithm that performs a local-search optimisation of a solution. Specifically, a substitution is applied to the current state (S):

$$S'(t) = \{S_1, S_2, \ldots, -S_X, \ldots, S_N\}, X \sim U(1, N) \tag{3.2}$$

If the change to the solution improves the fitness of the solution, the change is kept ($S(t+1) = S'(t)$) otherwise the change is rejected ($S(t+1) = S(t)$).

The outer-loop initialises a solution and performs an update to the model, M, using Hebbian learning (Hebb, 1961) using the solution found after local search:

$$m_{ij}(T+1) = \gamma[m_{ij}(T) + \delta S_i(T)S_j(T)] \tag{3.3}$$

where γ is a linear threshold function capping the learned correlation strength to a magnitude of 1 and δ is the learning rate. The learned matrix M can be interpreted as an associative memory model of the optimisation problem's local optima, representing the relative frequency of correlations experienced in previous promising solutions. In rHN-G, after the model has been updated, the solution is reinitialised by independently assigning a -1 or $+1$ with equal probability to each entry. This completes the outer-loop. The solution resets ensure the exploration of different local optima in the solution space.

The substitution made to a solution, S_X, is constructed probabilistically using the model. An individual variable s_i is first selected. The probability some other discrete variable

s_j is included in the substitution is determined by the correlation's strength with s_i: a random number, r, is sampled from $U(\overline{M}, 1)$ which sets the threshold correlation strength required to be included in the substitution. If the magnitude of the correlation is greater than the threshold limit r, then the discrete state s_j is added to the block. The correlation sign determines the state assignment of s_j, i.e., agrees (positive) or disagrees (negative) with the decision variable. This constructs a substitution that is then applied to the solution.

The model acts to reorganise the neighbourhood of a solution by compressing the solution space and causing single-point substitutions to represent multi-variable substitutions rather than single variable substitutions as performed when the model is empty. Specifically, rHN-G will start by hill-climbing in the original solution space, searching neighbours that differ only by a single variable. As the model is updated, correlations that frequently occur between variables (that emerge in multiple local optima) are captured by the model. This, in turn, transforms the substitution applied to a solution to higher-orders of organisation, allowing local search to continue in a new and intelligent neighbourhood.

The adaptive performance of rHN-G for solving an Modular Constraint problem of size $N = 200$, $k = 10$ (module size) and $n = 20$ (number of modules) is demonstrated in Figure 3.3. rHN-G easily finds the solutions to modules. The model first learns the modules, transforming adaptive substitutions made to the solution from individual units to modules (seen in the example solution trajectory after 30 solutions). However, it is not able to reliably substitute all modules to solve the between module constraints (there still exists large conflict between modules). After 250 solutions, the model not only reliably solves the conflict between modules, but also discovers how modules can combine, further transforming adaptive substitutions to multiple modules (seen in the example solution trajectory after 250 solutions). By doing so, rHN-G can satisfy the between-module constraints efficiently and reliably.

3.1.3 Bayesian Optimisation Algorithm (BOA)

We use the Bayesian Optimisation Algorithm[1] as an example of an algorithm that uses Model-Informed Generation to explore the search space. The Bayesian Optimisation Algorithm uses a Bayesian network to model the distribution of solutions that have greater than average fitness in a population of solutions. The model is then sampled to generate new solutions, and these solutions replace solutions that have lower than average fitness in the population. Details of the the Bayesian Optimisation Algorithm

[1]The "Bayesian Optimisation Algorithm" (referred to in this thesis) differs from the potentially more familiar optimisation method called "Bayesian Optimisation". The latter refers to using a probabilistic model to approximate the fitness function (surrogate model). Whereas, in this thesis, the "Bayesian Optimisation Algorithm" refers to the method of building and sampling from a Bayesian Network model to replace the recombination operator in a genetic algorithm.

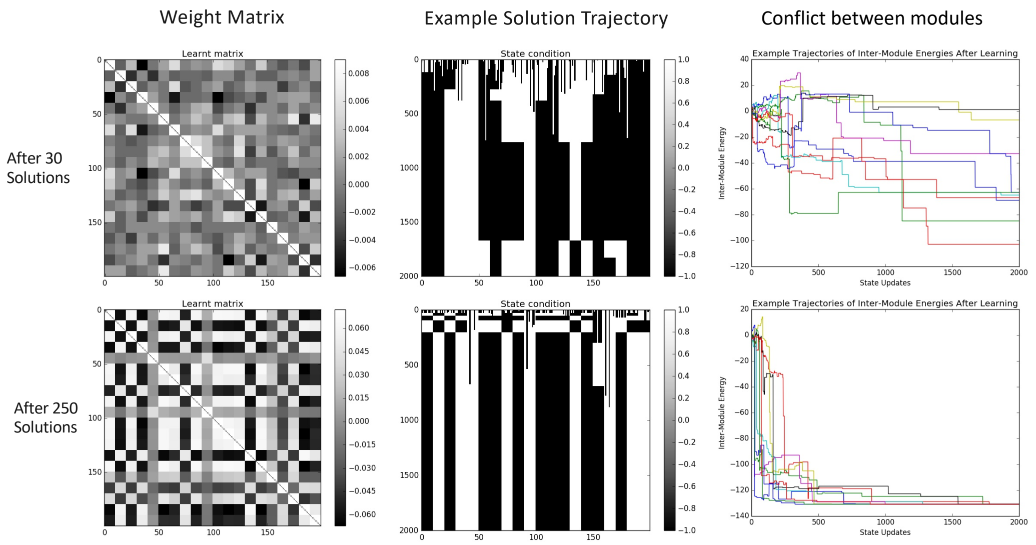


FIGURE 3.3: The performance of rHN-G is adapted during optimisation of an Modular Constraint problem with parameters $N = 200$, $k = 10$ and $n = 20$. The figure shows the correlations weights, an example of a solution trajectory and the trajectory of the conflict between modules during optimisation after 30 and 250 solutions. After 30 solutions, the model has learned to separate modules and can perform a substitution of independent modules, although unable to satisfy all the between module constraints (some conflict remains). After 250 solutions, the model has learned how to combine some modules, resulting in a simultaneous change to multiple modules whilst also reliable solving the between module constraints and consequently finding a globally optimal solution.

algorithm are available in Pelikan et al. (1999), and the algorithm is summarised in Chapter 2.

The Bayesian Optimisation Algorithm works by initialising a population of candidate solutions from a uniform distribution of possible solution states. The fitness of individual solutions is evaluated. Truncation selection (this study used the top 50% of the population) is then applied to the population to select promising solutions, which are then used to construct a Bayesian network model to represent the distribution of solutions. The model is then sampled, using forward sampling to generate new solutions to replace the bottom 50% of the population. This process is repeated until a termination criterion is met. The Bayesian Optimisation Algorithm uses a Bayesian network model to capture the underlying structure in the problem domain. It is capable of modelling partial solutions, representing a lower-dimensional model of the high-dimensional problem space. The Bayesian Optimisation Algorithm is very capable of learning the type of problem structure we explore in this section.

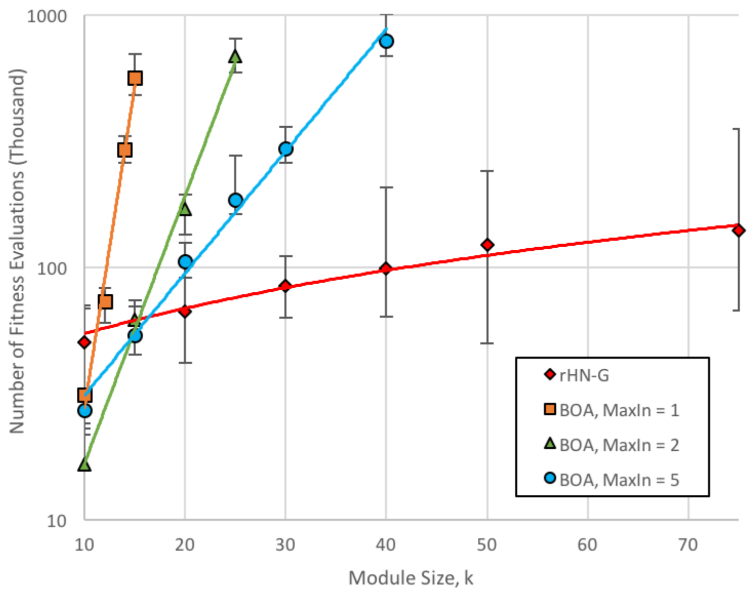


Figure 3.4: Average number of fitness evaluations required to solve the Modular Constraint problem, size $N = 10k$. The error bar shows the range of values from the test distribution. The solid lines are trend lines calculated from the average data points, the Bayesian Optimisation Algorithm results use an exponential fit, and the rHN-G result uses a linear fit.

3.1.4 Model-Informed Generation and Model-Informed Variation Performance Evaluation

The Modular Constraint problem is used to evaluate the performance of the Bayesian Optimisation Algorithm and rHN-G algorithms. The termination criterion used for each algorithm is when the algorithm can find the global optimum reliably (95% of the population is a globally optimal solution). The computational complexity is measured as the number of function evaluations performed to reach the termination criterion (i.e. model induction is assumed to be 'free'). A pre-experiment study was performed to find the minimum population size that provides a successful run for the Bayesian Optimisation Algorithm. For rHN-G the learning rate was kept constant at $\gamma = 0.0002$, and the local search steps set to $10N$. Optimisation of these parameters was not necessary - although this would reduce the total number of fitness evaluations, it does not affect the comparative result. For the first experiment, we use an Modular Constraint problem where the number of modules remains constant ($n = 10$), and the size of the module (k) is varied ($N = 10k$).

Figure 3.4 presents the average number of fitness evaluations required to solve the Modular Constraint problem. The error bars represent the maximum and the minimum number of fitness evaluations observed in the 30 independent repeats. For the Bayesian Optimisation Algorithm, we also present results using various model complexities for the Bayesian network. The complexity is controlled by parameter *MaxIn* which limits the maximum number of incoming edges into a node of the Bayesian network. The superior performance of the rHN-G method allowed us to quantify results for significantly larger problem sizes than for the Bayesian Optimisation Algorithm. For all the Bayesian Optimisation Algorithm experiments, the number of fitness evaluations scales exponentially with module size. Even when the model's capacity increases significantly beyond the minimum necessary for this problem, the time complexity remains exponential in module size. rHN-G is considerably more efficient here.

The improvement seen in the Bayesian Optimisation Algorithm by increasing MaxIn is misleading as the computational cost of constructing the model is not included in this analysis. The time-complexity for building the Bayesian network scales exponentially with the maximum number of incoming edges ($MaxIn$) to a node in the Bayesian network as $(N_p)^{MaxIn}$ where N_p is the number of problem variables. Interestingly, however, increasing the model complexity improves the performance of the Bayesian Optimisation Algorithm. The increase in model capacity is not necessary to capture the Modular Constraint problem's structure: a pairwise model is sufficient or $MaxIn = 1$. Consequently, the difference in the algorithm's performance is due to the methods used to exploit information from the model to inform search.

For the deceptive-trap problem, it is acknowledged that the scalability of model induction is exponential in the module size. The result here confirms that even for a modular constraint problem where the solutions to the blocks can be easily found (blocks are not deceptive), the Bayesian Optimisation Algorithm still scales exponentially in relation to the size of the module. This confirms the theoretical analysis performed by Mills et al. (2014) which showed an algorithm like the Bayesian Optimisation Algorithm would scale exponentially whilst a Multi-Scale Search Algorithm would scale polynomial with respect to the size of the module. Thus, the exponential scaling here isn't a consequence of the deception; the Bayesian Optimisation Algorithm requires exponential time to solve constraint optimisation problems with large modules, deceptive or not.

3.1.4.1 Efficiency Gained by Model-Informed Variation

In the previous experiment, we observed that increasing the model capacity $MaxIn$ can improve the performance of the Bayesian Optimisation Algorithm, but that it was still unable to find a global optimum in polynomial time. We further demonstrate the performance difference between Model-Informed Variation and Model-Informed Generation by removing all the differences due to model induction. The Bayesian Optimisation Algorithm and rHN-G have very different methods of identifying and learning the structure of the problem domain. Whilst there is an apparent disparity between the sophistication of the model-building techniques, namely induction of the Bayesian network compared to Hebbian learning of a simple pairwise correlation matrix, it is important to highlight what is being learned by the models. The Bayesian Optimisation Algorithm will learn

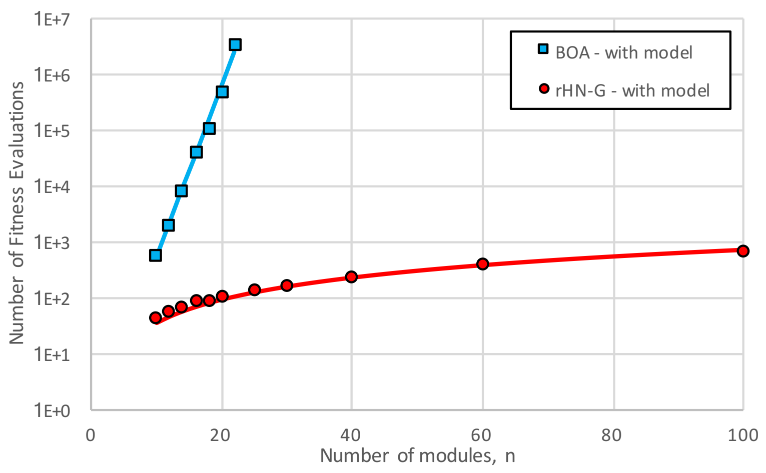


FIGURE 3.5: Average number of fitness evaluations required to solve the Modular Constraint problem given the Bayesian Optimisation Algorithm and rHN-G have been supplied with the problem structure. The Bayesian Optimisation Algorithm scales exponentially (2^n) and rHN-G scales as $n \log n$.

from relationships that have been selected to have above-average fitness. rHN-G, on the other hand, learns from locally optimal solutions, increasing the 'signal' of high fitness correlations from the 'noise' of spurious ones. Consequently, we could argue that the previous results are due to the difficulty of learning the model, as rHN-G uses fitter training examples than the the Bayesian Optimisation Algorithm. Further, the inductive bias of rHN-G is more aligned with the problem structure than the Bayesian Optimisation Algorithm. All of which would suggest that rHN-G would be a faster algorithm than the Bayesian Optimisation Algorithm for this problem, thus confounding the assessment of Model-Informed Variation. To remove all these issues, we remove the complexity of the model-building from both algorithms.

In the next experiment, models that correctly represent the modules independently are provided to both algorithms. Giving the algorithms the correct model from the outset shows that each algorithm's performance is not due to the ability to encode the problem structure in the model but instead due to its ability to exploit the information stored in its model. To further verify its the method of exploitation rather than model induction, both the Bayesian Optimisation Algorithm and rHN-G are restricted so that no model learning occurs during their iterative search. The result, therefore, evaluates the performance difference between Model-Informed Variation and Model-Informed Generation only. One can see that providing the model to the algorithms removes the dependency on the modules' size: all that remains is to satisfy the between-module constraints. For this reason, the module size is kept constant ($k = 10$), and the number of modules in the problem varied.

Figure 3.5 presents the results for the Bayesian Optimisation Algorithm and rHN-G given the model (the model has correctly identified the within module relationships). The Bayesian Optimisation Algorithm shows an exponential scaling as the number of modules increases, specifically $0.5(2^n)$, whereas the rHN-G required $1.1(n \log n)$. This result confirms that the efficiency saving seen between the Bayesian Optimisation Algorithm and rHN-G is not a result of the model induction complexities. This experiment removes any difficulty with identifying and learning the structure. Rather, it is the difference between using the model to inform generation and using the model to inform variation. Model-Informed Generation cannot generate optimal solutions until it has learned the dependencies between modules (as well as within). Learning between-module dependencies from a local optima sample is very difficult for both methods because good combinations of modules occur only very slightly more often than incorrect combinations of modules in the Modular Constraint problem, as demonstrated by the results in Figure 3.4. For rHN-G this is not an issue: rHN-G does not need to learn how modules combine because Model-Informed Variation puts the modules together correctly and easily. It may be possible to incorporate Model-Informed Variation into an Estimation of Distribution Algorithm, enabling access to the advanced model space of the Bayesian Optimisation Algorithm. Indeed, Lima et al. (2006) demonstrates this using the model to adapt a mutation operator and shows an improvement in the Bayesian Optimisation Algorithm's performance. The results differ here as rHN-G uses the model to update a solution iteratively and we show a polynomial vs exponential difference. We leave the work of understanding of how Model-Informed Generation and Model-Informed Variation can be used synergistically as future work.

In summary, the experiments show that for some EDAs, for which BOA is an example, they find the Module Constraint problem structure particularly challenging (causing exponential scaling). This is because of the way they apply selection. Specifically, selection is applied to filter a subset of the population. This population sub-set is then used to construct a model. The model is then sampled to generate new solutions, replacing the solutions not selected for model building. In this case, selection is performed at the population level (does this solution have greater fitness than the mean fitness of the population). As a result, selection can ignore the weak between-module constraints in favour of the strong within-module constraints. As there are many more ways to have incorrect module combinations compared to correct combinations, the model learns incorrect between-module relationships early in the optimisation process. As a result, the algorithm cannot satisfy the between module constraints. However, exploiting the problem structure becomes easy when selection is applied at the individual level (is this individual solution better than this other individual solution). For BOA, the population could be initialised using local search therefore removing the likelihood of learning spurious correlations. However, this solution fails if the problem contained hierarchical structure (nested modules). In this case, local search would then have to be applied at the module representation. and therefore the Bayesian Optimisation Algorithm would fail again. Whereas a Multi-scale search algorithm such as restart Hopfield network with Generative Associations would continue to be successful.

3.2 Model-Informed Variation Using a Neural Network Model

In the previous section, we showed that rHN-G, specifically the idea of multi-scale search, could efficiently solve the Modular Constraint problem. The performance is attributed to an adaptive reorganisation of a solution's neighbourhood (performed by Model-Informed Variation). This adaptation is recursively applied, transforming the variation to successively higher orders of organisation, from individual units to modules, to a combination of modules. However, the model used by rHN-G is limited to a single level representation. We hypothesise that a model capable of explicitly inducing a multi-layer representation will be capable of overcoming problem structure that a single layer representation (such as the case for rHN-G) cannot. Further, an explicit multi-level representation provides a model space more in line with the Multi-Scale Search Algorithm idea and thus extend the current capability of Multi-Scale Search Algorithm's. Deep learning provides a model space that can explicitly represent multiple scales of variation in a dataset. We, therefore, hypothesises that deep learning provides a suitable model space to advance the idea of a Multi-Scale Search Algorithm.

The use of neural network models to perform Model-Informed Generation has been explored (Churchill et al., 2014; Probst, 2015; Probst and Rothlauf, 2015; Santana, 2017); however, results are not competitive with algorithms like the Bayesian Optimisation Algorithm. Therefore in this section, we perform a preliminary study to develop our understanding of how a neural network model can learn and exploit information to adapt a solution's neighbourhood. We do this by replacing the correlation model used by rHN-G with a single layer autoencoder model. We limit our analysis to a single layer model to verify that the method for learning and exploiting information is in line with rHN-G. We then perform experiments to understand how incorporating a neural network can enhance the performance of a Multi-Scale Search Algorithm's and provide a preliminary understanding of what problem structures differentiate the model capacities. The work presented in this section provides the founding knowledge to develop the Deep Optimisation algorithm.

3.2.1 Restart-Autoencoder with Generative Associations (rA-G)

The Restart Autoencoder with Generative Associations (rA-G) uses the same algorithm as rHN-G but replaces the correlation matrix with an autoencoder model using a single hidden layer. We selected the autoencoder model due to its capability of dimensional reduction using unsupervised learning, a relatively simple and robust model, and easily extendable to a deep architecture. Further, the autoencoder architecture can be constructed using a layerwise learning process (Hinton and Salakhutdinov, 2006). The layerwise construction of a deep neural network fits naturally with the idea of multiple scales of transitions in individuality, further explored in chapter 4. Additionally, the autoencoder architecture has been used for both deterministic and probabilistic interpretations, providing the potential for Model-Informed Generation and Model-Informed Variation. We leave the detailed discussion about the autoencoder model and suitability for learning and exploiting deep problem structure to chapter 4 where the Deep Optimisation algorithm is developed. In this section, we only explore if an autoencoder model can replace the correlation model used by rHN-G. The rA-G algorithm is presented in Algorithm 3.

Algorithm 3: rA-G Algorithm

```
Initialize: Autoencoder model;
while Termination Criteria not met do
    Initialize: Solution using random distribution;
    while Search iteration < max search iterations do
        Select solution variable at random;
        Select a hidden node that is likely to respond to the selected variable based
          on the connection strength;
        Construct a higher-order substitution based on the hidden nodes connection
          strength with other variables;
        Make a change to the solution, informed by the model;
        if Change to Solution improves Solution quality then
            Accept change to Solution;
        else
            Reject change to Solution;
        Update the Autoencoder model using the optimised solution;
```

rA-G uses an autoencoder model to learn a higher-level representation of a solution. The autoencoder model consists of an encoder (E) and decoder (D) network that transforms a solution (S) to a hidden representation (H) and then back to the original input representation. Specifically, $S_r = u(X_r) = u(D(E(S)))$, where S and S_r are the solution and solution reconstruction respectively, X_r is the output from the decoder and u is a unit step function, with a threshold $t = 0$. Specifically,

$$u(X) = \begin{cases} 0, \text{ if } X < t \\ 1, \text{ if } X \geq t \end{cases} . \tag{3.4}$$

The encoder network performs a transformation from the input units, S, to hidden units H using the hyperbolic-tangent (TanH) nonlinear activation function (f) on the sum of the weighted inputs. Specifically, $H = f(WS + b_e)$ where W and b_e are the weights and bias respectively for encoder network. The decoder network generates a reconstruction, X_r, from the hidden representation at H using $X_r = f(W'H + b_d)$, where W' is the transpose of the encoder weights W (i.e., the weights are tied), b_d is the decoder bias. The encoder network induces a feature representation from the dataset, represented by the hidden layer. The decoder network uses the feature information to generate the input. Thus, the autoencoder is trained to learn a reconstruction of its input. Learning a reconstruction is a trivial task, i.e., the identity matrix. However, constraining the hidden representation to a lower dimensionality than the input encourages the representation to learn a compression of input space. Thus, the feature representation (hidden layer representation) captures the salient features. These features explain the prominent variance in the dataset, thus capturing the most important information to maximise its ability to reconstruct the input.

The weights are initialised using the uniform distribution $U(-0.01, 0.01)$. Both the encoder and decoder biases are initialised to 0. Training is performed using the back-propagation algorithm and gradient descent to minimise the reconstruction error (mean squared error) between X_r and input S, Equation 3.5 where n is the number of training examples used per an update of the model. In rA-G, the model is updated using online learning, and therefore $n = 1$ and a solution is only used once to update the model parameters. As in rHN-G, the learning rate is an important parameter as it controls the size of the update made to the model's parameters from the single data point.

$$\text{error} = \frac{1}{n}\sum_{i=1}^{n}(S_i - D(E(S_i)))^2 \tag{3.5}$$

3.2.1.1 Model-Informed Variation for rA-G

Model-Informed Variation is performed by selecting a random solution variable s_i. A hidden unit is then selected that responds to s_i, based on the connection strength vector W_i between the decisions variable s_i and all hidden units H. The mean connection strength is used as a threshold, specifically: $\sum_{j=1}^{N_h} |W_{ij}|/N_h$, where N_h is the number of hidden units. Hidden nodes with a connection magnitude greater than the threshold are possible candidates that respond to s_i.

Then, the solution is then encoded to provide an activation response of $H = E(S)$. The model uses the TanH activation function. Therefore, initially, the model will have an activation response close to zero. As the model learns information, the activation response can tend towards the limits of the nonlinear function (-1 and 1). Thus activation responses further away from zero indicate that the hidden neuron has learned something meaningful. Therefore, a hidden neuron is selected if its activation value is greater than a threshold randomly sampled from $U(0, 1)$. If multiple activation values are greater than the threshold, a single hidden unit from the set of units exceeding the threshold is selected randomly. The selected hidden unit will have a strong response to the current solution and be responding to solution variable s_i.

If a hidden unit has been selected, it is then used to construct a higher-order substitution using the connection vector $W'j$ between the individual hidden node h_j and all output units X_{ri}. A random threshold, calculated between the maximum connection strength ($max(|W'_j|)$) and the average connection strength ($\sum_{i=1}^{N} W'_{ij}/N$, where N is the number of output units), is applied. Output units with a connection strength to H_j greater than the threshold are included in the substitution (S_X). The sign of the change made to the included variables is determined by the agreement or disagreement between the signs of the connection weights with the selected variable s_i. Specifically, suppose an output unit has a connection weight to the hidden unit with the same sign as connection weight between s_i and the hidden unit. In that case, the change made to the output unit is the same as the change made to the s_i variable. If the weights have different signs, the change made to the output variable is the opposite of that made to the s_i variable. This completes the construction of the higher-order substitution.

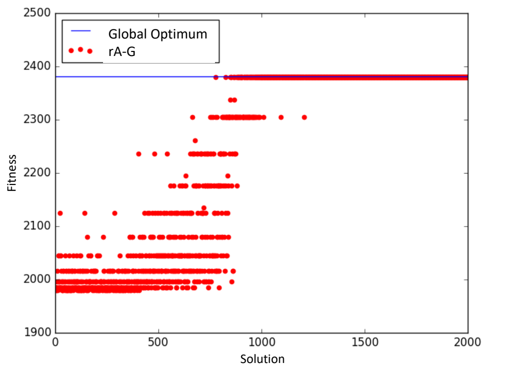
Initially, when the model has small connection strengths, the weights and activation response (H) will be close to 0. Consequently, the likelihood of any connection being included in a higher-order substitution will be small. Therefore only a single-bit substitution will likely be made, allowing for search in the original solution space. As the model learns information about the problem structure, the activation response increases, increasing the likelihood of using information from the model to construct a higher-order substitution. The method presented here is specifically designed to replicate the online adaptive process that rHN-G performs. In Chapter 4, we develop a more general and robust method applicable to work with a deep neural network.

3.2.2 Evaluation of rA-G

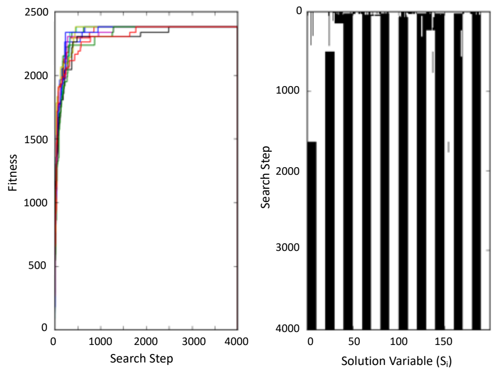
The performance of rA-G is evaluated using a Modular Constraint problem containing 200 variables ($N = 200$) with 20 building blocks ($n = 20$) of size 10 ($k = 10$). Here there are 2^{20} ways to combine the modules, two of which are globally optimal. Consequently, this represents an intractable problem for a hill-climber or random search to solve. The number of search steps per solution was set to 4000, and the algorithm was set to terminate after 2000 solutions. The autoencoder model used 40 hidden neurons at the hidden layer – more than sufficient to represent a dimensional compression of the problem. Gradient descent is used to update the model parameters, using a learning rate of 0.01.

Figure 3.6 presents each solution's fitness at the end of the search process. Initially, rA-G performs single-unit substitutions only and finds solutions to the modules. This is presented as solutions with sub-optimal fitness in Figure 3.6(a). Like rHN-G, rA-G becomes easily trapped as it is unable to satisfy the between-module dependencies. After around 400 solutions, the autoencoder model starts to inform adaptive higher-order substitutions, which we see in Figure 3.6(a) as solutions with greater fitness are being found. Learning continues allowing for greater fitness solutions to be found. Finally, the rA-G correctly learns the problem structure such that, given a random solution, rA-G can reliably find the global optimum solution.

Figure 3.6(b) presents the fitness trajectory of 10 solutions and the solution trajectory for a single solution when the autoencoder model has learned the problem sufficiently to allow the algorithm to find a global optimum reliably (after 2000 solutions). Each trajectory starts with a solution initialised at random. The fitness trajectory plot shows that a solution begins with low fitness due to random initialisation. The solutions are then iteratively updated, and in all 10 cases, a globally optimal solution is found. The solution trajectory presents the adaptive changes made to a solution. Both single-bit and module substitutions are being performed, confirming that rA-G provides the same continuous adaptation capability as rHN-G (after about 1000 solutions).



(a) Fitness of a solution (after search). Overtime, the model is used to inform search, allowing the algorithm to change multiple variables simultaneously and consequently enabling search to find solution with higher fitness



(b) Fitness trajectory of 10 independent solutions and a solution trajectory for a single solution using the model learned at step 2000. rA-G finds a globally optimal solution by making changes to complete modules in a single search step.

FIGURE 3.6: Performance of rA-G on the Modular Constraint problem of size 200 with 20 modules.

Figure 3.7 presents the reconstruction error for a solution before training. Initially, the model produces poor reconstructions of solutions. Whilst the reconstruction error is noisy, there is a systematic reduction in the reconstruction error over time, indicating that the model is learning the structural regularity over time. At around solution 1000, the training error rapidly reduces, indicating that the model has accurately captured the problem structure. We observe that in comparing with Figure 3.6(a), this point coincides with finding a globally optimal solution reliably.

Figure 3.8 presents the weights of the autoencoder after it has been updated using 30, 130, 300 and 2000 solutions. After 30 solutions, the weights appear noisy; however, some hidden neurons are starting to represent modules (a grouping of weights, connecting 10 inputs variables and a single hidden neuron, with similar values and the same sign). This structure becomes clearer after 130 solutions. After 300 solutions, a building block is represented clearly by at least a single hidden neuron; this coincides with Figure 3.6(a)

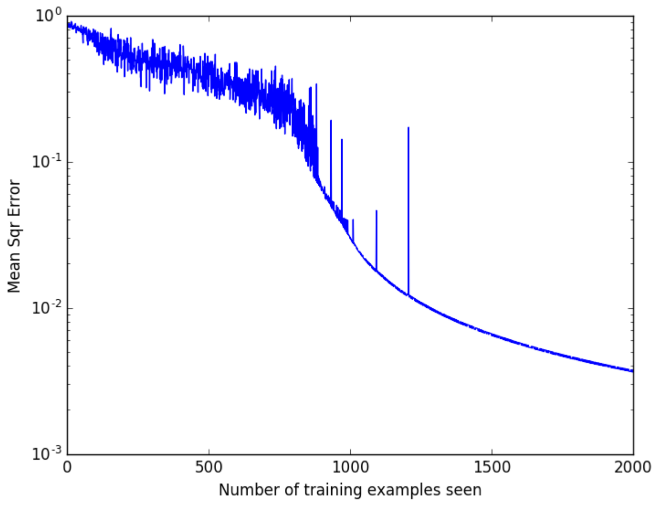


Figure 3.7: Solution reconstruction error.

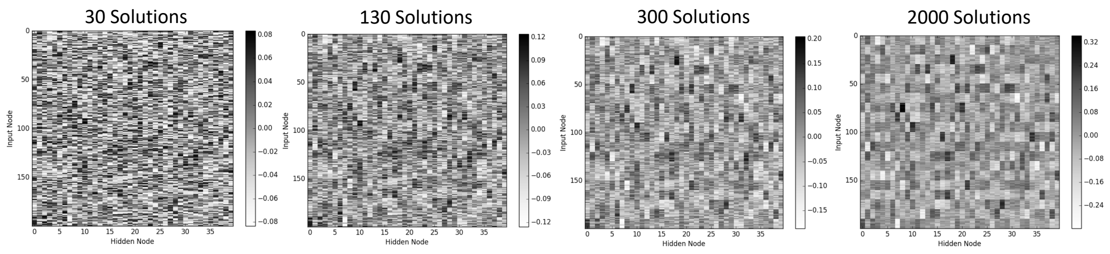


Figure 3.8: The weights between the input and hidden layer in the single layer autoencoder network used by rA-G. The hidden layers learn to separate and represent the modules in the problem (shown as a single hidden node having a strong connection with a 10 input nodes)

where the quality of a solution found improves from the baseline, in which case Model-Informed Variation is making module substitutions to improve a solution. After 2000 solutions, there is a more explicit modular organisation in the weights, which in turn facilitates reliably search to find the global optimum as seen in Figure 3.6(a).

We further confirm that rA-G has both learned the Modular Constraint problem structure and can efficiently exploit this information by evaluating its ability to escape local optima. We take the model learned after 2000 runs and test if rA-G can find a globally optimal solution when given a solution that is the furthest 'distance' away in terms of module solutions, i.e., all module solutions are correct, but requires $n/2$ module substitutions to assemble them into a globally optimal solution. In this case, initialisation of $S = 1^N$. Figure 3.9 presents the fitness trajectory of 10 independent runs and a solution trajectory from one of the runs. rA-G can find the global optimum solution in all runs. Further, the solution trajectory shows that rA-G performed complete and independent module changes to reach the global optimum solution. Thus, rA-G has successfully adapted the neighbourhood of a solution from individual variables to individual modules, allowing the algorithm to hill-climb in module space.

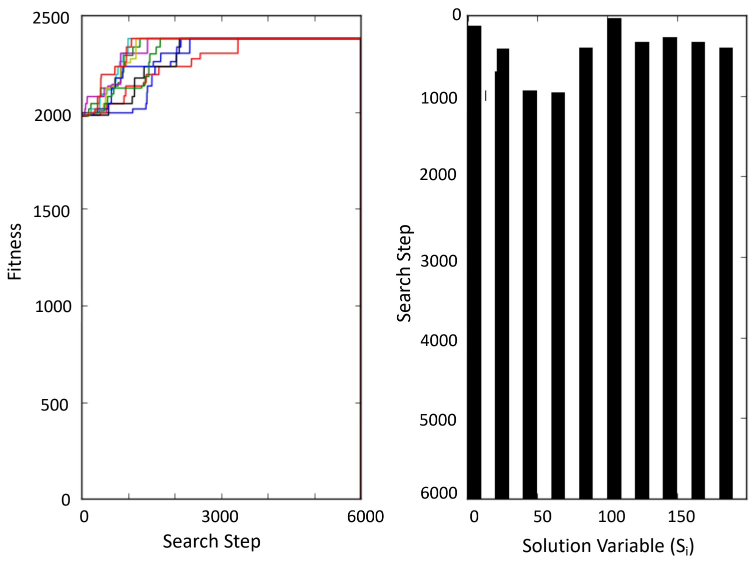


FIGURE 3.9: Given a locally optimal solution, furthest from the global optimum, rA-G is capable of escaping the local to find the global optimum solution.

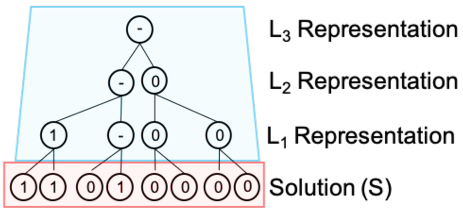
These experiments show that rA-G is capable of operating in the same manner as rHN-G. Specifically, capable of using the model to inform partial substitutions to be made to a solution. We have found that hidden nodes represent the module structure of the problem. The method developed to exploit this information is designed to replicate the behaviour of rHN-G. However, we find limitations with this method, which are demonstrated in the following section.

3.2.3 Challenging Problem Structure

In developing rA-G, we have shown that the performance, in terms of functionality, is the same as rHN-G. However, we are particularly interested in how an autoencoder model, and more specifically a multi-level neural network, can provide optimisation capacity that a correlation matrix, and more generally a single-level representation, cannot. Therefore, in this section, we perform an exploratory study to understand the types of problem structures that differentiate the performance between rHN-G, which employs a correlation matrix, and rA-G, which employs a single-layered neural network.

3.2.3.1 Hierarchy

We hypothesis that a deep neural network model provides a model capacity to represent hierarchical problem structure. We therefore first explore the ability to represent and exploit a multi-level problem structure using a shallow model. A benchmark that has been used to evaluate the performance of an MBOA to overcome the problem characteristic of hierarchical structure is the Hierarchical If and Only If (HIFF) (Watson et al., 1998). HIFF is a consistent constraint problem - meaning that all constraints can be simultaneously satisfied - for which there are two possible solutions. The objective is to find a solution that satisfies the maximum number of constraints imposed on the problem.



(a) HIFF problem structure and representation example of a solution of size 8 where '-' represents a null value. Satisfying a 2nd level constraint can require a change to 2 solution variables. Satisfying a 3rd level constraint can require a change to 4 solution variables.

HIFF		
a b	$t(a,b)$	$f(a,b)$
1 1	1	1
0 0	0	1
Otherwise	-	0

(b) HIFF transformation t, and fitness function f. '-' represents a null value. a and b represent the two nodes from the lower-level representation.

FIGURE 3.10: The Hierarchical If and only If optimisation problem

The generalised hierarchical construction is summarised from Watson et al. (1998) here. The solution state to the problem is $x = \{x_1, \ldots, x_i, \ldots, x_N\}$, where $x_i \in \{0, 1\}$ and N is the size of the problem, p represents the number of levels in the hierarchy and N_p represents the number of modules of length L_p at each hierarchical level. Each module containing k variables is converted into a low-dimensional representation of length L_p/R by a transformation function t, where R is the ratio of reduced dimensionality creating a new higher-order string $V^{p+1} = \{V_1^{p+1}, \ldots, V_{N_p L_p/R}^{p+1}\}$, where V_i^{p+1} is the output of a transformation function using inputs from V^p. In HIFF, $k = 2$ and $R = 2$ using the transformation function detailed in Table 3.10(b), where a solution to a module is a

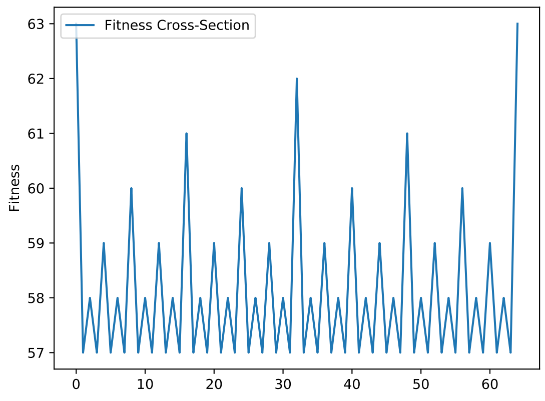


FIGURE 3.11: A cross-section of the fitness landscape between the two global optimal solutions of size 64 at $S \in 1$ and $S \in 0$ placed at 0 and 64 on the x-axis. The x-axis corresponds to the number of solution variables, starting from S_0, that has changed to the other global solution, i.e., changed from 1 to 0. The y-axis presents the fitness of that solution. The fitness landscape presented by HIFF has a fractal structure that requires recursive adaptation to the size of the change made to the solution to follow the gradients that direct the search to a globally optimal solution.

positive correlated pair of variables. The function is also detailed in section 5.2.3.2 were the construction is used in the developed synthetic problem.

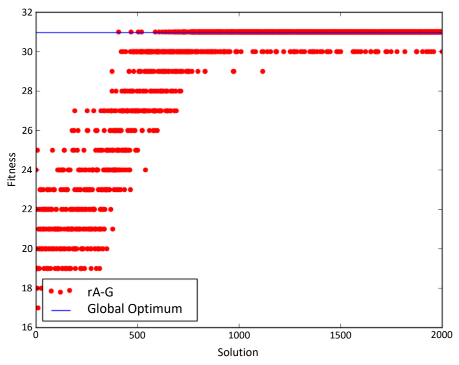
The structure is inspired by a bottom-up construction of sub-functions that combine to create high-order sub-functions. The output of low-level sub-functions is used as an input to higher-level sub-functions. A sub-function only produces a non-null output if its internal constraints are satisfied. Therefore, solutions to high order sub-functions only become apparent when solutions to the low-level sub-functions are found. A substitution that invalidates a low-level sub-function means that all higher-level sub-functions that depend on the lower-level sub-function are also invalidated. Thus, an optimiser must identify and conserve low-order solutions at multiple scales to search at the high order space.

An illustration of the hierarchical problem structure is presented in Figure 3.10(a). Nodes at the 2nd level representation are dependent on the state of 4 solution variables. A fitness reward is only provided if the solution variables are all 1's or all 0's. There is no fitness information between these solutions at this level of representation. The information to find the combination that satisfies 2nd level unit is provided by the 1st level units that correctly combine pairs of bits. Searching in variable pairs (local search at the 1st representation scale) allows for finding the correct combinations for the 2nd level units as the correct combination is a neighbour at the 1st level representation scale. This is consistent at all representation levels, leading to the fractal fitness landscape presented in Figure 3.11. If an algorithm can collapse the problem's dimensionality at one level, the fitness landscape presented at the next level is as straightforward to navigate as the previous level.

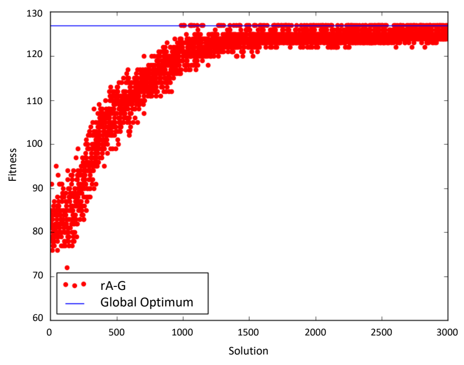
We evaluate rA-G's performance using a problem size of 32 (HIFF_{32}) and 128 (HIFF_{128}). The autoencoder model used 25 and 100 hidden nodes, respectively. A learning rate of 0.003 was used for both problem sizes. Note that, in this section, we are only performing a preliminary investigation into how rA-G performs on this challenging problem structure. An extensive performance evaluation is performed in chapter 5.

Figure 3.12, presents the performance of rA-G on HIFF_{32} (Figure 3.12(a)) and HIFF_{128} (Figure 3.12(b)). rA-G successfully finds a globally optimal solution even though the problem contains a hierarchical structure (a multi-level representation of how modules combine) despite the model being limited to a single level representation. Figure 3.13 presents an example of a solution trajectory that was successful in finding a global optimum for the HIFF_{32} problem (solution example 900 from Figure 3.12(a)). rA-G can perform adaptive changes at multiple levels of representation, from pairs of variables and up to modules of size half the solution size.

Figure 3.14 presents the change in the autoencoder weights during the optimisation of HIFF_{32}. It is observed that the autoencoder model first represents the 1st level modules (two variables), where a hidden neuron responds to a module at the 1st level



(a) Fitness of solution found at end of search process for the HIFF$_{32}$



(b) Fitness of solution found at end of search process for the HIFF$_{128}$

FIGURE 3.12: rA-G performance in solving HIFF, a problem containing hierarchical structure, using a single layered autoencoder model.

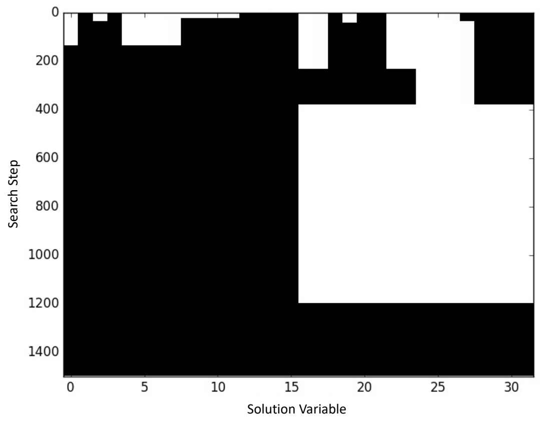


FIGURE 3.13: Solution trajectory of rA-G in solving HIFF$_{32}$. rA-G performs a changes to the solution of different sizes.

representation, Figure 3.14.a. As the algorithm proceeds, the weights start representing modules at the 2nd level modules (modules of size 4) (Figure 3.14.b) and then 3rd and 4th level modules (Figure 3.14.c). Information about the lower-level modules appears lost. However, at closer inspection, the difference in the connection weights is sufficient to represent the multi-level representation. Where weaker connections represent the higher-level modules and stronger connections represent the lower-level modules.

