UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science

The Ability of Differential Evolution Principles and Multi-Parent Crossover Techniques to Solve Random-Linkage Modular Problems

by

Loizos Kounios

A dissertation submitted in partial fulfilment of the degree of
MSc Artificial Intelligence
by examination and dissertation

Supervisor: Dr Richard A. Watson
Examiner: Dr James Dyke

September 6, 2013

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science

THE ABILITY OF DIFFERENTIAL EVOLUTION PRINCIPLES AND MULTI-PARENT CROSSOVER TECHNIQUES TO SOLVE RANDOM-LINKAGE MODULAR PROBLEMS

by Loizos Kounios

The crossover operator in a genetic algorithm has been used to solve certain building-block problems that mutation-only methods cannot. However, it is well-understood that this advantage of crossover is lost when optimising random-linkage problems where the bits of each module are arbitrarily placed across the genome. Conventional crossover is therefore not applicable in engineering domains where the epistatic linkage map of a problem is not known *a priori*. This motivates a large body of work on algorithms that aim to identify and exploit the linkage structure of a problem.

Here, we introduce multi-parent crossover techniques that use similarities and differences between evolved individuals to guide genetic variation. The first operator takes the difference between two members of the population which provides information about subsets of bits that are epistatically-dependent. The second operator identifies subsets of epistatically-dependent alleles by using similarities between individuals to vote for what value each locus should take. We use the operators in the context of differential evolution, converted for use on binary problems, and show that the algorithms are able to reliably solve random-linkage building-block problems without *a priori* knowledge about the structure of the problem. We show that the crossover operators are able identify and exploit modular structure, without requiring explicit learning mechanisms. We show with numerical simulation that uniform crossover techniques and restart hill-climbers scale exponentially with the size of the problem on two classes of random-linkage modular problems. We show that there is a significant polynomial versus super-polynomial time complexity difference between the two algorithms we introduce, and identify what causes the difference.

Contents

Acknowledgements

I would like to begin by thanking my supervisor, Dr Richard A. Watson, for his invaluable input, help and support throughout all this. Richard spent far more energy on helping me than he had to and for that I am truly grateful. I would also like to thank my co-supervisor, Dr James Dyke, for providing me with thought-provoking questions and remarks during the demonstration. Special thanks go to Dr Rob Mills whose PhD thesis and work helped introduce me to the field.

Lastly, I would like to thank everyone who helped proof-read this thesis, no matter how boring I am sure it was for them to read.

Chapter 1

Introduction

A large body of work in the genetic algorithm (GA) literature has been focused on investigating the ability or inability of crossover to exploit the principle of problem decomposition in problems with modular or building-block structure [1–5]. A polynomial versus exponential time complexity distinction has only been shown using tight-linkage crossover (e.g., one- or two-point crossover) [2, 5]. However, the tight-linkage assumption (i.e., that epistatically-dependent loci within a module are close together on the genome, enabling modules to be selected as a heritable units during crossover events, as per the building-block hypothesis [6, 7]) compromises the applicability of this work.

In design-engineering and optimisation, the underlying structure of a problem is not generally available, meaning that there is no information about the correct ordering of the variables which would facilitate problem decomposition and effective crossover. This led to work that abandons building-block ideas altogether and utilises uniform crossover which does not assume tight-linkage and does not attempt to select on particular epistatically-dependent subsets of loci. Another body of work was on *linkage-learning* and estimation of distribution algorithms. Work on linkage-learning attempts to overcome this limitation by automatically identifying and exploiting problem structure, regardless of where across the genome epistatically-dependent loci are situated [8–10]. This approach has been successful on practical problems in many domains. The idea that the GA is capable of implicitly exploiting problem structure is discarded, and, instead, this class of algorithms utilises machine learning techniques to explicitly learn and exploit the structure of the problem.

In this thesis, we introduce two operators that use similarities and differences between evolved individuals to guide genetic variation. We show that the principles of the variation operator of differential evolution (DE) [11–13] can be converted and applied to binary problem spaces. The DE variation operator uses differences between individuals which are added to another individual to direct the search. The second multi-parent crossover operator directs the search by doing the inverse: utilise similarities between individuals. Because the operators use more than two parents, we refer to this class of operators as *multi-parent crossover*. We show that this type of variation is capable of exploiting information that is implicit within the population to solve random-linkage building-block problems, without using any explicit machine learning techniques.

Although DE is not naturally applicable to discrete spaces, its application has not been limited to that of continuous spaces, having been successfully applied to image processing, combinatorial optimisation, permutation and constraint satisfaction problems [14–18]. However, because the DE variation operator controls both the magnitude and the direction of genetic changes, it seems unsuitable for use in binary optimisation problems, including those used to investigate the principles of building-block recombination in the GA [1–5, 19]. Nonetheless, here we show that, in binary problem spaces, constructing a difference vector by taking the difference of two individuals and super-imposing it onto another individual is sufficient for identifying epistatically-dependent subsets of loci and crossing over coherent units. In continuous spaces, the difference vector determines how much each variable changes and in which direction; that is, the emphasis is on how much and in which direction each variable changes rather than on which variables to change. In binary spaces, each locus of the difference vector takes either -1, 0, or 1. This determines which loci change and which do not, and whether they change from 0 to 1 or from 1 to 0. Thus, the magnitude of any change is fixed, but only the loci with a non-zero difference are affected. Simply put, the difference of two individuals identifies a schema that is super-imposed on another individual. Conversely, by utilising the similarities between individuals, the value at a locus changes only when the other two parents have a different value. The effect is that subsets of epistatically-dependent loci are identified by the similarities between individuals. This type of crossover is called occurrence-based scanning (OB-Scan) [20], but it has not yet been used to solve random-linkage building-block problems.

We use both variation operators in the context of DE. Because we retain the concept of differential mutation, we call the algorithm utilising the differences between individuals *discrete differential evolution* (dDE). We call the algorithm utilising the similarities *multi-parent evolution* (MPE). We test the algorithms on one separable and one nearly-decomposable problem, both of which are pathologically difficult for techniques producing uninformed variation. Although the tight-linkage versions of these problems are solvable by a GA with one-point or two-point crossover in polynomial time, the random-linkage or "shuffled" versions of the problems are just as difficult for the GA (with any type of crossover) as they are for the uniformed methods. Here we show that the algorithms we introduce solve the problems reliably. This is achieved without any explicit machine learning methods or data structures to represent linkage information; rather it is possible because the operators are able identify and exploit the modular structure of the solution space that is revealed implicitly by the distribution of individuals in the population.

In the remainder of this thesis we define the problems we use for testing and the two algorithms, and present the results of numerical simulations. We provide empirical results for applications of uniform crossover and random-restart hill-climbers. In Chapter 2, we provide the reader with background reading on DE, modularity, what it means for a search process to search in multiple scales and discuss related work. In Chapter 3, we introduce the test problems we use for testing and the two algorithms. In Chapter 4, we show the behaviour of the algorithms and operators, provide numerical analysis for the time complexity of various algorithms, and investigate the differences between dDE and MPE. In Chapter 5, we discuss multiple aspects of our work, including why the problems we use for testing are difficult to solve, and what it means for our algorithms to "learn". Lastly, Chapter 6 concludes the thesis.

Chapter 2

Background

Here, we provide the required background for the reader and discuss related work. In Section 2.1, we provide an overview of the differential evolution algorithm. In Section 2.2, we briefly explain what modularity is, provide an overview of the basic terminology and explain what it means for a search process to search in multiple scales. Lastly, in Section 2.3, we discuss related work by briefly analysing a few of the algorithms that exploit modular structure.

2.1 Differential Evolution, DE

Differential evolution (DE) was introduced in 1995 by R. Storn and K. V. Price (University of California, Berkeley) as a stochastic solution optimiser for real-valued problems [11–13].

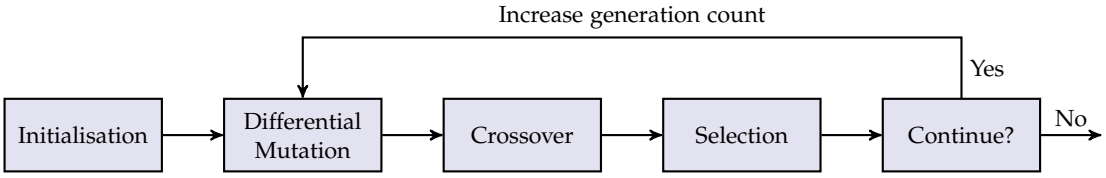

Figure 2.1: The differential evolution optimisation process. The cycle of operations includes differential mutation, crossover and selection.

The DE optimisation process relies on evolutionary mechanics, such as genetic mutation; crossover; and survival of the fittest, to iteratively evolve a population of candidate solutions. After the population of candidate solutions is initialised to uniform, random real-valued numbers, the optimisation process comprises three stages: (i) differential mutation; (ii) crossover; and (iii) selection. The fitness function parameters are represented as vectors. DE makes use of three different populations of vectors: (1) a target (or parent) vector matrix, $\boldsymbol{X}$, where $\boldsymbol{x}_i$, is the ith target vector; (2) a mutant (or donor) vector matrix, $\boldsymbol{V}$, where $\boldsymbol{v}_i$, is the ith mutant vector produced by the differential mutation operator; and (3) a trial (or child) vector matrix, $\boldsymbol{U}$, where $\boldsymbol{u}_i$, is the ith trial vector (or child) and is produced by crossing over elements between the ith target and

mutant vectors. A brief description of the DE optimisation process (Figures 2.1 and 2.2) follows.

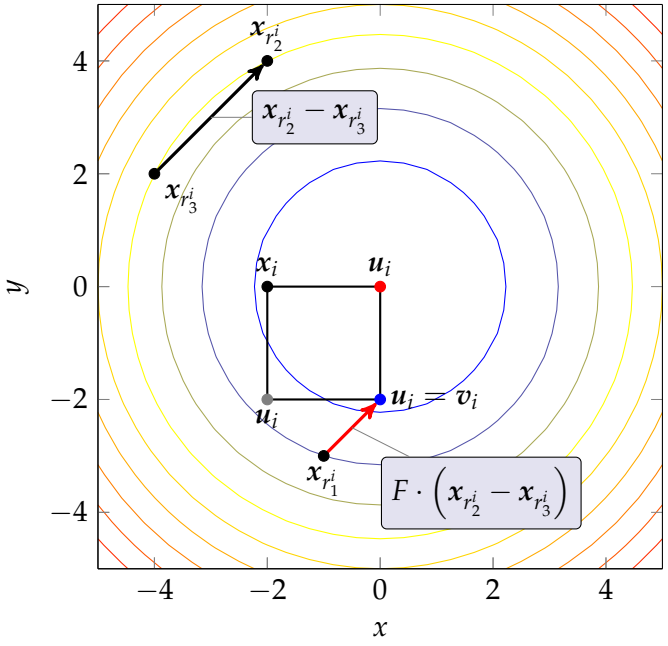

Figure 2.2: The behaviour of differential evolution in the continuous space. In the background, the contour of a 2-dimensional sphere function. The difference of two target vectors, $\boldsymbol{x}_{r_2^i}$ and $\boldsymbol{x}_{r_3^i}$, is scaled by a factor, F, and added to the base target vector, $\boldsymbol{x}_{r_1^i}$, to generate the mutant vector, $\boldsymbol{v}_i$. From there, crossing over the ith target and mutant vectors can generate three distinct trial vectors (children), $\boldsymbol{u}_i$. Selection would discard the gray trial vector because it is less fit than the ith target vector; retain the blue trial vector because it is equally fit; or retain the red trial vector because it is fitter.

Initialisation

The population of NP target vectors, $\boldsymbol{X}$, (known as *individuals*, *chromosomes* or *genes*) is uniformly initialised to real-valued numbers. Each ith vector, $\boldsymbol{x}_i$ is a N-dimensional candidate solution to the optimisation problem.

Differential Mutation

Differential mutation is the process of taking the difference of one or more sets of individuals and adding it to another individual. A number of differential mutation variants have been developed. These variants decide the combination of vectors which are used to perturb an individual. For example, the classical DE algorithm utilises the *rand/1* mutation scheme which forms the ith mutant vector, $\boldsymbol{v}_i$, using three randomly chosen, mutually exclusive target vectors: $\boldsymbol{x}_{r_1^i}, \boldsymbol{x}_{r_2^i}, \boldsymbol{x}_{r_3^i}$, where $r = [1, NP]$ and $r_1^i \neq r_2^i \neq$

$r_3^i \neq i$. The mutant vector is formed as shown below:

$$\boldsymbol{v}_i = \boldsymbol{x}_{r_1^i} + F \cdot \left(\boldsymbol{x}_{r_2^i} - \boldsymbol{x}_{r_3^i} \right),$$

where F is the scaling factor which is typically in the range of $F = [0.4, 1]$.

Crossover

Crossover is used as further means of introducing diversity. The ith trial vector, $\boldsymbol{u}_i$, is formed by performing crossover on the ith target and mutant vectors. In *binomial* (bin) crossover elements from the two vectors are crossed over arbitrarily, whereas in *exponential* (exp) crossover elements are crossed over as contiguous blocks (i.e., linkage-preserving two-point crossover). The proportion of elements taken from each vector is determined by the crossover rate, Cr, control parameter, with a crossover rate of 1, $Cr = 1$, meaning that the trial vector is purely mutant (i.e., no elements are taken from the target vector).

Selection

DE employs a "greedy" replacement scheme in the selection stage. Straightforwardly, the ith trial vector, $\boldsymbol{u}_i$, replaces the ith target vector, $\boldsymbol{x}_i$, in the next generation only if it is equally fit or fitter. By removing fitness-proportionate selection altogether and coupling each ith individual of the target and trial vectors in a one-to-one elitist tournament, the population explores multiple areas of the fitness landscape simultaneously.

Ending Criteria

The DE optimisation process ends when one or more of the following criteria is true: (i) a pre-specified maximum number of generations, G_{max}, has been met; (ii) a lack of appreciable change in fitness for a number of generations has been observed; and (iii) a pre-specified objective, such as a fitness value, has been successfully met.

Nomenclature

DE variants follow a simple convention to describe their properties as follows: *DE/x/y/z*, where "DE" stands for "Differential Evolution"; x and y describe the target vectors which are used to produce the mutant vector in the differential mutation operator; and z denotes the type of crossover operator used. For example, *DE/rand/1/bin* refers to a DE algorithm which uses a randomly chosen target vector as the base and one difference vector in the differential mutation operator, and crossover is binomial. A plethora of DE variants and operators exist in the literature and a number of works has been focused on identifying which are the best-performing [21–27].

2.2 Modularity

Billions of years ago, life on Earth was so incredibly simple it could be described by only a handful of cell types. Today, however, life is so complex and diverse that it is difficult to grasp what the driving force behind it is. Despite the number of different species, the complexity and diversity observed in nature is a result of the same underlying process: biological evolution. An important outcome of evolution that is ubiquitous in biological systems is modularity. A system is modular when it consists of smaller, relatively autonomous, interconnected sub-systems, such as the human brain consisting of different functional areas or a vehicle consisting of the engine, the wheels etc. When these sub-systems communicate, emergent, collective behaviour is observed. Although it is clear that modularity in biological systems was the result of evolution, with evidence confirming its existence [28, 29], how and why it arose remains an open question [30]. In engineered systems, however, modularity does not emerge naturally: human intervention is required to serve as a blueprint.

Evolutionary Computation (EC) has not had great success in its attempt to produce the complexity and diversity that exists in nature, with the lack of modularity cited as one of the missing components. By taking a divide-and-conquer approach and breaking down the problem into multiple sub-problems, a search process can solve each sub-problem and assemble the sub-solutions to generate the overall solution. This process is called *problem decomposition*. Modular problems are, by definition, decomposable and are suitable for problem decomposition. But to appropriately decompose a problem, human expertise is required and it is generally unavailable. Hence, the challenge lies in introducing mechanisms capable of automatically discovering and exploiting the underlying modular structure of a problem. An overview of the modularity concepts and terminology follows.

2.2.1 Building-Blocks, Schemata, Modules

The building-block hypothesis (BBH) was introduced [6, 7] to explain the success of the original genetic algorithm (GA) [31]. Straightforwardly, the BBH suggests that a GA can discover building-blocks that can be assembled to produce higher-order solutions. A building-block is defined as a low-order, above-average fitness schema of short defining length. Under the assumption of tight physical linkage, linkage-preserving crossover (e.g., one- or two-point crossover) is less disruptive for low-order, short schemata, while a schema of above-average fitness is more likely to be selected. This enables the crossover operator to identify high-fitness sub-solutions, or building-blocks, that can be assembled to produce overall solutions of higher fitness.

Although the BBH illustrated how crossover can be used to implicitly scale search and assist the optimisation process by assembling building-blocks, it was initially difficult to identify problems that showed a principled distinction between mutation-only processes and GAs. The royal road function is an example of a synthetic problem that failed to do so [32]. In later years, a number of works introduced synthetic problems that successfully demonstrated the benefits of crossover and identified under which conditions its use is beneficial [1–5].

Building-blocks and modules are related concepts. Both are used to describe partial solutions to a problem, but they have two main differences: (i) building-blocks assume *a priori* tight physical linkage (i.e., short defining length), but modules do not; and (ii) building-blocks must be of low-order, but not modules.

