



AUTHOR:

TITLE:

YEAR:

OpenAIR citation:

This work was submitted to- and approved by Robert Gordon University in partial fulfilment of the following degree:

OpenAIR takedown statement:

Section 6 of the “Repository policy for OpenAIR @ RGU” (available from <http://www.rgu.ac.uk/staff-and-current-students/library/library-policies/repository-policies>) provides guidance on the criteria under which RGU will consider withdrawing material from OpenAIR. If you believe that this item is subject to any of these criteria, or for any other reason should not be held on OpenAIR, then please contact openair-help@rgu.ac.uk with the details of the item and the nature of your complaint.

This is distributed under a CC _____ license.

The Role of Walsh Structure and Ordinal
Linkage in the Optimisation of Pseudo-Boolean
Functions under Monotonicity Invariance

Lee A. Christie

A thesis submitted in partial fulfilment of the requirements of

Robert Gordon University

for the degree of

Doctor of Philosophy

February, 2016

Declaration

I hereby declare that this thesis is a record of work undertaken by myself. That it has not been the subject of any previous application for a degree and that all sources of information have been duly acknowledged.

Lee A. Christie

February, 2016

Acknowledgements

Firstly, I would like to acknowledge the invaluable support of my supervisory team: Professor John McCall and Dr David Lonie, for their feedback, guidance, and words of wisdom which kept me on track during this four-year-long PhD journey. In addition, I would also like to thank my examiners: Professor Jose Lozano and Dr Andrei Petrovski, for their insightful comments and suggestions.

I thank my mum, dad, and brothers for their support and encouragement throughout my life, my education, and endeavours without which I would not have achieved half of what I have. I thank my friends and fellow students I've known throughout my time as a PhD student, for their encouragement, inspiration, and distraction.

Finally, I would like to thank the IDEAS Research Institute for their support and funding of this project; and Robert Gordon University, for being a second home to me throughout the last ten years of undergraduate, postgraduate, and doctoral studies.

Abstract

Optimisation heuristics rely on implicit or explicit assumptions about the structure of the black-box fitness function they optimise. A review of the literature shows that understanding of structure and linkage is helpful to the design and analysis of heuristics. The aim of this thesis is to investigate the role that problem structure plays in heuristic optimisation.

Many heuristics use *ordinal operators*; which are those that are invariant under monotonic transformations of the fitness function. In this thesis we develop a classification of pseudo-Boolean functions based on rank-invariance. This approach classifies functions which are monotonic transformations of one another as equivalent, and so partitions an infinite set of functions into a finite set of classes. Reasoning about heuristics composed of ordinal operators is, by construction, invariant over these classes.

We perform a complete analysis of 2-bit and 3-bit pseudo-Boolean functions. We use Walsh analysis to define concepts of necessary, unnecessary, and conditionally necessary interactions, and of Walsh families. This helps to make precise some existing ideas in the literature such as benign interactions.

Many algorithms are invariant under the classes we define, which allows us to examine the difficulty of pseudo-Boolean functions in terms of function classes. We analyse a range of ordinal selection operators for an EDA. Using a concept of directed ordinal linkage, we define precedence networks and precedence profiles to represent key algorithmic steps and their interdependency in terms of problem structure. The precedence profiles provide a measure of problem difficulty. This corresponds to problem difficulty and algorithmic steps for optimisation.

This work develops insight into the relationship between function structure and problem difficulty for optimisation, which may be used to direct the development of novel algorithms. Concepts of structure are also used to construct easy and hard problems for a hill-climber.

Publications

Some parts of the work presented in this thesis have appeared in following publications:

- [1] L. A. Christie, D. P. Lonie and J. A. W. McCall, "Partial structure learning by subset Walsh transform," in *UK Workshop on Computational Intelligence (UKCI)*, pp. 128-135, 2013.
- [2] L. A. Christie, J. A. W. McCall and D. P. Lonie, "Minimal Walsh structure and ordinal linkage of monotonicity-invariant function classes on bit strings," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 333-340, 2014.
- [3] A. E. I. Brownlee, J. A. W. McCall and L. A. Christie, "Structural coherence of problem and algorithm: an analysis for EDAs on all 2-bit and 3-bit problems," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 2066-2073, 2015.
- [4] J. A. W. McCall, L. A. Christie and A. E. I. Brownlee, "Generating Easy and Hard Problems using the Proximate Optimality Principle," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 767-768, 2015.

Glossary

The following is a list of many of the common symbols used throughout this thesis with a reference to where they are defined.

ℓ	The size of a problem. (p. 32)	f	A fitness function. (p. 3)
Ω or X	The search space. (p. 3)	X_i	Random variable i of X . (p. 33)
\mathbf{x}	A candidate solution. (p. 33)	x_i	Element i of candidate \mathbf{x} . (p. 32)
α	A column vector of Walsh coefficients. (p. 36)	$f_{\mathbf{x}}$	The fitness of \mathbf{x} in fitness function f . (p. 32)
γ	Vector of values in \mathbf{x} of variables in subset γ . (p. 15)	Δf_i	Perturbation on X_i . (p. 43)
γ	A clique (a subset) of variables in X . (p. 17)	$R_f(\mathbf{x})$	The rank of \mathbf{x} in function f . (p. 49)
α_γ	Walsh coefficient of the clique γ . (p. 17)	\mathbf{C}_f	The equivalence class of which f is a member. (p. 50)
Γ	Linkage partition. (p. 15)	$f \sim g$	Functions f and g are rank equivalent. (p. 50)
\mathbf{f}	A column vector of fitness values. (p. 37)	\bar{x}	For Boolean value, $1 - x$. (p. 40)
δ_i	A difference between two fitness levels in a function. (p. 80)	H_ρ	$2^\ell \times 2^\ell$ Hadamard matrix. (p. 37)
Δ	A matrix of delta values. (p. 98)	$\text{sgn}(x)$	The sign of x , i.e. $-1, 0$, or 1 . (p. 51)

Contents

1	Introduction	1
1.1	Overview	1
1.2	Research Questions.....	1
1.3	Summary of Thesis	2
2	Literature Review	3
2.1	Search Heuristics	3
2.1.1	Black-Box Optimisation	3
2.1.2	Neighbourhood and Local Optima	4
2.1.3	Local Search and Hill-Climbers	6
2.1.4	Population-Based Metaheuristics	7
2.1.5	Competent Genetic Algorithms.....	9
2.1.6	Estimation of Distribution Algorithms	10
2.1.7	Distribution Estimation Using Markov Random Fields	11
2.1.8	Perturbation Methods	13
2.2	Structure of Optimisation Problems	15
2.2.1	Linkage Structure	15
2.2.2	Pseudo-Boolean Functions	16
2.2.3	Walsh Decomposition of Pseudo-Boolean Functions	17
2.3	Algorithms and Problem Difficulty.....	18
2.3.1	No Free Lunch Theorem	18
2.3.2	Proximate Optimality	18
2.3.3	Measures of Problem Difficulty.....	20
2.3.4	Bad Linkage and Spurious Correlations	22
2.3.5	Unnecessary Benign Interactions.....	24
2.3.6	Deception and Malign Interactions	25
2.4	Research Questions.....	27
3	Background.....	29

3.1	Terminology and Notation	29
3.1.1	Vectors and Vector Spaces.....	29
3.1.2	Functions on Bit Strings and Bit Vectors	32
3.1.3	The Space of Pseudo-Boolean Functions	34
3.1.4	Linear Transformation and Matrices	35
3.1.5	Walsh Coefficients as Basis Vectors	36
3.1.6	Walsh-Hadamard Transform.....	37
3.1.7	Function Transformations.....	40
3.2	Linkage Identification by Perturbation.....	43
3.2.1	Linkage Partition and Perturbations	43
3.2.2	Non-Linearity / Non-Monotonicity Detection	44
3.2.3	Heckendorn and Wright's Detect-Linkage Algorithm	45
3.2.4	Streeter's Optimisation Algorithm.....	47
4	Functions and Rank Equivalence	49
4.1	Rank Equivalence	49
4.2	Directed Ordinal Linkage.....	51
4.3	Pseudo-Boolean Benchmarks Functions.....	54
4.3.1	Definitions and Identities	54
4.3.2	Constant Functions	55
4.3.3	Needle-in-a-Haystack Functions	56
4.3.4	Ones Function.....	57
4.3.5	Zeros Function.....	59
4.3.6	Binary Value Function	60
4.3.7	1-Dimensional Checkerboard Function	62
4.3.8	Leading-Ones Function.....	64
4.3.9	Order-k Trap Function.....	66
4.3.10	Goldberg's Fully-Deceptive Order-3 Function.....	69
4.3.11	2-Bit and 3-Bit Function Values and Walsh Coefficients.....	70
4.4	1-Bit Pseudo-Boolean Functions	76
4.5	Counting Function Classes	77

4.6	Summary.....	78
5	2-Bit Pseudo-Boolean Functions	79
5.1	Counting 2-Bit Classes.....	79
5.2	Walsh Families and Delta Conditions	80
5.3	Automated Calculation of Walsh Families	88
5.4	Directed Ordinal Linkage and Epistasis.....	90
5.5	Precedence Networks and Precedence Profiles.....	92
5.6	Delta Linkage Detection	98
5.7	Structural Coherence	100
5.8	Summary.....	105
6	3-Bit Pseudo-Boolean Functions	107
6.1	Counting 3-Bit Classes.....	107
6.2	Walsh Families and Delta Conditions	108
6.3	Automated Calculation of Walsh Families	110
6.4	Conditionally-Necessary Interactions	112
6.5	Precedence Networks and Precedence Profiles.....	117
6.6	Equivalent Average Costs Network Sets	123
6.7	Precedence Networks Hierarchy	126
6.8	Parallelisation of Precedence Networks	128
6.9	Delta Linkage Detection	130
6.10	Structural Coherence	133
6.11	Summary.....	134
7	Higher-Dimensional Pseudo-Boolean Functions	135
7.1	Combining 3-Bit Classes	135
7.1.1	Concatenation of Non-Overlapping Functions	135
7.1.2	Stitching of Overlapping Functions	136
7.2	Precedence Networks	139
7.2.1	Limited Connectivity Precedence Network Algorithm.....	140
7.2.2	Sampling	145
7.3	Subset Walsh Transform	147

7.3.1	Description of Algorithm	147
7.3.2	Results	150
7.3.3	Theoretical Analysis	156
7.4	Proximate Optimality on Hill-Climbing Algorithms.....	159
7.4.1	Concept	160
7.4.2	Problem Generation and Evaluation.....	161
7.4.3	Results.....	163
7.4.4	Conclusions	164
7.5	Summary	165
8	Conclusions and Further Work	167
8.1	Conclusions	167
8.2	Further Work	168
8.2.1	Refinement of 3-Bit Structural Coherence	168
8.2.2	Further Development of Precedence Network Learning	169
8.2.3	Construction of Benchmark Functions.....	170
8.2.4	Subset Walsh Transform Sweep.....	171
8.2.5	Necessary Structure for Optimisation.....	172
8.2.6	Non Pseudo-Boolean Functions.....	173
	Appendix A – List of 3-Bit Classes (Python 3).....	175
	Appendix B – Calculating 3-Bit Families (ANSI C)	177
	Appendix C – Compiling and Running Sources	183
	Bibliography	185

1 Introduction

1.1 Overview

Optimisation is the task of locating the input associated with the global maximum or minimum value of some function. Optimisation has many real-world applications.

For an arbitrary black-box function the only way to guarantee that the optimum is found is by exhaustive search, which is usually computationally intractable. Heuristics may be applied to optimise a function, but these rely on assumptions about the function, for example, that small changes in input generally correspond to small changes in output; that output is constructed from the sum of smaller functions on the input variables; or that adjacent variables in input representation are more related than distant ones. These assumptions all refer to various aspects of the *structure* of the function.

Some heuristics build explicit representations of their estimate of the function's structure based on statistical modelling and independence testing. Others are understood to implicitly learn the structure by the convergence of a population of solutions maintained by the algorithm.

It is clear that theoretical understanding of function structure will aid the development and analysis of optimisation heuristics. This is the issue that this thesis attempts to address.

1.2 Research Questions

The aim of this thesis is an investigation into concepts of structure and linkage. This aim can be broken down into the following research questions:

1. What is the relationship between problem structure and problem difficulty?
2. How can we use structure to usefully classify problems?
3. Can we use structure to bound the number of algorithmic steps?
4. Can structure analysis motivate the development of novel algorithms?

1.3 Summary of Thesis

Chapter 2 presents an examination of the relevant literature on black-box optimisation metaheuristics, problem structure, and variable linkage, to support the formulation of relevant research questions.

Chapter 3 lays out the foundational concepts required by the main body of the thesis. This includes terminology and notation, definitions of Walsh structure and structure in terms of linkage partition, and methods of computing the above.

Chapter 4 defines ranks and function classes we construct to classify problems into a finite number of classes for a given problem length; this is in a way invariant under a range of operators applied by many common heuristics. We define ordinal linkage partition and directed ordinal linkage; which we use with monotonicity-invariant classes of functions.

Chapter 5 gives a detailed analysis of the structure of all classes of length-2 pseudo-Boolean functions. Here we describe the necessary Walsh structures to maintain the ranks, and connect this concept to ordinal linkage. We define the algorithmic steps sufficient to reach a global optimum in each case.

Chapter 6 summarises the result of applying the same detailed analysis to the structure of all classes of length-3 pseudo-Boolean functions, and shows that non-unique sets of necessary Walsh structure arise with the introduction of 3 variables.

Chapter 7 describes the extent of applicability and implications of the low-dimensional analysis of structure to higher-dimensional function spaces, with suggestions for guiding the development of novel algorithms.

Chapter 8 concludes the thesis, reflects on the relevant contributions and gives suggestions for future work.

2 Literature Review

In this chapter, we examine the relevant literature. This discussion is grouped into three sections. First we discuss the application of search heuristics to the optimisation of functions; then we discuss the structure of functions on which they operate; and then of the coherence between the two. Lastly, we form relevant research questions which are motivated by the relationship between search heuristics and objective functions.

2.1 Search Heuristics

In this section we give a brief introduction to a variety of search heuristics applied to optimisation problems. The emphasis is on their relationship with implicit or explicit structure learning. The concept of structure is then expanded upon in later sections.

2.1.1 Black-Box Optimisation

Optimisation is the search for the global optimum of a function f . This is sometimes called the *objective function*. A single-objective black-box optimisation function maps each element of a given domain Ω to the codomain Y , such that Y admits a total ordering, formally given by (1). The domain of the function Ω is often called the *search space*.

$$f : \Omega \rightarrow Y \quad (1)$$

A member x of the search space Ω is called a *candidate*. For a given function, there may be more than one global optimum. The problem *objective* specifies whether we define the *global optimum* to be the argument for the maximum, $\operatorname{argmax}_x f(x)$, or argument for the minimum, $\operatorname{argmin}_x f(x)$.

The literature classifies problems as *continuous optimisation* (where the search space is continuous, e.g. the real numbers) or *discrete optimisation* (where the search space is discrete, e.g. the integers). In practice, all metaheuristic search operates on discrete finite domains, as a limitation of the precision of the digital representation [5]. There is a subset of discrete optimisation: *combinatorial optimisation*, where the search space is finite. The search space of a combinatorial optimisation problem can take forms such as sets, vectors, combinations, or permutations with/without repetition and be of any finite discrete alphabet [6].

2.1.2 Neighbourhood and Local Optima

The domain may be a nominal set of candidates with no defined relationship between the candidates. An example of such a function is shown in Table 1.

x	Apple	Cherry	Orange	Peach	Pear
$f(x)$	1.98	0.03	0.88	2.0	1.58

Table 1 – Example of a function containing no obvious neighbourhood. The domain of f is $\{Apple, Cherry, Orange, Peach, Pear\}$ and the codomain is \mathbb{R} . For a maximisation objective, the global optimum is *Peach*.

For a function such as the above, the global optimum can be reached in time linear with respect to the size of the search space by exhaustively evaluating each candidate. However, for optimisation problems in general, this runtime is undesirable. For a combinatorial optimisation problem of ℓ variables, each of an alphabet of cardinality k , the search space size k^ℓ , for example 2^ℓ for a pseudo-Boolean function, hence the runtime of an exhaustive search is exponential in the length of the input. Worse still, for an unbounded discrete optimisation or a continuous optimisation problem the runtime of an exhaustive search is infinite.

A neighbourhood structure may be defined on the set. For example, if the domain is ordinal, two candidates may be considered *neighbours* if they are adjacent in the associated order. With a neighbourhood structure, the notion of local optima emerges. A *local optimum* is a candidate solution which has no neighbouring candidates which are better. Under this definition, we consider global optima also to be local optima. Informally, we can describe candidate x as *better* than candidate y if we are maximizing and $f(x) > f(y)$ or we are minimising and $f(y) > f(x)$.

x	0	1	2	3	4
$f(x)$	1.98	0.03	0.88	2.0	1.58

Table 2 – An example of a function for which an obvious neighbourhood structure exists on the domain – based on adjacent integers, i.e. x and y are neighbours if $|y - x| = 1$. The domain of f is $\{0, 1, 2, 3, 4\}$. For a maximisation objective the local optima are $\{0, 3\}$, of which 3 is also a global optimum.

Given a neighbourhood structure, a local optimum may be found by a series of moves from neighbour to neighbour. The area around any optimum where a given search heuristic leads to that optimum is called a *basin of attraction* [6]. The basin of attraction may be different for different search heuristics since it depends on how the search explores the search space and how the search terminates. An example of a basin of attraction for a hill-climber search is shown in Figure 1. If the search begins in the basin of attraction belonging to a global optimum, then the search strategy will lead to the global optimum without the need for exhaustive evaluation.

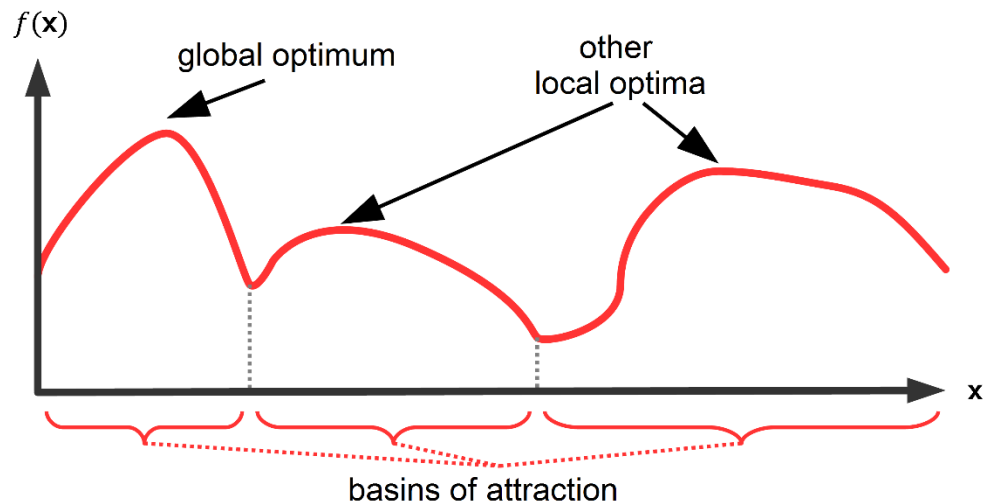


Figure 1 – Conceptual illustration of optima in an arbitrary function containing three local optima, one of which is the global optimum. The basins of attraction illustrated are those of a steepest ascent hill-climber.

Formally, a *metric* is any function d which defines the distance between two elements in a set, where d has the following four properties for all $x, y, z \in \Omega$ [7]:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0$ if and only if $x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$

An example of a metric on a multi-dimensional search space is the *Hamming metric* [8] (also called Hamming distance), which for a search space of $\Omega = \{0, 1\}^{\ell}$ is the number of variables which differ. A neighbourhood on such a search space may be defined such that two candidates are neighbours if they have a Hamming distance of 1.

For more complicated search spaces such as a permutation, it may be non-trivial to define the notion of neighbourhood structure. However, definitions for neighbourhood under permutations exist, such as two permutations differing by one switching of two elements [9].

2.1.3 Local Search and Hill-Climbers

The class of optimisation algorithms referred to as local search makes use of the *neighbourhood structure* of a search space, moving from one candidate at a time to a neighbouring candidate. *Hill-climber* search is a subset of local search. A simple hill-climber moves from one candidate in the search space to a neighbouring candidate of higher fitness.

For multi-dimensional optimisation, a local search strategy may find an improvement in more than one dimension to move. The search can follow the direction which gives the best improvement in fitness. This can be formalised as a fitness gradient where there exists a metric on both the domain and codomain, the gradient can be defined as $\Delta f(x)/\Delta x$. Following the largest local gradient is called *steepest ascent* [10] (as cited by [11]).

Most heuristics assume that neighbouring candidates correspond to similar function values. For a hill-climber to guarantee on a single run that it will reach the optimum, there is a smooth *fitness gradient* which may be followed from a randomly-chosen start point to the global optimum. A simple hill-climber search may be prevented from reaching the global optimum by encountering a *local optimum*. A local optimum is a point in the search space, whose neighbours correspond to less-optimal fitness, and hence is the optimum of the local region.

More advanced variations on hill-climber algorithms may avoid becoming trapped in a local optima. Examples are random restarts, tabu search [12] [13] [14], and simulated annealing [15] [16].

2.1.4 Population-Based Metaheuristics

One important branch of metaheuristics is population-based metaheuristics, the most common of which is the genetic algorithm (GA) as first introduced by Holland [17]. Variants on GAs include parallelisation schemes such as island models (Belding [18] as cited by Whitley et al. [19]).

Genetic algorithms are a form of evolutionary computing which uses selection, crossover, and mutation operators inspired by Darwinian natural selection. The canonical genetic algorithm (CGA) [17] is designed to search on pseudo-Boolean functions, however, genetic algorithms have been generalised to a wide variety of encodings.

In a population-based metaheuristic, a population consists of a collection of candidate solutions. The workflow consists of moving from population to population until stopping criteria are met. The means of generating a population from the previous varies between metaheuristics.

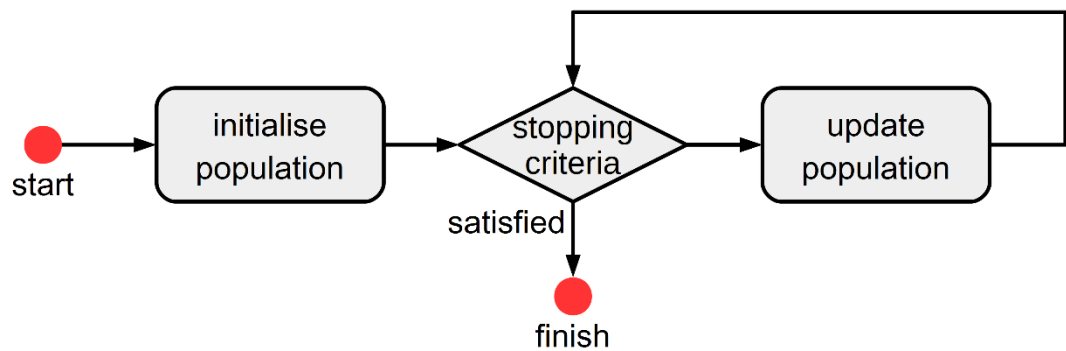


Figure 2 – The workflow of a population-based metaheuristic.

Different population-based metaheuristics have different means of updating the population. The cycle of a GA consists of applying selection to the current population, then generating the next population by the crossover-mutation procedure. This inner process which is iterated is summarised by (2).

$$\mathcal{P}^t \xrightarrow{\text{selection}} \mathcal{P}_S^t \xrightarrow{\text{crossover}} \mathcal{P}_{S^*}^t \xrightarrow{\text{mutation}} \mathcal{P}^{t+1} \quad (2)$$

Crossover is the recombination of two or more parent solutions, and mutation is the random modification of solutions to vary the population.

There are many selection methods, which can be classified as either ordinal or proportional. An *ordinal selection* operator bases its choice of individuals on ordinal (greater-than, less-than, equal-to) comparisons of fitness between candidate solutions in the current population. A *proportional selection* operator bases its choice of individuals in some way proportional to the numerical value of the fitness.

Ordinal Selection	
Truncation	Select the N individuals with the highest fitness values from \mathcal{P}^t .
Tournament	Uniformly at random choose M individuals from \mathcal{P}^t . Select the k individuals from this choice with the highest fitness. Repeat until N individuals have been selected.
Proportional Selection	
Roulette-Wheel	Iteratively select 1 individual with probability $p(\mathbf{x}) = \frac{f(\mathbf{x})}{\sum_{\mathbf{y} \in \mathcal{P}} f(\mathbf{y})}$ Repeat until N individuals have been selected.
Boltzmann	Iteratively select 1 individual with probability $p(\mathbf{x}) = \frac{e^{f(\mathbf{x})/T}}{\sum_{\mathbf{y} \in \mathcal{P}} e^{f(\mathbf{y})/T}}$ where T is a temperature parameter. Repeat until N individuals have been selected.

Table 3 – Examples of genetic selection operators. Adapted from Goldberg [20], Mitchell et al. [21], Davis [22], de la Maza & Tidor [23] as cited by Shakya [24, pp. 8-9].

Selection in a GA is the operator which uses information about the fitness of the candidates, therefore, any implicit learning of structure is done by selection. This makes the selection operator of particular interest to structure learning.

It is worthy of noting that ordinal selection operators are unaffected by any structure which may be produced by monotonic transformation of the objective function (that is, any transformation of the function which preserves equality and inequality relationships between candidate fitnesses). It is of interest to explore this further, and will be returned to in chapter 3.

2.1.5 Competent Genetic Algorithms

The linkage in genetic algorithms motivates schema theory. Schema theory considers that genetic algorithms work by processing *schema*, which are arrangements of variable values which correspond to higher fitnesses [17] [20]. However, schema which consist of more variables or of variables which are more spread out throughout the representation are more likely to be disrupted by crossover and mutation, making it difficult to maintain these schema in the population. Also, genetic algorithms are prone to *hitchhiking* [21], which is where arrangements of variables which do not contribute to high fitness are carried around in the population as they coincide in individuals in the population with high fitness schema.

Competent genetic algorithms were developed as a means of solving some of these issues in a genetic algorithm caused by mutation and re-combination operators [25]. The literature on competent genetic algorithms and perturbation [26] commonly classifies competent genetic algorithms into one of three categories:

1. Evolving representations/operators [25] [27] [28]
2. Probabilistic modelling (EDAs)
3. Perturbation methods (PMs)

Efforts to implement evolving representations/operators faced difficulty in relation to the algorithm's inability to respond quickly enough to selection [26]. Probabilistic modelling and perturbation methods are of more direct interest as they relate to explicit structure learning. We will next discuss EDAs, then PMs.

2.1.6 Estimation of Distribution Algorithms

Estimation of distribution algorithms (EDAs) are sometimes referred to as probabilistic model-building genetic algorithms. These algorithms build a probabilistic model of a *selected* sample population or a model weighted by fitness, and then sample that statistical model to generate new candidate solutions which have a high probability of having high fitness.

EDAs which use selection can use the same selection operators as genetic algorithms. The model-building/sampling step takes the place of the traditional GA crossover/mutation step.

The workflow of an EDA is given by (3) [29].

$$\mathcal{P}^t \xrightarrow{\text{selection}} \mathcal{P}_S^t \xrightarrow{\text{estimation}} p(x; \theta^t) \xrightarrow{\text{sampling}} \mathcal{P}^{t+1} \quad (3)$$

Since these steps are iterated, the operation of an EDA can alternatively be viewed as a progression from model to model as given by (4), rather than a progression from population to population.

$$p(x; \theta^t) \xrightarrow{\text{sampling}} \mathcal{P}^t \xrightarrow{\text{selection}} \mathcal{P}_S^t \xrightarrow{\text{estimation}} p(x; \theta^{t+1}) \quad (4)$$

EDAs are commonly classified based on the interaction of the variables: as univariate, bivariate, or multivariate [30] [31]. A univariate EDA considers each variable separately; a bivariate EDA such as BMDA [32] considers joint probability between pairs of variables, and a multivariate EDA consider joint probabilities or more than two variables. Early examples of EDAs: PBIL [33] and UMDA [34] were univariate. Multivariate EDAs use models such a Bayesian networks [35] and Markov networks [24].

2.1.7 Distribution Estimation Using Markov Random Fields

Distribution estimation using Markov random fields (DEUM) is a graphical EDA using a Markov random field (MRF) model, sometimes referred to as a Markov network [24]. The motivation for considering the DEUM EDA is that the Markov random field model is related to the Walsh coefficients of a function.

The univariate DEUM (DEUM_d) and bivariate DEUM (Is-DEUM) were introduced to demonstrate the use of Gibbs sampling, with DEUM_d assuming no interactions between the variables and Is-DEUM using a bivariate model [36].

DEUM has been used on real-world applications such as chemotherapy optimisation [37] [38], mushroom farming [39], and dynamic pricing [40].

Is-DEUM uses the known bivariate structure of the Ising spin glass problem. MRF models were introduced to model the Ising spin glass problem [41]. Structurally, a MRF is a graphical model using a hypergraph or simplicial set with weighted hyper edges. This is a collection of points, line segments, triangles, tetrahedra, and higher-dimensional simplices. Each simplex represents a *clique*, a set of zero or more of the variables.

The Hammersley-Clifford Theorem [42] [43] states that the joint probability distribution of a Markov random field can be factorised as a Gibbs distribution. The energy function $U(\mathbf{x})$ is used to compute the probability distribution of the model as given by (5) [44].

$$p(\mathbf{x}) = \frac{e^{-U(\mathbf{x})}}{\sum_{\mathbf{y}} e^{-U(\mathbf{y})}} \quad (5)$$

where $e^{-U(\mathbf{x})} = f(\mathbf{x})$

Note that this probability corresponds to the Boltzmann selection operator (Table 3, p. 8) used by other evolutionary algorithms such as Boltzmann GA [45] [46].

DEUM uses (6) and estimates the Walsh-Hadamard transform of this function [24] [47]. Additionally, this is an estimate based on a population using singular value decomposition (SDV) rather than an exhaustive sample [48].

$$-\ln(f(\mathbf{x})) = U(\mathbf{x}) \quad (6)$$

A version of DEUM (Is-DEUM_m) was developed for the Ising problem. Is-DEUM_m uses bitwise zero-temperature metropolis (BTZM) method. Maximum likelihood estimation (MLE) [49] and stochastic gradient descent [50] have also been used to estimate the structure in DEUM. There are also other EDAs which use Markov random field models, such as MEDA [51], MOA [52], and MARLEDA [53].

As with the genetic algorithm, EDAs can use ordinal selection or proportional selection. However, it is worth noting that the estimation step may use proportional information about the candidates, even if the selection is ordinal. Additionally, it is worth noting that DEUM may be used without selection if the estimation procedure carries information about fitness [54].

2.1.8 Perturbation Methods

Perturbation methods are algorithms which determine linkage in the underlying fitness function. The concept of linkage is loosely defined in the area of optimisation. Within genetics, linkage is defined by Winter et al. [55] as “the tendency for alleles of different genes to be passed together from one generation to the next”. Munetomo and Goldberg [56] [57] observe that this definition is not useful in optimisation as we wish to identify linkage in the underlying fitness function and that this is encoding-dependant. Linkage is sometimes regarded by the EDA community as the structure of the probabilistic model [58]. However, within this thesis we will take linkage to mean the dependence between variables in terms of additive separation as defined in section 2.2.1 as this is how linkage is typically regarded by perturbation methods.

Perturbation methods identify linkage by determining groups of interdependent variables. De Jong et al. [59] define *interdependence* as follows: “two variables in a problem are *interdependent* if the fitness contribution or optimal setting for one variable depends on the setting of the other variable”. Hence, and conversely, if variables are *independent*, the optimal setting of each variable and magnitude of effect on fitness for each possible value, may be determined without the context of knowledge of the setting of the other. We will look at the definition of linkage groups in the following section 2.2.1 on structure.

The first example of a perturbation method approach is linkage identification by nonlinearity check (LINC) [56]. LINC optimises pseudo-Boolean functions (functions on the domain $\Omega = \{0, 1\}^\ell$) and runs in $\mathcal{O}(\ell^2 2^k)$ time where k is the size of the largest group of interdependent variables [56]. This is the runtime of most early perturbation methods as in general, all combinations of two variables must be tested [60], however Streeter showed that the upper bound on complexity is $\mathcal{O}(\ell \log(\ell) 2^k)$ [61].

LINC considered variables interdependent if the *fitness contribution* of one variable depends on the setting of the other. The existence of such separate groups of variables is an aspect of structure which may be used to optimise is less time than the otherwise general upper bound of $\mathcal{O}(2^\ell)$.

A variant of LINC is linkage identification by non-monotonicity detection (LIMD) [56], which ignores *allowable non-linearity*. This means variables are considered interdependent if the *optimal setting* of one depends on the other, but *exact fitness contribution* is irrelevant. Since the goal is to locate a global optimum, the non-linearity condition is recognised as capturing unnecessary interactions – in the sense that such interactions do not affect the location of global optima. LIMD will not distinguish between two functions f and g where one is a monotonic transformation of the other.

Perturbation methods work by making small changes to a candidate \mathbf{x} and observing the effect on the fitness $f(\mathbf{x})$, referred to as fitness difference. Perturbation has also been used to detect linkage by estimating Walsh coefficients [62]. Other perturbation methods include dependency detection for distribution derived from df (D^5) [63] [60] and linkage identification with epistasis measures (LIEM) [64]. Additionally, it has been shown that hierarchical traps, a type of trap function with higher-order interactions between traps, can be solved in polynomial time with respect to the length of the problem input [59]. Another branch of perturbation methods is numerical optimisation [65] such as line search, trust-region methods, conjugate gradient methods, and quasi-Newton methods, which are usually applied to continuous optimisation problems and outside of the scope of this thesis.

2.2 Structure of Optimisation Problems

In section 2.1.2 we discuss neighbourhood structure on an optimisation function with regard to local optima and local search. In this section we discuss other relevant aspects of structure in optimisation problems: linkage structure as seen by perturbation methods, pseudo-Boolean functions and Walsh transforms.

2.2.1 Linkage Structure

A function of a subset $\gamma_i \subseteq \{x_0, x_1, \dots, x_{\ell-1}\}$ of the variables is called a *sub-function* f_i . An *additive decomposition* of a function is a sum of n sub-functions $\{\gamma_0, \dots, \gamma_{n-1}\}$ which is equivalent to the original function i.e. such that (7) holds.

$$f(\mathbf{x}) = f_0(\gamma_0) + f_1(\gamma_1) + \dots + f_{n-1}(\gamma_{n-1}) \quad (7)$$

where $(\forall i)(\gamma_i \subseteq \mathbf{x})$

A function is an *additively separable function* (ASF) [56] if and only if there exists an additive decomposition into two or more sub-functions where no variable appears in more than one sub-function. The resulting subsets of variables forms a *linkage partition* as in (8) where each γ_i is a *linkage group*.

$$\Gamma = \{\gamma_0, \gamma_1, \dots, \gamma_{n-1}\} \quad (8)$$

where $(\forall i \neq j)(\gamma_i \cap \gamma_j = \emptyset)$

The linkage partition Γ is a set of linkage groups γ (disjoint subsets of the variables X).

The general runtime for exhaustive search on a function of alphabet of size a is $\mathcal{O}(a^\ell)$. If the linkage partition of additive separation is known, the function may be optimised by optimising each linkage group separately for a runtime of $\mathcal{O}(n a^k)$ where k is the size of the largest linkage group, while still guaranteeing that the global optimum will be found [61].

2.2.2 Pseudo-Boolean Functions

Many evolutionary algorithms, including the canonical genetic algorithm (CGA), EDAs such as DEUM, and perturbation methods such as LINC operate on pseudo-Boolean functions. Pseudo-Boolean optimisation is a subset of combinatorial optimisation. A *pseudo-Boolean function* maps a string of ℓ binary digits to a codomain of real numbers \mathbb{R} , as given by (9).

$$f : \{0, 1\}^\ell \rightarrow \mathbb{R} \quad (9)$$

A number of standard benchmark functions have been devised on the space of pseudo-Boolean functions. The usual purpose of standard benchmark problems is to compare performance of search heuristics [58]. Here we introduce benchmarks which will be used later in discussions of structure and variable interaction.

Of particular interest to our work are benchmarks which can be examined over a very small problem length. These include the constant function (CONST), needle-in-haystack function (NEEDLE), ones function (ONEMAX) [66] [67], binary value function (BINVAL) [67], leading-ones function (LEADING) [67], a simplified chain variant of the checkerboard function (CHECK_{1D}) [68] [47, pp. 32-32], the order-k trap function (TRAP_k) [69] [70], and Goldberg's fully-deceptive order-3 function (GOLDBERG) [71] [72] [25]. Full discussion of chosen benchmark functions and their relevant properties is given in section 4.3.

Although pseudo-Boolean functions are a subset of the problems tackled by search heuristics, pseudo-Boolean optimisation has many practical applications including computer vision [73]. Additionally, real-world problems which in one form may be framed with non-binary variables, have been encoded using binary representations, and tackled as pseudo-Boolean optimisation problems [39] [37]. Many \mathcal{NP} problems are directly instances of pseudo-Boolean functions, including the maximum satisfiability [74], the Ising spin glass [41] problem, and the 0/1 knapsack [75] problem. Further, other \mathcal{NP} pseudo-Boolean functions can be reformulated in terms of an instance of the three-dimensional Ising spin glass problem [76] or other \mathcal{NP} -complete problems using \mathcal{NP} reductions.

2.2.3 Walsh Decomposition of Pseudo-Boolean Functions

The *Walsh decomposition* is an additive decomposition with sub-functions $\alpha_\gamma W_\gamma(\mathbf{x})$ of a pseudo-Boolean function [71]. Any pseudo-Boolean function can be uniquely specified by a set of Walsh coefficients α_k where k is a subset of variable indices 0 to $\ell - 1$. A subset of the indices means this coefficient relates to that specified subset of variables X_0 to $X_{\ell-1}$.

The coefficients multiply Walsh functions $W_\gamma(\mathbf{x})$. The Walsh functions $W_\gamma(\mathbf{x})$ are defined as given by (10).

$$W_\gamma(\mathbf{x}) = \prod_{i \in \gamma} \begin{cases} 1, & x_i = 1 \\ -1, & x_i = 0 \end{cases} \quad (10)$$

Note that this product is empty over the range $\prod_{\emptyset} \dots = 1$ hence, $W_{\emptyset}(\mathbf{x}) = 1$ for all \mathbf{x} .

It should be noted that an alternative convention used within the literature [71] [72] maps 1 to 0 and -1 to 1 (the opposite order of our chosen convention above). This only affects the sign of some coefficients, and does not alter any further analysis relevant to this work.

Any pseudo-Boolean function may be rewritten in the following form given by (11).

$$f(\mathbf{x}) = \sum_{\gamma \in K} \alpha_\gamma W_\gamma(\mathbf{x}) \quad (11)$$

$$K \subseteq \{0, 1, \dots, \ell - 1\}$$

As note above, $W_{\emptyset}(\mathbf{x}) = 1$ for all \mathbf{x} , hence the term $\alpha_{\emptyset} W_{\emptyset}(\mathbf{x})$ simplifies to $\alpha_{\emptyset} \cdot 1 = \alpha_{\emptyset}$. We refer to the α_{\emptyset} term as the *constant term* since it is independent of \mathbf{x} whereas all other terms in the Walsh expansion are non-constant functions of \mathbf{x} .

The non-zero Walsh coefficients indicates parts of the structure which are present [71]. A non-zero Walsh coefficient, on a clique of more than one variables, represents an interaction between the variables in the function. For example, the term $\alpha_{\{3,8\}}$ represents the bivariate interaction between variables X_3 and X_8 . If there is no bivariate interaction between these variables, the coefficient is zero.

2.3 Algorithms and Problem Difficulty

Having discussed search heuristics and structure of optimisation problems, we wish to tie the two concepts together. Here we examine concepts in the literature of coherence between heuristics and problems. This will help motivate relevant research questions.

2.3.1 No Free Lunch Theorem

The *no free lunch theorem* (NFL) [5] [77] states that any two (non-revisiting) black-box optimisation algorithms are equal when their performance is averaged over all possible problems. An improvement in performance in one problem or class of problems is balanced by loss of performance in another.

Wolpert and Macready [5] describe how NFL can be interpreted from a geometrical measure of the match between problem and algorithms, by the inner product between the two in the population simplex.

Since there are differences between the performance of different algorithms on different problem, it is of interest to examine how well particular problems and algorithms match up [78]. In particular, some problems are more interesting than others.

2.3.2 Proximate Optimality

Glover's *proximate optimality principle* [79] states that high fitness candidates have similar structures. This is a property of problems and representation, and of operators which operate on the representation space. Shown in Figure 3 are two candidates with similar high fitness for a travelling salesman problem (TSP) instance; these have many common elements of structure present. Figure 4 shows two similar-fitness low quality solutions with few common structures and two similar-fitness high quality solutions with many common structures. Assuming proximate optimality holds, EDAs should be able to find good candidates by sampling near other good candidates, and a GA nearing convergence should find good solutions at another fitness level.

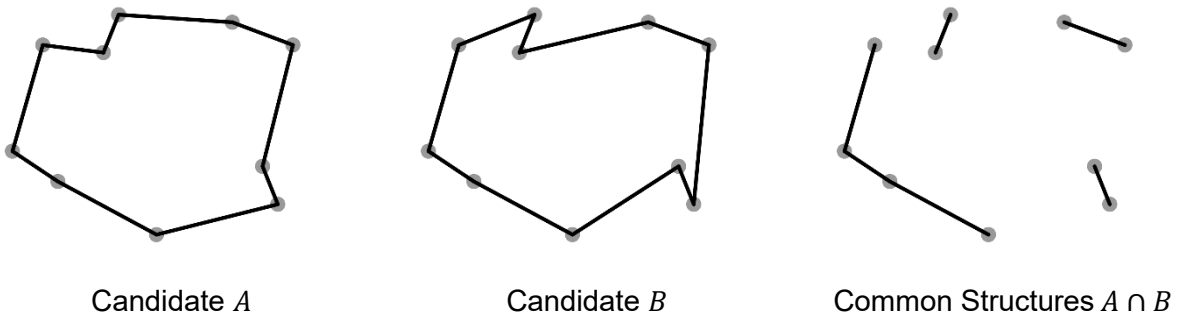


Figure 3 – Two similarly high-fitness solutions to a TSP instance, with many common structures in the intersection of the two solutions.

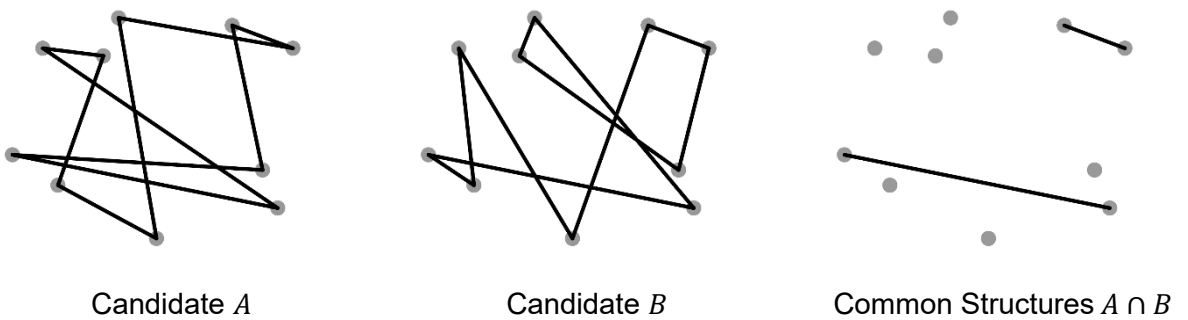


Figure 4 – Two similarly low-fitness solutions to a TSP instance, with few common structures in the intersection of the two solutions.

There is a related informal idea of the “big-valley hypothesis” which asserts that many search landscapes have a “big valley” structure. This means that local optima appear close to one another and close to global optima [80] [81].

2.3.3 Measures of Problem Difficulty

The no free lunch theorem does not prevent the analysis of problem difficulty. Analysis may be done on the basis of considering the set of specific real-world or benchmark problem, or by considering only functions with the same global optima [82].