We further confirm that HIFF does not require a multi-level representation to solve by investigating how rHN-G solves HIFF. Figure 3.15 presents the fitness of each solution found during the algorithm run and the learned correlation matrix for when rHN-G can reliably find a global optimum (after 500 solutions) for HIFF$_{32}$. The hierarchical structure of HIFF can be well approximated as the difference in correlation strength, i.e., large connection strengths represent variable relationships for low-level modules, and small connection strengths represent larger modules. The probabilistic method for constructing partial substitutions is then able to exploit this information to solve HIFF.

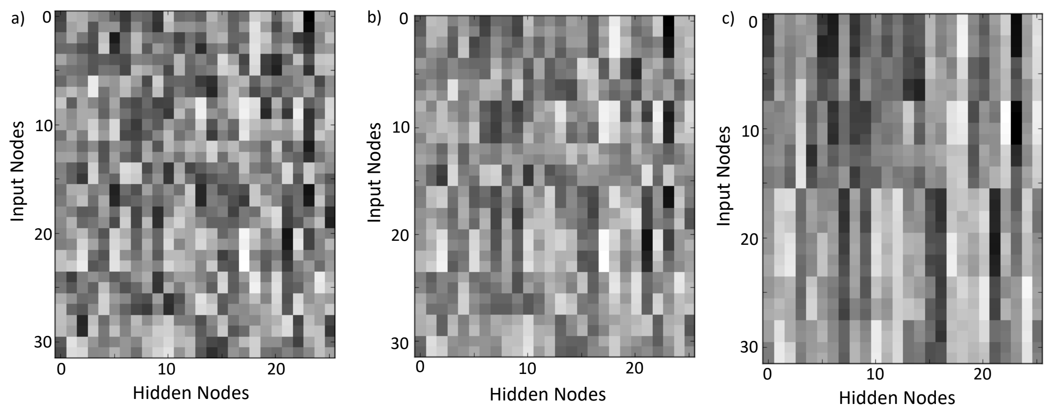


Figure 3.14: The weights of the autoencoder model during optimisation of a size 32 HIFF problem: a) after 300 solutions, b) after 400 solutions, and c) after 800 solutions. The autoencoder doesn't represent the hierarchical structure; instead, two lower-level units are combined and represented by a single hidden neuron. The low-level structure is encoded as difference in connection weights in the same hidden neuron.

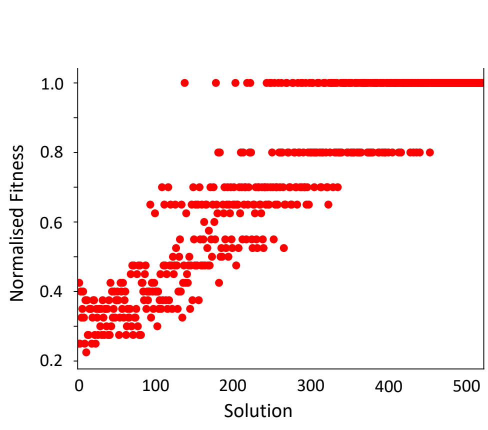
We find that both rHN-G and a single layer neural network model are sufficient to solve HIFF, yet the models used are shallow.

To conclude, for HIFF, it is unnecessary to maintain information about low-level modules when a higher-level module has been identified and learned, i.e., how a higher-level module was constructed. This is due to the strict tree structure used in HIFF. A high-level module encapsulates the low-level modules entirely. Therefore, using this higher-level module will always provide a greater improvement to a solution's quality than using any of the lower-level modules. Further, the models can represent the hierarchical structure as a difference in the connection weights. Consequently, we conclude that hierarchy alone is not sufficient to require a deep neural network to represent the problem structure. This suggests that a deep representation is necessary only if a substitution of a lower-level module is required, even when higher-order modules have been identified and learned. This is explored in Chapter 5.

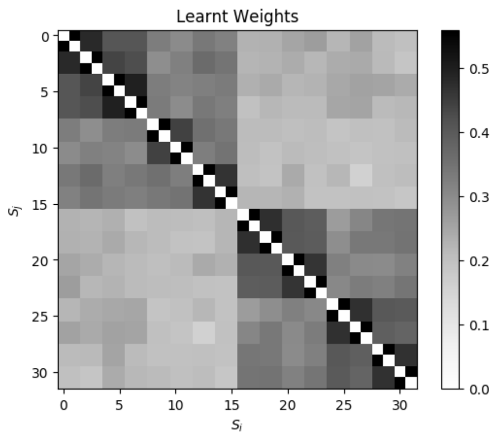
3.2.3.2 Higher-order relationships

A single-layered autoencoder model is more sophisticated than a correlation matrix. This section explores problem structure that can be learned and represented using an autoencoder model (rA-G) but not by a correlation matrix (rHN-G). Here we show that the autoencoder model can accurately learn the problem structure. However, the model-informed search method used by rA-G fails to exploit this information. This understanding leads to the development of a general Model-Informed Variation method used by Deep Optimisation and detailed in Chapter 4.

The machine learning benchmark named the 424 encoder problem (Ackley et al., 1985) has been used as an example to demonstrate the capability of a neural network model



(a) Fitness of each solution found by rHN-G.



(b) Correlation matrix learned when rHN-G can reliably find a global optimal solution.

FIGURE 3.15: Performance of rHN-G in solving HIFF. rHN-G can represent the hierarchical problem structure by differences in the correlation strength. The probabilistic method used to generate a partial change can construct partial solutions of different sizes, allowing rHN-G to solve the problem.

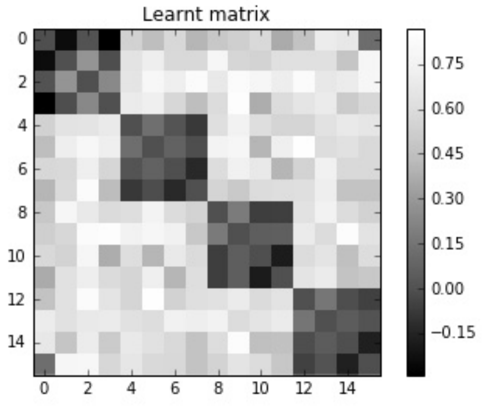
to learn a complicated compression of a dataset. The 424 encoder problem contains interesting properties that we explore in this section. The problem is to represent four data points with a dimensionality of four using two dimensions (a compression from four dimensions to two dimensions that accurately represents the four data points). The data points represent a one-hot solution, in which there exist four possible solutions points (1000, 0100, 0010 and 0001). It is possible to represent these data points using two binary bits, i.e., 2^2 data points. Thus a single-bit change (local) at the compressed representation transforms back to the input as changes between '1 hot' solutions. This problem is particularly interesting because there exists a learnable bijective mapping between the data points that cannot be represented using pairwise information: it cannot be encoded with a correlation matrix. Further, the '1 hot' solutions are a common representation in combinatorial problems, such as assignment problems.

To test the ability of rHN-G and rA-G to learn and exploit this type of problem structure, we create a problem that is a concatenation of 424 encoder modules. Each module is separate, and the fitness function is simply a summation of the fitness of each module - thus, the global optimum is one where all modules contain a one-hot solution. A modules fitness and compression is detailed in Table 3.1. This problem is trivially solved using a hill-climber. However, we are interested in the model's ability to learn the compression and subsequently substitute '1 hot' solutions into each module. We, therefore, assess the ability to exploit this information, i.e., maintain the fitness of the module but change the '1 hot' solution of an individual module. An optimisation problem that requires a substitution to module solutions to find the global optimum is developed in chapter 5. We test both rHN-G and rA-G on a problem size of sixteen containing four 424 encoder modules. Our preliminary investigation found that neither rHN-G nor rA-G can exploit this information to change between '1 hot' solutions. Therefore, we limit the discussion to why this is the case and the compression learned by the models.

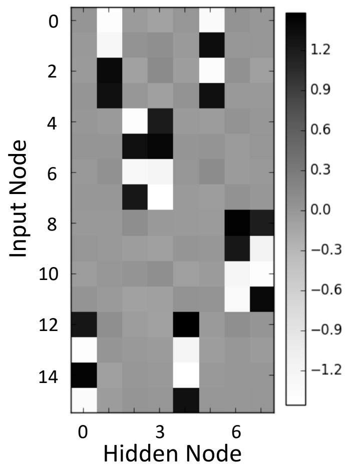
TABLE 3.1: Transformation and fitness function for a 424 module

424 Transformation

$a\ b\ c\ d$	$t(a, b, c, d)$	$f(a, b, c, d)$
1 0 0 0	0 0	1
0 1 0 0	0 1	1
0 0 1 0	1 0	1
0 0 0 1	1 1	1
Otherwise	-	0



(a) Correlation model in rHN-G



(b) Autoenocder model in rA-G

FIGURE 3.16: Weights learned by rHN-G and rA-G for the 424 module problem

Figure 3.16 represents the learned correlation matrix of rHN-G and the learned connection matrix of a single-layered autoencoder model used by rA-G. On average, a locally optimal solution (as easily found by local search) will show a variable that is positively correlated with other variables in the solution. For rHN-G, this means that the correlation model learns that variables should be positively correlated. Consequently, partial substitutions to modules cannot be constructed using the model, as other variables outside of the module will be used to construct the substitution. The learned weights evidence this as the connection strengths outside a module are greater than within a module. rA-G, on the other hand, is successful in separating and representing a perfect compression of the modules. Here, two hidden neurons are required to compress a single module - the hidden nodes work together to represent the compression. To demonstrate that the autoencoder can accurately learn the compression, we perform an experiment of generating random solutions at the solution level representation and the hidden representation for a problem containing 8 modules (problem size 32). Figure 3.17 shows the proportion of variable combinations, whether one of the '1 hot' solutions (partial solutions 1, 2, 3, or 4) or a combination that is not a '1 hot' solution (not partial solution), when generating 30 solutions by randomly initialising the solution representation and randomly initialising the 1st hidden layer and decoding to generate a solution. Random initialisation performed at the solution representation produces a random distribution of solutions. Therefore, only about 25% of the solutions contain a '1 hot' solution - four out of the possible sixteen combinations for each module - seen in Figure 3.17 with "Not PS" at 0.75 proportion generated. However, random initialisation at the hidden representation decodes back to the solution representation to produce a distribution of solutions that contain an improvement in the number of '1 hot' solutions for each module: the compressed representation has removed alternative combinations that are not a '1 hot' solution - for example for module 1 a partial solution (a '1 hot' solution) was always generated - whilst at the same time maintaining the ability to generate each '1 hot' solution (partial solutions 1 to 4). Thus, the autoencoder has compressed the space of possible configurations generatable.

As shown in Figure 3.16 two hidden neurons are required to represent a compression of the '1 hot' solutions. However, the method used by rA-G to exploit this information fails to substitute an alternative '1 hot' solution. This is because the technique used for constructing a substitution required for this problem structure is dependent on the state of (at least) one other hidden variable used in the compression. A substitution created only using a single hidden neuron from the weights learned by the autoencoder constructs a module substitution with values of 1100, 0011, 1001 or 0110 – all substitutions are not a '1 hot' solution and therefore will be rejected. However, we observed in Figure 3.17 that '1 hot' solutions can be generated from the hidden representation. This is because the value of the solution variable is dependent on two hidden neurons. Further, we envision problems where a solution variable is dependent on many hidden neurons. Therefore, we must develop a more robust method for exploiting information from the autoencoder model.

This experiment shows that the 424 encoder problem contains an interesting problem structure that requires two hidden neurons to be accurately represented. Further, the transformation presents an exact binary compression. Thus, the variation at the solution level that provides a local change to the transformed representation (a single bit) depends on the solution's current state (or the state of the other transformed bit), a property we explore later to construct overlapping dependencies. Further, the 424 encoder problem is not learnable using a correlation matrix. This section provides a stepping-stone to developing the DO algorithm that uses this knowledge of complex problem structure that is not present in other synthetic problems used to evaluate MBOAs.

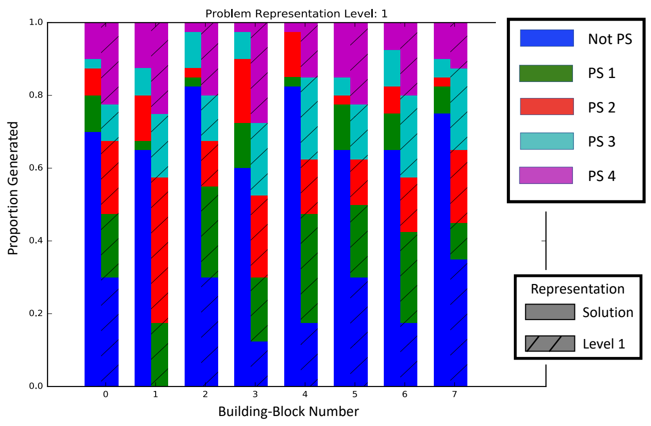


FIGURE 3.17: Compression of the variability for the '1 hot' module problem. At each representation level, 30 solutions are randomly generated. For each module, the solution distribution is presented. When randomly generating at the solution representation level, 25% of the distribution is a valid solution (four '1 hot' exist out of sixteen possible combinations for each module). When randomly generating solutions from the first hidden representation, the proportion of '1 hot' solutions is much larger, generally exceeding 50%. Consequently, the variability at the first hidden representation has been reduced whilst maintaining the capability to generate each '1 hot' solution for each module.

3.3 Summary

In this chapter, we evaluated the performance of Model-Informed Generation and Model-Informed Variation using rHN-G and the Bayesian Optimisation Algorithm as an example of Model-Informed Variation and Model-Informed Generation, respectively. We found that using Model-Informed Variation instead of Model-Informed Generation can reduce the time-complexity required to solve a modular constraint optimisation problem from exponential, using Model-Informed Generation, to polynomial, using Model-Informed Variation. We verified that this result is caused by differences in searching the solution space using the model and not by differences in model induction complexity. This is because Model-Informed Variation, unlike Model-Informed Generation, does not need to learn how modules combine; instead, it uses its model to adapt the neighbourhood of a solution, allowing the local search to proceed in a new and intelligent solution space.

To enhance rHN-G, and in more general Multi-Scale Search Algorithms (Multi-Scale Search Algorithm's), we then developed rHN-G to use an autoencoder neural network model. We called this algorithm rA-G and found that it was successful in learning and exploiting the same problem structure as rHN-G and provided the same functionality. We then performed a preliminary investigation into the types of problem structures that rA-G can represent that rHN-G cannot. We found that synthetic problems used in the literature to evaluate the performance of hierarchical problem structure (multiple scales of modules) were insufficient to require a multi-level representation and can be sufficiently approximated by a correlation matrix and a single-layered network (a shallow model). Further, we identified the 424 encoder problem, a benchmark used to understand learning algorithms, that cannot be represented using a correlation matrix but can be represented using an autoencoder model. However, rA-G cannot exploit this information due to the construction of a substitution (the Model-Informed Variation method) being dependent on the state of multiple hidden variables. This chapter has provided sufficient direction to develop an algorithm that uses a deep autoencoder to transform a solution's neighbourhood using Model-Informed Variation. This is presented in the next chapter, where we introduce the Deep Optimisation algorithm.

Chapter 4

The Deep Optimisation Algorithm

In this chapter, the Deep Optimisation algorithm (DO) is detailed, and the motivation for its algorithmic principles is discussed. Specifically, the idea of transforming the solution representation and subsequently the neighbourhood of a solution to enable search to continue in a reorganised space. The DO algorithm develops from the understanding of rA-G discussed in chapter 3 in both the method of model-informed search and model construction. First, this chapter introduces the concept of Deep Optimisation. Then, using the autoencoder model architecture, we detailed the implementation of the deep optimisation algorithm.

In the final section of this chapter, a comparison between the Model-Building Optimisation Algorithm (MBOA) used in this thesis and DO is made. Chapter 2 identified multiple algorithms that use machine learning models in the context of evolutionary computing. However, the choice of model and how the model is used to inform search differs between the algorithms. The comparison discusses how all the algorithms approximate transforming the neighbourhood of a solution and then updating a population of solutions by exploring the reorganised space. In doing so, we clarify the contribution DO makes to evolutionary search and create hypotheses for the types of a problem structure that will categorically differentiate the performance between these algorithms, i.e., what one algorithm can do those other algorithms cannot. The performance evaluation is performed in chapter 5.

4.1 The Deep Optimisation Algorithm

DO is inspired by the process of Evolutionary Transitions in Individuality observed in biology (West et al., 2015). Specifically, evolutionary processes operating over multiple levels of an organisation have the characteristic that multiple individuals at one level of organisation form associations that result in a new evolutionary unit at a higher level of organisation (Maynard Smith and Szathmáry, 1997). These new evolutionary units then become subject to an evolutionary process. Further, these new units can form relationships to create further levels of organisation. For example, unicellular life transitioned into multi-cellular organisms.

An evolutionary unit is an entity subjected to variation and selection and is initially independent of other evolutionary units. In terms of optimisation, an evolutionary unit represents any unit of variation — a partial solution (Watson, 2005) - that can be selected and combined with others to form a whole solution. For example, a change made to a single solution variable is the lowest level evolutionary unit, and a change made to multiple solution variables simultaneously (a building-block) is a higher-level evolutionary unit. At the start of the DO algorithm, each solution variable is a separate evolutionary unit. A transition describes a transformation process that induces higher-order evolutionary units. The transformation performs a coordinated restructuring of a single-unit change made to a solution, enabling a single-unit change to perform a simultaneous change to multiple solution variables before selection acts. Consequently, evolution is rescaled to operate at a higher level of organisation that facilitates movements between solutions that were far apart in the original solution space (Watson et al., 2011b).

This idea of rescaling the search process from searching combinations of bits to searching combinations of modules at multiple levels of an organisation familiar in many areas of engineering and in genetic algorithms (Goldberg, 2006; Mills and Watson, 2011). However, building-block ideas in genetic algorithms were inspired by crossover, which requires the need to represent the linkage of variables as neighbours. This limitation motivated the development of MBOAs and, in turn, their departure from the biology that inspired genetic algorithms. DO takes a different inspiration — specifically, from the Evolutionary Transitions in Individuality and not crossover. Each new level of representation in DO represents the formation of new evolutionary units that in turn becomes subject to evolutionary processes. These new units can then form relationships to create further levels of organisation as the next level is learned, and so on, in the analogue of successive hierarchical transitions in individuality (Watson and Szathmáry, 2016). Thus, a solution initially represented as individual units (original problem variables) is transformed into a representation of high-level units which in turn transforms the unit of selection - an entity which selection can act on (Gardner, 2015) - to higher orders of organisation.

The term transition is used in DO to describe the process of inducing higher-order evolutionary units from the existing set of lower-level evolutionary units. The components that make up the transition processes are:

1. Discovering how units combine — providing a signal for learning a compressed representation.

2. Inducing a compressed representation of the current level of organisation — rescaling the units to a higher-order.

3. Rescaling the process of variation and selection to operate using the higher-order units — to search in the space defined by the compressed representation.

The following subsections describe these three components. For clarity, note that the optimisation process (evolutionary search) is separated from the learning process; the evolutionary process is not used to evolve a neural network as in NeuroEvolution (Stanley and Miikkulainen, 2002). In DO, the compressed representation is induced (by standard machine learning methods) from information discovered by the evolutionary search using the lower-level organisation, and the results are used to instantiate a new evolutionary process at the higher level of representation. Also, note that, although a case can be made that biological Evolutionary Transitions in Individuality naturally implement a form of DO (as is the inspiration) (Watson and Szathmáry, 2016), here the DO implementation uses abstract machine learning methods that do not intend to model that biology in a direct way.

To aid in the understanding of concepts discussed in the next sections, we rely on the analogy of learning a language. Intuitively, we might learn to use a language by first recognising the combinations of sounds that make up common syllables, then combinations of syllables that make up simple words, and so on through phrases and sentences. With each rescaling of the representation, we are able to construct meaningful expressions at a higher level of organisation. This, in turn, provides a better signal for us to learn the next level of structure. Only with learning skills at all levels of organisation is it possible for an author to learn how to construct an effective story; search in the space of individual letters would be useless even if guided by an accurate objective function. Deep Optimisation uses a similar intuition to learn multiple levels of representation, making it increasingly easy to construct high quality solutions to an optimisation problem, as it learns how to combine existing units together in useful ways.

4.1.1 Discovering how lower-level units combine:

In familiar learning tasks, such as classification and regression tasks, a model is learned from a data set. In an optimisation problem, no such data set is provided. We aim to learn information about the problem structure (which problem variables 'go together'), but this information is only implicit in the problem's fitness function (or objective function). To create a data set from which this information can be discovered, we must first gain some information by 'probing' the problem. In DO, this is provided by using a local search process (a simple evolutionary process) where a fitness function guides the search process to locally optimal solutions. Although this process is uninformed (knows nothing about the problem structure except that provided by selection), a distribution of solutions from different starting positions can be used as a training set to extract useful information about the problem structure.

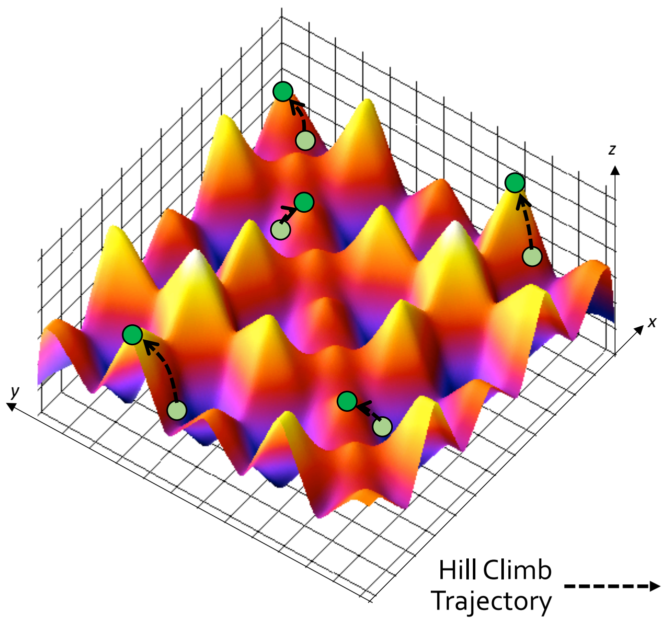


FIGURE 4.1: Hill-climbing in a search space can efficiently exploit local information about the problem structure but can easily become trapped at a sub-optimal solution (a local optimum). The search process can be repeatedly reset to random start points to explore different regions of the search space and provide a diverse distribution of locally optimal solutions.

Hill-climbing (also known as a "(1+1) evolutionary algorithm") is a simple evolutionary process sufficient for this. Figure 4.1 illustrates a fitness landscape of an optimisation problem. The fitness landscape is unknown for most optimisation problems and is used here only to help demonstrate the idea. The x and y axes represent the solution space and the z-axis represents the fitness of each solution. Hill-climbing is an efficient method to exploit local information about the search space but is susceptible to becoming trapped at sub-optimal solutions (a locally optimal solution). To escape a local optimum, a solution can be reset in a new but random region of the solution space. However, the solution is likely to become trapped at a different local optimum. In complex problems, the likelihood of finding a globally optimal solution via resets becomes exponentially small as the number of problem dimensions increases. Using our language analogy, this would process would be the same as attempting to write a story by searching in the space of individual letters.

Instead, an alternative approach to escaping a local optimum is to make a large change to the solution, such as a multi-variable substitution in a binary problem (or make word substitutions in the language analogy). Allowing multi-variable substitutions effectively transforms the neighbourhood of a solution such that solutions that were initially far apart become neighbours. A multi-variable substitution could be performed by making a random change to a sub-set of the solution variables (i.e., a group of random letters).

However, the likelihood of this producing an adaptive change vanishes as the size of the multi-variable substitution increases. Instead, one requires a multi-variable substitution that is *informed*, i.e., has knowledge of the problem structure and is consequently more likely to improve the fitness of a solution by enabling a large but coordinated change. However, the type of multi-variable substitution that will be adaptive is dependent on the problem instance — and it is this information that DO will learn.

A distribution of locally optimal solutions, found by searching combinations of units at the current level of organisation, contains combinations of variables that work well together to some degree (i.e., produce a word). The dimensionality of the variation observed in this distribution is less than the original dimensionality of possible variation. Consequently, a dimensional compression of the distribution of solutions can be performed to induce higher-order units of variation — extracting features of variation that appear in the distribution of locally optimal solutions. This much is a fairly standard approach in MBOAs (although DO uses the technique of learning from a distribution of locally optimal solutions (Iclanzan and Dumitrescu, 2007) rather than the more common practice of using a selected subset from a population of randomly generated solutions). But importantly, DO represents this compression using a standard autoencoder, enabling well-developed machine learning techniques to be applied and permitting multiple levels of deep structure to the learned.

4.1.2 Inducing a compressed representation

Learning a compressed representation of a distribution of solutions found at the current level of organisation permits a transformation of the space of solutions. The dimensional reduction will represent the dataset by extracting features representing the salient units of variation in the dataset. For instance, a regular relationship between variables will be extracted and represented by a higher-level representation variable. In our language analogy, this is the combination of letters forming words and thus the degrees of freedom are reduced from individual letters to individual words. Consequently, differences between solutions are described as differences between features (words rather than letters). Therefore, initially, far apart solutions (differing in the values of multiple variables) may become neighbours in the feature representation, i.e., differ by a single higher-level feature.

Figure 4.2 illustrates the corresponding effect on the fitness landscape by transforming the representation of a solution into a set of features. Relative to the feature (compressed) representation of a solution, fitness differences will be due to differences between features (unlike the original representation level where differences in individual variables cause fitness differences). Note, whilst the illustration shows a transformation of the fitness landscape, DO is not learning the fitness landscape nor a compressed representation of the fitness landscape. DO is learning a compression of the variation, which

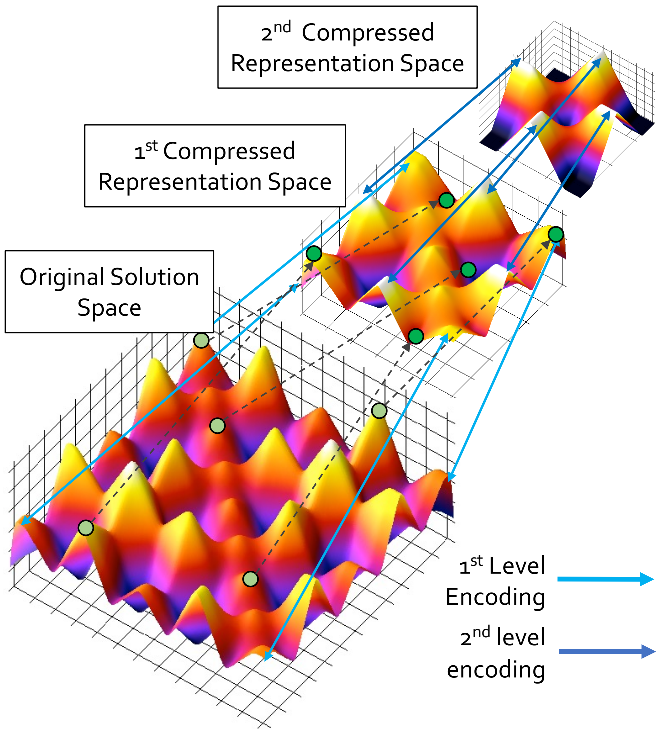
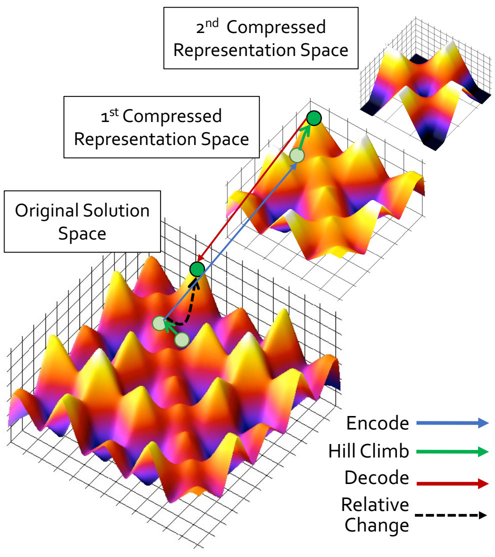

2nd Compressed Representation Space
1st Compressed Representation Space
Original Solution Space
1st Level Encoding
2nd level encoding

FIGURE 4.2: Learning a compressed representation of the solutions found at the lower-level organisation induces a higher-order search space. Solutions that were originally far apart appear closer together in the reorganised space. The induced space contains macro-scale local optima where local search in the compressed solution space can also become trapped at sub-optimal solutions. This space, in turn, can be compressed, constructing a multi-level representation of the solution space.

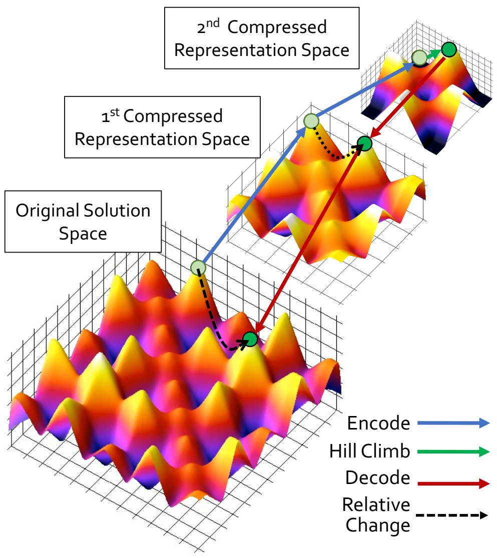
corresponds to the reorganisation of the neighbourhood of a solution and, consequently, the fitness landscape organisation.

4.1.3 Searching in the compressed representation:

The compressed representation allows for search to continue in a reorganised space. This is demonstrated in Fig. 4.3. A solution is first optimised in the original search space. Once at a local optimum, the solution is trapped, and hill-climbing can no longer be performed at this level of organisation. Next, the solution is encoded to the first compressed representation. Hill-climbing can be resumed at this level of organisation by changing individual features — local search at the compressed representation level. Specifically, a local variation is performed by making a change to a feature representation and then decoding back to the original level of organisation. Relative to the solution representation, the change appears as a large, coordinated change to the original problem variables — yet at the higher-order representation, the change was only a local change. The higher-order substitution enabled the solution to escape the local optimum and 'jump over' a fitness valley. Consequently, at the compressed representation, hill-climbing can continue to find superior solutions that may otherwise be pathologically difficult to find at the original level of organisation (illustrated in Fig. 4.3(a)). Referring to the language analogy, searching in the space of words will be able to find better solutions to our story more easily than searching in the space of letters.


2nd Compressed Representation Space
1st Compressed Representation Space
Original Solution Space
Encode
Hill Climb
Decode
Relative Change

(a) Hill-climbing in the original space gets trapped at a local optimum. The solution is then encoded to the compressed representation, where hill-climbing can resume. The change in the compressed space is decoded as a large but coordinated change in the original space. Consequently, enabling search to escape the local optimum.


2nd Compressed Representation Space
1st Compressed Representation Space
Original Solution Space
Encode
Hill Climb
Decode
Relative Change

(b) Hill-climbing in the deeper representation is required to escape the local optimum at the 1st compressed representation. This makes a large, coordinated change in the 1st layer space and an even larger coordinated change in the original space.

FIGURE 4.3: Hill-climbing in a higher-order representation can result in a large and coordinated change in the original solution representation. This can be achieved by encoding a solution to the higher-order representation, making a local variation in this new space, and then decoding back to the original representation.

4.1.4 Successive Transitions in Individuality

In the language analogy, searching in the space of words will only get us so far. It may lead us to identify phrases and sentences, but it remains incredibly challenging to write a story. In turn, searching in the compressed solution space can also become trapped, i.e., solutions that differ by a single higher-level feature do not improve the current solution, and therefore the problem contains macro-scale local optima (Fig. 4.3(a)). To escape a local optimum at the macro-scale, an additional compression is learned. By encoding to the second layer of organisation, hill-climbing can again be performed. Illustrated in Fig. 4.3(b), the local change at the second level of organisation is decoded back, performing a large and coordinated change to the first layer of organisation and an even larger but still coordinated change to the original solution representation (e.g., performing a sentence substitution at the second layer, which produces a large but coordinated change to many words at the first layer, which provides an even larger but still coordinated change at the original layer of letters). Further, the idea of inducing a compressed representation can be performed indefinitely, constructing a multi-level compressed representation of the search space.

Each level of organisation induces a reorganised representation of a solution, which in turn transforms the neighbourhood, and therefore fitness landscape, of the solution space, as illustrated in Fig. 4.3(b). The evolutionary search process does not change; it is just rescaled into a new representation. Specifically, each solution is updated by hill-climbing at the compressed representation finding locally optimal solutions in the compressed representation. The distribution of solutions found at the compressed representation level will contain information about how the units of variation at the compressed representation level combine to improve the fitness of a solution. An additional compressed representation can be induced, compressing the units of variation at the first layer of organisation to an even higher-order unit of variation — providing large, coordinated variation when decoded into the first layer of organisation.

The idea of searching in reorganised neighbourhoods shares similarities with the well-known Variable Neighbourhood Search method (Hansen et al., 2010), where a fixed set of neighbourhoods are manually defined before searching in them. However, in DO, a compression of a neighbourhood is induced from solutions found in a larger neighbourhood, providing a new and intelligent space to search in without requiring domain knowledge 'a priori'. Therefore, DO can be intuitively described as performing a local search in reorganised neighbourhoods induced by capturing information from solutions found to be adaptive in lower-order neighbourhoods.

4.1.5 The Autoencoder model

The modelling requirements to emulate the process of successive transitions in individuality require a model capable of dimensional compression using information only provided by the data points (unsupervised learning); a model capable of constructing a deep representation one organisation level at a time; and a model capable of transforming between different representation levels. The autoencoder model provides a suitable model space to explore the idea of successive transitions in individuality.

The autoencoder model is a feed-forward neural network separated into two networks: an encoder network that transforms an input to a hidden representation called an encoding. The decoder network transforms the encoded representation of the input back to the original input representation space. The encoder network provides the transformation of the original solution representation to a higher-order representation, and the decoder network provides the transformation of a higher-order representation to the original solution representation. The autoencoder architecture can induce a lower-dimensional representation, capturing the prominent factors of variation in a dataset without using labelled data. The training objective encourages the hidden representation to capture the input features that maximise reconstruction or minimise information loss. Consequently, the features extracted represent the directions of variance in the dataset, i.e., how solution variables co-vary in the dataset.

4.1.5.1 Deep Autoencoder

In the analogy of ETI, a transition describes the process of inducing a higher-order unit that allows for selection to act on. In inducing a higher-order evolutionary unit, variation and selection act on these units to find new relationships (relationships with this higher-order unit did not exist before its emergence). The dataset does not have information about how higher-order units can combine further to create even higher-order units - a deep representation of variable combinations that contribute to the fitness of a solution. Instead, DO must discover these relationships by inducing higher-order units and then searching in the space of higher-order units to find relationships between these units allowing for another transition to occur. Therefore, DO uses a layerwise construction process that borrows inspiration from the idea of greedy layerwise pre-training (Hinton and Salakhutdinov, 2006) to construct a deep representation.

In greedy layerwise pre-training, a single layer of an autoencoder is trained, which induces a compressed representation of the dataset. Then, a different single-layered autoencoder is trained to learn a compressed representation of the compressed representation learned by the first autoencoder — the input to the second autoencoder is the hidden layer response of the first autoencoder model. This method proceeds to construct a deep representation and is often referred to as layerwise stacking (Vincent et al., 2010), where multiple single-layered autoencoders are stacked one-by-one, with the input to one autoencoder being the hidden layer response of the preceding autoencoder. After training each layer, all networks are connected to construct a deep feed-forward neural network and then trained end-to-end to complete the supervised task - a fine-tuning process. The idea is that the compressed representations induced before fine-tuning initialised the deep network into a more suitable region of the model parameter space than a random initialisation, allowing fine-tuning to find a more optimal model than starting from random initialisation.

DO uses layer-wise building (Jain and Seung, 2008) method rather than layer-wise stacking (Vincent et al., 2010)). At first, the autoencoder model is initialised with a single hidden layer and trained. After that, a new hidden layer is added to the autoencoder model. In layerwise building, the autoencoder is trained to learn a reconstruction of the input. Therefore, all model parameters are updated during training (layer one and layer two weights), whereas in layerwise stacking, only the parameters of the second layer are updated. In this case, the lower levels are now influenced and encouraged to learn the higher-level compression - similar to the fine-tuning phase used in greedy layerwise pre-training. In turn, the lower level may no longer represent a good compression of the dataset. Instead, the lower levels are transforming the representation to enable the next layer compression to be more straightforward. Consequently, layerwise building allows for multiple layers to perform the compression.

The mechanism for transitions in DO is inspired by the same mechanism described above. However, in DO, information about how induced features interact at higher levels must be discovered. Therefore, DO uses an additional step in the layerwise learning process that performs an evolutionary search at the representation level to provide a dataset that contains information about how units at the current representation level interact. Then, a new layer is used to learn and extract features from this representation level using the dataset. For instance, initially, there is no information in random solutions to induce a compressed representation. Variation and selection must first be performed at the original level of organisation. Then the 1st compressed representation can be induced using this distribution of solutions found. However, at the 1st level of organisation, there will be no information about how the induced units (features of the dataset) combine, as selection has not been able to act on these new emergent units and therefore no information to learn a 2nd compression. To perform a 2nd compression, variation and selection is performed at the 1st compressed organisation providing a dataset that has lower dimensionality than the 1st compressed representation. A deep representation of the solution organisation is constructed by repeatedly performing a transition from search at one level of organisation, inducing a higher-order organisation and then continuing search using the higher-order organisation.

4.1.5.2 Encouraging a representation

Extracting, separating, and representing features, also in a hierarchical organisation, is the focus of representation learning (Bengio et al., 2013). Representation learning can be described as learning a representation that makes interpreting the original dataset more straightforward than at the original representation (Brahma et al., 2015). For instance, modelling or computing in the original space can be difficult due to the high dimensionality or complex entanglement of important relationships. Therefore, transforming the representation into a more straightforward, more organised space whilst minimising the information loss allows for more efficient computation for the desired task (Salakhutdinov and Hinton, 2009). What constitutes a good representation is dependent on the computational task required. For supervised learning tasks, the desired output for a given input is known. Subsequently, the signal for how a reorganisation of the input can simplify the computation of the output can be available. However, in unsupervised learning tasks, such as learning a compressed representation, it is not guaranteed that the induced representation will be useful or relevant to the application (Goodfellow et al., 2016). In the case of DO, we desire a compressed representation that extracts and separates the salient factors of variation in the data set.

It is possible to encourage particular properties of a representation by using regularisation techniques - i.e., applying particular inductive biases to encourage a favoured organisation that is more 'useful' for our application. For instance, a representation that provides a poor separation of modules or a poor representation of the relationships within a module will affect the capability of extracting information from the model to inform search. Many regularisation techniques can be applied to the autoencoder model to encourage particular properties in the induced representation. The types of regularisation explored in this thesis are the following:

Constraining the dimensionality of the hidden representation(s) — the dimensionality ratio is the difference between the dimensions of two layers. Having a dimensionality ratio of less than one creates an encoder network with decreasing dimensionality of the hidden layers (an under-complete representation). The autoencoder is forced to learn a compressed representation of the input by having a lower dimension of units at the hidden layer than the number of input units (or previous layer). Often this arrangement is referred to as a 'vanilla' autoencoder (Hinton and Salakhutdinov, 2006). As the objective function maximises the reconstruction of the input from the compressed representation, the hidden layer is encouraged to extract the most salient features of the input. However, a large compression ratio reduces the model's capacity and risks not capturing all important relationships. Alternatively, a weak compression or expansion increases the risk of learning the identity function.

L1 regularisation: Encourages a representation that minimises the total sum of the absolute weights. Weights (model parameters) that do not significantly contribute to the network's output to satisfy the objective function are penalised (significance controlled by the lambda term). The L1 regularisation causes a bias towards a sparse representation where neurons respond to few inputs.

L2 Regularisation: Encourages a representation that limits the influence of any one parameter in the model. The L2 regularisation penalises large weights compared to small weights. Therefore, a more efficient representation uses a number of small parameters rather than a single significant parameter to produce the desired output. Consequently, it minimises the response to noise.

The L1 and L2 regularisation are added to the training cost function. Other regularisation techniques such as sparsity regularisation (Ng et al., 2011) and contractive regularisation (Rifai et al., 2011) also contain interesting properties. However, exploratory experiments failed to improve on the result obtained using L1 and L2 regularisation. Therefore, this is left as future work.

Denoising: Encourages the representation to capture the relationships between units in order to reconstruct the missing or corrupted input. Denoising is applied during training. The autoencoder is trained to reconstruct the original input from a corrupt version of the input. Specifically, the input to the autoencoder is corrupted (a noisy version of the input), and the reconstruction error is calculated between the decoder output and the original (non-corrupt) input. For the autoencoder to reconstruct the original input, the autoencoder needs to learn the relationships between variables, as corrupt or missing data must be 'filled in' during reconstruction from the value of other variables. The corruption is applied directly to the input and can take various forms (Vincent et al., 2008, 2010). Dropout is a more general application of noise (Hinton et al., 2012b). Dropout removes any signal from an input unit. It, therefore, encourages the model to learn how to generate the value of the missing unit from the values of other input units. As denoising encourages learning relationships between variables, the risk of learning an identity function is reduced. Therefore, it reduces the sensitivity of the compression ratio. Consequently, expansions can be used without concern for learning the identity function. Denoising also encourages a robust representation: a small variation to the input is reconstructed back to the original input. This must be accounted for when applying variation at the hidden representation as small changes are likely to have no meaningful change at the solution representation level.

Variational: The Variational Autoencoder (Kingma and Welling, 2013) is a well-known probabilistic interpretation of the autoencoder model that applies a regularisation pressure to encourage the hidden space to be amenable to random sampling. Specifically, the hidden representation is encouraged and interpreted as a probability density function and can be sampled to generate outputs. In turn, decoding from the hidden representation is likely to be an example of the data distribution used to train the model. As such, a Variational Autoencoder is more in line with Estimation Distribution Algorithms and have been previously explored (Santana, 2017). However, for Multi-Scale Search, it is not necessary to have a complete solution generatable from the hidden representation. Instead, Multi-Scale Search requires the extraction of salient features only - extraction of the relationships between variables whilst not considering the randomness. Further, the regularisation used to encourage such a representation has been shown to reduce the representation capacity (Kumar et al., 2017; Ghosh et al., 2019) and to construct deep Variational Autoencoder remains ongoing research (Zhao et al., 2017). We, therefore, omit Variational Autoencoder in our study.