Epistasis & Dependencies

Systems in general have interactions between their variables and/or their sub-functions. The terms *epistasis* and *dependency* are used to describe these interactions. *Epistasis*, a term common to genetics, is used to describe the interaction between genes, and the term *dependency* typically describes the interaction between variables instead of genes.

Modular Interdependency

Depending on its order, l, and alphabet cardinality, a, a module can take a number of configurations, C. For example, a module of order l that is encoded as a binary string ($a = 2$) can take a total of $C = 2^l$ different configurations. Watson and Pollack [33] define the number of configurations a module can take that are optimal under some context as $C' = [1, C]$. A system does not facilitate decomposition when $C' = C$ and is therefore of no interest to us. Systems are modular strictly when $C' < C$. Systems with $C' = 1$ are *separable* and enable the solution of each module in complete isolation, but are not of much interest because completely autonomous components are not generally part of the same system. Hence, the literature is focused on decomposable and non-separable systems, or *nearly-decomposable*, $1 < C' < C$. These systems exhibit *modular interdependency*.

Modular Decomposition & Regularity

In a system, a component whose within dependencies are stronger than its without is a module. Typically, modular systems exhibit two types of modularity: (i) *functional modularity*; and (ii) *regularity*. The former facilitates functional decomposition and the latter the repetition of functional components—hence, functional modularity is a prerequisite for regularity. In the context of optimisation, exploiting functional modularity is the equivalent of reducing the dimensionality of a problem by transforming the search space to one of higher scale, whereas regularity involves the reuse of one solution in multiple areas of the problem, such as in the evolution of gaits [34] or the evolution of regular 3D objects [35]. In this thesis, we focus on functional modularity. That is, even though the solutions to each module for the problems we use might be the same, it need not be the case as it is only done to enhance visualisation and analytical tractability.

2.2.2 Exploiting Multiple Scales

For any search process to be able to exploit multi-scale problems, an adaptive mechanism is required that enables a transformation in the fitness landscape neighbourhood from searching in the space of order-1 schemata to searching in the space of functional components of order-k, $k > 1$. In order for the process to be fruitful, the order-k schemata must form cohesive units which reflect the structure of the problem. Hence, the efficiency of such a mechanism will be at its maximum when the optimisation problem exhibits modular structure [5, 36] that will allow it to substitute one *module solution*, or sub-solution, with another. Assuming that all module solutions are locally optimal, and there are relatively weak inter-module dependencies, multi-scale search can enable the jumping between module solutions, with the benefit of avoiding low-fitness intermediate configurations.

We define *macro-variation* as the mechanism that enables the substitution of module solutions; *module* as a subset of variables with strong dependencies; and *module solution* as a specific configuration of a module whose fitness cannot be improved by any single-variable change. Contrary to the definition of building-blocks, modules are not necessarily (i) of low order, nor is their order bounded by fixed values; or (ii) of short defining length, i.e., co-dependent variables can be arbitrarily placed across the genome. However, the idea that a module solution is of particularly high fitness is retained.

With the use of macro-variation, search can escape solutions that would be locally optimal under conventional variation. Consider a problem with two modules with two module solutions, or local optima, each: A and a for m_1, and B and b for m_2. Additionally, we assume that the Hamming distance between the two module solutions is large, meaning that a large number of simultaneous variable changes would be required to jump between A and a or B and b. Given that the within module dependencies are stronger than the without, any combination of module solutions (ab, aB, Ab and AB) would be locally optimal. An example fitness landscape is illustrated in Figure 2.3a.

Since all combinations of module solutions are locally optimal and sit several variables apart, single-variable variation would result in a decrease in fitness. Assuming that the module solutions have been successfully identified, a macro-variation operator would enable the direct substitution of, for example, a for A, thereby avoiding the intermediate fitness valley which includes sites of lower fitness. This results in a new neighbourhood connecting all locally optimal configurations, as shown in Figure 2.3b. Even though in this example we are only allowing macro-variation to change one module per time-step, convergence to the global optimum, AB, is guaranteed because the resulting optimisation problem is uni-modal.

A number of conclusions can be reached from the above. In the modified genotype neighbourhood resulting from an appropriate macro-variation operator, the substitution of one module solution for another corresponds to the transition between locally optimal configurations. Macro-variation produces multiple, neighbouring locally optimal configurations which can be compared by means of selection. When the resulting fitness landscape is uni-modal, the dynamics of the search process change from hill-climbing in the space of single-variable changes to hill-climbing in the space of module

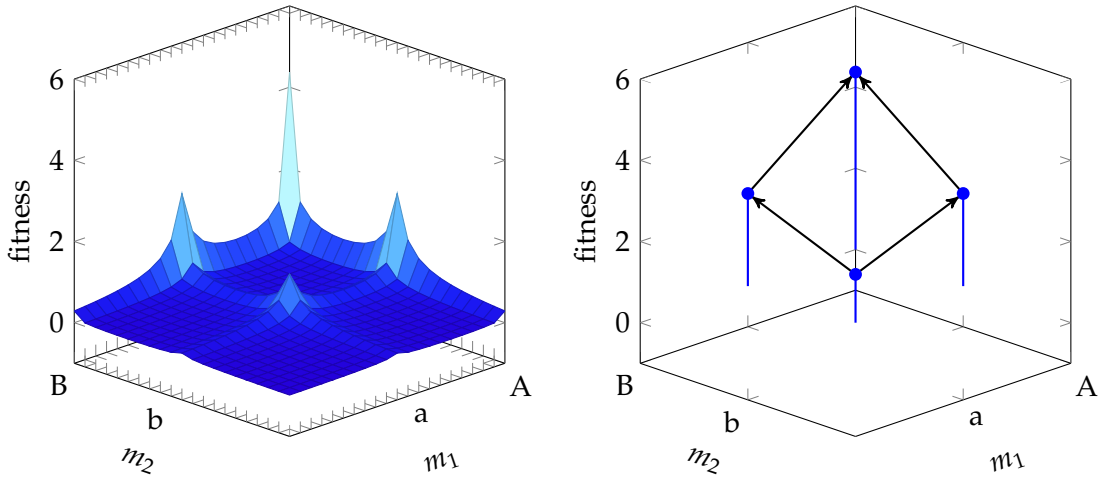

Figure 2.3: (a) An example of a multi-modal fitness landscape when the variation operator acts on units of order-1, i.e., neighbours differ by a bit. Escaping from any locally optimal configuration (e.g., *ab* to *aB* or *ab* to *Ab*) cannot be achieved using conventional means because it requires a decrease in fitness. (b) The same fitness landscape when macro-variation is utilised. The genotype neighbourhood is modified and search takes place in the space of module solutions. When searching in the space of module solution combinations, no intermediate solutions of lower fitness need to be visited.

solutions. In this example, a hill-climber in the new neighbourhood can reliably discover the global optimum, *AB*, with the worst genotype only two "macro-steps" away. Contrary to that, a hill-climber in the original neighbourhood is only capable of finding the global optimum when the genotype starts in the right basins of attraction. Obviously, in a two-module example such as the one described here, a random-restart hill-climber will only require a few restarts. However, the number of local optima increases exponentially with the number of multi-modal modules, and consequently starting in all the right basins of attractions would require an exponentially growing amount of restarts [5].

A number of synthetic problems have successfully illustrated the idea of modularity and building-blocks (e.g., [8, 19, 37]). These problems can, in general, be "shuffled" to create a problem class of random-linkage problems whose structure cannot be exploited by linkage-preserving crossover techniques, such as one- or two-point crossover. Despite the fact that these problems have successfully shown what modularity-exploiting algorithms are good for, they all hold the misguided assumption that the number of bits contained per module, or the number of modules contained in modules of later hierarchical levels, has to be small and fixed. This assumption can perhaps be attributed to the following: given that a hill-climber can solve a module and that the modules are separable, a hill-climber can solve all modules and hence the entire problem. On the other hand, given that a hill-climber cannot easily solve the modules, the solutions will have to be "guessed". Therefore, the required search to find the solution to a module would need to be exponential in the size of a module. So the modules have to be small. However, this reasoning is erroneous as it assumes *a priori*

knowledge about the size of the modules: that they are small. Additionally, it does not allow previous work to show a distinction between multi-scale and conventional search algorithms. In Sections 3.1.1 and 3.1.2, we introduce simple, random-linkage, modular problems, with two levels of hierarchy (i.e., many modules containing many bits). But, disregarding the assumption held in previous works, these problems contain large modules. Specifically, they consist of n modules of size k, both growing as a function of the overall problem size: $N = nk$, $n = k = \sqrt{N}$.

Here we have shown that searching with an appropriate macro-variation operator transforms the neighbourhood of the fitness landscape from one where the locally optimal configurations were separated by a number of lower-fitness intermediate sites to one where they are one variational step away. For this transformation to be successful, automatically and reliably discovering module solutions is essential. Therefore, the combination of an exploration mechanism that searches *for* module solutions and an exploitation mechanism that searches *in the space* of module solutions can significantly reduce the search space and the difficulty of problems that facilitate decomposition.

2.3 Related Work

2.3.1 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) and GAs share a number of characteristics, such as the use of candidate solution populations, and natural selection acting on the fitness of each individual to determine which areas of the fitness landscape would benefit from further exploration. Their main difference lies in how the population evolves across generations. In regular GAs, the mutation and crossover operators are used to generate new solutions while natural selection directs the search, whereas EDAs build models representing the structure of the problem and use them to produce the new generation of individuals. The objective of EDAs is to combine the best characteristics of population-based search and machine learning techniques to build a model of interactions between variables in a problem and identify the epistatic dependencies. In the following paragraphs we provide the reader with a brief overview of a small number of EDAs. We focus on the more prominent and sophisticated multivariate EDAs which are of more interest as they are targeted at problems with complex interactions between their variables. For a more detailed review on EDAs and building probabilistic models, we refer the reader to [38, 39].

BOA

Pelikan *et al.* [8] introduce the Bayesian optimisation algorithm (BOA), which constructs a Bayesian network representing the dependencies between problem variables. The probabilistic model is built from a number of above-average fitness candidate solutions, and is updated every generation. A greedy heuristic is used to find the Bayesian network structure that best fits the data from the aforementioned set. The

dependency graph is accompanied by a table for each vertex, which defines the variable value conditional probabilities in terms of the states of its parent vertices. Several Bayesian networks which consist of the dependency graph and the vertex conditional probabilities are generated using greedy search. New individuals are generated by sampling the constructed Bayesian network that best fits the data.

hBOA

The hierarchical Bayesian optimisation algorithm (hBOA) is an evolution of BOA whose objective is to solve hierarchical problems and improve the efficiency of building the probabilistic model that represents dependencies [40, 41]. Learning a Bayesian network structure is an NP-hard problem, making the problem intractable for large problems. The main difference between BOA and hBOA is the introduction of local structures in the form of decision graphs, which have been shown to improve efficiency. Also, a technique called *restricted tournament replacement* is utilised as a diversity maintenance mechanism to decide which members of the population should be replaced by the generated candidate solution.

DSMGA+

Dependency structure matrix driven genetic algorithm (DSMGA) is a hybrid of GAs and EDAs [42]. Like most EDAs, a model of the problem structure is constructed, but it is not used to generate new individuals by sampling. Instead, it is used to apply variation directly to the population. In the first stage, a dependency structure matrix (DSM) is constructed that uses the whole population to estimate the dependencies. In the second stage, a clustering of the data which indicates the structure of the problem is found. Lastly, *BB-wise crossover* is used on individuals to exchange building-blocks. Using the clustering as a guide for where the building-blocks are, crossover points are limited to without building-blocks, thus guaranteeing that no disruption takes place and enabling their exchange. DSMGA+ is a recursive version of DSMGA that enables the discovery of greater than pair-wise interactions between variables [9]. Additionally, the authors introduce what they call *sub-structural chromosome compression* which replaces a building-block with one bit when its alleles are close to converging to only two configurations, effectively reducing the problem complexity and the search space when parts of the problem have been solved.

EDA Properties

In the previous paragraphs we briefly described a number of algorithms that are described as estimation of distribution algorithms. With the exception of DSMGA+ from those described (and with only few exceptions in general), EDAs share one fundamental property: a model is used to generate new individuals. More importantly, we are interested in identifying what the differences between the approaches are. One of the main differences lies in the structure the model uses to represent dependencies and its ability to produce a relatively accurate estimate of the problem structure. The second

fundamental difference is how the model is used to generate new candidate solutions: whether new individuals are generated from the model or whether individuals are modified by macro-scale variation operators. Other differences include: (i) whether the model is reconstructed or updated after every generation; and (ii) whether the whole population or part of it is replaced every generation. Also, EDAs hold the assumption that modules are of short length, which is a problem when the length of the modules scales with the problem size (like for our problems). Yu *et al.* [43] showed that EDAs require a population size that is exponential in the length of the module to build an accurate model of the problem structure.

2.3.2 Other Approaches

EDAs are not the only approaches to exploiting modularity. Watson *et al.* [10] show a Hopfield network [44] that self-organises by uncovering correlations in locally optimal solutions, using Hebb's rule [45]. The learned correlations are used to modify the variation operator so that it achieves macro-scale mutation. In effect, the modified mutation operator warps the neighbourhood of the search space, enabling the operator to search in the space of modules instead of single variables. This approach has shown that, with sufficient learning, the algorithm is also capable of "remembering" solutions. Clune *et al.* [46] have shown that modularity is evolved by directly selecting for both the minimisation of the connection costs within a network as well as the maximisation of its performance. The objective was to evolve neural networks capable of solving an eight-pixel retina problem by identifying whether a target pattern existed in one of the retinas (L-OR-R) or both retinas (L-AND-R). Not only do the results show that selecting for both the performance and connection costs produces better performing and modular networks, but also that the modular networks adapt quicker than the entangled networks when switching between the two retina problems (L-OR-R and L-AND-R).

Chapter 3

Methods

In this chapter, we describe the main methods we use for our experiments. We begin by introducing the two classes of multi-scale modular problems we use, and then describe the two algorithms we use for testing, along with their macro-variation operators.

3.1 Multi-Scale Modular Problems

3.1.1 Scalable Building-Blocks: A Decomposable Problem

Using the tight-linkage version of this problem, Watson and Jansen [5] showed how sexuals can consistently outperform asexuals and reach areas of higher fitness. In this problem, all n order-k modules are multi-peaked, but, importantly, a simple hill-climbing process is capable of reaching each peak from many initial conditions. The full N-bit problem is constructed by concatenating, without between module epistasis, all modules:

$$F(\boldsymbol{g}) = \sum_{i=1}^{n} f(\boldsymbol{m}_i), \tag{3.1}$$

where $\boldsymbol{g} = \langle g_{r_1}, g_{r_2}, \ldots, g_{r_N} \rangle$ is the genotype of length N; $\boldsymbol{r} = \langle r_1, r_2, \ldots, r_N \rangle$ is a mapping between the variables and the randomised functional structure of the problem; $\boldsymbol{m}_i = \left\langle g_{r_{(i-1)k+1}}, g_{r_{(i-1)k+2}}, \ldots, g_{r_{ik}} \right\rangle$ is the ith module of the genotype containing k variables; $n = N/k$ is the number of modules; and f is the sub-function which determines the fitness contribution of each module:

$$f(\boldsymbol{m}_i) = \sum_{j=1}^{T_i} c\big(\boldsymbol{m}_i, \boldsymbol{t}_{ij}\big), \text{ where} \tag{3.2}$$

$$c\big(\boldsymbol{m}_i, \boldsymbol{t}_{ij}\big) = \begin{cases} w_{ij} & \text{if } \left|\boldsymbol{m}_i - \boldsymbol{t}_{ij}\right| = 0 \\ \left(1 + \left|\boldsymbol{m}_i - \boldsymbol{t}_{ij}\right|\right)^{-1} & \text{otherwise,} \end{cases} \tag{3.3}$$

where T_i is the number of optima within the ith module; $\boldsymbol{t}_{ij}$ is the jth target string (i.e., optimum) within the ith module; $\left|\boldsymbol{m}_i - \boldsymbol{t}_{ij}\right|$ is the Hamming distance between the

module configuration and the target string; and $w_{ij} > 1$ is the fitness contribution, or weighting, of the target string $\boldsymbol{t}_{ij}$.

The fitness of each module is a sum of the fitness contributions, as defined by c, determined by the comparison of the module configuration with the set of target strings, T. Maximal fitness contribution with regard to a particular target string can only be achieved when the module configuration matches the target string (i.e., $|\boldsymbol{m}_i - \boldsymbol{t}_{ij}| = 0$); otherwise, the fitness contribution is inversely proportional to the Hamming distance.

With some care in interpretation, Figure 3.1 shows the fitness landscape of a module as a function of the Hamming distance from the global optimum, with two optima, or target strings, $T = 2$, and the global and local optimum fitness contributions set to 2 and 1 respectively, $w_{go} = 2$ and $w_{lo} = 1$.