Many measures of problem difficulty have been proposed. One method of discussing the complexity of an optimisation problem is to classify the function as univariate, bivariate, or multivariate [30]. From the perspective of the Walsh expansion we can define *univariate* functions as those which consist of at most univariate Walsh coefficients with all higher-order coefficients equal to zero. Similarly, *bivariate* functions consist of at most bivariate coefficients (including univariates), and lastly, *multivariate* functions are functions containing at least one term of order 3 or higher. Similarly, other models, such as Bayesian networks have related definitions for describing the complexity of a function. Although EDAs are often classified as univariate, bivariate, or multivariate, as mentioned in section 2.1.6, based on the order k of variables interactions modelled, it is often observed in practice that a univariate EDA may perform well on a high-order multivariate problem.

One common way to measure problem difficulty for search heuristics is to give an asymptotic analysis of the runtime of a particular search heuristic when the problem length ℓ can be scaled [83]. This is directly analogous to the classical complexity theory analyses of the runtime of algorithms. For example, Droste's analysis [84] shows that the general lower-bound on performance for the algorithm cGA [85] on the ONEMAX problems is $\mathcal{O}(p\sqrt{\ell})$, where p is the population size, however, another univariate problem, the BINVAL problem has $\mathcal{O}(p\ell)$ runtime. The difference in runtime is attributed to the fact that the BINVAL problem (in contrast to the ONEMAX problem) is classed as an *exponential problem* meaning that the influence of difference variables vary in exponential degree, and the performance of cGA is highly affected by this factor.

Other measures of problem difficulty include consideration of their *modality* : the number of non-global local optima and size of basins of attraction to these optima [86]. In principle, problems with more local optima and larger basins of attraction stand more chance of trapping an algorithm on a non-global local optimum and are therefore more difficult to optimise. By contrast, problems with more global optima with larger basins of attraction to these optima are easier to optimise than those with many non-global local optima.

When measuring the computational complexity of optimisation, the dominant operation in the computation is typically assumed to be the number of function evaluations even when complex model building or other processing is involved in the intermediate steps. This may be different for real-world optimisations but is generally held to in theory [83].

Problems can also be classified as to whether their structure is known, partially known, or black-box [87]. However, the majority of related literature on the topic of search heuristics treat functions as black-box, so we will be focused mainly on the objective function as a black-box function. It is worth noting that use of known or partially-known structure is an important consideration of real-world applications, and in practice there is usually some known structure to problems. Additionally, it is useful to consider the known structure of a benchmark function when evaluating the performance of an EDA or perturbation method in learning structure.

2.3.4 Bad Linkage and Spurious Correlations

In discussing the types of structure present in a function, it is important to distinguish between structure present and non-existent structure treated as structure by some artefact of the design of the algorithm or imperfect knowledge about the function – as is inevitable with black-box optimisation. Bad linkage and spurious correlations are two examples of such non-existent structure.

Representations of linkage in evolutionary algorithms can be classified as physical or virtual linkage [88] [89]. Physical linkage refers to linkage based on proximity in the encoding. The canonical genetic algorithm (CGA) [17] encodes linkage in this way. Virtual linkage refers to the explicit tracking of linkage using a statistical model in an EDA such as Bayesian joint probabilities or Markov random field cliques. Chen et al. [88] note that physical linkage is inspired by biological evolution and hence has biological plausibility, but that virtual linkage can achieve better performance.

The terms *bad linkage* [88] or false linkage [90] [91] refer to unhelpful juxtaposition of variables in a system of physical linkage such as a genetic algorithm. Bad linkage derives from the biological notion of linkage, in which genes are linked if they appear on the same chromosome. In a genetic algorithm this refers to variables which appear close in the representation and are likely to be preserved together in the same offspring by crossover. Schema theory implies that variables which are adjacent in the allele string are less likely to be disrupted than those which are distant if the GA uses a k-point crossover operator. Hence, those closer are more tightly linked in a typical GA. The development of competent genetic algorithms, and EDAs in particular was motivated by the problem of bad linkage and the disruption of building blocks in genetic algorithms.

EDAs use virtual linkage and do not suffer from the same problem of bad linkage as a GA. However, with EDAs, *spurious correlations* [87] or spurious dependencies [92], arise when the sampled population of an EDA by chance indicates the existence of a correlation which is merely an artefact of the sampled population, and not representative of a real correlation in the population. For example, if by random chance $x_5 = x_{17}$ for all highly-fit solutions in a sampled population, the EDA may model this as an interaction between variables X_5 and X_{17} . Spurious correlations are affected by factors such as selection size and population size [87] since this increases the sample size and hence the likelihood that the sampling is representative of the real structure of the function.

When the structure of an optimisation problem is known *a priori*, the accuracy of a model built by an EDA can be measured in terms of precision, recall, and f-measure (terms borrowed from the domain of data mining). In relation to EDAs, we can define precision, recall, and f-measure as given by (12-14) (Witten and Frank [93] as cited by Brownlee et al. [58]).

$$\text{precision } (p) = \frac{\text{true interactions found}}{\text{interactions found}} \quad (12)$$

$$\text{recall } (r) = \frac{\text{true interactions found}}{\text{true interactions}} \quad (13)$$

$$\text{f-measure } (F) = \frac{2pr}{p + r} \quad (14)$$

Here, a *true interaction* is an interaction between variables in the function's structure; an *interaction found* is an interaction between variables in the EDA's model; and a *true interaction found* is an interaction present in both.

The value for f-measure ranges from 0 to 1, with the optimal value for f-measure, in the case of *perfect model structure*, that all and only true interactions are found being 1 [58]. Hence, f-measure can be used to rate the quality of the statistical model built by a run of an EDA, assuming the structure of the problem is known. An example of applying f-measure is calculating the f-measure between learned Markov network coefficients in DEUM [58] against the known non-zero Walsh coefficients of the fitness function. It is also used in evaluating the accuracy of Bayesian network-based EDAs.

2.3.5 Unnecessary Benign Interactions

When an interaction reinforces the cumulative effect of its parts, this interaction is considered *benign*. It should be emphasised that the absence of a benign interaction may be preferable to its presence. It has been observed that not all interactions are necessary to solve an optimisation problem [94] [95] [96] [97]. For example, a univariate EDA performs better than naively expected on many problems which are bivariate or multivariate.

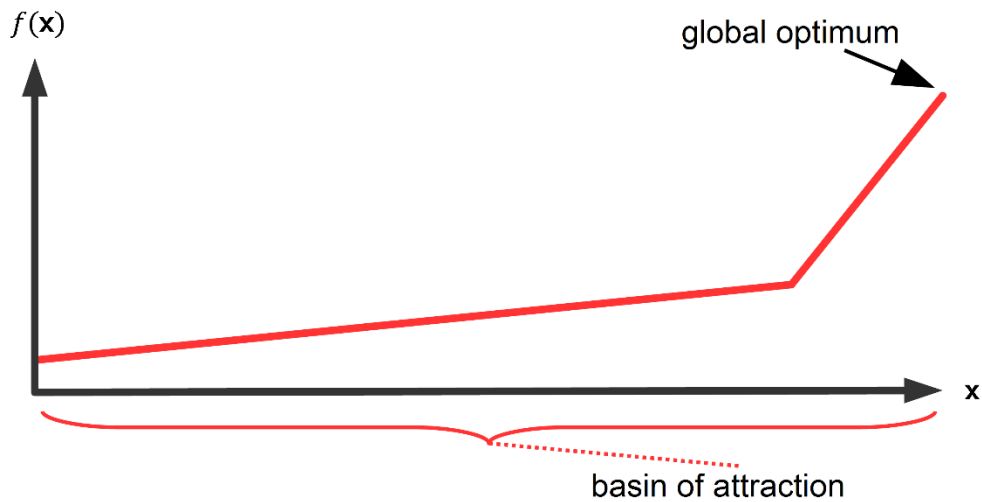


Figure 5 – Conceptual Illustration of a function containing unnecessary interactions. There is a change in gradient, however, the basin of attraction at both gradients leads to the same global optimum. The basins of attraction illustrated are those of a steepest ascent hill-climber.

It should be noted that if a monotonic transformation is applied to a function, this may produce a function with new interactions not present in the original function. Such interactions will always be unnecessary interactions since monotonic transformation preserves the invariants $f(y) > f(x)$ and $f(y) = f(x)$. Hence, any selection operator which is monotonically invariant will be unaffected by the presence of unnecessary interactions.

In an EDA with ordinal selection, the model will not be affected by unnecessary interactions unless the model-building step uses information about the fitnesses, such as in a Markov network EDA, hence, algorithms such as DEUM may model unnecessary interactions even with ordinal selection.

A genetic algorithm with ordinal selection will also be blind to benign interactions and therefore the presence of those interactions will not cause the GA to implicitly model them. However, the effect of hitchhiking, as discussed in section 2.1.5, may mean that these interactions are still modelled. Likewise, an EDA which only uses fitness information through an ordinal selection operator will not be lead to model unnecessary interactions except as spurious correlations as discussed in section 2.3.4. Additionally, an EDA such as DEUM which uses fitness information to build the model may directly model unnecessary interactions, even when the selection operator used is ordinal, or when no selection is used.

2.3.6 Deception and Malign Interactions

A *deceptive function* is one which leads optimisation away from the global optima by discovering *deceptive interactions* either explicitly or implicitly. Figure 6 illustrates the idea of what is called a *trap function*, this is one which has two unequal local optima, with the largest basin of attraction leading the optimisation towards the non-global local optimum and away from the global optimum.

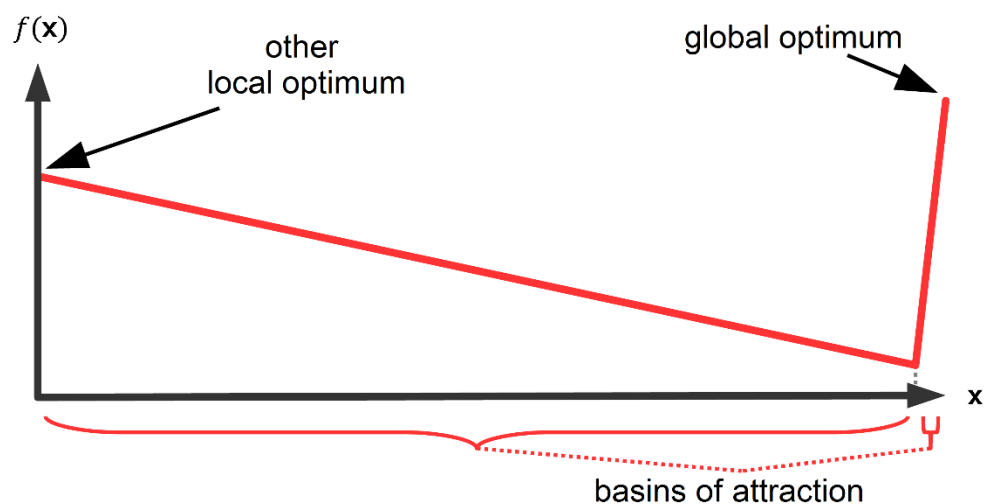


Figure 6 – Conceptual Illustration of a deceptive function containing two local optima, one of which is the global optimum. The global optimum has a considerably smaller basin of attraction than the other local optimum. The basins of attraction illustrated are those of a steepest ascent hill-climber.

Such functions may be contrasted with an *isolated* or *needle in a haystack* (NIAH) function [11], which are those where the solutions which are not the global optimum or adjacent to, are surrounded by equally-fit solutions, hence they do not lead the search in any direction.

Naively it may be assumed that the complete absence of information such as in the case of needle in haystack is the worst case, however it is known that deceptive interactions can be worse than isolated optima [86]. Misleading information in functions such as the k-trap directs the search away from global optima, which is not the case for isolated function such as needle in haystack. Some theoretical analysis has been done into the computational difficulty of escaping a sub-optimal basin [98].

In contrast to benign interactions, *malign interaction* reverses the combined effect of its parts [87]. Malign interactions [86] [99] are considered by the GA community to be synonymous with the concept of *deception*, which is when there are interactions which tend to lead a search heuristic away from the global optimum [86].

2.4 Research Questions

In section 2.2.1 we discussed the concept of linkage structure, and remark that for additively separable functions, a global optimum may be identified by separately optimising each linkage group, with time complexity $\mathcal{O}(a^k)$ where k is the number of variable in the linkage group and a is the size of the alphabet. We also mentioned in section 2.1.8 that the upper bound on optimising an additively separable pseudo-Boolean function is $\mathcal{O}(\ell \log(\ell) 2^k)$. In section 2.3.3 we discussed several measures of problem difficulty, including the univariate, bivariate, and multivariate classifications based on complexity of structure. These ideas motivate exploring the relationship between structure and problem difficulty.

In section 2.1.8 we discussed perturbation methods and noted that a function may be optimised while ignoring allowable non-linearity as is the case for non-monotonicity detecting perturbation methods. In section 2.1.4 we discuss population-based metaheuristics and their selection operators, including ordinal selection, which is a commonly-used class of selection, operators under which functions which are monotonic transformations of one another are indistinguishable. Further, we discussed in section 2.3.5 how benign interactions, which may be produced by monotonic transformation, are a relevant consideration for EDAs, and that ordinal selection can helpfully hide these unnecessary interactions from being modelled. It seems clear that there is a sense in which functions which are monotonic transformations of one another may be considered equivalent. This motivates using monotonicity-invariance as the basis for a formal definition of equivalence classes of functions.

In section 2.1 we discussed several different approaches to optimisation and in section 2.3 discussed the coherence between search heuristics and problems. This motivates exploring the algorithmic steps which may be used to solve different classes of problem, and how we can use knowledge of structure to direct the development of novel algorithms.

The research questions outlined are as follows:

1. What is the relationship between problem structure and problem difficulty?
2. How can we use structure to usefully classify problems?
3. Can we use structure to bound the number of algorithmic steps?
4. Can structure analysis motivate the development of novel algorithms?

3 Background

Having examined the relevant literature and identified research questions in chapter 2, we now introduce any further background concepts necessary to explore the research questions fully. This includes terminology and notation of vector spaces, linear transformation, Walsh-Hadamard transform, variable linkage partition, and methods of computing these descriptions of functions.

3.1 Terminology and Notation

3.1.1 Vectors and Vector Spaces

Functions of more than one variable can have structure between variables; a vector is one common way of representing multiple variables. A *vector* \mathbf{x} is a mathematical object which we will represent using the conventional notation of an ordered list of numbers, as in (15).

$$\mathbf{x} = [x_0 \ x_1 \ \cdots \ x_{\ell-1}] \quad (15)$$

A *vector space* is a set of vectors, where the operations of either adding together two vectors in the space or multiplying a vector in the space by a *scalar* (a number) produces another vector in the space [100]. An example of vector space is a real space of 3-dimensions as in (16).

$$\mathbf{x} \in \mathbb{R}^3 \quad (16)$$

An example of two vectors \mathbf{v} and \mathbf{w} in the space \mathbb{R}^3 are given by (17).

$$\begin{aligned} \mathbf{v} &= [1.0 \ 0.0 \ 1.5] \\ \mathbf{w} &= [5.5 \ 2.5 \ 1.0] \end{aligned} \quad (17)$$

An example of *vector addition* $\mathbf{v} + \mathbf{w}$ is given by (18). Note that, as a requirement of vector spaces, the resulting vector is also an element of \mathbb{R}^3 .

$$\mathbf{v} + \mathbf{w} = [6.5 \ 2.5 \ 2.5] \quad (18)$$

An example of *multiplication by a scalar* $5\mathbf{v}$ is given by (19). Note that, as a requirement of vector spaces, the resulting vector is also an element of \mathbb{R}^3 .

$$5\mathbf{v} = [5.0 \ 0.0 \ 7.5] \quad (19)$$

Here we list the *axioms* of a vectors space in (20), let \mathbf{u} , \mathbf{v} , and \mathbf{w} be vectors in a vector space and a , b be scalars. These axioms are necessary for a set of vectors to be considered a vector space [100].

$$\begin{aligned}
 (\forall \mathbf{u})(\forall \mathbf{v})(\forall \mathbf{w})(\mathbf{u} + (\mathbf{v} + \mathbf{w}) &= (\mathbf{u} + \mathbf{v}) + \mathbf{w}) \\
 (\forall \mathbf{u})(\forall \mathbf{v})(\mathbf{u} + \mathbf{v} &= \mathbf{v} + \mathbf{u}) \\
 (\exists \mathbf{0})(\forall \mathbf{v})(\mathbf{v} + \mathbf{0} &= \mathbf{v}) \\
 (\forall \mathbf{v})(\exists (-\mathbf{v}))(\mathbf{v} + (-\mathbf{v}) &= \mathbf{0}) \\
 (\forall a)(\forall b)(\forall \mathbf{v})(a(b\mathbf{v}) &= (ab)\mathbf{v}) \\
 (\forall \mathbf{v})(1\mathbf{v} &= \mathbf{v}) \\
 (\forall a)(\forall \mathbf{u})(\forall \mathbf{v})(a(\mathbf{u} + \mathbf{v}) &= a\mathbf{u} + a\mathbf{v}) \\
 (\forall a)(\forall b)(\forall \mathbf{v})((a + b)\mathbf{v} &= a\mathbf{v} + b\mathbf{v})
 \end{aligned} \quad (20)$$

Under these axioms, vector spaces may also be defined on finite fields. An example is the finite field $\text{GF}(2)$ with elements $\{0, 1\}$. The addition and multiplication operators are defined in Table 4.

+	0	1	×	0	1
0	0	1	0	0	0
1	1	0	1	0	1

Table 4 – The addition and multiplication operators for the field $\text{GF}(2)$.

$\text{GF}(2)$ has two elements; this can be extended to a vector space with 2^ℓ elements – each of which is in the set $\{0, 1\}^\ell$ – by performing bitwise operations, i.e. to add any two vectors \mathbf{u} and \mathbf{v} , corresponding terms in the vectors are added as in (21).

$$(\mathbf{w} = \mathbf{u} + \mathbf{v}) \Rightarrow (\forall i)(w_i = u_i + v_i) \quad (21)$$

Vectors are *linearly independent* if it is not possible to construct one of the vectors from a linear sum of the others. Vectors *span* the space if it is possible to construct any vector in the space from a linear sum of those vectors. *Basis vectors* of a space are any linearly independent set of vectors which span that space.

The standard basis vectors of an ℓ -dimensional space are the vectors consisting of a 1 in one element and 0 for every remaining element, for example the space \mathbb{R}^3 is spanned by the basis vectors \hat{x} , \hat{y} , and \hat{z} as defined by (22).

$$\begin{aligned}\hat{x} &= [1 \ 0 \ 0] \\ \hat{y} &= [0 \ 1 \ 0] \\ \hat{z} &= [0 \ 0 \ 1]\end{aligned}\tag{ 22 }$$

As basis vectors of \mathbb{R}^3 , a linear sum of \hat{x} , \hat{y} , and \hat{z} can construct any vectors in the space \mathbb{R}^3 , for example, the vector $[25 \ 5 \ 10] = 25 \hat{x} + 5 \hat{y} + 10 \hat{z}$.

3.1.2 Functions on Bit Strings and Bit Vectors

Having identified pseudo-Boolean functions as of interest, we now outline the terminology and notation which will be used. Recall that pseudo-Boolean functions are by definition on the domain $\{0,1\}^\ell$. An element of this domain can be written as a *bit string*, which is a concatenation of ℓ bits (variables on the domain $\{0,1\}$) as given by (23).

$$\begin{aligned} x_0x_1 \dots x_{\ell-1} \\ \text{where } x_i \in \{0,1\} \end{aligned} \tag{ 23 }$$

An example of a bit string of length 6 is given by (24).

$$010101 \tag{ 24 }$$

These conventions chosen will be used consistently throughout this thesis. It should be noted that variations of this notation are used in the literature, for example, indexing from 1 (i.e. $x_1x_2 \dots x_\ell$) or presenting bit strings from right-to-left (i.e. $x_{\ell-1}x_{\ell-2} \dots x_0$).

Fitness value of a pseudo-Boolean function $f(\mathbf{x})$ for a given bit string \mathbf{x} may be abbreviated using a bit string notation in subscript as given by definition (25).

$$\begin{aligned} f_{x_0x_1 \dots x_{\ell-1}} = f(x_0x_1 \dots x_{\ell-1}) \\ \text{where } x_i \in \{0,1\} \end{aligned} \tag{ 25 }$$

An example of this notation is given by (26).

$$f_{010101} = f(010101) \tag{ 26 }$$

There is a one-to-one correspondence between bit strings of length ℓ and vector spaces with ℓ elements in the field $\text{GF}(2)$ extended bitwise to 2^ℓ elements. Hence, it is useful to consider pseudo-Boolean functions from this perspective. A bit vector \mathbf{x} is a vector of ℓ bits as given by (27).

$$\mathbf{x} = [x_0 \ x_1 \ \cdots \ x_{\ell-1}] \quad (27)$$

where $x_i \in \{0, 1\}$

The underlying set of this vector space is given by (28).

$$\mathbf{x} \in \{0, 1\}^\ell \quad (28)$$

As shown, as with bit strings, by convention we will present all vectors left-to-right and all indexes from 0. Hence, the x_0 represents the first element and it written on the left; $x_{\ell-1}$ represents the last element and is written on the right. This convention is used throughout.

It will be useful to consider the search space X as a vector space, or as a multivariate random variable as given by (29). This allows us to decompose the function in terms of special basis function such as Walsh functions, and to construct probabilistic models based on variables.

$$X = X_0 \times X_1 \times \cdots \times X_{\ell-1} \quad (29)$$

where $X_i = \{0, 1\}$

A bit string \mathbf{x} can be considered as a *sample* of X , and x_i as a sample of X_i . This multivariate random variable interpretation is used by algorithms which build a probabilistic model (EDAs).

3.1.3 The Space of Pseudo-Boolean Functions

The set of all pseudo-Boolean functions of length ℓ is itself a real-valued vector space of length 2^ℓ , i.e. \mathbb{R}^{2^ℓ} . A pseudo-Boolean function can be represented as a vector \mathbf{f}_ℓ by listing the fitnesses of the candidates as in (30).

$$\mathbf{f}_\ell = \left. \begin{array}{c} f_{111\dots 1} \\ \vdots \\ f_{110\dots 0} \\ f_{010\dots 0} \\ f_{100\dots 0} \\ f_{000\dots 0} \end{array} \right\} 2^\ell \quad (30)$$

For example, the function $f(\mathbf{x}) = 3 + x_0 - 2x_1$ can be represented as in (31).

$$\mathbf{f} = \begin{bmatrix} f_{11} \\ f_{01} \\ f_{10} \\ f_{00} \end{bmatrix} = \begin{bmatrix} 3 + 1 - 2 \cdot 1 \\ 3 + 0 - 2 \cdot 1 \\ 3 + 1 - 2 \cdot 0 \\ 3 + 0 - 2 \cdot 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \\ 3 \end{bmatrix} \quad (31)$$

Here, the basis vectors of this space are the vectors where for all $\mathbf{x} \in \Omega$ the basis $e_{\mathbf{x}}$ is the vector with 1 in the position corresponding to $f_{\mathbf{x}}$ and 0 in all other positions. An example in the case of pseudo-Boolean function is given in (32).

$$e_{11} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, e_{01} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, e_{10} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, e_{00} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (32)$$

Hence in the above example, $\mathbf{f} = 2 e_{11} + 1 e_{01} + 4 e_{10} + 3 e_{00}$.

These delta functions also comprise all ℓ -bit pseudo-Boolean functions with a single fully-isolated global maximum with a value of 1, and all other candidates with a value of 0. As basis vectors, any pseudo-Boolean functions can be constructed from a linear sum of such functions.

3.1.4 Linear Transformation and Matrices

A *linear transformation* (or linear map) is a function t from a vector space to a vector space, which may be the same vector space. A linear transformation preserves sums and scalar multiplication, i.e. $t(\mathbf{v} + \mathbf{w}) = t(\mathbf{v}) + t(\mathbf{w})$ and $t(a\mathbf{v}) = a t(\mathbf{v})$ for all \mathbf{v}, \mathbf{w} in the space, and scalar a .

An *isomorphism* is a linear transformation t for which there is an inverse linear transformation which is a function t^{-1} which operates to undo the transformation, i.e. $t^{-1}(t(\mathbf{v})) = \mathbf{v}$ for all \mathbf{v} .

An *m-by-n matrix* is a rectangular array m rows and n columns. An example of a 3-by-5 matrix is given by (33).

$$\begin{bmatrix} 5 & 5 & 10 & 6 & 1 \\ 1 & 8 & 2 & 7 & 1 \\ 3 & 5 & 0 & 6 & 0 \end{bmatrix} \quad (33)$$

Matrices are of interest in this context since matrix-vector multiplication can be used to apply a linear transformation to a vector. The result of multiplying an m -by- n matrix with a length n column vector is a length m column vector, where for all i , element i is the row i of the matrix multiplied element-wise with the column vector as in (34).

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \\ ex + fy \end{bmatrix} \quad (34)$$

An example is given by (35), which is the application of a matrix multiplication to the column vector consisting of the elements 3 and 5.

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 6 \\ 10 \end{bmatrix} \quad (35)$$

This example linear transformation is an isomorphism. The inverse of this transformation is given by (36).

$$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 6 \\ 10 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} \quad (36)$$

3.1.5 Walsh Coefficients as Basis Vectors

Recall that the Walsh decomposition of a function is an additive decomposition applicable to any pseudo-Boolean function. The Walsh functions are also a set of basis vectors which span the space \mathbb{R}^{2^ℓ} . A set of Walsh coefficients can be written as a length 2^ℓ column vector as in (37).

$$\alpha_\ell = \left[\begin{array}{c} \alpha_{\emptyset|\ell} \\ \alpha_{\{0\}|\ell} \\ \alpha_{\{1\}|\ell} \\ \alpha_{\{0,1\}|\ell} \\ \alpha_{\{2\}|\ell} \\ \alpha_{\{0,2\}|\ell} \\ \alpha_{\{1,2\}|\ell} \\ \alpha_{\{0,1,2\}|\ell} \\ \vdots \\ \alpha_{\{0,1,2,\dots,\ell-1\}|\ell} \end{array} \right] \quad (37)$$

The Walsh functions $W_\gamma(\mathbf{x})$ are basis vectors in this space. For our earlier example $f(\mathbf{x}) = 3 + x_0 - 2x_1$, we can write the Walsh decomposition as in (38).

$$\alpha_\ell = \left[\begin{array}{c} \alpha_{\emptyset|\ell} \\ \alpha_{\{0\}|\ell} \\ \alpha_{\{1\}|\ell} \\ \alpha_{\{0,1\}|\ell} \end{array} \right] = \left[\begin{array}{c} 2.5 \\ 0.5 \\ -1.0 \\ 0.0 \end{array} \right] \quad (38)$$

Hence, the function can be written as $\alpha = 2.5 W_\emptyset(\mathbf{x}) + 0.5 W_{\{0\}}(\mathbf{x}) - W_{\{1\}}(\mathbf{x})$. Note that the coefficient of $W_{\{0,1\}}(\mathbf{x})$ in this function is zero, revealing something about the structure of the function – that X_0 and X_1 are independent. In contrast to the delta function basis, where zero terms do not immediately reveal information about the structure.

The Walsh coefficients may be computed by the Walsh-Hadamard transform, as described next, in section 3.1.6.

3.1.6 Walsh-Hadamard Transform

The *Walsh-Hadamard transform* is a method of calculating Walsh coefficients by using the Hadamard matrix [71]. The definition of the $n \times n$ *Hadamard matrix* [101] is given by (39).

$$H \cdot H^T = n I \quad (39)$$

The Walsh-Hadamard transform uses the Hadamard matrix of dimensions $2^\ell \times 2^\ell$ to calculate the Walsh coefficients of a function on bit string s of length ℓ . We use the notation H_ℓ to refer to this matrix.

Hadamard matrices of powers-of-2 dimensions can be constructed by Sylvester's construction [101] as given by (40).

$$H_\ell = \begin{bmatrix} H_{\ell-1} & H_{\ell-1} \\ H_{\ell-1} & -H_{\ell-1} \end{bmatrix}$$

where $H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ (40)

and $H_0 = [1]$

It is necessary to have *all* fitness values to determine the Walsh coefficients exactly, thus exhaustive evaluation of the function is required. The Walsh coefficients specify a particular function, and therefore if even one fitness value is different between two functions, their Walsh decomposition must be different in at least one Walsh coefficient.

As a consequence of the Walsh decomposition, a fitness vector may be determined from the system of linear equations represented in matrix-vector format as given by (41) [71].

$$\mathbf{f}_\ell \equiv H_\ell \boldsymbol{\alpha}_\ell \quad (41)$$

We can derive the inverse of the Hadamard matrix as given by (42),

$$H \cdot H^T \equiv n I$$

$$H \cdot H \equiv 2^\ell I$$

$$H \cdot \frac{1}{2^\ell} H \equiv I \quad (42)$$

$$H^{-1} \equiv \frac{1}{2^\ell} H$$

Using the inverse of the Hadamard matrix, we can compute the Walsh decomposition transform as given by (43).

$$\alpha_\ell = \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell \quad (43)$$

The subscript ℓ on \mathbf{f} , α , and H , will from now on usually be omitted in cases where the dimension of the vector or matrix is unambiguous.

Taking again our example $f(\mathbf{x}) = 3 + x_0 - 2x_1$; to determine the Walsh decomposition, we first determine the fitness vector \mathbf{f}_ℓ ; this is shown in (44).

$$\mathbf{f}_\ell = \begin{bmatrix} f_{11} \\ f_{01} \\ f_{10} \\ f_{00} \end{bmatrix} = \begin{bmatrix} 3 + 1 - 2 \\ 3 + 0 - 2 \\ 3 + 1 - 0 \\ 3 + 0 - 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \\ 3 \end{bmatrix} \quad (44)$$

Then we compute the Walsh-Hadamard transform; this is shown in (47).

$$\alpha_\ell = \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 4 \\ 3 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 10 \\ 2 \\ -4 \\ 0 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 0.5 \\ -1 \\ 0 \end{bmatrix} \quad (45)$$

Thus, the Walsh decomposition is $\alpha = 2.5 W_\emptyset(\mathbf{x}) + 0.5 W_{\{0\}}(\mathbf{x}) - W_{\{1\}}(\mathbf{x})$. Note that the coefficient of the term in $W_{\{0,1\}}(\mathbf{x})$ is zero since the last term in the α_ℓ vector is zero. This is expected since there is no interaction between the two variables.

The fast Walsh-Hadamard transform [102] is a more computationally efficient method of performing the Walsh-Hadamard transform. This calculates the Walsh coefficients in $\mathcal{O}(n \log n)$ addition/subtraction operations instead of $\mathcal{O}(n^2)$ as with the naive matrix multiplication method. Though exhaustive evaluation of the function is still required, which is likely the dominant operation.

To compute the fast Walsh-Hadamard transform, at each stage of the recursion, the first half of the output vector is computed by adding the corresponding input term and the input term from $n/2$ places along. The second half of the output vector is the same as the first but with subtraction instead of addition. Then the output vector is split in half and recursed on. The flow of data for a length 4 input vector is diagrammed in Figure 7.

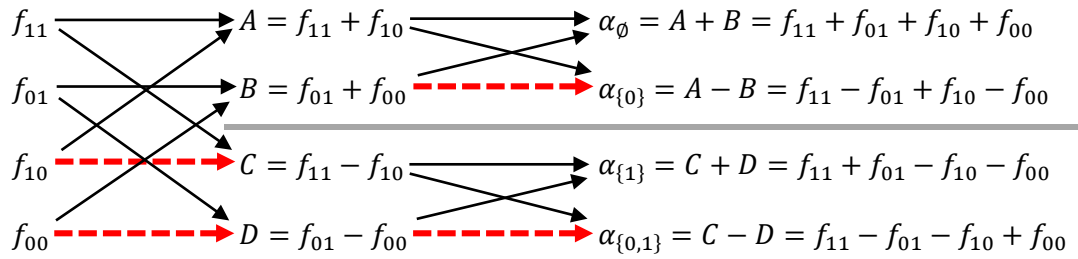


Figure 7 – Steps of the fast Walsh-Hadamard transform (FWHT) showing the divide-and-conquer procedure on four fitness values. Solid black lines indicates that the value is added; dashed red lines indicates that the value is subtracted. The intermediate steps have been assigned the labels A , B , C , and D to illustrate how these are added or subtracted in the next step.

3.1.7 Function Transformations

There are many operations which may be applied to produce variations on functions. Here we will discuss some commonly-used transformations and their relationship to structure.

Permutation of a function is a function such that the effect of 2 or more variables has been rearranged. For example in (46), g_A is a permutation of f_A where the variable x_2 takes the place of x_0 , the variable x_0 takes the place of x_1 , and the variable x_1 takes the place of x_2 .

$$\begin{aligned} f_A(\mathbf{x}) &= 2x_0 + x_1 \cdot \sqrt{x_2}; \quad \mathbf{x} \in \mathbb{R}^3 \\ g_A(\mathbf{x}) &= 2x_2 + x_0 \cdot \sqrt{x_1}; \quad \mathbf{x} \in \mathbb{R}^3 \end{aligned} \tag{ 46 }$$

A permutation of the variables may produce a function in which variables are moved between linkage groups, however, the total number of linkage groups and the size of each linkage group is preserved. In the case of pseudo-Boolean functions, a permutation may produce a function which the Walsh coefficients are permuted from the original function. For example $\alpha_{\{1,2\}}$ and $\alpha_{\{0,1\}}$ may be swapped.

Relabelling of a function is a function such that the effect of 2 or more labels for 1 or more variables has been rearranged. For example, in (47), g_B is a relabelling of f_B such that the effect of labels 0 and 1 has been swapped for variable x_1 .

$$\begin{aligned} f_B(\mathbf{x}) &= x_0 + x_1 \oplus x_2; \quad \mathbf{x} \in \{0, 1\}^3 \\ g_B(\mathbf{x}) &= x_0 + \bar{x}_1 \oplus x_2; \quad \mathbf{x} \in \{0, 1\}^3 \end{aligned} \tag{ 47 }$$

where $\bar{x}_i = (1 - x_i)$

For simplicity, in the above example, the variables have an alphabet of only two values each, but the multiple labels could be permuted in other ways. A relabelling does not change the linkage groups. In the case of pseudo-Boolean functions, for each single variable which is relabelled, all Walsh coefficients of cliques (sets of zero or more of the variables) including that variable will be negated. For example, $\alpha_{\{1\}}$ and $\alpha_{\{1,2\}}$ in the new function take on the old values of $-\alpha_{\{1\}}$ and $-\alpha_{\{1,2\}}$ respectively.

A variation of relabelling may be applied where there are different numbers of labels for the two functions. For example the ONEMAX function – the function on $\{0, 1\}^\ell$ whose output is the sum of the number of 1s in the input becomes an instance of attempting to break the code

in the game Mastermind using only black pegs, where for each peg, one label (the secret colour) has the effect of 1, and every other label has the effect of 0 [103].

A *concatenation* of functions is when a function of n variables and a function of m variables are added together *with permutation* such that no variable appears in both functions. This defines a function of $n + m$ variables. For example in (48), h_C is a concatenation of f_C and g_C . Note that the variables of g_C (variables x_0, x_1 , and x_2) have been permuted into variables x_2, x_3 , and x_3 .

$$\begin{aligned} f_C(\mathbf{x}) &= (3x_0 - 5x_1); & \mathbf{x} \in \mathbb{R}^2 \\ g_C(\mathbf{x}) &= (x_0 + x_1 \cdot x_2); & \mathbf{x} \in \mathbb{R}^3 \\ h_C(\mathbf{x}) &= (3x_0 - 5x_1) + (x_2 + x_3 \cdot x_4); & \mathbf{x} \in \mathbb{R}^5 \end{aligned} \quad (48)$$

Concatenation is often done with ‘trap’ functions [69] [70], which are functions of a small number k of variables, concatenated into much longer problems of length ℓ , this is also done in conjunction with further randomly-chosen permutation of the variables, so that related variables are shuffled around the function and not clustered together. Concatenated functions will have the linkage groups of the two old functions (after relabelling), and for pseudo-Boolean functions, the Walsh decomposition will be the sum of the Walsh decompositions of the two original functions (after relabelling).

A *monotonic transformation* of a function is a function produced by composing a function t with another function f , where t is monotonic increasing when its variable is in the codomain of f . For example in (49), g_D is a monotonic transformation where the transformation function $t(y) = 3 \cdot \ln(y) - 6$, note that $t(y)$ is monotonic increasing whenever y is in the codomain of f_D .

$$\begin{aligned} f_D(\mathbf{x}) &= 10 + 20 x_0^2 + 30 x_1^4; & \mathbf{x} \in \mathbb{R}^2 \\ g_D(\mathbf{x}) &= 3 \cdot \ln(f_D(\mathbf{x})) - 6; & \mathbf{x} \in \mathbb{R}^2 \end{aligned} \quad (49)$$

It is a property of monotonic transformations that $f(y) > f(x)$ implies $t \circ f(y) > t \circ f(x)$ and $f(y) = f(x)$ implies $t \circ f(y) = t \circ f(x)$. Hence, applying a monotonic transformation will not affect the behaviour of an algorithm if all operators (and model-building) are ordinal-based, i.e. that they operate only on less than, equal to, greater than comparison. This is not true of proportional operators since direct fitness value comparisons such as $f(y) - f(x)$ and $f(y)/f(x)$ are *not* invariants under monotonic transformation. A monotonic transformation could introduce new non-zero Walsh coefficients and new linkages between variables which

were previously not linked. This is discussed further in section 2.3.5 with regard to unnecessary interactions.

Multiple transformations of functions may be combined. Relabelling and permutations are commonly used in conjunction to produce variations of benchmark functions if the benchmark functions all have a common, predictable global optimum, or to create non-local structure in functions based on benchmarks where related variables are adjacent in the original function. Hence a set of benchmark functions which all have the same optima can be made less trivial.

3.2 Linkage Identification by Perturbation

Many linkage identification algorithms, or perturbation methods, have been proposed, in addition to EDAs which build an explicit model of variable interactions. In this section, we describe the algorithms which are most relevant to the work in the identified aims, although this is not an exhaustive list of perturbation methods.

3.2.1 Linkage Partition and Perturbations

Linkage detection as done by perturbation methods (PMs) uses small changes in the objective function's input and calculates the effect on the function's output. This is called *fitness difference*. The fitness difference for one-bit perturbation $\Delta f_i(\mathbf{x})$, is given in (50) and two-bit perturbation $\Delta f_{ij}(\mathbf{x})$ in (51), where $\bar{x}_i = (1 - x_i)$ for pseudo-Boolean functions.

$$\Delta f_i(\mathbf{x}) = f([x_0 \ x_1 \ \dots \ \bar{x}_i \ \dots \ x_{\ell-1}]) - f([x_0 \ x_1 \ \dots \ x_i \ \dots \ x_{\ell-1}]) \quad (50)$$

$$\Delta f_{ij}(\mathbf{x}) = f([x_0 \ x_1 \ \dots \ \bar{x}_i \ \dots \ \bar{x}_j \ \dots \ x_{\ell-1}]) - f([x_0 \ x_1 \ \dots \ x_i \ \dots \ x_j \ \dots \ x_{\ell-1}]) \quad (51)$$

Some conventions define perturbations using the negation of the definitions given above, however, the effect of using the other convention would only result in a global change of sign, which would not affect the detection of linkage, since changing the sign on all fitness differences does not change whether variables are additively separable or not since $-(a + b) \equiv (-a) + (-b)$.

The fitness difference by one-bit perturbation corresponds to the fitness gradient local to the point \mathbf{x} in one direction. The fitness difference by two-bit perturbation means moving in two dimensions, and the change in fitness will be the sum of the two corresponding one-bit fitness differences if the two variables are independent. This is formalised by the LINC algorithm as described next in section 3.2.2.

3.2.2 Non-Linearity / Non-Monotonicity Detection

Recall that linkage can be defined in terms of an additive separation of the variables into sub-functions (as discussed in 2.2.1). Linkage identification by non-linearity check (LINC) [56] detects linkage by making two one-bit perturbations $\Delta f_i(\mathbf{x})$, and $\Delta f_j(\mathbf{x})$, and checking that the effect of both perturbations at both loci together $\Delta f_{ij}(\mathbf{x})$ is not a sum of the effect of each. In other words, we can define the condition $E_{ij}(\mathbf{x})$ for linearity as given by (52) [56].

$$\mathcal{L}_{\text{LINC}}(i, j) \Leftrightarrow (\exists \mathbf{x})(\neg E_{ij}(\mathbf{x})) \quad (52)$$

where $E_{ij}(\mathbf{x}) \Leftrightarrow (\Delta f_{ij}(\mathbf{x}) = \Delta f_i(\mathbf{x}) + \Delta f_j(\mathbf{x}))$

If this condition $E_{ij}(\mathbf{x})$ is false for at least one string \mathbf{x} , then X_i and X_j are linked. If this relation is true for all strings, then they are unlinked [57]. Note that this linearity relation can hold for some strings even when non-linearity is present, for example in the trap function, the deceptive nature of the function means that most of the configurations of the trap sub-function are linear, leading away from the optimum. Only when one-bit from the optimum configuration is non-linearity detected. Thus, unless all strings are tested, the linkage detection is an estimate.

A variant on LINC is linkage identification by non-monotonicity detection (LIMD). LIMD uses a definition of linkage which disregards what is called *allowable non-linearity*. This has further been developed in the development of LIMD to consider only non-monotonic interactions [56]. These monotonic non-linearities are unnecessary or benign interactions. The condition for LIMD is given as defined by (53) [56].

$$\mathcal{L}_{\text{LIMD}}(i, j) \Leftrightarrow (\forall \mathbf{x})(P_{ij}(\mathbf{x}) \Rightarrow M_{ij}(\mathbf{x}))$$

where $P_{ij}(\mathbf{x}) \Leftrightarrow (\Delta f_i(\mathbf{x}) > 0 \text{ and } \Delta f_j(\mathbf{x}) > 0)$ (53)

and $M_{ij}(\mathbf{x}) \Leftrightarrow (\Delta f_{ij}(\mathbf{x}) > \Delta f_i(\mathbf{x}) \text{ and } \Delta f_{ij}(\mathbf{x}) > \Delta f_j(\mathbf{x}))$

3.2.3 Heckendorn and Wright's Detect-Linkage Algorithm

Heckendorn and Wright's DETECT-LINKAGE algorithm [62] defines a procedure for determining Walsh linkage by estimating Walsh coefficients using probes. A *probe* is defined by (54).

$$P(f, m, c) = \frac{1}{2^{\text{ONEMAX}(m)}} \sum_{i \in \mathcal{B}_m} (-1)^{\text{ONEMAX}(i)} \cdot f(i \oplus c) \quad (54)$$

Here, \mathcal{B}_m is defined as the set of submasks of the bitmask m , as defined by the (55).

$$\mathcal{B}_m = \{i \in \{0, 1\}^\ell : i \subseteq m\} \quad (55)$$

e.g. subsets of the mask [1 0 0 1] are {[1 0 0 1], [0 0 0 1], [1 0 0 0], [0 0 0 0]}.

The *background value* c , this is a random assignment of all variables *not* included in the mask. For example, if the mask [1 0 0 1] an example of a value for c might assign $x_1 = 1$ and $x_2 = 0$, i.e. [* 1 0 *], then $c = [0 1 0 0]$.

This will be combined using the subsets of the mask using a bitwise OR to produce {[1 1 0 1], [0 1 0 1], [1 1 0 0], [0 1 0 0]}. Note that all arrangements of X_0 and X_3 are tried against one fixed arrangement of X_1 and X_2 . In this example, the terms in the sum are given by (56).

$$\begin{aligned} & (-1)^2 \cdot f([1 1 0 1]) + (-1)^1 \cdot f([0 1 0 1]) \\ & + (-1)^1 \cdot f([1 1 0 0]) + (-1)^0 \cdot f([0 1 0 0]) \quad (56) \\ & = f([1 1 0 1]) - f([0 1 0 1]) - f([1 1 0 0]) + f([0 1 0 0]) \end{aligned}$$

This, multiplied by the factor $1/2^2$ outside the sum corresponds an estimate of the Walsh coefficient $\alpha_{\{0,3\}}$, which should be non-zero if the linkage is evident at the location defined by the background value c .