4.2 DO Algorithm: Implementation Details

Chapter 3 showed that an autoencoder model was sufficient for inducing a representation that captured the modules of an optimisation problem. However, the process of model-informed search used by rA-G, which employed a method that extracted information from the parameters of the network, inspired by rHN-G and analogous to sub-structure search (Lima et al., 2006), was limited to a single-layered neural network. Further, rA-G failed to exploit a module's distributed representation — a module compression represented using more than one hidden neuron. DO overcomes the limitation of rA-G by using a model-informed search process that explicitly transforms the representation of the complete solution and transforms local changes at this higher-level organisation back to the solution representation to construct a higher-order substitution. Importantly this process is independent of the depth at which search is performed and consequently allows for inducing and searching in deep representations of a solution. Finally, the process of constructing a deep representation draws inspiration from the layerwise process introduced by Hinton and Salakhutdinov (Hinton and Salakhutdinov, 2006) and the evolutionary transitions in individuality. As discussed previously, information about higher-order units is discovered by searching in the space defined by lower-order units. Therefore, a layerwise construction is necessary as information about the higher-order units is not initially present in the data used to train the autoencoder model.

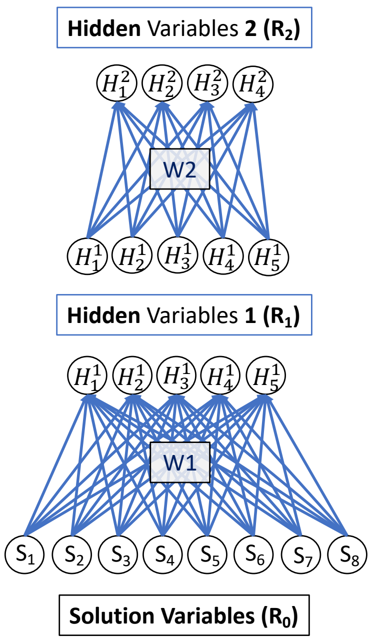
Deep Optimization (DO) uses local search as a simple evolution process to update a solution. Consequently, DO can be described as a local search algorithm that repeatedly transforms the neighbourhood of a solution by inducing a compressed representation of solutions found from performing a local search at the current level of organisation. A transition describes the process of inducing a compressed representation of a solution and then rescaling the local search to perform in the new compressed representation. This process iterates, repeatedly transforming the representation of a solution and consequently constructs a multi-level representation. DO uses the autoencoder model to emulate the transition process and Model-Informed Variation (local search in the compressed representation) due to our evidence of superior performance in chapter 3. Note, DO is not limited to a particular evolutionary search process. For instance, the process of crossover or generating new variation via random recombination could also be used. It remains for further work to understand the capability that rescaling different evolutionary processes can provide.

Algorithm 4: Deep Optimisation

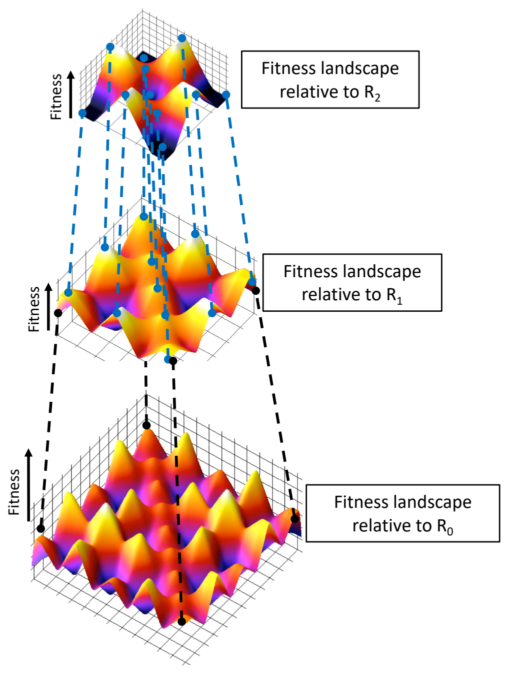
```
Initialize: Population of Solutions;
Initialize: Empty Autoencoder model (0 hidden Layers);
T=0;
while T < Maximum Number of Transitions do
    for Solution in Population do
        while Optimizing Solution do
            Apply variation and selection to Solution using the Autoencoder Model
              and Model-Informed Variation at Layer T
    // apply transition
    if Number of Hidden Layers in Autoencoder <= Maximum Model Depth then
        Add another hidden layer to the autoencoder model;
    Train autoencoder (all layers) using the population as training set;
    T=T+1;
```

The Deep Optimisation algorithm is detailed in Algorithm 4. Initially, the autoencoder model has no hidden layers (an empty model), and the population is initialised using the random uniform distribution to provide good coverage of the solution space. Each solution is then updated by performing local search at the solution representation defined by the model. At first, the autoencoder model is empty, therefore a solution is updated by performing local search at the solution representation level (single-variable substitutions). The "While Optimising Solution" loop refers to the condition at which search continues for a solution. In all cases for DO, this loop is stopped when Model-Informed Variation cannot perform a change that improves the fitness of the current solution. For example, when a single-variable substitution applied to any solution variable does not improve the fitness of the current solution the solution is said to be optimised, and subsequently the "While Optimisation Solution" loop is stopped.

The "For solution in Population" loop produces a distribution of locally optimal solutions relative to the neighbourhood defined by the representation of the solution. After updating each solution in the population, a hidden layer is added to the autoencoder model, initially producing a single-layered autoencoder model. The model is then updated via training using the distribution of solutions as the training set. The hidden layer represents a transformed representation of the original problem variables that captures the salient factors of variation in the training set (variation between locally optimal solutions). This completes a transition. The process repeats, with the distribution of solutions now being updated by performing local search at the first hidden layer via a process called Model-Informed Variation detailed in section 4.2.3. Intuitively, Model-Informed Variation is a process that performs local search at the compressed representation. Model-Informed Variation is repeatedly applied to each solution, resulting in a distribution of solutions that are locally optimal in the neighbourhood defined by the first hidden representation. Therefore, the distribution of solutions now contains information about how features extracted from the previous distribution can combine to form higher-order features. A new hidden layer is now added to the autoencoder (a total of two hidden layers), providing the capacity to learn a further compressed representation. The model is then updated using the distribution of solutions found by performing Model-Informed Variation at the first hidden layer representation. The new hidden layer can extract information about how the low-level features in the first hidden layer can combine to improve the fitness of a solution. Or, the lower level can be used with the new hidden layer to learn a complex compression that was not previously achievable using only a single hidden layer. This process continues to repeat, inducing multiple compressed representations of a solution and then continuing to search in the compressed space to find higher-order solutions. Each hidden layer transforms the representation of the solution by capturing features from the solutions found at the lower-level organisation. In turn, each compressed representation layer in the autoencoder model reorganises the effective neighbourhood of a solution as illustrated in Figure 4.4.



(a) Two layered encoder network


Fitness
Fitness landscape relative to R2
Fitness
Fitness landscape relative to R1
Fitness
Fitness landscape relative to R0

(b) Illustration of the transformed search space caused by the feature extraction.

FIGURE 4.4: The encoder network transforms the input to a hidden representation that represents features of solutions found in the search space below. The feature representation effectively transforms the neighbourhood of a solution where the neighbourhood is reorganised into differences between features rather than the original problem variables.

4.2.1 Autoencoder Details

The autoencoder model consists of an encoder (E) and decoder (D) network that transforms solution (S) to a hidden representation (H) and then back to the original input representation. Specifically, the reconstruction ($_\mathrm{r}$) of a solution (S) is calculated as $S_r = u(X) = u(D(E(S)))$, where S and S_r are the solution and solution reconstruction respectively, X is the output from the decoder and u is a unit step function. Specifically,

$$u(X) = \begin{cases} 0, \text{ if } X < t \\ 1, \text{ if } X \geq t \end{cases}, \tag{4.1}$$

where t is a threshold value set to the middle of the activation response used by the decoder output units.

The encoder network is composed of L hidden layers, each transforming the lower representation H^e_{l-1} into a high-order representation H^e_l, where H^e_l represents the activation response of layer l in the encoder network ($^\mathrm{e}$). This transformation is achieved by using the hyperbolic-tangent nonlinear activation function (f) on the sum of the weighted inputs. Specifically, $H^e_l = f(W_l H^e_{l-1} + b^e_l)$ where W_l and b^e_l are the weights and bias respectively for encoder layer l. At the first hidden layer ($l = 1$), $H^e_{l-1} = S$ (the encoder input). During training, dropout is used to corrupt the input S. Specifically, at the first hidden layer $H^e_{l-1} = N(S)$, where N is the dropout function which uses a drop out probability parameter to control, for each input unit, the likelihood of the input unit value being ignored.

The decoder network generates a reconstruction, X, from the hidden representation at H^e_L, often referred to as the bottleneck layer for the autoencoder. The decoder network contains the same number of layers as the encoder network. Each layer of the decoder transforms the hidden representation H^d_l back to a lower-order representation H^d_{l-1} via $H^d_{l-1} = f(W'_l H^d_l + b^d_l)$, where W'_l is the transpose of the encoder connection weights W_l (the weights are tied), b^d_l is the decoder bias and H^d_l is the decoder (d) activation response at layer l. At $l = L$, $H^d_L = H^e_L$ (the input to the decoder network is the output of the encoder network) and at $l = 1$, $H^d_1 = X$ (the output of the decoder network is a reconstruction of the input S).

The weights are initialised using the uniform distribution $U(-0.01, 0.01)$. Both the encoder and decoder biases are initialised to 0. DO uses the TanH activation function. Therefore, at training, a binary solution $[0, 1]^N$ of size N is transformed into the discrete representation $[-1, 1]^N$. The decoder output is in a continuous space and converted to a binary representation using the unit step function $S_r = u(X)$ where S_r is the reconstruct solution using the autoencoder model.

The size of each layer is a tuneable parameter. The compression factor, r_l, is the ratio between the number of input to output units at a given layer. Specifically, $N_l = \lceil r_l N_{l-1} \rceil$, where N_l and N_{l-1} are the number of units at the input and output of the layer l. A compression factor of $r_l = 0.8$ was found to be a good balance between encouraging a compressed representation whilst providing sufficient capacity to learn the compression.

4.2.2 Transition

The novel contribution of DO is the transition process, illustrated in Figure 4.5. Specifically, the induction of a new solution representation that consequently reorganises the neighbourhood of a solution. Induction is performed using the autoencoder model and a distribution of selected-for solutions - the population of solutions after local search at the current level of organisation has been performed. The induced compressed representation then allows for selection to be rescaled to a higher order of organisation (achieved using Model-Informed Variation). A transition is a combination of the following three procedures:

1. An additional hidden layer H_{n+1} is added to the autoencoder with n layers.

2. Previous learned parameters are retained, and the current distribution of solutions are used as training examples to update the autoencoder model (W_1 to W_{n+1}).

3. Model-Informed Variation is changed from operating on H_n to operating on H_{n+1}.

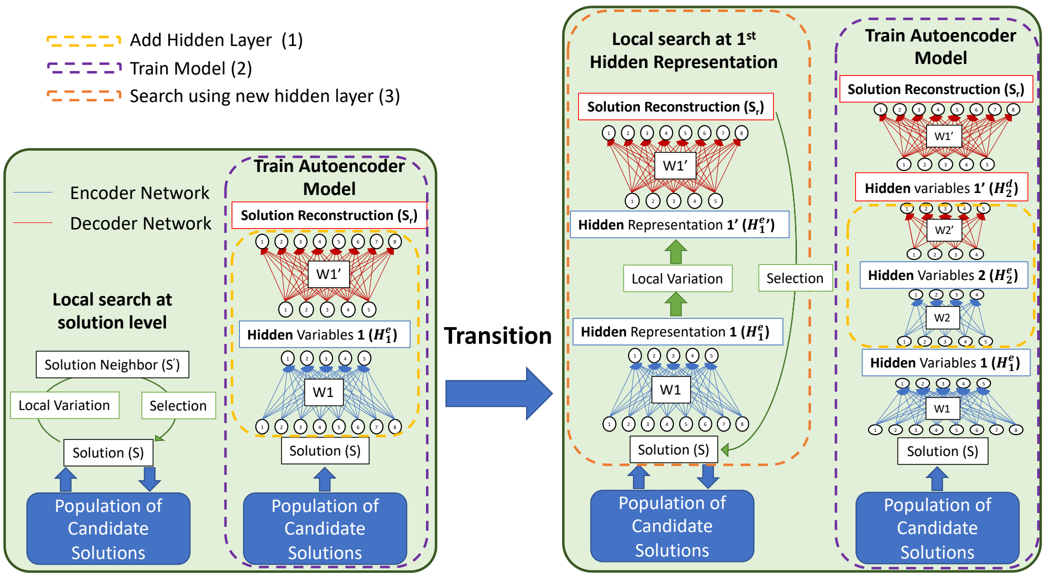

Add Hidden Layer (1)
Train Model (2)
Search using new hidden layer (3)
Encoder Network
Decoder Network
Local search at solution level
Solution Neighbor (S')
Local Variation
Selection
Solution (S)
Population of Candidate Solutions
Train Autoencoder Model
Solution Reconstruction (S,)
W1'
Hidden Variables 1 (H_1^e)
W1
Transition
Local search at 1st Hidden Representation
Hidden Representation 1' (H_1^e')
Hidden Representation 1 (H_1^e)
Hidden variables 1' (H_2^d)
W2'
Hidden Variables 2 (H_2^e)
W2

FIGURE 4.5: Schematic of the transition process performed by DO. First, the population of solutions is updated by performing local search at the current level of organisation (initially, the solution representation $L = 0$). A hidden layer, H_1 is then added to the autoencoder model (1). The autoencoder model is then trained using the distribution of solutions found (2). After training, a transition occurs, and each solution is updated by performing local search at the hidden representation (H_1) of the solution (3) - a process we call Model-Informed Variation. The process repeats: finding a distribution of locally optimal solutions relative to the current level of organisation, adding a further hidden layer, and updating the model, and then performing local search at the new hidden layer to update the solutions.

The addition of a single hidden layer to the autoencoder model (procedure 1) is analogous to the approach introduced by Hinton and Salakhutdinov (2006) for training deep neural networks. The layer-wise procedure is important for learning a meaningful representation of a solution at each hidden layer. Information about higher-order relationships is discovered in DO by searching using lower-order relationships. Consequently, DO learns from its dynamics (Watson et al., 2011a). There may be many possible mappings in which the problem structure can be represented. Thus, deeper layers are not only a representation of higher-order features of the problem structure but the ability to learn these features depends on the learned lower-level features. Therefore, if the lower layers do not contain a useful representation, attempting to train or perform Model-Informed Variation at deeper layers could be ineffective. Therefore, inducing and searching one layer at a time allows lower-order relationships to be reinforced or discouraged due to selection accepting or rejecting the relationship in subsequent distributions, respectively.

Training updates the model parameters of the autoencoder model using a distribution of solutions as a training set (procedure 2). The training loss function is an additive combination of the reconstruction error (between X and input S), L1 and L2 regularisation. λ and γ are the regularisation coefficients used to control the strength of the parsimony pressure applied to the model. Training is performed using the back-propagation algorithm and Adam optimiser to minimise the training cost function, Equation 4.2 where n is the number of training examples used in a batch. Recall S is the solution, W is the set of weights with W_l representing the weights for layer l in the network, N is the noise applied to the solution (for which dropout is used in this thesis), E is the encoder and D is the decoder.

$$loss(S, W) = \frac{1}{n}\sum_{i=1}^{n}(S_i - D(E(N(S_i))))^2 + \lambda\sum_{l=1}^{L}|W_l| + \gamma\sum_{l=1}^{L}||W_l||^2 \tag{4.2}$$

Note, the fitness value of a solution is not used for training. Concretely, DO is not modelling the fitness landscape to predict regions of higher fitness. Instead, DO learns a compression of the variation in the dataset.

This transition procedure (procedures 1 - 3) is performed repeatedly until the maximum depth of the autoencoder is reached, at which procedure 1 is no longer performed. The addition of a hidden layer to the model increases the network's capacity to learn a compressed representation of the lower-level relationships already identified. The low-level relationships can contain valuable information required later in the optimisation process. By continuing to represent this information gives DO the potential to exploit lower-order information at any point during optimisation. For DO, it is natural to use the lower-level information captured by the autoencoder. A new, deeper layer can learn to combine lower-level features or induce a higher-order abstraction of the relationships. This is an important distinction from the other algorithm's that effectively construct a new model of the data distribution after each update to the population. For DO, this would be equivalent of reinitialising the autoencoder model parameters each time before training the model. We hypothesise that maintaining and using previous information (lower-order information) will provide a clearer signal for extracting higher-order relationships and facilitating search at multiple representation levels during optimisation.

4.2.3 Model-Informed Variation

Model-Informed Variation (procedure 3) is a method that updates a solution by making a partial substitution. Selection is then used to decide if the substitution is accepted or not. The substitution is one that moves the solution to a local neighbour in the compressed representation defined by the model. In chapter 3, the method developed for rA-G was inspired by the process used by rHN-G. However, in rA-G, the method was limited to a single hidden layer and could not make a correct substitution when multiple hidden nodes represent a relationship between variables. For DO, these limitations are overcome by using the encoder and decoder networks in a more natural form. Specifically, the encoder network transforms a solution into a compressed representation. A change is made to the compressed representation. Then, the decoder network transforms this change back to the solution representation. Here, we focus only on binary problems where each solution variable is a unit that takes an ‘on’ or ‘off’ value.

Initially, search starts at the solution level representation (no transition has occurred). Therefore, local search is initially applied at the solution representation level. In the case of a binary problem, local search is a single-bit substitution. Specifically, a solution S is updated by substituting the value of a single variable selected at random, forming S'. The fitness of S' is evaluated, and if $F(S') \geq F(S)$, where F is the fitness function, S' replaces S otherwise it is rejected. This is repeated until no single variable substitution (local neighbour) improves the fitness of a solution.

After a transition, Model-Informed Variation is then used to update a solution - the model is used to inform the substitution to make to a solution during a local search step (single unit change). Selection continues as before, deciding if the substitution improves the fitness of the solution or not (determining if the substitution is adaptive or not).

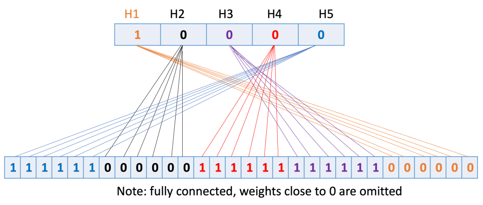
The algorithm for Model-Informed Variation performed by DO is presented in Algorithm 5. First, a solution is encoded to the higher-order representation using the encoder network, $H_l = E(S)$. The encoding represents a compressed representation of the solution. A change to the compressed representation ΔH is applied, forming H_l'. We explore different methods of ΔH. Next, we want to determine how the change at the compressed representation, ΔH, corresponds to the change at the original input representation, ΔS. We do this by using the decoder network to transform H_l and H_l' back to the solution representation. Specifically, $S_r = u(D(H))$ and $S_r' = u(D(H'))$ where u is the unit function that transforms the continuous output units of the decoder to binary units. Note, for non-binary problems, the unit function can be replaced with a problem specific operator to transform the decoder output into a solution - as explored in chapter 6. The model-informed substitution, ΔS, is constructed as the difference between S_r and S_r' and then applied to S, specifically $S' = S + \Delta S = S + (S_r' - S_r)$. We found that calculating ΔS relative to the reconstruction of S, S_r, was a more suitable method as the autoencoder model generally does not learn a perfect reconstruction. This completes the construction of a higher-order substitution to make at the solution level representation that is a local step in the compressed representation. The change to the solution is then evaluated, as before using selection. Specifically, if $S' \geq S$ the change is accepted; otherwise, it is rejected. The process of Model-Informed Variation is repeatedly performed until a local change at the hidden representation does not improve the solution. As DO can represent multiple representations of a solution, search can be performed at multiple levels to update a solution before using the solution to update the model. The number of layers to search in is defined in Algorithm 5 by *Model-Informed Variation Layers*.

Algorithm 5: DO - Solution update via Model-Informed Variation

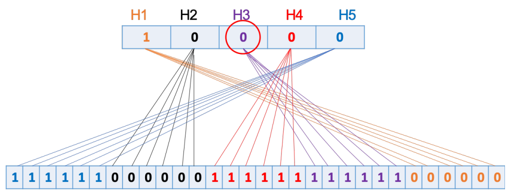
```
for $S$ in Population do
    while $S$ can be improved do
        for $L$ in Model-Informed Variation Layers do
            $H$ = Encode($S$,L)
            $H' = H + \Delta H$
            $S_r$ = u(Decode($H$))
            $S'_r$ = u(Decode($H'$))
            $\Delta S = S'_r - S_r$
            $S' = S + \Delta S$
            $\Delta F = F(S') - F(S)$
            if $\Delta F \geq 0$ then
                $S = S'$
```

An intuitive example of Model-Informed Variation can be described when the autoencoder model has learned the identity function. Specifically, a hidden node represents a single variable of the solution representation. Consequently, performing Model-Informed Variation at the hidden representation becomes equivalent to performing a local search at the solution representation – a change to a single hidden node causes a change to a single solution variable. Therefore, higher-order substitutions are made when the autoencoder model has learned a compressed representation and exploited this information using Model-Informed Variation. In this case, a change to a single hidden neuron would change multiple solution units simultaneously.

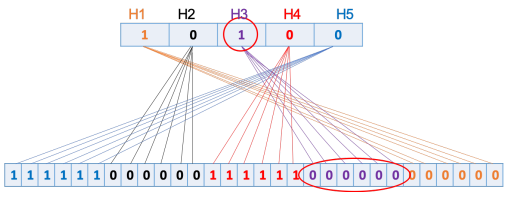
Figure 4.6 demonstrates how a local change at the encoded representation transforms local search to a high-order of organisation (simultaneous change to multiple variables at the solution level). The figure presents a single-layer network with tied weights. Lines between the solution and hidden representation indicate strong connections between the units. The encoder network transforms a solution to a compressed representation - corresponding to the solution's "location" in the compressed search space, Figure 4.6(a). A local change can then be made at this compressed representation - treating the space in the same manner as the original solution representation, Figure 4.6(b). The decoder network transforms the changed hidden representation back to the original solution space, which in turn causes a change to multiple variables at the solution representation to change simultaneously (in comparison to the original input), Figure 4.6(c). Therefore, the fitness of the modified solution and consequently the fitness effect of a local change at the compressed representation can be evaluated. Finally, this process is repeated, where the solution is re-encoded, a local change made at the compressed representation, and then decoded back to evaluate the effect of the higher-order substitution to the solution, Figure 4.6(d)



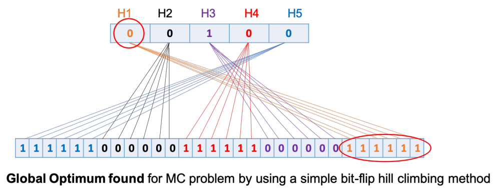
(a) Encoding of a solution from the population (a local optimum in the original solution space.)



(b) The third hidden neuron (H3), responds to fourth module.



(c) A change to H3 decodes to a change to the fourth module. Providing a simultaneous change to all variables in the building-block, enabling search to escape the local optimum and improve the quality of the solution


H1 H2 H3 H4 H5
0 0 1 0 0
1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1
Global Optimum found for MC problem by using a simple bit-flip hill climbing method

(d) The process is repeated, where a local variation applied to H1 decodes to a change to the fifth module

FIGURE 4.6: Illustration of how DO performs Model-Informed Variation and the subsequent effect it has on a solution for a problem with 5 modules (colour coded) or size 6 variables. The values of a module are all 1's or all 0's (the Modular Constraint problem). The autoencoder has learned a compression mapping, representing the two solutions for a module using a single hidden node. The network is fully connected; coloured lines indicate strong weights.

4.2.3.1 Local Change at Hidden Representation

The performance of Model-Informed Variation is dependent on the encoding of a solution to the higher-order representation, the change made at the higher-order representation, and how the change to the higher-order representation transforms back to the original solution representation. The methods we explore for calculating ΔH are as follows:

Substitution:

If an encoding does not provide a saturated response, i.e., the response of a hidden unit is close to 0, flipping may not be sufficient. Instead, increasing the response of an activation value can make a meaningful change (increasing the response of a feature). The assign method selects a single hidden unit at random and assigns either the minimum or maximum activation response. This method is the baseline method used by DO and unless otherwise specified, used in the experiments.

Encode:

For this method, we use the encoder network to inform which hidden nodes to change. Intuitively, the encoder network informs which hidden units respond to a solution variable. Therefore, if by selecting a solution variable to change, the encoder network can then inform which hidden neurons need to change in order to make the change to the solution variable. As the decoder weights are the transpose of the encoder weights, making the change at the hidden representation and then decoding will cause a change to the chosen solution variable and all other solution variables that co-vary with the chosen solution variable.

First, $\Delta H = E(S + \Delta S) - E(S)$, where ΔS is a single-unit substitution to the solution representation (the chosen solution variable). ΔH represents the sensitivity of the hidden units to a change to the solution variable. It was found that directly using ΔH is not always sufficient: the magnitude of the change at the hidden layer was generally weak. Therefore, we amplify the ΔH using a threshold function. Specifically, a random threshold is taken between the average and maximum magnitudes in ΔH. The threshold is then used to filter out hidden units (H_i where i represents the i'th hidden node in the hidden representation) with a response weaker than the threshold and therefore considered insensitive to ΔS. For hidden units that show a sensitivity (i.e., above the threshold value), the value is set to the minimum or maximum activation response. The choice of minimum or maximum activation value is dependent on the direction of the change calculated by the encoder. Specifically, if the response of a hidden unit is negative (moving from a positive activation response to a negative activation response), the minimum activation value is used, and vice versa. The algorithm for calculating ΔH by encoding is detailed in Algorithm 6.

Algorithm 6: Encoder informed change at hidden representation

```
$H$ = Encode(Solution,L);
$H^s$ = Encode(Solution+$\Delta S$,L);
$\Delta H = H^s - H$ ;                                   // Change at H
$a = \frac{1}{N_l}\sum_{i=1}^{N_l} |\Delta H_i|$ ;        // $N_l$ is number nodes in layer $L$
$z = \max(|\Delta H|)$ ;
$T = a + (z - a) \times U(0,1)$ ;                         // Threshold
$\Delta H = \begin{cases} sgn(\Delta H_i) - H_i & |\Delta H_i| > T \\ 0 & otherwise \end{cases}$
$H' = H + \Delta H$;
```

Whilst a more complex method compared to the substitution method, here we are interested in how the encoder information can be used to enhance search. Calculating ΔH using the encoder network helps overcome cases when hidden units share the same or similar response. For instance, if multiple hidden units have the same response to input units, then a substitution informed by making a change to only one of these hidden units will be suppressed by the other hidden units that do not want the input to change. However, all these hidden units will show a strong association with the same variable(s) and thus have a similar response to a change in a solution variable. By approximating which hidden units are responsible for the value of a single input unit, a variation to all these units provides a more robust method to making a sufficient change in H that will then decode to a meaningful change in S.

4.2.3.2 Evidence of Model-Informed Variation in DO

To demonstrate that Model-Informed Variation works at deep representations induced by the autoencoder model, we force DO to learn a deep representation of the problem structure for the Hierarchical If and Only If (HIFF) synthetic benchmark optimisation problem. In chapter 3 we showed that HIFF could by solved using only a single hidden layer, even though the problem structure has a multi-level organisation. Here, we force DO to learn the deep representation of the problem structure by using layerwise stacking where only weights at the deepest layer are trained (the weights of lower-levels are frozen during training). In this case, the first layer will learn how pairs of variables combine. The next layer then learns how these pairs of variables combine to form pairs of pairs of variables as the lower-level cannot change its weights to represent how four variables combine (this can only be achieved by the next layer). This occurs at each level, thus causing the problem structure of HIFF to be represented using a deep autoencoder model. Therefore, using the HIFF problem and constructing a deep representation using layerwise stacking demonstrates the capability of Model-Informed Variation to operate at the deep hidden layers. Figure 4.7.a shows the fitness trajectories for ten random solutions during optimisation. The dashed lines indicate a transition, five which occur. We observe that after each transition, DO can find solutions of higher fitness. DO achieves this by inducing a higher-order representation and transforming the variation to higher orders of organisation. Figure 4.7.b shows the variations accepted to a single solution before and after the first transition. Here we see that variation is rescaled from single-unit substitutions to double-unit substitutions. However, the solution becomes trapped as DO has only learned how pairs of variables combine and consequently only capable of performing substitutions of size two. Figure 4.7.c shows the accepted changes for a single solution throughout the entire optimisation process. Here we see that DO repeatedly rescales the variation to higher orders of organisation. Specifically, after each transition, a new hidden layer learns a compressed representation of the lower-level organisation, allowing for substitutions of higher-order to be performed. Importantly, Model-Informed Variation was capable of extracting information from all hidden layers and therefore overcoming the limitation of rA-G.

4.2.3.3 Diversity Maintenance

The restart Autoencoder with Generative Associations algorithm (user in Chapter 3) used online learning, and therefore before each Model-Informed Variation process, a solution was reinitialised using a random uniform distribution. This allowed for exploration of the search space. In DO, a distribution of solutions is optimised and maintained throughout. Therefore, a solution in the population is updated, rather than being reinitialised. This provides implicit diversity maintenance, where potentially meaningful solutions in previous neighbourhoods are maintained and only updated if a higher-order

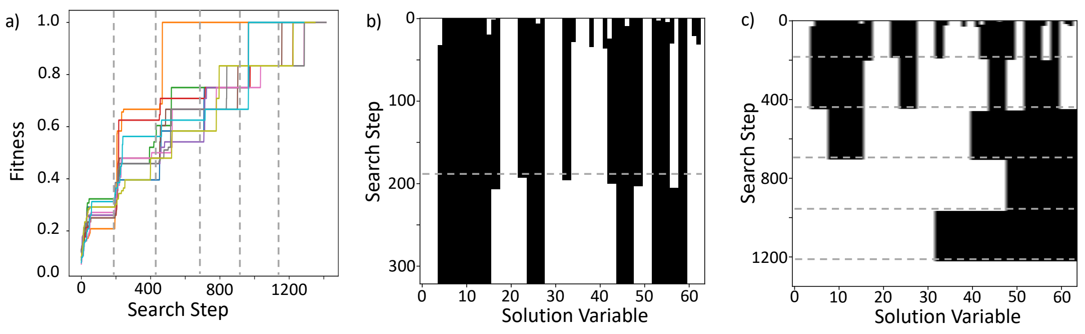


FIGURE 4.7: Demonstration of DO solving the HIFF problem of size 64 using stacked layerwise construction, forcing the capture of a multi-level representation of the problem structure of HIFF. a) Shows the fitness evolution for ten random solutions. A dashed line indicates a transition. b) shows the accepted changes made to a solution before and after the first transition - variation is rescaled from single variable substitutions to double variable substitution. c) shows the accepted changes made to a solution for all stages of the optimisation process and terminating at a global optimum. DO was successful in constructing a deep representation to capture the hierarchical structure and the Model-Informed Variation method was capable of searching at all layers of the autoencoder model.

neighbour has superior fitness. Thus, if a solution is not updated in a compressed representation, it is still used to update the model - the information in this solution can still be helpful to induce higher-order features and should not be disregarded prematurely. An Estimation of Distribution Algorithm, on the other hand, uses explicit diversity maintenance strategies. Without this, a solution in the distribution can be replaced by a newly generated solution. This replacement solution can contain very different information (corresponding to a different region of the solution space) and subsequently remove meaningful information from the optimisation process without knowing if the information could have been helpful.

4.2.3.4 Layerwise Search

Due to DOs capability to represent multiple representations of a solution, a search can be performed in multiple solution neighbourhoods using a single model. For DO's primary operation, search is performed at the deepest layer of the autoencoder model (bottleneck layer) to update a solution. After Model-Informed Variation has terminated, the solution is updated by performing a local search at the original solution representation (*Model-Informed Variation Layers* = {D, S}, where D is the deepest layer and S is the original solution level representation). By doing so, a solution undergoes an effective repair at the solution level organisation. For instance, a higher-level substitution can disrupt some lower-level fitness contributions but still get accepted as the net gain to fitness is greater. Therefore, by performing a local search at the solution level representation, any disruption to lower-level combinations that were previously found can be repaired, ensuring that a solution is optimal, at least, relative to the original solution variables. Further, the higher-order substitution could move the solution into a new but important region of the solution space in the lower-level neighbourhood. Thus, local search in this new area can fine-tune the quality of the solution in this space.

4.2.4 DO limited to a single hidden-layer

A claim of this research project is that inducing a deep representation of the solution enhances the capability of a simple evolutionary process. To support this claim beyond comparison between other algorithms that do not explicitly transform the representation of a solution, we perform experiments using a version of DO using an autoencoder model limited to a certain depth. Concretely, the autoencoder model is limited to a maximum representation depth of D hidden layers. We refer to this limited version of DO as $\mathrm{DO_D}$. For example, $\mathrm{DO_1}$ refers to DO using an autoencoder with a single hidden layer. Consequently, the transition process differs when the autoencoder has reached the maximum depth: instead of adding a new layer to the model, the parameters of the deepest layer are re-initialised at transition. Training proceeds as usual, and Model-Informed Variation is applied at the same hidden representation (Layer representation $= D$). In addition, the limited version of DO could proceed without performing a re-initialisation of the hidden layer parameters. However, we found in practice that re-initialisation allows the parameters to escape attractors in parameter space, allowing the layer to learn a different compression. For instance, solutions found by combining lower-order features can still be reconstructed well using only lower-level features - the reconstruction error is still small - even though there is a change in the dimensionality of the variation in the dataset. Consequently, there is not a clear training signal to encourage the layer to combine its feature representation as representing higher-order features as a combination of lower-order components is still accurate. Therefore, the re-initialisation allows for a layer to learn a new compressed representation that ignores lower-level components. Further, the idea of re-initialising a layer shares similarities with the operation of MBOAs, where the model is reconstructed (effectively re-initialised) after each population updates to capture the variation in the current population.

4.2.5 Model Hyper-parameters

An advantage of using a neural network model is that we can control the parsimony pressure by applying regularisation during learning. For all other MBOAs, the parsimony pressure is implicit in the scoring metric used during the construction of the model. The disadvantage for DO is the extra hyper-parameters that can be tuned to improve the training of a neural network. The set of hyper-parameters available for DO is the following:

1. **Learning Rate:** The learning rate controls the parameter update size per gradient descent step. In practice, we found a learning rate between 0.01 and 0.001 works

well. With more complex or larger problems working better with smaller learning rates.

2. **Batch Size:** Batch size controls the number of training examples used to calculate the error gradient. We find that a relatively small batch size (4-16 training examples) worked well for the problems explored here. Note that the training set size is generally relatively small compared to more familiar learning tasks - i.e., a training set size of 1000 is considered a large dataset in our experiments.

3. **Epochs:** Epochs refers to the number of times a training data point is used to update the parameters of the model. This can significantly improve DO - repeated exposure to the same data points allows the network to find a suitable configuration without the need for extra function evaluations that would find the same relationships in other solutions. In practice, we found that 20-50 epochs worked well.

4. **Optimizer Parameters:** Learning rate forms the primary parameter for optimisation methods as it defines the size of the gradient step to make at each iteration. However, more advanced methods such as stochastic gradient descent with momentum and Adam optimiser contain additional parameters. For our experiments, we used the Adam optimiser with its default values. Specifically, 0.9 for the exponential decay rate for the first moment and 0.999 for the exponential decay rate for the second moment.

5. **Training Set (Population) Size:**

   The population size sets the number of solutions used to update the model. A larger population size provides greater coverage of the solution space and therefore provides a greater distribution of locally optimal solutions for the model to induce relationships from. As is the case for all MBOAs, the population size is an important parameter with more complex or larger problems requiring a greater population size.

4.2.6 DO - Summary

Here we have introduced an implementation of the DO algorithm using an autonecoder model that allows for inducing a deep representation of a solution by performing transitions. Each transition step induces a higher-order representation of the solution space and allows for an evolutionary process (local search in this case) to be rescaled to higher orders of organisation. The idea of multi-scale search is not new (Watson et al., 2011b; Mills, 2010); however, DO is the first algorithm to represent multiple representations of a solution explicitly. Therefore, DO is the first example of a 'true' multi-scale search algorithm due to its ability to separate and represent multiple scales of representation and perform the search at each level of representation.

4.3 MBOA Comparison

DO emulates the process of evolutionary transitions in individuality to re-scale the search space, allowing for evolutionary search to continue in a redefined search space. Chapter 2 introduced many algorithms that combine machine learning and the emulation of evolutionary processes to explore a solution space - the class of Model-Building Optimisation Algorithms (MBOAs). Specifically, algorithms that use a machine learning model to learn information from a population of selected solutions, and then use this information to search for new solutions. The algorithms that have shown the best performance in a range of synthetic problems are the Parameter-less Pyramid Population algorithm (P3), the Linkage-Tree Genetic Algorithm (LTGA), and the hierarchical Bayesian Optimisation Algorithm (hBOA), and the Dependency Structure Matrix Genetic Algorithm version 2 (DSMGA-II). Considering the differences between these algorithms, with specific regard to the type of model used and the way a model is used to inform future exploration, the lack of categorical differentiation between them impedes our understanding of how the model affects adaptation for future exploration and how DO fits into this class of algorithm. For instance, a Bayesian network can represent multivariate dependencies, whereas a linkage tree can only represent bi-variate dependencies. However, performance comparisons between them show results that favour the use of a linkage-tree (Thierens and Bosman, 2013; Goldman and Punch, 2015; Hsu and Yu, 2015). These results open the question: is it necessary to use sophisticated models like a Bayesian network? If not, this implies that using a deep neural network will also not improve the performance of a MBOA.

All algorithms considered in this thesis apply variation and selection over multiple generations to explore a solution space. The set of possible variants a solution can take is controlled using a centralised method. Specifically, all solutions are updated using the same model and method for exploiting information from the model. Further, the model is constructed (or updated) by inducing relationships from the population of solutions. Here, we discuss how an individual solution is updated by exploring variants that are considered neighbours in the reorganised space induced by the model, i.e., differ by a single higher-order substitution. This method is either performed implicitly, using a combination of the population and learning model; or explicitly using the learning model only.

First, we discuss how the class of algorithms considered in this thesis also performs a similar re-scaling to DO, i.e., using the model to compress the search space and then updating solutions by exploring neighbours in compressed solution space. In turn, we examine the differences between the algorithms regarding their consequences for the structure they can represent, their methods to induce this structure, and how they use it to move in solution space. Table 4.1 summarises the differences between these algorithms. This section aims to understand what one algorithm might be able to do

Method	Properties								
	MIG	MIC	MIV	Pair-wise	Multi-v'te	Tree-based	Graph-based	Deep	'Houpi'
DO	×	×	✓	×	✓	×	✓	✓	✓
DO_1	×	×	✓	×	✓	×	✓	×	✓
LTGA	×	✓	×	✓	×	✓	×	×	×
hBOA	✓	×	×	✓	×	×	✓	×	×
DSMGA-II	×	✓	×	✓	×	×	✓	×	×
P3	×	✓	×	✓	×	✓	×	✓	×

TABLE 4.1: Summary of the key properties of the model-building optimisation algorithms used in this thesis, where 'Houpi' means that higher order units (of variation) are population independent.

that another cannot. We complete this section by making three hypotheses about the types of problem structures that distinguish the capabilities of these algorithms.

4.3.1 Higher-order substitution

First, we separate and categorise the different methods used by the algorithms considered in this thesis to construct a higher-order substitution. We characterise the methods as Model-Informed Generation, Model-Informed Crossover, and Model-Informed Variation. For a binary problem, suppose that in a distribution of promising solutions, we observe that two particular solution variables frequently take the same value as each other, i.e., 00 or 11. And suppose we learn a model that represents this relationship. In Model-Informed Generation, the model is used to generate 00 and 11 more often than 01 or 10. In Model-Informed Crossover, the model represents variables v_1 and v_2 as a linkage-set. Crossover is then performed between two random solutions using the linkage set as a crossover mask. Considering the model captured this information from the same distribution of solutions used to perform crossover, the crossover operation is likely to exchange between values 11 or 00. In Model-Informed Variation, the model compresses and represents the factor of variation as a single dimension, i.e., the model represents 00 and 11 as neighbours even though they are not neighbours in original space. The single-unit change is applied to the compressed representation unit, decoding back to the original problem variables as a simultaneous change to multiple solution variables.

For Model-Informed Generation, new solutions are generated by sampling the model, with generated solutions showing the greatest variation between the weakest links in the model. Model-Informed Generation is not preconditioned to account for the solution it is to replace or update. Consequently, as done by BOA, updating a solution using Model-Informed Generation alone does not update a solution by a single higher-order substitution - many higher-order units could be changed, and as a consequence, not update a solution with a neighbour in the compressed space (a difference to a single higher-order unit). The hierarchical Bayesian Optimisation Algorithm, however, uses Restricted Tournament Replacement, after Model-Informed Generation, that subsequently approximates the process of updating a solution that is local in the compressed solution space. Specifically, Restricted Tournament Replacement finds a solution in the current population that is the minimum Hamming distance away from the generated solution. Selection is then applied between the two solutions. The solution selected from the population is replaced if the generated solution is fitter. Therefore, a higher-order substitution is not defined by the model but rather a combination of the model and the population of solutions.

For Model-Informed Crossover, solutions are updated using crossover and a linkage-set to define the exchanged variables between two solutions. The linkage set controls the size of the potential change made to a solution. The actual change made to a solution is dependent on the difference between the two parent solutions at the variables within the linkage set. Consequently, a higher-order substitution is dependent on the regularity of variable states within the linkage set. For instance, with no regularity (maximised diversity), selecting a random solution from the population will cause a change analogous to a random perturbation - a random higher-order substitution - thus unlikely to be adaptive. However, suppose the population shows regularity at these variable locations, given that the linkage-sets are constructed to represent dependencies between variables is likely, then a change will represent a partial solution that has emerged from evolutionary search (has been selected for). Finally, suppose there is no diversity in the linkage-set, i.e., all variables have the same value, then Model-Informed Crossover does not perform a change as there is no diversity in the population to construct a change. As is the case for Model-Informed Generation, the model alone does not represent a higher-order substitution; a population is required to complete the process (the model only describes the dependency information and not the relationships). Model-Informed Crossover, however, is preconditioned with the solution it is updating and therefore does not use explicit diversity maintenance strategy (Thierens and Bosman, 2011). Consequently, a solution can be explicitly updated multiple times: the model is used repeatedly to update a solution before the model is reconstructed.

Model-Informed Variation, and specifically DO, has the potential to be an improvement on other optimisation algorithms because it is the only method that does not require a population to make a higher-order substitution. Consequently, the ability to vary is not dependent on the state of the current population. DO explicitly learns the directions of variation rather than an approximation using the model and a population as done with Model-Informed Crossover and Model-Informed Generation. DO induces a higher-order representation of a solution using an autoencoder model. A hidden layer represents the salient factors of variation in the distribution of locally optimal solutions at the organisation level below. Therefore, variation in the distribution of solutions is represented by the hidden nodes. Constructing a higher-order substitution is performed by using the encoder and decoder network. The encoder network is used to transform a solution into a compressed representation. A change at this representation level corresponds to a change to a higher-order unit. The decoder network is then used to inform a coordinated change at the lower-level representations that correspond to a change in the compressed representation. This completes the update made to a solution. Therefore, only the model is used to inform a higher-order variation to a solution.