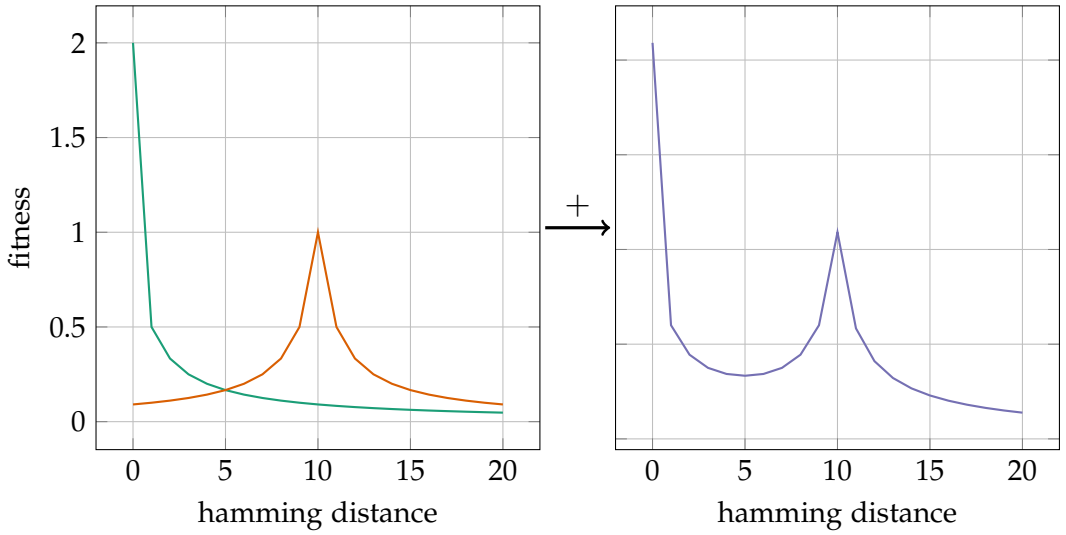

Figure 3.1: A 20-bit, two-optima scalable building-blocks problem module, with the Hamming distance between the two optima set to 10, $w_{go} = 2$ and $w_{lo} = 1$. In the x and y axes, the Hamming distance from the globally optimal target string and fitness respectively. On the left, the green line shows the global optimum fitness, and the orange line shows the local optimum fitness as a function of the Hamming distance from the global optimum. On the right, the two fitness contributions are added to create a module with multiple peaks, as per Equations (3.2) and (3.3).

Unlike the work by Watson and Jansen [5], the ordering of the variables here is "shuffled", or randomised. Hence, linkage-preserving crossover cannot readily exploit the problem structure because the variables of each module are arbitrarily distributed across the genome. To exploit the underlying modular structure of the problem, an algorithm must first be capable of discovering the epistatic linkage map itself. For the purposes of our investigation, we refer to this particular instantiation of the problem as the *scalable building-blocks* (SBB) problem because in contrast to, for example, the concatenated trap functions [37], the order of the blocks scales as a function of the overall problem size, $n = k = \sqrt{N}$.

When the number of optima per module is z, the overall number of locally optimal configurations is z^n (i.e., exponential in the number of modules, n). The global optimum is unique and is only found when all modules match the globally optimal target string. Three factors contribute to the difficulty of the problem: (i) the number of modules, n; (ii) the number of local optima per module, z; and (iii) the Hamming distance between the locally optimal configurations. An increase in n or z sees the number of overall locally optimal configurations in the search space increase exponentially, meaning that the unique global optimum is more difficult to find. On the other hand, increasing k will implicitly increase the Hamming distance between the locally optimal configurations, thus reducing the likelihood of producing the required variation to jump from one local optimum to another.

The algorithms we introduce are capable of globally optimising any instantiation of this problem class (e.g., complementary or non-complementary optima or with multiple optima per module). For the aims of this paper, we focus on a particular instance of the problem with two non-complementary optima per module. This is primarily used to show that the algorithms we introduce are not biased towards problems with complementary optima, but also show how conventional algorithms perform when the difficulty of the problem is relatively relaxed (i.e., the Hamming distance between optima is not maximal at $\lfloor (k/2) \rfloor$). For easier visualisation, the global optimum is set at all ones, $\boldsymbol{t}_{go} = 1^k \forall i$, whereas the local optimum, $\boldsymbol{t}_{lo}$, has ones and zeros at every odd and even index respectively.

The SBB has some unique characteristics in the way the modules are defined which differentiate it from other synthetic problems. Specifically, each module is multi-modal, but the basin of attraction for the fittest module solution is appreciable. Unlike, for example, the fully deceptive concatenated trap function [37] which makes the search difficult for any algorithm because of the difficulty of finding the globally optimal module solution. Also, the basins of attraction are a proportion of the length of the module instead of fixed-length, and that is what is important [5].

3.1.2 Variable Structural Modularity: A Nearly-Decomposable Problem

The second problem we use for testing is the *variable structural modularity* (VSM) problem [47], with the parameters to the problem identical with [10]. Our test problem is defined by a randomly initialised weight matrix, which results in a multi-modal, nearly-decomposable [33] problem of pair-wise constraints. Essentially, the VSM problem is a particular instantiation of the NP-complete weighted MAX-2-SAT problem. There are two classes of the problem: random constraints (RC), without explicit modularity that can be exploited; and modular constraints (MC) which is a macro-scale version of the RC class as explained below.

Even though both problem classes are challenging multi-modal problems, RC is not sufficient to illustrate the affordances and limitations of our algorithms because there is no underlying modular structure to exploit. By using MC, we want to show an idealised scenario where an algorithm that can discover and exploit the epistatic linkage of the problem can reliably solve the problem, while algorithms that cannot identify structural information find the problem pathologically difficult.

The RC matrix is defined by an $N \times N$ symmetric matrix, $\boldsymbol{W}$, whose diagonal elements all take the value d_i, $w_{ij} = d_i = 1$ if $i = j$, and the rest are uniformly initialised to fixed magnitude, variable sign values, $w_{ij} = \{-d_e, d_e\}$:

$$w_{ij} = \begin{cases} d_i & \text{if } i = j, \\ \{-d_e, d_e\} \text{ with equal probability} & \text{otherwise,} \end{cases}$$

where d_i is the self-weight, and $d_e \leq d_i$ is the magnitude of the interaction between variables. The MC matrix is defined in terms of the RC matrix, which results in a symmetric, explicitly modular matrix, $\boldsymbol{\Omega}$, of size $N \times N$, where $N = nk$; n is the number of the modules; and k is the order of each module. The MC weight matrix is initialised as follows:

$$\omega_{ij} = \boldsymbol{W}\left(\left\lfloor \frac{i}{k} \right\rfloor, \left\lfloor \frac{j}{k} \right\rfloor\right). \tag{3.4}$$

This creates a weight matrix with strong intra-module dependencies across the block diagonal, and blocks of much weaker, fixed magnitude and variable sign inter-module dependencies elsewhere. In MC, d_i is the strength of interaction between co-dependent, internal variables (i.e., within the same module); and $d_e \leq d_i$ is the magnitude of the interaction between co-dependent, external variables (i.e., between modules). An example RC matrix with its corresponding MC matrix can be seen in Table 3.1. The problems utilise state configurations represented in the Ising spin convention, $\sigma \in \{-1, +1\}$, $\boldsymbol{\sigma} = \langle \sigma_1, \sigma_2, \ldots, \sigma_N \rangle$, to calculate the fitness, which is given by:

$$f(\boldsymbol{g}) = -\sum_{i=1}^{N}\sum_{j=1}^{N} g_i \omega_{ij} g_j, \text{ or} \tag{3.5}$$

$$f(\boldsymbol{g}) = -(\boldsymbol{g}\boldsymbol{\Omega}\boldsymbol{g}^{\intercal}), \tag{3.6}$$

where $\boldsymbol{g}$ is the genotype, represented as a $1 \times N$ row vector, and $\boldsymbol{g}^{\intercal}$ is its transpose. This creates a minimisation problem of weighted, pair-wise constraints, where the fitness contribution is beneficial only when the sign of the product of two variables (g_i and g_j) is the same as the sign of their weighting, ω_{ij} (i.e., when the constraint is satisfied).

Weight Matrix												
RC				MC								
				1.00	1.00	0.01	0.01	0.01	0.01	0.01	0.01
				1.00	1.00	0.01	0.01	0.01	0.01	0.01	0.01
				0.01	0.01	1.00	1.00	**-0.01**	**-0.01**	0.01	0.01
				0.01	0.01	1.00	1.00	**-0.01**	**-0.01**	0.01	0.01
1.00	0.01	0.01	0.01	0.01	0.01	-0.01	-0.01	1.00	1.00	0.01	0.01
0.01	1.00	**-0.01**	0.01	0.01	0.01	-0.01	-0.01	1.00	1.00	0.01	0.01
0.01	-0.01	1.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	1.00	1.00
0.01	0.01	0.01	1.00	0.01	0.01	0.01	0.01	0.01	0.01	1.00	1.00

Table 3.1: Example RC matrix with its corresponding MC matrix ($N = 8$, $n = 4$, $k = 2$, $d_i = 1$, $d_e = 0.01$). A weight in RC and the corresponding module in MC are shown in bold.

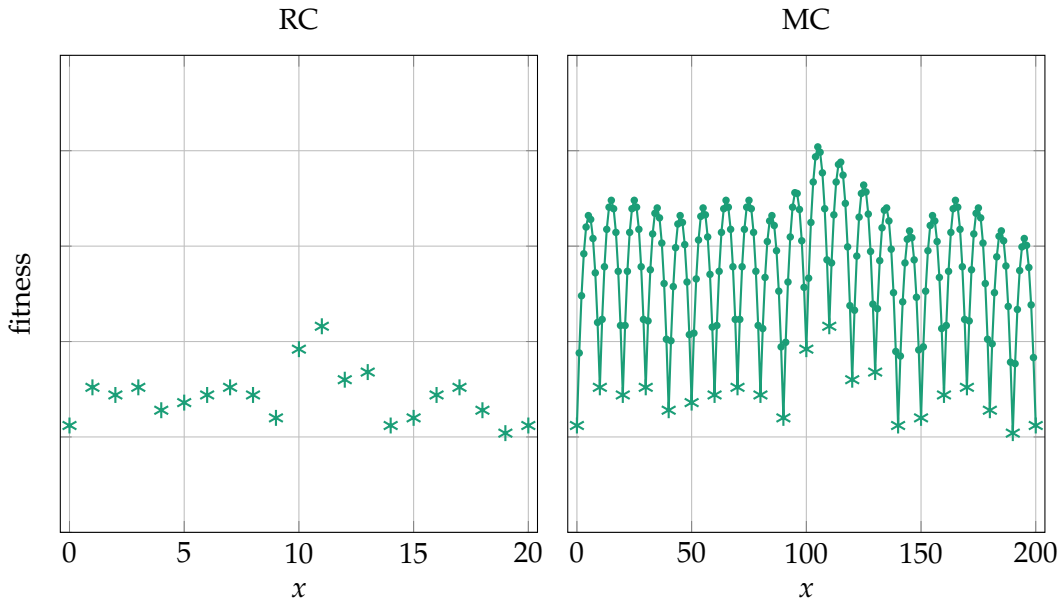

Figure 3.2: The cross-section for a VSM problem with $n = 20$ modules of length $k = 10$ created by the state configurations in the language $-1^x1^{(N-x)}$ for $x = [0, N]$. Asterisk markers in MC are the subset of configurations that are locally optimal. That is, configurations whose variables within each module take the same value; i.e., configurations in the language $-1^{kx}1^{(N-kx)}$ for $x = [0, n]$. These points correspond exactly to the points in RC when their magnitude is adjusted by a factor d_e. All intermediate points in MC are of worse fitness. Jumping from one locally optimal configuration to another requires the simultaneous change of all k variables within a module.

RC matrices create multi-modal fitness landscapes, but how to globally optimise this type of problem is beyond the scope of this work. Even if it might not be the best approach, by using a simple restart hill-climber it is possible to find high-fitness configurations for problems of size $N = n = 20$. However, in an equivalent MC matrix with, for example, $k = 10$, $N = nk = 200$, the optimisation problem becomes very difficult to solve reliably using only local information because the much stronger within than without module dependencies, $d_e \ll d_i$, along the block diagonal result in local optima corresponding to the configurations in the language $\left\{-1^k, 1^k\right\}^n$, which is exponential in the number of the modules (Figure 3.2). Additionally, since the Hamming distance between locally optimal configurations is maximal at k, the likelihood of producing the k-bit change required to substitute one module solution with another decreases exponentially with k when which variables need changing is known but not what value to change them to, or, in the more realistic scenario, when the module partitions are unknown, it decreases exponentially with the overall problem size [5]. Importantly, it is possible to shuffle the ordering of rows and columns to prevent linkage-preserving crossover from exploiting the tightly linked building-block structure.

When a mechanism is capable of identifying the modular structure of the MC problem and searching in the space of module solutions using an appropriate macro-variation operator, the optimisation problem becomes equivalent to hill-climbing in the unstructured RC problem. As such, the fitness-minimising trajectories in MC create an

idealised nearly-decomposable system [33]. The strong intra-module dependencies are quickly and trivially resolved, but the challenge lies in identifying the optimal module solution under a specific genetic context. That is, because of the existence of inter-module dependencies, there is no globally optimal module solution when considering the optimisation of a module in complete isolation. Unlike in the SBB problem where it is possible to solve each module in complete isolation, the VSM problem is more challenging because the problem is deceptive: no clear path to the global optimum is provided and modules can only be solved in partial isolation.

3.2 Discrete Differential Evolution, dDE

Here, we define the Discrete Differential Evolution (dDE) algorithm and describe its key functions. Similar to DE, the algorithm is simple to understand and straightforward to implement. Much of its simplicity comes from the fact that no explicit learning takes place. Unlike previous work, we discard the idea of explicit linkage-learning and revisit the abandoned idea of *implicitly* scaling search to search in the space of module solutions, or building-blocks, as per the building-block hypothesis [6, 7, 31]. Although DE is naturally suited to real-valued optimisation problems, a number of works in the literature have shown discrete DE variants applicable in a number of areas [14–18]. In this paper, however, we investigate the ability of the DE algorithmic principles to globally optimise discrete, random-linkage, modular problems.

For any search process be fruitful when solving problems facilitating decomposition, there are two requirements: (i) an exploration mechanism that searches *for* module solutions; and (ii) an exploitation mechanism that searches *in the space* of module solutions. The exploration mechanism in dDE is a simple population of hill-climbers. To utilise the information the hill-climbers discover and achieve macro-variation, other algorithms require an intermediate step: the explicit learning of the epistatic linkage map of the problem, using techniques such as Bayesian networks [8, 40] or correlation matrices [10]. Our approach differs in that there is no intermediate learning step: macro-variation is achieved implicitly. This is possible because the macro-variation operator is able to identify and exploit the underlying modular structure of the solution space that is emphasised implicitly by the distribution of individuals in the population. The following is a rough outline of the algorithm:

0. Randomly initialise current population, $\boldsymbol{X}$;
1. Generate a random number between 0 and 1. Compare the randomly generated number with a pre-determined probability and do one of the following accordingly:
   a. Perform bit-flip on $\boldsymbol{X}$ to generate a new population, $\boldsymbol{U}$;
   b. Perform macro-variation on $\boldsymbol{X}$ to generate a new population, $\boldsymbol{U}$;
2. Selection: one-to-one elitist tournament between each pair of ith individuals in $\boldsymbol{X}$ and $\boldsymbol{U}$, with the winners populating $\boldsymbol{X}$;
3. Increase generation count and go to step 1.

Note the lack of intermediate learning step and the two mutually exclusive modes of operation: bit-flip and macro-variation. The bit-flip operator is used to generate locally optimal configurations, whereas the macro-variation operator utilises information within the population when configurations are locally optimal to produce targeted, simultaneous variable changes. By probabilistically switching between the two modes of operation, the "greedy" selection scheme discards individuals when the individuals in the population (a) are not locally optimal, thus the macro-variation operator produces random rather than targeted variable changes; or (b) are locally optimal, thus the bit-flip operator only produces deleterious mutations. Consequently, the probabilistic switch between the two modes of operation in combination with the unique nature of the dDE macro-variation operator results in the implicit scaling of the search.

Overall Procedure

dDE relies on two control parameters: the population size NP and the probability of performing macro-variation, p_{mv}. The scaling factor, F, and crossover rate, Cr, control parameters are both fixed to 1. After the population is initialised to discrete values, each generation consists of the exploration mechanism (bit-flip) or the exploitation mechanism (DC), and selection. The overall procedure is shown visually in Figure 3.3, and in pseudo-code format in Figure 3.4. What follows is a detailed description of the stages of the algorithm.

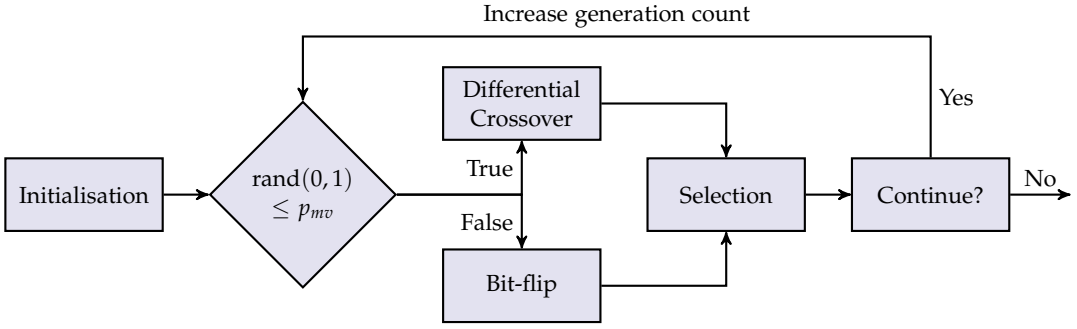

Figure 3.3: The DE optimisation process. The algorithm uses two modes of operation: bit-flip and macro-variation.