The algorithm returns the linkage in the form of a set of cliques corresponding to non-zero Walsh coefficients. The procedure is given by Algorithm 1. The linkage detection algorithm calculates linkage in $\mathcal{O}(2^k \ell^j \log \ell)$, where j is a parameter specifying the size of probes, k is the size of the largest linkage group, and ℓ is the problem size. This is an improvement over the runtime of deterministic approaches to finding linkage, which run in $\mathcal{O}(\ell^k)$ time [104].

```

1) initialise  $E$  to  $\emptyset$ 
2) for each mask  $m$  with  $\text{ONEMAX}(m) = j$ 
  a) if  $m \notin E$ 
    i) for  $i \leftarrow 1$  to  $N$ 
      (1)  $c \leftarrow$  random string in  $\mathcal{B}_m$ 
      (a) if  $P(f, m, c) \neq 0$ 
        (i)  $E \leftarrow E \cup \{m\}$ 
        (ii) break
    b) return  $E$ 

```

Algorithm 1 – Heckendorn and Wright’s DETECT-LINKAGE Algorithm [62] (notation modified).

The parameter j defines the size of the bitmasks which will be used, i.e. the number of 1s in the bitmask. Each mask of this size is tried, and N probes are done with each probe being on a new randomly-generated background value c . When the probe $P(f, m, c)$ returns a non-zero value, the clique is added to the set of known non-zero cliques E .

3.2.4 Streeter's Optimisation Algorithm

Streeter's ASFOPTIMISE algorithm [61] demonstrates that the linkage of an additively-separable function can be learned in $\mathcal{O}(2^k \ell \log \ell)$ function evaluations, which is an improvement on $\mathcal{O}(2^k \ell^2)$ or $\mathcal{O}(2^k \ell^j \log \ell)$ for all $j > 1$ function evaluations used by earlier approaches. A high level overview of ASFOPTIMISE is given in Algorithm 2.

- 1) **initialise** x to random string
- 2) **define** Γ as $\{\{0\}, \{1\}, \dots, \{\ell - 1\}\}$
- 3) local search to make x optimal with respect to 1-bit perturbations
- 4) **do** t **times**
 - a) **for** $i \leftarrow 0$ **to** $\ell - 1$
 - i) perform randomised test on position i
 - ii) if test succeeds:
 - (1) binary search to find j such that $\mathcal{L}(i, j)$
 - (2) update Γ
 - (3) local search to make x optimal with respect to newly discovered linkage
- 5) **return** x

Algorithm 2 – Overview of Streeter's ASFOPTIMISE algorithm [61] (notation modified).

The procedure uses a randomised test by generating two random assignments of the variables not currently known to be linked with variable x_i . To test for linkage on position i , two random strings are chosen s_A and s_B , then a 1-bit perturbation is performed on each: $\Delta f_i(s_A)$ and $\Delta f_i(s_B)$. If the two fitness differences are not the same, then some variable which is different in s_A and s_B is epistatically linked with X_i .

The process of binary search to find j such that $\mathcal{L}(i, j)$, begins by determining the set δ of variables (other than X_i) which differ between s_A and s_B . Then binary search is performed on this set to determine one X_j which is linked with X_i .

An example is given by (57) where $i = 3$.

$$\begin{aligned} \mathbf{s}_A &= [0 \ 1 \ 1 \ * \ 0 \ 0 \ 1 \ 1] \\ \mathbf{s}_B &= [1 \ 1 \ 1 \ * \ 1 \ 1 \ 0 \ 1] \\ \delta &= \{X_0, X_4, X_5, X_6\} \end{aligned} \tag{ 57 }$$

The first level of the binary search will use \mathbf{s}_A with $\{X_0, X_4\}$ changed on the left-hand-side of the binary search and use \mathbf{s}_A with $\{X_5, X_6\}$ changed on the right-hand-side of the binary search, i.e. the left hand side of the binary search will compare the fitness differences by perturbation at X_3 for $[0 \ 1 \ 1 \ * \ 0 \ 0 \ 1 \ 1]$ and $[1 \ 1 \ 1 \ * \ 1 \ 0 \ 1 \ 1]$. The next level of the binary search will compare the fitness differences by perturbation at X_3 for $[0 \ 1 \ 1 \ * \ 0 \ 0 \ 1 \ 1]$ and $[1 \ 1 \ 1 \ * \ 0 \ 0 \ 1 \ 1]$. At this level in the binary search the algorithm is comparing the effect of changing one bit (X_0) on the fitness difference $\Delta f_i(\mathbf{s}_A)$, which corresponds to a 2-bit perturbation $\Delta f_{ij}(\mathbf{s}_A)$, which can detect non-linearity at this point.

4 Functions and Rank Equivalence

In this chapter we define ranks, and rank-equivalence classes, which are used to describe a given monotonicity-invariant subspace of the function space. This classification is invariant under a variety of commonly-used operators and algorithms, allowing us to usefully reason about sets of functions. We also define ordinal linkage partition and directed ordinal linkage as a variation on descriptions of linkage structure which exists in the literature. We give a set of benchmark functions used as objects of study throughout this thesis to illustrate various points.

4.1 Rank Equivalence

We define the *rank* $R_f(\mathbf{x})$ of solution \mathbf{x} with respect to function f as the number of candidates in the finite search space X which correspond to a strictly smaller fitness value. This we define as given by (58), where $|S|$ represents the cardinality of set S . Note that as discussed in section 2.1.1, this also applies to digital representations of continuous domains.

$$R_f(\mathbf{x}) = |\{\mathbf{y}: \mathbf{y} \in X \wedge f(\mathbf{y}) < f(\mathbf{x})\}| \quad (58)$$

This definition of rank chosen is independent of whether the objective is maximisation or minimisation. Thus, for a maximisation objective, a higher rank is desired, and for a minimisation objective, a lower rank is desired.

We define two functions f and g as equivalent if they have the same search space and the rank of each solution is the same for each solution in the search space, formally as given by (59).

$$(f \sim g) \Leftrightarrow (\forall \mathbf{x} \in X)(R_f(\mathbf{x}) = R_g(\mathbf{x})) \quad (59)$$

By choosing some ordering on the search space $X = \{\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_{n-1}\}$, we can write a vector of ranks as in (60).

$$\mathbf{C}_f = [R_f^{\mathbf{s}_0} \ R_f^{\mathbf{s}_1} \ \dots \ R_f^{\mathbf{s}_{n-1}}] \quad (60)$$

This allows us to write the condition for rank-equivalence $f \sim g$ as given by (61).

$$(f \sim g) \Leftrightarrow (\mathbf{C}_f = \mathbf{C}_g) \quad (61)$$

As the term $\mathbf{C}_f = \mathbf{C}_g$ is an equality, the equivalence $f \sim g$ by construction satisfies the properties of reflexivity, symmetry, and transitivity necessary for an equivalence class.

As we will be focused on pseudo-Boolean function classes, we define a pseudo-Boolean class \mathbf{C}_f as given by the ordering specified in (62).

$$\mathbf{C}_f = [R_f^{111\dots 1} \ R_f^{011\dots 1} \ R_f^{101\dots 1} \ R_f^{001\dots 1} \ \dots \ R_f^{000\dots 0}] \quad (62)$$

We choose this ordering to be consistent with the order of fitnesses used in the form of the Walsh-Hadamard transform described in section 3.1.3. This vector \mathbf{C}_f can be used as a specific instance of the function, which is the *representative* of the class \mathbf{C}_f .

4.2 Directed Ordinal Linkage

We generalise the definition of non-monotonicity linkage by restricting the detection of linkage to be invariant under rank-equivalent functions by construction. Linkage is one view of structure in a function. Recall that linkage, including non-monotonicity detection, is typically defined as a symmetric relation, such that $\mathcal{L}(i, j) \Leftrightarrow \mathcal{L}(j, i)$, specifying a partitioning Γ of the variables X .

We use a definition of linkage, $\mathcal{L}_o(i, j)$ which is based on a difference in sign of rank difference, given by (63). This is an equivalent formulation to directed ordinal linkage based on sign of fitness difference that we presented in [2].

$$\begin{aligned} \mathcal{L}_o(i, j) &\Leftrightarrow \exists x : \text{sgn}(\Delta_j R_f(\mathbf{x}[X_i \rightarrow 1])) \neq \text{sgn}(\Delta_j R_f(\mathbf{x}[X_i \rightarrow 0])) \\ &\text{where } \Delta_j R_f(\mathbf{x}) = R_f(\mathbf{x}[X_j \rightarrow 1]) - R_f(\mathbf{x}[X_j \rightarrow 0]) \\ &\text{and } \mathbf{x}[X_i \rightarrow v] = [X_0 \ X_1 \ \dots \ X_{i-1} \ v \ X_{i+1} \ \dots] \quad (63) \\ &\text{and } \text{sgn}(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x \geq 0 \end{cases} \end{aligned}$$

The relationship $\mathcal{L}_o(i, j)$ does not necessarily imply $\mathcal{L}_o(j, i)$, for example, in the function shown in Table 5, $\mathcal{L}_o(0,1)$ is true, since for $\mathbf{x} = [0 \ 0]$, $\text{sgn}(\Delta_1 R_f(\mathbf{x}[X_0 \rightarrow 1])) = 1$ and $\text{sgn}(\Delta_1 R_f(\mathbf{x}[X_0 \rightarrow 0])) = 0$. However $\mathcal{L}_o(1, 0)$ is false, since for all \mathbf{x} , $\text{sgn}(\Delta_0 R_f(\mathbf{x}[X_1 \rightarrow 1])) = 1$ and $\text{sgn}(\Delta_0 R_f(\mathbf{x}[X_1 \rightarrow 0])) = 1$.

\mathbf{x}	[0 0]	[1 0]	[0 1]	[1 1]
$R_f(\mathbf{x})$	0	2	0	3

Table 5 – The ranks of one class of functions with asymmetric ordinal linkage.

Linkage between variables X_i and X_j present as either $\mathcal{L}_o(i, j)$ or $\mathcal{L}_o(j, i)$ will be considered present under the definition of non-monotonicity detection described earlier. However, some linkage will only be one way. Next we define the terminology we will use for this linkage.

We define the relationship between variables X_i and X_j as *interdependence*, which we represent algebraically as $X_i X_j$. Variables are interdependent if the ordinal linkage condition holds bi-directionally, as given by (64).

$$X_i X_j \Leftrightarrow \mathcal{L}_o(i, j) \wedge \mathcal{L}_o(j, i) \quad (64)$$

In contrast to non-linearity-based linkage detection, ordinal linkage may exist as an asymmetric relation. In the case of asymmetry we define the relationship as *dependence*, which we represent algebraically as $X_i \rightarrow X_j$ as given by (65).

$$X_i \rightarrow X_j \Leftrightarrow \mathcal{L}_o(i, j) \wedge \neg \mathcal{L}_o(j, i) \quad (65)$$

We define the relationship as *independence*, which we represent algebraically as $X_i + X_j$. Variables are independent if the ordinal linkage condition holds in neither direction, as given by (66).

$$X_i + X_j \Leftrightarrow \neg \mathcal{L}_o(i, j) \wedge \neg \mathcal{L}_o(j, i) \quad (66)$$

We use this notation concatenated into expressions for larger numbers of variables, using the order of operator precedence first $X_i X_j$, then $X_i \rightarrow X_j$, and lastly $X_i + X_j$, using left-to-right associativity.

For example, the expression $X_0 X_1 + X_2$ (67) means that X_0 and X_1 are interdependent on one another, and X_2 is independent of both of the former.

$$(X_0 X_1 + X_2) \Leftrightarrow (X_0 X_1) \wedge (X_0 + X_2) \wedge (X_1 + X_2) \quad (67)$$

In another example, the expression $(X_0 + X_1) \rightarrow X_2$ (68) means that X_0 and X_1 are independent from one another, and X_2 is dependent on both of the former.

$$((X_0 + X_1) \rightarrow X_2) \Leftrightarrow (X_0 + X_1) \wedge (X_0 \rightarrow X_2) \wedge (X_1 \rightarrow X_2) \quad (68)$$

All possible linkage between 3 variables can be written in this way as we will see in later chapters.

This notation cannot express all combinations of linkage, however. For example, the linkage between 4 variables described in (69) cannot be written linearly using this notation without repeating a variable.

$$\begin{aligned} & (X_0 \rightarrow X_1) \wedge (X_0 \rightarrow X_3) \wedge (X_2 \rightarrow X_3) \\ & \wedge (X_1 + X_2) \wedge (X_0 + X_2) \wedge (X_1 + X_3) \end{aligned} \quad (69)$$

4.3 Pseudo-Boolean Benchmarks Functions

In this section we give the complete definitions of the pseudo-Boolean benchmark functions referred to throughout the remainder of this thesis. We also state each benchmark's function values and Walsh coefficients for 2-bit and 3-bit instances as these are used in the following chapters.

4.3.1 Definitions and Identities

We make use of the following definitions and identities in this section. Let $\mathbf{0}_\ell$ denote a length 2^ℓ column vector populated by 0s, as given by (70).

$$\mathbf{0}_\ell = \left[\begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \right] 2^\ell \quad (70)$$

Let $\mathbf{1}_\ell$ denote a length 2^ℓ column vector populated by 1s, as given by (71).

$$\mathbf{1}_\ell = \left[\begin{array}{c} 1 \\ \vdots \\ 1 \end{array} \right] 2^\ell \quad (71)$$

Let δ_ℓ denote a length 2^ℓ column vector with 1 in the first position and the remainder populated by 0s, as given by (72).

$$\delta_\ell = \left[\begin{array}{c} 1 \\ 0 \\ \vdots \\ 0 \end{array} \right] 2^\ell \quad (72)$$

Multiplying H_ℓ by the $\mathbf{1}_\ell$ vector sums each column element for each row in the Hadamard matrix, since all but the first row contains an equal number of +1 as -1 , all but the first is zero, with the first being 1 times the number of columns (2^ℓ) hence (73).

$$H_\ell \mathbf{1}_\ell \equiv 2^\ell \delta_\ell \quad (73)$$

The delta vector selects the first element from each row of the Hadamard matrix (74).

$$H_\ell \delta_\ell \equiv \mathbf{1}_\ell \quad (74)$$

4.3.2 Constant Functions

The simplest example of a pseudo-Boolean function is one of the family of *constant functions* (CONST^ℓ). The constant function is defined as given by (75).

$$\text{CONST}_c^\ell(\mathbf{x}) = c \quad (75)$$

This function has an image consisting of a single element. Hence, every value is the global optimum with value c . An example is given by (76).

$$\text{CONST}_{42}^9([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0]) = 42 \quad (76)$$

The constant function is defined without using any variables, the linkage partition is given by (77).

$$\Gamma_{\text{CONST}_c^\ell} = \emptyset \quad (77)$$

The Walsh coefficients of the constant function are all 0, except the constant term, which is $\alpha_\emptyset = c$. Proof is given by the (78).

$$\begin{aligned} \alpha_\ell &= \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell \\ &= c \frac{1}{2^\ell} H_\ell \mathbf{1}_\ell \\ &= c \frac{1}{2^\ell} 2^\ell \boldsymbol{\delta}_\ell \quad (\text{from } H_\ell \mathbf{1}_\ell \equiv 2^\ell \boldsymbol{\delta}_\ell) \\ &= c \boldsymbol{\delta}_\ell \end{aligned} \quad (78)$$

4.3.3 Needle-in-a-Haystack Functions

In contrast to the constant function, the *needle-in-haystack function* (NEEDLE^ℓ) has only a single element of the domain being mapped to the global optimum. Thus, the image consists of two elements. The needle-in-haystack function is defined as given by (79).

$$\text{NEEDLE}^\ell(\mathbf{x}) = \prod_{i=0}^{\ell-1} x_i \quad (79)$$

This function maps the vector of all ones to the value 1, with every other input mapped to 0. Hence, for maximisation objective, the function's optimum is a vector of all ones, [1 1 ... 1]. An example is given by (80).

$$\text{NEEDLE}^9([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0]) = 0 \quad (80)$$

The needle-in-haystack function is fully connected, it is not an ASF as there is no additive separation of the variables, the linkage partition is given by (81).

$$\Gamma_{\text{NEEDLE}^\ell} = \{\{x_0, x_1, \dots, x_{\ell-1}\}\} \quad (81)$$

Every Walsh coefficient of the needle in haystack function is equal to $1/2^\ell$. Thus, this is an example of a function with complete structure. Proof is given by (82).

$$\begin{aligned} \alpha_\ell &= \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell \\ &= \frac{1}{2^\ell} H_\ell \boldsymbol{\delta}_\ell \\ &= \frac{1}{2^\ell} \mathbf{1}_\ell \quad (\text{from } H_\ell \boldsymbol{\delta}_\ell \equiv \mathbf{1}_\ell) \end{aligned} \quad (82)$$

Note that $\mathbf{1}_\ell$ is the first column of H_ℓ .

If a relabelling and/or permutation is applied to the function, the effect will be to map the global optimum to a different value in the search space. The vector of Walsh coefficients will then be a different column of H_ℓ , thus half of the coefficients will be equal to 1 and the other half will be equal to -1 .

4.3.4 Ones Function

The *ones function* (ONEMAX^ℓ) [66] [67] returns a sum of the input variables, equivalent to the count of the number of ones in the input. The definition is given by (83).

$$\text{ONEMAX}^\ell(\mathbf{x}) = \sum_{i=0}^{\ell-1} x_i \quad (83)$$

For maximisation objective, the function's optimum is a vector of all ones, $[1 \ 1 \ \dots \ 1]$. An example is given by (84).

$$\text{ONEMAX}^8([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0]) = 6 \quad (84)$$

Since the ones function consists of only univariate terms, it is fully separable, the linkage partition of the ones function is given by (85).

$$\Gamma_{\text{ONEMAX}^\ell} = \{\{x_0\}, \{x_1\}, \dots, \{x_{\ell-1}\}\} \quad (85)$$

The only non-zero Walsh coefficients in the ones function are the constant term $\alpha_\emptyset = \ell/2$ and all univariate terms $\alpha_{\{i\}} = 1/2$ ($\forall i \in \{0, \dots, \ell - 1\}$). Proof of coefficients is given on the following page.

Lemma 4.3.4: In ONEMAX^ℓ the constant term $(\forall \ell \in \{1, 2, \dots\}) (\alpha_{\emptyset|\ell} = \frac{\ell}{2})$, the univariate coefficients $(\forall \ell \in \{1, 2, \dots\})(\forall i \in \{0, \dots, \ell - 1\}) (\alpha_{\{i\}|\ell} = \frac{1}{2})$, and any other coefficient is 0.

Proof: by induction.

For $\ell = 1$, $\alpha_1 = \frac{1}{2^1} H_1 \mathbf{f}_1 = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$ thus $\alpha_{\emptyset|1} = \frac{1}{2} = \frac{\ell}{2}$, and $\alpha_{\{0\}|1} = \frac{1}{2}$, and no other coefficients exist.

Assume true for $\{1, 2, \dots, \ell - 1\}$, consider ℓ :

$$\begin{aligned}
\alpha_\ell &= \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell \\
&= \frac{1}{2^\ell} \begin{bmatrix} H_{\ell-1} & H_{\ell-1} \\ H_{\ell-1} & -H_{\ell-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\ell-1} + \mathbf{1}_{\ell-1} \\ \mathbf{f}_{\ell-1} \end{bmatrix} \quad \begin{array}{l} \text{(append "1" to } \mathbf{x}, \text{ fitness inc. by 1)} \\ \text{(append "0" to } \mathbf{x}) \end{array} \\
&= \frac{1}{2^\ell} \begin{bmatrix} 2H_{\ell-1} \mathbf{f}_{\ell-1} + H_{\ell-1} \mathbf{1}_{\ell-1} \\ H_{\ell-1} \mathbf{1}_{\ell-1} \end{bmatrix} \\
&= \frac{1}{2^\ell} \begin{bmatrix} 2H_{\ell-1} \mathbf{f}_{\ell-1} \\ H_{\ell-1} \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2^\ell} \begin{bmatrix} H_{\ell-1} \mathbf{1}_{\ell-1} \\ H_{\ell-1} \mathbf{1}_{\ell-1} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{2^{\ell-1}} H_{\ell-1} \mathbf{f}_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2^\ell} \begin{bmatrix} 2^{\ell-1} \delta_{\ell-1} \\ 2^{\ell-1} \delta_{\ell-1} \end{bmatrix} \quad (73) \\
&= \begin{bmatrix} \alpha_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \begin{bmatrix} 0.5 \delta_{\ell-1} \\ 0.5 \delta_{\ell-1} \end{bmatrix} \quad \begin{array}{l} \leftarrow \text{increments } \alpha_\emptyset \text{ by } 1/2 \\ \leftarrow \text{sets the new univariate term } \alpha_{\{\ell-1\}} = 1/2 \end{array}
\end{aligned}$$

By the induction hypothesis, $\alpha_{\emptyset|\ell-1} = \frac{\ell-1}{2}$, we derive $\alpha_{\emptyset|\ell} = \frac{\ell-1}{2} + \frac{1}{2} = \frac{\ell}{2}$, thus by the principal of induction, $(\forall \ell \in \{0, 1, \dots\}) (\alpha_{\emptyset|\ell} = \frac{\ell}{2})$.

The new univariate term $\alpha_{\{\ell-1\}}$ is set to $\frac{1}{2}$, and no other non-zero coefficients are added, and by the induction hypothesis all previous univariate terms are $\frac{1}{2}$ and no other previous coefficients (except α_\emptyset) are non-zero, thus by the principal of induction $(\forall \ell \in \{1, 2, \dots\})(\forall i \in \{0, 1, \dots, \ell - 1\}) (\alpha_{\{i\}|\ell} = \frac{1}{2})$ and no other Walsh coefficients (except α_\emptyset) are non-zero. ■

4.3.5 Zeros Function

The *zeros function* (ZEROMAX^ℓ) returns the count of the number of zeros in the input. This is a relabelling of the ONEMAX^ℓ function. The definition is given by (86).

$$\text{ZEROMAX}^\ell(\mathbf{x}) = \sum_{i=0}^{\ell-1} (1 - x_i) \quad (86)$$

For maximisation objective, the function's optimum is a vector of all zeros, $[0 \ 0 \ \dots \ 0]$. An example is given by (87).

$$\text{ZEROMAX}^8([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0]) = 2 \quad (87)$$

As a relabelling of the ONEMAX^ℓ , it has the same linkage partition as given by (88).

$$\Gamma_{\text{ZEROMAX}^\ell} = \{\{x_0\}, \{x_1\}, \dots, \{x_{\ell-1}\}\} \quad (88)$$

The only non-zero Walsh coefficients in the ones function are the constant term $\alpha_\emptyset = \ell/2$ and all univariate terms $\alpha_{\{i\}} = -1/2$ ($\forall i \in \{0, \dots, \ell - 1\}$). Proof of coefficients is given in (89).

$$\text{OneMax}^\ell(\mathbf{x}) = \alpha_\emptyset + \sum_{i=0}^{\ell-1} \alpha_{\{i\}} W_{\{i\}}(\mathbf{x}) = \frac{\ell}{2} + \frac{1}{2} \sum_{i=0}^{\ell-1} W_{\{i\}}(\mathbf{x})$$

$$\begin{aligned} \text{ZEROMAX}^\ell(\mathbf{x}) &= \ell - \text{OneMax}^\ell(\mathbf{x}) \\ &= \ell - \left(\frac{\ell}{2} + \frac{1}{2} \sum_{i=0}^{\ell-1} W_{\{i\}}(\mathbf{x}) \right) \\ &= \frac{\ell}{2} - \frac{1}{2} \sum_{i=0}^{\ell-1} W_{\{i\}}(\mathbf{x}) \quad \blacksquare \end{aligned} \quad (89)$$

4.3.6 Binary Value Function

The *binary value function* (BINVAL^ℓ) [67] is a univariate function which weights each variable exponentially, such that variable X_i has a weighting of 2^i . The binary value function is defined as given by (90).

$$\text{BINVAL}^\ell(\mathbf{x}) = \sum_{i=0}^{\ell-1} 2^i x_i \quad (90)$$

As with the ones function, for maximisation objective, the function's optimum is a vector of all ones, [1 1 ... 1]. The optimum value is $2^\ell - 1$, with variables of larger indices contributing a larger portion to this optimum value. An example is given by (91).

$$\text{BINVAL}^8([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0]) = 111 \quad (91)$$

Since the binary value function consists of only univariate terms, it is fully separable, the linkage partition of the binary value function is given by (92).

$$\Gamma_{\text{BINVAL}^\ell} = \{\{x_0\}, \{x_1\}, \dots, \{x_{\ell-1}\}\} \quad (92)$$

The only non-zero Walsh coefficients in the binary value function are the constant term $\alpha_\emptyset = \frac{2^\ell - 1}{2}$ and all univariate terms $\alpha_{\{i\}} = 2^{i-1}$ ($\forall i \in \{0, \dots, \ell - 1\}$). Proof of coefficients is given on the following page.

Lemma 4.3.6: In BINVAL^ℓ the constant term $(\forall \ell \in \{1, 2, \dots\}) (\alpha_{\emptyset|\ell} = \frac{2^\ell - 1}{2})$, the univariate coefficients $(\forall \ell \in \{1, 2, \dots\})(\forall i \in \{0, \dots, \ell - 1\})(\alpha_{\{i\}|\ell} = 2^{i-1})$, and any other coefficient is 0.

Proof, by induction:

For $\ell = 1$, $\alpha_1 = \frac{1}{2^1} H_1 \mathbf{f}_1 = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$ thus $\alpha_{\emptyset|1} = \frac{1}{2} = \frac{2^1 - 1}{2} = \frac{2^\ell - 1}{2}$, and $\alpha_{\{0\}|1} = \frac{1}{2} = 2^{0-1} = 2^{i-1}$ and no other coefficients exist.

Assume true for $\{1, 2, \dots, \ell - 1\}$, consider ℓ :

$$\begin{aligned}
 \alpha_\ell &= \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell \\
 &= \frac{1}{2^\ell} \begin{bmatrix} H_{\ell-1} & H_{\ell-1} \\ H_{\ell-1} & -H_{\ell-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\ell-1} + 2^{\ell-1} \mathbf{1}_{\ell-1} \\ \mathbf{f}_{\ell-1} \end{bmatrix} \quad \begin{array}{l} \text{(append "1" to } \mathbf{x}, \text{ fitness inc. by } 2^{\ell-1}) \\ \text{(append "0" to } \mathbf{x}) \end{array} \\
 &= \frac{1}{2^\ell} \begin{bmatrix} 2H_{\ell-1} \mathbf{f}_{\ell-1} + 2^{\ell-1} H_{\ell-1} \mathbf{1}_{\ell-1} \\ 2^{\ell-1} H_{\ell-1} \mathbf{1}_{\ell-1} \end{bmatrix} \\
 &= \frac{1}{2^\ell} \begin{bmatrix} 2H_{\ell-1} \mathbf{f}_{\ell-1} \\ H_{\ell-1} \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{2^{\ell-1}}{2^\ell} \begin{bmatrix} H_{\ell-1} \mathbf{1}_{\ell-1} \\ H_{\ell-1} \mathbf{1}_{\ell-1} \end{bmatrix} \\
 &= \begin{bmatrix} \frac{1}{2^{\ell-1}} H_{\ell-1} \mathbf{f}_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 2^{\ell-1} \boldsymbol{\delta}_{\ell-1} \\ 2^{\ell-1} \boldsymbol{\delta}_{\ell-1} \end{bmatrix} \quad (73) \\
 &= \begin{bmatrix} \alpha_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \begin{bmatrix} 2^{\ell-2} \boldsymbol{\delta}_{\ell-1} \\ 2^{\ell-2} \boldsymbol{\delta}_{\ell-1} \end{bmatrix} \quad \begin{array}{l} \leftarrow \text{increments } \alpha_\emptyset \text{ by } 2^{\ell-2} \\ \leftarrow \text{sets the new univariate term } \alpha_{\{i\}} = 2^{i-2}, \text{ where } i = \ell - 1 \end{array}
 \end{aligned}$$

By the induction hypothesis, $\alpha_{\emptyset|\ell-1} = \frac{2^{\ell-1} - 1}{2}$, we derive $\alpha_{\emptyset|\ell} = \frac{2^{\ell-1} - 1}{2} + 2^{\ell-2} = \frac{2^\ell - 1}{2}$, thus by the principal of induction, $(\forall \ell \in \{1, 2, \dots\}) (\alpha_{\emptyset|\ell} = \frac{2^\ell - 1}{2})$.

The new univariate term $\alpha_{\{\ell-1\}|\ell}$ is set to $2^{\ell-2} = 2^{(\ell-1)-1}$, and no other non-zero coefficients are added, and by the induction hypothesis, all previous univariate terms are 2^{i-1} and no other previous coefficients (except α_\emptyset) are non-zero, thus by the principal of induction $(\forall \ell \in \{1, 2, \dots\})(\forall i \in \{0, 1, \dots, \ell - 1\})(\alpha_{\{i\}|\ell} = 2^{i-1})$ and no other Walsh coefficients (except α_\emptyset) are non-zero. ■

4.3.7 1-Dimensional Checkerboard Function

The *1-dimensional checkerboard function* ($\text{CHECK}_{1\text{D}}^\ell$) [68] [47, pp. 32-32] counts the number of adjacent variables x_i, x_{i+1} such that $x_i \neq x_{i+1}$. The 1-dimensional checkerboard function is defined as given by (93).

$$\text{CHECK}_{1\text{D}}^\ell(\mathbf{x}) = \sum_{i=0}^{\ell-2} \begin{cases} 1, & x_i \neq x_{i+1} \\ 0, & x_i = x_{i+1} \end{cases} \quad (93)$$

For a maximization objective, this function has two global optima: $[1 \ 0 \ 1 \ 0 \ \dots]$ and $[0 \ 1 \ 0 \ 1 \ \dots]$ with an optimum value of $\ell - 1$. An example is given by (94).

$$\text{CHECK}_{1\text{D}}^\ell([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0]) = 3 \quad (94)$$

Since each variable in the 1-dimensional checkerboard function is connected to its adjacent neighbours in a chain, the function is not an ASF as there is no additive separation of the variables, the linkage partition is given by (95).

$$\Gamma_{\text{CHECK}_{1\text{D}}^\ell} = \{\{x_0, x_1, \dots, x_{\ell-1}\}\} \quad (95)$$

The only non-zero Walsh coefficients in the 1-dimensional checkerboard function are the constant term $\alpha_\emptyset = \frac{\ell-1}{2}$ and all *adjacent* bivariate terms $\alpha_{\{i, i+1\}} = -\frac{1}{2}$ ($\forall i \in \{0, \dots, \ell - 2\}$). Proof of coefficients is given on the following page.

Lemma 4.3.7: In CHECK_{1D}^ℓ the constant term ($\forall \ell \in \{2, 3, \dots\}$) ($\alpha_{\emptyset|\ell} = \frac{\ell-1}{2}$), the adjacent bivariate coefficients ($\forall \ell \in \{2, 3, \dots\}$) ($\forall i \in \{0, \dots, \ell-2\}$) ($\alpha_{\{i, i+1\}|\ell} = -\frac{1}{2}$), and any other coefficient is 0.

Proof, by induction:

For $\ell = 1, 2$, by direct calculation:

$$\alpha_1 = \begin{bmatrix} \alpha_{\emptyset} \\ \alpha_{\{0\}} \end{bmatrix} = \frac{1}{2^1} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\alpha_2 = \begin{bmatrix} \alpha_{\emptyset} \\ \alpha_{\{0\}} \\ \alpha_{\{1\}} \\ \alpha_{\{0,1\}} \end{bmatrix} = \frac{1}{2^2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 0 \\ 0 \\ -1/2 \end{bmatrix}$$

Assume true for $\{2, 3, \dots, \ell-1\}$, consider ℓ :

$$\begin{aligned} \alpha_\ell &= \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell \\ &= \frac{1}{2^\ell} \begin{bmatrix} H_{\ell-2} & H_{\ell-2} & H_{\ell-2} & H_{\ell-2} \\ H_{\ell-2} & -H_{\ell-2} & H_{\ell-2} & -H_{\ell-2} \\ H_{\ell-2} & H_{\ell-2} & -H_{\ell-2} & -H_{\ell-2} \\ H_{\ell-2} & -H_{\ell-2} & -H_{\ell-2} & H_{\ell-2} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\ell-1} + \begin{bmatrix} \mathbf{0}_{\ell-2} \\ \mathbf{1}_{\ell-2} \end{bmatrix} \\ \mathbf{f}_{\ell-1} + \begin{bmatrix} \mathbf{1}_{\ell-2} \\ \mathbf{0}_{\ell-2} \end{bmatrix} \end{bmatrix} \begin{array}{l} (\mathbf{x} \text{ ends } [\dots, 1, 1]) \\ (\mathbf{x} \text{ ends } [\dots, 0, 1], \text{ fitness } + 1) \\ (\mathbf{x} \text{ ends } [\dots, 1, 0], \text{ fitness } + 1) \\ (\mathbf{x} \text{ ends } [\dots, 0, 0]) \end{array} \\ &= \frac{1}{2^\ell} \begin{bmatrix} 2H_{\ell-1} \mathbf{f}_{\ell-1} \\ H_{\ell-1} \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2^\ell} \begin{bmatrix} H_{\ell-2} & H_{\ell-2} & H_{\ell-2} & H_{\ell-2} \\ H_{\ell-2} & -H_{\ell-2} & H_{\ell-2} & -H_{\ell-2} \\ H_{\ell-2} & H_{\ell-2} & -H_{\ell-2} & -H_{\ell-2} \\ H_{\ell-2} & -H_{\ell-2} & -H_{\ell-2} & H_{\ell-2} \end{bmatrix} \begin{bmatrix} \mathbf{0}_{\ell-2} \\ \mathbf{1}_{\ell-2} \\ \mathbf{1}_{\ell-2} \\ \mathbf{0}_{\ell-2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2^{\ell-1}} H_{\ell-1} \mathbf{f}_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2^\ell} \begin{bmatrix} 2 \times 2^{\ell-2} \boldsymbol{\delta}_{\ell-2} \\ \mathbf{0}_{\ell-2} \\ \mathbf{0}_{\ell-2} \\ -2 \times 2^{\ell-2} \boldsymbol{\delta}_{\ell-2} \end{bmatrix} \\ &= \begin{bmatrix} \alpha_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \boldsymbol{\delta}_{\ell-2} \\ \mathbf{0}_{\ell-2} \\ \mathbf{0}_{\ell-2} \\ -\boldsymbol{\delta}_{\ell-2} \end{bmatrix} \begin{array}{l} \leftarrow \text{increments } \alpha_{\emptyset} \text{ by } 0.5 \\ \leftarrow \text{sets the new bivariate term } \alpha_{\{i, i+1\}} = -0.5, \text{ where } i = \ell - 2 \end{array} \end{aligned}$$

By the induction hypothesis, $\alpha_{\emptyset|\ell-1} = \frac{(\ell-1)-1}{2}$, we derive $\alpha_{\emptyset|\ell} = \frac{(\ell-1)-1}{2} + \frac{1}{2} = \frac{\ell-1}{2}$, thus by the principal of induction ($\forall \ell \in [2, 3, \dots]$) ($\alpha_{\emptyset|\ell} = \frac{\ell-1}{2}$).

The new bivariate term is set to -0.5 , and no other non-zero coefficients are added, and by the induction hypothesis, all previous bivariate terms are -0.5 and no other previous coefficients (except α_{\emptyset}) are non-zero, thus by the principal of induction ($\forall i \in [0, \ell-2]$) ($\alpha_{\{i, i+1\}} = -0.5$) in all $\ell \geq 2$ and no other coefficients (except α_{\emptyset}) are non-zero. ■

4.3.8 Leading-Ones Function

The *leading-ones function* (LEADING^ℓ) [67] is the total number of ones in the function until the first instance of a zero. In other words, the index of the lowest-indexed zero, or ℓ if there are no zeros. The leading ones function is defined as given by (96).

$$\text{LEADING}^\ell(\mathbf{x}) = \sum_{i=0}^{\ell-1} \prod_{j=0}^{i-1} x_j \quad (96)$$

As with the ones function, for maximisation objective, the function's optimum is a vector of all ones, [1 1 ... 1]. The optimum value is ℓ . An example is given by (97).

$$\text{LEADING}^8([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0]) = 4 \quad (97)$$

The leading-ones function is fully connected, it is not an ASF as there is no additive separation of the variables, the linkage partition is given by (98).

$$\Gamma_{\text{LEADING}^\ell} = \{\{x_0, x_1, \dots, x_{\ell-1}\}\} \quad (98)$$

The constant term $\alpha_\emptyset = \frac{2^\ell - 1}{2^\ell}$, all other coefficients are non-zero, $\alpha_k = \frac{2^\ell - 1}{2^\ell} - \frac{2^m - 1}{2^m}$ where m is the highest index in clique k . Note that if $m = \ell - 1$ (the bottom half of α), the above equation simplifies to $\alpha_k = \frac{1}{2^\ell}$. Proof of coefficients is given on the following page.

Lemma 4.3.8: In LEADING^ℓ the constant term ($\forall \ell \in \{1, 2, \dots\}$) ($\alpha_{\emptyset|\ell} = \frac{2^\ell - 1}{2^\ell}$), and all other coefficients are ($\forall \ell \in \{1, 2, \dots\}$) ($\forall k \subseteq \{0, \dots, \ell - 1\}$) ($\alpha_{k|\ell} = \frac{2^\ell - 1}{2^\ell} - \frac{2^m - 1}{2^m}$), where m is the highest index in the clique k ,

Proof, by induction:

$$\text{For } \ell = 1, \quad \alpha_0 = \frac{1}{2^1} H_1 \mathbf{f}_1 = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}, \quad \text{thus } \alpha_{\emptyset|1} = \frac{2-1}{2} = \frac{2^1-1}{2^1} = \frac{2^\ell-1}{2^\ell} \quad \text{and}$$

$$\alpha_{\{0\}|1} = \frac{1}{2} = \frac{1}{2} - \frac{0}{2} = \frac{2-1}{2} - \frac{1-1}{1} = \frac{2^1-1}{2^1} - \frac{2^0-1}{2^0} = \frac{2^\ell-1}{2^\ell} - \frac{2^m-1}{2^m}.$$

Assume true for $\{1, 2, \dots, \ell - 1\}$, consider ℓ :

$$\begin{aligned} \alpha_\ell &= \frac{1}{2^\ell} H_\ell \mathbf{f}_\ell \\ &= \frac{1}{2^\ell} \begin{bmatrix} H_{\ell-1} & H_{\ell-1} \\ H_{\ell-1} & -H_{\ell-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\ell-1} + \boldsymbol{\delta}_{\ell-1} \\ \mathbf{f}_{\ell-1} \end{bmatrix} \quad \begin{array}{l} \text{(append "1" to } \mathbf{x}, \text{ fitness + 1 if } \mathbf{x} = [1 \dots 1]) \\ \text{(append "0" to } \mathbf{x}) \end{array} \\ &= \frac{1}{2^\ell} \begin{bmatrix} 2H_{\ell-1}\mathbf{f}_{\ell-1} + H_{\ell-1}\boldsymbol{\delta}_{\ell-1} \\ H_{\ell-1}\boldsymbol{\delta}_{\ell-1} \end{bmatrix} \\ &= \frac{1}{2^\ell} \begin{bmatrix} 2H_{\ell-1}\mathbf{f}_{\ell-1} \\ H_{\ell-1}\mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2^\ell} \begin{bmatrix} H_{\ell-1}\boldsymbol{\delta}_{\ell-1} \\ H_{\ell-1}\boldsymbol{\delta}_{\ell-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2^{\ell-1}} H_{\ell-1} \mathbf{f}_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2^\ell} \begin{bmatrix} \mathbf{1}_{\ell-1} \\ \mathbf{1}_{\ell-1} \end{bmatrix} \\ &= \begin{bmatrix} \alpha_{\ell-1} \\ \mathbf{0}_{\ell-1} \end{bmatrix} + \frac{1}{2^\ell} \begin{bmatrix} \mathbf{1}_{\ell-1} \\ \mathbf{1}_{\ell-1} \end{bmatrix} \quad \leftarrow \text{adds } \frac{1}{2^\ell} \text{ to every coefficient} \end{aligned}$$

Adding $\frac{1}{2^\ell}$ to every coefficient, we can derive an expression for $\alpha_{\emptyset|\ell}$ and $\alpha_{k|\ell}$:

$$\alpha_{\emptyset|\ell} = \alpha_{\emptyset|\ell-1} + \frac{1}{2^\ell} = \frac{2^{\ell-1} - 1}{2^{\ell-1}} + \frac{1}{2^\ell} = \frac{2(2^{\ell-1} - 1)}{2 \times 2^{\ell-1}} + \frac{1}{2^\ell} = \frac{2^\ell - 2}{2^\ell} + \frac{1}{2^\ell} = \frac{2^\ell - 1}{2^\ell}$$

$$\alpha_{k|\ell} = \alpha_{k|\ell-1} + \frac{1}{2^\ell} = \frac{2^{\ell-1} - 1}{2^{\ell-1}} - \frac{2^m - 1}{2^m} + \frac{1}{2^\ell} = \frac{2^\ell - 2}{2^\ell} - \frac{2^m - 1}{2^m} + \frac{1}{2^\ell} = \frac{2^\ell - 1}{2^\ell} - \frac{2^m - 1}{2^m}$$

Thus by the principle of induction the expressions hold for all $\ell \in \{1, 2, \dots\}$. ■

4.3.9 Order-k Trap Function

The *order-k trap function* (TRAP_k) is designed to be deceptive [71] [72]. Deceptive functions lead an optimisation algorithm away from the optimum. An example is given by (99) for a trap size of $k = 4$, and length $\ell = 8$ (two concatenated traps).

$$\begin{aligned} \text{TRAP}_4^8([1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0]) &= g_4(4) + g_4(2) \\ &= 4 + 1 \\ &= 5 \end{aligned} \tag{ 99 }$$

The trap function is typically of a small order k , such that k divides ℓ , and concatenated, i.e. the first k bits form the first trap, the next k bits form the next trap as given by (100).

$$\text{TRAP}_k^\ell(\mathbf{x}) = \sum_{i=0}^{\ell/k} g_k \left(\sum_{j=0}^{k-1} x_{ik+j} \right) \tag{ 100 }$$

We define the trap sub function as given by (101), adapted from Deb et. al. [69] (as cited by Cantú-Paz et. al. [105])

$$g_k(u) = \begin{cases} k - u - 1, & u < k \\ k, & u = k \end{cases} \tag{ 101 }$$

Alternatively, functions can be defined by concatenating trap functions of different orders k .

As shown by the Walsh decomposition, where there are low-order cliques with non-overlapping sets of the variables, such as in the concatenated trap, this is an ASF. Thus, the concatenated k -trap function is an example of an additively separable function. The function is split in to $\frac{\ell}{k}$ functions of length k . The linkage partition for the k -trap is given by (102).

$$\begin{aligned} \Gamma_{\text{TRAP}_k^\ell} &= \{\gamma_0, \gamma_1, \dots, \gamma_{\ell/k-1}\} \\ \text{where } \gamma_i &= \{X_{ki}, \dots, X_{k(i+1)-1}\} \end{aligned} \tag{ 102 }$$

We state the Walsh coefficients for the order-2, order-3, and order-4 traps in Table 6, Table 7, and Table 8 respectively. These show the coefficients for trap number i . The constant term of the overall function is the sum of constant terms from individual traps.

α_\emptyset	$\alpha_{\{ik\}}$	$\alpha_{\{ik+1\}}$	$\alpha_{\{ik,ik+1\}}$
$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$

Table 6 – Walsh coefficients for trap i of a concatenated order-2 trap function.

α_\emptyset	$\alpha_{\{ik\}}$	$\alpha_{\{ik+1\}}$	$\alpha_{\{ik,ik+1\}}$
1	0	0	$\frac{1}{2}$
$\alpha_{\{ik+2\}}$	$\alpha_{\{ik,ik+2\}}$	$\alpha_{\{ik+1,ik+2\}}$	$\alpha_{\{ik,ik+1,ik+2\}}$
0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

Table 7 – Walsh coefficients for trap i of a concatenated order-3 trap function.