Model-Informed Crossover and Model-Informed Variation share similarities as they both repeatedly use the model to update a single solution before the model is updated. However, Model-Informed Variation requires no population to construct a higher-order substitution. Consequently, the ability to vary is not dependent on the state of the current population. This provides DO with important capabilities, such as being able to explore multiple levels of representation even as the population and model evolve. Further, as DO explicitly represents a transformed representation of a solution, the process of Model-Informed Generation (random initialisation of the hidden representation) and (or) Model-Informed Crossover (exchange between the hidden representations of two solutions) could also be used at each representation level. We envision an algorithm that takes advantage of all these approaches; however, this remains out of scope for this thesis and, instead noted here as a direction for future research. To summarise, the set of algorithms updates a solution by constructing a higher-order substitution that moves the solution to a neighbour in the compressed representation learned by the model. For Model-Informed Generation (of which the hierarchical Bayesian Optimisation Algorithm is an example) and Model-Informed Crossover (of which the Linkage-Tree Genetic Algorithm, the Parameterless Population Pyramid algorithm and the Dependency Structure Matrix Genetic Algorithm II are all examples), a population is required to construct a higher-order substitution; for Model-Informed Variation (of which DO is an example) the model alone is sufficient for constructing a higher-order substitution.

4.3.2 Inducing a Compressed Representation

Now we separate and categorise the different models. For the hierarchical Bayesian Optimisation Algorithm (hBOA), a Bayesian network model is used to capture the distribution of solutions that have been selected. For each variable, a decision tree encodes its conditional probabilities. Subsequently, the model learned by hBOA represents the linkage information (which variables show dependencies) and the weights of the links between the variables (the conditional probabilities). Each node in the network represents the value of a solution variable. Parent nodes in the Bayesian network with no, fewer, or weaker connections with their parent units than their child units represent higher-order units. Concretely, a change to a parent solution variable causes a change to each of their child solution variables' probability (a change to a single unit causes a change to multiple units simultaneously). Constructing the model requires identifying dependencies between variables. As the orders of interactions increase, the computational requirement to calculate the dependency strength increases exponentially. Therefore, hBOA uses pairwise interactions to approximate the higher-order interactions so that the model induction does not have exponential time complexity.

In the case of the Linkage-Tree Genetic Algorithm (LTGA), the Parameterless Population Pyramid algorithm (P3) and the Dependency Structure Matrix Genetic Algorithm II (DSMGA-II), these methods construct a hierarchical representation of the dependency structure between variables in selected solutions. Each node represents a grouping of variables called a linkage-set, with a parent node representing a combination of the two child nodes. A leaf node represents an individual variable, and the root node represents the complete solution representation - in the case of LTGA and P3, the model structure is a binary tree, in DSMGA-II the model allows nodes to have multiple parents. Each linkage-set is used to inform a crossover mask during recombination between two solutions to create a new solution (Model-Informed Crossover). The linkage model is constructed using a dependency structure matrix and agglomerate clustering, where a linkage-set is created by joining two variables that show the greatest dependency. Calculating the dependency between a linkage-set grows exponentially with the size of the group (as order of interactions can increases). Therefore, the calculation uses a pairwise approximation (an average of pairwise distances between all variables in one linkage-set and all variables in another linkage-set), which in turn was shown to perform well (Thierens, 2010; Pelikan et al., 2011).

DO uses a deep autoencoder network constructed one layer at a time. The model is updated using the population of solutions. The hidden layer represents a compressed representation of the solution space, capturing the salient factors of variation in the distribution of solutions. The compression attempts to maximise reconstruction (minimising information loss). Therefore, the compressed representation will try to capture relationships between variables that co-vary and represent these by the hidden nodes. Variables that show no clear relationship should not be compressed and instead represented as a separate hidden unit that can vary independently. Relationships learned by the model are induced by minimising the reconstruction error between the output of the decoder and the original input and therefore not limited to a pairwise approximation, unlike the other algorithms.

All algorithms have a polynomial time complexity for the learning process. The LTGA uses the simplest model and therefore has the lowest time complexity to build. P3 uses many instances of the same model used in LTGA. Consequently, a higher-time complexity occurs than LTGA. DSMGA-II, hBOA and DO all have higher time-complexity for the learning process. This was observed in our experiments although not directly measured (e.g., pe, performing a single generation with the same population size was noticeably slowing in all three algorithms compared to LTGA and P3). And this can be attributed to the learning process for the more complex models. Interestingly, there is no evidence in the literature to show why one would use an algorithm with a more complex model (as the scaling with regards to the number of function evaluations is similar). Therefore, using a complex model does not appear to show an advantage in comparison to using algorithms which use a simpler model (and are more efficient to build).

4.3.2.1 Multi-level Representations

As discussed in the previous section, these algorithms implicitly induce a higher-order substitution and update each solution by applying these substitutions to a solution. A distinguishing feature of DO is the modelling capability of multi-level representation. These algorithms can implicitly represent multiple scales of evolutionary units based on the difference in frequency in the population, i.e., more likely to make a lower-order change than a higher-order change as more variability at the higher-order units are present in the population. However, a more distinguishing factor is that the algorithm's repeatedly reconstruct their models rather than update the existing model (unlike DO). Each generation, and consequently the reconstruction of a new model from a new distribution of solutions, acts to repeatedly adapt the type of change that can be made to a solution, or rather, induce a new compression. However, as lower-order units combine to form higher-order units, variations between the lower-order units may no longer be present in the population (or become rarer). In turn, this leads to a loss of information about lower-order units. DO, on the other hand, does not reconstruct the model after each generation. Instead, each generation in DO performs an explicit transition, where a new layer is added to the autoencoder model, allowing a higher-order representation of the current representation level to be induced. Consequently, each layer of the model represents a reorganisation of the solution space. This difference is illustrated Figure 4.8. The explicit multi-level representation allows DO to maintain information about the lower-level units while at the same time inducing a new compressed representation. Consequently, learning the lower-order units can help DO reduce the complexity of learning how to combine lower-order units, as the model already contains this information - we explore this further with problems that require deep structure in chapter 5.

In addition, P3 operates differently from hBOA, LTGA and DSMGA-II. Specifically, P3 uses stacked instances of LTGA, where each population is populated with solutions found using the LTGA instance directly below, illustrated in Figure 4.8. Thus, P3 performs a type of layer-wise learning and multi-level representation analogous to DO. Each LTGA instance represents a higher-order reorganisation of the neighbourhood of a solution from the LTGA instance below. Using stacked instances of LTGA, P3 maintains a population and separate linkage-tree model at each representation layer, therefore providing a capability to maintain information about multiple scales of substitutions - a model and population for each scale of organisation.

The ability to represent a multi-level representation of a solution's neighbourhood allows DO and P3 to search at multiple scales throughout the optimisation. P3 performs a bottom-up search. Specifically, a solution is initialized and then sequentially optimised within each of the reorganised neighbourhoods starting at the lowest-order organisation and finishing at the highest order. DO is also capable of performing a bottom-up and a top-down search. In this case, high-order changes can make large updates to a solution - moving the solution into a better basin of attraction - and then perform a lower-order search to 'fine-tune' or 'repair' the solution - moving the solution to the attractor within the basin by search at lower-level representations. In this thesis, we only explore DO using a top-down approach where the deepest layer is used first to update a solution, and then the solution is updated by performing a search at the lowest level representation.

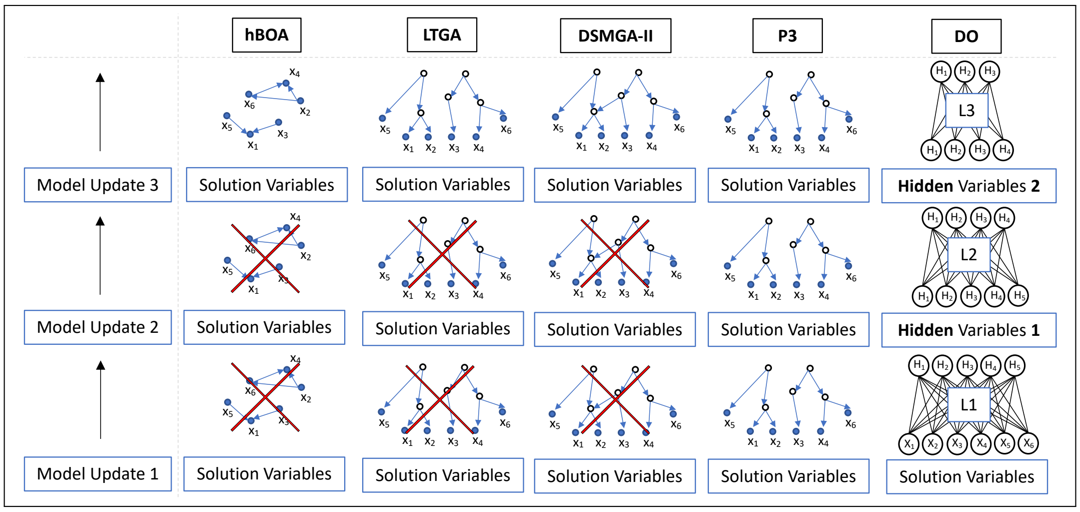


FIGURE 4.8: Comparison of the models and how the models are used by MBOAs considered in this thesis to optimise a solution to a problem. hBOA, LTGA and DSMGA-II construct a new model to induce higher-order substitutions. P3 and DO maintain information about lower-level substitutions.

Here we conclude that hBOA, LTGA and DSMGA-II fail to maintain the information about previous neighbourhoods used to find higher-order units. We hypothesise that DO and P3 are capable of solving problems that require searching at multiple and different neighbourhoods, whereas hBOA, LTGA and DSMGA-II are not.

4.3.3 Distinguishing Characteristics

In all discussed methods, updating an individual solution uses a centralised method — a single model that is used to update all solutions. Therefore, the set of all possible adaptive changes for any given solution must be captured and represented by the model. Which change is adaptive is dependent on the solution that is being updated. Consequently, the challenges an MBOA must overcome can be attributed to the distribution of changes required to update the population of solutions. Therefore, the set of all variations — precisely the challenge of separating, representing and constructing higher-order substitution — will directly affect the model induction and exploitation capability of an MBOA. To emphasise, the models are used to learn relationships between variables that co-vary to collapse the variation degrees-of-freedom and are not used to learn relationships of the fitness function. We make the following hypothesis regarding the characteristics between higher-order substitutions that will differentiate the performance of the models used by MBOAs:

1. The Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid use a strict binary tree to represent the dependencies between variables and are therefore restricted to representing non-overlapping dependencies. For DO, the hierarchical Bayesian Optimisation Algorithm and the Dependency Structure Matrix Genetic Algorithm II, this is not the case. Consequently, when the set of higher-order substitutions has overlapping solution variables, the Linkage-Tree Genetic Algorithm and P3 will necessarily fail to improve a solution, whereas DO, the hierarchical Bayesian Optimisation Algorithm and the Dependency Structure Matrix Genetic Algorithm II will not.

2. Model-Informed Generation and Model-Informed Crossover use a population of solutions to update a solution considered a neighbour in the compressed search space. Model-Informed Variation, on the other hand, does not use a population as the model contains all the necessary information. Therefore, if search removes the diversity in a population, MBOAs using Model-Informed Generation or Model-Informed Crossover (The Linkage-Tree Genetic Algorithm, Parameterless Population Pyramid, the Dependency Structure Matrix Genetic Algorithm II and the hierarchical Bayesian Optimisation Algorithm) will necessarily fail to find this fitter solution, were as Model-Informed Variation (performed by DO) will not.

3. All MBOAs, at some point, use pairwise statistics between variables to construct the model, whereas DO does not. Consequently, when the set of higher-order substitutions appear independent when measured using pairwise statistics (or cannot be well approximated using pairwise statistics), all MBOAs will necessarily fail to search in the higher-order organisation, whereas DO will not.

The first hypothesis refers to overlapping dependencies, which has been identified as a potential limitation during the development of EDA's (Bosman and Thierens, 1999; Pelikan et al., 1999). However, benchmarks containing overlap used to demonstrate a multivariate models' performance turned out to be solvable in polynomial time using The Linkage-Tree Genetic Algorithm and P3 (Goldman and Punch, 2015; Hsu and Yu, 2015; Chen et al., 2017). Therefore, an important distinction we make here is that we specifically refer to overlap in the set of higher-order substitutions rather than dependencies in a fitness function.

The second hypothesis refers to the differences in how exploration in the space of higher-order units of variation differs between algorithms. To demonstrate the differences between Model-Informed Generation, Model-Informed Crossover, and Model-Informed Variation further, consider the case when the model represents each solution variable as a higher-order substitution. Model-Informed Generation can generate all possibilities equally likely; using Restricted Tournament Replacement and given a sufficiently large population, a solution will be updated by a single change to the solution variable. For Model-Informed Crossover, the search reduces to an approximation of local search, where each solution variable is perturbed once. For Model-Informed Variation, the search reduces to local search - hence Model-Informed Variation is the only algorithm that is consistent with the idea of using local search to initialise a solution and not considered a hybridisation as in the case of algorithms that use Model-Informed Crossover or Model-Informed Generation (Hauschild and Pelikan, 2011). Therefore Model-Informed Generation and Model-Informed Crossover are limited approximations of local search in the neighbourhood defined by the model.

Finally, the third hypothesis refers to the model construction methods used by all MBOAs. All models are constructed by adding linkage between nodes that show the greatest measure of dependency in the distribution of solutions. However, as the order of the statistics increases, the complexity of calculating the dependency information increases exponentially. Therefore, all MBOAs excluding DO use pairwise statistics to approximate the dependency information to simplify model construction. On the other hand, DO learns the parameter values by incrementally updating the weights of the network in the direction that reduces the reconstruction error. Therefore, DO is not limited to pairwise statistics. The characteristic of pairwise independent functions is known to cause failure to MBOAs that construct models using pairwise statistics (Martins and Delbem, 2016).

In chapter 5, we explore how optimisation problems can explicitly create the challenges described in these hypotheses and then evaluate each MBOA and DO's performance to overcome these challenges.

4.4 Summary

In this chapter, we first introduced the Deep Optimisation algorithm, an algorithm implementation that emulates the process of Evolutionary Transitions in Individuality using a deep neural network. A transition is achieved by inducing a compressed representation of a solution and training an autoencoder model to learn the reconstruction of locally optimal solutions. The locally optimal solutions contain selected relationships between variables. Regularly occurring relationships are collapsed and represented in the hidden layer of the autoencoder model, compressing the representation from variation between individual solution variables to variation between combinations of variables that were found to co-adapt. Search is then applied to the new compressed representation, where local changes in the compressed representation are decoded back to the original representation to perform a higher-order substitution to a solution (simultaneous change to multiple variables informed by the model). A further transition occurs by adding a new hidden layer to the autoencoder model, training the layer on locally optimal solutions found by searching in the layer below. Search is then rescaled further, and the process repeated, creating a deep representation of a solution space and emulating multiple successive transitions.

Then we discussed and compared the functional differences between MBOAs to further our understanding of how DO differs and contributes to the class MBOAs. We find that all MBOAs approximate the process of inducing a compression of the search space and updating an individual solution by exploring a solution's local neighbourhood. Thus, each MBOA approximates a transition step as performed by DO. However, DO does not require a population to search in the compressed representation, whereas all other MBOAs do; and DO can explore multiple compressed spaces, whereas other MBOAs lose information about previous solution spaces due to the reconstruction of a model and the change to a population's diversity during the search. Finally, all MBOAs use a centralised method to update a single solution. Therefore, the model and search methods used by an MBOA must represent all adaptive substitutions and apply the context-sensitive substitution (the adaptive substitution is dependent on the solution being updated). Consequently, the complexities that differentiate the performance between MBOAs are confined to the set of higher-order substitutions (captured by and exploited from the model) required to update solutions. Specifically, the complexity of learning the set of substitutions and the complexity of applying the substitution. From this identification, we create hypothesises for problem structure that will categorically differentiate the performance between MBOAs. These are: overlap in higher-order substitutions, a set of higher-order substitutions that require greater than pairwise statistics to identify, and performing local search using the set of higher-order substitutions. In chapter 5, we evaluate these hypothesises.

Chapter 5

Exploring Characteristics of Problem Difficulty

This chapter aims to understand how deep learning and explicitly searching in deep representations of a solution can enhance evolutionary search. As discussed in chapter 4, there are algorithmic differences between the Model-Building Optimisation Algorithms investigated in this thesis. However, experimental results have not provided a categorical differentiation between what optimisation algorithms can and cannot do (polynomial vs exponential running times) (Thierens and Bosman, 2013; Hsu and Yu, 2015; Goldman and Punch, 2015). This suggests that the model type and the way of using the model do not significantly affect an algorithm's performance. Subsequently, opening the question for the need for sophisticated models. Therefore, this chapter's motivation is to evaluate the algorithmic differences identified in chapter 4 and how this affects the capability of an MBOA to exploit problem structure and subsequently find solutions. Whilst DO is in the same algorithm class, throughout this chapter, we use MBOA to refer to the Parameterless Population Pyramid algorithm, the Linkage-Tree Genetic Algorithm, the Dependency Structure Matrix Genetic Algorithm, and the hierarchical Bayesian Optimisation Algorithm, and we refer to DO explicitly.

We explore these differences by developing a synthetic optimisation problem that allows for clear, independent, and proportional control of problem characteristics identified in chapter 4. The problem construction separates the complexity of learning a representation of a reorganised neighbourhood and exploring in the reorganised neighbourhood, the two main functions an MBOA performs. This separation allows for clear differentiation between an MBOAs ability to learn the relationships between variables and exploit these relationships to find higher-order solutions. Therefore, performance differences can be attributed to either the choice of model or how the model is used to inform search.

We identify that overlapping higher-order substitutions, non-pairwise identifiable higher-order substitutions, and local search using higher-order substitutions are distinct problem characteristics that cause challenges to some MBOAs and not others. Further, we find problem characteristics that DO limited to a single-layered autoencoder model fails to overcome that DO, using a deep autoencoder model, shows polynomial running time. Finally, we identify that overlapping higher-order substitutions is a problem characteristics that is sufficient to differentiate the performance between MBOAs for the first time. The results presented in this chapter support the main claim of this thesis.

5.1 Introduction

MBOAs exploit low-order relationships that emerge in the distribution of solutions that are selected. Consequently, they assume that the lower-order relationships contribute to higher-order solutions in some way - generally as a combination. The methods for controlling the problem structure complexity generally involve increasing the degree of complexity within a fixed-sized problem (Kauffman et al., 1993; Deb and Goldberg, 1994; Chen et al., 2012; Chang et al., 2011; Wang et al., 2013) or increasing the problem size, i.e., including more of the same by increasing the problem size (Thierens, 1995; Watson et al., 1998; Pelikan and Goldberg, 2001; Yu et al., 2005; Coffin and Smith, 2007). Evaluation of an algorithm's performance requires measuring the change in function evaluations due to the change in complexity. A change to an algorithm's performance due to a change in the problem complexity indicates that the algorithm finds the complexity challenging in a fixed-sized problem. However, this requires a measure for the degree of complexity. Alternatively, a scalability analysis can be performed, where the size of the problem is increased and the number of function evaluations required to find a global optimum is measured. If the problem is constructed such that a global optimum can be found in polynomial time, algorithms showing exponential running times fail to exploit the problem structure. Further, the scalability analysis can provide a method of comparing the performance between algorithms that show polynomial scaling - indicating differences in the efficiency of an algorithm to exploit the problem structure to find a global optimum.

Therefore, we focus on developing a synthetic optimisation problem that contains a clear and controllable problem structure, such that if an MBOA can exploit that problem structure, the global optimum solution is easy to find; otherwise, it requires exponential running time. MBOAs assume that a problem can be decomposed into smaller, easier to solve subproblems. Solutions to these subproblems can then be used to find higher-order solutions. A subproblem is referred to as a Building-Block (module) (Holland et al., 1992a; Goldberg, 1989a), and we refer to a solution of a module as a partial solution. In using a module structure, we can explore problem characteristics aligned with a MBOA's operation.

Recall that in chapter 4 we identified the following differences between DO and the other algorithms:

1. **Model Capacity:** Tree vs Graph dependency structure to model relationships between variables.
2. **Model-Informed Search:** Model-Informed Generation, Model-Informed Crossover and Model-Informed Variation.
3. **Model Construction:** Pairwise statistics vs reconstruction error used to find the parameters of the model.

In the following sections, we review the relevant literature for each of these problem characteristics.

5.1.1 Characteristics for Model Exploitation

The first stage of an MBOA is to learn information about the problem structure. As the problems are black-box functions, information can only be learned from the solution representation. Selection is used to encourage solutions to contain information about how problem variables combine to increase the fitness of a solution. As such, models are then used to extract this information from the solution by observing regularities in a distribution of solutions. The challenge for an MBOA then becomes ts ability to use this information from the model to inform search.

5.1.1.1 Hierarchical Dependencies

In decomposable problems, the dependencies within and between modules contribute to the complexity of a problem. A decomposable problem in its simplest form is an additive decomposable function. In this case, there exist no dependencies between modules: the global function is a summation of lower-order functions. Subsequently, the optimal partial solution for a module is the correct partial solution for the global function. These problems were sufficient to challenge GAs when the problem linkage was not tight (Deb and Goldberg, 1994; Thierens, 1995).

Partially decomposable functions (or non-additive decomposable problems) contain dependencies between modules. Subsequently, the optimal solution of a module depends on the partial solution of other modules to construct a globally optimal solution. The dependencies between modules can be either weaker (Watson et al., 2011b) or only observable when a module contains a partial solution (Watson et al., 1998; Pelikan and Goldberg, 2001). To satisfy dependencies between modules, search must be able to search in combinations of partial solutions to avoid disrupting the already found partial solutions to modules.

Often used in evaluating the performance of MBOAs is the deceptive-trap (Trap-K) optimisation problem (Deb and Goldberg, 1993). The problem is an additive separable problem constructed out of many modules that each contain the deceptive-trap function. The deceptive-trap function has a fitness gradient that leads all configurations away from the optimal configuration of all 1's to all 0's. The global solution is where all modules contain the optimal configuration of all 1's. This problem could not be solved using a simple hill-climber and instead required more sophisticated techniques like an EDA. However, a limitation of this problem is that finding a solution to a module scales exponentially with respect to the size of a module. Therefore, the module size had to be small. Instead, if a module was not deceptive, then the size of the module could be large. Mills (2010) showed that an optimisation problem with an obvious nearly separable modular structure can be pathologically difficult to solve for a hill-climber but still easy to solve for a MBOA method despite having large modules. In this case, local search can solve anyone module (there is no need for exponential search to find the within module solution). However, a simple hill-climbing algorithm would fail to solve the entire problem as solving each module optimally is not fully reliably. Nonetheless, a distribution of local optima is sufficient to identify the modular structure. If exploited appropriately, by searching in the space of module solutions, the global optimum can be easily found but remains pathologically difficult for a hill-climber. Therefore, in using this structure, we can evaluate an MBOA without the concern of complexity in finding the partial solutions.

Described above is a single level transformation of the search space, from searching using individual solution variables to searching using individual modules. Hierarchical decomposable functions define a recursive application of this idea. Specifically, hierarchy describes a multi-layered combination of modules to construct the global problem (Watson et al., 1998; Pelikan and Goldberg, 2001). The structure is inspired by a bottom-up mechanism to construct a solution. Specifically, low-order partial solutions are used to construct higher-order partial solutions, which construct even higher-order partial solutions (higher-order partial solutions contain a particular combination of lower-order partial solutions). Consequently, the dimensionality of the search is recursively reduced from variation to the original problem variables to variation of low-order partial solutions to variation to combinations of low-order partial solutions, and so on. This multi-level nesting of modules requires an algorithm to repeatedly adapt and reorganise the neighbourhood of a solution to higher orders of an organisation to explore the problem space efficiently. Pelikan et al. (2003) describes the following problem challenges an MBOA must overcome to exploit hierarchical problem structure efficiently:

1. **Decomposition:** An MBOA must be capable of separating the modules at different levels of hierarchy.

2. **Chunking:** An MBOA must be capable of representing partial solutions at each level of the hierarchy.

3. **Niching:** An MBOA must be capable of preserving alternative partial solutions to a module until it becomes clear which partial solution is the correct partial solution to use (Diversity Maintenance).

However, as shown in chapter 3, synthetic problems containing hierarchy did not require deep model (a model capable of representing multiple levels of organisation) to learn the multi-level structure. Specifically, the problem characteristic hierarchy was insufficient to require an MBOA to represent each level of the hierarchical structure. Instead, the model only needed to learn the current level of organisation and then use this to find the higher-order solutions, i.e., it is not necessary to maintain how the low-order partial solutions combined to create a higher-order partial solution. In this chapter, we investigate and develop a problem characteristic that does require a deep representation (requires maintaining information about how higher-order partial solutions are formed from lower-order partial solutions)

5.1.1.2 Model-Informed Search

Model-Informed search refers to the function of exploiting information from the model. As discussed in section 4.3, Model-Informed Generation (MIG) approximates a single local search step each generation (before the model is reconstructed), Model-Informed Crossover (MIC) approximates local search performed to each dimension of the solution once each generation, and Model-Informed Variation (MIV) is the only approach to perform local search in the reorganised neighbourhood. We hypothesise that a solution trajectory (the sequence of substitutions made to find a higher-order solution) that requires multiple changes to a single dimension to find a higher-order solution can only be followed by local search and subsequently Model-Informed Variation. For example, a higher-order solution is found by searching the space of lower-order solutions and not just by combining lower-order solutions.

This hypothesis is connected to the longest path problem (Horn et al., 1994) where the global optimum solution is found by following a particular trajectory in the search space. The longest path problem defines a path from any solution point to the global optimum that can be followed by single-unit substitutions (hill-climbing at the solution level representation). However, the path is designed to scale exponentially as the problem size increases, causing local search to exhibit exponential running time. Interestingly, a GA using crossover scaled polynomially for the longest path problem (Horn et al., 1994). However, a follow-up paper by Horn (Höhn and Reeves, 1996) showed that the GA was finding the solution in unexpected ways. Firstly, the GA took advantage of a royal-road type structure induced in the path due to the recursive construction used to construct the path. Therefore, the GA could take 'short-cuts' in the path, reducing the long path to a shorter path. Of greater interest, and the prominent reason for how the GA found the global optimum, was the insight made into the slope function. The slope functions led local search to the start of the path at a solution of all zeros. Therefore, that slope function resembles the one-max problem. In the one-max problem, the population converges towards solutions containing all 1's. In following the slope function, the population loses its diversity and therefore only follows the path using mutation, reducing to local search, and taking exponential running time. The GA was, therefore, not following the path. Instead, the slope function optimum was a two-variable substitution away from the long-path global optimum. As such, the GA was accidentally finding the global optimum of the long path while trying to solve the slope function. Nevertheless, the identification that a population following a path can reduce the population's diversity and consequently limit the variation available is inspiring. Specifically, Model-Informed Generation and Model-Informed Crossover use a population in order to search the neighbourhood of a solution, whereas Model-Informed Variation does not. Therefore, if the path reduces the diversity in population, methods that rely on a population to make a substitution to follow the path have the potential to fail due to the loss of diversity and, therefore, their ability to make the required change.

5.1.2 Characteristics of Model Induction

The challenge of model induction refers to the ability of an MBOA to learn a compressed representation of the search space. Specifically, representing relationships between variables that emerge during the optimisation process such that the optimisation process can exploit this information. As discussed in section 4.3, there are significant differences between the models.

5.1.2.1 Overlap

A key distinction between the models used by MBOAs is using a tree or graph structure to represent the dependencies between the solution variables. A key limitation of a tree data structure is that a child node can only have a single parent. This concept is translated to an optimisation problem as the ability to capture overlap or not, i.e., higher-order substitutions that share variables; or reversed, a variable is a member of more than one higher-order substitution. The problem challenges an optimiser must overcome in the presence of overlap are (Yu et al., 2005):

1. Separate and represent modules that overlap, allowing for variation to combine the partial solutions.
2. During substitution, do not disrupt the partial solution to other modules that share the same problem variables, i.e., conserve the state of the other overlapped modules while performing a variation to the selected module.

It is hypothesised that models capable of representing overlapping structures (the hierarchical Bayesian Optimisation Algorithm, the Dependency Structure Matrix Genetic Algorithm and DO) will outperform models limited to tree structures (i.e., the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm) (Sadowski et al., 2013; Thierens and Bosman, 2011; Hsu and Yu, 2015). However, experiments have failed to show this (Pelikan et al., 2003; Goldman and Punch, 2015; Hsu and Yu, 2015). Results show that while the Linkage-Tree Genetic Algorithm is less efficient than the hierarchical Bayesian Optimisation Algorithm (Pelikan, 2010; Pelikan et al., 2011) and the Dependency Structure Matrix Genetic Algorithm (Hsu and Yu, 2015), the Linkage-Tree Genetic Algorithm does not fail (show an exponential running time). An unexpected result considering the complexity associated with overlap.

The NK landscape problem is a popular benchmark used to evaluate the performance of overlap. The objective of the NK landscape problem (Kauffman et al., 1993) is to maximise equation 5.1. A problem of size n is decomposed into n modules - a module for each variable. Each module takes k additional arguments. (if $k = 0$, a module represents a solution variable only). The module transforms the variables into a real value using a lookup table that arbitrarily assigns real values to all possible binary combinations in the module $(2k + 1)$. All module outputs are summed to calculate the solution fitness. With $k \geq 1$, overlap is introduced between modules. Due to the random assignment of real values to binary combinations in a module, it is unclear how complex the overlap is.

$$F_{nk}(X_0, X_1, , X_{n-1}) = \sum_{i=0}^{\lfloor \frac{n-1}{step} \rfloor} f_i(X_{i \times step}, \prod(X_i)) \tag{5.1}$$

In the NK problem, overlap refers to the dependencies in the fitness calculation. However, it is not well understood if this overlap causes overlap between higher-order substitutions to improve a solution. Recall that the model is not used to induce the fitness function; instead, the model is used to learn substitutions that can be applied to update a solution.

Instead, synthetic problems attempt to explore overlap with greater clarity (Yu et al., 2005). Overlap has been characterised as (Yu et al., 2005; Tsuji et al., 2006; Chang et al., 2011; Chen et al., 2012; Wang et al., 2013):

1. **Overlapping agreement:** If the partial solutions for two overlapping modules have the same variable assignments for the shared variables, the global solution consists of optimal partial solutions
2. **Conflicting Overlap:** If the partial solutions for two overlapping modules require different variable assignments for the shared variables, the global solution consists of a mix of optimal and sub-optimal partial solutions

With these cases of overlap classification, synthetic problems were developed to provide proportional control of the overlap complexity. The control involved increasing the size of the overlap, and the degree of conflict between partial solutions (Wang et al., 2013). However, the complexity increased the number of function evaluations performed for algorithms capable of modelling overlapping interactions. Further, algorithms hypothesised not to be capable of modelling overlapping interactions did not show exponential running times as the complexity increased. Therefore, the results fail to differentiate the performance between models capable of modelling overlap and models that are not. It is unclear how changing the characteristics of agreement and conflicting overlap affects the problem challenges.

As discussed in section 4.3, the model is used to update all solutions in the population. We hypothesise that to categorically differentiate the performance between a tree data structure and graph data structure, the overlap must explicitly refer to the set of higher-order substitutions (the directions of variation that can be applied to a solution). Further, the complexity of overlap should refer to the number of variables in a substitution that overlaps. In doing so, as the size of overlap increases, the ability of a tree model to perform this search decreases. A tree model would be reduced to exhaustive exploration using the lower-order substitutions (decompositions of the overlapping structure) that do not overlap. We, therefore, conclude that the synthetic problems found in the literature are unsuitable to categorically differentiate the performance difference between a network and tree model. In this chapter, we develop a suitable overlapping structure that causes overlapping higher-order substitutions.

5.1.2.2 Pairwise Statistics

The algorithms studied in this thesis use pairwise statistics to construct the multivariate models. In contrast, DO uses an objective function and the back-propagation to update the model's parameters. Therefore, when pairwise statistics are not sufficient to capture higher-order dependencies between variables, we hypothesis that DO will be capable of learning the problem structure that all other algorithms studied in this thesis cannot. Coffin and Smith (2007) termed this property "pairwise independent functions". These functions represent a class of functions that appear independent when measuring statistics between a pair of variables but contain strong dependencies when observing relationships between higher-order variables. These types of functions have been identified to cause failure in all algorithms studied in this thesis (Coffin and Smith, 2007; Chen and Yu, 2009; Iclanzan, 2011). An example of a pairwise independent function is parity. Determining if the function is even or odd requires observing all variable interactions. The concatenated parity function (Coffin and Smith, 2007) was introduced as a simple example to exemplify this difficulty for the Estimation of Distribution Algorithms. It is an additively decomposable problem composed of a concatenation of parity functions. The function to optimise is as follows:

$$CPF(S) = \sum_{m=0}^{M} parity(S_m), \; parity(S_m) = \begin{cases} C_{even}, & \text{if } \sum S_m \text{ is even} \\ C_{odd}, & \text{otherwise} \end{cases} . \tag{5.2}$$

where S is the solution representation, M is the number of parity functions, S_m is the subset of S that parity function m uses as its variable inputs and C_{even} and C_{odd} are constants. The global solution is where all modules are even (or odd depending on the constants used in the problem). Coffin and Smith (2007) showed that the concatenated parity function causes the hierarchical Bayesian Optimisation Algorithm to fail. However, this problem is not difficult to solve (Chen and Yu, 2009): a local-search algorithm using single-bit mutation can successfully find the global solution as each module is only ever a single-bit mutation away from the optimal partial solution. Consequently, a model is not required to solve the problem.

The Hierarchical Pairwise Allelic Independent Function problem (Iclanzan, 2011) attempts to construct a problem that requires rescaling parity modules to higher orders of organisation and consequently require a model. The lower-level modules are each a parity function. The modules are then combined using the same construction as the Hierarchical If and only If problem Watson et al. (1998). The representation of a module parity function is its parity value, 1 or 0, for an odd or even module, respectively. However, the problem did not cause an exponential running time for MBOAs. We hypothesis that this is due to representing solutions using a binary unit. Therefore, a model only needs to learn two partial solutions for each module and not the parity function in order to search at the higher-level representation to satisfy higher-order dependencies. In doing so, the complexity of the pairwise independence is removed.

The parity function module example that causes pairwise independent functions appears as an extreme case. Martins and Delbem (2016) proposes that a multi-modal function can create pairwise independent functions. Specifically, when observing a distribution of solutions, the statistics show independence between variables. In this case, it is not that modules are explicitly pairwise independent, as in the concatenated parity function and Hierarchical Pairwise Allelic Independent Function problems. Rather, when observing a distribution of solutions, their interactions appear pairwise independent. Therefore, we do not need to restrict the modules to the extreme case of parity functions. We exploit this observation to construct a pairwise independent function that contains a sub-set of partial solutions that create differences between the partial solutions to appear independent. This problem characteristic is expected to differentiate the performance of DO from all other algorithms studied in this thesis.

5.2 Synthetic Optimisation Problem

In this section, we investigate the types of problem structures that distinguish the capabilities of MBOAs. Specifically, to identify problem characteristics that cause a polynomial vs exponential time complexity differentiation between the MBOAs. Previous works comparing these algorithms have not provided examples of such a distinction (Thierens and Bosman, 2013; Hsu and Yu, 2015; Goldman and Punch, 2015). Here, we construct a problem to allow explicit evaluation of the problem characteristics that have been identified in chapter 4 and section 5.1 that are hypothesised to cause exponential running times for some MBOAs and not others. The problem is carefully developed such that if an algorithm can overcome the problem characteristics, finding a global optimum takes polynomial time; otherwise, the algorithm takes exponential time. This allows for a scaling analysis to identify problem characteristics that an MBOA can do (polynomial running time) and problem characteristics that an MBOA cannot do (exponential running time).

5.2.1 Problem Structure

Chapter 4 identified that MBOAs share the concept of adapting the neighbourhood of a solution to enhance search, but do so in different ways. Therefore, we focus on investigating the differences between the type of neighbourhood compression a model can perform (differences in model capacity and construction) and the differences between the methods used to exploit information from the model to inform search. We do this because an MBOA may be capable of accurately reorganising a solution's neighbourhood (sufficient model capacity) but cannot efficiently exploit the information to explore it. Conversely, an MBOA may be capable of efficiently exploring the neighbourhood of a solution, but the model cannot learn it. To explore these capabilities separately, we use a construction that separates the complexity of reorganising a solutions neighbourhood (model induction) and the type of search to perform in the reorganised neighbourhood (model informed search). We achieve this by separating the fitness function, illustrated in Figure 5.1 that normally maps a solution S directly to fitness, into two parts: a compression mapping C, that maps S into a higher-level binary representation R, and an environmental mapping E, that maps R into fitness; detailed in equation 5.3,

$$F(S) = E(R) = E(C(S)) \quad . \tag{5.3}$$

In this way, C is used to control how difficult it is to induce the model, and E is used to control how difficult it is to find higher-order solutions. The C map (compression) defines the complexity of the neighbourhood reorganisation an MBOA must perform to see the signal for combining variables (provided by the E map). The C maps used here perform a compression that allows each unit in R to vary independently given a suitable change to S. The E map (environment) defines how variables in R combine to provide fitness contributions and, subsequently, provide the signal to find higher quality solutions. The types of E mappings investigated here are relatively straightforward. For instance, a global optimum solution can be found in polynomial time when using a hill-climber or an MBOA when $S{=}R$ (when C is an identity map).

What makes this problem construction useful is that the change at S, informed by the model of an MBOA, required to make a single-unit change at R can be defined independently from how these changes are used to find higher-order solutions. This problem construction enables us to attribute the performance differences between MBOAs to specific problem characteristics related to either the reorganisation of the neighbourhood of a solution, by changing C, or the ability to explore the reorganised solution neighbourhood, by changing E. Next, we describe particular instances that contain the problem characteristics that we want to explore.

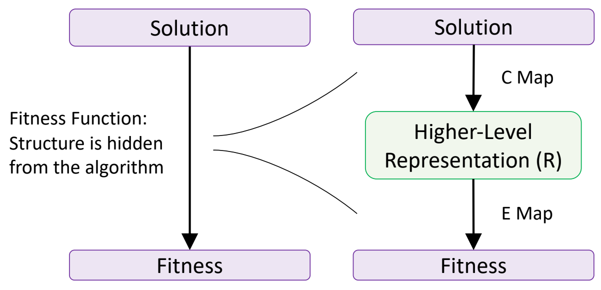


Figure 5.1: We separate a solution to fitness mapping using a higher-level representation R. The C map controls the type of neighbourhood reorganisation an MBOA must perform. The E map controls how search must be performed in the reorganised neighbourhood (higher-level representation) to find higher-order solutions

In introducing this synthetic problem, some unfamiliar terminology is introduced. Table 5.1 provides a glossary of acronyms used.

5.2.2 The Compression Mapping (C): From Solution Space to a New Representation

The Compression Mapping (C) defines the relationships an MBOA model must learn and represent in order to efficiently search at the higher-level representation R and consequently follow the fitness signal. An MBOA can only apply variation to the solution

Term	Description
R	Higher-level binary representation
C	Compression mapping: maps from solution space, S, to R
E	Environmental mapping: maps from R to fitness
BB	Building block: a subset of solution variables
PS	Partial solution: values assigned to a BB
Compression mappings	
NOV	Non-overlapping variation (easiest case)
OV	Overlapping variation (medium difficulty)
NDOV	Non-Distinguishable Overlapping variation (medium difficulty)
NPOV	Non-Pairwise overlapping variation (hardest case)
Environment mappings	
GC	Generating combinations (easiest case)
HGC	Hierarchically generating combinations (medium difficulty)
RS	Rescaling search (hardest case)

TABLE 5.1: Glossary of acronyms and terms used in Section 6

representation S. Therefore, to move efficiently at R, the MBOA must induce the relationships in C such that model-informed search applies the correct variation at S to move at R.

To ensure that C is learnable as the problem size increases, we construct a compression map using a concatenation of modules, illustrated in Figure 5.2 — a familiar construction used for evaluating the performance of MBOAs. Each module maps a disjoint set of solution variables (S_m) to representation variables R_m as detailed in Equation 5.4,

$$R = C(S) = R_1, \ldots, R_m = module(S_1), \ldots, module(S_m) \quad , \tag{5.4}$$

where *module* is a module mapping and m is the number of modules. A module performing a compression from four solution units to two binary representation units is sufficient for the purposes of this paper, i.e., to distinguish the model induction abilities of all the algorithms (compression's to one unit can be solved using a simple model Iclanzan and Dumitrescu (2007)). The compression (two binary units in R) identifies four particular configurations in S (each containing four solution bits) from the sixteen possible that we call partial-solutions (PS). Each PS is represented by a unique binary code at R_m in the two binary units of R. A Combination in S_m that is not a PS is represented at R_m by null values. The E map only rewards non-null values in R. Therefore, in finding a PS to a module, the PS must be substituted with an alternative PS to avoid deleterious fitness changes.

The partial solution set controls the compression complexity of a module. We synthetically determine this to induce particular characteristics in the higher-order unit substitutions required in S to vary between partial solutions (i.e., the change required to move to an alternative partial solution). Note that there exist other operators to move between

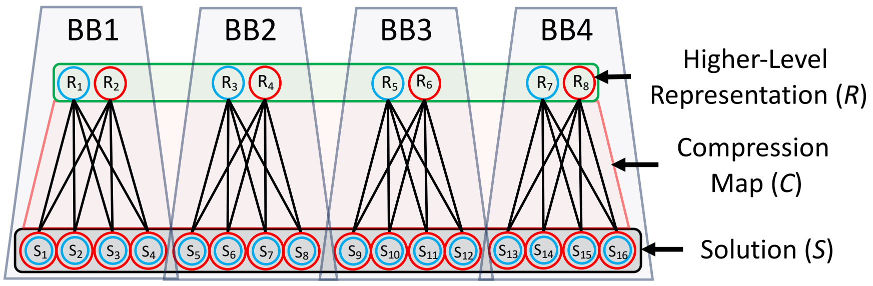

BB1
BB2
BB3
BB4
Higher-Level Representation (*R*)
Compression Map (*C*)
Solution (*S*)
R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10} S_{11} S_{12} S_{13} S_{14} S_{15} S_{16}

FIGURE 5.2: The problem is constructed using a concatenation of modules (building-block 1, building-block 2 etc). Each module contains a mapping from solutions variables (S) to binary representative variables (R).

solutions; we specifically refer to the substitutions that allow movements from a partial solution to its nearest partial solution using a single substitution, i.e., a unit step makes a movement to a neighbouring solution in the compressed representation (a single-unit change in R). The types of characteristics explored in this thesis are: non-Overlapping Variation (nOV), Overlapping Variation (OV), non-Distinguishable Overlapping Variation (nDOV) and non-Pairwise Overlapping Variation (nPOV). For all compression mappings, local search can find a partial solution to each sub-function easily; however, local search will not be able to vary between solutions as, at least, a two-unit substitution is required to move between partial solutions. Therefore, searching in the space of modules to solve dependencies between the modules (defined by environment mapping textitE) requires a variation of a higher-order organisation. Figure 5.3 illustrates, for each variation set type (module mapping), the variation required in S to make a change in R.