Initialisation

Individuals in dDE must be initialised to discrete values. The operators used are insensitive to the choice of variable allocation conventions. That is, the population can be initialised to follow either Ising spin conventions ($\sigma \in \{-1, +1\}$) or evolutionary computation conventions ($x \in \{0, 1\}$).

Bit-flip

For the exploration aspect of higher-scale search, dDE uses a simple bit-flip operator. The operator randomly selects a variable, $j = [1, N]$, in all NP individuals and flips

1: $\boldsymbol{X} \leftarrow$ Randomly initialise NP individuals to discrete values.
2: $\boldsymbol{a} \leftarrow$ Evaluate the NP individuals in $\boldsymbol{X}$.
3: **while** ending criterion is not satisfied **do**
4: **if** rand$(0,1) \leq p_{mv}$ **then** ▷ Perform differential crossover.
5: **for** $i \leftarrow 1$ to NP **do**
6: $r_1^i \leftarrow$ randi$(1, NP)$ ▷ $r_1^i \neq i$
7: $r_2^i \leftarrow$ randi$(1, NP)$ ▷ $r_2^i \neq r_1^i \neq i$
8: $\boldsymbol{v}_i \leftarrow \gamma(\boldsymbol{x}_i + \boldsymbol{x}_{r_1^i} - \boldsymbol{x}_{r_2^i})$ ▷ Equation (3.7).
9: **end for**
10: **else** ▷ Perform bit-flip.
11: **for** $i \leftarrow 1$ to NP **do**
12: $j \leftarrow$ randi$(1, N)$ ▷ Variable to flip.
13: $\boldsymbol{v}_i \leftarrow \boldsymbol{x}_i$ ▷ Copy the individual.
14: Flip v_{ij} ▷ Flip the variable.
15: **end for**
16: **end if**
17: $\boldsymbol{U} \leftarrow \boldsymbol{V}$ ▷ $Cr = 1$
18: $\boldsymbol{b} \leftarrow$ Evaluate the NP individuals in $\boldsymbol{U}$.
19: **for** $i \leftarrow 1$ to NP **do** ▷ Selection.
20: **if** $\boldsymbol{u}_i$ is as fit as or fitter than $\boldsymbol{x}_i$ **then**
21: $\boldsymbol{x}_i \leftarrow \boldsymbol{u}_i$
22: $a_i \leftarrow b_i$
23: **end if**
24: **end for**
25: Increase generation count.
26: **end while**

Figure 3.4: Pseudo-code for dDE. NP is the population size; N is the problem size; rand(x, y) is a uniform floating-point random number generator in the range $[x, y]$; and randi(x, y) is a uniform integer random number generator in the range $[x, y]$.

it. Note that the operator could be modified to flip more variables per individual, but that is unnecessary because its objective is to identify module solutions; not to jump between optima.

Differential Crossover

Central to the scaling of the search in dDE is the idea of taking the difference between two individuals and adding it to another individual—a process called differential mutation in the DE literature (Section 2.1). Here, we show that, in the discrete rather than continuous space, by taking the difference of two locally optimal configurations and adding it to another locally optimal configuration, macro-variation is achieved, with the resulting configuration taking the variables required to produce a jump between local optima. Consequently, by using the operator directly on the population of locally optimal configurations produced by the hill-climbing process, the epistatic linkage map of the problem is emphasised by the differences between the individuals in the population.

Note that, this operator does not introduce variation by flipping; instead, it emphasises the variables on which crossover should be performed on to substitute one module

solution with another. Consider a problem with three modules with two module solutions each: m_1 with A and a; m_2 with B and b; and m_3 with C and c. Also assume that there are three individuals within the population: (1) aBc; (2) ABC; and (3) aBC. Say we take the difference between (2) and (3). The non-zero values in the difference vector emphasise the crossover points. That is, (2) and (3) have the same module solution in modules m_2 and m_3, but not in m_1: (2) has A and (3) has a. Hence, the differences between A and a are identified. Given the crossover points, non-zero values in the difference vector denote the values (1) should take to move from a to A or stay at a. Because this process does not include flipping, we call this operator *differential crossover* (DC).

DC is the equivalent of the differential mutation operator, but bound by the constraints of search in binary spaces. For our purposes, we introduce the *d-current/1* mutation scheme, as shown below:

$$\boldsymbol{v}_i = \gamma(\boldsymbol{x}_i + \boldsymbol{x}_{r_1^i} - \boldsymbol{x}_{r_2^i}), \tag{3.7}$$

where γ is a threshold function resetting the variables to legal binary values, $r = [1, NP]$ and $r_1^i \neq r_2^i \neq i$. By adding the difference vector to the base vector, $\boldsymbol{x}_i$, the threshold function, γ, simulates the effect of super-imposing the non-zero values of the difference vector onto $\boldsymbol{x}_i$. There are three things to note here. First: because the algorithm operates in the discrete space, the magnitude of the perturbation is fixed and what varies is the direction. This allows for the removal of the scaling factor, F, control parameter. Second: the operator generates mutant vectors, $\boldsymbol{v}_i$, but because the crossover rate is fixed to $Cr = 1$, the trial and mutant vectors are identical.[1] Third: because no crossover takes place between the ith target and mutant vectors, we are able to use the ith target vector as the base vector in Equation (3.7). The effect of the operator is shown in Table 3.2 and also visually in Figure 3.5.

Selection

There are no differences between the DE and dDE selection stages: the ith trial vector, $\boldsymbol{u}_i$, takes the place of the ith target vector, $\boldsymbol{x}_i$, in the next generation if-and-only-if it is as fit or fitter. However, there are a few key aspects of it that enable the probabilistic switch between the bit-flip and DC operators. First, because the fitness of each ith pair from the target and trial vectors populations is compared, by preceding the selection stage with the bit-flip operator, the overall procedure is identical to that of a population of hill-climbers—meaning that each individual is likely to be exploring a different area of the fitness landscape which is a vital prerequisite for the success of the DC operator. Second, the "greedy" and steady-state replacement scheme discards individuals when the operators are not effective. That is true when bit-flip is performed on locally optimal configurations or when DC is performed on non-locally optimal configurations. Any other replacement method would not allow for the probabilistic switch between the two modes of operator and, consequently, the scaling of the search would have to be explicit.

[1] The crossover rate is set to 1 because, if DC generated mutant vectors whose elements were arbitrarily crossed over with target vectors to produce trial vectors (as is the case in DE), the linkage information that DC retained would be disrupted.

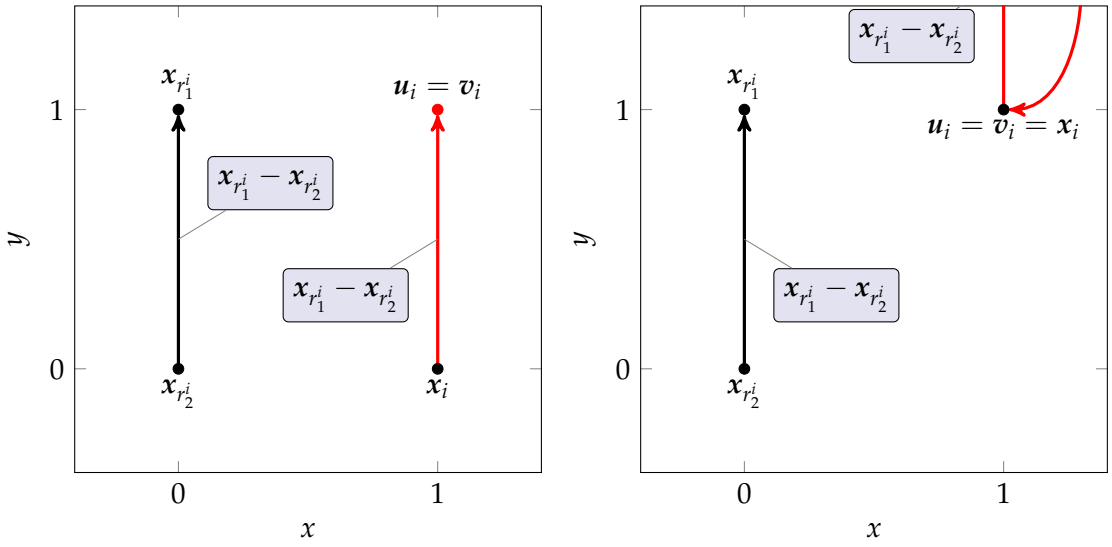

Figure 3.5: The differential crossover operator behaviour shown visually. Since the objective is to generate children which retain modules and we do not assume *a priori* knowledge of complementary optima, no naive bit-flipping takes place. Instead, the threshold function, γ, simulates the effect of superimposing the difference onto another individual (i.e., crossover).

3.3 Multi-Parent Evolution, MPE

The basic idea behind the DC macro-variation operator is to use the differences between individuals to produce targeted, simultaneous variable changes and thus search in the space of module solutions. An alternative to that would be to use *similarities* between individuals. This is equivalent to performing *occurrence-based scanning* (OB-Scan) crossover [20, 48]. Essentially, OB-Scan is a generalisation of uniform crossover that uses multiple parents. Instead of uniformly crossing over elements between two parents, the allele at each locus of the child is determined by majority voting. Note that when an even number of individuals vote and there is a tie, it can be broken, for example, randomly. However, because the aim of the operator is to achieve macro-variation, tie-breaking would make the operation a stochastic process, which does not guarantee the substitution of module solutions. To avoid that, we use three parents, as follows:

$$\boldsymbol{v}_i = \gamma(\boldsymbol{x}_i + \boldsymbol{x}_{r_1^i} + \boldsymbol{x}_{r_2^i}), \tag{3.8}$$

where $\boldsymbol{X}$ is represented in the Ising spin convention ($\{-1,+1\}$); γ is a threshold function resetting the variables to legal binary values; $r = [1, NP]$; and $r_1^i \neq r_2^i \neq i$. Or using the mode instead:

$$\boldsymbol{v}_i = \mathrm{Mo}(\boldsymbol{T}_i), \tag{3.9}$$

where $\boldsymbol{X}$ is represented in the evolutionary computation or Ising spin convention ($\{0,1\}$ or $\{-1,+1\}$); $\boldsymbol{T}_i = (\boldsymbol{x}_i^\mathsf{T} \quad \boldsymbol{x}_{r_1^i}^\mathsf{T} \quad \boldsymbol{x}_{r_2^i}^\mathsf{T})^\mathsf{T}$ is a $3 \times N$ matrix with each row being a state configuration; and Mo is a function that returns the mode in each column. The effect of the operator is shown in Table 3.2.

	Crossover Method	
Vector	DC	OB-Scan
$\boldsymbol{x}_{r_1^i}$	0010010010010	0010010010010
$\boldsymbol{x}_{r_2^i}$	1010110000101	1010110000101
Difference	-000-000+0-+-	
Intermediate	0 0 1 010	
$\boldsymbol{x}_i$	1001010100110	1001010100110
$\boldsymbol{v}_i$	0001010110010	1010010000110

Table 3.2: The behaviour of differential crossover (DC) and occurrence-based scanning (OB-Scan). Unlike the uniform, one- and two-point crossover operators, DC and OB-Scan deterministically generate children which retain the locally optimal configurations of the modules. DC: Taking the difference of two individuals defines a tertiary (-1 shown as $-$, 0, $+1$ shown as $+$) difference vector, or, for clarity, an intermediate schema, which is superimposed on a parent individual. This forces some bits of the parent to take the 0 allele, others to take the 1 allele, and others are left unchanged. OB-Scan: The allele the child takes is decided by majority voting. Hence, the allele of a child at a specific bit is only different from the parent if-and-only-if that allele exists in both auxiliary vectors. For easier visualisation, the bits of the children that differ from their parents are shown in colour.

For the purposes of this thesis, we use OB-Scan in the context of dDE. That is, we replace the DC operator with the OB-Scan operator and keep everything else the same. We do that because the hypotheses and motivations for using the bit-flip operator and DE selection stage which we discuss in the previous section also hold for the use of OB-Scan. Additionally, this allows for a more direct comparison of the two operators. The pseudo-code for the algorithm is identical to dDE, but with Equation (3.9) replacing Equation (3.7). We call this algorithm *multi-parent evolution* (MPE).

3.4 Other Algorithms

To show the difference between the macro-variation crossover operators and uniform crossover, we use what we call *uniform crossover evolution* (UCE). UCE is the same algorithm as dDE, but with DC replaced by uniform crossover. We also use two random-restart hill-climbers: HC and HC*. HC is a typical bit-flip hill-climber with a mutation rate of $\mu = 1/N$, which allows us to show that the probability of starting in all the right basins of attraction decreases exponentially with the number of local optima. HC* is a hill-climber with its mutation rate set to h/N, where h is the Hamming distance between the optima, which allows us to show that even when the hill-climber is given information about the problem, the probability of changing all the right variables simultaneously decreases with the size of the problem.

Chapter 4

Simulation Results

In this chapter, we provide empirical results for the behaviour and performance of dDE and MPE. In Section 4.1, we investigate how the performance of dDE and MPE is affected by changes to the control parameters. In Section 4.2, we investigate the behaviour of dDE and MPE and their respective macro-variation operators on fixed-size SBB and VSM problems. In Section 4.3, we show how dDE, MPE and a number of conventional algorithms scale with the problem size. Lastly, in Section 4.4, we compare dDE and MPE by investigating what the differences between the DC and OB-Scan macro-variation operators are.

4.1 Control Parameter Tuning

Here, we use 100-bit SBB and VSM problems, with 10 modules of length 10 ($n = k = 10$, $N = nk = 100$) to investigate how changes in the population size, NP, and probability of performing macro-variation, p_{mv}, control parameters affect the behaviour of dDE and MPE. The ranges of control parameters tested were $p_{mv} = [0.005, 0.905]$ with a step size of $+0.1$ and $NP = [50, 550]$ with a step size of $+50$, resulting in a total of 100 sets of control parameters. We measure the number of successful executions (i.e., global optimum found) and the number of accumulated function evaluations across 30 independent executions of the SBB and VSM problems. These results are used to identify the control parameters that are used for the experiments in the following sections.

Figure 4.1 shows the convergence probability and Figure 4.2 the number of accumulated function evaluations. dDE is less sensitive to changes in the control parameter settings, as it retains the ability to reliably solve the problems, with increases in the population size making up for larger p_{mv} values. Conversely, MPE can only solve the problems reliably with p_{mv} fixed at the minimum value tested: 0.005. The choice of control parameters is affected by the difficulty of the problem for both dDE and MPE. For the successful optimisation of the VSM problem, a larger p_{mv} value requires a greater increase in population size compared to that on the SBB problem. dDE is more robust than MPE for both problems. The outcome is similar when considering the function evaluations instead of the convergence probability. But both algorithms

perform similarly when their control parameters are optimally tuned. Specifically, both require a low p_{mv} to achieve optimal performance (albeit dDE can afford a higher value).

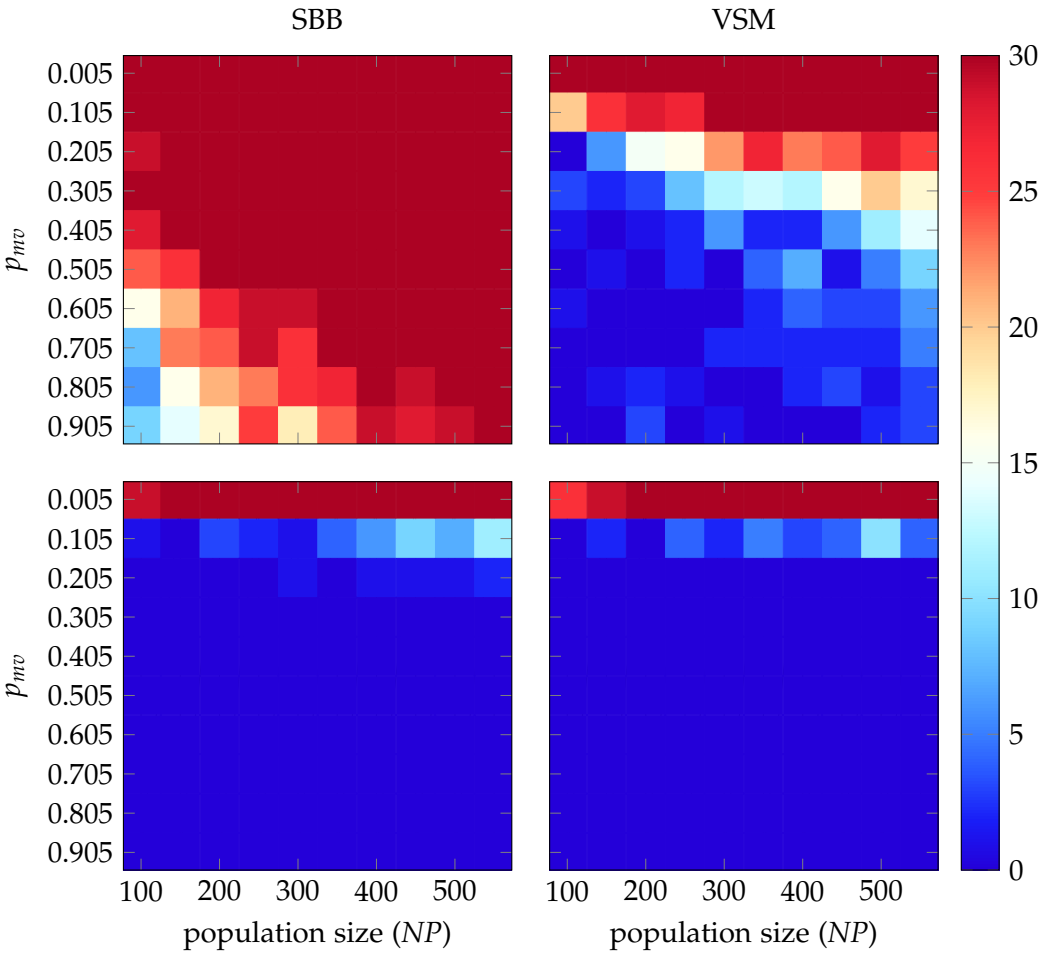

Figure 4.1: How the convergence probability of dDE (top row) and MPE (bottom row) is affected by changes in the population size, NP and the probability of performing crossover, p_{mv}. Various sets of the control parameters were tested on 30 independent executions of 100-bit SBB and VSM problems, with $n = k = 10$. The colour is the number of successful executions. MPE does not improve its probability of convergence with increases in the population size and is only capable of solving the problems reliably when p_{mv} is fixed at the minimum value tested. On the other hand, dDE retains remains reliable when both the population size and p_{mv} are increased.