α_\emptyset	$\alpha_{\{ik\}}$	$\alpha_{\{ik+1\}}$	$\alpha_{\{ik,ik+1\}}$
$\frac{21}{16}$	$-\frac{3}{16}$	$-\frac{3}{16}$	$\frac{5}{16}$
$\alpha_{\{ik+2\}}$	$\alpha_{\{ik,ik+2\}}$	$\alpha_{\{ik+1,ik+2\}}$	$\alpha_{\{ik,ik+1,ik+2\}}$
$-\frac{3}{16}$	$\frac{5}{16}$	$\frac{5}{16}$	$\frac{5}{16}$
$\alpha_{\{ik+3\}}$	$\alpha_{\{ik,ik+3\}}$	$\alpha_{\{ik+1,ik+3\}}$	$\alpha_{\{ik,ik+1,ik+3\}}$
$-\frac{3}{16}$	$\frac{5}{16}$	$\frac{5}{16}$	$\frac{5}{16}$
$\alpha_{\{ik+2,ik+3\}}$	$\alpha_{\{ik,ik+2,ik+3\}}$	$\alpha_{\{ik+1,ik+2,ik+3\}}$	$\alpha_{\{ik,ik+1,ik+2,ik+3\}}$
$\frac{5}{16}$	$\frac{5}{16}$	$\frac{5}{16}$	$\frac{5}{16}$

Table 8 – Walsh coefficients for trap i of a concatenated order-4 trap function.

In the case of the TRAP_k function, we only use the order-2 and order-3 traps in the remainder of this work, so will simply state the derivations of the order-2 and order-3 traps here. The coefficients for order-4 and above are derived similarly.

The Walsh coefficients TRAP_2^2 are derived as shown by (103).

$$\alpha = \frac{1}{2^2} H\mathbf{f} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 3 \\ 1 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 3/4 \\ 1/4 \\ 1/4 \\ 3/4 \end{bmatrix} \quad (103)$$

The Walsh coefficients TRAP_3^3 are derived as shown by (104).

$$\alpha = \frac{1}{2^3} H\mathbf{f} = \frac{1}{8} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 8 \\ 0 \\ 0 \\ 4 \\ 0 \\ 4 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1/2 \\ 0 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} \quad (104)$$

It should be noted that in practice in the literature, order-4 and order-5 instances of the TRAP_k function are usually used concatenated into a longer problem as additively separate sub-functions, and are not used as arbitrarily-long order traps.

4.3.10 Goldberg's Fully-Deceptive Order-3 Function

Goldberg's fully-deceptive order-3 function (GOLDBERG) [71] [72] [25] is a 3-bit function with the following values given by Table 9, and the Walsh coefficients given by Table 10.

f_{111}	f_{011}	f_{101}	f_{001}	f_{110}	f_{010}	f_{100}	f_{000}
30	0	0	14	0	22	26	28

Table 9 – Function values for Goldberg's fully-deceptive order-3 function.

α_{\emptyset}	$\alpha_{\{0\}}$	$\alpha_{\{1\}}$	$\alpha_{\{0,1\}}$	$\alpha_{\{2\}}$	$\alpha_{\{0,2\}}$	$\alpha_{\{1,2\}}$	$\alpha_{\{0,1,2\}}$
15	-1	-2	3	-4	5	6	8

Table 10 – Walsh coefficients for Goldberg's fully-deceptive order-3 function.

It should be noted that using Goldberg's convention for the Walsh functions, the coefficients would all be of the same magnitude as above, although all positive except $\alpha_{\{0,1,2\}} = -8$. The coefficients specified are adapted for our convention.

The function GOLDBERG is defined over 3 bits. We present proof of coefficients by directly calculating from the Walsh-Hadamard transform as given by (105).

$$\begin{aligned}
 \alpha &= \frac{1}{2^3} Hf \\
 &= \frac{1}{8} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 30 \\ 0 \\ 0 \\ 14 \\ 0 \\ 22 \\ 26 \\ 28 \end{bmatrix} \\
 &= \frac{1}{8} \begin{bmatrix} 120 \\ -8 \\ -16 \\ 24 \\ -32 \\ 40 \\ 48 \\ 64 \end{bmatrix} = \begin{bmatrix} 15 \\ -1 \\ -2 \\ 3 \\ -4 \\ 5 \\ 6 \\ 8 \end{bmatrix} \tag{ 105 }
 \end{aligned}$$

4.3.11 2-Bit and 3-Bit Function Values and Walsh Coefficients

As 2-bit and 3-bit functions will be used heavily in this thesis, we present the function values for the selected benchmark functions in the case of $\ell = 2$ and $\ell = 3$.

Function	f_{11}	f_{01}	f_{10}	f_{00}
CONST _c ²	c	c	c	c
ONEMAX ²	2	1	1	0
ZEROMAX ²	0	1	1	2
BINVAL ²	3	2	1	0
CHECK _{1D} ²	0	1	1	0
NEEDLE ²	1	0	0	0
LEADING ²	2	0	1	0
TRAP ₂ ²	2	0	0	1

Table 11 – Function values for benchmark instances in 2-bits.

Function	R_f^{11}	R_f^{01}	R_f^{10}	R_f^{00}
CONST _c ²	0	0	0	0
ONEMAX ²	3	1	1	0
ZEROMAX ²	0	1	1	3
BINVAL ²	3	2	1	0
CHECK _{1D} ²	0	2	2	0
NEEDLE ²	3	0	0	0
LEADING ²	3	0	2	0
TRAP ₂ ²	3	0	0	2

Table 12 – Ranks for benchmark instances in 2-bits.

Here we group the functions by conventional description of complexity and present the Walsh coefficients. In a later section we will return to view the complexity of these benchmarks based on perturbation and ordinal linkage.

Function	α_{\emptyset}	$\alpha_{\{0\}}$	$\alpha_{\{1\}}$	$\alpha_{\{0,1\}}$
Zero-Dimensional				
CONST _c ²	c	0	0	0
Univariate				
ONEMAX ²	2	$\frac{1}{2}$	$\frac{1}{2}$	0
ZEROMAX ²	2	$-\frac{1}{2}$	$-\frac{1}{2}$	0
BINVAL ²	$1\frac{1}{2}$	$\frac{1}{2}$	1	0
Bivariate/Multivariate				
CHECK _{1D} ²	$\frac{1}{2}$	0	0	$-\frac{1}{2}$
NEEDLE ²	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
LEADING ²	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
TRAP ₂ ²	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$

Table 13 – Walsh coefficients of benchmark instances in 2-bits.

Function	f_{111}	f_{011}	f_{101}	f_{001}	f_{110}	f_{010}	f_{100}	f_{000}
CONST _c ³	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>
ONEMAX ³	3	2	2	1	2	1	1	0
ZEROMAX ³	0	1	1	2	1	2	2	3
BINVAL ³	7	6	5	4	3	2	1	0
CHECK _{1D} ³	0	1	2	1	1	2	1	0
NEEDLE ³	1	0	0	0	0	0	0	0
LEADING ³	3	0	1	0	2	0	1	0
TRAP ₃ ³	3	0	0	1	0	1	1	2
GOLDBERG	30	0	0	14	0	22	26	28

Table 14 – Function values for benchmark instances in 3-bits.

Function	R_f^{111}	R_f^{011}	R_f^{101}	R_f^{001}	R_f^{110}	R_f^{010}	R_f^{100}	R_f^{000}
CONST _c ³	0	0	0	0	0	0	0	0
ONEMAX ³	7	4	4	1	4	1	1	0
ZEROMAX ³	0	1	1	4	1	4	4	7
BINVAL ³	7	6	5	4	3	2	1	0
CHECK _{1D} ³	0	2	6	2	2	6	2	0
NEEDLE ³	7	0	0	0	0	0	0	0
LEADING ³	7	0	4	0	6	0	4	0
TRAP ₃ ³	7	0	0	3	0	3	3	6
GOLDBERG	7	0	0	3	0	4	5	6

Table 15 – Ranks for benchmark instances in 3-bits.

Again, we group the functions by conventional description of complexity and present the Walsh coefficients.

Function	α_{\emptyset}	$\alpha_{\{0\}}$	$\alpha_{\{1\}}$	$\alpha_{\{0,1\}}$	$\alpha_{\{2\}}$	$\alpha_{\{0,2\}}$	$\alpha_{\{1,2\}}$	$\alpha_{\{0,1,2\}}$
Zero-Dimensional								
CONST _c ³	c	0	0	0	0	0	0	0
Univariate								
ONEMAX ³	$\frac{3}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0
ZEROMAX ³	$\frac{3}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0
BINVAL ³	$\frac{7}{2}$	$\frac{1}{2}$	1	0	2	0	0	0
Bivariate								
CHECK _{1D} ³	1	0	0	$-\frac{1}{2}$	0	0	$-\frac{1}{2}$	0
Multivariate								
NEEDLE ³	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
LEADING ³	$\frac{7}{8}$	$\frac{7}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
TRAP ₃ ³	1	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
GOLDBERG	15	-1	-2	3	-4	5	6	8

Table 16 – Walsh coefficients of benchmark instances in 3-bits.

4.4 1-Bit Pseudo-Boolean Functions

For 1-bit, every function is a member of one of three classes. The classes are listed in Table 17.

We compute the Walsh coefficients of a 1-bit function as shown in (106)

$$\boldsymbol{\alpha} = \frac{1}{2^0} H_0 \mathbf{f}$$

$$\begin{bmatrix} \alpha_{\emptyset} \\ \alpha_{\{0\}} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_0 \end{bmatrix} \quad (106)$$

$$\begin{bmatrix} \alpha_{\emptyset} \\ \alpha_{\{0\}} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(f_1 + f_0) \\ \frac{1}{2}(f_1 - f_0) \end{bmatrix}$$

As in the general case – the constant term α_{\emptyset} is the mean value of the function. The other coefficient $\alpha_{\{0\}}$ is half of the difference between the function values. This is zero in the case of a constant function, positive if $f_0 < f_1$, and negative if $f_0 > f_1$.

If $f_0 = f_1$, the linkage partition class is $\Gamma = \emptyset$, otherwise it is $\Gamma = \{\{X_0\}\}$.

The three classes for 1-bit functions are summarised in Table 17.

Class	Condition	$\alpha_{\{0\}}$	Linkage	Example
[0 0]	$f_0 = f_1$	zero	\emptyset	CONST ²
[1 0]	$f_0 < f_1$	positive	$\{\{X_0\}\}$	ONEMAX ²
[0 1]	$f_0 > f_1$	negative	$\{\{X_0\}\}$	ZEROMAX ²

Table 17 – The number of function classes in 1-bit for a given number of distinct fitness levels (number of ranks). There are 3 distinct classes for 1-bit functions.

4.5 Counting Function Classes

In this section, we discuss counting the number of classes for a given problem length. This motivates our focus on 2-bit and 3-bit problems for the following survey.

An *injective function* is one which has a one-to-one mapping of elements of the domain to elements of the image, i.e. no two bit strings have the same fitness. The number of injective function classes is given by the factorial $2^\ell!$. To calculate the total number of classes including non-injective functions we define a function $t(\ell)$, for a given bit string length ℓ , this gives the total number of function classes for a given length. The function $t(\ell)$ is derived by summing over the number of classes for a given number of ranks, n , which we will express as a function, $c(n, \ell)$.

For bit string length ℓ , a class C is a vector of 2^ℓ elements. For n ranks, each element is an integer from 0 to $n - 1$ inclusive. The number of possibilities is bounded above by n^{2^ℓ} . At each value for n we must consider that some vector permutations will contain fewer than n distinct ranks, e.g. for $n = 2$, the vector $[1 \ 1 \ \dots \ 1]$ does not represent a valid class as it does not contain exactly 2 distinct ranks. The class for the constant function is the class $[0 \ 0 \ \dots \ 0]$. To avoid counting invalid vectors, we iteratively subtract for each $k < n$ the number of classes at this number of ranks: $c(k, \ell)$ multiplied by the number of ways in which they can be arranged in n ranks, which is the binomial coefficient, or the number of ways to choose k elements from n , written C_k^n . The number of classes for n ranks for problem length ℓ is given by (107).

$$c(n, \ell) = n^{2^\ell} - \sum_{k=1}^{n-1} c(k, \ell) C_k^n \quad (107)$$

$$\text{where } C_k^n = \frac{n!}{k!(n-k)!}$$

Note that the base case of this recursive definition is implicit, since when $n = 1$, the sum is empty over the range $\sum_{\emptyset} \dots = 0$ hence, $c(1, \ell) = 1^{2^\ell} + 0 = 1$.

The number of distinct ranks n can vary from 1 to 2^ℓ . Summing the value of $c(n, \ell)$ for the cases $1 \leq n \leq 2^\ell$ (108, p. 78) gives the total number of classes $t(\ell)$ for a specified problem length ℓ , by summing over $c(n, \ell)$ for values $1 \leq n \leq 2^\ell$.

$$t(\ell) = \sum_{n=1}^{2^\ell} c(n, \ell) \quad (108)$$

For comparison, values for $t(\ell)$ have been given in Table 18 for $0 \leq \ell \leq 8$.

Length, ℓ	Solution Space, 2^ℓ	Injective Classes, $2^\ell!$	All Classes, $t(\ell)$
0	1	1	1
1	2	2	3
2	4	24	75
3	8	40 320	545 835
4	16	20 922 789 888 000	5 315 654 681 981 355
5	32	$\sim 2.631 \times 10^{35}$	$\sim 2.355 \times 10^{40}$
6	64	$\sim 1.269 \times 10^{89}$	$\sim 1.408 \times 10^{99}$
7	128	$\sim 3.856 \times 10^{215}$	$\sim 6.586 \times 10^{235}$
8	256	$\sim 8.578 \times 10^{506}$	$\sim 3.469 \times 10^{547}$

Table 18 – Number of injective function classes and total number of function classes for a given length of pseudo-Boolean functions.

As the number of classes grows superexponentially with the length of the problem (see Table 18), the survey of function classes looks at 3-bits as this is the largest set which remains amenable to exhaustive survey. We will first discuss 2-bit classes, and then apply the same methods to 3-bit classes. The approach is computationally intractable as ℓ increases, however, we will be able to draw conclusions from the survey of 2-bit and 3-bit classes which are general for ℓ -dimensions.

4.6 Summary

In this chapter we have given our definitions for rank equivalence and directed ordinal linkage, based on function classes invariant under monotonic operators. We also defined the benchmark functions we will use throughout the thesis, and summarised the small space of $\{0, 1\} \rightarrow \mathbb{R}$ functions (1-bit pseudo-Boolean functions). In the following chapters we will analyse 2-bit and 3-bit spaces similarly.

5 2-Bit Pseudo-Boolean Functions

In this chapter we explore the infinite set of pseudo-Boolean functions in 2-dimensions by using a finite set of equivalence classes. This chapter provides a detailed description of the aforementioned space of classes which we refer to as *2 bit classes*. We completely determine the possible and minimal Walsh structures of these classes and the precedence profiles and connect the Walsh structures with ordinal linkage, then we discuss algorithmic steps to solve function classes. Following chapters will extend the work into higher dimensions.

5.1 Counting 2-Bit Classes

A 2-bit equivalence class is completely determined by specifying ranks for the 4 possible values of \mathbf{x} . Previously stated is the general definition of a class vector (62, p. 50). For convenience we restate the specific case for 2-bit pseudo-Boolean functions as (109).

$$\mathbf{C}_f = [R_f^{11} \ R_f^{01} \ R_f^{10} \ R_f^{00}]$$

$$\text{where } R_f^{x_0x_1} = R_f([x_0 \ x_1]) \quad (109)$$

$$\text{and } R_f(\mathbf{x}) = |\{\mathbf{y}: \mathbf{y} \in \{0, 1\}^2 \wedge f(\mathbf{y}) < f(\mathbf{x})\}|$$

Table 19 shows the number of 2-bit function classes with a breakdown of number of classes for each valid number of ranks. The number of classes for 4 ranks (24) is the number of injective function classes.

Num. Ranks, n	Num. Classes, $c(n, 2)$
1	1
2	14
3	36
4	24
Total	75

Table 19 – The number of function classes in 2-bits for a given number of distinct fitness levels (number of ranks). There are 75 distinct classes for 2-bit functions.

5.2 Walsh Families and Delta Conditions

As an example, take the class [3 1 0 1]. This class corresponds to the infinite set of functions whose values are laid out on the real number line as shown in Figure 8.

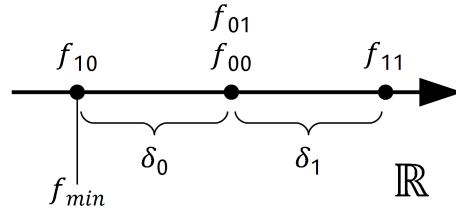


Figure 8 – Mapping fitness levels of the general instance of class [3 1 0 1] to the real number line \mathbb{R} .

The values of f_{min} , δ_0 , and δ_1 specify an *instance* of the class. For example, the function $\mathbf{f} = [9.0 \ -0.7 \ -6.1 \ -0.7]^T$ is an instance defined by $f_{min} = -6.1$, $\delta_0 = 5.4$, and $\delta_1 = 9.7$. It should be noted that while any function values, including f_{min} may be negative or zero, the values for δ_i are always strictly positive.

By stating the fitness values in the fitness vector \mathbf{f} in terms of f_{min} , δ_0 , and δ_1 we get this class' delta expansion as given by (110).

$$\begin{bmatrix} f_{11} \\ f_{01} \\ f_{10} \\ f_{00} \end{bmatrix} = \begin{bmatrix} f_{min} + \delta_0 + \delta_1 \\ f_{min} + \delta_0 \\ f_{min} \\ f_{min} + \delta_0 \end{bmatrix} \quad (110)$$

Recall the expression for the Walsh-Hadamard transform is $\alpha = \frac{1}{2^t} H \mathbf{f}$ (43, p. 38). Here we substitute the values of the fitness vector with the delta expansion for this class as given by (111).

$$\begin{bmatrix} \alpha_{\emptyset} \\ \alpha_{\{0\}} \\ \alpha_{\{1\}} \\ \alpha_{\{0,1\}} \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_{min} + \delta_0 + \delta_1 \\ f_{min} + \delta_0 \\ f_{min} \\ f_{min} + \delta_0 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 4\bar{f} \\ \delta_1 - \delta_0 \\ \delta_1 + \delta_0 \\ \delta_1 + \delta_0 \end{bmatrix} \quad (111)$$

The first coefficient, α_{\emptyset} is the arithmetic mean of the four values. We do not consider this constant term to be part of the structure, since $\alpha_{\emptyset} \neq 0$ is never a necessary condition for preserving the ranks, it simply translates all function values along the real number line.

The second coefficient, $\alpha_{\{0\}}$ is given by the fitness vector times one quarter the second row of the Hadamard matrix. By substituting the delta expansion of the fitnesses as in (112) below.

$$\begin{aligned} \alpha_{\{0\}} &= \frac{1}{4}(f_{min} + \delta_0 + \delta_1 - f_{min} - \delta_0 + f_{min} - f_{min} - \delta_0) \\ &= \frac{1}{4}(\delta_1 - \delta_0) \end{aligned} \tag{ 112 }$$

Here we show that $\alpha_{\{0\}} = \frac{1}{4}(\delta_1 - \delta_0)$. Hence, $\alpha_{\{0\}}$ is zero if and only if $\delta_0 = \delta_1$. This element of the structure may be present but is *unnecessary*, since there exists an instance of the class where it is zero. We refer to the expression $\delta_0 = \delta_1$ as this coefficient's *delta condition*.

There is no valid (positive) assignment of values to δ_0 and δ_1 which makes $\frac{1}{4}(\delta_0 + \delta_1)$ zero, hence we say that it is necessary, since it is non-zero in all instances of the class. No delta condition exists for $\alpha_{\{1\}}$ or $\alpha_{\{0,1\}}$ in this example class.

Returning to the general case; ignoring the constant term, there are 3 coefficients in the Walsh expansion. Considering each to be either zero or non-zero, there are 8 combinations. We enumerate these possibilities using the IDs 0 to 7 as given by Table 21 (p. 82).

We give an ID (which is a power of two) and pictorial representation to each of the possible non-zero Walsh coefficients. The ID of a structure is the sum of the IDs of the individual non-zero Walsh coefficients. The symbols and IDs are given in Table 20.




Symbol and ID			
	1	2	4
Non-Zero Coefficients	$\alpha_{\{0\}}$	$\alpha_{\{1\}}$	$\alpha_{\{0,1\}}$

Table 20 – Symbols and IDs for possible non-zero Walsh coefficients for 2-bit.




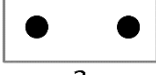
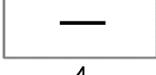
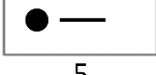
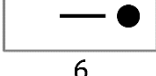
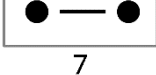
ID	Symbol	Non-Zero Coefficients	Description
0		$\{\}$	All coefficients zero.
1		$\{\alpha_{\{0\}}\}$	Only univariate coefficient 0 non-zero.
2		$\{\alpha_{\{1\}}\}$	Only univariate coefficient 1 non-zero.
3		$\{\alpha_{\{0\}}, \alpha_{\{1\}}\}$	Bivariate coefficient zero.
4		$\{\alpha_{\{0,1\}}\}$	Only bivariate coefficient non-zero
5		$\{\alpha_{\{0\}}, \alpha_{\{0,1\}}\}$	Univariate coefficient 0 zero.
6		$\{\alpha_{\{1\}}, \alpha_{\{0,1\}}\}$	Univariate coefficient 1 zero.
7		$\{\alpha_{\{0\}}, \alpha_{\{1\}}, \alpha_{\{0,1\}}\}$	All three coefficients non-zero.

Table 21 – Possible combinations of zero and non-zero coefficients for 2-bits.

We define the *Walsh family* of a class as the set of IDs for each instance of the class. For example if the non-zero Walsh coefficients may be either $\{\alpha_{\{1\}}, \alpha_{\{0,1\}}\}$ (structure 6) or $\{\alpha_{\{0\}}, \alpha_{\{1\}}, \alpha_{\{0,1\}}\}$ (structure 7), the Walsh family is $\{6, 7\}$.

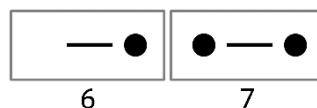


Figure 9 – An example Walsh family: $\{6, 7\}$. This is the Walsh family of 12 distinct classes.

The structure which is a subset of all possible structures for the family we refer to as the *minimal structure*. In this example of the family $\{6, 7\}$, the structure without the unnecessary coefficient – structure 6 – is the minimal structure of this class.

By taking the Walsh-Hadamard transform of the delta expansion of all 75 equivalence classes, we obtain Table 22 (p. 83), which shows the Walsh family for each class. The result is summarised in Table 23 (p. 84).

Class	Family	Class	Family	Class	Family
[0 0 0 0]	{0}	[3 0 1 2] †	{5,7}	[2 0 1 3] †	{6,7}
[1 1 1 0]	{7}	[0 1 1 1]	{7}	[0 2 2 1]	{7}
[1 1 0 1]	{7}	[0 2 2 0]	{4}	[0 2 1 2]	{7}
[2 2 0 0]	{2}	[1 2 2 0]	{7}	[0 3 1 1]	{5,7}
[2 2 1 0]	{7}	[0 2 0 2]	{1}	[0 3 2 0]	{7}
[2 2 0 1]	{7}	[0 3 0 0]	{7}	[1 3 2 0] †	{6,7}
[1 0 1 1]	{7}	[1 3 1 0]	{6,7}	[0 3 0 2]	{7}
[2 0 2 0]	{1}	[1 2 0 2]	{7}	[1 3 0 2] †	{3,7}
[2 1 2 0]	{7}	[1 3 0 1]	{3,7}	[0 1 2 2]	{7}
[2 0 0 2]	{4}	[2 3 0 0]	{7}	[0 1 3 1]	{6,7}
[3 0 0 0]	{7}	[2 3 1 0] †	{6,7}	[0 2 3 0]	{7}
[3 1 1 0]	{3,7}	[2 3 0 1] †	{3,7}	[1 2 3 0] †	{5,7}
[2 1 0 2]	{7}	[0 0 2 2]	{2}	[0 1 1 3]	{3,7}
[3 1 0 1]	{6,7}	[0 0 3 0]	{7}	[0 2 0 3]	{7}
[3 2 0 0]	{7}	[1 1 3 0]	{5,7}	[1 2 0 3] †	{5,7}
[3 2 1 0] †	{3,7}	[0 0 0 3]	{7}	[0 0 3 2]	{7}
[3 2 0 1] †	{6,7}	[1 1 0 3]	{5,7}	[0 0 2 3]	{7}
[2 0 2 1]	{7}	[1 0 2 2]	{7}	[1 0 3 2] †	{3,7}
[2 0 1 2]	{7}	[1 0 3 1]	{3,7}	[1 0 2 3] †	{6,7}
[3 0 1 1]	{5,7}	[2 0 3 0]	{7}	[0 3 2 1] †	{5,7}
[3 0 2 0]	{7}	[2 1 3 0] †	{5,7}	[0 3 1 2] †	{5,7}
[3 1 2 0] †	{3,7}	[1 0 1 3]	{6,7}	[0 2 3 1] †	{6,7}
[3 0 0 2]	{7}	[2 0 0 3]	{7}	[0 2 1 3] †	{3,7}
[3 1 0 2] †	{6,7}	[2 1 0 3] †	{5,7}	[0 1 3 2] †	{6,7}
[3 0 2 1] †	{5,7}	[2 0 3 1] †	{3,7}	[0 1 2 3] †	{3,7}

Table 22 – Walsh families of all 2-bit equivalence classes. Result first published in [3].

(† denotes one of the 24 classes of injective functions)

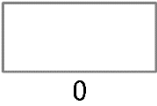
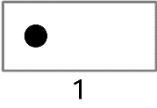

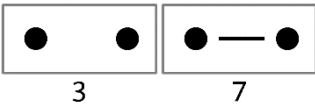
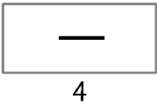
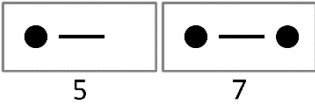
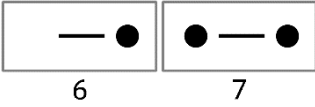
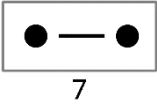
Family	Symbol	Deltas	Classes	Example
{0}		0	1	$[0\ 0\ 0\ 0] - \text{CONST}^2$
{1}		1	2	$[2\ 0\ 2\ 0]$
{2}		1	2	$[2\ 2\ 0\ 0]$
{3,7}		2	4	$[3\ 1\ 1\ 0] - \text{ONEMAX}^2$
		3	8	$[0\ 1\ 1\ 3] - \text{ZEROMAX}^2$ $[3\ 2\ 1\ 0] - \text{BINVAL}^2$
{4}		1	2	$[0\ 2\ 2\ 0] - \text{CHECK}_{1D}^2$
{5,7}		2	4	$[3\ 0\ 1\ 1]$
		3	8	$[3\ 0\ 2\ 1]$
{6,7}		2	4	$[3\ 1\ 0\ 1]$
		3	8	$[3\ 2\ 0\ 1]$
{7}		1	8	$[3\ 0\ 0\ 0] - \text{NEEDLE}^2$
		2	24	$[3\ 0\ 2\ 0] - \text{LEADING}^2$
				$[3\ 0\ 0\ 2] - \text{TRAP}_2^2$

Table 23 – Summary of Walsh families of 2-bit classes.

From this result, we notice that only the above 8 distinct families are possible in the set of 2-bit pseudo-Boolean functions, and that each Walsh coefficient is either necessarily zero, necessarily non-zero, or optional (conditioned on a relationship between the δ_i values, but independent of any other coefficient).

Further, we observe that the classes corresponding to all benchmark functions listed may have only their canonical structures (the structure of the named instance) with the exception of the classes of the univariate functions ONEMAX^2 , ZEROMAX^2 , and BINVAL^2 . These, and all other fully univariate (with both univariate coefficients necessarily non-zero) classes in 2-bits may contain an optionally non-zero bivariate coefficient.

In the case of ONEMAX^2 , or the ZEROMAX^2 , the bivariate coefficient $\alpha_{\{0,1\}} = \frac{1}{4}(\delta_1 - \delta_0)$ which is zero if and only if δ_0 (the fitness increase from changing either bit in the string $[0\ 0]$ to 1) is equal to δ_1 (the fitness increase from then setting the remaining bit to 1 to reach the string $[1\ 1]$). If this is not the case, there is non-additive interaction between the two variables, which is modelled by a non-zero bivariate coefficient.

We define a delta condition as an equation in terms of the delta values δ_i which, if satisfied, will mean that a Walsh coefficient is zero. For example, in the above example, if and only if the delta condition $\delta_0 = \delta_1$ is satisfied, then $\alpha_{\{0,1\}} = 0$. We say that a coefficient does *not* have a delta condition in the event that either the coefficient is *always* zero, or *never* zero.

In the case of BINVAL^2 , the bivariate coefficient $\alpha_{\{0,1\}} = \frac{1}{4}(\delta_2 - \delta_0)$ is zero if and only if δ_0 (the fitness increase from flipping the x_0 in $[0\ 0]$ to $[1\ 0]$) is equal to δ_2 (the fitness increase from flipping the same x_0 in $[0\ 1]$ to $[1\ 1]$). If this is not the case, then there is a non-additive interaction where the second variable affects the first, which is modelled by a non-zero bivariate coefficient.

Similarly, any 2-bit *fully univariate* classes – which are those where the univariate terms must be non-zero, but the variables are additively separable – will have a delta condition which makes the bivariate coefficient non-zero if the function is not additively separable. This is true of the class to which the ONEMAX^2 belongs, and the class to which the BINVAL^2 belongs, and all other fully univariate classes in 2-bits contain a function which is a relabelling and/or permutations of one of these classes. Relabellings and permutations do not change the set of non-zero Walsh coefficients, hence, the Walsh family $\{3, 7\}$ exists, but $\{3\}$ alone does not.

Interestingly, when the two necessary parts of structure are one univariate coefficient and the bivariate coefficient, we get the family $\{5, 7\}$ or $\{6, 7\}$ but never the family $\{5\}$ or $\{6\}$. The explanation for this is not in terms of additively separable functions (since all of these instances require the bivariate coefficient and are hence *not* additively separable). However, it can be shown that the rank vectors for each of the 12 classes in one of these families is a permutation of the rank vector of one of the 12 fully univariate classes. It follows that the same delta condition from $\alpha_{\{0,1\}}$ is permuted to either $\alpha_{\{0\}}$ or $\alpha_{\{1\}}$, since the delta conditions are derived

from the expression in the deltas δ_i which equals the value of the coefficient. These expressions are permuted by the permutation of the fitness vector. Hence, the same reasoning which explains why fully univariate classes have an unnecessary bivariate term, explains why classes with a necessary bivariate term and one necessary univariate term also have an unnecessary univariate term.

Examining how the number of deltas relates to the delta conditions – for 2 deltas, there is an unnecessary coefficient when there are 2 equal fitnesses in the middle with one below and one above (e.g. [3 0 1 1]). If there is an unnecessary coefficient, the delta condition is $\delta_0 = \delta_1$ and occurs for whichever row of the matrix multiplication produces the alignment shown in (113). We observe from the exhaustive survey that one such alignment always exists.

$$\begin{aligned}
4\alpha_k &= \pm (f_{min} + \delta_0 + \delta_1) \\
&\mp (f_{min} + \delta_0) \\
&\mp (f_{min} + \delta_0) \\
&\pm (f_{min})
\end{aligned}
\tag{ 113 }$$

Similarly we observe that an unnecessary coefficient cannot exist when there are two equal solutions at the lowest fitness (e.g. [3 2 0 0]) or the highest fitness [1 0 2 2]).

If there are 3 deltas, an unnecessary coefficient exists with the condition $\delta_0 = \delta_2$ for whichever row of the matrix multiplication produces the alignment shown in (114). One such alignment always exists.

$$\begin{aligned}
4\alpha_k &= \pm (f_{min} + \delta_0 + \delta_1 + \delta_2) \\
&\mp (f_{min} + \delta_0 + \delta_1) \\
&\mp (f_{min} + \delta_0) \\
&\pm (f_{min})
\end{aligned}
\tag{ 114 }$$

In general the delta condition for 2-bit classes is always when two delta values are equal.

Also, note that this means that all 24 *injective* functions classes in 2-bits contain one optional coefficient. This is because any injective function classes in 2-bits are precisely those with 3 deltas. It is noteworthy that the three benchmark functions regarded as the *hardest* in our 2-bit set, namely NEEDLE², LEADING², and TRAP₂², are all *non-injective* functions, and are thus able to contain no unnecessary structure, in all of these cases containing complete structure.

We show that the number of structures is generally higher for classes with more deltas. The small number of classes and small number of structures and deltas makes this trend difficult to see. See Table 24. Later we will show the equivalent result for 3-bits.

Number of Structures	Number of Deltas			
	0	1	2	3
1	1	14	24	0
2	0	0	12	24

Table 24 – Number of classes for each number of deltas and number of possible Walsh structures for 2-bit classes.

Grouping the classes by families, we can clearly see that for 2-bits, for all theoretical structures 0 – 7 are represented, i.e. there exists a class with that structure. We can show by construction this is true since there must be at least one instance of any structure, since the structures represent a partitioning of the \mathbb{R}^4 vector space of Walsh coefficients.

Additionally, if a class can be represented by a single coefficient, this is the only possible structure for this class. If said structure is a univariate term, then the class consists of functions of that variable only, meaning that adding another structure element (involving the other variable) would not produce a function in the same class since the resulting function would be a function of both variables. If said structure is the bivariate term, then the fitness is a function of the relationship between the variable ($X_0 = X_1$ or $X_0 \neq X_1$) and there is no fitness contribution of the individual variables, hence adding one would not produce a function in the same class.

We will also see this pattern later for 3-bits, i.e. that all theoretical structures are represented, and that when a class can be represented by a single coefficient, then that is the only possible structure for that class.

5.3 Automated Calculation of Walsh Families

Earlier, we discussed using the delta expansion of a function to compute delta conditions, and using the delta conditions to determine Walsh families. An alternate way to discover the Walsh families is to generate an exhaustive collection of functions given certain constraints.

Recall that reduced structure is caused by equivalent delta values. Hence, it is possible to satisfy (or not satisfy) any given delta condition by limiting the co-domain to only integer fitness values. There will be some finite range of integers necessary to construct all possible structure. We represent this range as 0 up to and including a given *bound* U . The form of these functions is given by (115).

$$f : \{0, 1\}^2 \rightarrow \{0, 1, \dots, U\} \tag{ 115 }$$

Each function is transformed using the Walsh-Hadamard transform and normalised by discarding the α_\emptyset term. Recall that the *structure* of the function is defined as the set of non-zero coefficients, e.g. $\{\alpha_{\{0\}}, \alpha_{\{0,1\}}\}$ we enumerate as the structure 5.

For each function transformed, the class and structure are recorded. If this (class, structure) tuple has not been seen before for some smaller value of U , it is recorded as a *discovered* class-structure pair.

		Bound (U)						
		0	1	2	3	4	≥ 5	
Number of Deltas	0	1	0	0	0	0	0	...
	1	0	14	0	0	0	0	...
	2	0	0	36	12	0	0	...
	3	0	0	0	24	24	0	...

Table 25 – Number of discovered valid structures for pseudo-Boolean classes in 2 dimensions. Table values state the number of structures discovered at the specified bound which were not discovered at a smaller bound.

As we increase the radius U , we record the discovered structures. These are shown in Table 25. We see that all class-structure pairs are discovered by trying all values of $U \leq 4$, and no further class-structure pairs are discovered for larger bounds.

We can see that delta condition $\delta_0 = \delta_2$ can be satisfied with function image $\{0, 1, 2, 3\}$ (i.e. $f_{min} = 0, \delta_0 = \delta_1 = \delta_2 = 1$) and unsatisfied for the function image $\{0, 1, 2, 4\}$ (i.e. $f_{min} = 0, \delta_0 = \delta_1 = 1, \delta_2 = 2$, or $f_{min} = 0, \delta_0 = \delta_2 = 1, \delta_1 = 2$). Thus, a bound of $U = 4$ can produce any structure which can exist for 2-bits where the codomain is the set of real numbers. Likewise, for classes with the delta condition $\delta_0 = \delta_1$, the bound of 3 is sufficient.

We see therefore that it is sufficient to set the bound to $U = 4$ to correctly identify the Walsh families for each 2-bit class using exhaustive evaluation.

5.4 Directed Ordinal Linkage and Epistasis

In genetics, epistasis is the study of gene interaction. For 2-bit functions we can compare three chosen functions, the $ONEMAX^2$, $LEADING^2$, and $CHECK_{1D}^2$. All three functions' fitness values are shown in Table 26.

$ONEMAX^2$			$LEADING^2$			$CHECK_{1D}^2$		
$X_0 + X_1$	x_0		$X_0 \rightarrow X_1$	x_0		$X_0 X_1$	x_0	
	0	1		0	1		0	1
x_1	0	1	0	0	1	0	0	1
x_1	1	2	1	0	2	1	1	0

Table 26 – Comparison of fitness values from variable interactions of $ONEMAX^2$, $LEADING^2$, $CHECK_{1D}^2$ functions.

We see the ones fitness is an additive contribution of the two variables with the linkage partitioned into two groups of one variable each, $X_0 + X_1$. For the checkerboard 1D the appropriate setting of each variable is dependent on the other, in the case of two bits, the fitness is the product of the two variables $X_0 X_1$, although not true for higher dimensions, we still use this notation explained in section 4.2. The Leading ones function is an example of directed linkage, for which we use the notation $X_0 \rightarrow X_1$. The fitness landscapes are plotted in Figure 10.

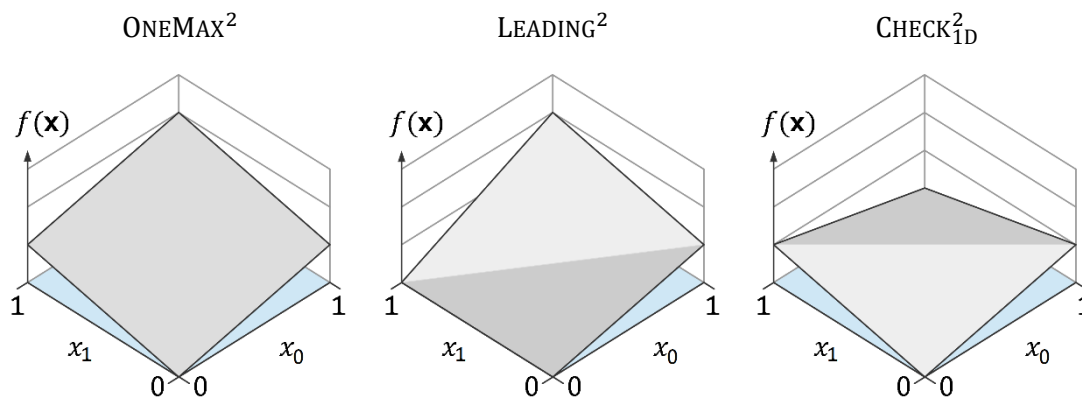


Figure 10 – Comparison of fitness landscapes of $ONEMAX^2$, $LEADING^2$, $CHECK_{1D}^2$ functions.

Looking at these fitness landscapes we compare the adjacent vertices and assign $>$, $=$, or $<$ to each edge corresponding to the difference in the sign (sgn) of the fitness difference, $+1$, $-$, or -1 as in section 4.2 The illustration in Figure 11 shows the comparison of signs on parallel edges to detect linkage.

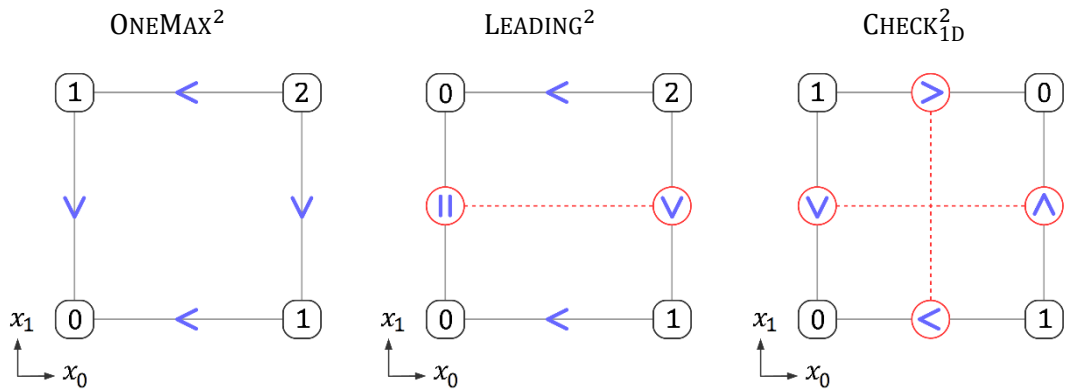


Figure 11 – Detecting directed ordinal linkage of $ONEMAX^2$, $LEADING^2$, $CHECK_{1D}^2$ functions. The dotted lines represent parallel edges with different signs in the fitness difference of the two candidates on that edge of the Hamming space. For instance, with $LEADING^2$, as we move from 0 to 1 on variable X_0 , the sign of the fitness difference on X_1 changes, therefore X_1 is dependant on X_0 (written as $X_0 \rightarrow X_1$).

5.5 Precedence Networks and Precedence Profiles

Next we look at the algorithmic steps we can use to solve a problem. A non-revisiting algorithm (one which does not evaluate the same candidate twice) will evaluate some or all of the 4 candidates in the solution space for a 2-bit function, and terminate. We consider how different algorithms may visit the different linkage partitions of a problem. A *precedence network* refers to a directed acyclic graph where vertices represent linkage groups. These precedence networks imply algorithms to locate a global optimum. The algorithms defined by these networks or the networks themselves may be refer to as precedence networks.



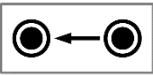

ID	Symbol	Linkage	Cost	Algorithm
A0	 A0	X_0X_1	4	Exhaustive search – all 4 candidates are evaluated in an arbitrary order.
B0	 B0	$X_0 \rightarrow X_1$	3	Both values for X_0 tried with an arbitrary setting of X_1 , then optimal setting for X_0 tried with the remaining setting of X_1 .
B1	 B1	$X_1 \rightarrow X_0$	3	Both values for X_1 tried with an arbitrary setting of X_0 , then optimal setting for X_1 tried with the remaining setting of X_0 .
C0	 C0	$X_1 + X_0$	3	Use <i>either</i> the algorithm for B0 or B1. (arbitrary)

Table 27 – Description of all 4 precedence networks for 2-bit functions. The precedence networks have been denoted A0, B0, B1 and C0 for reference. A, B, and C name the three possible arrangement invariant under relabelling, since the case B has two possible relabellings, they are numbered B0 and B1.

Since the network C0 does not specify the order in which to visit linkage partitions, we consider only the networks {A0, B0, B1} for this analysis. We call these three networks the *fully-specified* precedence networks. As specified above, the C0 case can be solved by either B0 or B1. To determine the cost of a fully specified network, take the number of arrangements of the first linkage group, plus one less the number of arrangements of the remaining groups. This cost function is given by (119) and applies to any dimensionality of precedence network.

$$cost(\gamma) = 2^{|\gamma_0|} + \sum_{i=1}^{n-1} (2^{|\gamma_{i1}|} - 1) \quad (116)$$

To determine the cost of network C0, take the arithmetic mean of the cost for B0 and B1 to obtain the expected cost. Since in this case the costs for B0 and B1 is the same, it is the same for C0.

Other options for evaluating the network C0 exist, but are not more efficient than 3 function evaluations. The implications of including parallel processing on this analysis – including the cases for 3-bits – is returned to in chapter 6.

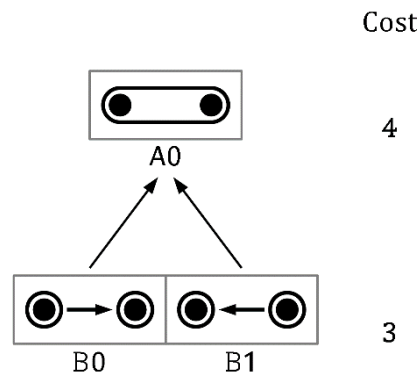


Figure 12 – All 2-bit fully-specified precedence networks represented pictorially. Connecting arrows show the relationship of which network will solve every class solvable by another class.

Using the fully-specified 2-bit precedence networks {A0, B0, B1}, we can construct a simple hierarchy, wherein A0 solves any function class solvable by either B0 or B1, although it requires more function evaluations.