To perform a higher-order variation that moves between two partial solutions, the MBOA's model must learn a successful compression such that the search can move between the partial solutions. Figure 5.3 provides a schematic of the solution space for a module, the fits configurations within the space (partial solutions highlighted) and consequently provides an insight into the type of compression a model must learn for each case in order to be able to search in the space of partial solutions. The figure represents the paths between configurations via single-variable changes. For all scenarios, moving between fit solutions requires at least a two-variable change. The orange line represents the paths to other fit solutions using a two-variable change. Consequently, for a model to move between fit solutions, the model must compress the space such that the solutions along this path are not visited. This schematic helps to distinguish the characteristics between each compression mapping. For the non-overlapping variation scenario Figure 5.3.a, the path between two fit solutions is separated from any other. Consequently, a model can compress the space between these two solutions, without affecting the compression of other paths. For overlapping variation Figure 5.3.b, the path between two fit solutions (such as 0000 and 0101) shares a path between other

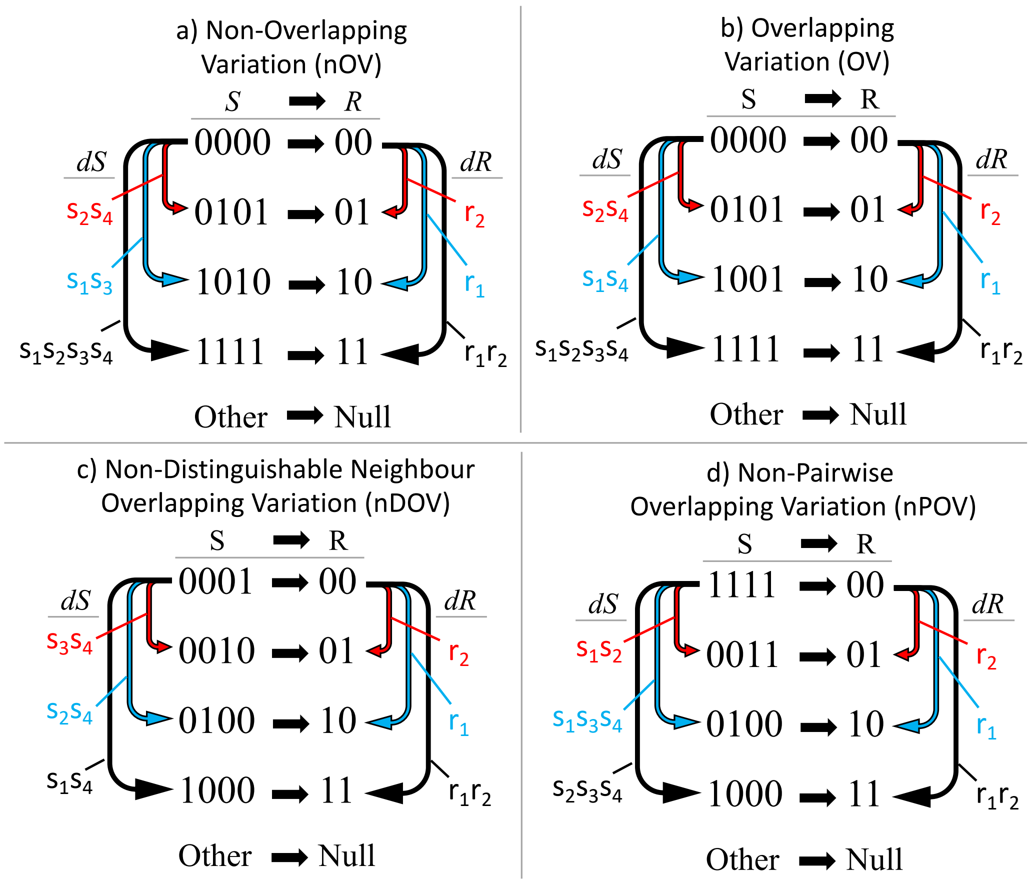


FIGURE 5.3: A module contains four partial solutions of size four each represented by a unique two-bit binary code at R. Alternative combinations are represented by nulls. The set of partial solutions controls the substitution, ΔS, an MBOA must make to a solution to move to an alternative partial solution and consequently make a higher-order unit change at R. A multi-unit substitution in S is required to make a single-unit change in R. The characteristics of multi-unit substitutions explored are (a) non-overlapping, (b) overlapping, (c) non-distinguishable overlapping and (d) non-pairwise overlapping. Examples of the variables substituted in order to move from a partial solution to an alternative partial solution are presented.

fit solutions (such as 0000 and 1001). Here, compressing a path affects the compressing between other fit solutions and a model must therefore be able to account for this. For non-Deterministic Overlapping Variation Figure 5.3.c, all paths overlap, and any fit solution is reachable within a two-unit substitution from any other fit solutions. The complexity of this mapping is difficult to understand. We include this section as it is found to have an interesting behaviour in Chapter3 and has applications in more applied problems (one-hot representation for assignment). For non-Pairwise Overlapping Variation Figure 5.3.d, there exist two distinct groups that are separated by a path of three variable change. Therefore, in order to move between all solutions, the model must learn a compression over a path of three variable changes. This schematic has provided an alternative view of the challenges caused by each compression mapping. Precisely what causes the challenges to the model requires deep investigation. However, this schematic may provide an insight into how these compression mappings can be generalised. Further, in section 5.3.4.4 we demonstrate that the challenge of learning each compression mapping is different with the non-Overlapping Variation being the easiest case and non-Pairwise Overlapping Variation being the most challenging case.

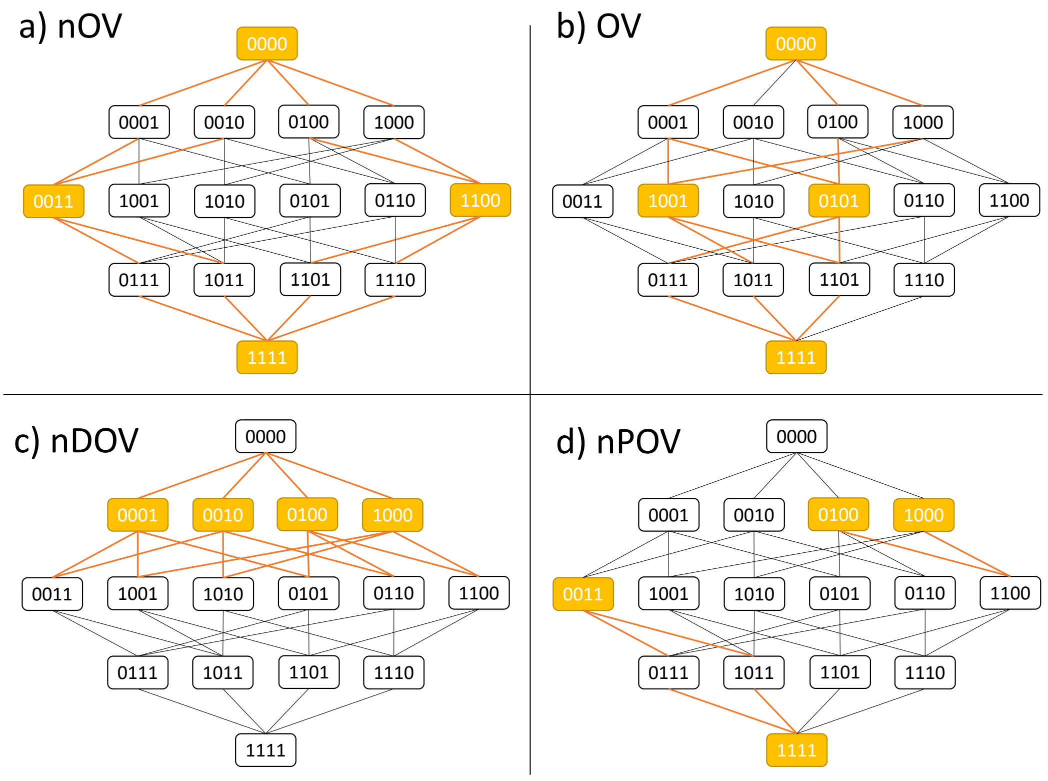


FIGURE 5.4: A module of dimension four contains four fit solutions that are highlighted in yellow. Moving between a partial solution, in all cases, requires a variation greater than a single-variable change. A model is required to compress the representation such that search can move between the fit (highlighted solutions) without moving to sub-optimal solutions first (non-highlighted solutions. The orange lines represent a path is two variable substitutions away to an alternative fit solution and as such demonstrates the compression the model must learn, such that these fit solutions become neighbours. For non-overlapping variation (a), the path between the two fit solutions is separated from any other. For overlapping variation (b), the path between two fit solutions (such as 0000 and 0101) shares a path between other fit solutions (such as 0000 and 1001). For non-Deterministic Overlapping Variation (c), all paths overlap, and any fit solution is reachable within a two-unit substitution from any other fit solutions. For non-Pairwise Overlapping Variation (d), there exist two distinct groups that are separated by a path of three variable change.

5.2.2.1 Non-Overlapping Variation (nOV)

The non-Overlapping Variation case presents the baseline complexity where substitutions performed to move between partial solutions do not overlap. All MBOAs can represent and overcome non-Overlapping Variation relationships (Pelikan et al., 2003; Goldman and Punch, 2015; Hsu and Yu, 2015). The set of partial solutions:

{0000,0101,1010,1111} create an instance of higher-order substitutions that do not overlap. Specifically, the single-point substitution (ΔS) that changes a partial solution to an alternative neighbouring partial solution (a hamming distance of 2 away) is accessed by a substitution in the set $\Delta S = \{\{s_1 s_3\},\{s_2 s_4\}\}$, where s_n represents that the value of variable n is changed. In the case of non-Overlapping Variation, each entry is disjointed from all other entries, i.e., a variable is one element of the substitution set.

5.2.2.2 Overlapping Variation (OV)

The Overlapping Variation case presents a complexity where higher-order substitutions contain overlap. The set of partial solutions {0000,0101,1001,1111} is an instance that contains overlap in the set of higher-order substitutions. Specifically, the single-point substitution (ΔS) that changes a partial solution to an alternative neighbouring partial solution (a hamming distance of 2 away) is accessed by a substitution in the set $\Delta S = \{\{s_1 s_4\},\{s_2 s_4\},\{s_1 s_3\},\{s_2 s_3\}\}$. In the case of OV, each entry is not disjoint, i.e., a variable is in multiple elements of the substitution set.

In the example illustrated in Figure 5.3.b, for partial solution 0000, the single-point substitutions that change the partial solution to a neighbouring partial solution share the variable s_4. For example, improving S, in the case of OV, the substitutions that changes S to an alternative partial solution is $\{s_1,s_4\}$ or $\{s_2,s_4\}$, where s_4 is shared. Whereas in the case of no overlap (nOV), the the substitutions that changes S to an alternative partial solution is $\{s_1,s_3\}$ or $\{s_2,s_4\}$, where no variable is shared.

Concretely, our definition of overlap refers to the set of substitutions required to make a higher-order change. If a higher-order substitution set is disjoint from all other substitutions, the problem does not contain an overlap; otherwise, the overlap is present. Importantly, in this synthetic problem, the overlap cannot be bypassed; i.e., a decomposition of these substitutions causes a deleterious fitness benefit. If this were not the case, the complexity could reduce to non-Overlapping Variation (Mills, 2010). We ensure this property in our synthetic problem construction by using the null representations for non-partial solutions and only rewarding non-null values in R.

The overlap problem characteristic has been previously hypothesised to be a limitation of the modelling capacity of the Linkage-Tree Genetic Algorithm. However, experimental studies that use synthetic problems containing a type of overlap have failed to show an exponential running time for the Linkage-Tree Genetic Algorithm (Thierens and Bosman, 2011; Pelikan, 2010; Pelikan et al., 2011; Hsu and Yu, 2015). In these cases, the property of overlap generally refers to the dependencies for calculating a fitness contribution (Kauffman et al., 1993; Sherrington and Kirkpatrick, 1975; Yu et al., 2005; Wang et al., 2013). Instead, we refer to overlap as property between linkage information in the set of substitutions that move between partial solutions.

5.2.2.3 Non-Distinguishable Overlapping Variation (nDOV)

The non-Distinguishable Overlapping Variation case is a compression mapping that is inspired by the 4-2-4 Encoder problem (Ackley et al., 1985). In chapter 3 we showed that a pairwise correlation matrix (used by the restart Hopfield Network with Generative Associations algorithm) was not able to learn this structure, but an autoencoder model was (used by the restart Autoencoder with Generative Associations algorithm). This, in turn, led to the development of the Model-Informed Variation method used by DO, and we, therefore, include it here to evaluate the performance of other MBOAs. We call this type of overlap 'non-distinguishable' as variations between all partial solutions are of size two. Therefore, there is no distinction regarding locality information between partial solutions using the hamming weight (all alternative partial solutions are a hamming distance of two away). We are particularly interested in this compression because the set of partial solutions are all '1-hot' solutions. Subsequently, this case shares similar characteristics for solutions to Knapsack or discrete assignment problems encoded as a '1-hot' representation. Therefore, we hypothesise that applied problems constrained by assignments will require a similar type of substitutions presented by this complexity.

The set of partial solutions, {0001,0010,0100,1000} creates the instance used in this thesis that contains non-distinguishable overlap. Specifically, the single-point substitution (ΔS) that changes a partial solution to an alternative neighbouring partial solution (a hamming distance of 2 away) is accessed by a substitution in the set $\Delta S = \{\{\mathrm{s_1 s_2}\},\{\mathrm{s_1 s_3}\},\{\mathrm{s_1 s_4}\}\{\mathrm{s_2 s_3}\},\{\mathrm{s_2 s_4}\},\{\mathrm{s_3 s_4}\}\}$. In the case of non-Distinguishable Overlapping Variation, each entry is not disjoint, and all alternative partial solutions are considered neighbours to a partial solution (all alternative partial solutions are a hamming distance of 2 away). This complexity is considered more challenging than Overlapping Variation as a partial solution's neighbour in R cannot be identified by the hamming distance.

5.2.2.4 Non-Pairwise Overlapping Variation (nPOV)

Finally, the non-Pairwise Overlapping Variation case contains an overlapping complexity where the linkage cannot be identified using pairwise statistics. Specifically, the differences between solutions, and consequently the set of substitutions that moves between partial solutions, appear uni-variate using pairwise statistics — a property called pairwise independent functions (Martins and Delbem, 2016). The set of partial solutions, {1111,0011,0100,1000} creates the instance used in this thesis that contains non-pairwise identifiable overlap. The single-point substitution (ΔS) that changes a partial solution to an alternative neighbouring partial solution (a hamming distance of 2 away) is accessed by a substitution in the set $\Delta S = \{\{\mathrm{s_1 s_2}\},\{\mathrm{s_1 s_3 s_4}\}\}$. The complexity is contained at variabless_1, s_2 and s_3, variable s_4 is used to maintain a consistent module size across module types and thus takes the same value as variable s_3 (does not add to the complexity).

5.2.2.5 Using Pairwise Statistics to Identify the Problem Structure

Figure 5.5 presents the mutual information between a pair of variables for each module type. Specifically, how much information is obtained about variable j from observing the value of variable i. In the case of C=non-Overlapping Variation, the linkage information does not overlap (clear separation). For C=Overlapping Variation, the linkage information does overlap, but there is a separation between variables s_1 and s_2. For C=non-Distinguishable Overlapping Variation, the linkage information does overlap, but there is no signal to separate the overlap into substitutions of size two using pairwise statistics. For C=non-Pairwise Overlapping Variation, variables s_1, s_2 and s_3s_4 appear independent using pairwise statistics yet two variables must change simultaneously to avoid a deleterious fitness effect. For our experiments, a problem instance uses the same compression mapping for all modules in the problem. In all compression mappings, a module can be separated using pairwise information. Therefore, the complexity of the compression mapping (C) is attributed to the particular characteristics of a module mapping, i.e., if an algorithm can learn a module mapping, learning more of the same only increases the complexity of separating each module. The optimal partial solution to use in each module is a function of the dependencies that exist between the modules, which are defined by the environment mapping (E).

5.2.3 The Environment mapping (E) from the new representation to fitness

The environment map (E) defines how partial solutions, and therefore units in representation code (R), interact to provide a higher-order solution. These dependencies can be constructed from a differential in the dependency strengths within and between modules (Watson et al., 2011b). In doing so, satisfying within module dependencies are favoured over satisfying between module dependencies. The Modular Constraint problem used in chapter 3 is an example. The alternative is to restrict the visibility of dependencies between modules until the module has been satisfied (Watson et al., 1998; Pelikan and Goldberg, 2001). The HIFF problem used in chapter 3 is an example. The latter construction is used for this synthetic problem. This removes parameters associated with the number of dependencies and strength of dependencies when defining the higher-order interactions using a differential in the dependency strengths, as this can be an important and potentially sensitive parameter that affects the complexity of a problem (Watson et al., 2011b).

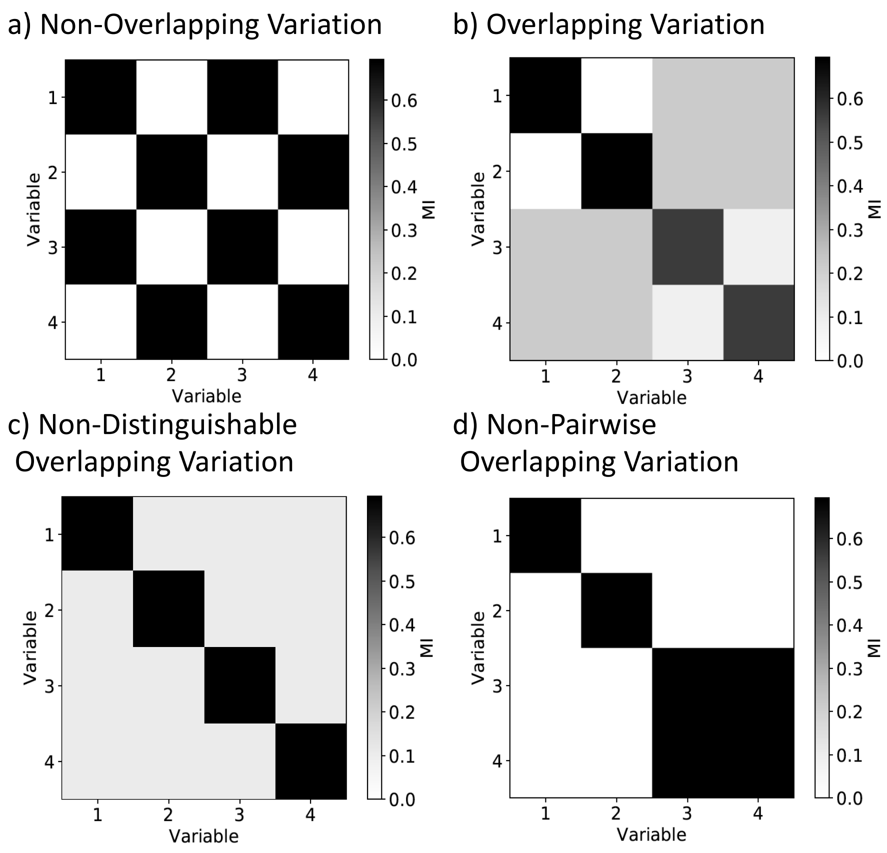


FIGURE 5.5: The mutual information within a module for each compression mapping (C) type.

The partial solutions are easily found by rewarding only non-null values in R using equation 5.5. Specifically, all environment maps have the contribution,

$$F_r = \sum_{i=1}^{m} f(R_i), \qquad f(R_i) = \begin{cases} 0, & \text{if } R_i = \text{null} \\ 1, & \text{otherwise} \end{cases} . \tag{5.5}$$

where, m is the number of modules in the problem.

We explore different E mapping types that induce different search characteristics. The Hierarchically Generating Combinations (HGC) mapping requires an algorithm to rescale variation to higher orders of organisation repeatedly. The Rescaling Search (RS) mapping requires an algorithm to perform a local search in the reorganised representation. Finally, the Generating Combinations (GC) mapping is the baseline case where Model-Informed Variation, Model-Informed Generation and Model-Informed Crossover can easily follow the fitness signal to higher-order solutions. Consequently, E=Generating Combinations tests an MBOA's ability to learn the compression mapping C.

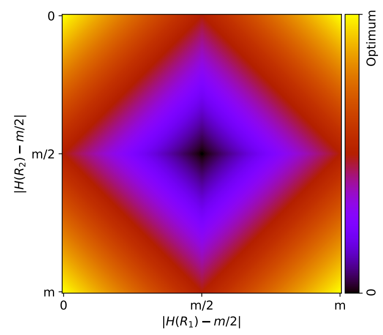
Important to note is that during optimisation an MBOA compresses the search space. Consequently, it is important that during optimisation, the complexity in the compression mapping is not removed (by reducing the set of partial solutions) and is maintained during the entire search process (otherwise the complexity is removed from compression map C). This is achieved by ensuring that all partial solutions are required at all stages of the search process until a global optimum is found (i.e., the loss of the ability to access a partial solution will result in failure of the algorithm to find a globally optimal solution). We ensure this property in the synthetic optimisation problem by separating R into two sets, R_1 and R_2. Each set contains only a single representation unit from each module; thus, each set contains m representation units from the compression of the solution (compression of a module produces two representation units). Finding higher-order solutions compresses the representations sets independently. In turn, a compression only occurs between modules and not within modules, ensuring all partial solutions are required during optimisation and therefore the complexity remains in the compression mapping of a module.

5.2.3.1 Generating Combinations (GC)

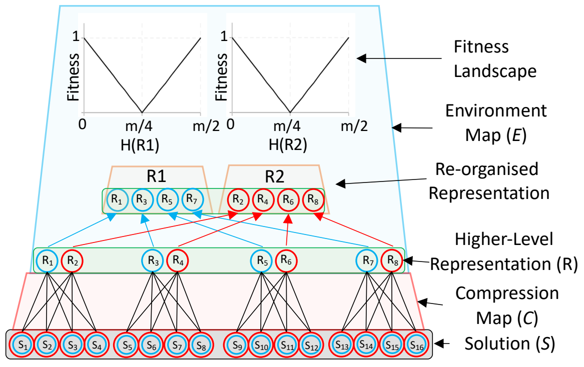
The baseline environment mapping, called Generating Combinations (GC), requires an MBOA to only combine a partial solution one at a time. The partial solution to be chosen is the one that matches with the majority of the partial solution present in the current solution. Consequently, the signal that leads to a globally optimal solution can be identified from a distribution of randomly generated solutions that conserve partial solutions. Thus, all MBOAs, regardless of the differences between the methods used to exploit information, can efficiently follow a signal to higher-order solutions. Therefore the E=Generating Combinations mapping evaluates an MBOA's capability to learn the compression map C. The fitness (equation 5.6), is a summation of the Hamming weight $(H())$ distance of each R set from the middle hamming weight, $(m/4)$. The separation of R_1 and R_2 produces a problem containing four global optima with all values in R_1 being equal to 1 or 0, and the same for R_2. Consequently, a globally optimal solution is one that contains the same partial solution in all modules (thus 4 global optima),

$$F = F_r + |H(R_1) - \frac{m}{4}| + |H(R_2) - \frac{m}{4}| \quad . \tag{5.6}$$

Where F is the fitness of a solution, m is the number of building blocks in a solution. An illustration of the fitness landscape is presented in Figure 5.6(a). The combined problem structure (both C and E map is illustrated in in Figure 5.6(b)). For E=Hierarchically Generating Combinations, if an MBOA can learn C, the fitness landscape is easy to navigate. The landscape provides a clear signal to accept variations that increase the consistency of partial solution across modules. If an MBOA fails, it is simply due to an MBOAs inability to learn the compression map C as all methods for moving in the landscape (Model-Informed Generation, Model-Informed Crossover and Model-Informed Variation), and therefore all MBOAs, are sufficient to follow this signal. The separation of R_1 and R_2 produces a problem containing four global optima. The optimal solution for R_1 and R_2 is either all ones or zeros, each global optimum containing the same partial solution for all modules.



(a) Fitness landscape relative to R



(b) Problem structure E=Generating Combinations.

FIGURE 5.6: The Fitness landscape (a) and problem structure for E=Generating Combinations (b). If an MBOA can separate and represent the modules then all model-informed search methods can efficiently navigate to a globally optimal solution.

5.2.3.2 Hierarchically Generating Combinations (HGC)

Interesting cases for MBOAs arise when a subset of low-order components can form high-order components, such as the case of hierarchical problem structure. This nesting combines lower-level components into a higher order of organisation repeatedly. Consequently, searching for higher-order components requires variation between the lower-level components. As the problem contains multiple levels of organisation, the induced problem challenge is reorganising the neighbourhood of a solution to higher orders repeatedly. This reorganisation requires correctly decomposing the problem, representing the solutions to modules at the required levels of organisation and maintaining diversity in the population such that alternative partial solutions are not lost during selection (Pelikan and Goldberg, 2003b, 2006). Here we use the same hierarchical construction used in (Watson et al., 1998) as all MBOAs can overcome this characteristic when C is the identity map or non-Overlapping Variation (Pelikan et al., 2003; Goldman and Punch, 2015; Hsu and Yu, 2015). This is because higher-order combinations can be efficiently identified by generating a distribution of solutions using the lower-level combinations. Therefore, this mapping evaluates an MBOA's ability to represent combinations of substitutions at multiple scales. We name this mapping Hierarchically Generating Combinations (HGC).

The hierarchical dependencies are represented by a binary tree containing $D = log_2(2m)$ layers. Each layer l represents 2^{D-l} parent nodes. Each node withing layer l (r_l)

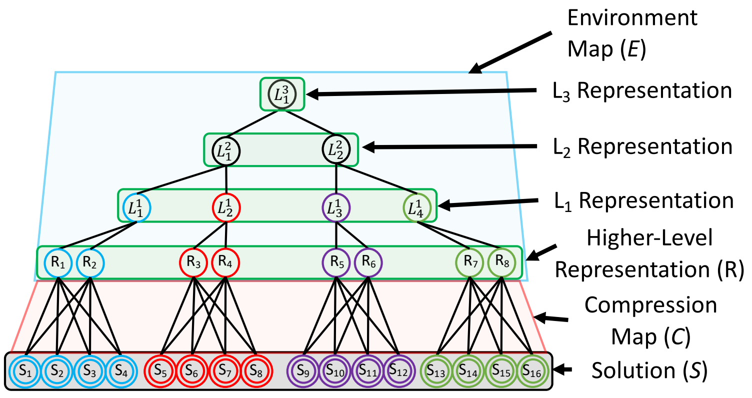


FIGURE 5.7: Problem structure for the Hierarchically Generating Combinations instance. Fitness is calculated as the sum of all nodes in the tree that have a non-null value.

represents an additive combination (A) of two child nodes (r^i_{l-1}, r^j_{l-1}) from the layer below. A parent node represents the common value if the child nodes contain the same value (excluding nulls), otherwise, the value of the parent node is null. Therefore higher-order dependencies are only identifiable when all lower-order dependencies have been satisfied and subsequently higher-order components can only be found by searching in the space that conserves all lower-order components. This compression is provided in Equation 5.7. A parent node with a non-null value is assigned a fitness benefit of 1; nodes with a null value are assigned a fitness benefit of 0. The total fitness, Equation 5.8, is the sum of all fitness contributions from all parent nodes,

$$r^i_{l+1} = A(r^i_l, r^j_l), \quad A(r^i_l, r^j_l) = \begin{cases} r_i, & \text{if } r_i = r_j \\ \text{null}, & \text{otherwise} \end{cases}, \tag{5.7}$$

$$F = F_r + \sum_{l=0}^{D} \sum_{i=0}^{2^{D-l}} f(r^i_l), \quad f(r^i_l) = \begin{cases} 0, & \text{if } r^i_l = \text{null} \\ 1, & \text{otherwise} \end{cases}. \tag{5.8}$$

The problem structure is presented in Figure 5.7. The compression requires an MBOA to repeatedly identify and compress a pair of variation operators from the representation level below to find higher-order solutions.

Depth of Overlap

The compression caused by the hierarchical structure reduces the dimensionality at each layer by combining a pair of nodes at the lower level. We can induce two effects in the search space. The compression can increase the number of variables that overlap in the set of substitutions, or the compression can remove complexity in the modules (reduce overlapping complexity to non-overlapping). Compressing higher-order nodes that are from different modules increases the number of overlapping variables as a higher-order change requires a substitution to multiple modules at the lower-level, each of which contains overlap. Therefore, the number of variables that overlap within a single higher-order substitution grows as the compression depth increases. However, when the compression is performed within a module, i.e., the two representation units of a module are compressed to a single representation unit (four partial solutions to two partial solutions), the higher-order substitution is reduced to one with no overlap as the substitution only needs to move between two solutions and not four.

This property provides a method for evaluating how an MBOAs performance is affected as the number of overlapping variables increases. Specifically, by controlling the layer at which a compression within a module occurs, controls the size of a higher-order substitution that contains overlap and consequently the number of variables that overlap in a higher-order substitution. Figure 5.8 illustrates how the depth of overlap can be controlled. The figure contains a problem of size N=16. Each node on the bottom layer is a solution variable, and each node on a higher level is a compressed representation variable of the solution. The lines between nodes represent the linkage between nodes. At representation L_1, there are four representation nodes colour coded to identify the dependent solution variables. Here a solution variable is a member of multiple representation nodes at L_1. At representation L_2, there are two representation nodes numbered to identify the dependent solution variables. Here a solution variable is a member of only a single representation unit at L_2. Thus, the problem has been reduced from higher-order substitutions containing overlap at representation L_1 to higher-order substitutions not containing overlap at L_2.

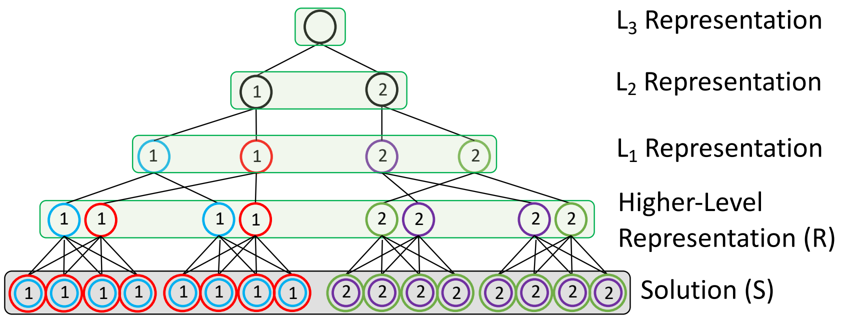


FIGURE 5.8: E=Hierarchically Generating Combinations with overlap up to L_1 Representation. At L_2 Representation a change to a representation unit, e.g, unit 1, causes a change to solution units that are disjoint from other solution. At the L_1 Representation, changes to representation units, e.g., colour red, shares solution variables with other representation units, e.g., colour blue, at the same level. Therefore, L_1 Representation contains overlap and L_2 Representation does not contain overlap.

The ability to control the depth of overlap enables us to evaluate the performance of an algorithm more deeply. We know that all MBOAs can solve hierarchical problems when the compression map C contains no overlap (as in the case when C=non-Overlapping Variation) (Pelikan and Goldberg, 2006; Thierens and Bosman, 2013; Hsu and Yu, 2015; Goldman and Punch, 2015). Therefore, by evaluating the performance of an MBOA with respect to change in depth of overlap (d), instead of the size of the problem (N), we can assess the performance of an MBOA with respect to the number of variables that overlap - the number of shared variables in a single higher-order substitution. An algorithm that can overcome overlapping structures will be insensitive to the number of variables that overlap (change in D).

5.2.3.3 Rescaling Search (RS)

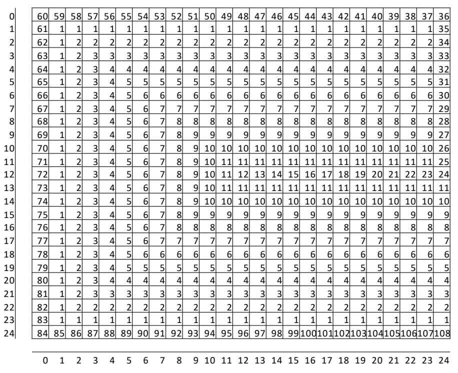
The last environment mapping type we explore is when higher-order components are not found by combining lower-order components. They are found by searching in the space of lower-order components. Therefore, this mapping evaluates an MBOA's capability to exploit information from the model and search in the learned representation (reorganised neighbourhood); hence we name this mapping Rescaling Search (RS). This characteristic is created by defining a unique path (UP) to the global solution. Coordinates provide the location on the path: The Hamming weight of each R subset is used as coordinates for a 2D fitness mapping:

$$F = F_r + UP[H(R_1), H(R_2)] \quad . \tag{5.9}$$

The 2D unique path UP fitness mapping is illustrated in Figure 5.9 and the structure of the problem is illustrated in Figure 5.10. The mapping contains a monotonically increasing slope function that takes any solution towards the start of the path at coordinates $(R_1, R_2) = (m, m/2)$. The path proceeds to coordinates $(m, 0) \rightarrow (0, 0) \rightarrow (0, m) \rightarrow (m, m)$, where (m, m) is the global optimum. Each step along the path increases fitness, and thus deviations from the path cause a deleterious fitness. A non-local change would cause a change to the other coordinate, causing a deviation from the path. Note, the path length scales polynomially with respect to the problem size and is easy to follow via local search in R, thus differentiating it from the long path problem Horn et al. (1994).

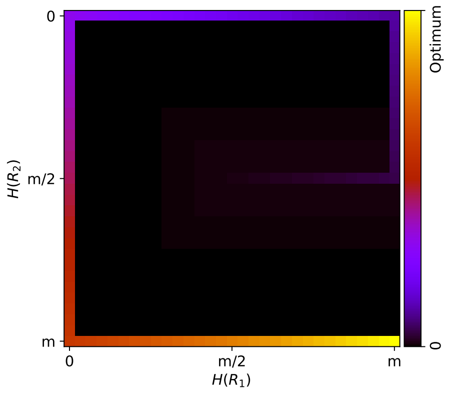
5.2.3.4 Example Solution Trajectories

Figure 5.11 presents examples of solution trajectories, from a random solution initialisation (start of Model-Informed Variation steps) to a globally optimal solution (end of Model-Informed Variation steps) for each environment map (E) type. In the case of E=Generating Combinations, search first identifies partial solutions and then continues by substituting individual partial solution. The accepted partial solution is the one that agrees with the majority in the solution and can be determined by all model-informed search methods. In the case of E=Hierarchically Generating Combinations, search is repeatedly rescaled to higher orders of organisation, where the last adaptive



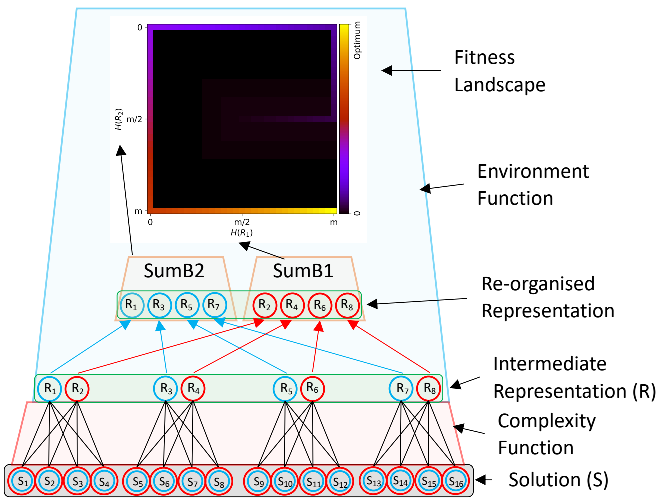
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36
1	61	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	35
2	62	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	34
3	63	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	33
4	64	1	2	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	32
5	65	1	2	3	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	31
6	66	1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	30
7	67	1	2	3	4	5	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	29
8	68	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	28
9	69	1	2	3	4	5	6	7	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	27
10	70	1	2	3	4	5	6	7	8	9	10	10	10	10	10	10	10	10	10	10	10	10	10	10	26
11	71	1	2	3	4	5	6	7	8	9	10	11	11	11	11	11	11	11	11	11	11	11	11	11	25
12	72	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	73	1	2	3	4	5	6	7	8	9	10	11	11	11	11	11	11	11	11	11	11	11	11	11	11
14	74	1	2	3	4	5	6	7	8	9	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
15	75	1	2	3	4	5	6	7	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
16	76	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
17	77	1	2	3	4	5	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
18	78	1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
19	79	1	2	3	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
20	80	1	2	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
21	81	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
22	82	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
23	83	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
24	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108

(a) Absolute Values in Landscape



(b) Fitness Landscape Visualised

FIGURE 5.9: Environment function for the E=Rescaling Search map. The hamming weight of R_1 and R_2 are used as coordinates for the landscape UP



Fitness Landscape
Environment Function
Re-organised Representation
Intermediate Representation (R)
Complexity Function
Solution (S)
SumB2
SumB1
R_1 R_3 R_5 R_7 R_2 R_4 R_6 R_8
R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10} S_{11} S_{12} S_{13} S_{14} S_{15} S_{16}
$H(R_2)$
$H(R_1)$
Optimum

FIGURE 5.10: Problem structure for E=Rescaling Search. The gradient is simple to follow if an MBOA can accurately learn C and perform local search using the model.

variation makes a simultaneous change to half of the solution variables. In the case of E=Rescaling Search, search cycles through multiple partial solution for each module to find the global optimum. Note for E=Rescaling Search all partial solutions are used to search for the global optimum but not necessarily used in the global optimum.

5.2.4 Summary

The combinations of the compression map (C) and environment map (E) create a complex fitness landscape relative to solution (S). The capability of an MBOA to reorganise the neighbourhood of a solution by capturing relationships using the model is evaluated by differences in the compression map (C). The capability of an MBOA to exploit the information from the model to explore the reorganised neighbourhood is evaluated

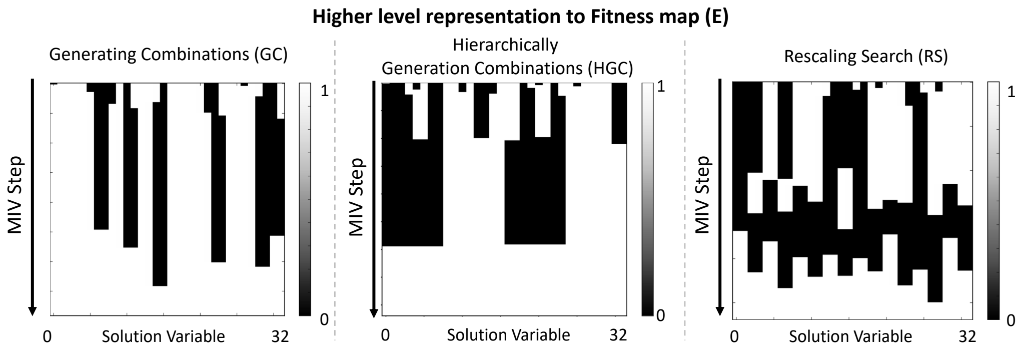


FIGURE 5.11: An example of a solution trajectory, from a random solution (start of Model-Informed Variation) to a globally optimal solution (end of Model-Informed Variation) for each E map when C=nOV.

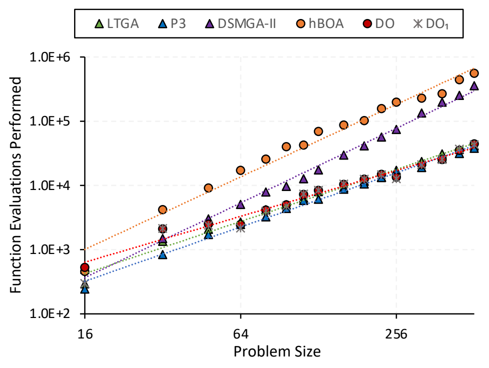
by differences in the environment map (E). The synthetic construction ensures that if an MBOA does not accurately reorganise and search within the neighbourhood, it will take exponential time to find a globally optimal solution. Thus, a scaling analysis will suitably demonstrate if an MBOA can or cannot overcome the problem characteristics. We have created four distinct types for compression mapping: non-overlapping, overlapping, non-distinguishable overlap, and non-pairwise overlapping linkage information types. We have also created three distinct types for environment mapping: generating combinations, hierarchically generating combinations, and rescaling search types. For each type, we have created an instance of the characteristics, which we found to be sufficient to differentiate the performance of the MBOAs explored in this thesis. In our experiments, we explore all twelve combinations to investigate the performance differences between MBOAs.

5.3 Performance Evaluation

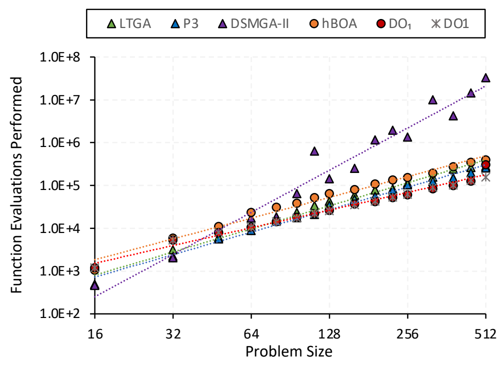
In this section, we assess the performance of the Linkage-Tree Genetic Algorithm (LTGA), the Parameterless Population Pyramid algorithm (P3), the hierarchical Bayesian Optimisation Algorithm (hBOA), the Dependency Structure Matrix Genetic Algorithm (DSMGA-II) and DO on all combinations of complexity and environment maps. Where appropriate, we also include results for DO$_l$ (The autoencoder model in Deep Optimisation is limited to l hidden layers). The performance of an algorithm is evaluated by performing a scalability analysis. Each algorithm is run until the global optimum solution is found. The data points present the average number of function evaluations performed to find the global optimum solution in up to 10 independent runs (for runs with greater than 10^7 function evaluations, three independent runs are performed). For algorithms using a population, the population size is set such that within all independent runs, a global optimum is found. P3 does not use a population and therefore requires no population sizing. For all other algorithms, the population size was found by incrementally increasing the population size until the algorithm was able to find a globally optimal solution for all independent runs. All algorithms use local search at the solution level representation to initialise the population of solutions. In doing so, the solution distribution, before any model-building occurs, contains partial solutions (removing potential challenges associated with finding the partial solutions). Algorithms are terminated if a global solution is not found within 10^9 function evaluations, and therefore no data point is provided. An advantage of LTGA, hBOA, DSMGA-II and P3 is that they do not have additional parameters to tune. A disadvantage is that this does not allow control over the inductive bias. DO, like other neural network methods, has several tuneable parameters. These values were selected from preliminary results on a small sample of problems. Specifically, hidden layer compression ranged from 0.6 to 0.9, with the lower complexity compression of non-Overlapping Variation using the higher compression ratio of 0.6 and the more challenging compressions, such as non-Pairwise Overlapping Variation, requiring the lower compression ratio of 0.9. DO continues to perform a transition until a globally optimal solution is found. Throughout our experiments, the maximum number of layers achieved was 9 (9 transitions performed), in the case of C=non-Distinguishable Overlapping Variation and E=Hierarchically Generating Combinations and problem size 512. Additional parameters were set as follows dropout rate: 0.2, epochs: [100:400] (with more challenging cases requiring more epochs), learning rate = [0.001:0.01] with more challenging cases requiring a smaller learning rate. Regularisation parameters in the range L1 = $[1\times10^{-3}:1\times10^{-5}]$, L2 = $[2\times10^{-3}:2\times10^{-6}]$.. In general, DO used the assign method for searching in the hidden representation. For the compression mapping C=non-Distinguishable Overlapping Variation with environment mapping E=Rescaling Search and E=Hierarchically Generating Combinations, the encode method was used. Greater detail of the parameters used by DO for each experiment can be found in appendix A.2. Note, that the computational time red for parameter tuning and population sizing are not added to the complexity of an MBOA in this thesis. The focus of this thesis is to understand the performance differences between MBOAs due to the types of models used (specifically the difference between model capacity and construction methods) and how the model is used to inform search.