4.2 Operation on 400-variable Problems

Here, we investigate the behaviour of dDE and MPE on 400-bit SBB and VSM problems, with 20 modules of length 20 each ($n = k = 20$, $N = nk = 400$). For the SBB problem, we use 2 optima per module: the globally optimal target string consists of all 1s and the locally optimal target string of 1s and -1s at every odd and even index respectively.

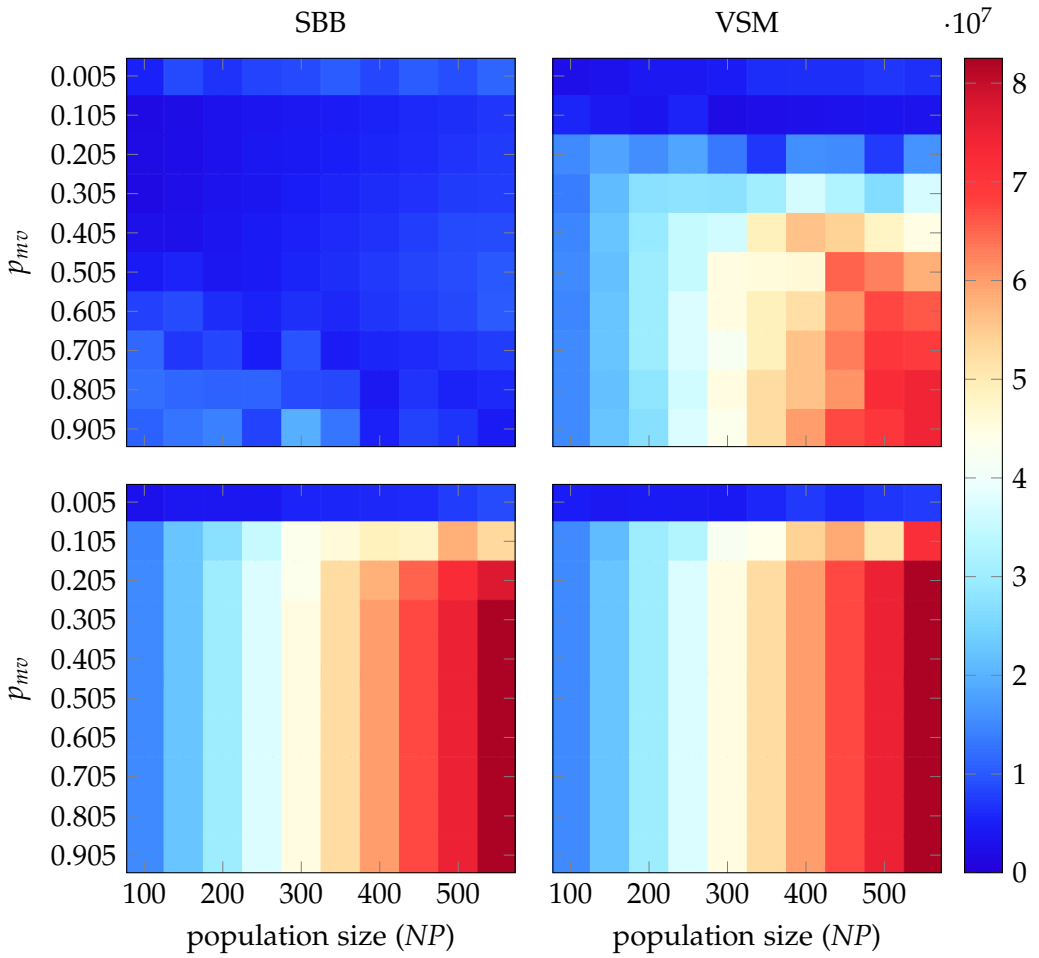

Figure 4.2: Accumulated function evaluations for dDE (top row) and MPE (bottom row). Various sets of the control parameters were tested on 30 independent executions of 100-bit SBB and VSM problems, with $n = k = 10$. The colour is the number of accumulated function evaluations across all 30 executions. Because dDE is less sensitive to fluctuations in the control parameter settings, the number of function evaluations are generally less compared to MPE. However, both dDE and MPE perform their best with relatively low p_{mv} values.

The Hamming distance between the optima is 10 and the fitness weighting for the global and local optima is $w_{go} = 10$ and $w_{lo} = 1$ respectively. For the VSM problem, we set the within module constraints $d_i = 1$ and the between module constraints $d_e = 0.01$. For dDE on the SBB problem, the population size was set to $NP = 200$ and $p_{mv} = 0.2$, and on the VSM problem the population size was the same with p_{mv} set to 0.1. For MPE, the population size was set to $NP = 700$ and $p_{mv} = 0.005$ on both the SBB and VSM problems.

Figure 4.3 shows the state transition diagrams across 5000 generations of dDE and MPE on the SBB and VSM problems, with the black and white colours translating to variables with the values -1 and 1 respectively. As can be seen from the single-variable changes, the optimisation process spends the initial stages searching for locally optimal configurations to identify module solutions and satisfy the exploration aspect

of higher-scale search. When there are locally optimal configurations in the population, the search is transformed to a higher scale, with the macro-variation operators producing targeted, simultaneous variable changes. Effectively, the neighbourhood of the genotypes in the fitness landscape changes from one where the neighbouring genotypes differ by a single variable to one where they differ by a module solution. But unlike previous work utilising explicit learning mechanisms, searching in the space of module solutions is achieved implicitly, by using the macro-variation operators directly on the population.

The ability of the macro-variation operators to search in the space of module solutions can be shown by initialising the population to locally optimal configurations and setting $p_{mv} = 1$ such that only macro-variation takes place. Figure 4.4 shows the state transition diagrams across 500 generations. As can be seen from the diagrams, DC and OB-Scan are both capable of producing targeted, simultaneous variable changes to exchange module solutions. The new individuals are simply the product of applying DC or OB-Scan directly on the individuals within the population. Because the individuals are locally optimal, DC and OB-Scan can utilise their differences and similarities to search in the space of module solutions. As a result, no explicit learning mechanism is required; the population itself is the model representing the epistatic linkage map of the problem.

There are a few things to note here. First: the algorithms we introduce are not biased towards complementary optima, as can be seen from their behaviour on the SBB problem. Second: both algorithms are capable of solving decomposable (SBB) and nearly-decomposable (VSM) problems. Third: the variables within a module are shown as contiguous blocks (i.e. tightly linked), but that is only done to show a better picture of the behaviour of the algorithms. Fourth: there are no obvious differences between dDE and MPE. Even though MPE solves the SBB problem faster than dDE, the results are only one execution and are not indicative of the performance of the algorithms. However, they are sufficient for showcasing the behaviour of the algorithms and the macro-variation operators. We compare the algorithms in detail in Section 4.4.

4.3 Time Complexity

Here, we examine empirically how the time complexities of dDE, MPE, UCE and random-restart hill-climbers scale with the problem size on the SBB and VSM problems.

We ran 30 independent executions of each problem size on all algorithms and measure the mean function evaluations before the global optimum is found. We only show search instances that meet the convergence probability threshold: the global optimum is found at least 27 out of 30 times (90%). For the SBB problem, we set the globally optimal target string to all 1s and the locally optimal target string to 1s and -1s at every odd and even index respectively. For the VSM problem, we set the within module constraints weighting to $d_i = 1$ and the between module constraints magnitude to $d_e = 0.01$, and randomly initialise the weight matrix in every execution. The range of problem sizes tested was $n = k = [5, 20]$ for both problem. In the SBB problem, the global optimum is a string of all 1s. In the VSM problem, all locally optimal

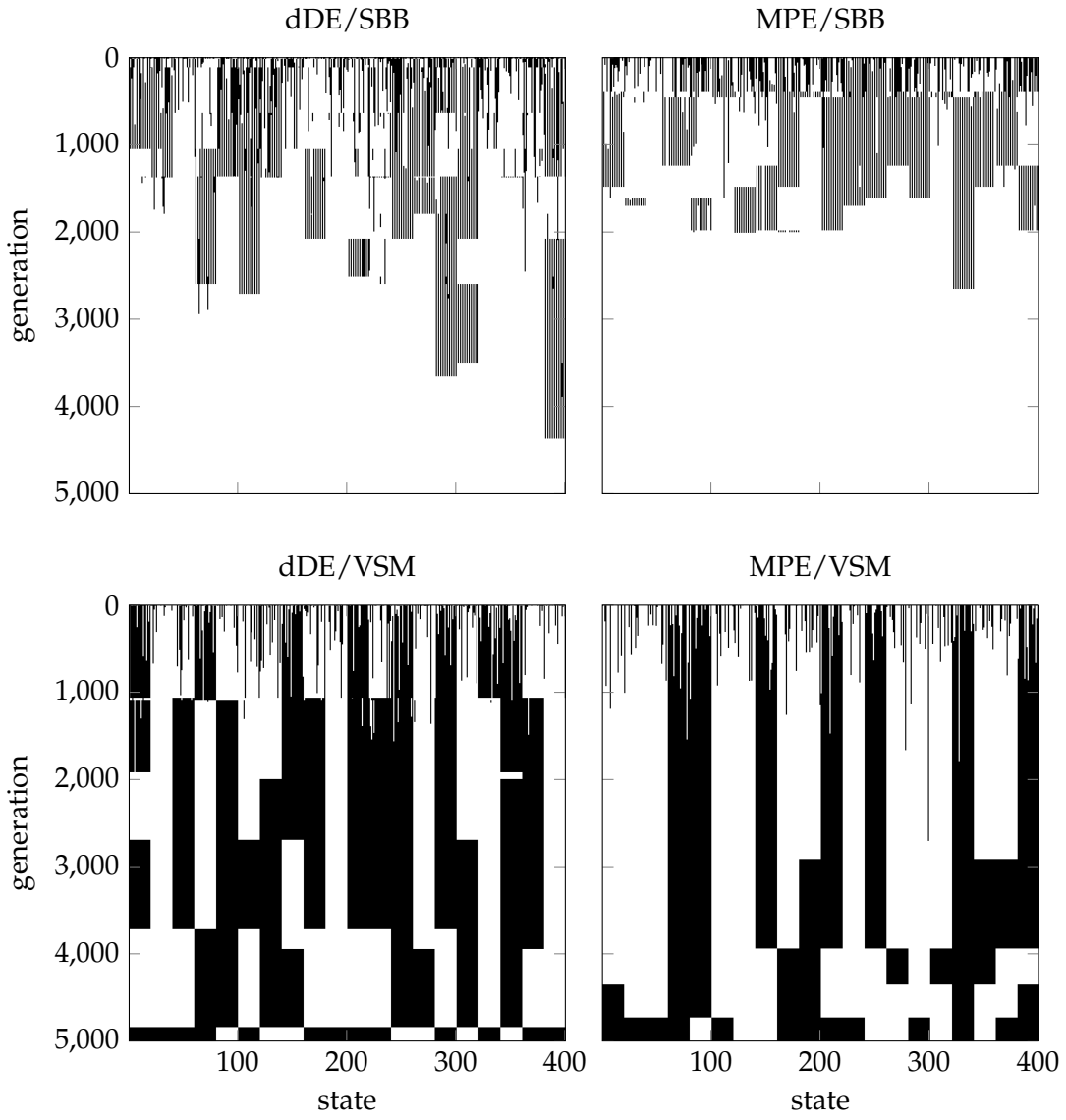

Figure 4.3: State transition diagrams for dDE and MPE on 400-bit SBB and VSM problems, with $n = k = 20$. Both algorithms are capable of exploiting the problem structure and producing targeted, simultaneous variable changes. Initially, both methods perform single-variable changes to direct the search to locally optimal configurations and satisfy the exploration requirement of multi-scale search. Then, when there is information to exploit within the population (i.e., when individuals are locally optimal), DC and OB-Scan produce macro-variation as observed by the substitution of modules in every generation. As shown in the SBB problem, the optima need not be complimentary for the operators to search in the space of modules (i.e., there is no bias for complementary optima).

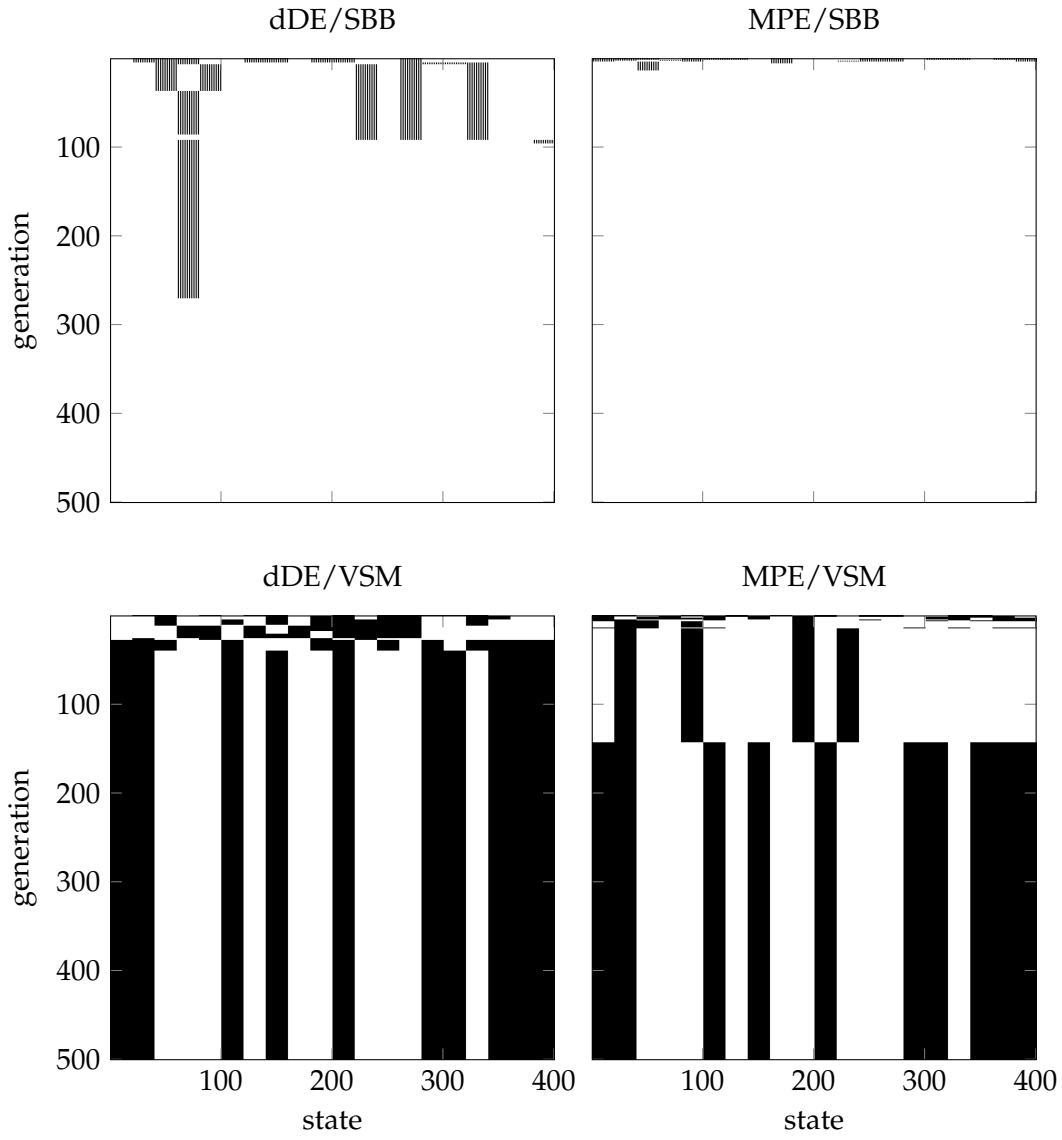

Figure 4.4: State transition diagrams for dDE and MPE on 400-bit SBB and VSM problems, with $n = k = 20$. To show that the macro-variation operators can exploit the problem structure and achieve macro-variation, the population is initialised to locally optimal configurations and p_{mv} fixed to 1 so that only the macro-variation operators are applied: DC for dDE or OB-Scan for MPE. Both DC and OB-Scan achieve macro-variation as evident by the substitution of modules (i.e., targeted, simultaneous variable changes). New individuals are generated by applying the operators directly on the individuals within the population. No explicit learning mechanism is required to build a model of the problem structure.

state configurations were enumerated and evaluated to identify the globally optimal target string for every randomly generated weight matrix. We terminate an algorithm instance prematurely if the time complexity is clearly super-polynomial in N.

dDE, MPE and UCE: the population size starts at $NP = 50$ and is increased by 50 until the convergence probability threshold is met. dDE: $p_{mv} = 0.2$ for the SBB problem and $p_{mv} = 0.005$ for the VSM problem. MPE: $p_{mv} = 0.005$ for both the SBB and VSM problems. UCE: $p_{mv} = 0.005$ for both the SBB and VSM problems. HC: the mutation rate is set to $\mu = 1/N$ and the search process restarts every N consecutive generations without fitness improvement for both the SBB and VSM problems. HC*: the mutation rate is set to the Hamming distance between the optima: $\mu = \lfloor (k/2) \rfloor / N$ for the SBB problem and $\mu = k/N$ for the VSM problem. The search process restarts every $10 \cdot N$ consecutive generations without fitness improvement to allow the mutation operator sufficient time to jump between the optima.