We now run each algorithm {A0, B0, B1} on each class and return the probability that a global optimum will be reached for a given class c as $[P(A0, c) P(B0, c) P(B1, c)]$ where $P(a, c)$ is the probability of algorithm a reaching the optimum of a function in class c . This result we call the *precedence profile* of a class. For example, the class [3 1 0 1] has the precedence

profile $[1 \ 1/2 \ 3/4]$ since the algorithm $A1$ always reaches the global optimum of this class, the algorithm $E0$ reaches the global optimum with $1/2$ probability, and the algorithm $E1$ reaches the global optimum with $3/4$ probability. If choosing one of these algorithms at random, the probability of reaching the global optimum is the average $3/4$, which we use as a measure of problem difficulty.

Table 28 shows the precedence profiles of all 75 2-bit function classes.

Class	Profile	Class	Profile	Class	Profile
[0 0 0 0]	[1 1 1]	[3 0 1 2] †	[1 1/2 1/2]	[2 0 1 3] †	[1 1/2 1/2]
[1 1 1 0]	[1 1 1]	[0 1 1 1]	[1 1 1]	[0 2 2 1]	[1 1 1]
[1 1 0 1]	[1 1 1]	[0 2 2 0]	[1 1 1]	[0 2 1 2]	[1 1 1]
[2 2 0 0]	[1 1 1]	[1 2 2 0]	[1 1 1]	[0 3 1 1]	[1 3/4 1/2]
[2 2 1 0]	[1 1 1]	[0 2 0 2]	[1 1 1]	[0 3 2 0]	[1 1/2 1/2]
[2 2 0 1]	[1 1 1]	[0 3 0 0]	[1 3/4 3/4]	[1 3 2 0] †	[1 1/2 1/2]
[1 0 1 1]	[1 1 1]	[1 3 1 0]	[1 1/2 3/4]	[0 3 0 2]	[1 1 3/4]
[2 0 2 0]	[1 1 1]	[1 2 0 2]	[1 1 1]	[1 3 0 2] †	[1 1 1]
[2 1 2 0]	[1 1 1]	[1 3 0 1]	[1 1 1]	[0 1 2 2]	[1 1 1]
[2 0 0 2]	[1 1 1]	[2 3 0 0]	[1 3/4 1]	[0 1 3 1]	[1 1/2 3/4]
[3 0 0 0]	[1 3/4 3/4]	[2 3 1 0] †	[1 1/2 1]	[0 2 3 0]	[1 1/2 1/2]
[3 1 1 0]	[1 1 1]	[2 3 0 1] †	[1 1 1]	[1 2 3 0] †	[1 1/2 1/2]
[2 1 0 2]	[1 1 1]	[0 0 2 2]	[1 1 1]	[0 1 1 3]	[1 1 1]
[3 1 0 1]	[1 1/2 3/4]	[0 0 3 0]	[1 3/4 3/4]	[0 2 0 3]	[1 1 3/4]
[3 2 0 0]	[1 3/4 1]	[1 1 3 0]	[1 3/4 1/2]	[1 2 0 3] †	[1 1 1/2]
[3 2 1 0] †	[1 1 1]	[0 0 0 3]	[1 3/4 3/4]	[0 0 3 2]	[1 3/4 1]
[3 2 0 1] †	[1 1/2 1]	[1 1 0 3]	[1 3/4 1/2]	[0 0 2 3]	[1 3/4 1]
[2 0 2 1]	[1 1 1]	[1 0 2 2]	[1 1 1]	[1 0 3 2] †	[1 1 1]
[2 0 1 2]	[1 1 1]	[1 0 3 1]	[1 1 1]	[1 0 2 3] †	[1 1/2 1]
[3 0 1 1]	[1 3/4 1/2]	[2 0 3 0]	[1 1 3/4]	[0 3 2 1] †	[1 1/2 1/2]
[3 0 2 0]	[1 1 3/4]	[2 1 3 0] †	[1 1 1/2]	[0 3 1 2] †	[1 1 1/2]
[3 1 2 0] †	[1 1 1]	[1 0 1 3]	[1 1/2 3/4]	[0 2 3 1] †	[1 1/2 1/2]
[3 0 0 2]	[1 1/2 1/2]	[2 0 0 3]	[1 1/2 1/2]	[0 2 1 3] †	[1 1 1]
[3 1 0 2] †	[1 1/2 1/2]	[2 1 0 3] †	[1 1/2 1/2]	[0 1 3 2] †	[1 1/2 1]
[3 0 2 1] †	[1 1 1/2]	[2 0 3 1] †	[1 1 1]	[0 1 2 3] †	[1 1 1]

Table 28 – Precedence profiles of all 2-bit equivalence classes. († denotes one of the 24 a classes of injective functions)

A0	B0	B1	Avg.	Deltas	Classes	Example
1	1	1	1	0	1	[0 0 0 0] – CONST ²
				1	10	[0 2 2 0] – CHECK _{ID} ²
				2	16	[3 1 1 0] – ONEMAX ²
				3	8	[3 2 1 0] – BINVAL ²
1	1	3/4	11/12	2	4	[3 0 2 0] – LEADING ²
1	3/4	1	11/12	2	4	[3 2 0 0]
1	1	1/2	5/6	3	4	[3 0 2 1]
1	1/2	1	5/6	3	4	[3 2 0 1]
1	3/4	3/4	5/6	2	4	[3 0 0 0] – NEEDLE ²
1	1/2	3/4	3/4	2	4	[3 1 0 1]
1	3/4	1/2	3/4	2	4	[3 0 1 1]
1	1/2	1/2	2/3	2	4	[3 0 0 2] – TRAP ₂ ²
				3	8	[1 2 3 0]

Table 29 – Summary of precedence profiles of 2-bit classes.

With the details of Walsh families and the details of precedence profiles, we are able to construct a table stating how many 2-bit classes exist for each pairing of Walsh family and precedence profile.

A0	B0	B1	{0}	{1}	{2}	{3,7}	{4}	{5,7}	{6,7}	{7}
1	1	1	1	2	2	12	2	0	0	16
1	1	3/4	0	0	0	0	0	0	0	4
1	3/4	1	0	0	0	0	0	0	0	4
1	1	1/2	0	0	0	0	0	4	0	0
1	1/2	1	0	0	0	0	0	0	4	0
1	3/4	3/4	0	0	0	0	0	0	0	4
1	1/2	3/4	0	0	0	0	0	0	4	0
1	3/4	1/2	0	0	0	0	0	4	0	0
1	1/2	1/2	0	0	0	0	0	4	4	4

Table 30 – For 2 bits, the number of classes for each combination of Walsh family and precedence profile.

We observe that classes tend to cluster together in multiples of 4. To explain the prevalence of 4s in the table we observe that considering both value relabelling and order relabelling (discussed in section 3.1.7, p. 40) each 2-bit function has another 3 equivalent up to relabelled functions totalling 4.

Exceptions exist where there are not 4 classes equivalent up to relabelling. For the {0}, {1}, and {2} families; these represent less than 2-bit functions, i.e. they have a linkage partition of no variables $\Gamma = \emptyset$, only the first variable $\Gamma = \{\{X_0\}\}$, or only the second variable $\Gamma = \{\{X_1\}\}$ respectively. Additionally the {4} families has a similar symmetry as it is just the bivariate coefficient, there are only two possibilities: *+ve* or *-ve* bivariate coefficient.

5.6 Delta Linkage Detection

As previously stated in section 5.2, for the case of 2-bits, there can be at most one unnecessary piece of structure. Here we consider whether we can use the delta expansion to detect whether the variables are linked (a bivariate class) or not (which we shall call a *non-bivariate class*). To detect non-bivariate problems in 2-bits we only need to look at the product of the fourth row (row 3) of the Hadamard matrix with the delta expansion. In this section we describe the formal relationship between using the delta value and ordinal independence criteria. This we can show by exhaustive evaluation of the 75 classes, that this holds true for all 2-bit classes.

Instead of writing out the delta expansion in terms of positive δ symbols, we can automate the process more easily by defining a *delta matrix* Δ which is populated by 0s and 1s representing the coefficients of the δ values (since each delta is present or not present). The coefficient of f_{min} , from the delta expansion is omitted as these coefficients will always cancel for any non-empty clique. The construction of the delta matrix is given in (117).

$$\Delta = \begin{bmatrix} D_{11}^0 & D_{11}^1 & D_{11}^2 \\ D_{01}^0 & D_{01}^1 & D_{01}^2 \\ D_{10}^0 & D_{10}^1 & D_{10}^2 \\ D_{00}^0 & D_{00}^1 & D_{00}^2 \end{bmatrix} \quad (117)$$

$$\text{where } D_{\mathbf{x}}^r = \begin{cases} 1, & R_f(\mathbf{x}) > r \\ 0, & \text{otherwise} \end{cases}$$

The above construction is not completely matched with the delta expansion in the case of less than 2^ℓ distinct ranks, however, the difference is only in redundant repeated columns, which do not affect the result.

We define the result of the matrix multiplication as the *delta condition vector* $V_{\{0,1\}}$ as given by (118).

$$V_{\{0,1\}} = [H_2]_{row:3} \cdot \Delta \quad (118)$$

where $[H_2]_{row:3} = [1 \quad -1 \quad -1 \quad 1]$

If the delta condition vector $V_{\{0,1\}}$ is zero, the class is never bivariate. If the non-zero elements of the vector all have the same sign, the class is always bivariate. If the vector contains both positive and negative terms, then the bivariate term is unnecessary.

One way we may express this condition that a vector is not completely zero and all non-zero elements are of the same sign is the terms given by (119).

$$\mathcal{L}_\delta(V) \Leftrightarrow \left(\sum_i |V_i| \neq 0 \right) \wedge \left(\left| \sum_i V_i \right| = \sum_i |V_i| \right) \quad (119)$$

By comparing the result of this condition, with the result of applying the ordinal independence criteria $X_0 + X_1$ (equivalence 66, p. 52), we observe that for every class on 2-bits, the absence of linkage according to this formulation based on Walsh expansion is equivalent formulation to the ordinal independence criteria. This is stated by (120).

$$X_0 + X_1 \Leftrightarrow \neg \mathcal{L}_\delta(V_{\{0,1\}}) \quad (120)$$

Thus, we see that for 2-bits the delta expansion, which is in terms of Walsh coefficients, can be connected with ordinal independence, which is in terms of linkage partition.

5.7 Structural Coherence

In this section we consider the difficulty of learning the linkage of each 2-bit class for an EDA and compare the population size and selection size necessary for different categories of 2-bit classes. The experiments in this section were produced and published in collaboration with Dr Alexander Brownlee, University of Stirling in [3]. My contribution to this work was in providing the structural data on all 2-bit function classes, and further summary details presented in this section in Table 33 and Figure 14. Readers of [3] should be aware that the notation has been adapted to be consistent with the conventions used in this thesis.

EDAs explicitly model structure, therefore there is a relationship between EDA preference and the structure of the function. Echegoyen et al. [106] show that the relationship between the model and structure affects performance. It is of interest how well EDAs detect essential structure. As shown in the previous section (5.6); ordinal linkage for 2-bit pseudo-Boolean problems is determined by whether the bivariate term is necessary to maintain the ranking. The difficulty is based on the minimum population size required to have a statistically-significant result for the statistical independence test chi-squared χ^2 detection of the interaction between X_0 and X_1 (the condition $\mathcal{L}_0(0, 1)$). The chi-squared test is one test an EDA may use to detect interactions. This result is applicable to any EDA using this method.

An algorithm detects the linkage correctly if there exist linkage between the variables and the interaction is picked up by the EDA, or if there is no linkage and no linkage is detected by the EDA. The algorithm fails to detect linkage correctly if there exists linkage between the variables but it is not detected, or if there is no linkage but a spurious correlation is modelled.

We define S_{min} as the minimum population size required such that the chi-squared value as given by (121) exceeds the critical value $c = 3.84$ (probability 0.95 for 1 degree of freedom); if no population size is sufficient to correctly detect the linkage then $S_{min} = \infty$.

$$\chi_{0,1}^2(S) = \sum_{X_0, X_1 \in \{0,1\}^2} \frac{(S \cdot p(X_0 X_1) - S \cdot p(X_0) \cdot p(X_1))^2}{S \cdot p(X_0) \cdot p(X_1)} \geq 3.84 \quad (121)$$

Here we sum over all 4 possible configurations of X_0 and X_1 , $p(X_0)$ represents the proportion of the population which has X_0 in that configuration, and similarly with $p(X_0)$ and $p(X_0 X_1)$. S represents that population size.

We can rearrange this formula by dividing out the S term, to find the minimum value for S_{min} , the minimum population required to detect the linkage with statistical significance as given by (122, p. 101).

$$S_{min} = \begin{cases} \left\lceil \frac{3.84}{\chi_{i,j}^2/S} \right\rceil, & \chi_{i,j}^2 \neq 0 \\ \infty, & \chi_{i,j}^2 = 0 \end{cases} \quad (122)$$

$$\text{where } \chi_{i,j}^2/S = \sum_{X_i X_j \in \{0,1\}^2} \frac{(p(X_i X_j) - p(X_i) \cdot p(X_j))^2}{p(X_i) \cdot p(X_j)}$$

For every 2-bit class, for the two chosen selection methods, we can explicitly calculate this value by calculating the probability of selection for each candidate, and hence calculating the values for $p(X_i X_j)$, $p(X_i)$, and $p(X_j)$.

With tournament selection, there are 16 possible tournaments arising for each combination of 4 possible candidates. For a given function class, for each tournament, the winner of the tournament is determined. If there is no tie, the higher-fitness candidate will be selected twice, contributing 1/8 to its total probability of selection. In the case of a tie, the first candidate is chosen. Since for every tournament A, B , the tournament B, A is also run, in the case of a tie, from the two tournaments, A will be selected once and B will be selected once each contributing 1/16 to their total probability of selection.

With truncation selection, the top $1/4$, $1/3$ or $1/2$ of the candidates are selected, assuming that where there are ties, the corresponding block of the population is comprised of equal parts of the equal-rank candidates. An example is given in Figure 13.

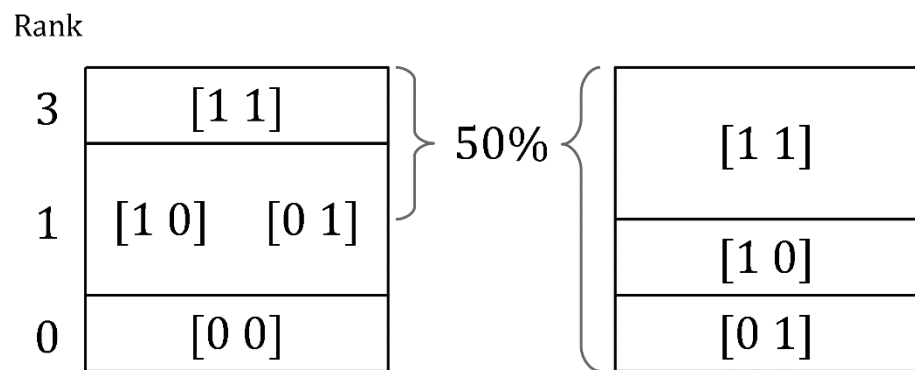


Figure 13 – Example of selecting the top 50% for the class of the $ONEMAX^2$ function. The candidate [1 1] is selected with 0.5 probability and [1 0] and [0 1] are each selected with 0.25 probability.

For this result we group the 2-bit classes into categories by the number of candidates at each rank. Each class in a category is represented by a rank vector which is a permutation of a rank vector of any other class in the same category. These categories are listed in Table 31.

Cat. ID	Description	Num. Deltas	Num. Classes	Global Optima	Example Class
0	1 distinct.	0	1	4	$[0\ 0\ 0\ 0] - \text{CONST}^2$
1	2 distinct: 3 maxima, 1 minimum.	1	4	1	$[3\ 0\ 0\ 0] - \text{NEEDLE}^2$
2	2 distinct: 1 maximum, 3 minima.	1	4	3	$[1\ 1\ 1\ 0]$
3	2 distinct: 2 maxima, 2 minima.	1	6	2	$[0\ 2\ 2\ 0] - \text{CHECK}_{1D}^2$
4	3 distinct: 1 maximum, 2 minima.	2	12	1	$[3\ 0\ 2\ 0] - \text{LEADING}^2$ $[3\ 0\ 0\ 2] - \text{TRAP}_2^2$
5	3 distinct: 1 maximum, 1 minimum	2	12	1	$[3\ 1\ 1\ 0] - \text{ONEMAX}^2$ $[0\ 1\ 1\ 3] - \text{ZEROMAX}^2$
6	3 distinct: 2 maxima, 1 minimum	2	12	2	$[2\ 2\ 1\ 0]$
7	4 distinct.	3	24	1	$[3\ 2\ 1\ 0] - \text{BINVAL}^2$

Table 31 – Categories of 2-bit functions based on number of candidates at each rank.

We observe that the population size and selection size has an effect on the correct detection of linkage. We also observe that building a model from negative selection (selecting the least fit individuals) can actually improve accuracy of linkage detection for certain classes.

Cat.	Class	Family	X_0X_1	Tournament		Truncation Top			Truncation Bottom			Truncation T+B				
				Best	Worst	0.25	0.33	0.5	0.25	0.33	0.5	0.25	0.33	0.5		
0	[0 0 0]	{0}	N	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	[3 0 0 0]	{7}	Y	96	35	∞	61	∞	15	15	15	15	15	15	15	15
	[0 3 0 0]	{7}	Y	96	35	∞	61	∞	15	15	15	15	15	15	15	15
	[0 0 3 0]	{7}	Y	96	35	∞	61	∞	15	15	15	15	15	15	15	15
	[0 0 0 3]	{7}	Y	96	35	∞	61	∞	15	15	15	15	15	15	15	15
2	[0 1 1 1]	{7}	Y	35	96	15	15	15	∞	61	∞	15	15	15	15	15
	[1 0 1 1]	{7}	Y	35	96	15	15	15	∞	61	∞	15	15	15	15	15
	[1 1 0 1]	{7}	Y	35	96	15	15	15	∞	61	∞	15	15	15	15	15
	[1 1 1 0]	{7}	Y	35	96	15	15	15	∞	61	∞	15	15	15	15	15
3	[2 2 0 0]	{2}	N	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 0 2 0]	{1}	N	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 2 2 0]	{3}	Y	15	15	4	4	4	4	4	4	4	4	4	4	4
	[2 0 0 2]	{3}	Y	15	15	4	4	4	4	4	4	4	4	4	4	4
	[0 2 0 2]	{1}	N	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 0 2 2]	{2}	N	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	[3 2 0 0]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 3 0 0]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 0 3 0]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 0 0 3]	{7}	Y	16	14	∞	4	4	4	4	4	4	4	4	4	4
	[3 0 2 0]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 3 2 0]	{7}	Y	16	14	∞	4	4	4	4	4	4	4	4	4	4
	[0 2 3 0]	{7}	Y	16	14	∞	4	4	4	4	4	4	4	4	4	4
	[0 2 0 3]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[3 0 0 2]	{7}	Y	16	14	∞	4	4	4	4	4	4	4	4	4	4
	[0 3 0 2]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 0 3 2]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 0 2 3]	{7}	Y	726	81	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
5	[3 0 1 1]	{5,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
	[0 3 1 1]	{5,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
	[0 1 3 1]	{6,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
	[0 1 1 3]	{3,7}	N	143	143	∞	61	15	∞	61	15	∞	∞	∞	∞	∞
	[3 1 0 1]	{6,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
	[1 3 0 1]	{3,7}	N	143	143	∞	61	15	∞	61	15	∞	∞	∞	∞	∞
	[1 0 3 1]	{3,7}	N	143	143	∞	61	15	∞	61	15	∞	∞	∞	∞	∞
	[1 0 1 3]	{6,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
	[3 1 1 0]	{3,7}	N	143	143	∞	61	15	∞	61	15	∞	∞	∞	∞	∞
	[1 3 1 0]	{6,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
	[1 1 3 0]	{5,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
	[1 1 0 3]	{5,7}	Y	23	23	∞	10	15	∞	10	15	∞	∞	∞	∞	∞
6	[1 0 2 2]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 1 2 2]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 2 1 2]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 2 2 1]	{7}	Y	14	16	4	4	4	∞	4	4	4	4	4	4	4
	[1 2 0 2]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 1 0 2]	{7}	Y	14	16	4	4	4	∞	4	4	4	4	4	4	4
	[2 0 1 2]	{7}	Y	14	16	4	4	4	∞	4	4	4	4	4	4	4
	[2 0 2 1]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[1 2 2 0]	{7}	Y	14	16	4	4	4	∞	4	4	4	4	4	4	4
	[2 1 2 0]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 2 1 0]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 2 0 1]	{7}	Y	81	726	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
7	[3 2 1 0]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 3 1 0]	{6,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[3 1 2 0]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[1 3 2 0]	{6,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞
	[1 2 3 0]	{5,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞
	[2 1 3 0]	{5,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[3 2 0 1]	{6,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 3 0 1]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[3 0 2 1]	{5,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 3 2 1]	{5,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞
	[0 2 3 1]	{6,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞
	[2 0 3 1]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[3 0 1 2]	{5,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞
	[0 3 1 2]	{5,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[3 1 0 2]	{6,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞
	[1 3 0 2]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[1 0 3 2]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 1 3 2]	{6,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[0 2 1 3]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	[2 0 1 3]	{6,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞
[0 1 2 3]	{3,7}	N	173	173	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	
[1 0 2 3]	{6,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	
[1 2 0 3]	{5,7}	Y	46	46	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	
[2 1 0 3]	{5,7}	Y	14	14	∞	4	4	∞	4	4	∞	∞	∞	∞	∞	

Table 32 – Minimum population sizes required to detect actual or spurious correlation for all 2-bit functions with statistically significant $\chi_{0,1}^2 > 3.84$ – first published in [3].

Note that there are symmetries in the result. For example, the category 1 (the NEEDLE² and variable re-ordered variants) and category 2 (functions where only one value is sub-optimal) have equivalent results except that the result for positive selection and the result for negative selection are swapped. This is because one category is the inverse of the other in terms of whether solutions are grouped at the top fitness vs the bottom fitness.

If we count the number of 2-bit classes for which linkage is correctly detected as a function of population size S , we see that across the set of 2-bit classes tournament selection selecting the best individual performs equally to selecting the worst, and for each selection size, top selection performs equally to bottom selection. We also see that each selection size for top-and-bottom selection performs equally to one another. This is detailed in Table 33 for each interval and these 5 cumulative distributions are graphed in Figure 14 (p. 105) for comparison.

Population Size (as half-open intervals)	Number of Classes for which Linkage is Detected Correctly (of 75 total)				
	Tournament (Best)	Top Selection (0.25)	Top Selection (0.33)	Top Selection (0.5)	Top+Bottom Selection (0.5)
	Tournament (Worst)	Bottom Selection (0.25)	Bottom Selection (0.33)	Bottom Selection (0.5)	Top+Bottom Selection (0.33)
[1, 4)	17	17	17	17	17
[4, 10)	17	23	35	35	27
[10, 14)	17	23	43	35	27
[14, 15)	29	23	43	35	27
[15, 16)	31	27	47	43	35
[16, 23)	35	27	47	43	35
[23, 35)	43	27	47	43	35
[35, 46)	47	27	47	43	35
[46, 61)	55	27	47	43	35
[61, 81)	55	27	47	43	35
[81, 96)	63	27	47	43	35
[96, 143)	67	27	47	43	35
[143, 173)	63	27	47	43	35
[173, 726)	55	27	47	43	35
[726, ∞)	63	27	47	43	35

Table 33 – Number of 2-bit classes for which linkage is correctly detected as a function of population size, comparing different selection types.

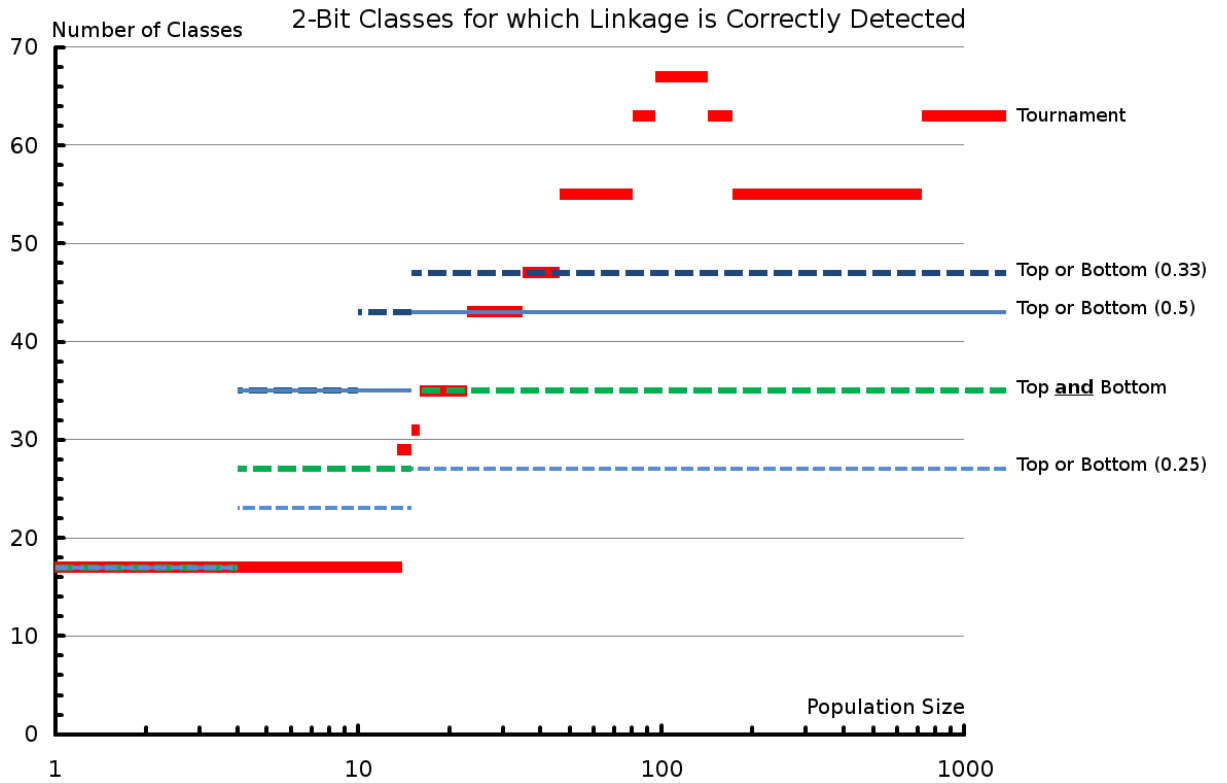


Figure 14 – Number of 2-bit classes for which linkage is correctly detected plotted against population sizes, comparing different selection types. All jump discontinuities depicted are right-continuous.

5.8 Summary

In this chapter we have examined the space of 2-bit classes. We have defined the concepts of Walsh families, delta conditions, precedence networks, and precedence profiles. We have seen that the precedence networks set out algorithmic steps which can be used to solve a function, and that the precedence profile gives a description of the difficulty of the problem class. In the following chapter we will describe extending this to 3-bit classes.

6 3-Bit Pseudo-Boolean Functions

In this chapter we explore the set of *3-bit classes*. Conclusions drawn from this chapter will necessarily be more summative than the analysis of 2-bit classes, since the 3-bit classes are too numerous to list unabridged. The structure of this chapter is set up to compare and contrast our observations with that of the 2-bit classes. Some of the work in this section was first published in [2]. There is also some recent literature which looks at the space of 3-bit classes in [106] [107] on taxonomy of injective 3-bit function classes under equivalence based on finite and infinite population models.

6.1 Counting 3-Bit Classes

A 3-bit equivalence class consists of ranks for the 8 possible values of x . Table 34 shows the number of 3-bit function classes with a breakdown of number of classes for each valid number of ranks. The number of classes for 8 ranks (40 320) is the number of injective function classes. In total, there are 545 835 function classes.

Num. Ranks, n	Num. Classes, $c(n, 3)$
1	1
2	254
3	5 796
4	40 824
5	126 000
6	191 520
7	141 120
8	40 320
Total	545 835

Table 34 – The number of function classes in 3-bits for a given number of distinct fitness levels (number of ranks). There are 545 835 distinct classes for 3-bit functions.

6.2 Walsh Families and Delta Conditions

When we count the number of distinct delta conditions, we find 395 possible distinct delta conditions (ignoring coefficients which are always zero or always non-zero). These are only those equations generated from expressions from the delta expansion which contain both positive and negative coefficients of delta values. Setting this expression equal to zero we get an equation, which we present rearranged such that terms with negative coefficients are moved to the other side of the equation.

From exhaustive evaluation of this set, we observe that the most prolific delta condition, which occurs 41 496 times in the set of equivalence classes, is the condition $\delta_0 = \delta_2$. The least common conditions are $\delta_0 = 3\delta_1$ and $\delta_1 = 3\delta_0$, which each only occur 56 times. There are shown in Table 35.

Delta Condition	Occurrences	Example Class and Coefficient
$\delta_0 = \delta_2$	41 496	[0 1 2 3 3 3 3 3] / $\alpha_{\{0,1\}}$
...		
$\delta_0 = 3\delta_1$	56	[1 1 1 1 0 5 5 5] / $\alpha_{\{2\}}$
$\delta_1 = 3\delta_0$	56	[3 0 0 3 0 3 3 7] / $\alpha_{\{0,1,2\}}$

Table 35 – The most and least prolific delta conditions in 3-bit classes.

To get an idea of the possible complexity of delta conditions in this space, we can sum the absolute values of the coefficients of the terms. The most complex, using this measure, are the four conditions shown in Table 36.

Delta Condition	Occurrences	Example Class and Coefficient
$\delta_0 + 2\delta_1 + 3\delta_2 + 2\delta_3 + \delta_4 = \delta_5$	4 032	[3 0 1 4 2 5 5 7] / $\alpha_{\{0,1,2\}}$
$\delta_0 + 2\delta_1 + 3\delta_2 + 2\delta_3 + \delta_4 = \delta_6$	8 064	[3 0 1 4 2 5 6 7] / $\alpha_{\{0,1,2\}}$
$\delta_0 = \delta_1 + 2\delta_2 + 3\delta_3 + 2\delta_4 + \delta_5$	4 032	[1 1 3 4 0 5 6 7] / $\alpha_{\{2\}}$
$\delta_0 = \delta_2 + 2\delta_3 + 3\delta_4 + 2\delta_5 + \delta_6$	8 064	[1 2 3 4 0 5 6 7] / $\alpha_{\{2\}}$

Table 36 – All delta conditions in 3-bit classes where the sums of the coefficients is at its maximum (the sum equals 10).

		Bound (U)															Number of Deltas									
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	≥ 16	0	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	254	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	5 796	2 940	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	40 824	37 464	16 240	3 920	784	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	126 000	152 544	92 064	67 536	16 128	5 376	1 344	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	191 520	254 016	198 912	175 392	104 832	28 224	12 096	6 720	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	141 120	154 560	225 792	142 464	151 872	57 792	24 192	16 128	8 064	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	40 320	25 536	83 328	41 664	65 856	28 224	21 504	16 128	8 064	0	0	0	0	0	0	0	0	0	0	

Table 37 - Number of discovered valid structures for 3-bit pseudo-Boolean classes. Table values state the number of structures discovered at the specified bound which were not discovered at a smaller bound.

6.3 Automated Calculation of Walsh Families

Recall the method of automated discovery of Walsh families of the space of 2-bit function describe in section 5.3. Again, we are able to use only integer values with a fixed upper bound U to discover the Walsh families of all 545 835 3-bit classes. Table 37 (p. 109) shows the distribution of valid structure discoveries at each increment of the bound U for 0 and 15. Having tried a bound of up to 17, we conjecture that no further structures are discovered for higher values of U . Further details in only this section 6.3 depend on this conjecture.

It is worthy of note that for the case of 7 deltas, although there are no additional structures discovered at radius 14 (which were not already seen for radius 13), there are new structures found for at radius 15. Thus, no additionally structures being detected at radius 16 and 17 does not prove that there are none discovered for higher radii.

Recall that the highest sum of absolute values of coefficients has a sum of 9 in the absolute value of coefficients on one side, and 1 on the other, for example the delta condition $\delta_0 + 2\delta_1 + 3\delta_2 + 2\delta_3 + \delta_4 = \delta_5$. For integer deltas, the left hand side is at least 9 (when all five deltas on the LHS are 1), hence $\delta_5 \geq 9$. Since this does not include the δ_6 term, which is strictly positive, 1 must be added for this term. There, the sum of all deltas is at least 15 (from the first five set to one, and the last, nine, plus one for the missing term: $5 + 9 + 1$). Thus in order to satisfy this delta condition, a bound of $U \geq 15$ is needed. This is suggestive that the conjectured bound of $U = 15$ is sufficient for all structures.

Below we show the relationship between the increasing number of deltas and the number of possible structures in a 3-bit function. There are some numbers of structures, 9, 11, 13, 14, and 15 which are not present in the data, that is, no 3-bit Walsh family contains that number of members. We see a trend of increasing number of possible structures as the number of deltas increases.

Num. Structs.	Number of Deltas							
	0	1	2	3	4	5	6	7
1	1	254	2 744	6 720	4 032	0	0	0
2	0	0	3 052	16 072	23 856	10 080	0	0
3	0	0	0	11 760	30 240	21 504	8 064	0
4	0	0	0	6 272	41 664	53 760	14 784	2 688
5	0	0	0	0	12 096	41 664	34 944	5 376
6	0	0	0	0	8 064	24 864	24 192	9 408
7	0	0	0	0	5 376	18 816	16 128	2 688
8	0	0	0	0	672	15 456	25 536	8 064
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	2 688	6 720	5 376
11	0	0	0	0	0	0	0	0
12	0	0	0	0	0	2 688	9 408	4 032
13	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	1 344	2 688

Table 38 – Number of classes for each number of deltas and number of possible Walsh structures for 3-bit classes.

6.4 Conditionally-Necessary Interactions

By exploring the result of our automated calculation of Walsh families, we discover types of structure not present in 2-bit classes. Recall that in 2-bit classes, structure was either necessarily zero, necessarily non-zero, or optional. We see that this analysis of 3-bit structures is useful since it highlights an incompleteness in considering elements of Walsh structure to be simply necessary or unnecessary (spurious).

If we take an instance of the benchmark class $C_{\text{BINVAL}^3} = [7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1 \ 0]$ such as the instance BINVAL^3 , and increase the bivariate term $\alpha_{\{0,1\}}$ (previously 0) by a small amount, we see that when $x_0 = x_1$, fitness values (f_{000} , f_{110} , f_{001} , and f_{111}) are incremented by this amount, when $x_0 \neq x_1$, the fitness values (f_{100} , f_{010} , f_{101} , and f_{011}) are decremented by this amount. As there exists a finite spacing between each adjacently-ranked candidate, there is such a change we can make to $\alpha_{\{0,1\}}$ which will not permute the ranks.

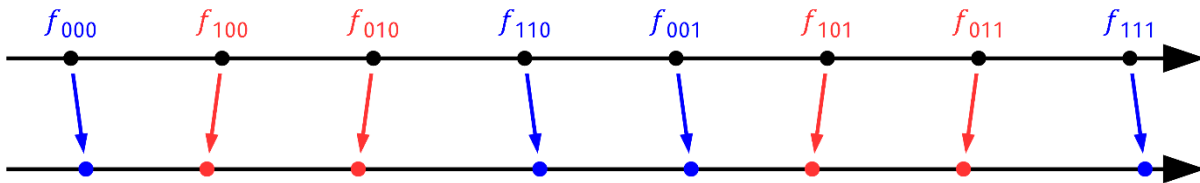


Figure 15 – An instance of the class C_{BINVAL^3} (above), transformed by adding a small, positive bivariate term $\alpha_{\{0,1\}} > 0$ (below). As long as the term is small enough, the relative ranks of the candidates are undisturbed, and the function is still in the class C_{BINVAL^3} .

In contrast, if we perform the same transformation of adding a small bivariate term to the benchmark class $C_{\text{ONEMAX}^3} = [7 \ 4 \ 4 \ 1 \ 4 \ 1 \ 1 \ 0]$ the same four values are incremented with the remaining decremented, however, we are now guaranteed to have generated an instance of a different class, not C_{ONEMAX^3} since f_{001} has moved away from f_{010} and f_{100} , additionally, f_{110} has moved away from f_{011} and f_{101} . This is now an instance of $[7 \ 4 \ 4 \ 3 \ 6 \ 1 \ 1 \ 0]$. In general all cases with zero coefficients will split in this way when the zero coefficient is changed.

However, if we set all three coefficients to the same value, we get back to the original class. Figure 16 shows three transformations applied to C_{ONEMAX^3} , first adding a small positive value to $\alpha_{\{0,1\}}$, then $\alpha_{\{0,2\}}$, then $\alpha_{\{1,2\}}$, each of the same magnitude. Each transformation moves the instance outside of the current class, with the last restoring to the original class.

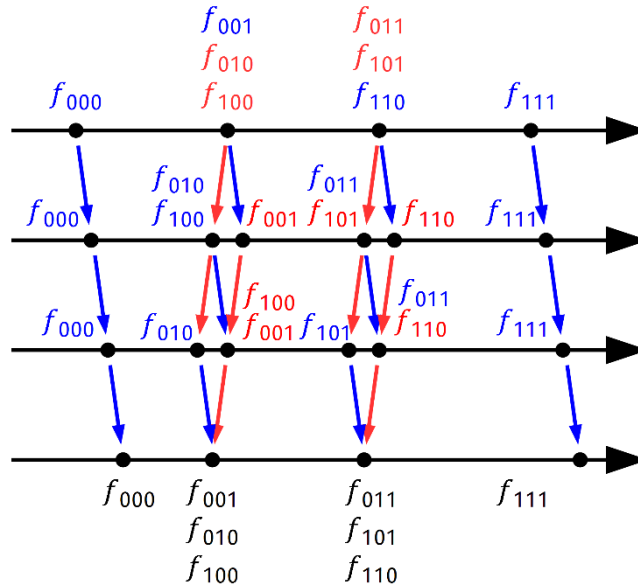


Figure 16 – An instance of the class C_{ONEMAX^3} (top row), through a series of transformations, but adding a small, positive value $d > 0$ to the zero bivariate terms $\alpha_{\{0,1\}} = d$ (second row), $\alpha_{\{0,1\}} = \alpha_{\{0,2\}} = d$ (third row), then $\alpha_{\{0,1\}} = \alpha_{\{0,2\}} = \alpha_{\{1,2\}} = d$ (fourth row).

For the Walsh structures in 3-bits, we have 7 possible structure elements. We adopt the enumeration convention described in Table 39.

Condition	Binary Enumeration	Hex Enumeration
$\alpha_{\{0\}} \neq 0$	0000001	01
$\alpha_{\{1\}} \neq 0$	0000010	02
$\alpha_{\{0,1\}} \neq 0$	0000100	04
$\alpha_{\{2\}} \neq 0$	0001000	08
$\alpha_{\{0,2\}} \neq 0$	0010000	10
$\alpha_{\{1,2\}} \neq 0$	0100000	20
$\alpha_{\{0,1,2\}} \neq 0$	1000000	40

Table 39 – Enumeration of structure elements for 3-bits.

These can be combined into 128 possible structures which are enumerated in Figure 17.

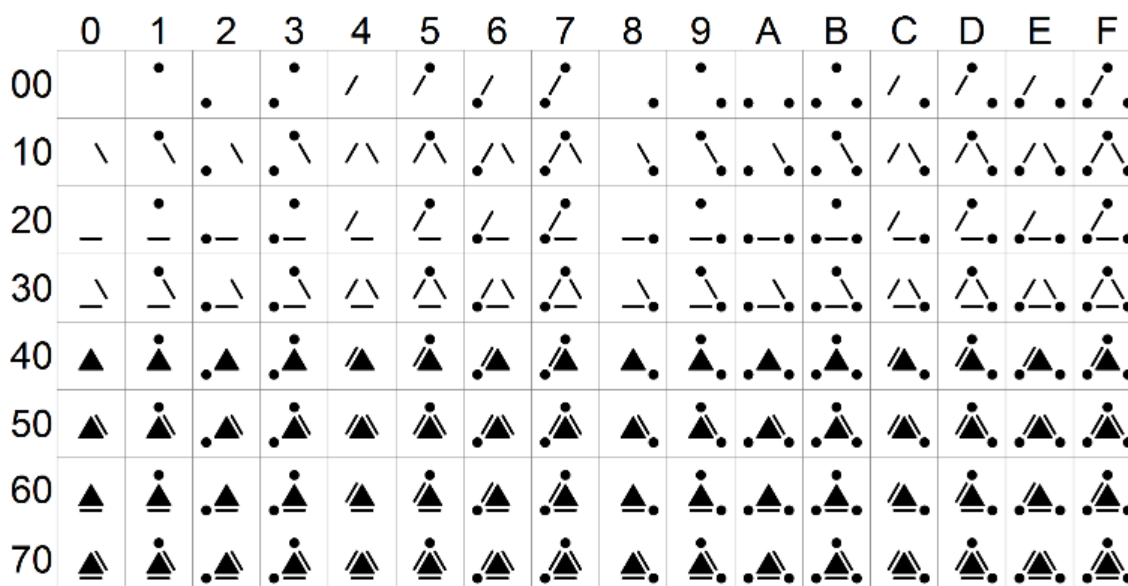


Figure 17 – All 128 possible Walsh structures for 3-bits, enumerated 00 to 7F. The ID of a given structure is the sum of the row header with the column header.

In 2-bit classes, recall that every Walsh coefficient was either necessarily zero, necessarily non-zero, or optional (conditioned on a delta condition). When we move from 2-bit classes to 3-bit classes we see the emergence of another case we will call *conditionally-necessary* structures.

Recall that in 2-bits, the ONE_{MAX}^2 and $BINVAL^2$ functions both belonged to the $\{3, 7\}$ Walsh family, meaning that both univariate terms were necessary and the bivariate term was optional. This trend continues with $BINVAL^3$ in 3-bits with every non-univariate term being optional as shown in Figure 18.

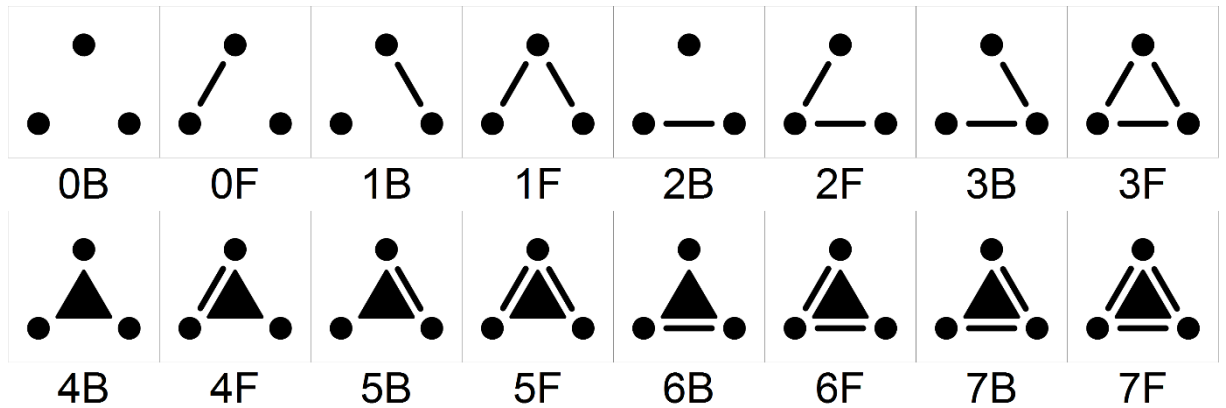


Figure 18 – All possible Walsh structures for C_{BINVAL^3} .

By contrast, however, the ONE_{MAX}^3 function despite having the same minimal structure (0B) and same maximal structure (7F) than $BINVAL^3$, has only two other possible structures, shown in the Figure 19. The key difference is that the bivariate terms are all conditionally-necessary, one is non-zero if and only if the others are non-zero.