5.3.1 Model Induction

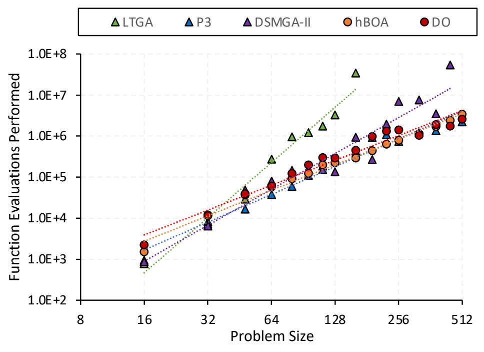
In this subsection, we explore the model induction capabilities of the algorithms by keeping the environment mapping simple and varying the difficulty of the compression mapping. The Generating Combinations environment mapping (E=Generating Combinations) is used to evaluate the performance of a MBOA to learn the compression mapping types. This is because if an MBOA can successfully induce the correct higher-order substitutions (reduce the dimensionality of search), finding a globally optimal solution is easy for all model-informed search methods. Figure 5.12 presents the scalability of all MBOAs.



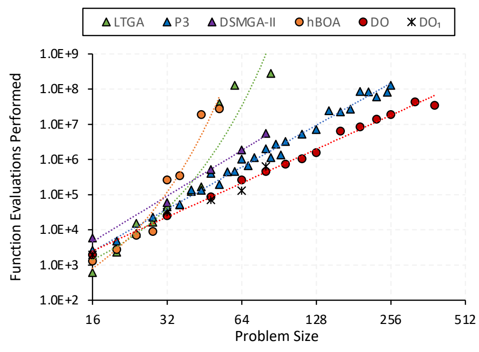
(a) C=non-Overlapping Variation: All MBOAs show polynomial scaling.



(b) C=OV: All MBOAs show polynomial scaling.



(c) C=non-Distinguishable Overlapping Variation: All MBOAs show polynomial scaling.



(d) C=non-Pairwise Overlapping Variation: LTGA and hBOA show exponential scaling. DSMGA-II and DO_1 fail to find solutions for large problem sizes. DO and P3 show polynomial scaling.

FIGURE 5.12: Performance evaluation of an MBOA's model capacity to learn the different complexities in the neighbourhood reorganisation.

For the non-Overlapping Variation mapping type (C=nOV), all algorithms show a polynomial scaling (see Figure 5.12(a)). We know that learning pairwise dependencies is straightforward for all models. Therefore, the results verify that all model-informed search methods are sufficient to find a globally optimal solution in polynomial time for E=Generating Combinations when the model can learn C.

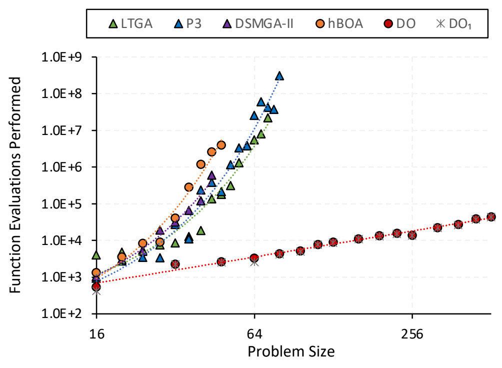
For Overlapping Variation mapping type (C=OV), all algorithms show a polynomial scaling (see Figure 5.12(b)). DSMGA-II shows a significant change in the performance compared to C=non-Overlapping Variation. The degree of overlap is constrained to within modules (between modules, there are no overlapping relationships), the complexity is insufficient to cause exponential running times to LTGA nor P3. Specifically, LTGA and P3 can overcome the overlapping complexity by searching at the level of complete module solutions. There are four possible combinations in this case, rather than at the local neighbourhood, where there are two possible combinations. As such, this makes the algorithms less efficient but does not cause the algorithm to scale exponentially. DO_1 also shows polynomial scaling for this case, indicating that a deep model is not necessary to capture this problem structure.

For the non-Distinguishable Overlapping Variation mapping type (C=non-Distinguishable Overlapping Variation), we observe an interesting result where LTGA shows poor scalability, whereas P3 is successful (see Figure 5.12(c)). P3, hBOA, DSMGA-II and DO all show a polynomial scaling. It is not clear from this experiment how the complexity of C directly affects the model performance in LTGA. For the other algorithms, as with the overlapping complexity (C=OV), we observe that overlap constrained to small modules does not cause exponential running times.

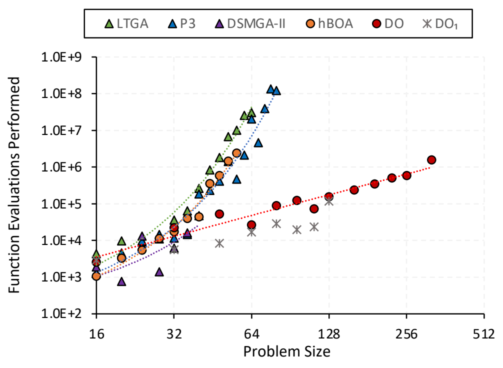
For the non-Pairwise Overlapping Variation mapping type (C=nPOV), DO and P3 are the only algorithms that scale polynomial (see Figure 5.12(d)). DSMGA-II initially shows polynomial scaling but fails to produce a result for problem size greater than 80 and therefore appears to fail abruptly. As expected, the use of pairwise statistics limits the MBOAs ability to capture higher-order dependencies. It was unexpected to observe that P3 was successful. Recall P3 uses multiple linkage-tree models. We hypothesise P3's capability results from using a multi-level representation of the search distribution where the lower-level representations filter out a subset of the partial solutions and thus remove the complexity at higher representations. As it is unnecessary to model all partial solutions (i.e., to find all the global optima), a potential method for finding a global optimum is to use only a subset of the partial solutions and, consequently, reduce the complexity in C. This may be the method used to overcome the complexity in this compression. In general, DO used two hidden layers to be able to find a globally optimal solution reliably were as for other compression complexities, a single layer is sufficient. This can be observed in the results, where DO$_1$ only fails in the non-Pairwise Overlapping case. We explore this further in section 5.3.4.4. For larger problems, up to four hidden layers were used. This is a feature of DO. Specifically, as the problem gets larger, the likelihood of not accurately compressing a single module increases. Therefore, it can take an extra transition to learn a better compression, even though the complexity in the problem does not require the extra hidden layer.

5.3.2 Model-Informed Search

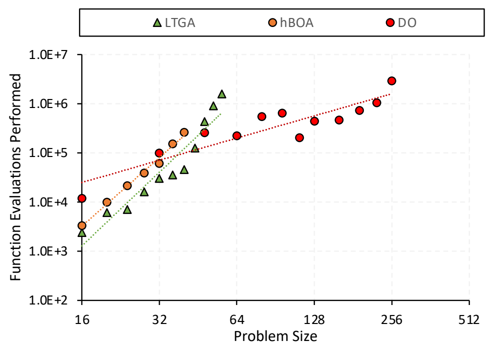
In this subsection, we explore the model-informed search capabilities of the algorithms. The Rescaling Search environment mapping (E=RS) is used to evaluate an MBOA's capability to use higher-order substitutions to explore the compressed search space. Specifically, for this problem, once the solution is on the path, only a single higher-order substitution can improve the solution. In addition, following the entire path to reach the global optimum solution requires each higher-order substitution to be used. Therefore, a model must accurately represent each higher-order substitution, and the model-informed search method must be capable of applying each higher-order substitution. Figure 5.13 presents the scalability performance of all MBOAs.



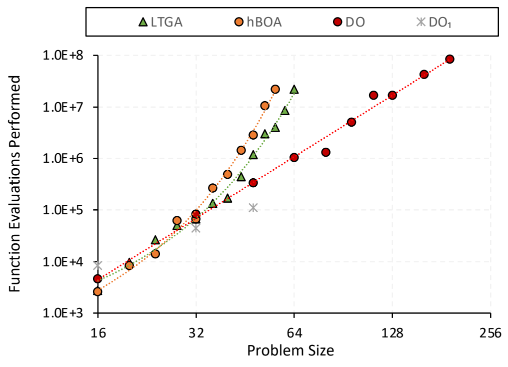
(a) C=non-Overlapping Variation: Only DO shows polynomial scaling.



(b) C=OV: Only DO shows polynomial scaling.



(c) C=non-Distinguishable Overlapping Variation: Only DO shows polynomial scaling.



(d) C=non-Pairwise Overlapping Variation: Only DO shows polynomial scaling.

FIGURE 5.13: Performance evaluation of all MBOAs capability to exploit information from the model to inform search

For C=non-Overlapping Variation, C=Overlapping Variation and C=non-Distinguishable Overlapping Variation mapping types (see Figure 5.13(a) and Figure 5.13(b) and 5.13(c) respectively), DO is the only algorithm that shows polynomial scaling. Therefore, Model-Informed Variation is the only model-informed search method that shows polynomial scaling. We know that all models can learn the C=non-Overlapping Variation mapping type (Figure 5.12(a)); therefore, the failure observed here is due to the model-informed search method only, specifically the Model-Informed Generation and Model-Informed Crossover methods, and not the model induction complexity.

For C=non-Pairwise Overlapping Variation mapping type (see Figure 5.13(d)), we observe that DO_1 fails but DO does not. This further supports that C=non-Pairwise Overlapping Variation requires a deep model to represent the compression accurately. Further, for these experiments, it became increasingly difficult to find solutions for a large problem size (greater than 256). This was primarily due to the time required to tune hyper-parameters for DO. As the problem size increases, the ability to compress each module accurately appears to be reduced. The problem requires all modules to be accurately represented and separated. Consequently, if a single module is not accurately represented, then the algorithm can fail to find the global optimum. We have already seen that all MBOAs except DO are unable to learn C=non-Pairwise Overlapping Variation (evidenced by Figure 5.12(d). Here we also see that MBOAs other than DO are unable to perform local search in the space represented by the model. Here, DO is capable of learning C=non-Pairwise Overlapping Variation such that the higher-order substitutions are separated precisely, and that local search can be performed to follow the narrow path. Concretely, DO is overcoming the combined challenges that other MBOAs fail to do even when the problem challenges are separated.

To further demonstrate that E=Rescaling Search presents a problem challenge that is easy for local search, we perform an experiment with $C=I$, where I is the identity mapping. Figure 5.14 presents the results for all MBOAs and includes a hill-climber that perform single-bit substitutions to S. All algorithms do not use local search at the solution representation (population is not initialised using local search); otherwise, this would not test an MBOAs capability. Instead, the MBOA must learn that each variable is an independent substitution (identity map). The hill-climber was able to find a global solution easily, along with DO. Note, here DO must learn the identity function first and therefore is less efficient than the hill-climber. However, even when there is no complexity in the model induction, other MBOAs, that use Model-Informed Generation or Model-Informed Crossover, fail. This result verifies that failures observed in the experiments with E=Rescaling Search are due to how the model is used to inform search and not due to model-induction difficulty, supporting hypothesis two.

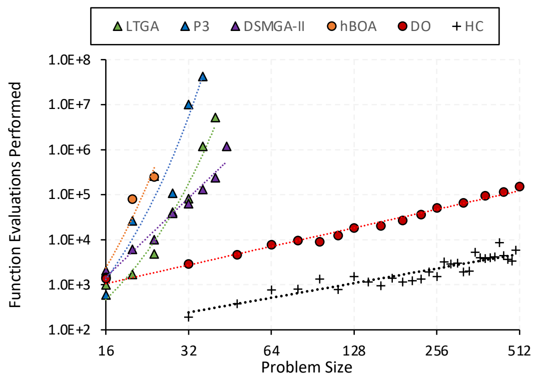
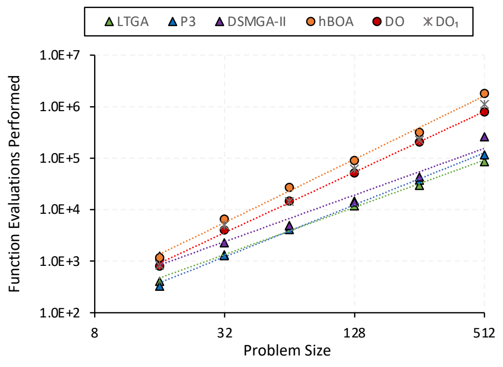


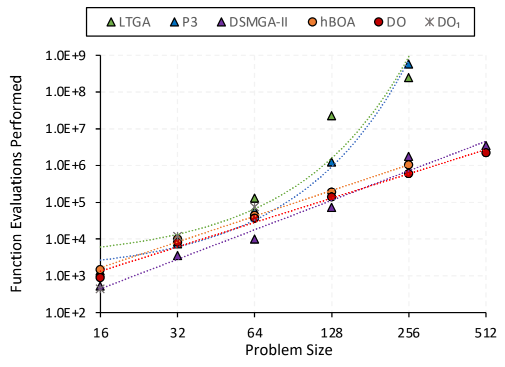
FIGURE 5.14: Local search outperforms MBOAs that use Model-Informed Generation or Model-Informed Crossover and therefore cannot perform local search using the model.

5.3.3 Multi-Level Representation

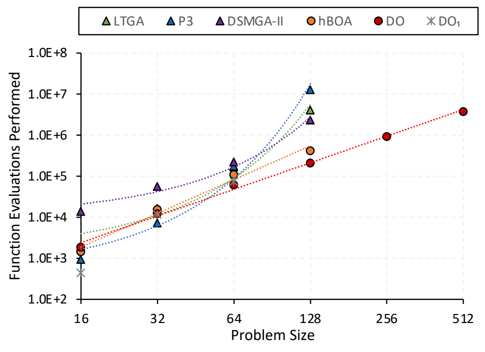
In this subsection, we explore the multi-level representation capabilities of all MBOAs. The Hierarchically Generating Combinations environment mapping (E=Hierarchically Generating Combinations) is used to evaluate the performance of an MBOA to repeatedly induce higher-order substitutions by combining lower-level substitutions (repeatedly compress and reorganise the solution space). For these experiments, the linkage information in E is assigned at random such that overlap occurs between building blocks at all scales of organisation in the hierarchy. Figure 5.15 presents the scalability performance of all MBOAs.



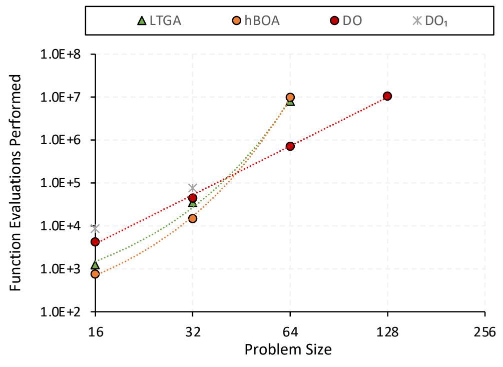
(a) C=non-Overlapping Variation: All MBOAs show polynomial scaling.



(b) C=OV: LTGA and P3 show exponential scaling.



(c) C=non-Distinguishable Overlapping Variation: Only DO and hBOA show polynomial scaling.



(d) C=non-Pairwise Overlapping Variation: Only DO shows polynomial scaling.

FIGURE 5.15: Performance evaluation of each MBOA's capability to recursively reorganise the neighbourhood of a solution to higher orders of organisation. LTGA and P3 show exponential scaling in case of overlap (C=OV). Only DO shows polynomial scaling in the case of non-pairwise overlap C=non-Pairwise Overlapping Variation.

For the non-Overlapping Variation mapping type (C=nOV), all algorithms show a polynomial scaling (see Figure 5.15(a)). We know that pairwise learning dependencies are straightforward for all algorithms. Therefore, the results verify that all algorithms can recursively compress a solutions neighbourhood and all model-informed search methods are sufficient to find a global optimum. Therefore, algorithms showing exponential scaling are unable to learn the complexity in the compression mapping.

For the Overlapping Variation mapping type (C=OV), LTGA and P3 show exponential scaling, whereas hBOA, DSMGA-II and DO show polynomial scaling (see Figure 5.15(b)). The presence of overlap in higher-order substitutions is sufficient to cause methods that use tree-based models to fail. This result is because searching at higher layers of the hierarchical representation requires larger substitutions to be made. In the case of C=OV, these large substitutions contain many overlapping variables (as illustrated in Figure 5.8). This result is the first to demonstrate a problem type that hBOA and DSMGA-II can solve that LTGA and P3 cannot. Further, DO_1 fails because it cannot represent deep problem structure, whereas DO succeeds. The effect of deep representation is further explored in section 5.3.3.1 by performing experiments with a controlled depth of the overlap.

For the non-Distinguishable Overlapping Variation mapping type(C=non-Distinguishable Overlapping Variation), DO and hBOA are the only algorithms that show a polynomial scaling (see Figure 5.15(c)). Although hBOA scaled polynomially in terms of function evaluations, the computational time used by hBOA to analyse the structure of these evaluations exceeded our experimental limitation. We were unable to achieve a single data point for N=256. This suggests that using a Bayesian network to approximate a deep structure causes a significant increase in the time complexity of model induction and is therefore unsuited to problems with deep problem structures of this type.

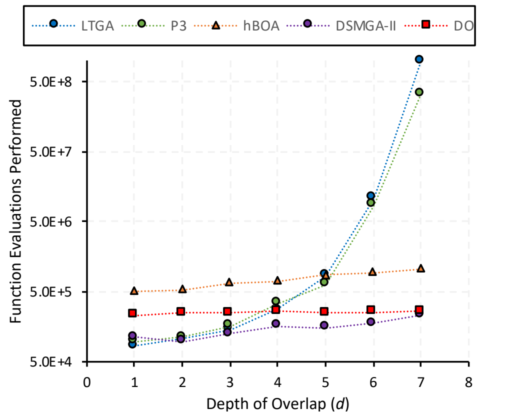
For the non-Pairwise Overlapping Variation mapping type (C=nPOV), DO is the only algorithm to show polynomial scaling (see Figure 5.12(d)). All other MBOAs fail due to the inability to learn C=non-Pairwise Overlapping Variation as evidence by Figure 5.12(d). In this case, DO is able to overcome the challenge of both pairwise independence and deep structure.

5.3.3.1 Problem Depth

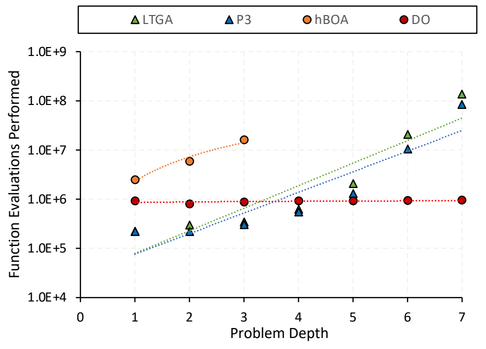
In this section, we perform experiments that control the depth of overlap in the E=Hierarchically Generating Combinations environment mapping case to further our understanding. Our experiments found that the size of overlapping substitutions (due to the hierarchical problem structure) caused significant challenges for MBOAs. By controlling the depth of overlap, we can explore how the size of substitution that contains overlap affects the performance of an MBOA.

The depth of overlapping modules can be controlled by limiting the layer at which linkage is shuffled in the hierarchical structure of E=Hierarchically Generating Combinations. Therefore, we can explore how the depth of overlap challenges these algorithms (whilst keeping the problem size constant). We perform experiments using the C=Overlapping Variation and C=non-Distinguishable Overlapping Variation complexity for problem size 256 and change the depth, d, at which linkage in E is shuffled. At layers greater than d no overlap is present in the higher-order substitutions, as previously illustrated in Figure 5.8. Thus d controls the maximum size of a higher-order substitution that will contain overlap and consequently the maximum number of variables that can be overlapping. Further, we include results for when DO is limited to shallow representations where the depth of the autoencoder is limited to L hidden layers, denoted DO_L.

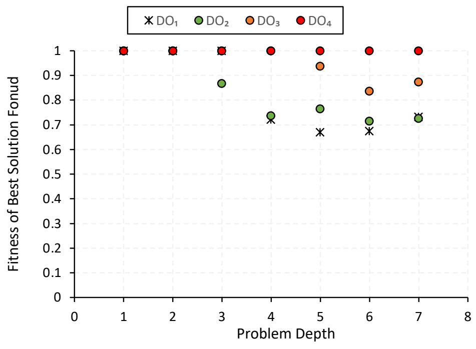
We expect that deeper autoencoders (larger L) will be required to solve problems with deeper overlap (larger d).



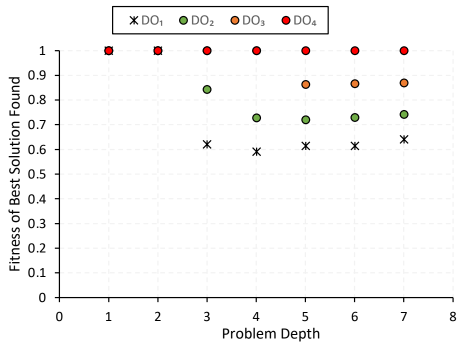
(a) For C=OV, N=256, an increase in overlap depth causes LTGA and P3 to scale exponentially.



(b) For C=non-Distinguishable Overlapping Variation, N=256, an increase in overlap depth causes LTGA and P3 to fail.



(c) Fitness of best solution found by DO limited to different autoencoder depths for For C=OV, N=256.



(d) Fitness of best solution found by DO limited to different autoencoder depths for For C=OV, N=256.

FIGURE 5.16: The effect of depth of overlap on the performance of an algorithm. LTGA and P3 are sensitive to the change in the depth of overlap (as the size of a substitution required to find a higher-order solution containing overlap increases). The depth of the model used by DO is also sensitive to the depth of overlap, and consequently, a shallow model is not capable of representing the problem structure to find a global optimum.

Figure 5.16(a) and Figure 5.16(b) show that LTGA and P3 are sensitive to the size of the overlap in the problem. Specifically, as the depth of the overlap increases, the number of function evaluations required to find a globally optimal solution increases significantly. DSMGA, hBOA and DO show no significant sensitivity and therefore shows that the complexity of overlap is not a challenge for the model induction methods of these algorithms. Finally, we show that representing this complexity using the autoencoder model requires a multi-layered network. Figure 5.16(c) and Figure 5.16(d) show the fitness of the best solution found by DO in its normal operation (not limited to maximum depth) and depth limited versions in 10 repeats. The result shows that as d increases, only deeper models can find a globally optimal solution. Limited models fail to find the globally optimal solution once the problem depth becomes greater than the model depth. Further, as the model depth increases, the ability to perform higher-order substitutions increases and, consequently, so does the ability to find solutions with higher fitness. Therefore, as the depth of overlap increases, the depth of the neural network required to capture the problem structure efficiently also increases.

5.3.4 Discussion

To recall, from section 4.3, the hypotheses for the types of problem characteristics that differentiate the performance of MBOAs are:

1. the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm use a strict binary tree to represent the dependencies between variables and are therefore limited to representing non-overlapping dependencies. For DO, the hierarchical Bayesian Optimisation Algorithm and the Dependency Structure Matrix Genetic Algorithm, this is not the case. Consequently, when higher-order substitutions have overlapping solution variables, the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm will necessarily fail, whereas DO, the hierarchical Bayesian Optimisation Algorithm and the Dependency Structure Matrix Genetic Algorithm will not.
2. Model-Informed Variation is the only model-informed search method that performs multiple steps in multiple dimensions of the reorganised neighbourhood of a solution. Therefore, when a fitter solution is found by searching in the reorganised space, rather than just combining higher-order units, Model-Informed Generation and Model-Informed Crossover (all MBOAs excluding DO) will necessarily fail, were as Model-Informed Variation (performed by DO) will not.
3. All MBOAs, at some point, use pairwise statistics between variables to construct the model, whereas DO does not. Consequently, when the set of higher-order substitutions appears independent, when measured by pairwise statistics, all MBOAs will necessarily il to search in the higher-order organisation, whereas DO will not.

The MBOA distinctions we conclude from these experiments are:

1. the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm fail to learn and exploit overlapping higher-order substitutions, which therefore causes exponential running time as the size of overlap increases, the hierarchical Bayesian Optimisation Algorithm, the Dependency Structure Matrix Genetic Algorithm and DO are successful here (see Figure 5.15 and Figure 5.16). This result distinguishes the induction capability between using a tree of graph data structure to capture the relationships between variables, supporting hypothesis 1.

2. All MBOAs, specifically Model-Informed Generation and Model-Informed Crossover, fail to find solutions in the neighbourhood defined by the model that is not just a combination of higher-order units. DO, Model-Informed Variation, is successful here (see Figure 5.13). This result distinguishes the model informed search capability, supporting hypothesis 2.

3. All MBOAs fail to learn nPOV (pairwise independent) higher-order substitutions. Therefore, this causes exponential running times as the number of operators increases, DO is successful here (see Figure 5.12(d)). This result distinguishes the induction capability of DO from other MBOAs that use pairwise statistics to construct their models. The results support hypothesis 3, however the Parameterless Population Pyramid algorithm was able to overcome the challenge of non-Pairwise Overlapping Variation for the simplest environment mapping (E=Generating Combinations). Therefore, further experiments are required to understand how the Parameterless Population Pyramid algorithm was able to overcome this challenge even though the model using pairwise statistics.

4. A deep representation, and the ability to search in deep representations, are required when generating hierarchical combinations of higher-order substitutions that overlap (see Figure 5.15) and for complex mappings such as non-Pairwise Overlapping Variation (see Figure 5.12(d)). This result distinguishes the capability of deep representations from shallow representations.

5.3.4.1 Overlap

As previously hypothesised, the linkage tree model (the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm) fails when higher-order substitutions overlap (share variables). Specifically, as the size of overlap increases, the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm show exponential running times. For problems containing small-sized overlap (see Figure 5.12(b)), the linkage model is sufficient to bypass the overlap. This is the case for E=Generating Combinations and consequently the linkage tree can represent the complete module, of size four, rather than the two-variable higher-order substitution that moves between solutions within a module (partial solutions). Model-informed crossover then needs to find the correct parent, which contains one of the four complete partial solutions. Therefore, whilst not as efficient as using the correct higher-order substitution that updates the solution with a local neighbour, it is sufficient to overcome the challenge in polynomial time. In the case of E=Hierarchically Generating Combinations, where the number of variables overlapping in a higher-order substitution increases with the increasing level of organisation, the linkage tree fails to model the higher-order substitutions as a complete set — it cannot represent that one group of variables can combine to form two different higher-order groups. Therefore, Model-Informed Crossover fails to perform the higher-order substitutions, causing the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm to fail (see Figure 5.15(b) and Figure 5.15(c)).

We can attribute this failure directly to the introduction of overlap as the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm show polynomial scaling when C=non-Overlapping Variation and E=Hierarchically Generating Combinations (hierarchical and not overlapping). Further, the experiment that explores a change in depth of overlap (and consequently the size of overlap), Figure 5.16(a) and Figure 5.16(b), show that the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm are sensitive to the change in depth of overlap (and therefore size). This is a consequence of the algorithm not being able to capture the overlapping linkage information. The models of the hierarchical Bayesian Optimisation Algorithm, the Dependency Structure Matrix Genetic Algorithm and DO use a graph data structure and are capable of representing overlap and consequently did not fail due to overlap. This is the first result that categorically differentiates the performance between existing MBOAs. Specifically, the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm (linkage-tree model) fail due to overlap and the hierarchical Bayesian Optimisation Algorithm and the Dependency Structure Matrix Genetic Algorithm (models capable of modelling overlapping dependencies) do not. This supports previous hypothesis made about the distinguishing capabilities between these algorithms (Sadowski et al., 2013; Thierens and Bosman, 2011; Hsu and Yu, 2015).

We find that hierarchy alone is not sufficient to require a deep model. DO can find a global optimum for hierarchical structure problems with a strict tree nesting using a single-layered network (using a shallow representation of the multi-level structure) as first verified in chapter 3 and further supported in Figure 5.15(a). For problems containing a graph nesting structure, as caused by the overlap, a multi-level model is required by DO. Therefore, the combination of overlap and hierarchy creates deep problem structure (supported by Figure 5.15 and Figure 5.16). To represent the problem structure in hierarchical problems that contain overlap, the model must maintain lower-order information as these lower-order units are shared between multiple higher-order units (causing overlap). In this case, the model must construct higher-order substitutions from a combination of lower-order units that can also share linkage with other higher-order substitutions. For DO, this requires an extra hidden layer, and therefore requires a deep network model to represent, as seen in the results.

Interestingly, the non-distinguishable overlapping case, nDOV ('1-hot' partial solutions), created an interesting challenge for DO, especially in Rescaling Search, where local search is essential to find the globally optimal solution. The compression learnt by the autoencoder model did not always match the compression designed in the optimisation problem. Concretely, the encoding learned by the autoencoder model can be different to the problem yet still accurately compress all partial solutions. As a result, this causes single-bit substitutions in the latent representation to fail to move to the correct partial solution required by the optimisation problem to find a higher-order solution. Specifically, what is considered a neighbourhood in the intermediate representation of the problem (in R) may not be a neighbour in latent space learned by the autoencoder model. Instead, the partial solutions that are a single-bit substitution away in the problem are a two-bit substitution away in the latent representation (i.e., partial-solutions 0001 and 0100 are encoded as 00 and 10 respectively in the problem, but as 00 and 11 by the autoencoder). To overcome this difficulty, the Model-Informed Variation encode method was developed (detailed in chapter 4) to use the encoder to inform the change to make the latent representation (section 4.2.3.1). Intuitively, this method informs what change to make in the latent representation in-order to change the values of a selected solution variable. In determining which latent variables to change, and then changing them, the decoder network is then used as before to determine how the change at the latent representation changes the solution variables. This method is more robust to the challenge presented by C=non-Distinguishable Overlapping Variation as the encoder can inform the change made to the latent representation to be two bits simultaneously. The difference between this method and the single-bit substitutions can be seen in appendix A.1. We suggest further research into understanding how to improve the search performed at the latent representation of a solution.

The advantage of the linkage tree models is its efficiency in the case of C=non-Overlapping Variation. We observe in Figure 5.15(b) that the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm are particularly efficient compared to the other MBOAs. We hypothesis that this is due to the inductive bias of the models used. The hierarchical Bayesian Optimisation Algorithm and DO use graph-based models, whereas the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid algorithm use a tree model constructed using agglomerative clustering. Thus, the linkage-tree model will always build a hierarchical representation. Hence, the inductive bias of the model is well aligned with the hierarchical problem structure. Consequently, the model can extract weak signals that can be present in the distribution of solutions for how lower-order units combine into higher-order units. Therefore, fewer solutions are required to identify the higher-order components. Whilst the hierarchical Bayesian Optimisation Algorithm and DO can represent a broader range of problem structures, the model space is more extensive and can, therefore, require more data to find the correct model for the problem structure.

5.3.4.2 Rescaling Search

All MBOAs fail to find solutions that require searching in the space defined by the model, rather than combing units, see Figure 5.13 and Figure 5.14. As failure occurs in the baseline compression map (C=non-Overlapping Variation), for which we know all algorithms can represent (evidenced by Figure 5.12(a)), this failure can be directly attributed to the way models are used to inform the search. As discussed in Section 4.3, all algorithms approximate the process of local search in the reorganised neighbourhood defined by the model. However, only DO is capable of exploring all dimensions of the search space repeatedly. Here we see that this significantly affects the performance of other MBOAs to such an extent that local search is sufficient to outperform MBOAs (Figure 5.14). Further, DO was also capable of performing a local search when the compression mapping was complex (C=non-Pairwise Overlapping Variation). As evidenced by Figure 5.13, MBOAs were unable to learn the non-Pairwise Overlapping Variation compression mapping. Here, DO can learn this mapping and perform a local search in this space.

5.3.4.3 Pairwise Independence

All MBOAs, except the Parameterless Population Pyramid algorithm and DO, fail on problems containing non-Pairwise Overlapping Variation operators (see Figure 5.12(d)). This is due to the model's inability to compress the solution space, as evidenced by the failure in the baseline environment (E=Generating Combinations). As hypothesised by (Martins and Delbem, 2016), this is primarily due to the use of pairwise statistics used to detect the linkage information. Further research is required to identify if the models can represent the structure if greater than pairwise statistics are used. Additionally, we show that DO$_1$ also fails here, showing that a multi-layered network is required to learn the compression mapping. We investigate this further in the following section.

5.3.4.4 Learning the Compression

Here, we verify that C=non-Distinguishable Overlapping Variation is the only compression mapping that requires a multi-layer network to learn a compression of a single module. Further, we verify that C=Overlapping Variation and C=non-Distinguishable Overlapping Variation (complexity containing overlap) require at least two units that share input variables to learn the compression. C=non-Overlapping Variation is the only mapping type that can be represented using separated units. To verify this, we performed an experiment where an autoencoder model is trained to learn a compression of the partial solutions for each compression mapping type (I.e., a single module). The autoencoder model is set with the minimum capacity required to compress the space, i.e., four solutions to be represented by two hidden units. Only the set of partial solutions is used to train the autoencoder model. Dropout and regularisation are not used for this experiment.

Figure 5.17 presents the compression learned by a single-layered autoencoder model for compression mappings C=non-Overlapping Variation, C=Overlapping Variation and C=non-Distinguishable Overlapping Variation. For each sub-figure, there are four graphs showing:

1. The top left graph presents the reconstruction error for each partial solution and the average reconstruction error during training.
2. The bottom left graph presents the hidden activation response of the 1st hidden layer of each partial solution after training.
3. The top right graph presents the network weights for the first layer.
4. The bottom right graph presents decoder output when the hidden representation values are set to (11), (1-1), (-1,1) and (-1,-1). This shows if the model can generate each partial solution by making unit changes in the latent representation.

For the C=non-Overlapping Variation, C=Overlapping Variation and C=non-Distinguishable Overlapping Variation mappings, a single-layered autoencoder model is sufficient to learn a compressed representation of the set of partial solutions. We observe that all partial solutions have a low reconstruction error. Therefore, the model can generate partial solutions from the compressed representation well. The latent activation response by encoding each partial solution shows that the encoding of each partial solution is saturated (the activation response is maximised to 1 or -1) and separated by at least one unit change to a latent variable. For C=Overlapping Variation and C=non-Distinguishable Overlapping Variation, the weights of the network show that some input units are connected to both hidden units. Whereas for C=non-Overlapping Variation, an input unit is only strongly connected to a single hidden unit. Consequently, demonstrating that overlap is not present in C=non-Overlapping Variation, but is present in C=Overlapping Variation and non-Distinguishable Overlapping Variation. Finally, all partial solutions can be generated by performing unit changes to the latent representation. Thus, C=non-Overlapping Variation, C=Overlapping Variation and C=non-Distinguishable Overlapping Variation can be learnt using a single hidden layer.

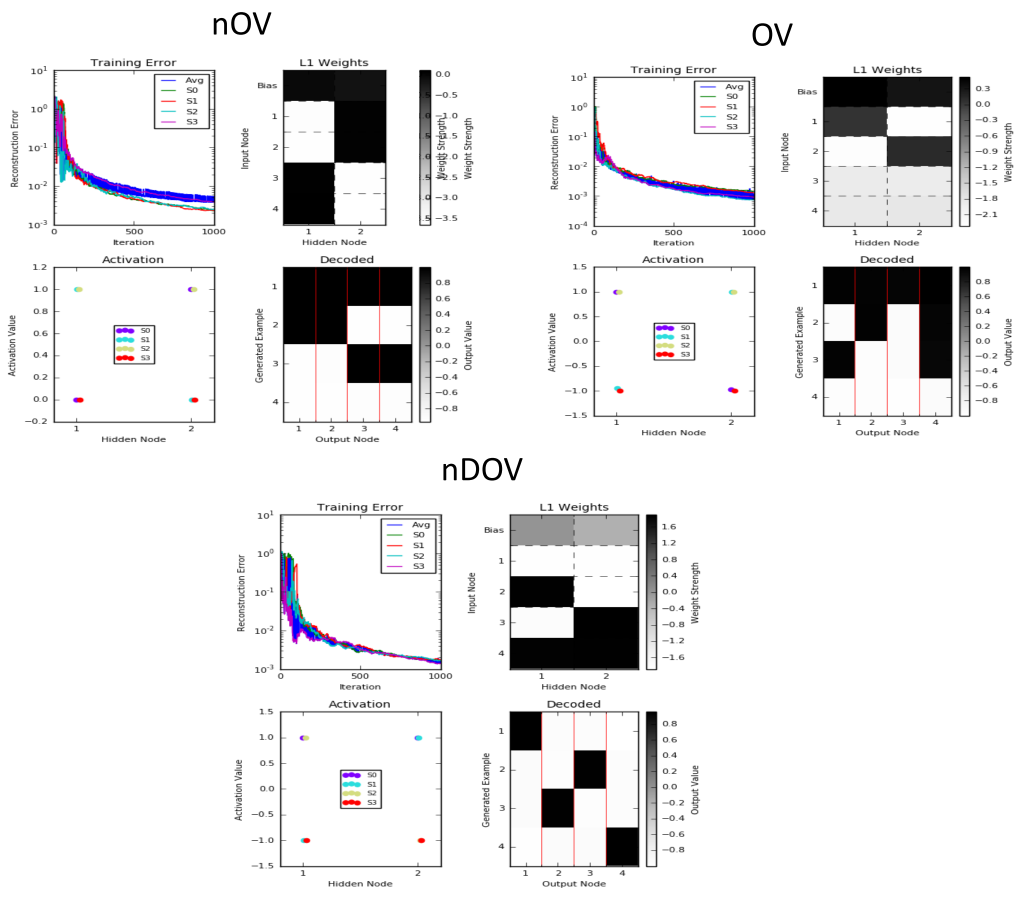


FIGURE 5.17: A single layered autoencoder learning the compression of a single module containing C=non-Overlapping Variation, C=Overlapping Variation and C=non-Distinguishable Overlapping Variation. All modules we compressed successfully.

A more interesting behaviour is observed for the non-Pairwise Overlapping Variation mapping (C=non-Pairwise Overlapping Variation) case. Figure 5.18 presents the compression learnt by a single hidden layer autoencoder of size two; a single hidden layer autoencoder of size three; and a two-layered autoencoder with layer size three (for the first layer) and two (for the second layer).

In the case of an autoencoder with a single hidden layer of size two hidden units (Figure 5.18.a), the model only represents three of the four partial solutions. We can see this in the reconstruction error plot, where a single partial solution has a poor reconstruction error, and the decoder output plot produces “0000” (not a partial solution) and not “1000” (the missing partial solution). In the case of an autoencoder model with a single hidden layer of size three hidden units (Figure 5.18.b), the model can only compress the variables three and four. This is observed in the weights plot, where input units three and four are strongly connected to the same single hidden unit and input units one and two are strongly connected to individual hidden units (representing an identity transformation). The space has been compressed from four dimensions to

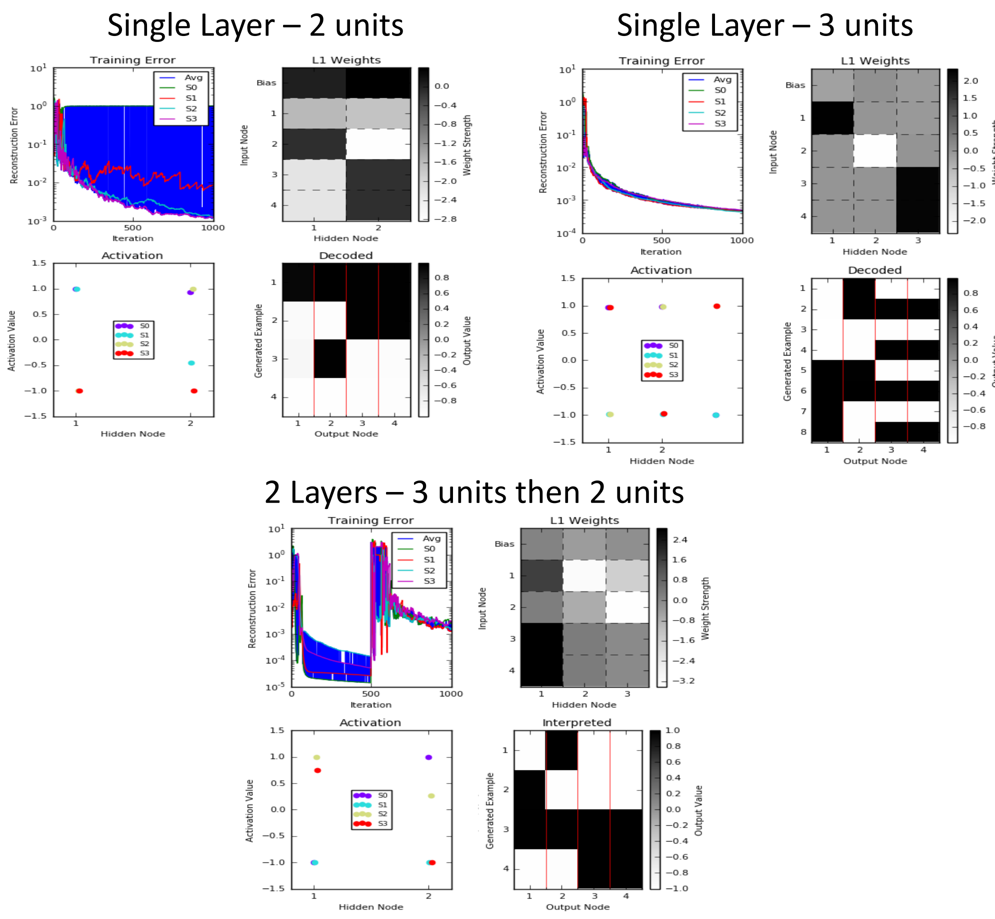


FIGURE 5.18: An autoencoder model learning the compression of a single module containing C=non-Pairwise Overlapping Variation. A single hidden layer fails to compress the space. A two-layered autoencoder model is required to compress the space.

three and generates all partial solutions (evidenced by the reconstruction plot and decoder output plot). However, it has failed to compress the neighbourhood space such that a single unit change at the hidden representation transforms into a change between partial solutions. Therefore, local movements in the hidden representation will cause substitutions to solutions that are not partial solutions, resulting in deleterious fitness. Finally, in the case of a two-layered autoencoder with three hidden nodes in the first layer and two hidden nodes in the second layer (Figure 5.18.c), the model can compress the solution space. Here, the model was trained using the layerwise method (as visible in the error plot where the reconstruction error increased sharply due to introducing a the 2_{nd} hidden layer). All partial solutions now show good reconstruction. The activation response of the second hidden layer shows that the encoding is saturated and separated by single-unit changes. Finally, all partial solutions can be generated.

Here, we have shown that mapping C=Overlapping Variation and C=non-Distinguishable Overlapping Variation require two hidden units to represent an accurate dimensional compression of the solution space as input variables are shared between multiple hidden units. In the case of C=non-Overlapping Variation, two hidden units are still required;

however, each unit only represents two input units and therefore presents a separated decomposition of the module. Only C=non-Pairwise Overlapping Variation requires a deep model to represent the compression mapping. However, the experiments performed to evaluate the performance of MBOAs showed that when E=Hierarchically Generating Combinations, C=Overlapping Variation or C=non-Distinguishable Overlapping Variation also required a deep model. In this case, while the compression mapping alone was not sufficient to require a deep model, in combination with the hierarchical structure, a deep model is required. Further research is needed to fully understand the mechanisms that cause the requirement for a deep model. Consequently, the results presented here suggest that multiple characteristics can cause the conditions that require a deep model. This thesis has only identified that pairwise independent functions and hierarchy with overlapping dependencies cause deep problem structure. We envision there exist many more characteristics that require a deep model to represent and that subsequently DO can learn and exploit that other MBOAs cannot.