Figure 4.5 shows the time complexities on the SBB problem. The time complexities of UCE, HC, and HC* are clearly super-polynomial in N, as can be seen from the upwards curvature of the lines when plotted on log-log axes. Conversely, dDE and MPE show a distinctive advantage as their ability to search in the space of module solutions reduces the search space significantly. The time complexity for dDE is clearly polynomial in N, as evident from the downwards curvature of the line on log-log axes. Although MPE is also capable of exploiting the modular structure of the problem, its time complexity diverges from that of dDE. However, for the range of problem sizes tested, it is unclear whether the time complexity of MPE is polynomial or super-polynomial in N (see Section 4.4). Note that the difficulty of the problem is relatively relaxed, with the Hamming distance not maximal at $\lfloor (k/2) \rfloor$. This increases the likelihood that uniform crossover retains modules from $\approx 2^{-2k}$ to $\approx 2^{-2\lfloor (k/2) \rfloor}$, and also increases the likelihood that HC* produces the variation required to jump between optima. dDE and MPE are unaffected by the Hamming distance between the optima because they use informed methods and perform targeted variation.

Figure 4.6 shows the time complexities on the VSM problem. The results are not significantly different than those on the SBB problem: UCE, HC and HC* scale super-polynomially in N, as can be seen from the upwards curvature of the line when plotted on log-log axes. The performance of dDE and MPE is clearly superior because of their ability to identify and exploit the modular structure of the problem. Specifically, dDE scales polynomially in N, as evident from the downwards curvature of the line when plotted on log-log axes. However, the time complexities of MPE and dDE clearly diverge on the VSM problem. MPE scales super-polynomially in N, as can be seen from the upwards curvature of the line on the log-log axes plot. Note that the Hamming distance between optima is maximal at k, meaning that probability that uniform crossover retains the linkage information and successfully substitutes a module solution is approximately $\approx 2^{-2k}$. As a result, the super-polynomial time complexity of UCE shows earlier than in the SBB problem, at $n = k = 11$.

Overall, the conventional methods (UCE, HC and HC*) scale super-polynomially in N on both the SBB and VSM problems. In fact, due to their lack of sophisticated search mechanisms, their performance is largely unaffected by the greater difficulty of the

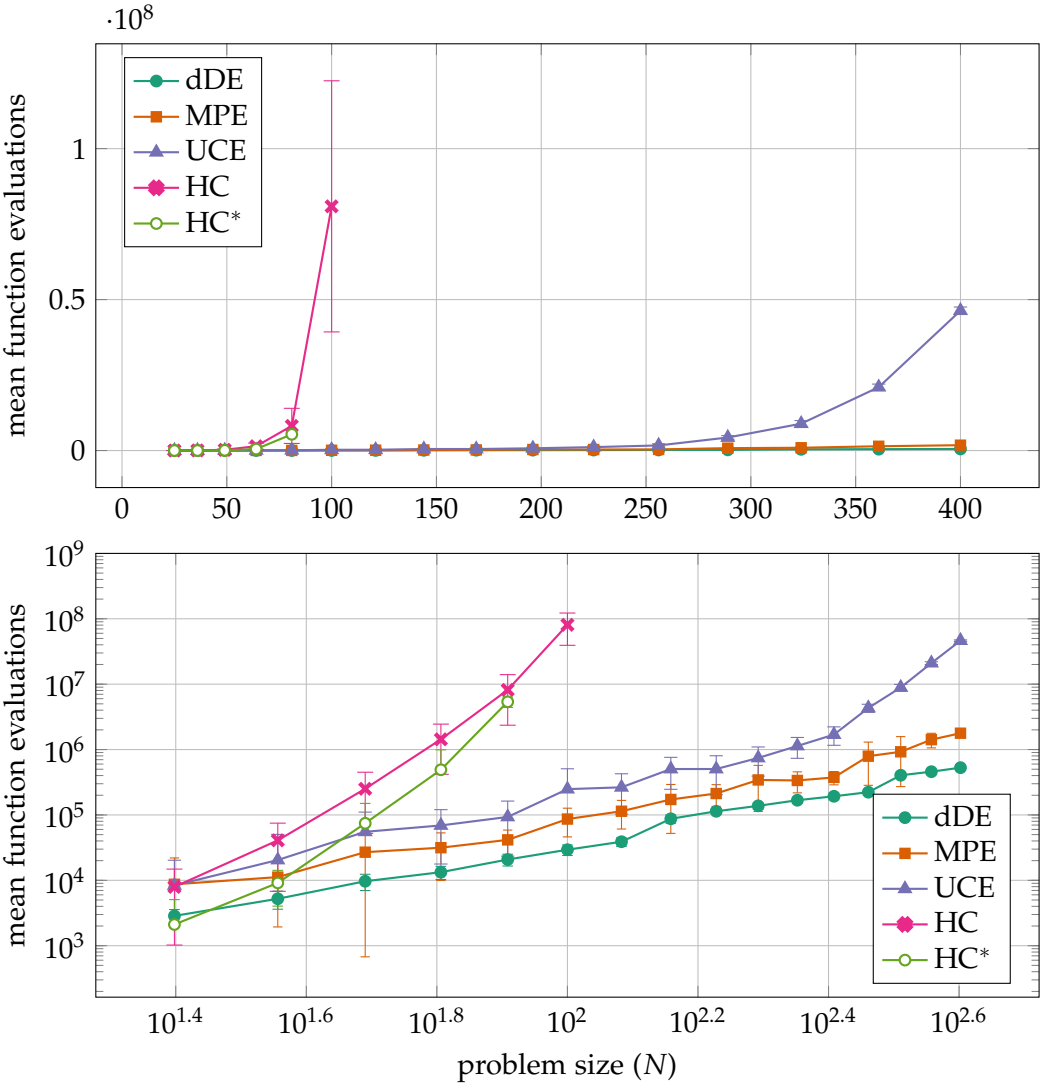

Figure 4.5: Mean measured function evaluations for dDE, MPE, UCE, HC, and HC* on 30 independent executions of the SBB problem, with the number of modules and the length of each module scaling as a factor of the overall problem size: $N = nk$; $n = k = \sqrt{N}$. The results show only the search instances that found the global optimum at least 27 times out of the 30 executions (90%). Top: the empirical data are shown on linear axes. Bottom: the empirical data are shown on log-log axes. The expected running time for UCE, HC and HC* is super-polynomial in N, as evident from the upwards curvature of the line on log-log axes. Conversely, because dDE can exploit the modular structure of the problem and search in the space of module solutions its expected running time is polynomial in N. The time complexities of dDE and MPE clearly diverge, but it is unclear whether MPE scales polynomially or super-polynomially in N (see Section 4.4).

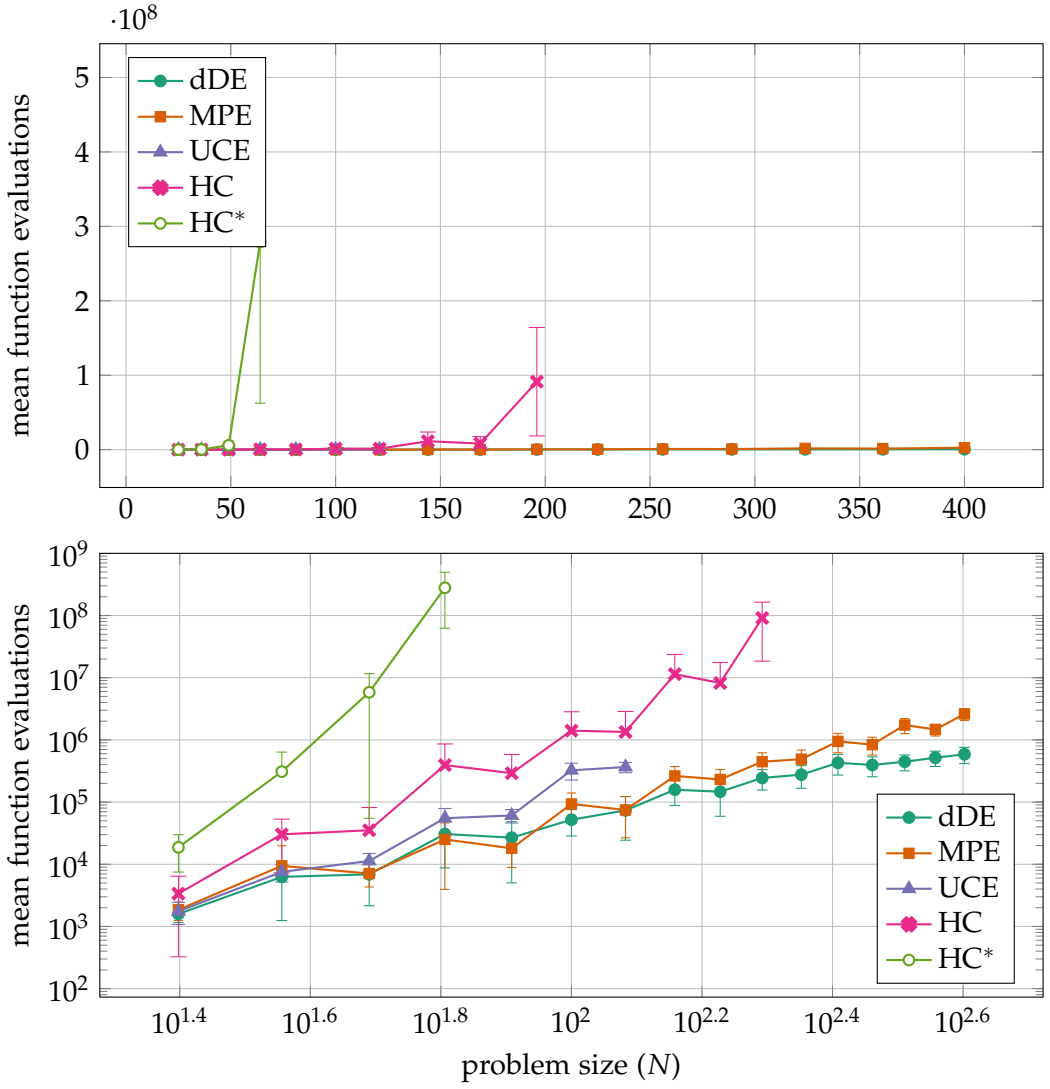

Figure 4.6: Mean measured function evaluations for dDE, MPE, UCE, HC, and HC* on 30 independent executions of the VSM problem, with the number of modules and the length of each module scaling as a factor of the overall problem size: $N = nk$; $n = k = \sqrt{N}$. The results show only the search instances that found the global optimum at least 27 times out of the 30 executions (90%). Top: the empirical data are shown on linear axes. Bottom: the empirical data are shown on log-log axes. Due to their inability to exploit the underlying modular structure of the problem, UCE, HC and HC* scale super-polynomially in N as can be seen from the upwards curvature of the line when the data is plotted on log-log axes. Conversely, the expected running time for dDE is polynomial in N, as the line plotted on log-log axes curves downwards. Despite its ability to exploit modular structure, the time complexity for MPE is super-polynomial in N as can be seen from the upwards curvature of the line on log-log axes.

nearly-decomposable VSM problem. However, the increase in Hamming distance between the optima ($\lfloor (k/2) \rfloor$ and k for the SBB and VSM problems respectively) reduces the likelihood that HC* can produce the necessary variation to jump between optima and the likelihood that uniform crossover can retain the linkage information. As a result, UCE was incapable of solving all instances of the VSM problem. Also HC* performs worse than HC on the VSM problem while the opposite was true on the SBB problem. Conversely, there is no significant difference between the performance of dDE on the SBB and VSM problems: the time complexity is polynomial in N for both. Lastly, even though the time complexity of MPE on the VSM problem is super-polynomial in N, it is unclear whether that is the case for the SBB problem as well. We investigate this in detail in the following section.

4.4 dDE vs MPE

In this section, we look at the time complexities of dDE and MPE exclusively and also investigate what causes the qualitative difference between the two algorithms by comparing the two macro-variation operators: differential crossover (DC) for dDE and occurrence-based scanning (OB-Scan) for MPE.

In Figure 4.7, we show the time complexities of dDE and MPE on larger instances of the SBB problem ($n = k = [5, 30]$) plotted on log-log axes. While the time complexity of MPE was unclear in the earlier experiment, it is now obvious that MPE scales super-polynomially in N, as evident from the upwards curvature of the line. As shown earlier, dDE scales polynomially in N, with the line curving downwards. For completeness, Figure 4.8 shows the results on the VSM problem. The results are the same: dDE scales polynomially in N and MPE super-polynomially in N. However, the cause for the qualitative difference between the two algorithms is not clear.

Figure 4.9 shows the recorded population sizes dDE and MPE required to meet the convergence probability threshold: finding the global optimum at least 27 out of 30 times (90%). There is a significant difference in the population sizes the two methods require. Specifically, dDE solves the largest problems (SBB: $n = k = 30$; VSM: $n = k = 20$) reliably with a population size of $NP = 150$. Conversely, for MPE to reliably solve the problems, it requires population sizes of $NP = 2900$ for the $n = k = 23$ SBB problem and $NP = 800$ for the $n = k = 20$ VSM problem. More importantly, the population sizes scale significantly different. The population size for dDE scales at a rate of $\approx 5n$ and for MPE at a rate of $\approx 1.4^n$. This significant difference in the population sizes required is the cause of the qualitative difference between the two algorithms.

To identify differences in the population dynamics of the two algorithms, we initialise the population to locally optimal configurations, set $p_{mv} = 1$ such that only macro-variation is performed, and measure the number of module solution occurrences in module 1 of an $n = k = 20$ VSM problem across 100 generations (Figure 4.10). Although the VSM problem has complementary optima and the globally optimal module solution depends on the genetic context, MPE causes the population to converge to a specific module solution. Conversely, dDE maintains a roughly equal proportion of

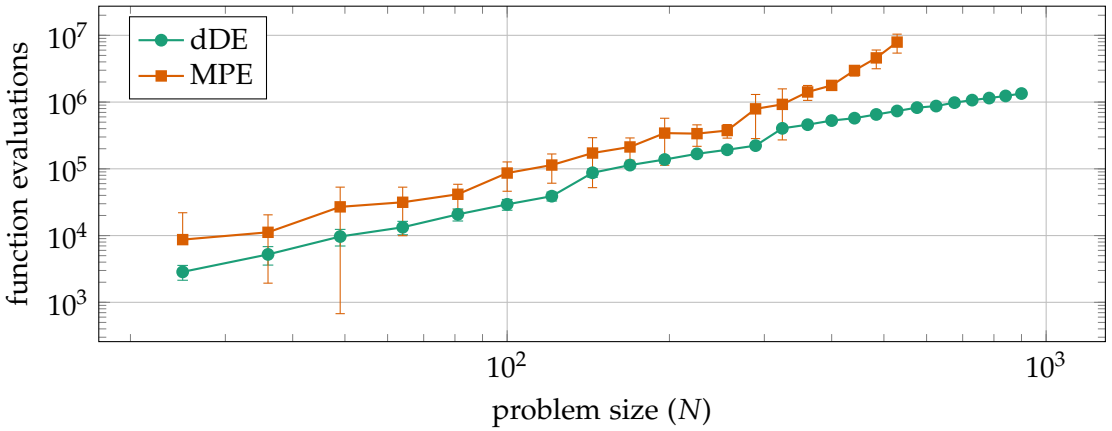

Figure 4.7: Mean measured function evaluations for dDE and MPE for 30 independent executions on the SBB problem, with the number of modules and the length of each module scaling as a factor of the overall problem size: $N = nk$; $n = k = \sqrt{N}$. The results show only the search instances that found the global optimum at least 27 times out of the 30 executions (90%). The empirical data are shown on log-log axes. Despite their similarities and their ability to identify and exploit modular structure, the time complexities for the two methods diverge. The time complexity for MPE grows super-polynomially in N as can be seen from the upward curvature and slope of the line when plotted on log-log axes. Conversely, the dDE time complexity grows polynomially in N as evident from the downward curvature of the line.