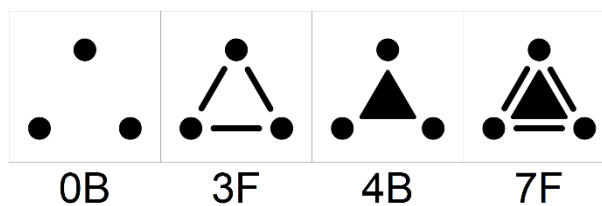


Figure 19 – All possible Walsh structures for $C_{ONE_{MAX}^3}$.

We note that these conditionally-necessary structures do not preclude the existence of a unique minimal structure in this instance, however, we can find instances where there is no clear minimal structure.

In the case of the class $[0\ 1\ 2\ 4\ 2\ 4\ 4\ 7]$, we see that the minimal structure is $2B$ (all univariates and one bivariate $\alpha_{\{1,2\}}$), however the bivariate term can be removed and replaced with two other supporting bivariate terms $\alpha_{\{0,1\}}$ and $\alpha_{\{1,2\}}$, in this sense the bivariate term $\alpha_{\{1,2\}}$ is not simply necessary or unnecessary, but conditional on the presence of the other two bivariate terms. The possible structures are shown in Figure 20.

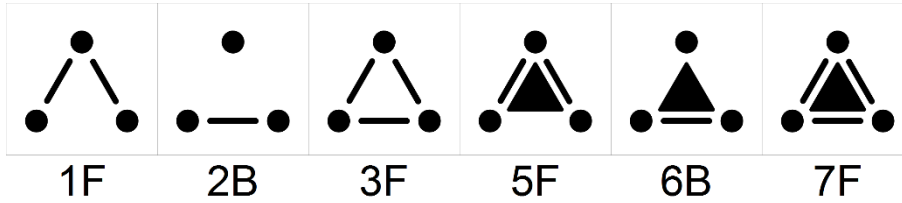


Figure 20 – All possible Walsh structures for $C_{[0\ 1\ 2\ 4\ 2\ 4\ 4\ 7]}$.

In the case of the class $[2\ 0\ 1\ 5\ 5\ 2\ 4\ 5]$, at least 6 of the possible 7 structure elements must be non-zero. The candidates for allowable zero elements are the trivariate term $\alpha_{\{0,1,2\}}$, the bivariate term $\alpha_{\{0,2\}}$, or the univariate term $\alpha_{\{0\}}$ however, no two or three of these may be zero together. The possible structures are shown in Figure 21.

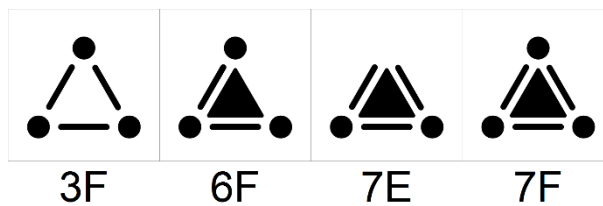


Figure 21 – All possible Walsh structures for $C_{[2\ 0\ 1\ 5\ 5\ 2\ 4\ 5]}$.

6.5 Precedence Networks and Precedence Profiles

As with the 2-bit classes, we can construct all possible precedence networks for evaluating 3-bit functions. First we limit the profiles to fully-specified precedence networks. We explore this concept as a measure of complexity of functions.

A fully-specified precedence network is a precedence network which contains one prescribed topological ordering over the linkage groups, these are listed in Table 40.

Label	Network	Cost	Label	Network	Cost
A0	$X_0X_1X_2$	8	D0	$X_0 \rightarrow X_1 \rightarrow X_2$	4
B0	$X_0 \rightarrow X_1X_2$	5	D1	$X_0 \rightarrow X_2 \rightarrow X_1$	4
B1	$X_1 \rightarrow X_0X_2$	5	D3	$X_1 \rightarrow X_2 \rightarrow X_0$	4
B2	$X_2 \rightarrow X_0X_1$	5	D4	$X_2 \rightarrow X_0 \rightarrow X_1$	4
C0	$X_0X_1 \rightarrow X_2$	5	D2	$X_1 \rightarrow X_0 \rightarrow X_2$	4
C1	$X_0X_2 \rightarrow X_1$	5	D5	$X_2 \rightarrow X_1 \rightarrow X_0$	4
C2	$X_1X_2 \rightarrow X_0$	5			

Table 40 – All 3-bit fully-specified precedence networks.

For instance, B1 specifies that variable X_1 should be set by trying both possible assignments against an arbitrary setting of the other variables, this takes 2 function evaluations, then the variables X_0X_2 should be set together by evaluating all 4 assignments of those variables against the optimal setting for the fixed variable. Since one such assignment has been tried already, this takes 3 function evaluations. In total, B1 takes 5 function evaluations. We write this precedence network as $X_1 \rightarrow X_0X_2$ and the network set $\{B0, B1, B2\}$ can be expressed in the form $\bullet \rightarrow \bullet\bullet$.

Figure 22 shows the fully-specified networks and shows which networks will necessarily solve instances of classes solved to another. The required number of function evaluations (cost) is shown. The sequence of steps in the network gives an upper bound on cost to be guaranteed an optimal solution of the problem using perturbation steps. The costs of evaluating a network may be calculated by 2^k for the first partition (where k is the number of variables in the first partition) plus $2^k - 1$ per each successive partition (where k is the number of variables in the current partition.)

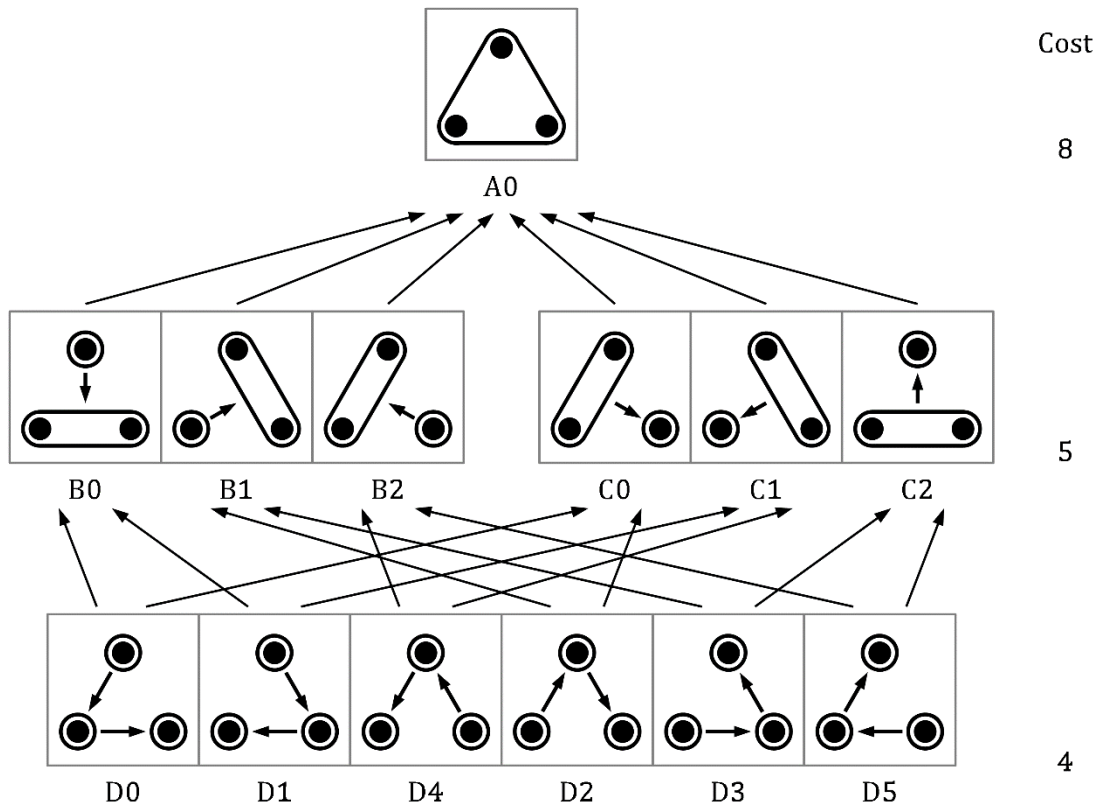


Figure 22 – All 3-bit fully-specified precedence networks represented pictorially. Connecting arrows show the relationship of which network will solve every class solvable by another network

When we test every possible 3-bit class, we see that every instance in which a precedence network is predicted to solve a function, it does. However, there are many additional cases in which a precedence network solves a function despite not being predicted by the specified hierarchy. We may expect that the directed ordinal linkage would specify which precedence profiles would be applicable, however, this is not the case.

A good example is the class of the benchmark function $CHECK_{1D}^3$. The directed ordinal linkage suggests that the profile $X_1 \rightarrow X_0X_2$ would not solve the classes in all runs, however, we can show that it does, in all runs. On evaluating X_1 there is an even chance of either 0 or 1 being chosen, however, since this problem has two global optima – one in which X_1 is set to 0, and one in which X_1 is set to 1 – a global optimum is always reached. This is diagrammed in Figure 23.

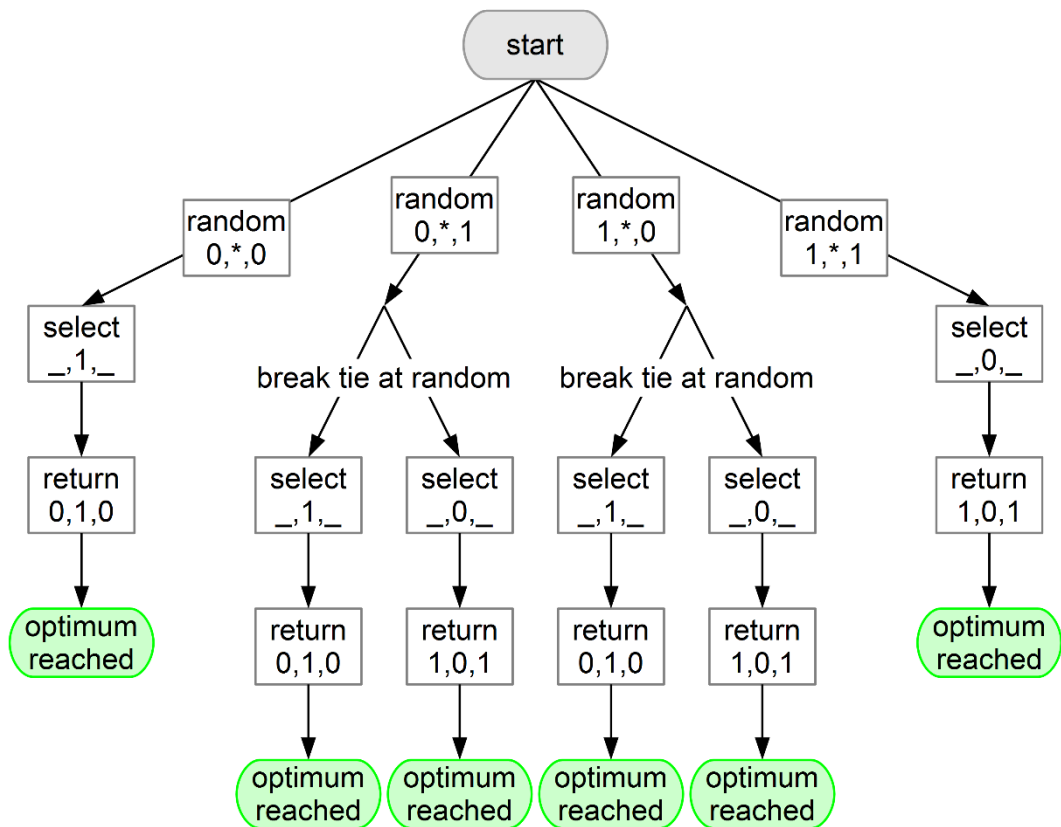


Figure 23 – All possible optimisation paths for $CHECK_{1D}^3$ problem under the precedence network B1 ($X_1 \rightarrow X_0X_2$) showing that the optimum is always reached.

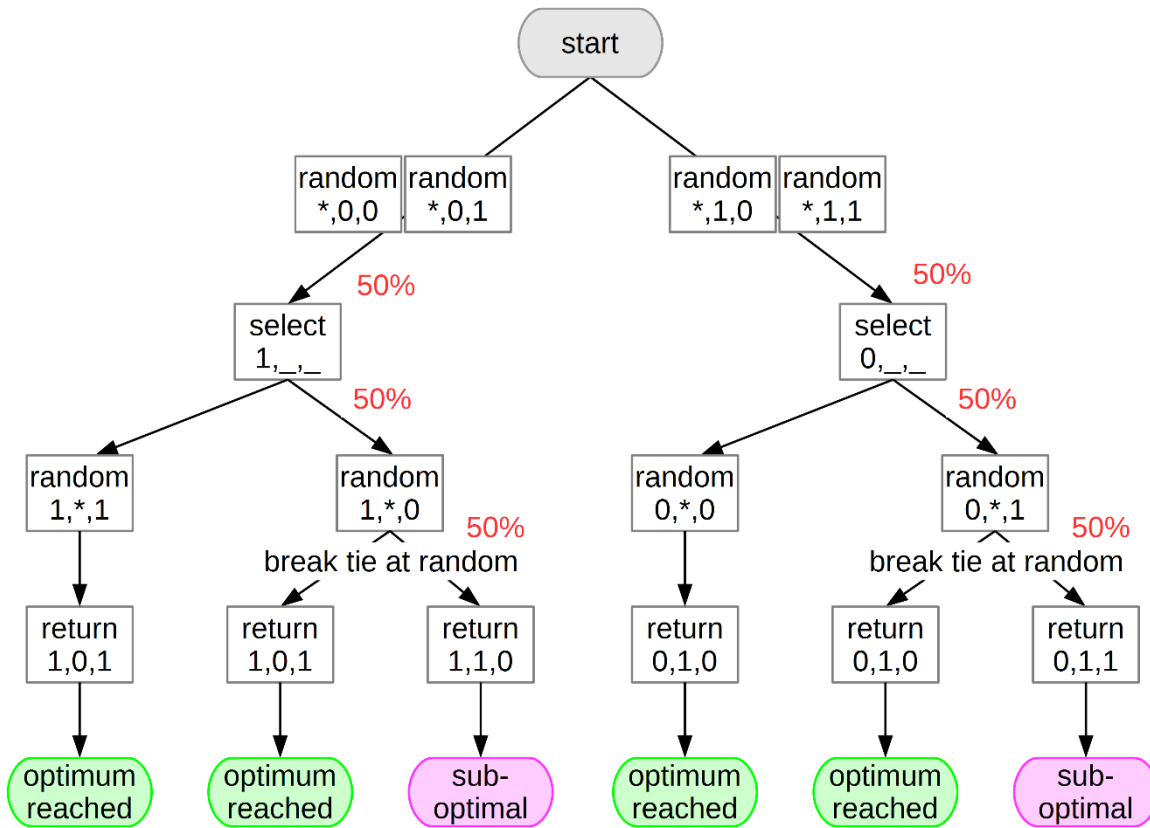


Figure 24 – Under the precedence network $D_0 (X_0 \rightarrow X_1 \rightarrow X_2)$, there is a chance of $1/4$ ($50\% \times 50\% \times 50\% + 50\% \times 50\% \times 50\%$) that an optimum is not reached for $CHECK_{1D}^3$ problem, hence the probability of reaching the optimum is $3/4$.

From this we derive the idea of the precedence profile, since a function does not fit exactly in one category. The probability of arriving at the global optimum (assuming maximisation criteria) for a given function and precedence network can be calculated, and when this is done for all possible fully-specified precedence networks, we refer to this as the *precedence profile* of the function for a given length.

Table 41 shows the precedence profiles of eight benchmark functions.

Function	Precedence Network												
	A0	B0	B1	B2	C0	C1	C2	D0	D1	D4	D2	D3	D5
$\text{CONST}_c^3 / \text{ONEMAX}^3 / \text{ZEROMAX}^3 / \text{BINVAL}^3$	1	1	1	1	1	1	1	1	1	1	1	1	1
CHECK_{1D}^3	1	1	1	1	1	1	1	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{3}{4}$
LEADING^3	1	1	$\frac{3}{4}$	$\frac{5}{8}$	1	$\frac{3}{4}$	$\frac{5}{8}$	1	$\frac{3}{4}$	$\frac{5}{8}$	$\frac{3}{4}$	$\frac{9}{16}$	$\frac{15}{32}$
NEEDLE^3	1	$\frac{5}{8}$	$\frac{5}{8}$	$\frac{5}{8}$	$\frac{5}{8}$	$\frac{5}{8}$	$\frac{5}{8}$	$\frac{15}{32}$	$\frac{15}{32}$	$\frac{15}{32}$	$\frac{15}{32}$	$\frac{15}{32}$	$\frac{15}{32}$
TRAP_3^3	1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$

Table 41 – 3-bit precedence profiles of common benchmark functions for a maximisation objective. Functions are sorted in increasing order of difficulty (based on average of probabilities).

We observe from Table 41 that the order of decreasing average probability tracks with order of increasing complexity in the sense that we first have the univariate problems, then the bivariate, then the multivariate. Within the multivariate, the isolated function (needle in haystack) is harder than the leading ones, and the trap function is harder than the isolated function. We can regard this average as one possible measure of complexity for a given problem length.

The simplest set of classes of truly 3-bit functions are the univariate functions - those in which each variable is considered separately and the resulting affect additive, i.e. the linkage partition completely separates the effect of each variable.

Here we see that the Needle in Haystack function is classified as easier than the 3-trap under our classification. The needle in haystack function may be considered the most difficult function to optimise as there is no structure learnable without exhaustive evaluation. This is not the case, as the trap function is more difficult to optimise as it contains counter-productive structure which is actively misleading to learn. Thus our view of complexity under precedence profiles highlights this. This is also observed by Kallel et al. [86].

The 3-bit Checkerboard 1D problem does not have a precedence profile as follows from the function's directed ordinal linkage. Although the function is composed of a single linkage group, we see that it is solvable by a larger number of networks than just A0. This is because the checkerboard problem has two global optima. The existence of two global optima means that some of the structure which may have to be learned to accurately capture this complexity is unnecessary if the objective is to arbitrarily locate one optima or the other.

By contrast in Figure 24 we see a failure to guarantee an optimal solution. The sub-optimal solution exists if the assigned value for X_0 differs from the random background setting of X_2 when assigning X_1 (this occurs with 1/2 probability) and the value which is the same as X_1 is chosen (this occurs with 1/2 probability), hence, the search fails with 1/4 probability.

We see that for 3-bit classes, this notion of precedence profiles correlates with expected function difficulty classification as described. Hence, it may be reasonable to assume that an extension of this idea of a precedence network to higher dimensions may be useful. This is discussed in more detail under further work.

6.6 Equivalent Average Costs Network Sets

In this section we explore the probability of successfully optimising a function using a given fully-specified precedence network, subject to the function being an arbitrary instance of a rank equivalence class chosen *uniformly at random* from the set of 3-bit rank equivalence classes. Note that this is distinct from selecting functions uniformly at random, in which case, the class would not be uniformly selected.

Recall that for fully-specified 2-bit networks, the cost can be 3 or 4 and that the two fully-specified cost 3 networks were a permutation of each other. Hence, on average across the set of all 2-bit classes it is trivial that we would expect the same performance from these two permutations.

For fully-specified 3-bit networks, there are two distinct network sets (B and C) which share a common cost (5 function evaluations). The costs for each network set is below in Table 42.

Set	Networks	Cost
A	A0	$(2^3) = 8$
B	B0, B1, B2	$(2^2) + (2^1 - 1) = 5$
C	C0, C1, C2	$(2^1) + (2^2 - 1) = 5$
D	D0, D1, D4, D2, D3, D5	$(2^1) + (2^1 - 1) + (2^1 - 1) = 4$

Table 42 – Cost for each fully-specified 3-bit precedence network.

Each network in the set union $B \cup C = \{B0, B1, B2, C0, C1, C2\}$ has the same cost (5 fitness evaluations). We can calculate the number of classes which have each given probability P and calculate the expected probability and fully-specified network in the set B or the set C to find the optimum on a randomly-selected function class. This is shown in Table 43.

Probability (P)	B0, B1, B2		C0, C1, C2	
	Classes (N)	$P \times N$	Classes (N)	$P \times N$
1/4	34 792	8 698	-	-
3/8	22 056	8 271	-	-
1/2	120 996	60 498	288 548	144 274
5/8	50 304	31 440	600	375
2/3	-	-	6 336	4 224
3/4	134 788	101 091	38 844	29 133
5/6	-	-	864	720
7/8	28 248	24 717	24	21
1	154 651	154 651	210 619	210 619
Sum	545 835	389 366	545 835	389 366
Weighted Average	$\frac{389\,366}{545\,835}$		$\frac{389\,366}{545\,835}$	

Table 43 – For 3-bit network sets $B = \{B0, B1, B2\}$ and $C = \{C0, C1, C2\}$, the average probability of finding the global optimum is the same (approx. 71.3%) when selecting a class uniformly at random.

We see that although there are large differences in distribution of probability between the two network sets, the weighted average is the same ($389366/545835$). We see from the D networks in Table 44 that the same average does not hold for a different cost network, since the lower computational effort corresponds to a lower expectation of finding a global optimum.

Probability (P)	D0, D1, D4, D2, D3, D5	
	Classes (N)	$P \times N$
$1/8$	15 080	1 885
$3/16$	14 272	2 676
$1/4$	65 464	16 366
$9/32$	2 776	$780 \frac{3}{4}$
$5/16$	20 232	$6 322 \frac{1}{2}$
$3/8$	69 224	25 959
$7/16$	9 640	$4 217 \frac{1}{2}$
$15/32$	4 248	$1 991 \frac{1}{4}$
$1/2$	91 196	45 598
$17/32$	200	$106 \frac{1}{4}$
$9/16$	10 848	6 102
$19/32$	1 104	$655 \frac{1}{2}$
$5/8$	33 640	21 025
$21/32$	3 880	$2 546 \frac{1}{4}$
$11/16$	7 272	$4 999 \frac{1}{2}$
$23/32$	200	$143 \frac{3}{4}$
$3/4$	91 780	68 835
$25/32$	200	$156 \frac{1}{4}$
$13/16$	9 576	$7 780 \frac{1}{2}$
$27/32$	1 464	$1 235 \frac{1}{4}$
$7/8$	28 048	24 542
$29/32$	2 136	$1 935 \frac{3}{4}$
$15/16$	4 192	3 930
$31/32$	704	682
1	58 459	58 459
Sum	545 835	308 930
Weighted Average	$\frac{308\,930}{545\,835} = \frac{61\,786}{109\,167}$	

Table 44 – For 3-bit precedence network set $D = \{D0, D1, D4, D2, D3, D5\}$, success probability when selecting a class at random.

6.7 Precedence Networks Hierarchy

Having looked at the fully-specified network sets, we wish to introduce analysis which goes beyond only the fully-specified precedence networks. We introduce the simplification by considering only the sets invariant under permutations. The work in this section was first published in [2].

As shown in section 6.6, where the fully-specified precedence networks for 3-bits were given, we see that network sets can be arranged in a hierarchy. This hierarchy exists because any two variables which can be solved given a prescribed ordering $X_i \rightarrow X_j$ can also be solved by exhaustive evaluation of those two variables $X_i X_j$. If we include *under-specified networks* (networks which are not fully specified), we also find variables which can be optimised in any order $X_i + X_j$, which can be given an order $X_i \rightarrow X_j$ or exhaustively evaluated $X_i X_j$.

Allowing under-specified networks, there are 9 network sets. The list of all relationships between a 3-bit network set and another 3-bit network set which can be used to solve all classes the first can is given in Table 45. Examples of each is given since some relationships may not be self-evident without specifying the necessary re-permutation of the variables. The hierarchy produced by these relationships is shown in Figure 25.

General Network Set	Specified Network Set	General Example	Specific Example
$\bullet + \bullet + \bullet$	$\bullet \rightarrow \bullet + \bullet$	$X_0 + X_1 + X_2$	$X_0 \rightarrow X_1 + X_2$
$\bullet \rightarrow \bullet + \bullet$	$(\bullet + \bullet) \rightarrow \bullet$	$X_0 \rightarrow X_1 + X_2$	$(X_0 + X_2) \rightarrow X_1$
$\bullet \rightarrow \bullet + \bullet$	$\bullet + \bullet \bullet$	$X_0 \rightarrow X_1 + X_2$	$X_0 \rightarrow X_1 X_2$
$\bullet \rightarrow \bullet + \bullet$	$\bullet \rightarrow (\bullet + \bullet)$	$X_0 \rightarrow X_1 + X_2$	$X_0 \rightarrow (X_1 + X_2)$
$(\bullet + \bullet) \rightarrow \bullet$	$\bullet \rightarrow \bullet \rightarrow \bullet$	$(X_0 + X_1) \rightarrow X_2$	$X_0 \rightarrow X_1 \rightarrow X_2$
$\bullet \rightarrow (\bullet + \bullet)$	$\bullet \rightarrow \bullet \rightarrow \bullet$	$X_0 \rightarrow (X_1 + X_2)$	$X_0 \rightarrow X_1 \rightarrow X_2$
$\bullet + \bullet \bullet$	$\bullet \rightarrow \bullet \bullet$	$X_0 + X_1 X_2$	$X_0 \rightarrow X_1 X_2$
$\bullet + \bullet \bullet$	$\bullet \bullet \rightarrow \bullet$	$X_0 + X_1 X_2$	$X_1 X_2 \rightarrow X_0$
$\bullet \rightarrow \bullet \rightarrow \bullet$	$\bullet \rightarrow \bullet \bullet$	$X_0 \rightarrow X_1 \rightarrow X_2$	$X_0 \rightarrow X_1 X_2$
$\bullet \rightarrow \bullet \rightarrow \bullet$	$\bullet \bullet \rightarrow \bullet$	$X_0 \rightarrow X_1 \rightarrow X_2$	$X_0 X_1 \rightarrow X_2$
$\bullet \rightarrow \bullet \bullet$	$\bullet \bullet \bullet$	$X_0 \rightarrow X_1 X_2$	$X_0 X_1 X_2$
$\bullet \bullet \rightarrow \bullet$	$\bullet \bullet \bullet$	$X_0 X_1 \rightarrow X_2$	$X_0 X_1 X_2$

Table 45 – All 3-bit network set, with network sets which can be substituted.

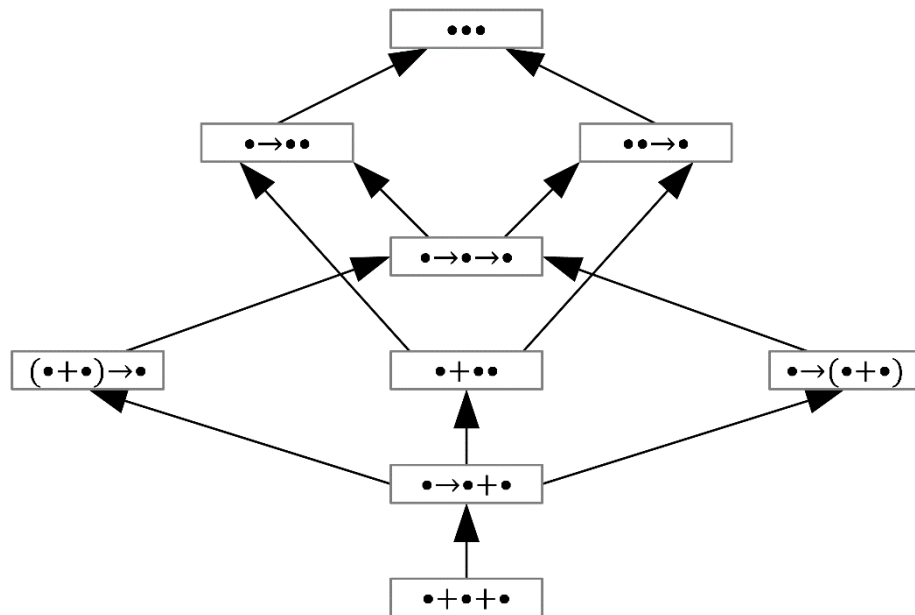


Figure 25 – Hierarchy of 3-bit network sets.

6.8 Parallelisation of Precedence Networks

The linkage groups for a precedence network can be processed in parallel. Here we shown how this can be done for 3-bit networks. We will see that the minimum number of time steps is dictated by the length of the ordering $\bullet \rightarrow \bullet$. The work in this section was first published in [2].

In the case of the network set $\bullet + \bullet + \bullet$ we can process two values for each variable. This means evaluating any single arbitrary assignment (e.g. f_{000}) and comparing each of that assignment with one variable flipped (i.e. in this case f_{100} , f_{010} , and f_{001}) to find the optimal setting for each variable. In this case, 4 evaluations are necessary, and the 4 values of \mathbf{X} can be chosen in advance and processed in parallel.

In the case of the network set $\bullet \bullet \bullet$ we must evaluate all 8 possible assignments of the variables to determine the correct setting of any. Since we know the 8 values of \mathbf{X} which must be tried, they may be processed in parallel.

In the case of a network set involving a directed ordinal linkage (e.g. $\bullet \rightarrow \bullet \rightarrow \bullet$) we do not know ahead of time which 4 values must be tried, so we evaluate an arbitrary assignment of the variables (e.g. f_{000}) and the same with the first variable in the chain flipped (i.e. in this case if the first variable is X_0 we would evaluate f_{100}). These 2 evaluations may be done in parallel. Then we evaluate the result of flipping the next variable given the optimum of the first (i.e. f_{*10} where $*$ is the known optimal setting). Then we evaluate the result of flipping the last variable (i.e. f_{**1}). Since we are using information about the result of earlier evaluations, at least three time steps are required to optimise the function in 4 fitness evaluations.

This constraint may be changed by using a different precedence network. As shown, a more costly network from one of the sets $\bullet \bullet \rightarrow \bullet$, $\bullet \rightarrow \bullet \bullet$, or $\bullet \bullet \bullet$ may be used (requiring 5, 5, or 8 function evaluations respectively, but being parallelisable in 2, 2, or 1 time steps respectively). This represents a trade-off between number of function evaluations required and number of time steps required.

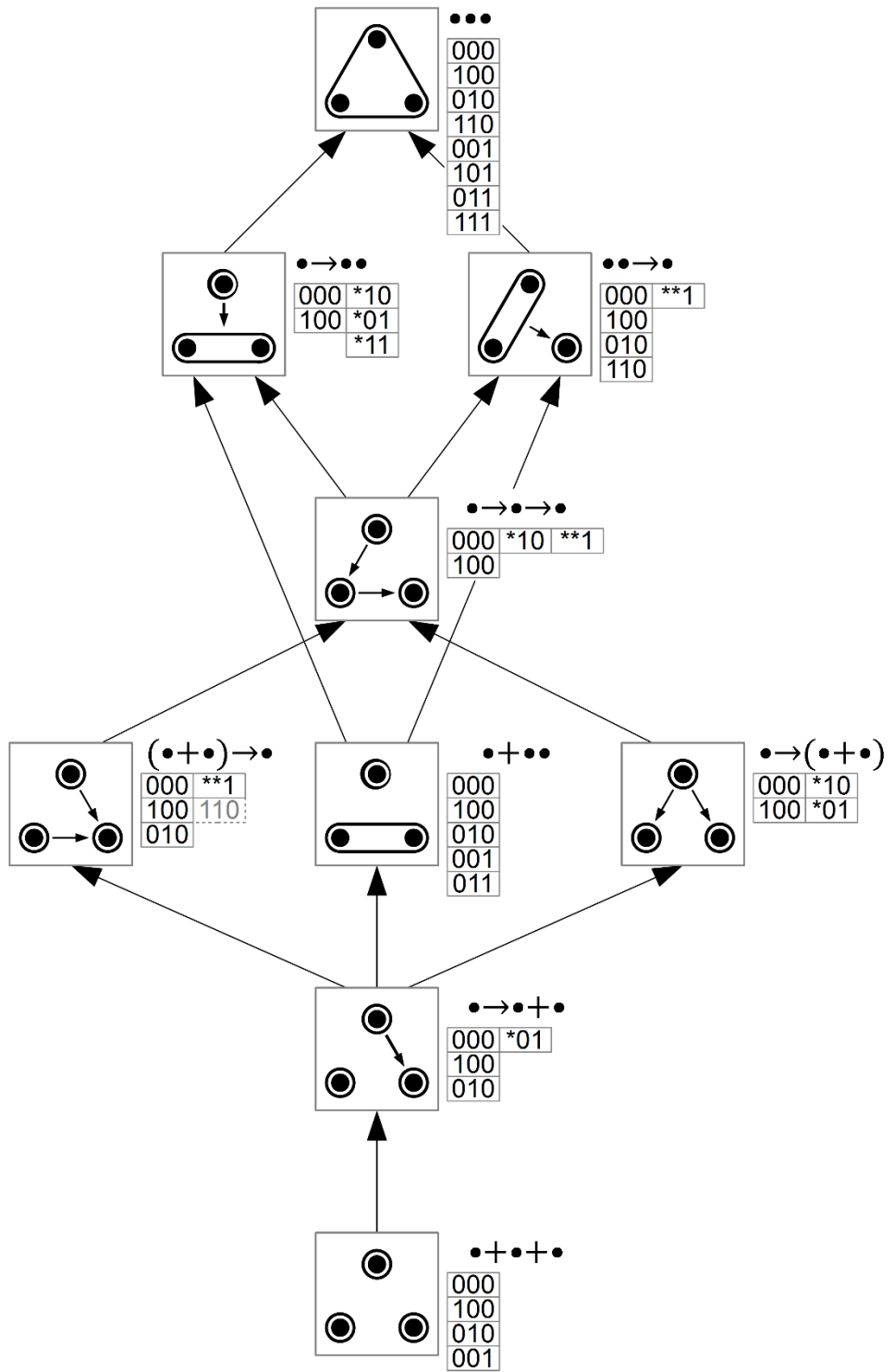


Figure 26 – All 3-bit networks sets, with the parallelisation steps for the shown example network in each case. A column represents all function evaluations which may be done in parallel. If there are multiple columns, each column must wait until the columns to its left are complete before starting. * represents the known optimal setting from an earlier column.

6.9 Delta Linkage Detection

In this section we look at whether the same approach taken in section 5.6 (p. 98) to relate linkage to the delta expansion can be extended to 3-bit classes. We show, by counter-example, that the delta matrix cannot be used to predict the linkage for 3-bits.

For 3-bit functions we can use the delta matrix Δ to represent the delta values in the same way as for 2-bits as in (53).

$$\Delta = \begin{bmatrix} D_{111}^0 & D_{111}^1 & D_{111}^2 & \dots & D_{111}^6 \\ D_{011}^0 & D_{011}^1 & D_{011}^2 & \dots & D_{011}^6 \\ D_{101}^0 & D_{101}^1 & D_{101}^2 & \dots & D_{101}^6 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{000}^0 & D_{000}^1 & D_{000}^2 & \dots & D_{000}^6 \end{bmatrix} \quad (123)$$

$$\text{where } D_{\mathbf{x}}^r = \begin{cases} 1, & R_f(\mathbf{x}) > r \\ 0, & \text{otherwise} \end{cases}$$

Here we could select the rows of interest to our linkage as specified by (124).

$$\begin{aligned} V_{\{0,1\}} &= [H_3]_{row:3} \cdot \Delta \\ V_{\{0,2\}} &= [H_3]_{row:5} \cdot \Delta \\ V_{\{1,2\}} &= [H_3]_{row:6} \cdot \Delta \\ V_{\{0,1,2\}} &= [H_3]_{row:7} \cdot \Delta \end{aligned} \quad (124)$$

We observe that no combination of these four terms can be related directly to the non-monotonicity-detecting linkage detection. This can be shown by counter-example.

We examine two classes C_a and C_b which have the Walsh family $\{3F, 7F\}$. Structure 7F refers to having all terms non-zero, structure 3F refers to having the trivariate term zero with all other terms non-zero. Hence this family describes functions which can be instantiated with a bivariate function where all bivariate terms are necessarily non-zero. The two classes are listed in Table 46 (p. 131).

Class	$\mathcal{L}_o(0,1)$	$\mathcal{L}_o(1,0)$	$\mathcal{L}_o(0,2)$	$\mathcal{L}_o(2,0)$	$\mathcal{L}_o(1,2)$	$\mathcal{L}_o(2,1)$	Linkage
$C_a = [0 \ 1 \ 1 \ 4 \ 1 \ 4 \ 4 \ 4]$	Y	Y	Y	Y	Y	Y	$X_0X_1X_2$
$C_b = [2 \ 0 \ 1 \ 2 \ 6 \ 2 \ 2 \ 6]$	Y	Y	N	N	N	N	$X_0X_1 + X_2$

Table 46 – Example classes C_a and C_b with different ordinal linkage \mathcal{L}_o , both classes belonging to the Walsh family {3F, 7F}.

In this case the Walsh family indicates that the first two variables are required to be linked to the third, whereas the perturbations indicate no linkage.

Note that if we follow the precedence network $X_0X_1 + X_2$ for C_b we arrive at a global optimum every time, regardless of whether the linkage ground X_0X_1 is optimised first or the linkage group X_2 is optimised first. The result will be $[0 \ 0 \ 0]$ or $[1 \ 1 \ 0]$ each with probability 0.5; these candidates are both global optima for an instance of C_b .

The reason the Walsh structure requires linkage between these two linkage groups is seen by the result of perturbing two variables (X_0X_1) simultaneously from $[0 \ 1 \ *]$ to $[1 \ 0 \ *]$. If done when $X_2 = 0$ this two-bit perturbation will not change the fitness, if done when $X_2 = 1$ this two-bit perturbation will increase the fitness from the lowest rank to the second-lowest rank.

None of the candidates involved in this perturbation is a global optimum, hence ignoring this linkage should not degrade an algorithm's ability to optimise the function. From this perspective, we can see the Walsh coefficients responsible for maintaining this structure are unnecessary for optimisation yet necessary to maintain the rank equivalence class.

In Figure 27 the class is illustrated as a cube. Each square side has matching signs on parallel edges. To detect linkage, the diagonal signs across two parallel faces must be compared. This shows that although this class consists of three variables which are all linked to one another (due to the presence of all three bivariate terms), a two-bit perturbation is required to detect this.

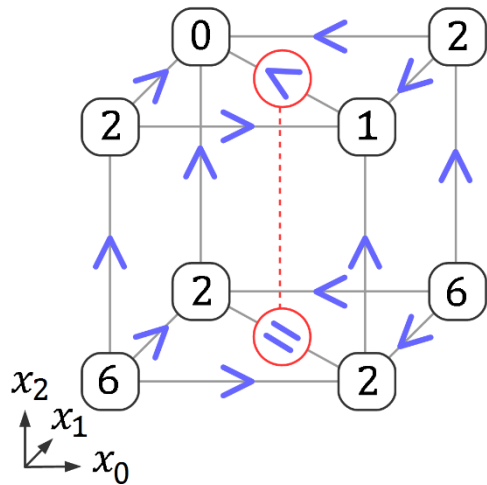


Figure 27 – The class [2 0 1 2 6 2 2 6] with the detection of linkage via two-bit perturbation highlighted.

6.10 Structural Coherence

In this chapter we consider the difficulty of learning the linkage of each 3-bit class for an EDA. This section is the presentation of the result on 3-bit classes from the work presented in section 5.7 (on 2-bit classes); this result was produced in collaboration with Dr Alexander Brownlee, University of Stirling and first published in [3].

In sections section 5.7 (p. 100), we show the minimum populations size required to detect linkage for all 2-bit classes. Recall that for some classes with no linkage between the variables, a spurious correlation is falsely detected (false positive) for larger population sizes. For the set of 3-bit classes we present a summary of the success for population sizes 100 and 500. We compare tournament (best and worst) and truncation selection (top, bottom, and top+bottom, for proportions 0.25, 0.33, and 0.5).

We use the method of calculating the pairwise linkage given in section 5.7 and compare the results for the three possible bivariate linkages. Probabilities for tournament selection were explicitly calculated using all 256 possible tournaments. Probabilities for truncation selection were explicitly calculated by determining the candidates in the top $\frac{1}{4}$, $\frac{1}{3}$ or $\frac{1}{2}$ of selection.

We observe that truncation selection (top) performs equally to truncation selection (bottom) across the set of 3-bit classes, and that tournament selection performs equally to inverse tournament selection. This matches the result for 2-bit classes. However, we observe that the selection size affects the success for top-and-bottom truncations for 3-bits whereas it does not for 2-bit classes. This is because for 3-bit classes, top-and-bottom truncation misses information carried by middle-ranked candidates.

For this analysis we assume that the minimal structure of each class is the structure with the lowest ID in the Walsh family. We discuss the limitations of this approach in further work.

		Linkage 95%		No Linkage 5%		Linkage 95%		No Linkage 5%		
		Population Size 100				Population Size 500				
		True Pos.	False Neg.	True Neg.	False Pos.	True Pos.	False Neg.	True Neg.	False Pos.	
Tournament (Best)	$\mathcal{L}_O(0,1)$	42%	53%	5%	0%	71%	24%	3%	2%	
	$\mathcal{L}_O(0,2)$	42%	53%	5%	0%	71%	24%	3%	2%	
	$\mathcal{L}_O(1,2)$	42%	52%	5%	0%	71%	24%	4%	2%	
Tournament. (Worst)	$\mathcal{L}_O(0,1)$	42%	53%	5%	0%	71%	24%	3%	2%	
	$\mathcal{L}_O(0,2)$	42%	53%	5%	0%	71%	24%	3%	2%	
	$\mathcal{L}_O(1,2)$	42%	52%	5%	0%	71%	24%	4%	2%	
Truncation (Top)	0.25	$\mathcal{L}_O(0,1)$	40%	56%	3%	1%	40%	55%	3%	1%
		$\mathcal{L}_O(0,2)$	40%	55%	4%	1%	40%	55%	4%	1%
		$\mathcal{L}_O(1,2)$	39%	55%	4%	1%	40%	54%	4%	1%
	0.33	$\mathcal{L}_O(0,1)$	70%	26%	2%	3%	72%	23%	2%	3%
		$\mathcal{L}_O(0,2)$	69%	26%	2%	3%	72%	23%	2%	3%
		$\mathcal{L}_O(1,2)$	69%	25%	2%	3%	72%	23%	2%	3%
	0.5	$\mathcal{L}_O(0,1)$	67%	28%	2%	3%	72%	24%	2%	3%
		$\mathcal{L}_O(0,2)$	67%	28%	2%	3%	72%	23%	2%	3%
		$\mathcal{L}_O(1,2)$	67%	28%	2%	3%	72%	23%	2%	3%
Truncation (Bottom)	0.25	$\mathcal{L}_O(0,1)$	40%	56%	3%	1%	40%	55%	3%	1%
		$\mathcal{L}_O(0,2)$	40%	55%	4%	1%	40%	55%	4%	1%
		$\mathcal{L}_O(1,2)$	39%	55%	4%	1%	40%	54%	4%	1%
	0.33	$\mathcal{L}_O(0,1)$	70%	26%	2%	3%	72%	23%	2%	3%
		$\mathcal{L}_O(0,2)$	69%	26%	2%	3%	72%	23%	2%	3%
		$\mathcal{L}_O(1,2)$	69%	25%	2%	3%	72%	23%	2%	3%
	0.5	$\mathcal{L}_O(0,1)$	67%	28%	2%	3%	72%	24%	2%	3%
		$\mathcal{L}_O(0,2)$	67%	28%	2%	3%	72%	23%	2%	3%
		$\mathcal{L}_O(1,2)$	67%	28%	2%	3%	72%	23%	2%	3%
Truncation (T+B)	0.25	$\mathcal{L}_O(0,1)$	11%	84%	4%	0%	11%	84%	4%	0%
		$\mathcal{L}_O(0,2)$	11%	84%	5%	0%	11%	84%	5%	0%
		$\mathcal{L}_O(1,2)$	11%	83%	5%	0%	11%	83%	5%	0%
	0.33	$\mathcal{L}_O(0,1)$	36%	60%	4%	1%	40%	56%	3%	1%
		$\mathcal{L}_O(0,2)$	36%	59%	4%	1%	40%	55%	4%	1%
		$\mathcal{L}_O(1,2)$	36%	59%	4%	1%	40%	55%	4%	1%
	0.5	$\mathcal{L}_O(0,1)$	40%	56%	3%	1%	40%	55%	3%	1%
		$\mathcal{L}_O(0,2)$	40%	55%	4%	1%	40%	55%	4%	1%
		$\mathcal{L}_O(1,2)$	39%	55%	4%	1%	40%	54%	4%	1%

Table 47 – Details of success in detecting linkage for different selection methods for population sizes 100 and 500 over all 3-bit classes – first published in [3].