5.3.5 Autoencoder Model

The number of layers used by DO varied between experiments due to the complexity of a problem instance and the size of the problem instance. In the simplest case of non-overlapping dependencies (C=NOV), a single hidden layer is required to learn the problem structure. However, for large problem sizes, it was observed that DO performed up to 2 transitions for non-hierarchical configurations and up to 3 transitions for hierarchical configurations. For the compression mapping containing overlapping dependencies (C=OV) a similar behavior was observed with DO performing up to 3 transitions for the non-hierarchical configurations. For the hierarchical configuration, the number of layers used by DO closely followed the number of layers in the problem. However, as observed in the non-hierarchical configurations, as the problem size increased the number of layers used by DO increased beyond the number of layers in a problem instance. It is not clear whether the increased number of hidden layers was necessary for large problem instances containing shallow problem structure. An alternative hypothesis is that the effect of updating the model with updated solutions, and consequently solutions containing a stronger signal for good combinations of variables, is what enables the induction of a representation that is more suitable for MIV. Results for DO_1 (the model limited to a single hidden layer) supports this hypothesis, however, further investigation is required as DO_1 also failed on some large problem instances.

Finally, for the C=nDOV complexity, DO often required at least 2 hidden layers to find the globally optimal for the smallest problems instance. However, DO was not able to find solutions for large problem instances. For the large non-hierarchical configurations, we observed up to 5 transitions occurring. For larger instances that failed, we observed that the population of solutions was converging onto a sub-optimal solution.

We hypothesise that the phenomena of catastrophic forgetting McCloskey and Cohen (1989) is a contributing factor. Specifically, information learned at lower-levels that were found useful in early populations can be lost if a later population does not contain this signal (or this signal becomes weaker). This was observed particularly in the E=RS configuration.

Finally, our investigation primarily focused on separating the performance between the start-of-the-art MBOAs with respect to model induction and model informed search capabilities. In particular, we have focused on how a deep neural network can expand the class of MBOAs and consequently focused on what DO can do that other MBOAs cannot. To that end, we have not investigated the reverse. Specifically, what shallow models, or what other MBOAs, can do that DO cannot. It remains for future research to understand if there are other problem types that DO cannot solve (with polynomial scaling) but other MBOAs can. Understanding this would be valuable to help further the development of all MBOAs.

5.4 Summary

To conclude, the problem characteristics that differentiate the performance between MBOAs are summarised in Table 5.2.

Environment Mapping (E)	Compression Mapping (C)	hBOA	DSMGA	LTGA	P3	DO$_1$	DO
Generating Combinations (GC)	Non-Overlapping	✓	✓	✓	✓	✓	✓
	Overlapping	✓	✓	✓	✓	✓	✓
	Non-Distinguishable Overlap	✓	✓	✓	✓	✓	✓
	Non-Pairwise Overlap	×	×	×	✓	×	✓
Hierarchically Generating Combinations (HGC)	Non-Overlapping	✓	✓	✓	✓	✓	✓
	Overlapping	✓	✓	×	×	×	✓
	Non-Distinguishable Overlap	✓	✓	×	×	×	✓
	Non-Pairwise Overlap	×		×		×	✓
Rescaling Search (RS)	Non-Overlapping	×	×	×	×	✓	✓
	Overlapping	×	×	×	×	✓	✓
	Non-Distinguishable Overlap	×		×		×	✓
	Non-Pairwise Overlap	×		×		×	✓

TABLE 5.2: The success or failure of an MBOA to find a global optimum solution in polynomial time. Deep Optimisation is the only method that is successful for all problem types tested.

We have identified problem challenges that differentiate the performance of existing MBOAs from DO using a synthetic problem. In these constructions, we controlled several different problem characteristics, including the depth of the problem structure. We looked for characteristics that one algorithm could solve, and another algorithm could not in the formal sense of a polynomial vs exponential time complexity, respectively. DO was the only algorithm to solve all problem types (i.e., in polynomial time), whereas none of the other methods achieved this. This exploration identified that overlapping higher-order substitutions, non-pairwise identifiable higher-order substitutions and local search in the reorganised neighbourhood are distinct problem characteristics that DO can solve problems that other MBOAs cannot solve. The failure mechanisms of an algorithm, and more specifically the characteristics that differentiate the performance between MBOAs, can be attributed to the model capacity (overlap), model construction (pairwise independence) and model-informed search (local search using the model). Further, the results showing that DO outperforms DO_1 (i.e., DO is limited to a single hidden layer) suggests that it is the deep representation that enables these results. That is, deep learning is needed, and DO learns deep problem structure and exploits it properly. None of the other methods can do this successfully.

In summary, our findings suggest that the use of deep learning principles that have enjoyed success in many other domains also has significant potential in optimisation problems. The deep learning model used by DO is relatively straightforward in comparison to the more advanced architectures available. Nevertheless, the DO approach opens optimisation problems as another area where the techniques in deep learning can be applied.

A question that remains unanswered and of great interest to facilitate the practical use of MBOAs is the presence of characteristics explored here in real-world optimisation problems. The problem characteristic of overlap is relatively straightforward to construct: two functions that share the same arguments. We, therefore, expect this to be present in applied problems. Further, the C=non-Distinguishable Overlapping Variation case represents a '1 hot' assignment to a solution and consequently can be expected to feature in many applied problems (such as assignment problems like Knapsack). Of course, applied problems are not going to exclusively contain a particular characteristic, as we explored here. However, these results show what types of problem structure MBOAs can or cannot exploit. Therefore, given an optimisation problem that contains these characteristics, we can expect some MBOAs to be unable to exploit this structure.

Chapter 6

Application of Deep Optimisation

This chapter applies the DO algorithm to instances of Knapsack, Travelling Salesman and continuous optimisation problems to evaluate DO's performance on problems with complexities such as infeasible solutions, order-based problems and non-binary solutions (assignment problems). Here we show that DO, and more generally the idea of inducing and searching in deep representations of a solution space, can find solutions that otherwise are difficult to find. In the benchmark instances used in this chapter, the underlying problem structure is unknown. Consequently, it is not possible to determine if exploiting lower-order solutions to find higher-order solutions will ultimately lead to the globally optimal solution. Nevertheless, the primary aim of this chapter is to demonstrate that DO can be applied to different optimisation domains that contain additional problem challenges.

6.1 Multi-Dimensional Knapsack Problem

We first evaluate the performance of DO on the Multi-dimensional Knapsack Problem. We use benchmark Multi-Dimensional Knapsack Problem instances from Chu and Beasley (1998). Results for a simple genetic algorithm (GA), LTGA and dBOA (hBOA = dBOA + RTR) are provided by Martins et al. (2013). Here we compare these against results for DO and a variant of DO used as a control. Specifically, because DO has many differences from the other methods, we want to test whether deep representations are improving the problem-solving ability or whether this is due to the other algorithmic differences (such as Model-Informed Variation) or number of layers. To do this, we use DO_1 which is DO limited to use only one hidden layer in the autoencoder model. This control also enables us to assess whether the problem instances have a structure that deep models can exploit that a shallow model cannot. Further, we include results found by a single-bit local search (LS) and a 2-bit local search (2bLS), allowing for item swaps, to illustrate the performance improvement made by using learning models.

The objective of a Multi-Dimensional Knapsack Problem is to assign a set of items that maximises the combined value while within the m knapsack dimensions. Formally, a Multi-Dimensional Knapsack Problem is expressed as

$$\text{maximize} \quad \sum_{j=0}^{N} p_j x_j \quad , \tag{6.1}$$

$$\text{subject to} \quad \sum_{j=0}^{N} w_{ji} x_j \leq c_i, \quad i = 1, \ldots, m \quad , \tag{6.2}$$

$$x_j \in \{0, 1\} \quad , \quad j \in N \quad , \tag{6.3}$$

where p_j is the value of item j, N is the number of available items, x_j is a binary assignment determining if item j is selected, w_{ij} is the size of item j in dimension i and c_i is the total capacity of the knapsack in dimension i. The instances were constructed in the following way. The dimension size for each item w_{ij} was generated from a discrete uniform distribution $U(0, 1000)$. The capacity of each dimension c_i was calculated as $c_i = \alpha \sum_{j=1}^{N} w_{ij}$ where α is called the tightness ratio. Instances with a smaller tightness ratio and or higher knapsack dimension are generally considered more complex. Comparison is made using an instance size of N=100, knapsack dimensions (m) of 5, 10 and 30 and tightness ratios (α) of 0.25 and 0.75.

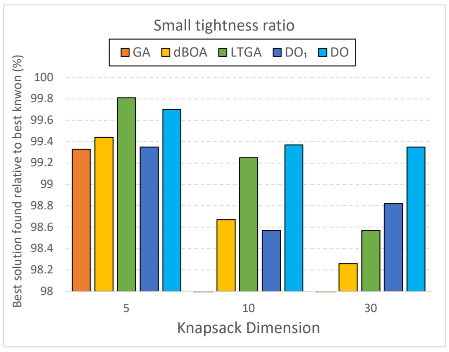
All algorithms, except DO, use a repair operator to overcome the problem challenge of infeasible solutions. Given an infeasible solution, the repair operator iteratively removes individual items, in the order of lowest to highest utility, calculated by $u_j = p_j / (\sum_{i=0}^{m} w_{ij})$, until no constraints are violated. Then, items are iteratively added, in the order of highest to lowest utility, to the solution until an item addition violates a constraint. When this occurs, the item is not assigned, and the repair is terminated. DO does not use a repair operator; if a variation to a solution violates a constraint, it is rejected rather than repaired. DO, therefore, only stays within the feasible region of the solution space. Comparing DO with Estimations of Distribution Algorithms that use a domain-specific repair operator puts DO at a significant disadvantage and serves to demonstrate both the applicability of DO (without using methods specific to this problem domain) and its ability to exploit problem structure in applied problems.

All algorithms use a population size of 1000 and run until the population converges. The population of dBOA and LTGA are initialised using LS to improve the signal of good variable combinations. For DO, LS and 2bLS all solution variables are initialised with 0s (an empty knapsack). The average best solution gap found for 10 instances of a problem type m, α are reported in Table 6.1 and plotted in Figure 6.1.

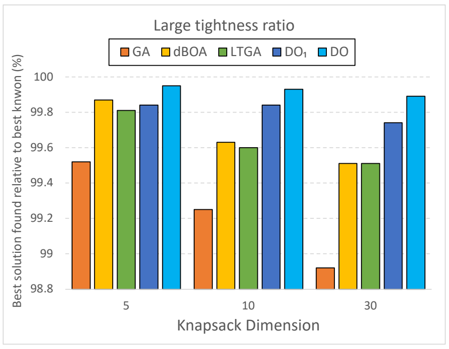
The comparison between LS and 2bLS shows a significant improvement when a variation operator can swap items in and out of the knapsack rather than only add items. This

TABLE 6.1: Performance evaluation on Multi-Dimensional Knapsack Problem instances

		% Gap from Optimum Fitness						
m	α	LS	2bLS	GA	dBOA	LTGA	DO_1	DO
5	0.25	14.69	6.79	0.67	0.56	**0.19**	0.65	0.30
10	0.25	15.44	6.03	N/A	1.33	0.75	1.43	**0.63**
30	0.25	14.91	4.70	N/A	1.74	1.43	1.18	**0.65**
5	0.75	5.02	1.69	0.48	0.13	0.19	0.16	**0.05**
10	0.75	5.66	1.42	0.75	0.37	0.40	0.16	**0.07**
30	0.75	6.03	0.84	1.08	0.49	0.49	0.26	**0.11**
	Model	N/A	N/A	N/A	Bayesian Network	Linkage Tree	AE 1D	AE 6D



(a) Small tightness ratio ($\alpha = 0.25$)



(b) Large tightness ratio ($\alpha = 0.75$)

FIGURE 6.1: The average quality of a solution found compared to the best known solution.

indicates the usefulness of domain-specific operators. The simple GA using a repair operator outperforms LS. dBOA, LTGA and DO all find superior solutions compared to the basic methods. Significantly, the results show that DO appears to provide good results compared to other MBOA's, with the greatest performance observed in the most complex cases (large m). This shows that DO can exploit more structure from the population than other MBOAs. Further, by comparing DO with DO_1, the results show that a deep model is required to exploit this structure.

In these randomly generated Multi-Dimensional Knapsack Problem instances, it is not clear what type of problem structure DO can exploit that other MBOAs cannot. Martins et al. (2014) found that LTGA can be improved by including a 2-point mutation, a repair operator and diversity maintenance. Similar to that used in the original paper (Chu and Beasley, 1998). However, this is not to say the model cannot improve optimisation. Using these additional domain-specific operators is a method for reducing the apparent problem complexity. Due to the random construction of these instances, the random structure may not be well suited for MBOA's. However, the results presented here are a comparison of the models used in MBOA's. Whilst improvements can be made to LTGA, dBOA and DO in the form of using domain-specific methods, for these results, the model is the primary method (or only method in the case of DO) that improves a candidate solution.

These results demonstrate that DO can solve some applied problems better than other methods. Whilst we understand from the experiments performed using the synthetic optimisation problem developed in this thesis that DO is capable of exploiting problem structure that these other algorithms cannot. It remains future work to understand if these problem characteristics are present in the knapsack problem. Nevertheless, DO was capable of finding solutions that DO_1 was not, therefore suggesting that DO's success is due to its ability to exploit deep structure.

6.2 Travelling Salesman Problem

In this section, we apply DO to the Travelling Salesman Problem (TSP). The TSP states: given a number of cities (n), a salesperson must visit exactly each city once and return to the starting city. The optimisation problem is to find the tour S, where (s_i, s_{i+1}) represents a step in the tour that goes from s_i to s_{i+1}, with a minimum cost where $D(i, j)$ represents the cost of travelling between city i and the city j. Asymmetric TSP problems are for cases where at least one entry is $D(i, j) \neq D(j, i)$.

$$Cost(S) = \sum_{i=1}^{n-1} D(s_i, s_{i+1}) + D(s_n, s_1) \tag{6.4}$$

6.2.1 TSP Representation

The solution for a TSP is a list of city locations. The TSP solution used by DO is represented using a connection matrix C of size N^2 where C_{ij} represents an edge. $C_{\mathrm{ij}} = 1$ signifies that j is the next location after i (the remaining entries are 0). There is a total of N connections, where each location is only connected to one other location (not itself) to construct a valid tour. This is a non-compact representation, but it is sufficient for demonstrating DO's ability for finding and exploiting structure. The autoencoder model is trained using the connection matrix representation of a TSP tour.

The autoencoder generates a connection matrix using the decoder network. Each entry in the generated connection matrix is in continuous space. A valid tour is constructed using an algorithm from the generated connection matrix. The algorithm to construct a TSP route is detailed in Algorithm 7. There are two stochastic elements included in the routine. The first is the starting location from which the tour is constructed from. Choosing a random starting position removes the bias associated with starting at the problem defined starting location. The second is selecting the next location in the tour when the output does not provide a positive connection between the current location and the remaining locations. The autoencoder model is trained using solution representations where a positive value represents a connection in the tour between two locations. A negative value indicates no connection in the tour is made between these two locations. When at a location that has all negative connections to other available locations (i.e., the model does not think any location should come after the current location), then to ensure a feasible solution, the next location is randomly selected from the set of locations that have not yet been visited. This algorithm is designed to ensure that learned sub-tours are correctly conserved during future search and a feasible and complete solution is constructed from the generated connection matrix. The construction method resembles the method used by Hopfield and Tank (Hopfield and Tank, 1985) but here, we use a max function rather than a probabilistic interpretation.

Algorithm 7: Interpretation for TSP Solution

```
Set Tour[0] = select, uniformly at random, starting position;
Set ValidLocs = all possible TSP Locations;
Remove Tour[0] from ValidLocs;
ConVec = Vector of size N;
i = 1;
repeat
    ConVec = connection vector generated from Autoencoder for location Tour[i-1] ;
    NextLoc = argmax(ConVec[ValidLocs]) ;
    if  ConVec[NextLoc] > 0 then
    else
        Tour[i] = NextLoc;
    Tour[i] = select, uniformly at random, from ValidLocs;
    remove Tour[i] from ValidLocs;
    i++;
until ValidLocs empty;
Cycle tour until Tour[0] = defined start location;
Tour[i] = defined start location;
```

Here we present an example of the steps performed by the algorithm to construct a tour given a connection matrix generated by the decoder. Table 6.2 is an example of a connection matrix generated by the decoder network.

	Go To Location					
At Location	A	B	C	D	E	F
A	-1	0.8	0.3	0.4	0.1	0.1
B	0.9	-1	0.6	0.3	0.2	0.8
C	1	0.7	-1	0.4	0.1	0.3
D	0.2	0.3	0.4	-1	0.6	0.4
E	0.3	0.3	0.5	0.3	-1	0.5
F	0.1	0.3	0.8	0.5	0.6	-1

TABLE 6.2: The connection matrix of a TSP solution generated by the decoder network

In this example, the algorithm first randomly selects to start the tour location C. All entries in column C are set to -1. This stops the construction process from returning to location C and therefore ensures no cycles are generated in the route construction. The connection vector for C (at row C) is then evaluated. Here, the maximum positive entry is in column A. Therefore, location A is selected as the next location in the tour after C. As location A has been used, all entries in column A are set to -1. The connection matrix after selecting locations C and A is presented in Table 6.3

	Go To Location					
At Location	A	B	C	D	E	F
A	-1	0.8	-1	0.4	0.1	0.1
B	-1	-1	-1	0.3	0.2	0.8
C	1	0.7	-1	0.4	0.1	0.3
D	-1	0.3	-1	-1	0.6	0.4
E	-1	0.3	-1	0.3	-1	0.5
F	-1	0.3	-1	0.5	0.6	-1

Table 6.3: The connection matrix of a TSP solution after locations *C* and *A* have been used to construct a route.

The algorithm continues until all locations have been selected. The tour is then reorganised so that the starting location, defined by the problem instance, is last in the tour (the tour order remains the same). This algorithm ensures that the route can be constructed from any connection matrix. Exploring the design decisions of this algorithm remains further work; however, this provides a basis for DO to be applied to TSP

Search in the latent space is performed as before, i.e., by making a single-unit substitution to latent variables. The only modification used here to apply to DO to the TSP problem is the interpreter function (Algorithm 7) that converts a decoded output to a TSP solution.

6.2.2 Results

Our aim here is to demonstrate that DO can exploit structure from TSP problems. We used six TSP instances from the TSP library (Reinhelt, 2014): three symmetric and three asymmetric and compare them with three other heuristic methods. The results that follow do not show DO outperforming state-of-the-art heuristic methods (these problems are not particularly difficult for Lin-Kernighan Helsgaun algorithm (Helsgaun, 2000)). Still, they show that DO can find and exploit deep structures to solve TSP problems better than shallow methods.

The performance of DO is compared with three local search methods: location-swap, location-insert, and 2-opt. The location swap heuristic consists of selecting, at random, two positions in a TSP tour and swapping the locations. The location insert heuristic chooses a position in the tour at random, removes the location from the position and inserts it at another random position. The 2-opt heuristic (Croes, 1958) involves selecting two edge connections between locations, swapping the connections and reversing the subtour between the connections. For our experiments, DO used the location insert heuristic before transition as it produces good training data for symmetric and asymmetric TSP cases. A local search is then applied to the solution representation when performing a search at the hidden layer of DO (first search is performed at a hidden layer, then search is performed at the solution representation).

The results, averaging over 10 runs, are provided in Table 6.4. DO solves all TSP instances each time, and the number of trials (training examples) is reported in column 5. Columns 6-8 report the percentage difference between the global optimum and the best found solution for a restart hill climber using a heuristic within the trials used by DO to find the global optimum. Note DO used the insert heuristic for all TSP instances; therefore, 2-opt can perform better on some small cases as observed. Columns 9-11 report the percentage difference when the heuristic search is allowed 10000 trials. These results confirm that DO is exploiting structure reliably, as the global optimum is not easily found for the larger instances.

Comparing with the Lin-Kernighan Helsgaun heuristic (LKH) is difficult as the Lin-Kernighan Helsgaun heuristic can potentially find the global optimum in one trial, even though it performs many k-opt moves within the search. Thus, the results from (Helsgaun, 2000) have been reported here to signify the instance difficulty. Specifically, asymmetric cases ‘ftv35’ and ‘p43’ appear to contain characteristics that are difficult for the Lin-Kernighan Helsgaun heuristic as it does not reliably find the global optimum (Helsgaun, 2000), whereas DO shows good performance.

The key to highlight here is that DO does not use any information about the problem instance. Specifically, the costs between cities are hidden from DO. Therefore, information about the problem structure is only induced from the distribution of solutions found. This separates DO from other algorithms that use the cost matrix to enhance the search process.

Problem Instance	Type	Number Locations	Performance DO (%)	Avg Trials req. for DO	Performance Using same trials as DO (%)			Performance Using 10000 trials (%)		
					Swap	Insert	2-opt	Swap	Insert	2-opt
fr26	Sym	26	0	30	4.6	1.2	0.2	0	0	0
brazil58	Sym	58	0	224	17.0	4.0	0.1	0.9	0	0
st70	Sym	70	0	806	25.8	6.1	0.4	20.9	3.9	0.02
ftv35	Asym	36	0	112	1.6	0.3	2.7	0.8	0	1.4
p43	Asym	43	0	393	0.3	0.1	0	0.1	0.02	0
ft70	Asym	70	0	1776	17.1	4.4	26.7	7.2	2.2	23.1

TABLE 6.4: DO exploits useful structure in TSP problems to find the global optimum (column 4-5) that is not found by a heuristic method within the same number of trials (columns 6-8) nor found easily within 10000 trials (columns 9-11). Values report percentage difference from the global optimum.

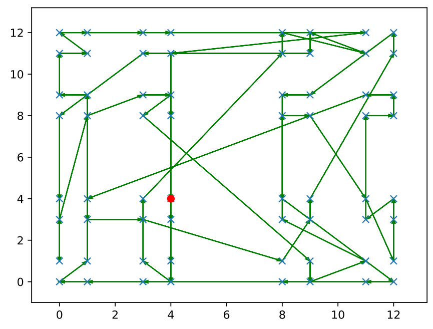
6.2.2.1 Euclidean Hierarchical TSP

Inline with our prior investigations into the types of problem characteristics that cause challenges to state-of-the-art algorithms. We explore some challenges with regard to TSP. Here we investigate the characteristics of hierarchical structure in a TSP problem.

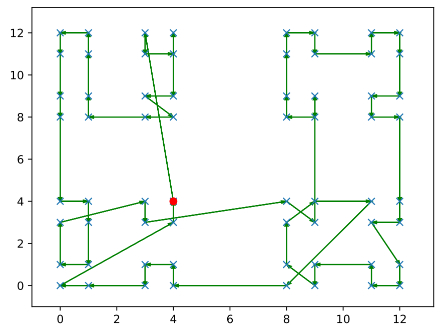
We synthetically construct a TSP problem that contains a hierarchical structure. This is achieved by placing pairs of locations as neighbours at the first level. At the next level, two locations that are a pair (a location-pair) are placed next to a different location-pair, where the distance between a location in a pair at the first level is lower than the distance to any location in the other pair at the next level. This process is repeated and is best illustrated in Figure 6.2.

We then evaluate DO's capability to find the globally optimal solution. Figure 6.2 presents examples of the solutions found when searching at different levels of representation induced by DO. When search is limited to the solution representation level, the solutions found contain low-order structures (see Figure 6.2(a)). These low-order structures are observed as connections between locations that are closest neighbours (the first level location pairs). However, the search becomes trapped as it cannot resolve the rare but large costs between these lower-order structures. When search is then performed at the 1^{st} hidden representation (see Figure 6.2(b)), DO is capable of finding solutions with a higher-order organisation. However, in all but one quadrant, there exist non-optimal configurations for the groups of four locations, which is observed by the diagonal connection between locations. Therefore, searching at the higher-order representation has improved the capability of search to find a higher-quality solution. However, a single layer was not sufficient to find the globally optimal solution. Search at the 2^{nd} hidden representation (see Figure 6.2(c)) is now capable of finding solutions with even higher orders of organisation. In this example, all quadrants contain partial-tours that resemble the tour of the globally optimal solution (shown in Figure 6.2(d)).

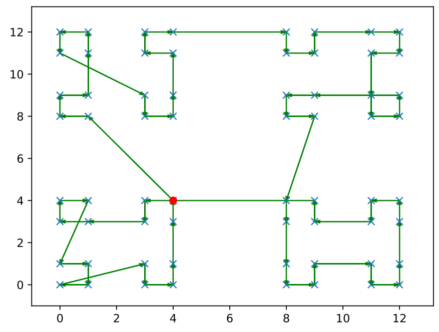
We see that DO is capable of learning the structure of the problem and finds the global solution by first prioritising solutions that satisfy the many connections between nearby locations and then solving the rare but long-range connections between clusters. Interestingly, DO was able to find the global solution just by searching within the 2^{nd} hidden representation (Figure 6.2(d)). Therefore, there is no clear evidence that the hierarchical structure caused the requirement for a deep network to exploit the structure. Further investigation is required to understand this. Additionally, whilst the heuristic 2-opt cannot solve this problem, we found that the Lin-Kernighan Helsgaun algorithm could easily find the global solution. Nevertheless, this demonstrates that DO is capable of learning the problem structure in a TSP problem and exploits this structure in a bottom-up approach where lower-order solutions are found and used to find higher-order solutions.



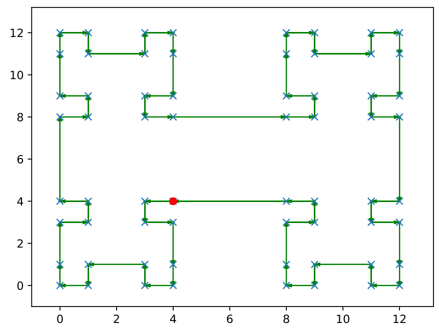
(a) Local search at solution level representation.



(b) Local search at 1st latent representation.



(c) Local search at 2nd latent representation.



(d) Global solution found when searching in 2nd latent representation.

FIGURE 6.2: Examples of the solutions found by DO when searching at different levels of representation induced by the autoencoder model for the synthetic hierarchical TSP problem.

A drawback for DO is using the connection matrix to represent a TSP route. A connection requires a representation size of NxN (where N is the number of locations in a travelling salesman problem instance). Using this representation significantly increases the size of the autoencoder model as the problem size increases, contributing to the computational time for learning. As a result, for problem sizes greater than 100 locations, DO became slow and did not obtain a result. It is hypothesised that the scaling of the connection matrix causes this (where a tour of size 100 locations requires an autoencoder's input to be of size 10000 nodes), affecting the computational time for learning and computation time for reconstructing a tour from the connection matrix. This is a significant drawback for DO as the results presented in this section are for TSP problems that are considered small in the literature. Therefore, in order for DO to be practical for larger TSP problems, DO must be able to learn and exploit structure from a more efficient representation of the route.

6.3 Continuous Optimisation Problem

In this thesis, we have concentrated on binary optimisation problems. The autoencoder model used by DO can easily be adapted to work with continuous variables. This section explores how DO can be applied to a continuous search domain by applying DO to solve the Griewank function.

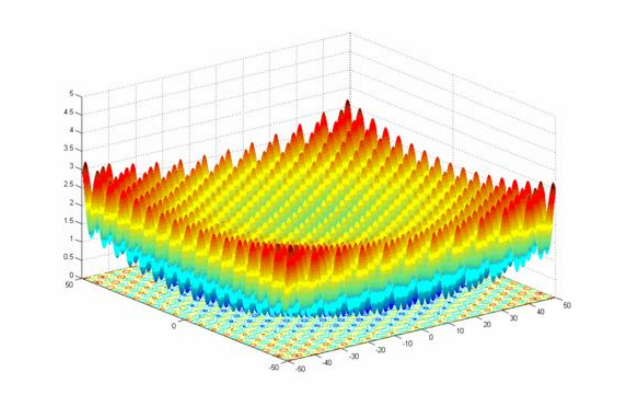


FIGURE 6.3: The 2D Griewank fitness landscape (Hedar, 2020).

The Griewank function creates many local optimal that are regularly distributed across the solution domain space, illustrated in Figure 6.3. There is a single global optimal solution located at $X = (0, \cdots, 0) = 0$.

The optimisation problem is to minimise the objective function,

$$f(x_1, \cdots, x_n) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}),\ -600 \leq x_i \leq 600 \tag{6.5}$$

where n is the number of dimensions. The domain for each dimension is bounded by -600 and 600.

The only modification made to DO to apply it to the continuous domain is the removal of the unit step function used to transform the output from the decoder to a solution. Here, the continuous output from the decoder function is directly used to represent a solution. The local search operator that applies directly to the solution representation is modified to make steps in the continuous space. Specifically, $X' = X \pm 0.001$.

We first perform experiments using a 2-dimensional problem so that we can visualise the results. We use DO with a single hidden layer and a population size of 100. Each solution is initialised using a random uniform distribution $U[-600, 600]$ (applied to each dimension).

Figure 6.4.a illustrates the solutions found by performing a local search at the solution representation (DO before the first transition occurs). All data points represent a locally optimal solution that is trapped due to the rugged landscape. Figure 6.4.b illustrates the solutions found by DO after the first transition occurs. In this case, solutions can escape the local optimum and find a solution nearer to the global optimum solution (located at the point [0.0,0.0]). Figure 6.4.c presents the fitness of each solution found when search is performed before and after a transition in DO (a transition occurs after 100 solutions). Here we see that the fitness of all solutions has been improved. DO has been able to capture the regularity in the optimisation problem and exploit this to escape the local optimal fitness peaks and find higher-order solutions.

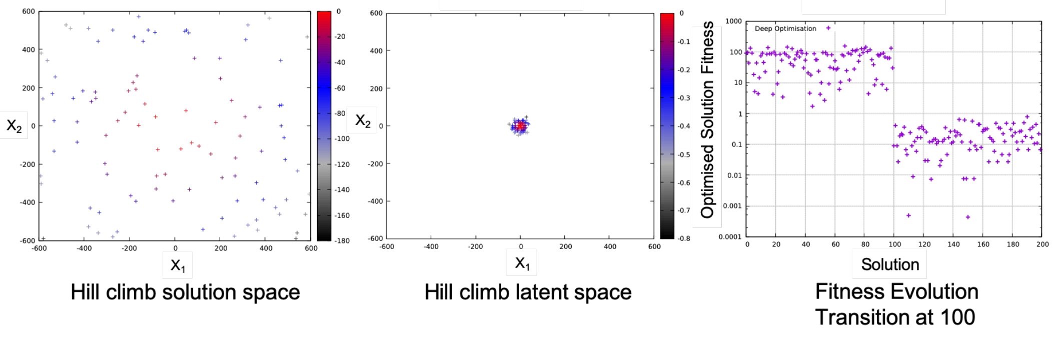


FIGURE 6.4: DO finds the global optimal solution for the Griewank Function of dimension 2 (N=2) with domain size -600:600 and global optimum at $X_1 = X_2 = 0.0$. a) Solutions found by hill climbing in solution space. b) solutions found by hill climbing in learnt latent space. c) first 100 solutions found by hill climbing in solution space and used as training examples for the autoencoder. Solutions 100-200 are found by hill climbing in the latent space, which finds superior solutions.

The model used by DO is initialised using small weights. Consequently, the decoded activation response from the model at initialisation will produce values near the globally optimal solution. To verify that DO is indeed learning and exploiting the structure, rather than an inherent bias caused by the initialisation process of the model, we perform experiments where the location of the globally optimal solution is translated. The cases we explore are with the global optimum located at [300,300], [300,-300], [-300,300] and [-300,-300]. Figure 6.5 shows the results and shows that DO was capable of locating the global optimum in all cases and confirming that DO is exploiting the problem structure to find higher-order solutions.

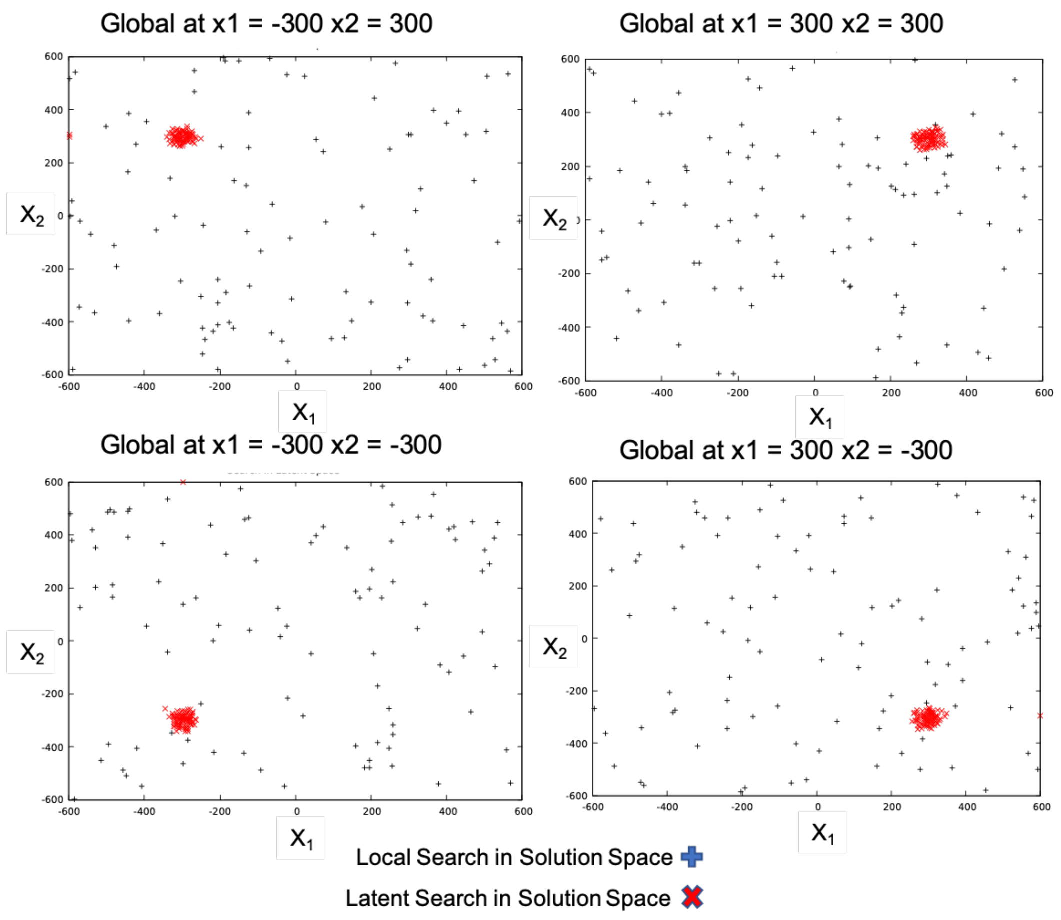


FIGURE 6.5: Variables (N=2) with domain size -600:600. Solutions found by a local-search hill-climbing algorithm in the solution space (blue) and latent space (red) for problems with the global optimum located at (-300, 300), (300,300), (-300,-300) and (300,-300). The search in latent space is reliably able to locate the global optimum.

Finally, we perform an experiment where the global optimum is located at [300, 300] and investigate the cases of no learning; large weight initialisation ($U[-0.1, 0.1]$ instead of $U[-0.01, 0.01]$), and normal operation. Figure 6.6 presents the solutions found by DO. In the no learning case (Figure 6.6.a), as previously suspected, using the model generates solutions located near [0,0]. This is due the small weight initialisation, causing the decoder output to be near 0 (recall DO uses the Tanh activation function). In the case of large weight initialisation (Figure 6.6.b), the model allows solutions to escape the local optimal as the large activation's allow for large random changes to be made to the solution. Therefore, this is analogous to making large random changes to the solution. Whilst an improvement on local search, DO does better where DO can exploit the problem structure and find solutions concentrated around the global optimum. This experiment further verifies that DO is learning and exploiting the problem structure in the Griewank function to find the global optimum.

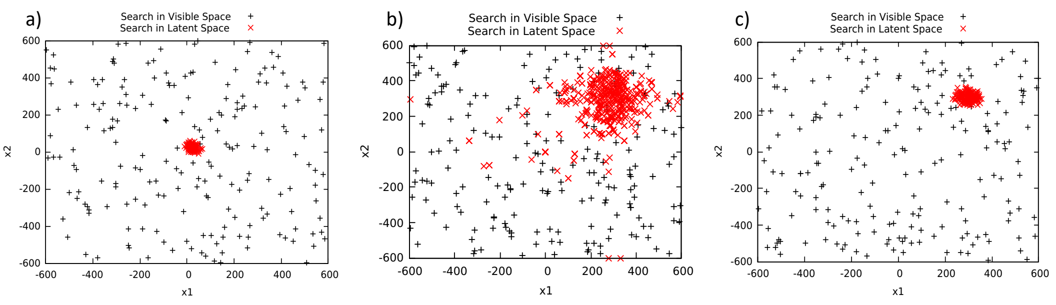


FIGURE 6.6: 2 Variables (N=2) with domain size -600:600. A) Solutions found with no learning and weight initialised using U(-0.01,0.01). b) Solution found with no learning and weight initialised using U(-1.0,1.0). c) Solutions found with learning. The effect of large random weights allows a coarse grain variation across the solution space

Finally, we apply DO to a higher-dimensional problem. Here DO is run on a problem with five dimensions. The global optimum is located at [300,300,300,300,300]. A population size of 500 is used by DO. Table 6.5 reports the best solution found, which was located after the fifth transition in DO (DO constructed an autoencoder model containing 5 layers). The solution found is the global optimum.

	Dimension				
	X1	X2	X3	X4	X5
Best Solution Solution	299.988	300.004	299.986	299.999	299.987
Within global peak?	True	True	True	True	True

TABLE 6.5: Best solution found by DO for Griewank function of size 5

Here we have shown that DO can be extended to the domain of continuous optimisation problems. It remains future work to fully understand how DO can overcome challenges in the continuous search domain. Nevertheless, the work presented here shows that DO can learn and exploit structure in its current form.

6.4 Summary

In this section, we have applied DO to Multi-Dimensional Knapsack Problem, TSP and continuous problem instances. These problems contain problem characteristics of infeasible solutions, discrete solution space and continuous solution space. More work is required to apply DO to more applied problems to make any significant claims about DO's 'real-world' performance. Here, we have demonstrated that DO can operate in different problem domains without domain specific modifications. The idea of search in deep induced representations of the search space remain consistent and only the initial solution representation is modified.

In addition, for a random Multi-Dimensional Knapsack Problem and TSP problem instances, the structure in each instance is unknown. DO, however, was able to learn and use some structure from these instances allowing DO to find higher-order solutions that were not found using simpler methods. However, DO was unable to find the globally optimal solution for all instances. We expect additional evaluation and development will help improve DO on these problems with unknown structures. Alternatively, a reason DO was unsuccessful in finding a global optimum for a problem instance may be due to the problem instances not containing meaningful problem structures. A considerable proportion of the benchmark problems are generated by using a random process. This is done with good intentions to provide an exploration of the problem space. However, DO relies on exploiting structure in the problem to find higher-order solutions. Consequently, it is not easy to quantify for randomly generated problems if the instance contains a meaningful structure. In such a case, we should not be surprised if DO cannot find the globally optimal solution. It remains an interesting and open question whether these applied problems contain any meaningful structure. Recent work has shown that some of the most challenging instances of problems do indeed contain structure (Sleegers and van den Berg, 2020b,a). This suggests that DO may be more suited and more likely to outperform algorithms outside the class explored in this thesis on a particular sub-set of the whole problem space where there exists structure. We also envision that DO can help to investigate this question. For example, by using techniques developed by the deep learning community to try to understand what the model has learned. By doing so, this may be able to reveal the structure of the problem instances, at multiple scales, that have been previously unidentified or not understood.

Chapter 7

Conclusions

The main contribution of this thesis is the development of the Deep Optimisation algorithm. The Deep Optimisation algorithm is inspired by the idea of Evolutionary Transitions in Individuality. In these transitions, the natural evolutionary process is repeatedly rescaled through successive levels of biological organisation. Each transition creates new higher-level evolutionary units that combine multiple units from the level below. For DO, the evolutionary process used is a simple hill-climber, but as higher-level representations are learned, the hill-climbing process is repeatedly rescaled to operate in successively higher-level representations. These higher-level representations collapse the dimensionality of the search space and consequently transform the neighbourhood of a solution. This rescaling enables search to change from searching combinations of primitive variables to searching combinations of variables in a higher-level encoding of the solution space, and so on through multiple levels. Hill-climbing in the compressed solution space allows the search to explore solutions outside the original neighbourhood.

The transition process is emulated in DO by using a deep auto-encoder model. By training an encoder (with a single hidden layer) to compress a distribution of locally optimal solutions, the effective reduction in the degrees of freedom created by selection at lower levels is reorganised. The hill-climbing process then transitions to operate in the compressed representation, encoded in the latent variables of the hidden layer. Variation in this new space can make 'leaps' (coordinated substitutions to multiple solution variables simultaneously) in the original solution space. Hill-climbing in the compressed space can still become stuck at a sub-optimal solution, but these coordinated substitutions mean that new solutions found also have an effective reduction in the degrees of freedom relative to the first level representation. Therefore, an additional transition can be performed by adding further layers, trained using the solutions found by searching at the compressed search space represented by the layer below. This process repeats to construct a deep model.

DO belongs to the class of Model-Building Optimisation Algorithms (MBOAs). MBOAs use machine learning methods and search inspired by evolutionary processes to explore a search space of an optimisation problem. The best-performing MBOAs do not use a deep learning model (DO is the only algorithm that does). In this thesis, we introduce an implementation of the Deep Optimisation algorithm and perform experiments that show that DO is capable of overcoming problem structures that other MBOAs cannot. We explored this performance differentiation using a synthetic problem. In these constructions, we controlled the complexity of the compression of the model that an MBOA must perform and the complexity of the search space an MBOA must be able to explore to find solutions of greater fitness using the model. In doing so, we found problem characteristics that one algorithm could solve, and another algorithm could not in the formal sense of a polynomial vs exponential time complexity. DO was the only method that solved all problem types (i.e., in polynomial time). This exploration identified that overlapping variation, non-pairwise overlapping variation, and local search in the reorganised neighbourhood are distinct problem characteristics that DO can solve that other MBOAs cannot. The results showing that DO outperforms DO_1 (i.e., DO limited to a single hidden layer) verify that it is the deep representation, not other characteristics of DO in isolation, that enables these results. That is, deep learning is needed, and DO learns deep problem structure and exploits it properly. Finally, for the first time, we were able to show a categorical differentiation between the performance of existing state-of-the-art MBOAs. Specifically, the presence of overlapping variation was sufficient to cause the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid to show exponential scaling whereas the hierarchical Bayesian Optimisation Algorithm and the Dependency Matrix Genetic Algorithm (and DO) showed polynomial scaling.

We then applied DO to instances of the Multi-Dimensional Knapsack Problem, Travelling Salesman Problem, and a Continuous Optimisation problem. DO outperformed other algorithms on Multi-Dimensional Knapsack problem instances. This is all the more impressive considering that the other methods using a problem-specific repair operator to handle infeasible solutions and DO does not. For TSP problems, we showed that DO was able to exploit structure and find solutions that basic methods (that do not use learning) could not. Finally, DO was able to explore the search space of a continuous optimisation problem. This exploratory study demonstrates that DO learns the structure of the problem well enough to enable the exploration of feasible solutions directly and deal with non-binary solution representations.