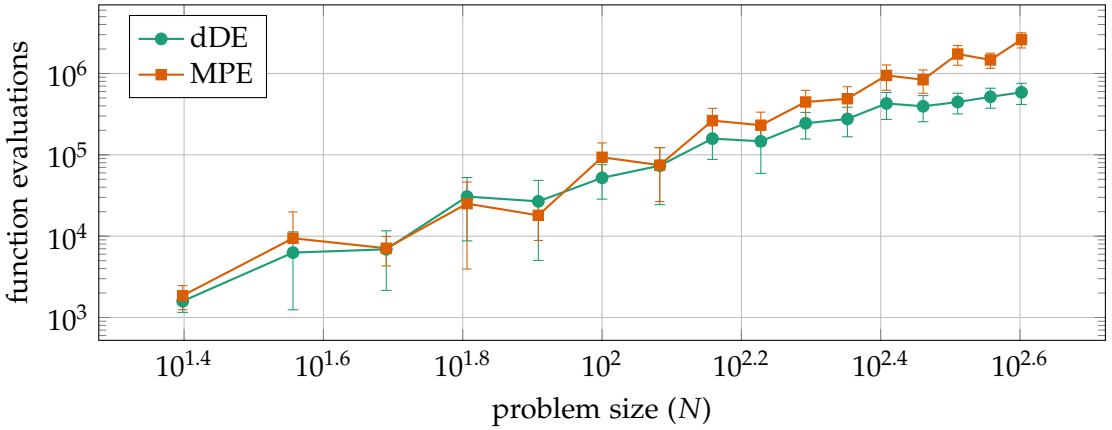

Figure 4.8: Mean measured function evaluations for dDE and MPE for 30 independent executions on the VSM problem, with the number of modules and the length of each module scaling as a factor of the overall problem size: $N = nk$; $n = k = \sqrt{N}$. The results show only the search instances that found the global optimum at least 27 times out of the 30 executions (90%). The empirical data are shown on log-log axes. Due to the increased difficulty of the VSM problem compared to the SBB problem, the time complexities for the two methods also diverge. The time complexities for dDE and MPE grow polynomially and super-polynomially in N respectively, as can be seen from the respective lines when the data are plotted on log-log axes.

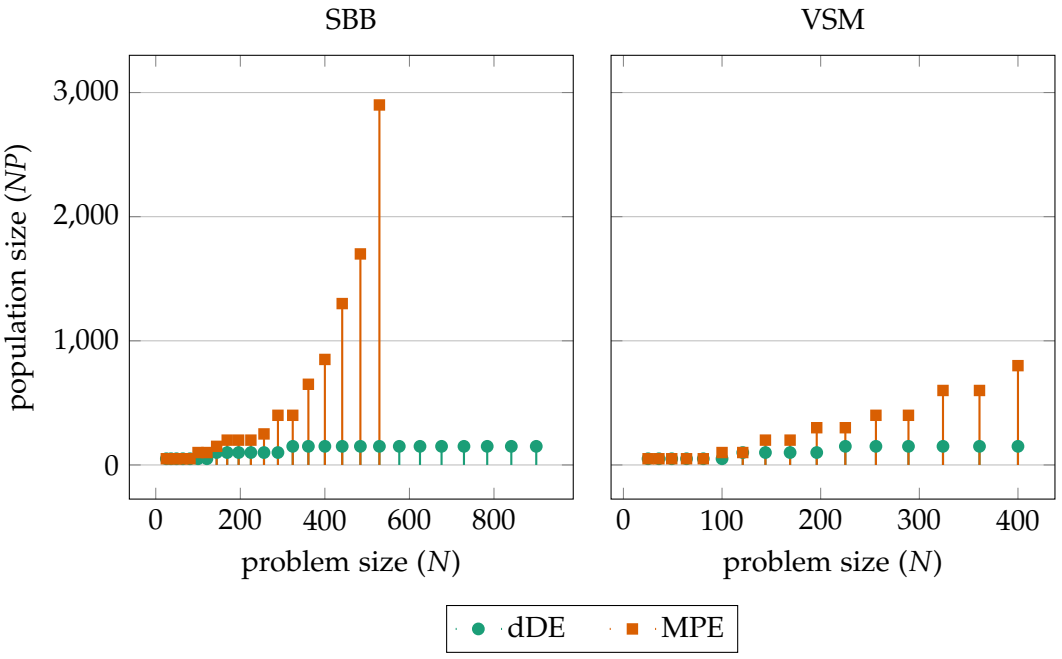

Figure 4.9: The recorded population size required for dDE and MPC to solve the SBB and VSM problems as the size of the problem increases. dDE solves the problem reliably with very low population sizes: $NP = 150$ for an SBB problem of $n = k = 30$ and a VSM problem of $n = k = 20$. Conversely, MPE requires very large population sizes to reliably solve the problems: $NP = 2900$ for an SBB problem with $n = k = 23$, and $NP = 800$ for a VSM problem with $n = k = 20$. The significant difference in the required population sizes is what ultimately causes the qualitative difference between the two algorithms.

each module solution. From this we can see that the population dynamics of the two algorithms are significantly different, with MPE converging to one module solution and dDE maintaining module solutions. These difference can only be attributed to the macro-variation operators.

We show the differences between the DC and OB-Scan macro-variation operators by: (1) initialising the population to one of four configurations: 1111, 0000, 1010 or 0101; (2) setting $p_{mv} = 1$ such that only macro-variation occurs; (3) removing the selection stage, such that the ith trial vector always replaces the ith target vector; and (4) measuring the number of module solution occurrences across 100 generations. Figure 4.11 shows the number of occurrences of each configuration across each generation when the population is initialised uniformly (top row) or non-uniformly (bottom row). Whether the population is uniformly or non-uniformly initialised, OB-Scan converges to one configuration although there is no selective pressure to direct the search to that specific configuration—specifically, it converges to the configuration that is in the majority. Conversely, DC retains roughly an equal proportion of all configurations, regardless of the initial proportion of individuals within the population. OB-Scan is clearly biased towards convergence, whereas DC reaches an equilibrium when the population is maximally different (i.e., the population includes an equal proportion of unique configurations).

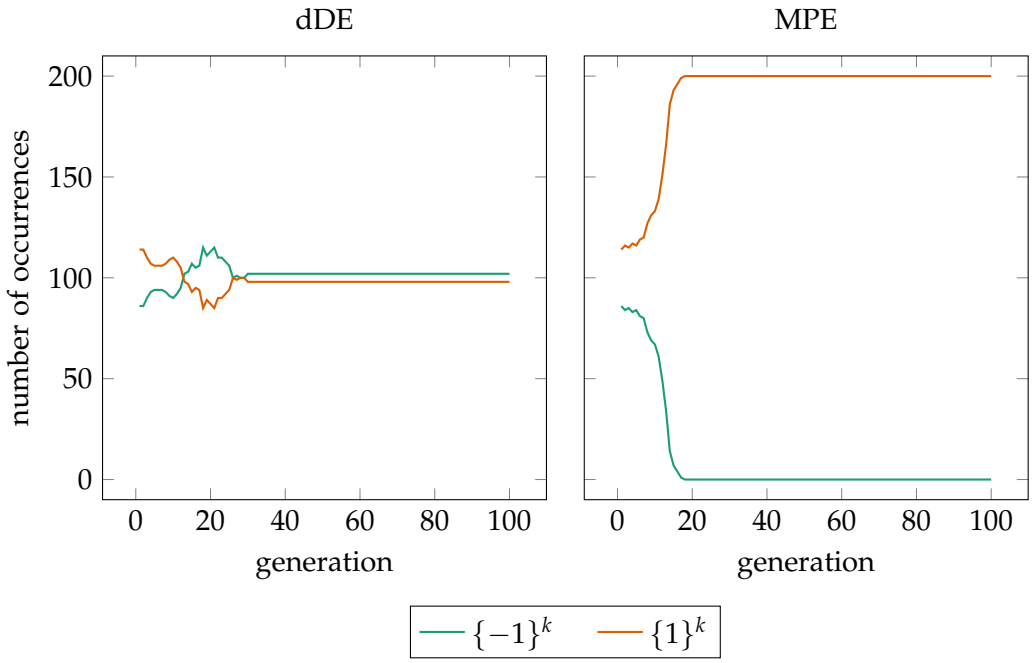

Figure 4.10: The distribution of configurations in module 1 of an $n = k = 20$ VSM problem when using the dDE or MPE. The population is uniformly initialised to locally optimal configurations (i.e., in the language $\{-1^k, 1^k\}^n$.). Even though the VSM problem has complementary optima, in MPE, all individuals in the population converge to one of the locally optimal configurations. Conversely, dDE retains an approximately equal proportion of each locally optimal configuration.

As a result, selection has to work against the OB-Scan convergence bias. Assuming that there is enough selective pressure to direct the search to the globally optimal configuration, the convergence bias remains an issue when we consider the genetic hitch-hiking effect. That is, it is likely that the population will converge to a sub-optimal module configuration when an individual is selected because of a fitness improvement introduced by variation somewhere else on the genome. This is true of DC as well, but the effect is not as pronounced because the operator is not biased towards convergence. Evidently, the larger population size requirement of MPE is attributed to the convergence bias of the macro-variation operator. As the number of modules increases, the likelihood of genetic hitch-hiking increases with it. But because the MPE macro-variation operator is biased towards convergence, the population converges to hitch-hiking, sub-optimal module solutions at a much quicker rate than in the case of dDE. This causes a race condition between the rate at which sub-optimal configurations are propagated because of the hitch-hiking effect and the size of the population required to prevent convergence.

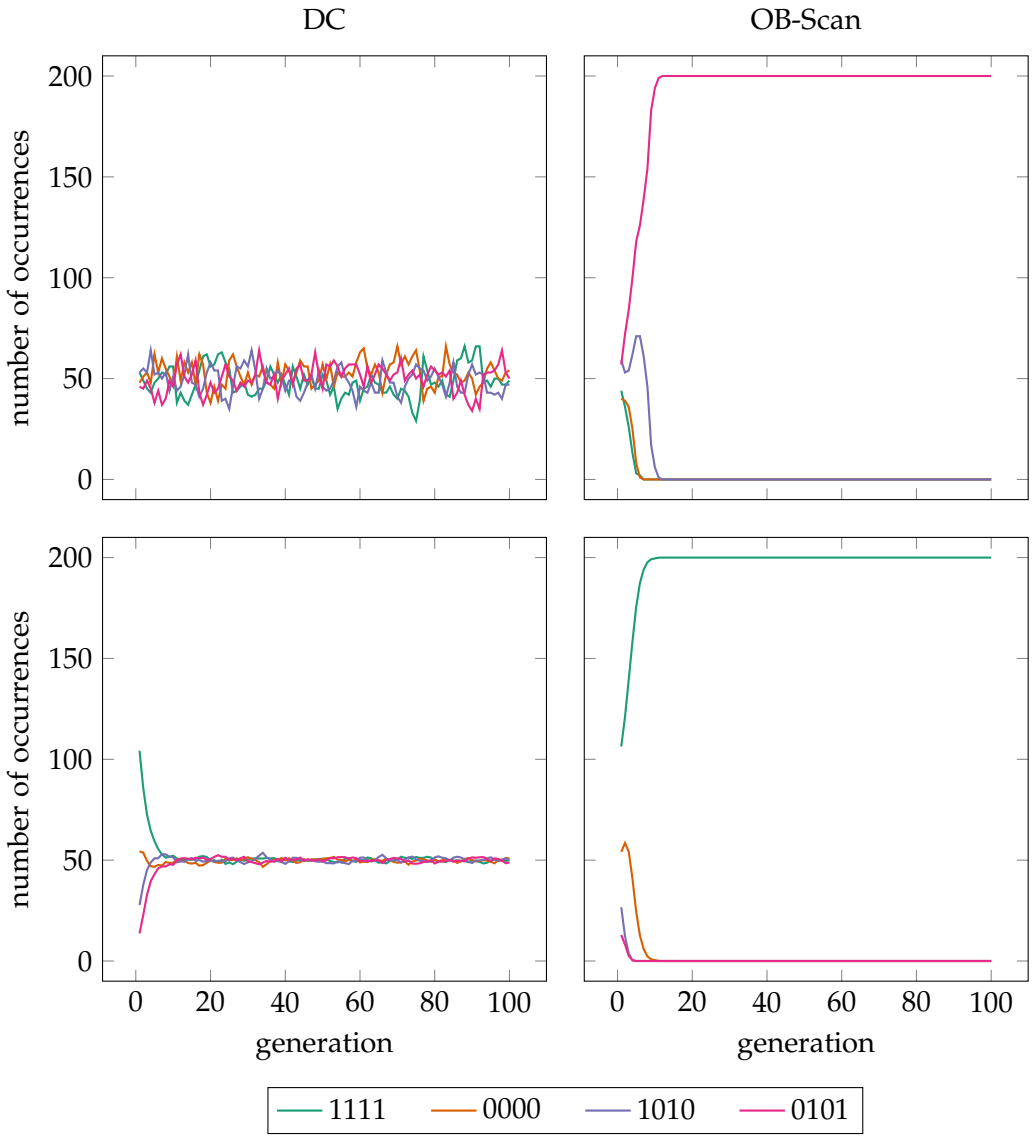

Figure 4.11: The distribution of configurations when using the DC and OB-Scan operators. The population is initialised uniformly (top row) or non-uniformly (bottom row) to four configurations (1111, 0000, 1010, 0101). Because the selection stage is removed (i.e., the ith trial vector always replaces the ith target vector), the configurations are chosen such that any crossover event between three of them yields one of the configurations. Even in the absence of selective pressure to direct the search, OB-Scan is clearly biased towards converging on one specific configuration, both when the population is uniformly and non-uniformly initialised. This means that selection has to work against the convergence bias of OB-Scan, with the effect being more significant when considering the genetic hitch-hiking effect. Conversely, DC maintains roughly an equal proportion of each configuration, whether the population was initialised uniformly or non-uniformly.

Chapter 5

Discussion

In this chapter, we discuss multiple aspects of our work, such as why the problems we used for testing are difficult, what it means for dDE and MPE to "learn", the limitations of our work and future work to address them.

5.1 Results Summary

From the algorithms tested, the only algorithm scaling polynomially with the problem size was dDE. Although MPE utilises a similar macro-variation operator and has been shown to exploit the modular structure of the problem, its time complexity is super-polynomial in the size of the problem. This qualitative difference is caused by the population size requirement of MPE, which is the result of an inherent bias of the OB-Scan operator to rapidly converge on specific configurations. Additionally, we have seen that conventional variation operators such as bit-flip mutation or uniform crossover are incapable of achieving macro-variation and thus scale super-polynomially with the problem size.

5.2 On the Difficulty of the SBB and VSM Problems

As we have seen from the results in Chapter 4, conventional optimisation approaches quickly become incapable of solving the SBB and VSM problems as their time complexities scale super-polynomially with the problem size, N. A random-restart hill-climber with a mutation rate of $\mu = 1/N$ is only able to find the global optimum when starting in all the right basins of attraction. Since the basins of attraction scale exponentially with the number of modules and module solutions in the problem at z^n, the number of restarts required grows exponentially with it (in our problems, $z = 2$, but n scales with the overall problem size: $n = \sqrt{N}$). What if a hill-climber is provided with the number of variables that need to be changed simultaneously? Given a hill-climber with a mutation rate of $\mu = k/N$, where k is the Hamming distance between the module solutions, the probability of choosing the right k variables to change decreases super-polynomially with the number of k-bit combinations: $n/\binom{N}{k}$. For uniform crossover to

produce an individual different from the two parents being recombined, the parents must differ by at least two modules. The number of potential children is 2^{2k}, of which only 2 are locally optimal, meaning that the probability of producing any of the locally optimal children decreases exponentially with the number of disagreeing loci (i.e., Hamming distance between optima). This shows why the SBB and VSM problems are pathologically difficult for conventional methods that are incapable of identifying and exploiting the underlying modular structure of the problem, and also agrees with the results of our numerical simulations.

5.3 Learning Correlations

Central to the successful scaling of the search is the process of exploring multiple areas of the fitness landscape to identify module solutions. To properly identify all module solutions without mistake (i.e., no false positive or negative correlations), a specific number of locally optimal configurations has to be sampled. In many of the current approaches, this number is controlled by the number of hill-climbing processes that are ran or by a learning rate control parameter. In the case of dDE and MPE, we do not explicitly set such a parameter because learning is not explicit. However, it is not difficult to see how the combination of the population size, NP, and probability of performing macro-variation, p_{mv}, control parameters is very similar to a learning rate.

Consider the following. During the hill-climbing process, alleles at specific loci or module solutions that confer high fitness will be selected. After these alleles or module solutions are discovered, the rate at which macro-variation is performed determines how quickly they are copied onto other individuals. To prevent early convergence, the population size must be sufficiently large with regard to the rate at which macro-variation is performed. That is, if the population size is too small or macro-variation is performed too often, diversity is lost quickly, meaning that the differences or similarities between the individuals in the population do not reflect the structure of the problem. This issue is very similar to that of tuning the learning rate value: setting it too high results in learned correlations that are not representative of the problem structure.