6.11 Summary

In this chapter we have examined the space of 3-bit classes. We have discovered conditionally-necessary interactions in the Walsh families. We have shown an equivalent expected probability of reaching an optimum over the equal-cost B and C network sets, suggesting that this is a useful classification. We have shown by counter-example that the delta linkage equivalence does not extend as described to 3-bits. In the following chapter, we discuss higher-dimensional function spaces.

7 Higher-Dimensional Pseudo-Boolean Functions

In chapters 5 and 6 we presented an exhaustive computation on all 2-bit and 3-bit classes. As explained in section 4.5 this is computationally intractable for higher dimensional function spaces. In this section we discuss extending and applying what we learn from these small cases. We discuss the construction of larger functions with overlapping concatenation of 3-bit classes, and the applicability of precedence networks to higher dimensions, then we present an algorithm for estimating the Walsh structure of larger functions, and lastly present a method of constructing easy or hard instances of a problem we construct to be solved by a hill-climber algorithm.

7.1 Combining 3-Bit Classes

In this section we discuss the applicability of using 3-bit functions to build larger functions with known minimal Walsh structure.

7.1.1 Concatenation of Non-Overlapping Functions

The simplest way to construct larger functions from 2-bit functions is to use concatenation. This is the same technique used to construct deceptive functions from trap functions, e.g. the TRAP_k^ℓ is an ℓ -bit function constructed by concatenating ℓ/k instances of the TRAP_k .

In the same way we can concatenate $\ell/3$ non-overlapping functions and apply a random permutation of the variables so that the linkage groups are not comprised of only adjacent alleles.

From this we can construct any order-3 additively separable function and know the minimal Walsh structure, since all such functions have structures composed of these classes. If we include functions which have some (or all) linkage groups of sizes < 2 , then we would have to include 1-bit and 2-bit function classes for concatenation too if the length of the problem to be constructed is not a multiple of 3.

7.1.2 Stitching of Overlapping Functions

It is an open question how we usefully decompose non-additively-separable functions into sums of smaller functions, which has been the subject of research [108]. Here we wish to construct functions from a sum of sub-functions which are not an additive separation (since variables will necessarily appear in more than one sub-function), we observe the risk constructing functions where the interaction of overlapping Walsh coefficients creates functions for which we do not know the minimal Walsh structure.

In this section we show an example of how stitching together carefully-chosen 3-bit bivariate functions can work to create two rank-equivalent 5-bit functions with different linkage partition properties, based on the alternate minimal Walsh structures of the sub-functions. The structure of the four functions are shown in Figure 28.

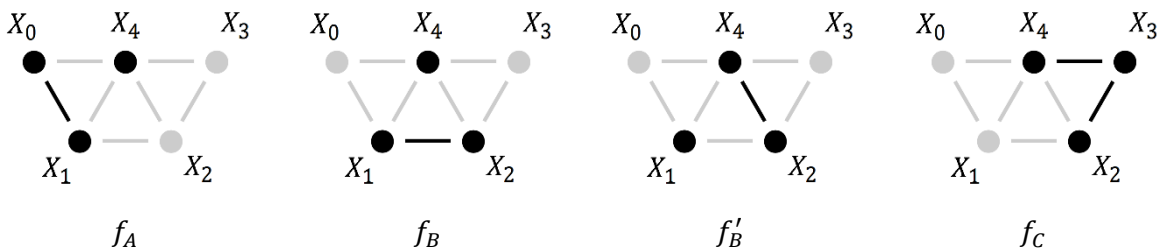


Figure 28 – Walsh structures of example 3-bit sub-functions for stitching. f_B and f'_B are rank-equivalent.

When three functions are added together as $f_A + f_B + f_C$ or $f_A + f'_B + f_C$, the middle sub-function f_B acts as a hinge which changes the structure between a bivariate chain structure f or an additively separable function f' .

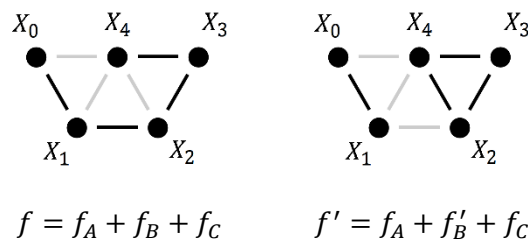


Figure 29 – Stitched 5-bit functions. $f_A + f_B + f_C$ and $f_A + f'_B + f_C$ are rank-equivalent.

The function f' can be separated into two new sub-functions f_D and f_E , whereas function f is not separable.

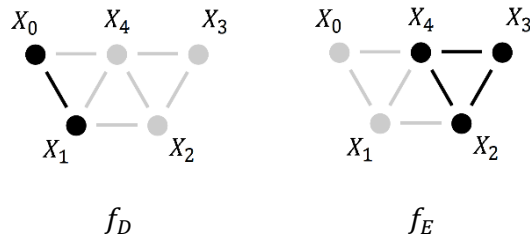


Figure 30 – New additive separation f_D, f_E of 5-bit function $f' = f_D + f_E$.

To do this construction we need to choose function values. The chosen 3-bit functions are defined by their Walsh coefficients, listed in Table 48.

	f_A	f_B	f'_B	f_C
$\alpha_{\{0\}}$	-1			
$\alpha_{\{1\}}$	3	30	40	
$\alpha_{\{2\}}$		20	70	8
$\alpha_{\{3\}}$				2
$\alpha_{\{4\}}$	2	20	30	6
$\alpha_{\{0,1\}}$	-5			
$\alpha_{\{1,2\}}$		10		
$\alpha_{\{2,3\}}$				-2
$\alpha_{\{3,4\}}$				-2
$\alpha_{\{2,4\}}$			-10	

Table 48 – Walsh coefficients for example 3-bit sub-functions for stitching. f_B and f'_B are rank-equivalent.

Functions f_A and f_C are instances of the R_{f_A} and R_{f_C} with minimal Walsh structure. f_B and f'_B are rank-equivalent classes with alternative possible minimal Walsh structures. Either $\alpha_{\{1,2\}}$ or $\alpha_{\{2,3\}}$ are necessary coefficients. This can be shown by performing the Walsh-Hadamard transform of the delta expansion of each function individual as in section 6.2 when the function variables are permuted as $\{X_0, X_1, X_2\}$.

	f_A		f_B	f'_B		f_C
f_{10**0}	11	f_{*00*0}	110	130	f_{**010}	16
f_{10**1}	7	f_{*00*1}	70	90	f_{**000}	12
f_{01**0}	3	f_{*10*0}	30	50	f_{**001}	4
f_{00**0}	-1	f_{*01*0}	-10	10	f_{**100}	0
f_{01**1}	-1	f_{*10*1}	-10	10	f_{**011}	0
f_{11**0}	-5	f_{*11*0}	-50	-70	f_{**110}	-4
f_{00**1}	-5	f_{*01*1}	-50	-70	f_{**101}	-8
f_{11**1}	-9	f_{*11*1}	-90	-150	f_{**111}	-20

Table 49 – Function values for example 3-bit sub-functions for stitching. f_B and f'_B are rank-equivalent.

We choose large values for functions f_B and f'_B such that the Walsh coefficients are all larger in magnitude than the coefficients in f_A and f_C . This magnification prevents the structures interacting in a way which would cancel out the desired construction.

7.2 Precedence Networks

The precedence network uses directed ordinal linkage to extend the idea of a linkage partition. A precedence network is a directed acyclic graph. Each vertex corresponds to a set of one or more variables which are *interdependent*, and each edge corresponds to a *directed dependence* relation. Here we present an algorithm to construct a precedence network – limited connectivity precedence network algorithm (LCPNA).

A *topological ordering* (or topological sort) is an ordering induced on the vertices of a directed acyclic graph such that for every edge $A \rightarrow B$, the vertex A appears before B in the ordering. For some directed acyclic graphs, there are more than one possible topological ordering. In some cases, only a single topological ordering is possible. If a network has a single topological ordering, we refer to this a *fully-specified network*.

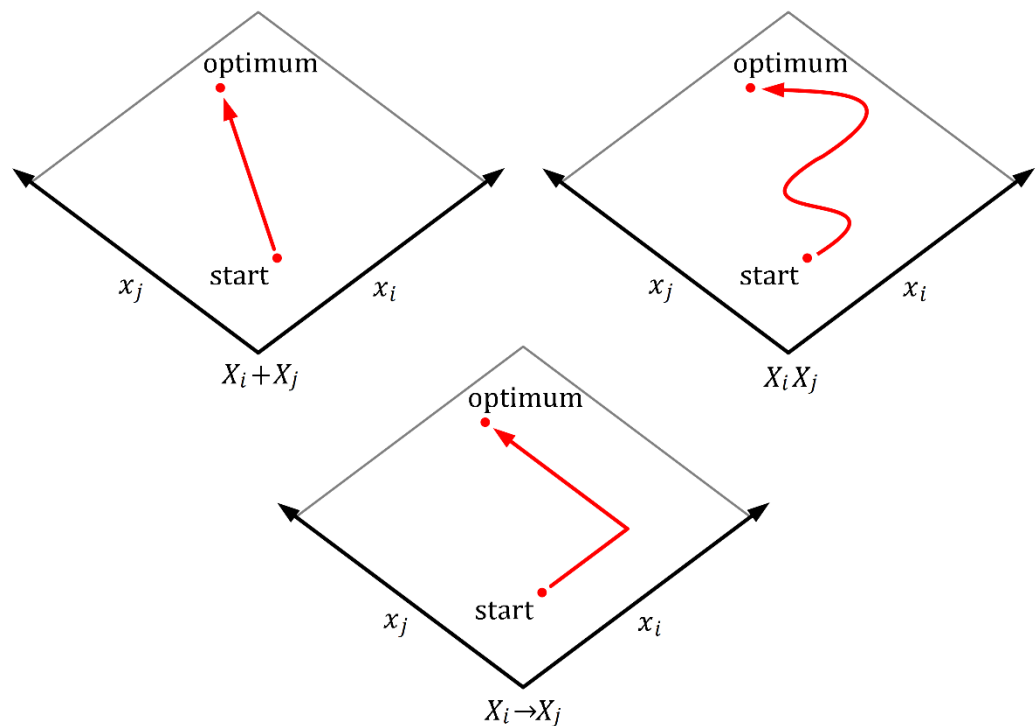


Figure 31 – Conceptual illustration of the exploration path required to locate the global optima for the variables X_i and X_j when the dependence relationship is known. In the case of dependence, the full cross-section $X_i X_j$ may need to be exhaustively explored

7.2.1 Limited Connectivity Precedence Network Algorithm

We described precedence networks in sections 5.5 (on 2-bit functions) and 6.5 (on 3-bit functions). In this section we discuss how we can estimate a precedence network for a higher-dimensional function where the search space is too large to exhaustively evaluate.

Here we propose a precedence network learning algorithm based on the $\ell \log(\ell) 2^k$ perturbation algorithm *ASF*OPTIMISE by Streeter [61]. We call this algorithm limited connectivity precedence network algorithm (LCPNA). The pseudocode is given in Algorithm 3 below.

```

1) define  $G$  as digraph with  $\ell$  unconnected vertices
2) calculate  $asside = n \frac{\ell}{k} 2^k$ 
3) while  $evals + 4 + asside < budget$ 
  a)  $i \leftarrow$  random vertex from  $G$ 
  b)  $s_0 =$  random string of length  $\ell$ 
  c)  $s_1 =$  copy of  $s_0$ 
  d) for each vertex  $j$  in  $G$  from which  $i$  is not reachable
    i)  $s_1[j] =$  random assignment
  e)  $s'_0 =$  copy of  $s_0$ 
  f)  $s'_1 =$  copy of  $s_1$ 
  g)  $s'_1[i] = rand(\{0, 1\}^{|\ell|} - \{s_0[i]\})$ 
  h) if  $sgn(f(s_1) - f(s_0)) \neq sgn(f(s'_1) - f(s'_0))$ 
    i) binary search to find  $\Gamma_j \rightarrow \Gamma_i$  or break when  $evals \geq budget - asside$ 
    ii) add edge  $\Gamma_j \rightarrow \Gamma_i$  to  $G$ 
    iii) if  $G$  contains a cycle
      (1) if number of variables in cycle  $> k$ 
        (a) remove edge  $\Gamma_j \rightarrow \Gamma_i$  from  $G$ 
      (2) else
        (a) contract vertices in cycle into single vertex
4) return  $G$ 

```

Algorithm 3 – Limited connectivity precedence algorithm (LCPNA)

To learn the precedence of variables, not just the linkage partition, we use our definition of directed ordinal linkage given in section 4.2. We generate random strings s_0 and s_1 where the non-influencers of the target linkage group i are varying, and the influencers of i are held fixed, then generate random strings s'_0 and s'_1 which are copies of s_0 and s_1 where one or more of the variables in the target linkage group is different.

If there is an effect of the variables not yet discovered as being influencers of i then this will be detected $\text{sgn}(f(s_1) - f(s_0)) \neq \text{sgn}(f(s'_1) - f(s'_0))$. We then use binary search to find a linkage group in $NI(i)$, which is the set of linkage groups which have not been identified as preceding i and move add an edge from that linkage group to i . If a cycle is detected, all variables from all linkage groups in the cycle are merged into one linkage group.

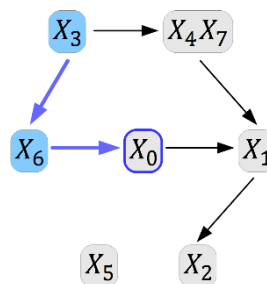


Figure 32 – An example state of precedence network learning. Linkage group $\{X_0\}$ is highlighted. The influencers of $\{X_0\}$ are the 2 linkage groups $I(\{X_0\}) = \{\{X_3\}, \{X_6\}\}$, and the non-influencers of $\{X_0\}$ are the 4 linkage groups $NI(\{X_0\}) = \{\{X_1\}, \{X_2\}, \{X_5\}, \{X_4, X_7\}\}$.

	s_0	s_1	s'_0	s'_1
$\{X_0\}$	$x \leftarrow \text{rand}(\{0, 1\})$	x	$y \leftarrow 1 - x$	y
<p>The current target i has two random values selected (<i>without</i> replacement). One value is used for s_0 and s_1, the other is used for s'_0 and s'_1. If $i = 1$ (as in our example) there are only 2 possible values – both are chosen. If $i > 1$, then 2 random (distinct) values of $\{0, 1\}^{ i }$ are chosen.</p>				
$\{X_3\}$	$c_1 \leftarrow \text{rand}(\{0, 1\})$	c_1	c_1	c_1
$\{X_6\}$	$c_2 \leftarrow \text{rand}(\{0, 1\})$	c_2	c_2	c_2
<p>The influencers are held constant at a random value c.</p>				
$\{X_1\}$	$a_3 \leftarrow \text{rand}(\{0, 1\})$	$b_3 \leftarrow \text{rand}(\{0, 1\})$	a_3	b_3
$\{X_2\}$	$a_4 \leftarrow \text{rand}(\{0, 1\})$	$b_4 \leftarrow \text{rand}(\{0, 1\})$	a_4	b_4
$\{X_5\}$	$a_5 \leftarrow \text{rand}(\{0, 1\})$	$b_5 \leftarrow \text{rand}(\{0, 1\})$	a_5	b_5
$\{X_4, X_7\}$	$a_5 \leftarrow \text{rand}(\{0, 1\}^2)$	$b_5 \leftarrow \text{rand}(\{0, 1\}^2)$	a_6	b_6
<p>The non-influencers have two random values selected (<i>with</i> replacement). One value is used for s_0 and s'_0, the other is used for s_1 and s'_1. For any given non-influencer a may equal b or not. If $a = b$ for <i>all</i> non-influencers, the selection is repeated so at least one varies.</p>				

Table 50 – Setting of strings in example state (Figure 32) in order to test of linkage (to find a linkage group which influences linkage group i).

There are four possible changes which may occur at the current step, depending on which linkage group is found to contain an influencer of X_0 . The four possibilities are illustrated in Figure 33. If no linkage is detected at this step, the graph is the same for the next iteration step.

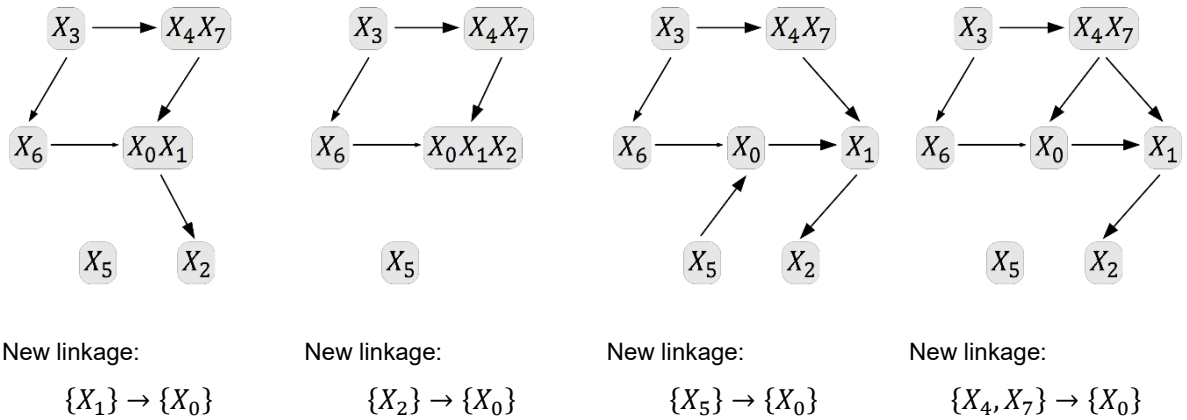


Figure 33 – Starting from the given example state (Figure 32), all possible next states if a variable is found to influence $\{X_0\}$. Note that if $\{X_1\}$ or $\{X_2\}$ is detected as influencing $\{X_0\}$ a cycle is created and the variables in the cycle are contracted into one vertex as a new linkage group. If none of the four variables is detected as an influence in this step, then the next state will be unchanged from the current state.

One change we make compared with ASFOPTIMISE is that we only optimise at the end of the procedure. This does not improve the complexity but reduces the number of evaluations. For this we need to calculate the maximum complexity of the network and set aside function evaluations from a pre-decided budget of evaluations, and use those set-aside evaluations at the end to evaluate the network. The procedure of sampling is described in section 7.2.2.

It is possible that a cycle will be created between n linkage partitions such that the total number of variables exceeds the limit $(|\gamma_0| + |\gamma_1| + \dots + |\gamma_{n-1}|) > k$. This occurs when there is a chain of such n linkage groups $\gamma_0 \rightarrow \gamma_1 \rightarrow \dots \rightarrow \gamma_{n-1}$ and the linkage $\gamma_{n-1} \rightarrow \gamma_0$ is detected, creating the cycle. The network can now not be sampled within the pre-determined budget of evaluations.

7.2.2 Sampling

Random sampling of a precedence network can be performed in one pass in $O(2^k)$ fitness evaluations where k is the maximum number of variables in a single vertex. The pseudocode is given in Algorithm 4 below.

```
1) init best ← null
2) init best_value ← null
3) T ← random topological sort of G
4) repeat n times
5) define A as new empty dictionary
  a) for i ← 0 to |T| − 1
    i) define x as new length ℓ vector
    ii) for j ← 0 to i − 1
      (1) x[j] = A[j]
    iii) for j ← i + 1 to |T| − 1
      (1) x[j] ← random assignment
    iv) A[i] ← exhaustive evaluate to find best assignment for T[i]
  b) if best = null or f(A) > best_value
    i) best ← A
    ii) best_value ← v
6) return best
```

Algorithm 4 – Precedence network sampling

First, a topological sort is chosen, then at each step, the variables in earlier partitions are assigned their best values, the variables in later partitions are assigned randomly chosen values (chosen once per step), and every combination of values for variables in the current partition is tried.

Let A be the variables for which the optimum is known (initialise $A = \emptyset$), each linkage group is sampled separately, in the order in which they appear in the topological sort.

To sample one linkage group γ requires $2^{|\gamma|}$ function evaluations to exhaustively determine the optimal setting for the variables in γ . The variables in A should be assigned to determined their optimum and the other variables ($X - A - \gamma_i$) should be set to one a *random* assignment, fixed for the duration of the $2^{|\gamma|}$ evaluations. Assuming there are *no* interactions

such that any variables in $X - A - \gamma$ affect the optimal setting of γ , this will discover the optimal setting for γ . A should be updated to contain γ .

Sampling can be repeated. If the precedence network represents the exact structure of the function, one sample will determine a global optimum. Repeated sampling n times should be applied when the network is an estimate of the linkage.

The overall number of function evaluations $c(\Gamma, n)$ is given by (125).

$$c(\Gamma, n) = n \sum_{\gamma \in \Gamma} 2^{|\gamma|} \quad (125)$$

Since the structure is unknown before construction, we need to use an upper bound on the complexity for a problem of length ℓ for which we limit linkage groups to a maximum size of k . This upper bound is given by (126). This number of function evaluations should be set aside before structure learning for chosen parameters n and k .

$$c(\Gamma, n) \leq n \frac{\ell}{k} 2^k \quad (126)$$

if $(\forall \gamma \in \Gamma)(|\gamma| \leq k)$

7.3 Subset Walsh Transform

In this section we discuss the result of applying the Walsh-Hadamard transform to selected subsets of variables and how this may direct novel algorithms. The algorithm, results and analysis described in this section were first published in [1].

As discussed earlier, finding the exact Walsh structure of a pseudo-Boolean black-box function requires exhaustive evaluation of the search space. It may be desirable to construct an estimation of the structure.

7.3.1 Description of Algorithm

The subset Walsh transform performs a Walsh-Hadamard transform of a selected subset of the variables. For a chosen strict subset of the variables $S \subset X$, let $k = |S|$, one may evaluate 2^k instances of the variables S for a given fixed setting of the remaining variables R , where $R = X - S$. We call this a sampling. We denote the resulting vector of fitnesses as \mathbf{f}_S . The ordering of this vector is given by inheritance from $\{0, 1\}^\ell$ using the projection onto $\{0, 1\}^k$ of the variables in S .

A set of Walsh coefficients may be obtained by applying the Walsh-Hadamard transform in the usual way, as given by (127). We call this the *subset Walsh transform*.

$$\alpha_S = \frac{1}{2^k} H_k \mathbf{f}_S \quad (127)$$

The obtained set of Walsh coefficients represent those coefficients sufficient to reconstruct the \mathbf{f}_S given the specified setting of the remaining variable R .

The question remains, how close α_S is to the true Walsh coefficients α . If we assume that S represents an additively-separable partition of the variable X under f , then any setting of R will not affect the same subfunction as S does, so the setting of R will only offset the values of \mathbf{f}_S by a constant amount independent of S . Given that any two functions g and h which differ only by a constant amount, the Walsh coefficients of g and h will differ only in the constant term - any two samplings will produce the same Walsh coefficients for S with the exception of the constant term α_\emptyset .

We see that this procedure gives correct coefficients for cliques which are subsets of the selected subset (*completely-contained structure*) and statistical estimate for cliques which overlap, but are not subsets of, the selected subset (*partially-contained structure*).

The matrix B , as given by (128), is a 2^k -by- k matrix with rows as the values of X in the same order as in the Walsh-Hadamard transform. This is used to populate the selected k variables in the selected variables S .

$$B = \left[\begin{array}{cccc} 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ 1 & 0 & 1 & \dots & 1 \\ 0 & 0 & 1 & \dots & 1 \\ & & & \vdots & \\ 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \end{array} \right] 2^k \quad (128)$$

The remaining $(\ell - k)$ bits in the remaining variables $X - S$ are generated randomly once per sample. The process is repeated n times to obtain n samples of the Walsh coefficients. The mean μ and standard deviation σ are calculated for each 2^k coefficients.

The pseudocode for the Subset Walsh Transform is given by Algorithm 5 (p. 149) for parameter n representing the number of samples.

- 1) **define** A as 2^k -by- n matrix
- 2) **define** μ as length 2^k column
- 3) **define** σ as length 2^k column
- 4) **for** $j \leftarrow 0$ **to** $n - 1$
 - a) **define** $r \leftarrow$ row of $(\ell - k)$ random bits
 - b) **define** f as length 2^k column
 - c) **for** $i \leftarrow 0$ **to** $2^k - 1$
 - i) **define** $s \leftarrow$ row i of B
 - ii) **define** x as length ℓ bitstring
 - iii) for indices in k , populate elements of x with elements of s left-to-right
 - iv) for indices not in k , populate elements of x with elements of r left-to-right
 - v) $f[i] \leftarrow \text{evaluate}(x)$
 - d) column $A[j] \leftarrow \frac{1}{2^k} H_k f$
- 5) **for** $i \leftarrow 0$ **to** $2^k - 1$
 - a) $\mu[i] \leftarrow$ mean of row $A[i]$
 - b) $\sigma[i] \leftarrow$ stdev of row $A[i]$
- 6) **return** columns μ and σ

Algorithm 5 – Subset Walsh Transform. First published in [1].

7.3.2 Results

Here we show the result of running the subset Walsh transform on the first six variables of several length-20 functions. These will illustrate the way in which the subset Walsh transform detects complete and partial structure of a given selected subset.

The functions used are the $ONEMAX^{20}$, $CHECK_{1D}^{20}$ and an arbitrary length-20 function (the construction of which is given) with chosen non-zero alphas (with overlapping, but without low-order cliques within high-order cliques). Each was done using a sample size of 5. Finally the result on $TRAP_4^{20}$ is given using a sample size of 20.

For the function $ONEMAX^{20}$, the set of non-zero Walsh coefficients is only the univariate coefficients. All structure is contained within the selected partition, since for any given selected subset, there cannot be a non-zero Walsh coefficient crossing the cut of the partition. Hence, the subset Walsh transform will detect complete structure and no partial structure. The structure is illustrated in Figure 35.

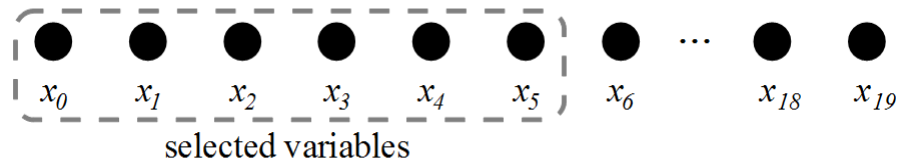


Figure 35 – The Walsh structure of a length-20 univariate function such as $ONEMAX^{20}$. The first six variables are selected. There is no expected non-zero standard deviation from the subset Walsh transform.

We see in Figure 36 that all six univariate coefficients are identified and there are no spurious correlations.

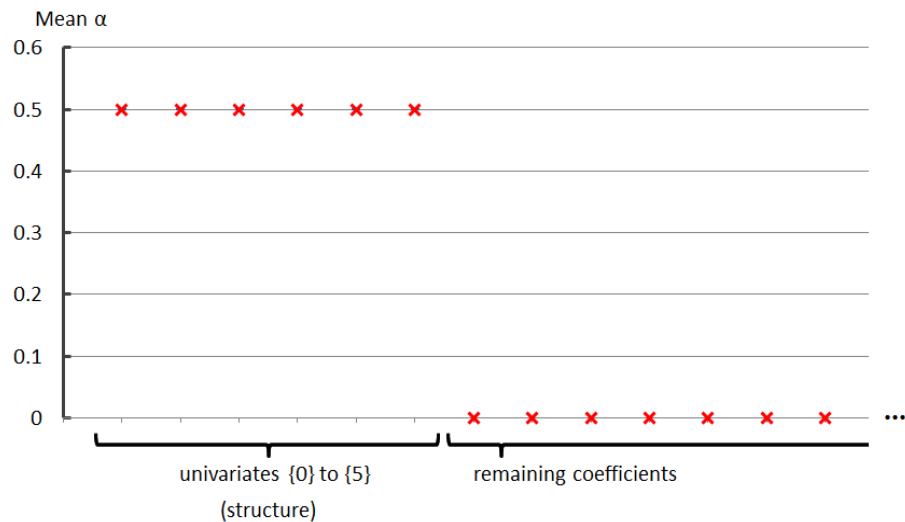


Figure 36 – Result for running subset Walsh transform with sample size 5 on the first six variables on $ONEMAX^{20}$.

For the function $CHECK_{1D}^{20}$, the set of non-zero Walsh coefficients is only the bivariate terms formed from pairs of *adjacent* variables. For any partitioning (non-empty strict subset) of the variables there will be at least one non-zero Walsh coefficient which crosses the cut. For the selected subset shown, the subset Walsh transform will detect complete structure for the first 5 bivariate neighbours and partial structure on variable X_5 . The structure is illustrated in Figure 37.

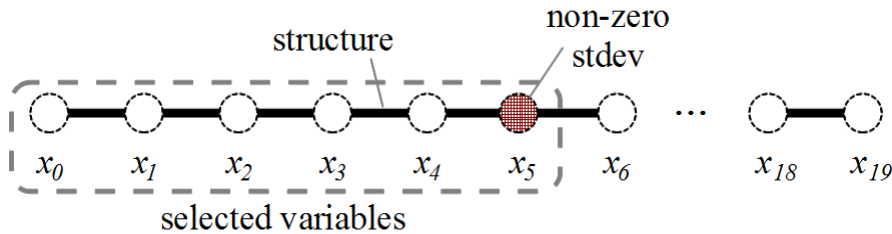


Figure 37 – The Walsh structure of a length-20 bivariate chain function such as $CHECK_{1D}^{20}$. The first six variables are selected. Expected non-zero standard deviation from subset Walsh transform is labelled.

We see that in the result, all bivariate neighbours in the subset are detected correctly, and all other coefficients are correctly identified as zero, except $\alpha_{\{5\}}$, which as expected has a variance. This is because X_5 is part of the clique $\{5,6\}$, of which the variable X_6 is not in the subset, hence this is partial structure. The result is shown in Figure 38.

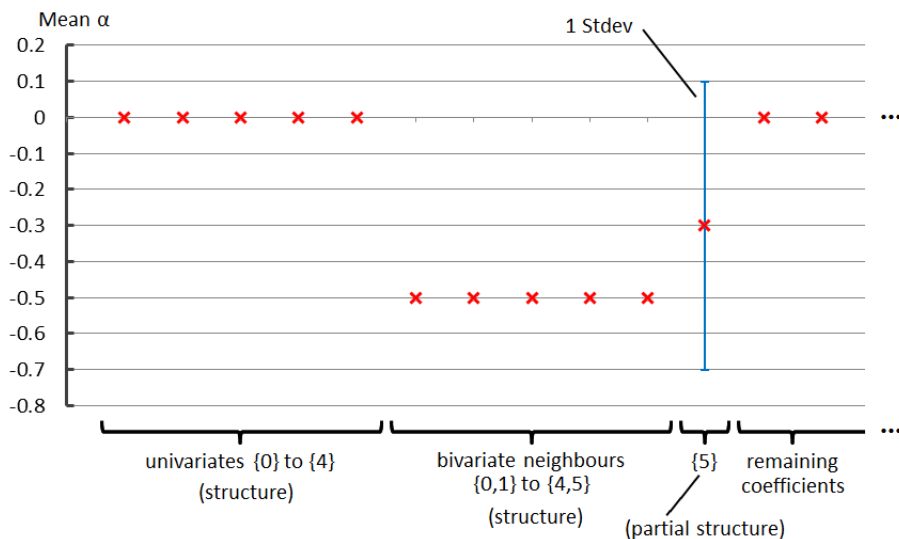


Figure 38 – Result for running subset Walsh transform with sample size 5 on the first six variables on $CHECK_{1D}^{20}$.

For the next example, we constructed a function in which the only non-zero Walsh coefficients are $\alpha_{\{3\}}$, $\alpha_{\{0,4,5\}}$, $\alpha_{\{5,6,7\}}$, $\alpha_{\{0,1,8,9,10\}}$, $\alpha_{\{2,10,11\}}$, $\alpha_{\{12,13,14,15,16\}}$, $\alpha_{\{17,18\}}$, and $\alpha_{\{19\}}$. We choose this structure to illustrate an overlap of structure with the variables $\{X_0, X_1, X_2, X_3, X_4, X_5\}$. All other coefficients including the constant term and their lower-order sub-cliques are zero. The structure is illustrated in Figure 39.

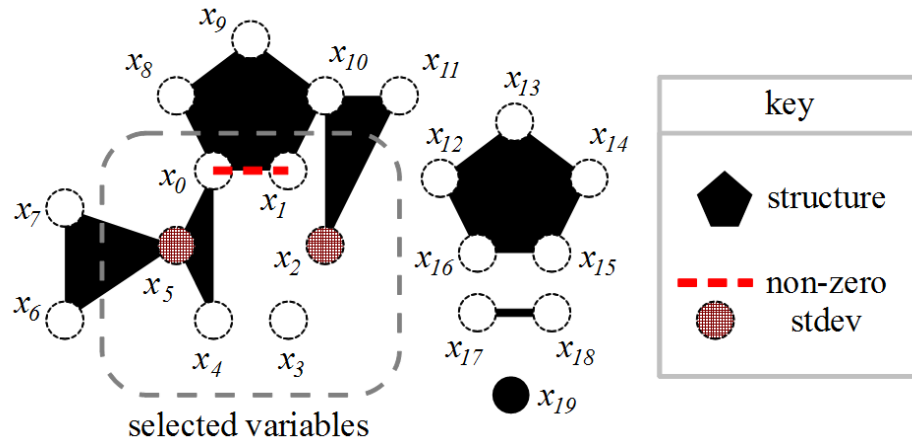


Figure 39 – The Walsh structure of a function in which the only non-zero Walsh coefficients are $\alpha_{\{3\}}$, $\alpha_{\{0,4,5\}}$, $\alpha_{\{5,6,7\}}$, $\alpha_{\{0,1,8,9,10\}}$, $\alpha_{\{2,10,11\}}$, $\alpha_{\{12,13,14,15,16\}}$, $\alpha_{\{17,18\}}$, and $\alpha_{\{19\}}$. The first six variables are selected. Expected non-zero standard deviation from subset Walsh transform is labelled.

Here, the only complete structure in the subset is $\alpha_{\{0,4,5\}}$, which is correctly detected, and the included variables in the three partial structure cliques $\{2\}$, $\{5\}$, and $\{0, 1\}$ have a variance indicating partial structure was detected. All other coefficients are zero. The result is shown in Figure 40.

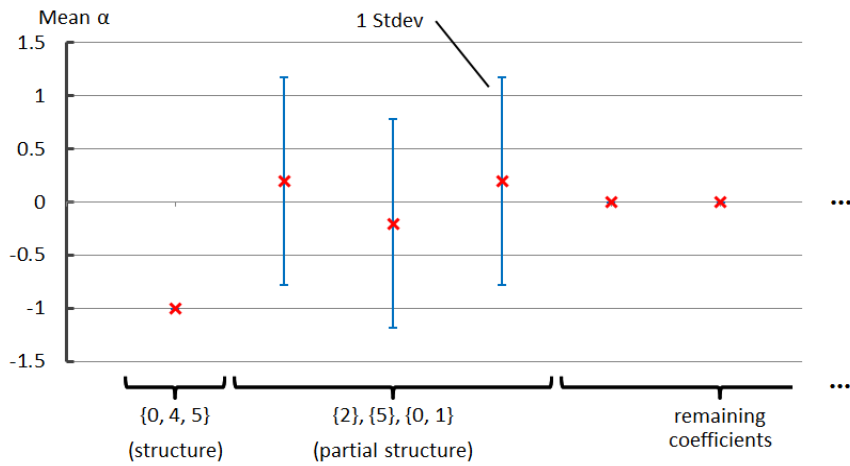


Figure 40 – Result for running subset Walsh transform with sample size 5 on the first six variables on a function in which the only non-zero Walsh coefficients are $\alpha_{\{3\}}$, $\alpha_{\{0,4,5\}}$, $\alpha_{\{5,6,7\}}$, $\alpha_{\{0,1,8,9,10\}}$, $\alpha_{\{2,10,11\}}$, $\alpha_{\{12,13,14,15,16\}}$, $\alpha_{\{17,18\}}$, and $\alpha_{\{19\}}$.

The function TRAP_4^{20} consists of size order-4 maximal cliques concatenated. The selected subset includes the complete structure for one of the traps and half of the structure for the second trap. The structure is shown in Figure 41.

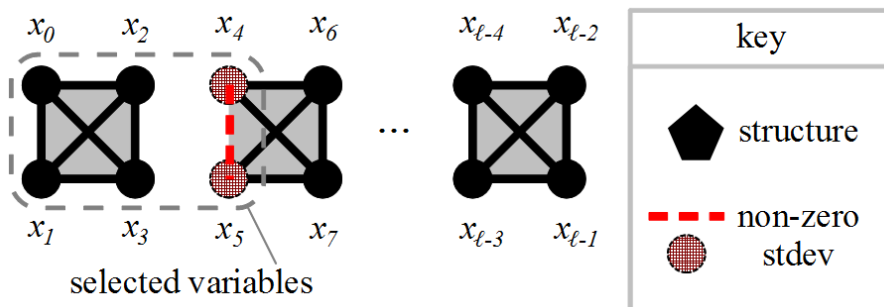


Figure 41 – The Walsh structure of a length-20 function of concatenated order-4 maximal cliques such as TRAP_4^{20} . The first six variables are selected. Expected non-zero standard deviation from subset Walsh transform is labelled.

For the function TRAP_4^{20} , the sample size was increased to 20 due to tendency to miss variance on small sample sizes. Here, the Walsh structure for the complete first trap is correctly identified, and there are variances on the three cliques involved in the second trap, indicating that there are higher-order terms involving variables X_4 and X_5 . Note however, that the mean values of the sample do correctly identify those alphas in this instance, although the mean result for partial structure should not be used where a variance exists. The result is shown in Figure 42.

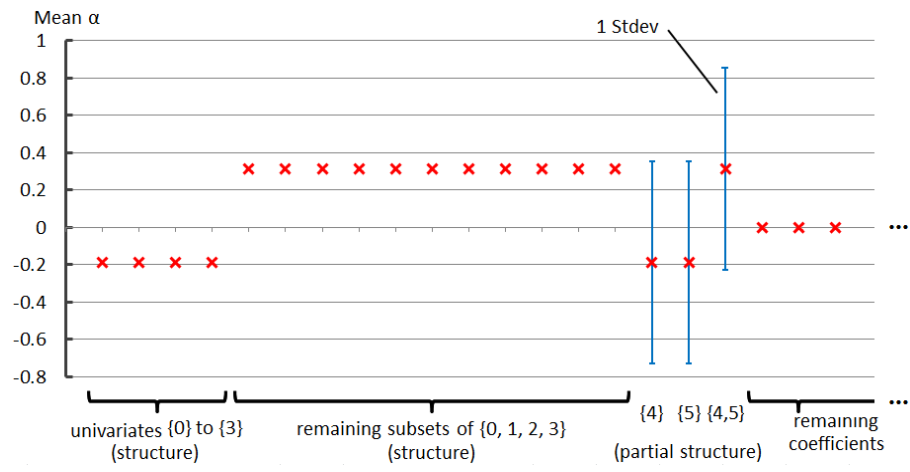


Figure 42 – Result for running subset Walsh transform with sample size 20 on the first six variables on $TRAP_4^{20}$.

7.3.3 Theoretical Analysis

In this section we show that for a given cut of the variables, the subset Walsh transform will return an estimate of the structure within the partition, uninfluenced by structure wholly outside the partition, and with non-zero variance precisely on the parts of the structure crossing the cut. We show this by showing how the Walsh-Hadamard transform affects a partitioning of the function into sub functions.

In this analysis, without loss of generality, we order the variables such that the selected subset, S is the first k variables. However, the same analysis holds for reorderings of the variables. Consider a partitioning Γ of the variables X into two disjoint subsets S of the variables as given by (129), and R as given by (130).

$$S = \{X_0, X_1, \dots, X_{k-1}\} \quad (129)$$

$$R = \{X_k, X_{k+1}, \dots, X_{\ell-1}\} \quad (130)$$

For a sample \mathbf{x} of variables X , let \mathbf{s} represent the sample of variables S and \mathbf{r} represent the sample of variables R . Any binary function considered can be rewritten in terms of f_S , f_P , and f_R , as given by (131).

$$f(\mathbf{x}) = f_S(\mathbf{s}) + f_P(\mathbf{s}, \mathbf{r}) + f_R(\mathbf{r}) \quad (131)$$

where $f_S(\mathbf{s})$ is the function created from all non-zero Walsh coefficients involving only the variables in S , where $f_R(\mathbf{r})$ is the function created from all non-zero Walsh coefficients involving only the variables in R , and where $f_P(\mathbf{s}, \mathbf{r})$ is the function formed from the remaining non-zero Walsh coefficients.

The Walsh coefficients for the function can be calculated from applying the Walsh-Hadamard transform to each part of this expansion in turn as given by (132).

$$\begin{aligned}
\boldsymbol{\alpha} &= \frac{1}{2^\ell} H \mathbf{f} \\
&= \frac{1}{2^\ell} H(\mathbf{f}_S + \mathbf{f}_P + \mathbf{f}_R) \\
&= \frac{1}{2^\ell} H \mathbf{f}_S + \frac{1}{2^\ell} H \mathbf{f}_P + \frac{1}{2^\ell} H \mathbf{f}_R \\
&= \boldsymbol{\alpha}_S + \boldsymbol{\alpha}_P + \boldsymbol{\alpha}_R
\end{aligned} \tag{ 132 }$$

First we consider the last term, $\boldsymbol{\alpha}_R$. Within one sampling, only the k variables in S vary, and the value of f_R is only affected by the $\ell - k$ variables in R . Hence, within one sampling, the value of $f_R(\mathbf{r})$ is fixed at some arbitrary constant c , therefore we can rewrite the function as given by (133).

$$\begin{aligned}
f_R(\mathbf{r}) &= c \\
f_S(\mathbf{s}) + f_P(\mathbf{s}, \mathbf{r}) + f_R(\mathbf{r}) &= f_S(\mathbf{s}) + f_P(\mathbf{s}, \mathbf{r}) + c \\
f(\mathbf{x}) &= f_S(\mathbf{s}) + f_P(\mathbf{s}, \mathbf{r}) + c
\end{aligned} \tag{ 133 }$$

Since each row of the Hadamard matrix has equal numbers of +1 and -1 (with the exception of the first row), the estimated coefficients from this term are given by (134).

$$\boldsymbol{\alpha}_R = \frac{1}{2^\ell} H_k \begin{bmatrix} c \\ c \\ \vdots \\ c \end{bmatrix} = \begin{bmatrix} \alpha_\emptyset \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{ 134 }$$

which is non-zero for all coefficients except α_\emptyset – which we do not regard as part of the structure. Hence, the structure estimate is unaffected by f_R .

Next we consider the first term, $\boldsymbol{\alpha}_S$. This is k -dimensional. Since the contribution $\boldsymbol{\alpha}_S$ from the subset (135) does not depend on the random background setting, it is constant across the samples and will reflect the correct coefficients of this part of the structure.

$$\boldsymbol{\alpha}_S = \frac{1}{2^\ell} H_k \mathbf{f}_S \tag{ 135 }$$

However, the coefficients for the partial term (136) may change for each sample taken, therefore there will be a non-zero standard deviation in the result.

$$\alpha_p = \frac{1}{2^l} H_k \mathbf{f}_p \quad (136)$$

Hence, the non-zero standard deviation indicates that the sample includes partial structure which crosses the cut specified by the partition.

Stdev (σ)	Mean (μ)	In Structure?
non-zero	any	partial
0	non-zero	yes
0	0	no

Table 51 – Condition for detection of partial structure in Subset Walsh Transform.

However, if the sample is not adequate, partial structure may not be detected because $\sigma = 0$. This is an instance of a false negative.

7.4 Proximate Optimality on Hill-Climbing Algorithms

We have seen that problem difficulty relates to structure. In the literature there are concepts that try to explain when problems will be amenable to metaheuristics. One such measure described in the literature is Glover's proximate optimality principle [79, pp. 138-141]. We look at a more precise definition of POP related to structure, which we call *structural coherence*. Following from the work in this thesis, we have explored construction of problem instances using measures of structural coherence. Recall that proximate optimality principle assumes that high fitness candidates have similar structures.