In sum, our findings suggest that the use of deep learning principles, which have enjoyed success in many other domains, also has significant untapped potential in optimisation problems. In this thesis, the implementation of the DO algorithm employed only fairly basic deep learning methods, but the DO approach opens optimisation problems as another area where the vast and rapidly accelerating knowledge in deep learning techniques can be applied. In doing so, we provide the founding understanding of how a deep neural network can be used within the domain of evolutionary computing. DO thus demonstrates, for the first time, that deep learning methods can be used for optimisation problems and provides results that are distinct from existing methods and the understanding of how differences from existing methods effects the performance of the algorithm.

Finally, this thesis has implications on evolutionary theory. Our results show that the adaptive principles of multi-scale evolution can solve problems that single scale evolutionary processes cannot. Importantly, the underlying principles involved are in essence nothing more than local gradient methods in various representational spaces. DO uses local gradients in an error signal to adjust the model parameters during learning, and local hill-climbing in the latent space to discover solutions. Evolution by natural selection is capable of following local gradients in fitness signals and the Evolutionary Transitions in Individuality demonstrate that evolution has the capability to rescale the neighbourhood of the representation space in which it operates. Consequently, the fact that gradient methods can do this in DO suggests that natural selection can achieve the same by following these gradients. We note, however, that DO uses a machine learning model to induce higher-order representations, suggesting a need to better-understand the role of induction in natural adaptation. Put differently, the results suggest that a single-scale model of biological evolution does not capture important features of the natural evolutionary process and that multi-scale model induction is a relevant paradigm with different adaptive capabilities (Watson and Szathmáry, 2016; Czégel et al., 2019).

We now review the claims of this thesis:

- **Deep Optimisation can find solutions that other Model-Building Optimisation Algorithms cannot**

  The best performing Model-Building Optimisation Algorithms (MBOAs) do not use neural network models. The Deep Optimisation algorithm is the first MBOA to use a deep neural network and shows that it is capable of finding solutions that other MBOAs cannot. In chapter 5 we showed that DO was the only algorithm capable of overcoming all problem challenges explored. No other MBOA was able to achieve this result. Therefore, we show that using a deep neural network as the model can improve the performance of an MBOA.

- **A deep model can exploit problem structure that a shallow model cannot.**

  The model space of a deep neural network model provides the capacity to learn and represent multiple-layers of organisation. Consequently, the use of a deep neural network model provides DO with a novel capability. Specifically, the ability to induce and search at multiple scales of organisation. In chapter 5 we showed that DO (capable of constructing a deep network) was able to find solutions to problems that DO limited to a single hidden layer (a shallow representation of the problem structure) could not. The problem characteristics that a shallow model could not overcome were multiple-scales of higher-order overlapping substitutions and higher-order substitutions that contain greater than pairwise dependencies. We then showed that DO limited to a single hidden layer was unable to find solutions that DO in its normal operation could when applied to problem instances of the Multi-Dimensional Knapsack problem and the Travelling Salesman Problem in chapter 6. Thus, the results presented show that there exist deep problem structure (problem structure that cannot be well represented using a shallow network model).

- **Overlap is sufficient to differentiate the performance between MBOAs**

  Previous experiments have failed to distinguish the performance capability between MBOAs. Specifically, it has been previously hypothesised that overlap is a problem characteristic that will cause models that use a tree data-structure (such as the Linkage-Tree Genetic Algorithm and the Parameterless Population Pyramid) to fail where as MBOAs using more sophisticated models that are capable of representing graphs (such as the hierarchical Bayesian Optimisation Algorithm, the Dependency Matrix Genetic Algorithm) will not (Sadowski et al., 2013; Thierens and Bosman, 2011; Hsu and Yu, 2015). In this thesis, we have developed overlapping problem structure that specifically refers to overlap of variables in higher-order substitutions. We show that this characteristic of overlap is sufficient to differentiate the performance between the hierarchical Bayesian Optimisation Algorithm and the Linkage-Tree Genetic Algorithm for the first time. Further, the experiments show that the failure is due to the models used to learn the problem structure and not due to the difference in the methods used to inform search using the model.

- **Model-informed variation can find solutions that other model-informed search methods cannot**

  There exist different methods to use the model to inform search. We characterised these methods into Model-Informed Generation (MIG), Model-Informed Crossover (MIC) and Model-Informed Variation (MIV). It was hypothesised that the method of Model-Informed Variation can exploit problem structure from the model that Model-Informed Generation could not (Mills et al., 2014). In chapter 3, we supported this hypothesis by performing experiments showing that an MBOA using Model-Informed Variation can find solutions to a problem that an MBOA using Model-Informed Generation could not. In chapter 5, we developed an environment mapping that required local search in the reorganised space (defined by the model) to find higher-order solutions. This caused the methods used by MBOAs to fail (Model-Informed Generation and Model-Informed Crossover). DO, and consequently Model-Informed Variation, was the only method able to overcome this problem challenge and therefore distinguishing the capability of different methods used to inform search using the model.

7.1 Further Research Directions

In this thesis, the implementation of DO used a specific deep learning model and particular model-learning methods. However, we envision more advanced implementations that take advantage of sophisticated methods that have proven to be successful in other domains where deep learning has been applied. The application of Deep Optimisation needs to be broadened to other problem domains to further develop the algorithm of DO and understand the types of problem structure in other domains. We summarise the direction of potential future research below:

- Understanding how other deep learning models and advanced methods used in other domains can be used to improve DO. Recent advancements in deep learning have provided state-of-the-art results in other domains. This thesis connects the framework of MBOAs with deep learning, enabling further research to explore how other deep learning models and techniques can be applied to solving optimisation problems.
- Understand how other regularisation techniques can be used to improve the induced representation in DO and how this corresponds to a change in the performance of Model-Informed Variation. Preliminary experiments using sparsity regularisation (Ng et al., 2011) and contractive regularisation (Rifai et al., 2011) did not provide a significant advantage over using a tuned L1 and L2 regularisation on the synthetic problem. However, these regularisation's (or others) may be more suitable for more applied problems.
- A greater exploration of DO on applied problems. This thesis concludes by introducing DO to a small sample of applied problem instances, which showed promising results. A further area of research is to apply DO to more benchmark problem instances and investigate DOs performance, understand the types of problem structure that exists in these problem instances, and the types of problem structure that DO was able to learn and exploit.
- We identified problem characteristics that were sufficient to differentiate the performance of MBOAs. Further research is required to understand if the identified characteristics are 'special' cases or whether it is abundant in other problem instances. Further, we expect other problem characteristics to exist that DO can overcome those other algorithms cannot, and also problem characteristics that DO

in its current form cannot overcome. Identifying more problem characteristics provides a greater understanding of how to further improve DO and, more generally, the class of MBOAs.

- Understanding how Model-Informed Variation can be improved. DO updates a solution by substituting a latent variable and decoding this change to the solution level to construct a substitution to apply to the solution. However, a potential limitation of this method is that it requires a well organised latent representation such as where a single latent variable represents a particular feature and that these features are represented at the minimum or maximum activation strength. Therefore, the development of a more robust exploration method in the latent representation has the potential to improve DO significantly.
- Understanding how Model-Informed Generation, Model-Informed Crossover and Model-Informed Variation can be combined to explore the compressed representation induced by the model in an MBOA. As DO transforms the representation of a solution, Model-Informed Generation, Model-Informed Crossover, and Model-Informed Variation can all be used to find new solutions. Consequently, the process of Model-Informed Generation, Model-Informed Crossover can also be applied to the induced representations. In this thesis, we only explored how Model-Informed Variation can be used in DO. However, Model-Informed Generation and Model-Informed Crossover may provide additional benefits that Model-Informed Variation does not. We therefore envision a process that takes advantage of all approaches.
- Understand if applied problems contain the problem challenges that differentiate the performance of MBOAs and if so, to what degree these problem characteristics are responsible for the performance differences between algorithms. Further, identify the types of higher-order substitutions performed by MBOAs to help identify the types of problem structure that is being exploited for problems with unknown structure.
- It is hypothesised that the Deep Optimisation algorithm is not limited to autoencoder architecture described here. We expect the concept to apply to other architectures that have been widely used in other domains and suitable in the context of Deep Optimisation. For instance, Convolution Neural Networks for translation invariant features

Appendix A

Appendix

A.1 Model Informed Variation Methods

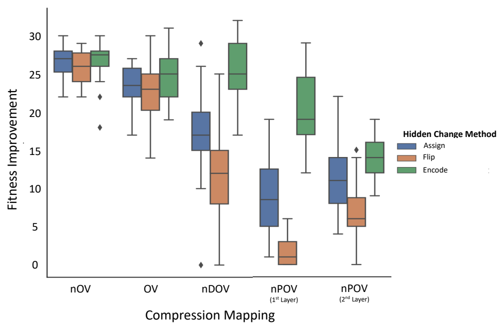
The induced representation of a solution learned by the autoencoder model reorganises the neighbourhood of the problem. However, the neighbourhood induced may not be an optimal reconfiguration that allows single-point substitutions in the latent representation to decode to the correct change at the original solution level. Further, this thesis focuses on binary optimisation problems (binary solution representations), whereas the latent representation is in continuous space (bounded by -1 and 1). DO showed impressive performance using single-bit substitutions in the latent representation. However, a limitation found when performing experiments using the non-distinguishable overlapping variation complexity (C=nDOV) and rescaling the environment (requires local search in the reorganised neighbourhood) was that using a single-bit substitution was not always robust. This was not due to the autoencoder not learning a compression, but instead due to the compression learnt. For C=nDOV, there exists many compression mappings to compress the solution space accurately. Therefore, in some compression's, two PSs will be neighbours (single-bit substitution away), and in others, they will not be (two-bit substitutions away). To overcome this challenge, the MIV encode method was developed. Here, the encoder informs the change to make at the latent representation by observing which latent variables respond to the change in a solution variable at the input. These sensitive hidden units are then changed and the decoder used to produce the solution response. In this section, we perform experiments to demonstrate the differences between MIV methods available to DO.

The experiments are performed by training the autoencoder model using locally optimal solutions relative to the solution representation. Then, for each variation method, 30 solutions are improved by performing the MIV method. The size of substitutions made, the fitness effects caused by each substitution and the number of function evaluations performed are all measured for each solution. Each MIV method used the same model and the starting solutions. All three environments of the synthetic optimisation problem are used to evaluate the performance of each MIV method.

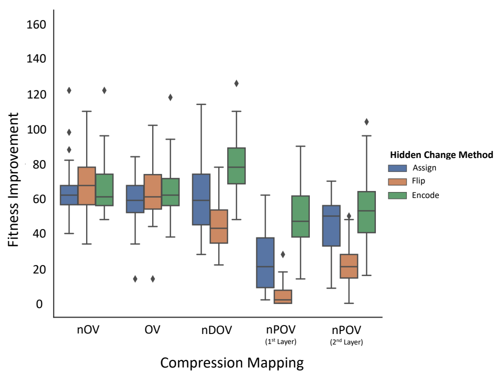
A.1.1 Search at in the Latent Variation

All MIV methods start by encoding the current solution to generate a latent representation of the solution. The methods used to apply a variation to the encoded latent representation are:

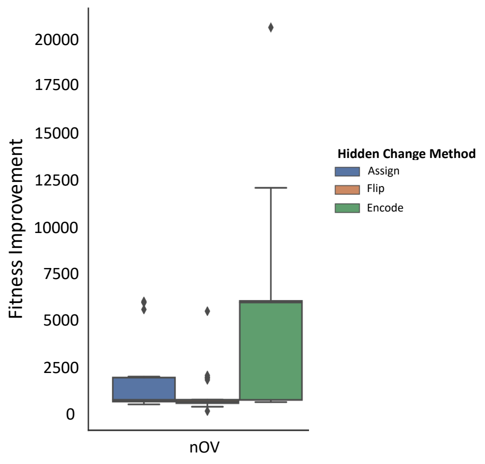
1. **Single Unit 'Flip' :** randomly select a hidden unit and flip the activation value.
2. **Assign:** randomly select a hidden unit and assign the activation value to a maximum or minimum activation value.
3. **Encode:** randomly select an input unit and calculate the sensitivity of a hidden unit's activation response to the change of that input unit. For the hidden units with enough sensitivity, assign the hidden activation values to a maximum or minimum activation value.



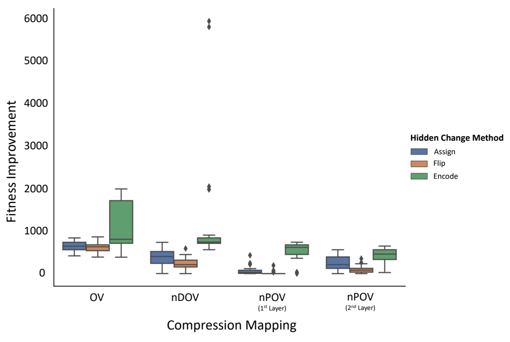
(a) Generation Combinations, E=GC



(b) Hierarchically Generation Combinations, E=HGC



(c) Rescaling Search, E=RS - C=nOV



(d) Rescaling Search, E=RS

FIGURE A.1: Comparison of methods for calculating the latent variation for problems E=GC, E=HGC and E=RS for problem size 128. The results show a distribution of the improvements made to a solution during optimisation. For nPOV, we perform experiments using an autoencoder with a single hidden layer (1st layer) and two hidden layers (2nd layer). The encode method shows a greater robustness for more complex mappings.

Figure A.1 presents the fitness improvements made to all complexity functions for all fitness functions for problem 128. The observations are:

- For the E=GC and E=HGC environment the encode method shows a measurable improvemnt for C=nDOV and C=nPOV. Assign method shows greater performance than single unit 'flip' for C=nDOV and C=nPOV.
- For the E=RS environment mapping, the differentiation between the MIV methods is most prominent. This is not surprising as the environment mapping is designed such that local such must be performed in the latent representation. Here, the encode MIV method is most efficient method and superior for the C=nDOV and C=nPOV compression complexities.

- The encoder MIV method shows the greatest improvement for the C=nDOV complexity.

A.1.2 Substitution to a solution

In this section, the changes made to a solution are explored in more detail. In the previous experiment, we measured the fitness improvement achieved for different types of movement in the latent representation. Here we explore how different methods for interpreting the decoder output can also affect the informed substitution. In these experiments, the latent variation is set to the assign method and kept consistent for all runs. Here, we explore four solution variation methods. These are:

1. **Sampling:** the decoder output is interpreted as a probability vector, and a solution is generated by sampling each output unit and generating a complete solution. The difference between the input and generated solution is considered the variation made to the solution.
2. **Partial Sampling:** the decoder output is interpreted the same as the sampling method. However, in this case both the original hidden representation ($H = E(S)$) and perturbed hidden representation ($H' = H + \Delta H$) are decoded. The difference between the output generated from the original hidden representation and the perturbed hidden representation is considered the substitution made to the solution.
3. **Threshold:** a unit threshold function is used to transform the decoder out put to a solution. Specifically,

$$u(X) = \begin{cases} 0, \text{ if } X << t \\ 1, \text{ if } X \geq t \end{cases}, \tag{A.1}$$

   where t is a threshold value set to $f(0.0)$ (the middle of the activation response used by the decoder output units).
4. **Partial Threshold:** the decoder output is interpreted the same as the threshold method. However, in this case both the original hidden representation ($H = E(S)$) and perturbed hidden representation ($H' = H + \Delta H$) are decoded. The difference between the output generated from the original hidden representation and the perturbed hidden representation is considered the substitution made to the solution.

Note that in the previous experiment, the Partial Threshold method is used. This is the same method used by DO in the thesis (and is described in chapter 4) and used in all experiments presented in the thesis.

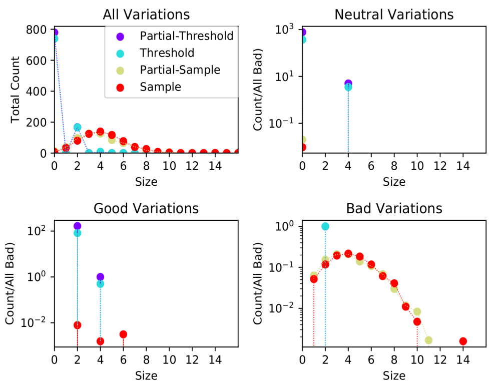
Figure A.2 presents a comparison of the substitutions made using different output methods for calculating the change to a solution for a problem size of 128. The substitutions are categorised according to whether it provided a positive (good) fitness change, a negative (bad) fitness change, or a neutral effect on the fitness. The frequency of good, neutral and bad substitutions is normalised by the number of function evaluations performed.

It is clear to observe why sampling methods perform poorly. For instance, a change to the hidden representation that creates a change to the decoded output that can provide a positive fitness improvement if using the threshold method can be destroyed if one unit is sampled incorrectly (the number of possible substitutions is much greater than the threshold method). In general, partial-threshold is more efficient than the threshold method, which we suspected because the autoencoder model is unlikely to perform a perfect compression and, consequently, a perfect reconstruction. Therefore by comparing the response of a change at a latent variable with the reconstruction rather than the input allows for removing inaccuracies in the reconstruction (and consequently inaccuracies in the learned compression).

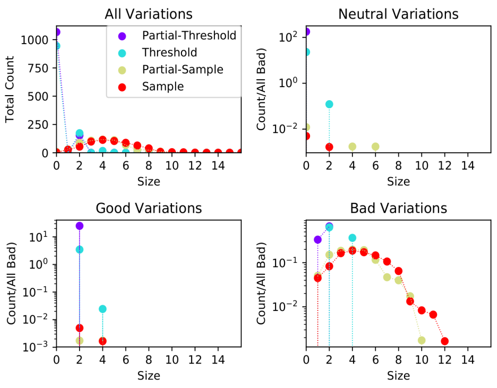
In the C=nPOV case, sampling methods perform as efficiently as the threshold method. This can be explained by the activation response presented in Figure A.3. For each problem complexity, a sample of 10 solutions are forward propagated through the autoencoder model and the activation values measured. Each response is plotted to present a visualisation of the average response. The model used for the C=nPOV function is unique in that the output response is saturated. Therefore, sampling the output will have similar behaviour to the threshold. This is unrealistic in expectation of the model in practice; however, it is an interesting observation. The sampling method can be interpreted as a MIG method for DO. Therefore the synergy of using MIG, MIV (and potentially a MIC method) could improve the performance of DO further.

A.2 Experiment Parameters

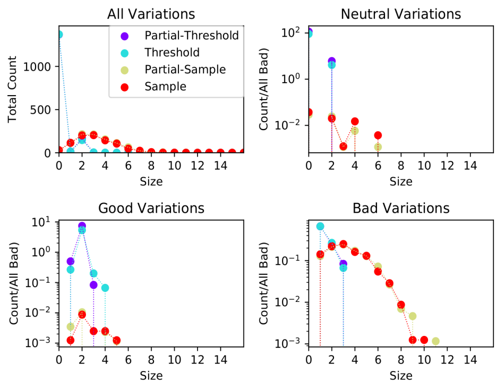
P3, hBOA and LTGA used the default parameters for constructing and exploiting the model (Goldman and Punch, 2015). DO used the following parameters. For problem sizes not listed, a linear interpolation between the parameters was used. default parameters are listed in Table A.1.



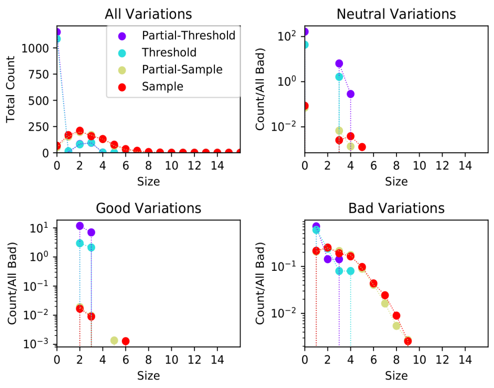
(a) non-Overlapping Variation, C=nOV



(b) Overlapping Variation, C=OV



(c) non-Distinguishable Overlapping Variation, C=nDOV

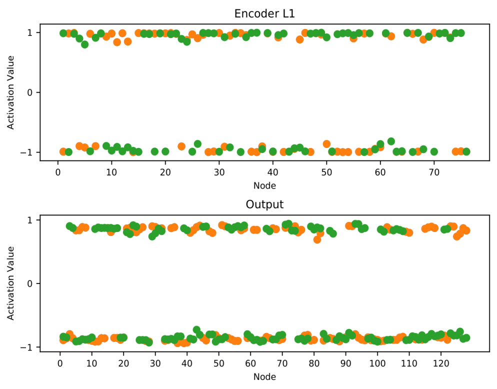


(d) non-Pairwise Overlapping Variation, C=nPOV

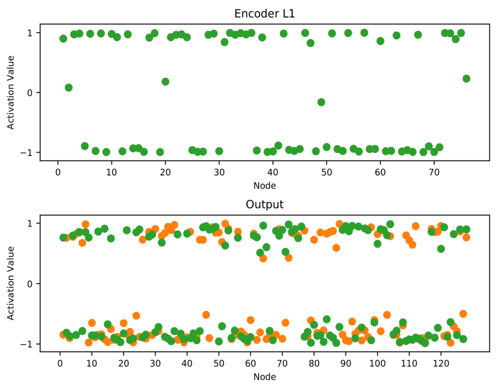
FIGURE A.2: Comparison of methods for calculating the variation made to a solution for problem size 128. The results show the size of a substitution made to a solution and its effect on the fitness of a solution. The method of Partial Threshold shows the greatest performance across all compression complexity.

Compression Mapping	Problem Size	Training Set Size	Learning Rate	Compression Ratio	L1 Regularisation	L2 Regularisation	Variation Method
nOV	16	32	0.0025	0.8	0.005	0.005	Assign
	32	32	0.0025	0.8	0.01	0.005	Assign
	64	32	0.0025	0.8	0.025	0.005	Assign
	128	32	0.002	0.8	0.05	0.005	Assign
	256	38	0.002	0.8	0.1	0.005	Assign
	512	38	0.002	0.8	0.2	0.02	Assign
OV	16	80	0.0025	0.6	0.0075	0.03	Assign
	32	80	0.0025	0.6	0.02	0.03	Assign
	64	80	0.002	0.6	0.02	0.03	Assign
	128	96	0.002	0.6	0.03	0.02	Assign
	256	112	0.002	0.6	0.08	0.01	Assign
	512	135	0.002	0.6	0.3	0.005	Assign
nDOV	16	90	0.0025	0.8	0.002	0.02	Encode
	32	96	0.002	0.8	0.0075	0.05	Encode
	64	224	0.002	0.8	0.008	0.2	Encode
	128	324	0.002	0.8	0.03	0.4	Encode
	256	444	0.002	0.8	0.03	0.8	Encode
	512	740	0.002	0.8	0.04	1.0	Encode
nPOV	16	100	0.002	0.9	0.0005	0.005	Assign
	32	128	0.002	0.9	0.004	0.0075	Assign
	64	400	0.002	0.9	0.0075	0.005	Assign
	128	840	0.002	0.9	0.025	0.005	Assign
	256	2000	0.002	0.9	0.03	0.0075	Assign

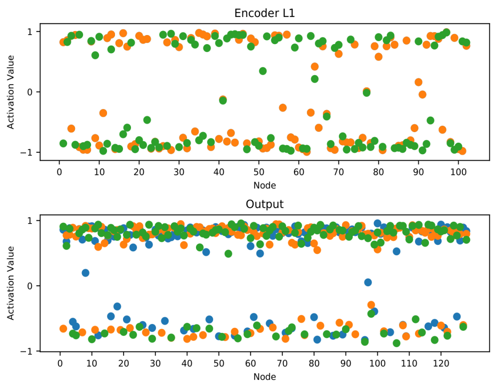
TABLE A.1: Parameters used for Deep Optimisation in experiements



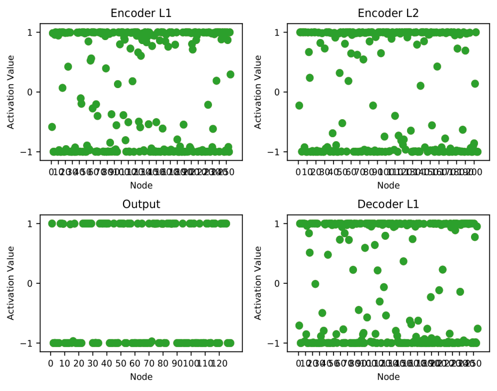
(a) non-Overlapping Variation, C=nOV



(b) Overlapping Variation, C=OV



(c) non-Distinguishable Overlapping Variation, C=nDOV



(d) non-Pairwise Overlapping Variation, C=nPOV

FIGURE A.3: Activation response for model training of different complexities. C=nPOV response is unique with output activation's completely saturated. Therefore, calculating dx by sampling performs well.

Bibliography

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

Uwe Aickelin, Edmund K Burke, and Jingpeng Li. An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society*, 58(12):1574–1585, 2007.

Lee Altenberg. Genome growth and the evolution of the genotype-phenotype map. In *Evolution and biocomputation*, pages 205–259. Springer, 1995.

Lee Altenberg et al. The evolution of evolvability in genetic programming. *Advances in genetic programming*, 3:47–74, 1994.

David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study.* Princeton university press, 2006.

Shumeet Baluja. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science, 1994.

Shumeet Baluja. Deep learning for explicitly modeling optimization landscapes. *arXiv preprint arXiv:1703.07394*, 2017.

Shumeet Baluja and Scott Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1997.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

Hans-Georg Beyer. An alternative explanation for the manner in which genetic algorithms operate. *BioSystems*, 41(1):1–15, 1997.

Peter AN Bosman and Dirk Thierens. *Linkage information processing in distribution estimation algorithms*, volume 1999. Utrecht University: Information and Computing Sciences, 1999.

Peter AN Bosman and Dirk Thierens. The roles of local search, model building and optimal mixing in evolutionary algorithms from a bbo perspective. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 663–670, 2011.

Justin Boyan and Andrew W Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1(Nov):77–112, 2000.

Pratik Prabhanjan Brahma, Dapeng Wu, and Yiyuan She. Why deep learning works: A manifold disentanglement perspective. *IEEE transactions on neural networks and learning systems*, 27(10):1997–2008, 2015.

Jamie Caldwell, Joshua Knowles, Christoph Thies, Filip Kubacki, and Richard Watson. Deep optimisation: Multi-scale evolution by inducing and searching in deep representations. In Pedro A. Castillo and Juan Luis Jiménez Laredo, editors, *Applications of Evolutionary Computation*, pages 506–521, Cham, 2021. Springer International Publishing. ISBN 978-3-030-72699-7.

JR Caldwell and Richard A Watson. How to get more from your model: the role of constructive selection in estimation of distribution algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 101–102, 2017.

JR Caldwell, Richard A Watson, C Thies, and Joshua D Knowles. Deep optimisation: Solving combinatorial optimisation problems using deep neural networks. *arXiv preprint arXiv:1811.00784*, 2018.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A Lozano. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18(2):286–300, 2013.

Jen-Hao Chang, Chung-Hsiang Hsueh, Hsuan Lee, Tian-Li Yu, and Tsung-Yu Ho. A test function with full controllability over overlapping and confliction between subproblems. *TEIL Report No. 2011003*, 2011.

Ping-Lin Chen, Chun-Jen Peng, Chang-Yi Lu, and Tian-Li Yu. Two-edge graphical linkage model for dsmga-ii. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 745–752, 2017.

Si-Cheng Chen and Tian-Li Yu. Difficulty of linkage learning in estimation of distribution algorithms. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 397–404, 2009.

Wei-Ming Chen, Chung-Yu Shao, Po-Chun Hsu, and Tian-Li Yu. A test problem with adjustable degrees of overlap and conflict among subproblems. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, page 257–264, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450311779.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.

Alexander W Churchill, Siddharth Sigtia, and Chrisantha Fernando. A denoising autoencoder that guides stochastic search. *arXiv preprint arXiv:1404.1614*, 2014.

David Jonathan Coffin and Robert Elliott Smith. The limitations of distribution sampling for linkage learning. In *2007 IEEE Congress on Evolutionary Computation*, pages 364–369. IEEE, 2007.

Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

Chris Cox. *Inferring and exploiting compact models of evolutionary problem structure.* PhD thesis, University of Southampton, 2015.

Chris R Cox and Richard A Watson. Inferring and exploiting problem structure with schema grammar. In *International Conference on Parallel Problem Solving from Nature*, pages 404–413. Springer, 2014a.

Chris R Cox and Richard A Watson. Solving building block problems using generative grammar. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 341–348, 2014b.

Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)*, 45(3):1–33, 2013.

Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

Dániel Czégel, István Zachar, and Eörs Szathmáry. Multilevel selection as bayesian inference, major transitions in individuality as structure learning. *Royal Society open science*, 6(8):190202, 2019.

Jeremy S De Bonet, Charles Lee Isbell Jr, and Paul A Viola. Mimic: Finding optima by estimating probability densities. In *Advances in neural information processing systems*, pages 424–430, 1997.

Kenneth A De Jong. *Evolutionary Computation: A Unified Approach.* MIT Press, 2006.

Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, USA, 1975. AAI7609381.

Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4):311–338, 2000.

Kalyanmoy Deb and David E. Goldberg. Analyzing deception in trap functions. In *Foundations of Genetic Algorithms*, volume 2 of *Foundations of Genetic Algorithms*, pages 93 – 108. Elsevier, 1993.

Kalyanmoy Deb and David E Goldberg. Sufficient conditions for deceptive and easy binary functions. *Annals of mathematics and Artificial Intelligence*, 10(4):385–408, 1994.

Agoston E Eiben and Cornelis A Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.

Agoston E Eiben and James E Smith. *Introduction to evolutionary computing.* Springer, 2015.

Andy Gardner. The genetical theory of multilevel selection. *Journal of evolutionary biology*, 28(2):305–319, 2015.

Partha Ghosh, Mehdi SM Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From variational to deterministic autoencoders. *arXiv preprint arXiv:1903.12436*, 2019.

Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

David E Goldberg. Simple genetic algorithms and the minimal. *Deceptive Problem, Genetic Algorithms and Simulated Annealing*, pages 74–88, 1987.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989a. ISBN 0201157675.

David E Goldberg. *Genetic algorithms.* Pearson Education India, 2006.

David E Goldberg, Kalyanmoy Deb, and Dirk Thierens. Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, 32(1):10–16, 1993.

David E Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, 3(5):493–530, 1989.

DE Goldberg. Genetic algorithms in search, optimization, and machine learning, addison-wesley, reading, ma, 1989. *NN Schraudolph and J*, 3(1), 1989b.

Brian W. Goldman and William F. Punch. Parameter-less population pyramid. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 785–792, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326629.

Brian W Goldman and William F Punch. Fast and efficient black box optimization using the parameter-less population pyramid. *Evolutionary computation*, 23(3):451–479, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

Georges Harik et al. Linkage learning via probabilistic modeling in the ecga. *IlliGAL report*, 99010, 1999a.

Georges R Harik et al. Finding multimodal solutions using restricted tournament selection. In *ICGA*, pages 24–31. Citeseer, 1995.

Georges R Harik, Fernando G Lobo, and David E Goldberg. The compact genetic algorithm. *IEEE transactions on evolutionary computation*, 3(4):287–297, 1999b.

Georges Raif Harik. *Learning Gene Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms*. PhD thesis, University of Michigan, USA, 1997. UMI Order No. GAX97-32090.

Mark Hauschild and Martin Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and evolutionary computation*, 1(3):111–128, 2011.

DO Hebb. *The organization of behavior: a neuropsychological theory*. New York:[sn], 1961.

Robert B Heckendorn, Soraya Rana, and Darrell Whitley. Test function generators as embedded landscapes. *Foundations of Genetic Algorithms*, 5:183–198, 1999.

Abdel-Rahman Hedar, 2020.

Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

Francisco Herrera, Manuel Lozano, and Ana M Sánchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18(3):309–338, 2003.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012a.

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

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012b.

Christian Höhn and C. Reeves. Are long path problems hard for genetic algorithms? In *PPSN*, 1996.

John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1992a.

John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992b.

John J Hopfield and David W Tank. “neural” computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

Jeffrey Horn, David E Goldberg, and Kalyanmoy Deb. Long path problems. In *International Conference on Parallel Problem Solving from Nature*, pages 149–158. Springer, 1994.

Shih-Huan Hsu and Tian-Li Yu. Optimization by pairwise linkage detection, incremental linkage set, and restricted/back mixing: Dsmga-ii. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 519–526, 2015.

David Iclanzan. Hierarchical allelic pairwise independent functions. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 633–640, 2011.

David Iclanzan and Dan Dumitrescu. Overcoming hierarchical difficulty by hill-climbing the building block structure. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1256–1263, 2007.

Viren Jain and Sebastian Seung. Natural image denoising with convolutional networks. *Advances in neural information processing systems*, 21:769–776, 2008.

Hillol Kargupta. The gene expression messy genetic algorithm. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 814–819. IEEE, 1996.

Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

Stuart A Kauffman et al. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, USA, 1993.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *arXiv preprint arXiv:1711.00848*, 2017.

Kevin N Laland, Tobias Uller, Marcus W Feldman, Kim Sterelny, Gerd B Müller, Armin Moczek, Eva Jablonka, and John Odling-Smee. The extended evolutionary synthesis: its structure, assumptions and predictions. *Proceedings of the Royal Society B: Biological Sciences*, 282(1813):20151019, 2015.

Claudio F Lima, Martin Pelikan, Kumara Sastry, Martin Butz, David E Goldberg, and Fernando G Lobo. Substructural neighborhoods for local search in the bayesian optimization algorithm. In *Parallel Problem Solving from Nature-PPSN IX*, pages 232–241. Springer, 2006.

Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

Hod Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. *Journal of Biological Physics and Chemistry*, 7(4):125, 2007.

Michele Lombardi, Michela Milano, and Andrea Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244:343–367, 2017.

Jean P Martins and Alexandre CB Delbem. Pairwise independence and its impact on estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 27: 80–96, 2016.

Jean P Martins, Carlos M Fonseca, and Alexandre CB Delbem. On the performance of linkage-tree genetic algorithms for the multidimensional knapsack problem. *Neurocomputing*, 146:17–29, 2014.

Jean P Martins, Constâncio Bringel Neto, Marcio K Crocomo, Karla Vittori, and Alexandre CB Delbem. A comparison of linkage-learning-based genetic algorithms in multidimensional knapsack problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 502–509. IEEE, 2013.

John Maynard Smith and Eörs Szathmáry. *The major transitions in evolution*. Oxford University Press, 1997.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *arXiv preprint arXiv:2003.03600*, 2020.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

Ryszard S Michalski. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine learning*, 38(1):9–40, 2000.

Rob Mills. *How micro-evolution can guide macro-evolution: Multi-scale search via evolved modular variation*. PhD thesis, University of Southampton, 2010.

Rob Mills, Thomas Jansen, and Richard A Watson. Transforming evolutionary search into higher-level evolutionary search by capturing problem structure. *IEEE Transactions on Evolutionary Computation*, 18(5):628–642, 2014.

Rob Mills and Richard A Watson. Multi-scale search, modular variation, and adaptive neighbourhoods. *Author's Original*, 2011.

M Mitchell, J H Holland, and S Forrest. The royal road for genetic algorithms: Fitness landscapes and ga performance. *Ann Arbor*, 1001, 1 1991.

Heinz Mühlenbein and Thilo Mahnig. Fda-a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary computation*, 7(4): 353–376, 1999.

Heinz Mühlenbein and Gerhard Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *International conference on parallel problem solving from nature*, pages 178–187. Springer, 1996.

Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In *Foundations of genetic algorithms*, volume 3, pages 73–88. Elsevier, 1995.

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity.* Courier Corporation, 1998.

G Pavai and TV Geetha. A survey on crossover operators. *ACM Computing Surveys (CSUR)*, 49(4):1–43, 2016.

Martin Pelikan. Analysis of estimation of distribution algorithms and genetic algorithms on nk landscapes. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1033–1040, 2008.

Martin Pelikan. Nk landscapes, problem difficulty, and hybrid evolutionary algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 665–672, 2010.

Martin Pelikan and David E Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 511–518. Morgan Kaufmann Publishers Inc., 2001.

Martin Pelikan and David E. Goldberg. Hierarchical boa solves ising spin glasses and maxsat. In *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartII*, GECCO'03, page 1271–1282, Berlin, Heidelberg, 2003a. Springer-Verlag. ISBN 3540406034.

Martin Pelikan and David E Goldberg. A hierarchy machine: Learning to optimize from nature and humans. *Complexity*, 8(5):36–45, 2003b.

Martin Pelikan and David E Goldberg. Hierarchical bayesian optimization algorithm. In *Scalable optimization via probabilistic modeling*, pages 63–90. Springer, 2006.

Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. Boa: The bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99*, volume 1, pages 525–532, 1999.

Martin Pelikan, David E Goldberg, and Shigeyoshi Tsutsui. Hierarchical bayesian optimization algorithm: toward a new generation of evolutionary algorithms. In *SICE 2003 Annual Conference (IEEE Cat. No. 03TH8734)*, volume 3, pages 2738–2743. IEEE, 2003.

Martin Pelikan, Mark W Hauschild, and Dirk Thierens. Pairwise and problem-specific distance metrics in the linkage tree genetic algorithm. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1005–1012, 2011.

Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In *Advances in Soft Computing*, pages 521–535. Springer, 1999.

Massimo Pigliucci and Gerd B Müller. *Evolution-the extended synthesis*. The MIT Press, 2010.

Jean-Yves Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63(3):337–370, 1996.

Malte Probst. Denoising autoencoders for fast combinatorial black box optimization, 2015.

Malte Probst and Franz Rothlauf. Deep boltzmann machines in estimation of distribution algorithms for combinatorial optimization. *arXiv preprint arXiv:1509.06535*, 2015.

Nestor V Queipo, Raphael T Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and P Kevin Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

Elizabeth Radetic and Martin Pelikan. Spurious dependencies and eda scalability. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 303–310, 2010.

Reinhelt. {TSPLIB}: a library of sample instances for the tsp (and related problems) from various sources and of various types, 2014.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml*, 2011.

Franz Rothlauf. Representations for genetic and evolutionary algorithms. In *Representations for Genetic and Evolutionary Algorithms*, pages 9–32. Springer, 2006.

Krzysztof L Sadowski, Peter AN Bosman, and Dirk Thierens. On the usefulness of linkage processing for solving max-sat. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 853–860, 2013.

Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

Ralf Salomon. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278, 1996.

Roberto Santana. Gray-box optimization and factorized distribution algorithms: where two worlds collide, 2017.

Roberto Santana, Pedro Larrañaga, and Jose A Lozano. Protein folding in simplified models with estimation of distribution algorithms. *IEEE transactions on Evolutionary Computation*, 12(4):418–438, 2008.

Kumara Sastry and David E Goldberg. Designing competent mutation operators via probabilistic model building of neighborhoods. In *Genetic and Evolutionary Computation Conference*, pages 114–125. Springer, 2004.

David Sherrington and Scott Kirkpatrick. Solvable model of a spin-glass. *Physical review letters*, 35(26):1792, 1975.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Herbert A Simon. *The Sciences of the Artificial.* The MIT Press, 1969.

Joeri Sleegers and Daan van den Berg. Looking for the hardest hamiltonian cycle problem instances. In *IJCCI*, pages 40–48, 2020a.

Joeri Sleegers and Daan van den Berg. Plant propagation & hard hamiltonian graphs. *Evo* 2020*, page 10, 2020b.

Emilie C Snell-Rood. Selective processes in development: implications for the costs and benefits of phenotypic plasticity. *Integrative and comparative biology*, page ics067, 2012.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

D Thierens. Analysis and design of genetic algorithms. *Doctoral dissertation, Katholieke Universiteit Leuven, Leuven, Belgium*, 1995.

Dirk Thierens. Scalability problems of simple genetic algorithms. *Evolutionary computation*, 7(4):331–352, 1999.

Dirk Thierens. The linkage tree genetic algorithm. In *International Conference on Parallel Problem Solving from Nature*, pages 264–273. Springer, 2010.

Dirk Thierens and Peter A.N. Bosman. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, page 617–624, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450305570.

Dirk Thierens and Peter AN Bosman. Hierarchical problem solving with the linkage tree genetic algorithm. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 877–884, 2013.

Dirk Thierens and David E Goldberg. Mixing in genetic algorithms. In *ICGA*, pages 38–47, 1993.

Miwako Tsuji, Masaharu Munetomo, and Kiyoshi Akama. A crossover for complex building blocks overlapping. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, page 1337–1344, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595931864.

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

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

Riccardo Volpato and Guangyan Song. Active learning to optimise time-expensive algorithm selection, 2019.

Ky Khac Vu, Claudia D'Ambrosio, Youssef Hamadi, and Leo Liberti. Surrogate-based methods for black-box optimization. *International Transactions in Operational Research*, 24(3):393–424, 2017.

Günter P Wagner and Lee Altenberg. Perspective: complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, 1996.

Shih-Ming Wang, Jie-Wei Wu, Wei-Ming Chen, and Tian-Li Yu. Design of test problems for discrete estimation of distribution algorithms. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 407–414, 2013.

Richard A Watson. *Compositional evolution: interdisciplinary investigations in evolvability, modularity, and symbiosis.* PhD thesis, Brandeis University, 2002.

Richard A. Watson. On the unit of selection in sexual populations. In Mathieu S. Capcarrère, Alex A. Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors, *Advances in Artificial Life*, pages 895–905, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31816-3.

Richard A Watson. *Compositional evolution: the impact of sex, symbiosis and modularity on the gradualist framework of evolution.* Mit Press, 2006.

Richard A Watson, Christopher L Buckley, and Rob Mills. Optimization in "self-modeling" complex adaptive systems. *Complexity*, 16(5):17–26, 2011a.

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

Richard A Watson, Rob Mills, and Christopher L Buckley. Transformations in the scale of behavior and the global optimization of constraints in adaptive networks. *Adaptive Behavior*, 19(4):227–249, 2011b.

Richard A Watson, Rob Mills, CL Buckley, Kostas Kouvaris, Adam Jackson, Simon T Powers, Chris Cox, Simon Tudge, Adam Davies, Loizos Kounios, et al. Evolutionary connectionism: algorithmic principles underlying the evolution of biological organisation in evo-devo, evo-eco and evolutionary transitions. *Evolutionary biology*, 43(4): 553–581, 2016.

Richard A Watson and Eörs Szathmáry. How can evolution learn? *Trends in ecology & evolution*, 31(2):147–157, 2016.

Richard A Watson, Günter P Wagner, Mihaela Pavlicev, Daniel M Weinreich, and Rob Mills. The evolution of phenotypic correlations and "developmental memory". *Evolution*, 68(4):1124–1138, 2014.

Thomas Weise, Raymond Chiong, and Ke Tang. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology*, 27(5):907–936, 2012.

Stuart A West, Roberta M Fisher, Andy Gardner, and E Toby Kiers. Major evolutionary transitions in individuality. *Proceedings of the National Academy of Sciences*, 112(33): 10112–10119, 2015.

L Darrell Whitley. Fundamental principles of deception in genetic search. In *Foundations of genetic algorithms*, volume 1, pages 221–241. Elsevier, 1991.

David H. Wolpert. Ubiquity symposium: Evolutionary computation and the processes of life: What the no free lunch theorems really mean: How to improve search algorithms. *Ubiquity*, 2013(December), December 2013.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Tian-Li Yu, Kumara Sastry, and David E. Goldberg. Linkage learning, overlapping building blocks, and systematic strategy for scalable recombination. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, page 1217–1224, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930108.

Wei Zhang and Thomas G Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Reseach*, 1:1–38, 2000.

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Learning hierarchical features from generative models. *arXiv preprint arXiv:1702.08396*, 2017.