5.4 Population Size

As we discussed in the previous section, the population size is very important for preventing high-fitness subsets of loci from taking over the population before sufficient exploration has taken place. Assuming that every individual in the population is unique, the size of the population also determines how much control the macro-variation operators have over the changes produced. That is, depending on the differences and similarities between two individuals, a number of modules are super-imposed onto another individual. So given that individual A is one module solution away from the global optimum, two other individuals whose only difference is the module solution individual A needs must exist within the population. But how large does the population need to be to achieve this? Even though we do not provide a mathematical

derivation, we can see from our results that for dDE to reliably solve the problem, the required population is $\approx 5n$, where n is the number of modules. This means that the population size scales polynomially with n. Conversely, MPE requires significantly larger population sizes to solve reliably. From our experiments, the minimum population size MPE requires to solve the SBB and VSM problems scales super-polynomially in n, at $\approx 1.4^n$.

5.5 More Optima Per Module

In the problems we tested, each module had only $z = 2$ module solutions. Because the macro-variation operators in dDE and MPE are crossover operators, when more than 2 optima exist per module, $z > 2$, a crossover event between three module solutions is not guaranteed to produce another module solution. Despite this fact, both dDE and MPE are capable of solving such problems for reasons we explain below.

Assuming that a macro-variation event produced a non-optimal module configuration, there are two possible outcomes: (1) the selection stage discards the child because it is not as fit; or (2) the child replaces the parent because other changes on the genome provide an overall fitness improvement (i.e., genetic hitch-hiking). In the first case, there is no issue because the child is simply discarded. As the generations pass, selective pressure will direct the search and reduce the number of module solutions within the population, which will result in meaningful macro-variation. In the second case, the non-optimal module configuration can potentially be replaced by another macro-variation operation. This, however, is not guaranteed for the same reason the non-optimal module configuration was produced: crossing over three distinct configurations does not necessarily yield a module solution. But because dDE and MPE perform bit-flip or macro-variation probabilistically, the bit-flip operator will fix the non-optimal module configuration. The secondary role of the bit-flip operator is to fix the non-optimal module configurations that can be potentially produced by the macro-variation operators. Consequently, although DC and OB-Scan are not guaranteed to produce module solutions when $z > 2$, dDE and MPE are capable of solving problems with more than two optima per module.

5.6 Switching Between Hill-Climbing and Macro-Variation

In the versions of dDE and MPE we introduce, we switch between bit-flip and macro-variation probabilistically, with the frequency of each operator determined by the probability of performing macro-variation control parameter, p_{mv}.

Let us first consider what the issues with the current approach are. Given any modular optimisation problem, a search mechanism requires sufficient time for the discovery of module solutions. Because we are switching between bit-flip and macro-variation probabilistically, "learning" issues can arise when the control parameters are not properly tuned (see Section 5.3). These problems can be countered by taking an *ad-hoc* approach and setting p_{mv} to a very low value and the population size to a relatively

large value. But in doing so, when the population is locally optimal, the large majority of generations are spent performing deleterious mutations only for selection to discard the individuals. Even though this is not a significant issue, it prevents dDE and MPE from performing optimally. Nevertheless, this approach allows for the successful optimisation of problems with more than two module solutions per module for reasons discussed in Section 5.5. What are the alternatives to this, however?

The simplest alternative would be to use a hard threshold that switches between bit-flip and macro-variation. Consider a scenario where bit-flip is performed for t generations, after which macro-variation is performed until the algorithm terminates.[1] There are two issues associated with this approach. First, how do we determine the value for t? To properly set t, we need to have *a priori* information about the size of the problem, so that we switch to macro-variation only after all individuals settle on local optima. More importantly, however, using a hard threshold can potentially disrupt the optimisation process when modules include more than two module solutions, as explained in Section 5.5. This would assume *a priori* knowledge about the number of module solutions per module: that they are limited to $z = 2$. Since we are not interested in cases where *a priori* information is available, setting a hard threshold for switching between the two modes of operation is not a better option.

An alternative to setting a hard threshold would be to start with a low value for p_{mv} and use some form of simulated annealing [49] to increase it by a factor x in every generation so that macro-variation occurs more frequently. Similar to the hard threshold case, the first issue is how to determine what x should be. If it is a factor of the generation limit and that is set incorrectly, macro-variation would occur too often too early, which would result in insufficient exploration. This could also cause issues when there are more than two module solutions per module. If x is set incorrectly, macro-variation will potentially produce non-optimal module configurations at a faster rate than bit-flip can fix them (as it occurs less frequently with each generation). This approach suffers from the same problems as the use of a hard threshold, but it is more efficient when we assume *a priori* knowledge of only two module solutions per module.

Another alternative would be to take a similar approach to that of Qin *et al.* in their *self-adaptive differential evolution* (SaDE) [50] algorithm. SaDE starts with a number of differential evolution variants (DE/rand/1/bin, DE/best/1/bin etc.) whose execution probabilities are initially the same. As the optimisation process progresses, the frequency at which each variant is executed changes as a factor of how well it performs on the problem. In our algorithms, the implementation would be slightly different. Since we know that exploration must come before exploitation, bit-flip and macro-variation should not be performed with equal probability initially; instead, bit-flip should start with a much greater proportion of execution time (like we do for p_{mv}). With every generation, selection determines the effectiveness of the two operators and adjusts their execution probability accordingly. As a fail-safe, a limit should be put in place such that none of the operators takes over with a probability of 1. Clearly, this approach does not suffer from the problems the hard threshold and simulated annealing approaches do. Bit-flip still occurs when individuals need to be fixed and its efficiency

[1] By using a hard threshold, search scales explicitly rather than implicitly, but we consider this scenario for completeness.

does not depend on the how early we switch to macro-variation or how quickly macro-variation takes over. In fact, this approach would also prevent the wastefulness of the probabilistic switch when bit-flip produces only deleterious mutations.

5.7 Limitations & Future Work

Although we have illustrated how differential evolution principles and multi-parent crossover operators can successfully optimise random-linkage, modular problems, there is still a number of areas which we did not cover sufficiently. Below, we discuss the limitations of our work along with future work that can address them.

Firstly, we do not provide analytical time complexities. Although we used empirical results to demonstrate the time complexities of the algorithms, our results do not support the general case, as the negative results of the conventional algorithms could be attributed to badly tuned control parameters. It is therefore best to show whether a method will fail using analytic methods. Similarly, our population size experiment was sufficient to show what causes the qualitative difference between dDE and MPE, but we do not use analytical methods to provide lower bounds on the population size control parameter. Therefore, we must provide analytical results on dDE and MPE.

The second issue would be to address the *ad-hoc* approach used to switch between bit-flip and macro-variation. As we discussed in Section 5.6, switching between the two operators probabilistically is not the most efficient approach, with issues of early misconvergence or spending the majority of generations performing the wrong operation. An approach similar to that of SaDE [50] sounds ideal, without sacrificing any of the capabilities provided by the probabilistic switch method. Hence, the development of self-adaptive dDE or MPE could improve the algorithms even further.

By testing dDE and MPE on the SBB and VSM problems, we were able to show that the algorithms are not biased towards complementary optima and that their application is not limited to separable problems. However, we have not ran any experiments on hierarchical problems, such as the hierarchical if-and-only-if (H-IFF) problem [19]. Hierarchical problems include more than two levels that need to be optimised sequentially. Since our approach scales search and produces macro-variation implicitly, we need to show whether our hypotheses hold when more than two levels are involved. Intuitively, there is no reason to believe that it would not. Differences and similarities between level-2 locally optimal configurations will again exist and will be exploited by the macro-variation operators to generate level-3 modules. However, one issue could be misconvergence attributed to low population sizes. If there is not enough diversity at the second hierarchical level, the macro-variation operators will not be able to produce the changes required to move to the next hierarchical level. But these are mere assertions. Testing on hierarchical problems would allow us to observe whether the implicit scaling hypothesis holds when the problem includes more than two hierarchical levels.

Lastly, in our work we provide no comparison with other linkage-exploiting algorithms. What can dDE or MPE do better? Yu *et al.* [43] show that the population size EDAs require to build a model reflecting the problem structure is exponential in the length

of the modules. In the problems we use, the length of a module scales as a factor of overall problem size ($k = \sqrt{N}$), meaning that the population size requirement scales exponentially with length of the module. This is not an issue for dDE and MPE. However, rHN-G [10], for example, does not suffer from this issue either. The difference between our approach lies in how the modular structure is discovered and exploited. While other algorithms use explicit learning mechanisms to represent correlations between variables, dDE and MPE achieve both the learning and the scaling of the search implicitly. But we have not yet shown a problem our approach can solve that other algorithms that do not suffer from the population requirement cannot.

Chapter 6

Conclusions

In this thesis, we presented two multi-parent crossover operators capable of identifying and exploiting modular structure when applied directly to a population of locally optimal individuals. We used the operators in the context of differential evolution and developed an algorithmic model that is capable of solving random-linkage modular problems, without top-down or problem-specific knowledge of how to decompose the problem. The first algorithm, which we call discrete differential evolution (dDE), utilises differences between individuals in the population to produce macro-variation and scale search. Conversely, the second algorithm, which we call multi-parent evolution (MPE), utilises similarities between individuals to achieve macro-variation.

Overall, the application of both algorithms was successful. The results showed that the operators were capable of producing targeted, simultaneous variable changes and that the overall algorithmic models were able to satisfy both the exploration and exploitation requirements of multi-scale search. Importantly, the scaling of the search and macro-variation are achieved implicitly, as neither of the two algorithms utilises an explicit separation of the exploration and exploitation requirements or explicit learning mechanisms that build a model representing the problem structure. Despite their similarities, we found that dDE is the best-performing of the two as it scales polynomially with the problem size, while MPE scales super-polynomially. We showed that the qualitative difference is attributed to different population dynamics caused by the different macro-variation operators. In the *Discussion* chapter we present a number of areas which were not covered sufficiently in our work. However, the results show great promise and dDE is clearly capable of competing with other similar algorithms.

References

[1] R. Watson, "Analysis of recombinative algorithms on a non-separable building-block problem," *Foundations of genetic algorithms*, vol. 6, pp. 69–89, 2001.

[2] R. Watson, "A simple two-module problem to exemplify building-block assembly under crossover," in *Parallel Problem Solving from Nature-PPSN VIII*, pp. 161–171, Springer, 2004.

[3] A. Prügel-Bennett, "When a genetic algorithm outperforms hill-climbing," *Theoretical Computer Science*, vol. 320, no. 1, pp. 135–153, 2004.

[4] T. Jansen and I. Wegener, "Real royal road functions–where crossover provably is essential," *Discrete applied mathematics*, vol. 149, no. 1, pp. 111–125, 2005.

[5] R. A. Watson and T. Jansen, "A Building-Block Royal Road Where Crossover is Provably Essential," *Proceedings of the 9th annual conference on Genetic and evolutionary computation GECCO 07*, p. 1452, 2007.

[6] D. Goldberg, "Genetic algorithms in search, optimization, and machine learning," 1989.

[7] S. Forrest and M. Mitchell, "Relative building-block fitness and the building-block hypothesis," *Ann Arbor*, vol. 1001, p. 48109, 1993.

[8] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "Linkage problem, distribution estimation, and bayesian networks," *Evolutionary computation*, vol. 8, no. 3, pp. 311–340, 2000.

[9] T.-L. Yu and D. E. Goldberg, "Conquering hierarchical difficulty by explicit chunking: substructural chromosome compression," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 1385–1392, ACM, 2006.

[10] R. A. Watson, R. Mills, and C. Buckley, "Transformations in the scale of behavior and the global optimization of constraints in adaptive networks," *Adaptive Behavior*, vol. 19, no. 4, pp. 227–249, 2011.

[11] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *ICSI, USA*, 1995.

[12] R. Storn and K. Price, "Minimizing the real functions of the icec 1996 contest by differential evolution," *Proc. IEEE Int. Conf. Evol. Comput.*, pp. 842–844, 1996.

[13] R. Storn, "On the usage of differential evolution for function optimization," in *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, pp. 519–523, 1996.

[14] I. D. Falco, A. D. Cioppa, and A. Tarantino, "Automatic classification of handsegmented image parts with differential evolution," *EvoWorkshops 2006*, pp. 403–414, 2006.

[15] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete differential evolution algorithm for the permutation flowshop scheduling problem," *Computers & Industrial Engineering*, vol. 55, no. 4, pp. 795–816, 2008.

[16] L. Wang, Q.-K. Pan, P. Suganthan, W.-H. Wang, and Y.-M. Wang, "A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems," *Computers & Operations Research*, vol. 37, no. 3, pp. 509–520, 2010.

[17] J. G. Sauer and L. Coelho, "Discrete differential evolution with local search to solve the traveling salesman problem: Fundamentals and case studies," in *Cybernetic Intelligent Systems, 2008. CIS 2008. 7th IEEE International Conference on*, pp. 1–6, IEEE, 2008.

[18] M. Randall, "Differential evolution for a constrained combinatorial optimisation problem," *International journal of metaheuristics*, vol. 1, no. 4, pp. 279–297, 2011.

[19] R. A. Watson, G. S. Hornby, and J. B. Pollack, "Modeling building-block interdependency," in *Parallel Problem Solving from Nature—PPSN V*, pp. 97–106, Springer, 1998.

[20] A. E. Eiben, P.-E. Raue, and Z. Ruttkay, "Genetic algorithms with multi-parent recombination," in *Parallel Problem Solving from Nature—PPSN III*, pp. 78–87, Springer, 1994.

[21] V. Feoktistov and S. Janaqi, "Generalization of the strategies in differential evolution," *Proc. 18th IPDPS*, p. 165a, 2004.

[22] K. Price, R. Storn, and J. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*. U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2005.

[23] J. Rönkkönen, S. Kukkonen, and K. Price, "Real-parameter optimization with differential evolution," *IEEE Congress on Evolutionary Computation*, pp. 506–513, 2005.

[24] D. Zaharie, "A comparative analysis of crossover variants in differential evolution," *Proceedings of the International Multiconference on Computer Science and Information Technology*, pp. 171–181, 2007.

[25] R. Storn, *Advances in Differential Evolution*. U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2008.

[26] S. Das, A. Abraham, U. Chakraborty, and A. Konar, "Differential evolution using a neighborhood based mutation operator," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 526–553, 2009.

[27] G. Jeyakumar and C. Shanmugavelayutham, "Convergence analysis of differential evolution variants on unconstrained global optimization functions," *International Journal of Artificial Intelligence and Applications (IJAIA)*, vol. 2, no. 2, 2011.

[28] S. B. Carroll, "Chance and necessity: the evolution of morphological complexity and diversity," *Nature*, vol. 409, no. 6823, pp. 1102–1109, 2001.

[29] A. Hintze and C. Adami, "Evolution of complex modular biological networks," *PLoS computational biology*, vol. 4, no. 2, p. e23, 2008.

[30] G. P. Wagner, M. Pavlicev, and J. M. Cheverud, "The road to modularity," *Nature Reviews Genetics*, vol. 8, no. 12, pp. 921–931, 2007.

[31] J. H. Holland, *Adaptation in Natural and Artificial Systems*, vol. Ann Arbor. 1975.

[32] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and ga performance," in *Proceedings of the first european conference on artificial life*, pp. 245–254, Cambridge: The MIT Press, 1992.

[33] R. A. Watson and J. B. Pollack, "Modular interdependency in complex dynamical systems," *Artificial Life*, vol. 11, no. 4, pp. 445–457, 2005.

[34] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding," 2013.

[35] J. Clune and H. Lipson, "Evolving 3d objects with a generative encoding inspired by developmental biology," *ACM SIGEVOlution*, vol. 5, no. 4, pp. 2–12, 2011.

[36] R. Mills and R. A. Watson, "Variable Discrimination of Crossover Versus Mutation Using Parameterized Modular Structure," *Proceedings of the 9th annual conference on Genetic and evolutionary computation GECCO 07*, pp. 1312–1319, 2007.

[37] K. Deb and D. E. Goldberg, "Analyzing deception in trap functions," in *FOGA*, vol. 2, pp. 98–108, 1992.

[38] P. Larranaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*, vol. 2. Springer, 2002.

[39] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Computational optimization and applications*, vol. 21, no. 1, pp. 5–20, 2002.

[40] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Hierarchical problem solving and the bayesian optimization algorithm.," in *GECCO*, pp. 267–274, 2000.

[41] M. Pelikan and D. E. Goldberg, "Escaping hierarchical traps with competent genetic algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 511–518, 2001.

[42] T.-L. Yu, D. E. Goldberg, A. Yassine, and Y.-P. Chen, "Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm," *Lecture Notes in Computer Science*, pp. 1620–1621, 2003.

[43] T.-L. Yu, K. Sastry, D. E. Goldberg, and M. Pelikan, "Population sizing for entropy-based model building in discrete estimation of distribution algorithms," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 601–608, ACM, 2007.

[44] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[45] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2002.

[46] J. Clune, J.-B. Mouret, and H. Lipson, "The evolutionary origins of modularity," *Proceedings of the Royal Society B: Biological Sciences*, vol. 280, no. 1755, 2013.

[47] R. Mills, "How micro-evolution can guide macro-evolution: Multi-scale search via evolved modular variation." April 2010.

[48] C.-K. Ting, "On the mean convergence time of multi-parent genetic algorithms without selection," in *Advances in artificial life*, pp. 403–412, Springer, 2005.

[49] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[50] A. Qin, V. Huang, and P. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, 2009.