We explore creating functions which have coherence from the point of view of a hill-climber algorithm. That is, we wish to create functions which are easy or hard for a hill-climber. The technique and experiments described in section 7.4 were first published in [4]. As second author, my contribution was to run the experiments, produce the diagrams, and contribute to the theory.

7.4.1 Concept

Recall the definition of a metric and the Hamming metric from section 2.1.2, as a metric on bit strings. The idea is to use an undirected graph of randomly-selected seed solutions chosen in the Hamming space, and construct a minimum spanning tree of that graph to create a fitness gradient which is smooth at most points, with few peaks and plateaus (these we call *coherent*), then to construct a maximum spanning tree of that graph to create a fitness gradient which is less smooth, with more peaks and plateaus (these we call *anti-coherent*).

As a conceptual illustration, we show a randomly-chosen set of seed points in a 2-dimensional Euclidean space in Figure 43. The minimum spanning tree (a) and maximum spanning tree (b) are shown. One seed point at the bottom of the figure was chosen as the unique global optimum. Each other seed is assigned a discrete fitness value based on path distance from the global optimum along the tree.

Every other point on the space is assigned an interpolated fitness based on the weighted average distance to the two nearest points. The result in each case is a number of cells, each with a fitness gradient. In the case of the minimum spanning tree, the fitness gradient tends to lead to the global optimum, with the exception of a few small plateaux. In the case of the maximum spanning tree, there are a number of local optima, including one high fitness local optimum at the opposite side of the space to the global optimum.

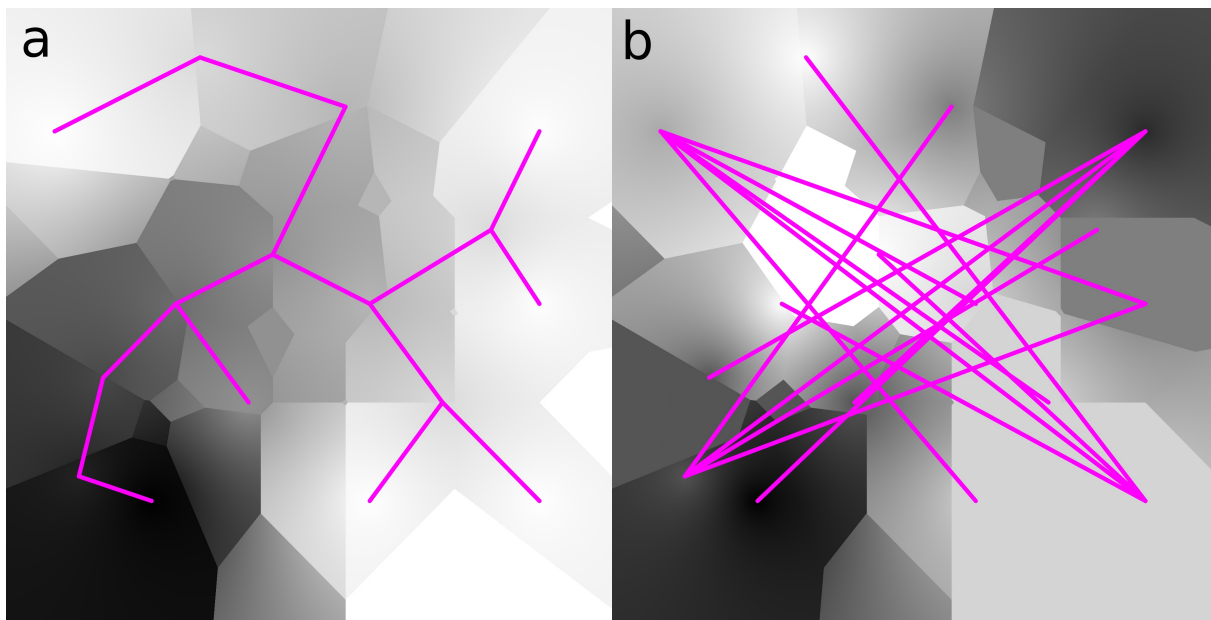


Figure 43 – Minimum spanning tree (a) and maximum spanning tree (b) for a randomly-chosen set of point in 2-dimensional Euclidian space.

7.4.2 Problem Generation and Evaluation

We generate problems in an ℓ -dimensional Hamming space. First we choose s seed solutions for some chosen parameter value $s \in \{2, 3, \dots\}$. The seed solutions are chosen uniformly at random without replacement from $\{0, 1\}^\ell$. The seed solutions are the vertices of a graph G .

Next, we generate a minimum spanning tree T of G (for coherent instances) using Prim's algorithm [110] or maximum spanning tree T of G (for anti-coherent instances) using Prim's algorithm with negative costs. If desired, a coherent instance and an anti-coherent instance may be generated from the same set of seeds.

```
1) define  $f_E$  as new empty dictionary
2) define  $f_H$  as new empty dictionary
3)  $s \leftarrow \emptyset$ 
4) for  $i \leftarrow 0$  to  $s - 1$ 
   a)  $s \leftarrow s \cup \text{random}(\{0, 1\}^\ell - s)$ 
5)  $T_E \leftarrow \text{min\_span\_tree}(s)$ 
6)  $T_H \leftarrow \text{max\_span\_tree}(s)$ 
7)  $s_E^* \leftarrow \text{random}(\text{leaves}(T_E))$ 
8)  $s_H^* \leftarrow \text{random}(\text{leaves}(T_H))$ 
9)  $f^* \leftarrow \max(\{\text{pathlength}(s^*, s_i) : \forall s_i \in s\})$ 
10)  $f_E[s^*] \leftarrow f_E^*$ 
11)  $f_H[s^*] \leftarrow f_H^*$ 
12) for each  $s_i$  in  $s - s_E^*$ 
   a)  $f_E[s_i] \leftarrow f_E^* - \text{pathlength}(s_E^*, s_i)$ 
13) for each  $s_i$  in  $s - s_H^*$ 
   a)  $f_H[s_i] \leftarrow f_H^* - \text{pathlength}(s_H^*, s_i)$ 
14) save  $f_E$  to easy file
15) save  $f_H$  to hard file
```

Algorithm 6 – Coherence problem instance generation

The fitness of the seed points are chosen. First a fitness of MAX is assigned to one randomly chosen vertex of T with only one neighbour (a leaf), where MAX is the maximum number of hops on T from any vertex to that leaf, this leaf will be the instance global optimum.

Next, each other seed is assigned a fitness of $(MAX - h)$ where h is the number of hops to the global optimum. As a result, each seed will be assigned a value in $\{0, 1, \dots, MAX\}$ with at least one seed at each level, and only one seed assigned the value of MAX .

For each point in the search space which is not a seed, we define the fitness of a point x in the as the weighted average of the two closest seed points p and q , given by (137), based on D_r , the hamming metric; ties are broken arbitrarily (based on the order in which seeds were randomly generated). As a result, each value in the space, will be assigned a value in $[0, MAX]$ with only one value assigned the value of MAX .

$$f(x) = f(p) \cdot (1 - t) + f(q) \cdot t$$

$$\text{where } t = \frac{D_r(x, p)}{D_r(x, p) + D_r(x, q)} \quad (137)$$

Algorithm 7 below shows the process of evaluating a candidate x .

```

1)  $f \leftarrow$  load from file
2) if  $x \in \text{keys}(f)$ 
   a) return  $f[x]$ 
3) else
   a)  $p, q \leftarrow$  two closest points in  $f$  to  $x$ 
   b)  $t \leftarrow D_r(x, p) / (D_r(x, p) + D_r(x, q))$ 
   c) return  $f[p] \cdot (1 - t) + f[q] \cdot t$ 

```

Algorithm 7 – Coherence problem function evaluation

7.4.3 Results

10 instances for each problem length 6 to 100 were generated using both max-span and min-span from the same seeds. We ran a multi-restart maximum-ascent hill-climber algorithm on each instance 100 times and plotted the results in Figure 44. The runtime of the min-span (coherent) instances is quadratic. The runtime of max-span instances (anti-coherent) was quadratic in the best case and exponential in the worst case, depending on the instance.

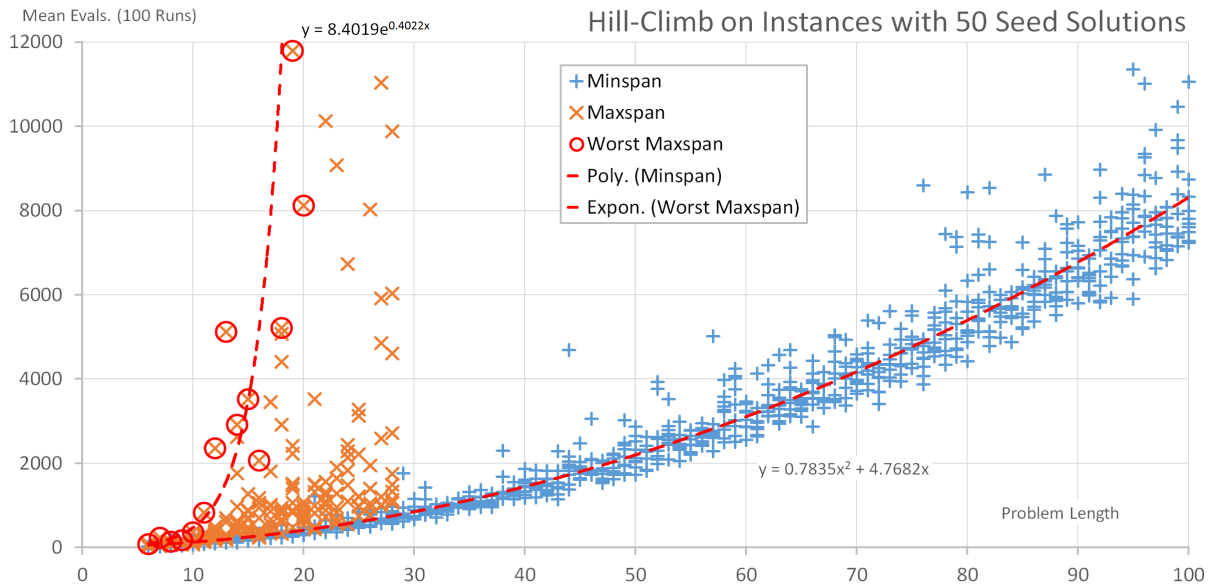


Figure 44 – Result of hill-climb on instances with 50 seed solutions.

7.4.4 Conclusions

The procedure described produces single-global-optima binary functions which can be easy or hard to optimise with a hill-climber. These are *coherent* and *anti-coherent* instances from the point of view of a hill-climber.

It is worth noting that there are cases in which even the minimum span procedure can generate instances with a relatively isolated basin of attraction to the global optimum. As an illustration, we show two seed points to the left of the graph on a one-dimensional Euclidian search space and one seed point to the right in Figure 45.

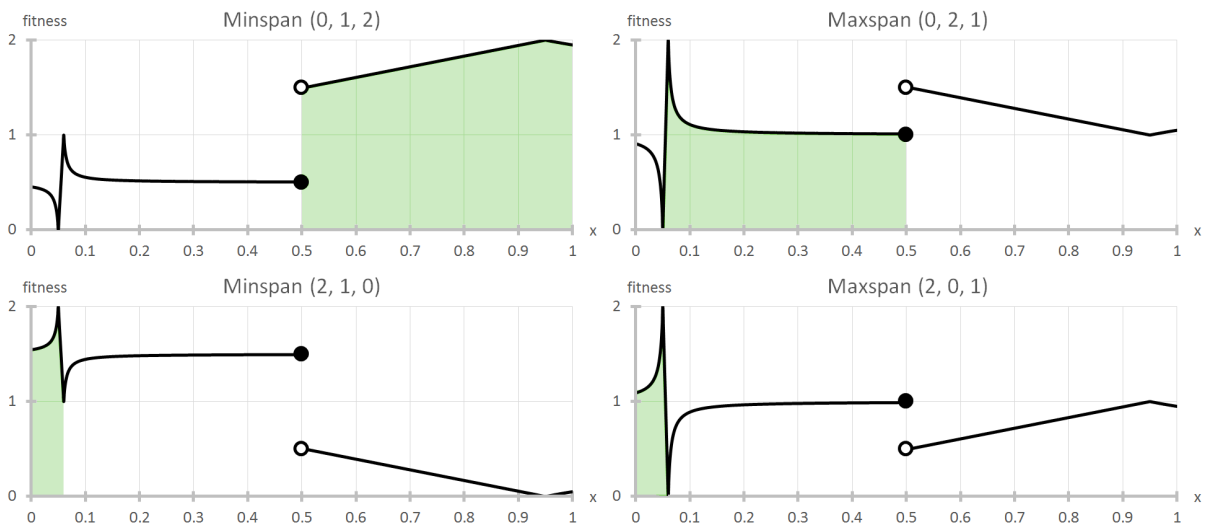


Figure 45 – All possible assignments of values to minimum spanning tree (left charts) and maximum spanning tree (right charts) for three selected seeds on one-dimensional Euclidean space. Basin of attraction to global optimum highlighted.

Observe that in the illustration, the fitness gradient is linear when interpolated between the two closest points ($p < x < q$), but falls off sharply as $\frac{1}{D}$ when extrapolating past the two closest points ($p < q < x$). If $f(p) > f(q)$ then this creates a local optimum with a wide basin of attraction, far from any seed solution.

The example given is an illustration in a Euclidian search space, and chosen pathologically to illustrate this behaviour. However, we see that difficulty of functions generated may be affected by other factors than the choice of minimum or maximum spanning tree. This may help explain the large variance in difficulty of max-span instances.

7.5 Summary

In this chapter we have discussed combining 3-bit classes into larger problem instances, and algorithms for learning precedence network structure and Walsh structure. We have outlined a method of generating easy and hard problems for a hill-climber based on proximate optimality structure. Next, we conclude the thesis and give suggestions for future work.

8 Conclusions and Further Work

8.1 Conclusions

In this thesis we addressed the following four research questions:

1. What is the relationship between problem structure and problem difficulty?

In chapters 5 and 6 we introduced the notion of precedence profiles, and discussed the relationship between precedence profiles and problem difficulty. We also discussed how the computational effort affects the probability of finding an optimum, across the set of function classes. In chapter 7 we constructed easy and hard functions based on ideas of structural coherence for hill-climbing.

2. How can we use structure to usefully classify problems?

In chapter 4 we defined a classification of pseudo-Boolean functions based on function classes invariant under monotonic operators. The performance of any evolutionary algorithm using only monotonic operators on any two functions of the same class is identical and reasoning about the performance of such an algorithm applies to the whole class. We define the notion of directed ordinal linkage as an extension of the existing definitions of linkage.

3. Can we use structure to bound the number of algorithmic steps?

In chapters 5 and 6 we analysed the linkage and directed ordinal linkage of the complete set of 2-bit and 3-bit pseudo-Boolean functions classes, and derived a notion of minimal Walsh structure. We derived a notion of *conditionally-necessary interactions* (those interactions which may be necessary or unnecessary depending on other interactions). We also defined precedence networks – an ordered evaluation of linkage partitions. We also discussed how population size, and hence computational cost, is affected by choice of selection operators.

4. Can structure analysis motivate the development of novel algorithms?

In chapter 7 we discussed a means of algorithmically constructing a precedence network describing the ordinal linkage structure of a problem, and a linkage-learning algorithm which can be used to optimise a function. We also described an algorithm for learning the Walsh structure of a function by sampling subsets of the variables. We discussed the construction of problem instances from our analysis of 3-bit functions.

8.2 Further Work

8.2.1 Refinement of 3-Bit Structural Coherence

In section 6.10 we discuss the structural coherence of 3-bit function classes and minimum population sizes required for detection of linkage.

Here we only consider the pairwise linkage. From the point of view of our definitions, as described in chapter 6 there are higher-order interactions than pairwise. Additionally, other algorithms in the literature uses higher-order interactions. We can extend this work either by using the perturbation-based definition of linkage, taking two variables to be linkage if they are in the same connected component of the linkage graph, taking into account the higher-order linkage.

We also note that our current choice of recognising linkage – by choosing the structure with the lowest-ID from a class' Walsh family – puts a bias which introduced an asymmetry on the bivariate terms. This problem arises because there exists 3-bit function classes where there is no clear minimal structure (as noted in section 6.4). For the case of the class [0 1 2 4 2 4 4 7] (see Figure 20, p. 116), we select 1F as the minimal structure and rate the algorithm's ability to detect linkages $\alpha_{\{0,1\}}$ and $\alpha_{\{0,2\}}$ while counting $\alpha_{\{0,2\}}$ as a false negative. From the Walsh family we see that the structure 2B (which contains $\alpha_{\{0,2\}}$ as the only bivariate term) is an equally-valid structure for this class.

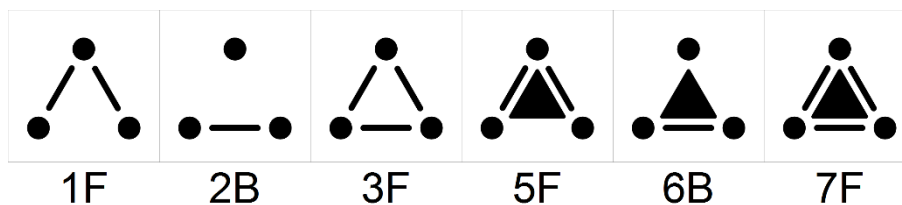


Figure 20 (p. 116) – All possible Walsh structures for [0 1 2 4 2 4 4 7].

One possible way to proceed is to introduce a notion of *non-dominated structure*, which would define as the set of structures (in this case {1F, 2B}) for which no strict subset of the structure elements exist in the family. A measure of a monotonic EDA's structure learning should acknowledge the existence of these two equally-value structures. If the EDA learns the structure 3F, then it has learned at least one unnecessary interaction, however, it is not clear which bivariate term(s) should be regarded as unnecessary in this case.

8.2.2 Further Development of Precedence Network Learning

The algorithm in section 7.2.1 as described does not easily learn the structure of the LEADING function, since the interactions involving variables $X_i \rightarrow X_j$ for large j are difficult to detect since it requires correct setting of all variables $X_0 \dots X_{j-1}$. A modification could be made for this case which stores information about known good settings of the variables influencing the current target allowing the algorithm to be more likely to try those values.

Methods could be investigated to determine the best way to cut a graph when the addition of new edges causes creation of a vertex which is too large to evaluate. For example, we could assign weights to vertices based on the result of linkage detection (e.g. using some non-monotonic information about the magnitude of the fitness differences). Minimum cut or similar procedure could be used to discard existing linkages of a lower strength.

We could randomly break linkages with some probability based on a cooling schedule such that all linkages would have an equal chance to be broken at the start of the network learning, but stronger linkages would have a lower chance to be broken as time passes.

One of the difficulties that occurs when implementing these procedures is that it is difficult to decide how to break apart structure. If variables are first contracted into linkage group, then one or more variables are found to be influencers of the group, those dependencies are shared by the members. This makes it difficult to well-define a procedure which would allow the interdependency between the variables of the group later. Figure 46 shows the structure $(X_0 + X_2) \rightarrow X_1$, the next step linkage $X_1 \rightarrow X_2$ is added, and in the last step, the linkage is removed again. From the previous step, it is not clear which variable is dependant on X_0 unless a memory of previous steps is kept. For more complex, nested structures, depending on the order of adding and removing linkage, the correct linkage may not be defined.

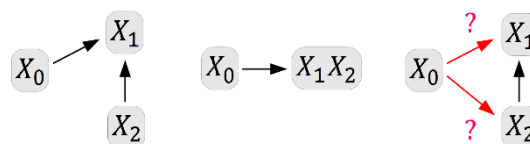


Figure 46 – Merge then split of variables X_1 and X_2 .

A work around could involve a procedure of disallowing the creation of linkage groups of size > 1 initially, then increasing the limit gradually as the temperature changes, allowing larger and larger linkage groups to be formed permanently.

8.2.3 Construction of Benchmark Functions

In section 7.1.2 we discuss the construction of functions by stitching together overlapping sub-functions. Further development of this procedure could construct problem instances with known Walsh structure. These could be used to evaluate the success of EDAs in structure learning.

Additionally, it may be possible to consider construction of new benchmarks with partially controlled complexity in terms of number of possible structures. Recall from Table 38 (p. 111) that the number of fitness levels is positively correlated with the size of the Walsh family, and hence generally correlated with the complexity of arrangement of optional or conditional Walsh coefficients.

In section 6.7 we list all possible precedence networks on 3-bit functions. It is clear that for higher-dimensional function spaces, there continues to be a complex set of possible precedence networks, and function classes which are most efficiently solved by a given precedence network.

We note that the LEADING benchmark function contains dependence from each pair of variables $(\forall i < j)(X_i \rightarrow X_j)$, however, the rest of our set of benchmark functions contains only independence or interdependence between variables. A further investigation of existing may reveal other existing benchmark functions or real-world problems with directed ordinal linkage. However, the existing literature does not address this concept in the construction of benchmark functions, although they form a large proportion of the space of function classes.

It may be worth constructing benchmark functions with one-way dependencies between variables since an algorithm which would be able to detect such linkage and construct a precedence network may be able to optimise such a function more efficiently than one which constructs an undirected linkage partition. With additional benchmark functions we could see whether modifications to the precedence network learning algorithm are a generally good modification in principal.

8.2.4 Subset Walsh Transform Sweep

The subset Walsh transform could be used repeatedly to discover the structure of a complete function similar to the procedure by Heckendorn and Wright [62] to learn the values of the Walsh coefficients in a function. This method could be used to learn Walsh coefficients for analysing or optimising a function.

Such an approach should be possible by sweeping over the problem. For an arbitrary permutation of the variables, we would start by running $\text{SWT}(\{X_0\})$ and if the procedure finds non-zero standard deviation indicating partial structure, add X_0 to a graph of partial structure. Then we would run $\text{SWT}(\{X_0, X_1\})$ to see if the partial structure involves X_1 , we may find this is complete structure and add it to the final output, or find the X_0 and X_1 are connected but have partial structure with another variable (in which case we try $\text{SWT}(\{X_0, X_1, X_2\})$), or are disconnected from one another but both have partial structure with another variable (in which case we try $\text{SWT}(\{X_0, X_2\})$ and $\text{SWT}(\{X_1, X_2\})$). This algorithm is outlined in Algorithm 8 below.

```
1)  $partial \leftarrow \emptyset$ 
2)  $complete \leftarrow \emptyset$ 
3) for  $i \leftarrow 0$  to  $\ell - 1$ 
   a) if  $partial = \emptyset$ 
      i) run SWT on  $X_i$ 
      ii) add complete structure to  $complete$ 
      iii) add partial structure to  $partial$ 
   b) else
      i) For  $d \leftarrow$  set of variables in each connected component of  $partial$ 
      ii)  $rv \leftarrow \text{SWT}(d \cup \{X_i\})$ 
      iii) for each complete structure  $\gamma$  in  $rv$ 
           (1)  $complete \leftarrow complete \cup \{\gamma\}$ 
           (2)  $partial \leftarrow partial - \{\gamma\}$ 
      iv) for each partial structure  $\gamma$  in  $rv$ 
           (1)  $partial \leftarrow partial \cup \{\gamma\}$ 
   c) return  $complete$ 
```

Algorithm 8 – Subset Walsh transform sweep

Each call to SWT would require a maximum of 2^k function evaluations where k is the size of the largest linkage group in the function, making the procedure tractable for small k , however, further study is required to determine the probability of failure for a given sample size s across a large number of calls to the SWT procedure, and hence the sample size required is a function of ℓ to determine the resulting runtime.

8.2.5 Necessary Structure for Optimisation

We have considered structure as necessary in the case that there exists no instance of the function's class which has the structure omitted. There are other ways in which we could consider whether structure is necessary.

Recall that the space of pseudo-Boolean functions can be represented using a Walsh function basis. In this sense, function classes are a partitioning of this space. These partitions can also be regarded as adjacent when there are members of two classes separated by any desired small $\epsilon > 0$. Also, difference classes have different degrees of freedom (defined by the number of deltas). The more degrees of freedom, the higher-dimensional subspace the partition represents. A class can be a face of the subspace of another.

For instance, we observe that near functions such as TRAP_2^2 function (of the class $[3 \ 0 \ 0 \ 2]$), there are other functions which contain unnecessary structure. Adding a small amount of noise to TRAP_2^2 would produce an instance of the injective class $\mathbf{C}_A = [3 \ 0 \ 1 \ 2]$ or $\mathbf{C}_B = [3 \ 1 \ 0 \ 2]$. Here, TRAP_2^2 is a lower-dimensional face which sits between \mathbf{C}_A and \mathbf{C}_B .

If we regard \mathbf{C}_A and \mathbf{C}_B as easier to optimise than TRAP_2^2 , we see that the effort to carefully model this lower-dimensional sub-face is actually making the function more difficult to model. This additional structure solely maintains the ranks of sub-optimal solutions (specifically the middle ranks) and is thus not necessary for locating this function's global optimum.

This topological view of the space of function classes with injective classes as ℓ -dimensional subspaces, and non-injective classes as sub-faces, indicates that further analysis of the space of functions in this topological sense could further inform the development of novel algorithms.

8.2.6 Non Pseudo-Boolean Functions

Much of the analysis in this thesis extends to higher-order alphabets, since functions on higher-order alphabets will still be grouped into linkage partitions and have directed ordinal linkage. However, the analysis from Walsh coefficients does not extend so easily, since the Walsh decomposition relates specifically to pseudo-Boolean functions. Natural extension would use a set of functions, such as general Fourier functions, to provide a basis from the vector space of functions in lieu of Walsh functions.

As these larger function spaces are a superset of the function space we have already explored, they would necessarily contain the observed phenomena such as conditionally-necessary interactions would be seen. There may be other functions which arise which require modification to our description of necessary/unnecessary/conditionally-necessary interactions which would inform the development of novel algorithms.

An obvious issue is that much of the analysis done in this thesis was on an exhaustive set of function classes. This becomes computationally intractable for larger alphabets, and so other methods of analysis would be required.

Appendix A – List of 3-Bit Classes (Python 3)

```
def rank(f):
    r = [0, 0, 0, 0, 0, 0, 0, 0]
    for x in range(8):
        for y in range(8):
            if y != x:
                if f[y] > f[x]:
                    r[x] = r[x] + 1
    return tuple(r)

def output(r):
    print(str(r).replace(", ", "").replace("(", "").replace(")", ""))

if __name__ == "__main__":
    seen = set()
    for a in range(8):
        for b in range(8):
            for c in range(8):
                for d in range(8):
                    for e in range(8):
                        for f in range(8):
                            for g in range(8):
                                for h in range(8):
                                    r = rank([h, g, f, e, d, c, b, a])
                                    if not r in seen:
                                        seen.add(r)
                                        output(r)

    print("END")
```


Appendix B – Calculating 3-Bit Families (ANSI C)

```
#include <stdio.h>

/* Returns the structure of the given specific function by applying
   the Fast Walsh-Hadamard Transform (FWHT) and converting the
   resulting non-zero structure to a numerical code. */
int find_structure(int* f) {
    int rv = 0;
    int ff[16];
    ff[0] = f[0] + f[4];
    ff[1] = f[1] + f[5];
    ff[2] = f[2] + f[6];
    ff[3] = f[3] + f[7];
    ff[4] = f[0] - f[4];
    ff[5] = f[1] - f[5];
    ff[6] = f[2] - f[6];
    ff[7] = f[3] - f[7];
    ff[8] = ff[0] + ff[2];
    ff[9] = ff[1] + ff[3];
    ff[10] = ff[0] - ff[2];
    ff[11] = ff[1] - ff[3];
    ff[12] = ff[4] + ff[6];
    ff[13] = ff[5] + ff[7];
    ff[14] = ff[4] - ff[6];
    ff[15] = ff[5] - ff[7];
    if (ff[8] - ff[9]) rv += 1;
    if (ff[10] + ff[11]) rv += 2;
    if (ff[10] - ff[11]) rv += 4;
    if (ff[12] + ff[13]) rv += 8;
    if (ff[12] - ff[13]) rv += 16;
    if (ff[14] + ff[15]) rv += 32;
    if (ff[14] - ff[15]) rv += 64;
    return rv;
}

/* Returns the number of distinct values for a specified class. */
int calc_num_ranks(int* clazz) {
    int rv = 0;
    int i, j, is_new;
    for (i = 0; i < 8; i++) {
        is_new = 1;
        for (j = 0; j < i-1; j++) {
            if (clazz[i] == clazz[j]) {
                is_new = 0;
            }
        }
        if (is_new) {
            rv++;
        }
    }
    return rv;
}

/* Finds the class for a specified function. */
void function_to_class(int* function, int* clazz) {
    int i, j, num_lower;
    for (i = 0; i < 8; i++) {
```

```

    num_lower = 0;
    for (j = 0; j < 8; j++) {
        if (function[j] < function[i]) {
            num_lower++;
        }
    }
    clazz[i] = num_lower;
}
}

/* Converts a clazz in-place into tokens for replacement. */
void tokenise(int* clazz) {
    int used;
    int rank, i;
    int token = -1;
    for (rank = 0; rank < 8; rank++) {
        used = 0;
        for (i = 0; i < 8; i++) {
            if (clazz[i] == rank) {
                clazz[i] = token;
                used = 1;
            }
        }
        if (used) {
            token--;
        }
    }
}

/* Creates an instance specified tokeneized class as fitnesses
with the given values. */
void detokenise(int* t_clazz, int* fitnesses, int* vals) {
    int i;
    for (i = 0; i < 8; i++) {
        fitnesses[i] = vals[-t_clazz[i] - 1];
    }
}

/* Computes the Walsh family for the specified tokenised
class with 3 distinct ranks. */
void find_family_3(int* t_clazz, int* family) {
    int v[3], f[8];
    for (v[0] = 0; v[0] <= 2; v[0]++) {
        for (v[1] = v[0] + 1; v[1] <= 3; v[1]++) {
            for (v[2] = v[1] + 1; v[2] <= 4; v[2]++) {
                detokenise(t_clazz, f, v);
                family[find_structure(f)] = 1;
            }
        }
    }
}

/* Computes the Walsh family for the specified tokenised
class with 4 distinct ranks. */
void find_family_4(int* t_clazz, int* family) {
    int v[4], f[8];
    for (v[0] = 0; v[0] <= 4; v[0]++) {
        for (v[1] = v[0] + 1; v[1] <= 5; v[1]++) {
            for (v[2] = v[1] + 1; v[2] <= 6; v[2]++) {
                for (v[3] = v[2] + 1; v[3] <= 7; v[3]++) {
                    detokenise(t_clazz, f, v);
                    family[find_structure(f)] = 1;
                }
            }
        }
    }
}

/* Computes the Walsh family for the specified tokenised

```

```

class with 5 distinct ranks. */
void find_family_5(int* t_clazz, int* family) {
    int v[5], f[8];
    for (v[0] = 0; v[0] <= 6; v[0]++) {
        for (v[1] = v[0] + 1; v[1] <= 7; v[1]++) {
            for (v[2] = v[1] + 1; v[2] <= 8; v[2]++) {
                for (v[3] = v[2] + 1; v[3] <= 9; v[3]++) {
                    for (v[4] = v[3] + 1; v[4] <= 10; v[4]++) {
                        detokenise(t_clazz, f, v);
                        family[find_structure(f)] = 1;
                    }
                }
            }
        }
    }
}

/* Computes the Walsh family for the specified tokenised
class with 6 distinct ranks. */
void find_family_6(int* t_clazz, int* family) {
    int v[6], f[8];
    for (v[0] = 0; v[0] <= 7; v[0]++) {
        for (v[1] = v[0] + 1; v[1] <= 8; v[1]++) {
            for (v[2] = v[1] + 1; v[2] <= 9; v[2]++) {
                for (v[3] = v[2] + 1; v[3] <= 10; v[3]++) {
                    for (v[4] = v[3] + 1; v[4] <= 11; v[4]++) {
                        for (v[5] = v[4] + 1; v[5] <= 12; v[5]++) {
                            detokenise(t_clazz, f, v);
                            family[find_structure(f)] = 1;
                        }
                    }
                }
            }
        }
    }
}

/* Computes the Walsh family for the specified tokenised
class with 7 distinct ranks. */
void find_family_7(int* t_clazz, int* family) {
    int v[7], f[8];
    for (v[0] = 0; v[0] <= 8; v[0]++) {
        for (v[1] = v[0] + 1; v[1] <= 9; v[1]++) {
            for (v[2] = v[1] + 1; v[2] <= 10; v[2]++) {
                for (v[3] = v[2] + 1; v[3] <= 11; v[3]++) {
                    for (v[4] = v[3] + 1; v[4] <= 12; v[4]++) {
                        for (v[5] = v[4] + 1; v[5] <= 13; v[5]++) {
                            for (v[6] = v[5] + 1; v[6] <= 14; v[6]++) {
                                detokenise(t_clazz, f, v);
                                family[find_structure(f)] = 1;
                            }
                        }
                    }
                }
            }
        }
    }
}

/* Computes the Walsh family for the specified tokenised
class with 8 distinct ranks. */
void find_family_8(int* t_clazz, int* family) {
    int v[8], f[8];
    for (v[0] = 0; v[0] <= 8; v[0]++) {
        for (v[1] = v[0] + 1; v[1] <= 9; v[1]++) {
            for (v[2] = v[1] + 1; v[2] <= 10; v[2]++) {
                for (v[3] = v[2] + 1; v[3] <= 11; v[3]++) {
                    for (v[4] = v[3] + 1; v[4] <= 12; v[4]++) {
                        for (v[5] = v[4] + 1; v[5] <= 13; v[5]++) {
                            for (v[6] = v[5] + 1; v[6] <= 14; v[6]++) {
                                for (v[7] = v[6] + 1; v[7] <= 15; v[7]++) {
                                    detokenise(t_clazz, f, v);
                                    family[find_structure(f)] = 1;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

/* Computes the Walsh family for the specified class. */
void find_family(int* clazz, int* family) {
    int i, num_ranks;
    /* Clears the array to store the Walsh family. */
    for (i = 0; i < 128; i++) {

```



```

    family[i] = 0;
}
num_ranks = calc_num_ranks(clazz);
if (num_ranks == 1) {
    /* Family is always {00}. */
    family[0] = 1;
} else if (num_ranks == 2) {
    /* Family contains only one element. */
    family[find_structure(clazz)] = 1;
} else {
    tokenise(clazz);
    switch (num_ranks) {
        case 3:
            find_family_3(clazz, family);
            break;
        case 4:
            find_family_4(clazz, family);
            break;
        case 5:
            find_family_5(clazz, family);
            break;
        case 6:
            find_family_6(clazz, family);
            break;
        case 7:
            find_family_7(clazz, family);
            break;
        case 8:
            find_family_8(clazz, family);
            break;
    }
}
}
}

/* Prints the specified function to STDOUT in decimal. */
void print_function(int* function) {
    int i;
    printf("[");
    for (i = 0; i < 8; i++) {
        if (i != 0) {
            printf(", ");
        }
        printf("%d", function[i]);
    }
    printf("]\t");
}

/* Prints the specified Walsh family to STDOUT in hex. */
void print_family(int* family) {
    int i, first = 1;
    printf("{");
    for (i = 0; i < 128; i++) {
        if (family[i]) {
            if (!first) {
                printf(", ");
            }
            printf("%02X", i);
            first = 0;
        }
    }
    printf("]\n");
}
}

```

```

/* Main method. */
int main(void) {
    int done = 0;
    int f[8];
    int clazz[8];
    int family[128];
    while(!done) {
        /* Reads a function. */
        if (scanf("%d %d %d %d %d %d %d",
                f, f+1, f+2, f+3, f+4, f+5, f+6, f+7)) {
            /* Gets the class. */
            function_to_class(f, clazz);
            /* Outputs the class. */
            print_function(clazz);
            /* Calculates the Walsh family. */
            find_family(clazz, family);
            /* Outputs the Walsh family */
            print_family(family);
        } else {
            done = 1;
        }
    }
    return 0;
}

```


Appendix C – Compiling and Running Sources

The list of 3-bit classes (Appendix A) can be run from the command line using the Python 3 interpreter. The classes are printed to standard output and can be redirected to a file. For example, if the source file is named `classes.py`, output can be sent to a file as such:

```
python3 classes.py >classes.txt
```

The 3-Bit families (Appendix B) can be compiled using GCC or any other standard C compiler. For examples, if the source file is named `families.c`:

```
gcc families.c -o families
```

The 3-bit families can be calculated by reading in lines of text from standard input representing ranks separated by spaces. The program will output the Walsh family to standard output of the input class, then wait for another class. The program will terminate on receiving another line of text such as the terminator "END". The output of the Python script for generating all 3-bit classes is in this format, and thus can be piped to the families generating program as follows:

```
families <classes.txt >families.txt
```

Alternatively, both programs may be run together using a pipe operator:

```
python3 classes.py | family >families.txt
```


Bibliography

- [1] L. A. Christie, D. P. Lonie and J. A. W. McCall, "Partial structure learning by subset Walsh transform," in *UK Workshop on Computational Intelligence (UKCI)*, pp. 128-135, 2013.
- [2] L. A. Christie, J. A. W. McCall and D. P. Lonie, "Minimal Walsh structure and ordinal linkage of monotonicity-invariant function classes on bit strings," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 333-340, 2014.
- [3] A. E. I. Brownlee, J. A. W. McCall and L. A. Christie, "Structural coherence of problem and algorithm: an analysis for EDAs on all 2-bit and 3-bit problems," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 2066-2073, 2015.
- [4] J. A. W. McCall, L. A. Christie and A. E. I. Brownlee, "Generating Easy and Hard Problems using the Proximate Optimality Principle," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 767-768, 2015.
- [5] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67-82, 1997.
- [6] C. M. Reidys and P. F. Stadler, "Combinatorial Landscapes," *Society for Industrial and Applied Mathematics Review (SIREV)*, vol. 44, no. 1, pp. 3-54, 2002.
- [7] A. V. Arkhangel'skii, "General Topology I: Basic Concepts and Constructures Dimension Theory," L. S. Pontryagin, Ed., Springer, 1990.
- [8] R. W. Hamming, "Error detecting an error correcting codes," *Bell Systems Technical Journal*, vol. 29, no. 2, pp. 147-160, 1950.
- [9] H. Braun, "On solving travelling salesman problems by genetic algorithms," in *Parallel Problem Solving from Nature (PPSN)*, pp. 129-133, 1991.
- [10] H. Mühlenbein, "How genetic algorithms really work, I. Fundamentals," *Parallel Problem Solvings from Nature (PPSN)*, pp. 15-26, 1992.

- [11] J. Horn, D. E. Goldberg and K. Deb, "Long path problems," *Lecture Notes in Computer Science*, vol. 866, pp. 149-158, 1994.
- [12] F. Glover, "Future paths for integer programming and links to artificial intelligence," in *Computers and Operations Research*, pp. 533-549, 1986.
- [13] F. Glover, "Tabu Search: Part I," *ORSA Journal on Computing*, vol. 1, pp. 190-206, 1989.
- [14] F. Glover, "Tabu Search: Part II," *ORSA Journal on Computing*, vol. 2, pp. 4-32, 1990.
- [15] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983.
- [16] P. J. M. Van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, Dordrecht, Holland: D. Reidel Publishing Company, 1987.
- [17] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, Oxford, England: U Michigan Press, 1975.
- [18] T. C. Belding, "The distributed genetic algorithm revisited," in *International Conference on Genetic Algorithms*, pp. 114-121, 1995.
- [19] D. Whitley, S. Rana and R. B. Heckendorn, "The island model genetic algorithm: on separability, population size and convergence," *Computing and Information Technology*, vol. 7, no. 1, pp. 33-47, 1999.
- [20] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley Professional, 1989.
- [21] M. Mitchell, J. H. Holland and S. Forrest, "When will a genetic algorithm outperform hill-climbing?," in *Advances in neural information processing systems*, vol. 6, J. D. Cowan, G. Tesauro, Alspector and Joshua, Eds., Morgan Kaufmann, pp. 51-58, 1994.
- [22] L. Davis, *Handbook of genetic algorithms*, New York, NY: Van Nostrand Reinhold, 1991.

- [23] M. de la Maza and B. Tidor, "An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 124-131, 1993.
- [24] S. K. Shakya, DEUM: A framework for an estimation of distribution algorithm based on Markov random fields, 2006.
- [25] D. E. Goldberg, B. Korb and K. Deb, "Messy Genetic Algorithms: Motivation, Analysis, and First Results," *Complex Systems*, vol. 3, no. 2, pp. 493-530, 1989.
- [26] Y. P. Chen, C. Y. Chuang and Y. W. Huang., "Inductive Linkage Identification on Building Blocks of Different Sizes and Types," *International Journal of Systems Science*, vol. 43, no. 12, pp. 2202-2213, 2012.
- [27] N. R. Pal, S. Nandi and M. K. & Kundu, "Self-crossover-a new genetic operator and its application to feature selection," *International Journal of Systems Science*, vol. 29, no. 2, pp. 207-212, 1998.
- [28] H. Kargupta, "SEARCH, Polynomial Complexity, And The Fast Messy Genetic Algorithm," University of Illinois, 1995.
- [29] E. Corsano, D. Cucci, L. Malagò and M. Matteucci, "Implicit model selection based on variable transformations in estimation of distribution," in *Learning and intelligent optimization*, Y. Hamadi and M. Schoenauer, Eds., Springer Berlin Heidelberg, pp. 360-365, 2012.
- [30] P. Larrañaga and J. A. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, 2002.
- [31] 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.
- [32] M. Pelikan and H. Mühlenbein, "The bivariate marginal distribution algorithm," in *Advances in Soft Computing*, London, Springer London, pp. 521-535, 1999.
- [33] S. Baluja and R. Caruana, "Removing the genetics from the standard genetic algorithm," in *International Conference Machine Learning*, pp. 38-46, 1995.

- [34] H. P. G. Mühlenbein, "From recombination of genes to the estimation of distributions i. binary parameters," in *Parallel Problem Solving from Nature (PPSN)*, pp. 178-187, 1996.
- [35] M. Pelikan, "BOA: the Bayesian optimization algorithm," in *Hierarchical Bayesian optimization algorithm*, Berlin, Springer Berlin Heidelberg, pp. 31-48, 2005.
- [36] S. K. Shakya, J. A. W. McCall and D. F. Brown, "Solving the Ising spin glass problem using a bivariate EDA based on Markov random fields," in *IEEE Congress on Evolutionary Computation (CEC)*. pp. 908-915, 2006.
- [37] A. Petrovski, S. Shakya and J. A. W. McCall, "Optimising Cancer Chemotherapy Using an Estimation of Distribution Algorithm and Genetic Algorithms," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 413-418, 2006.
- [38] A. E. I. Brownlee, M. Pelikan, J. A. W. McCall and A. Petrovski, "An application of a multivariate estimation of distribution algorithm to cancer chemotherapy," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 463-464, 2008.
- [39] Y. Wu, J. A. W. McCall, P. Godley, A. E. I. Brownlee, D. Cairns and J. Cowie, "Bio-control in mushroom farming using a Markov network EDA," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 2991-2996, 2008.
- [40] S. K. Shakya, F. Oliveira and G. Owusu, "Analysing the effect of demand uncertainty in dynamic pricing with EAs," in *Research and development in intelligent systems xxv*, M. Bramer, M. Petridis and F. Coenen, Eds., Springer London, pp. 77-90, 2009.
- [41] R. Kindermann and J. L. Snell, *Markov Random Fields and Their Applications*, American Mathematical Society, 1980.
- [42] J. M. Hammersley and P. Clifford, *Markov fields on finite graphs and lattices*, 1971.

- [43] R. Santana, "A Markov network based factorized distribution algorithm for optimization," in *Machine Learning: ECML*, Heidelberg, Springer, pp. 337-348, 2003.
- [44] S. K. Shakya, J. A. W. McCall and D. F. Brown, "Updating the probability vector using MRF technique for a univariate EDA," in *Starting AI Researchers' Symposium*, pp. 15-25, 2004.
- [45] A. Prügel-Bennett and J. L. Shapiro, "Analysis of genetic algorithms using statistical measures," *Physical Review Letters*, vol. 27, no. 9, 1994.
- [46] A. Rogers and A. Prügel-Bennett, "Modelling the dynamics of a steady-state genetic algorithm," *Foundations of Genetic Algorithms (FOGA)*, vol. 5, pp. 57-68, 1999.
- [47] A. E. I. Brownlee, *Multivariate Markov Networks for Fitness Modelling in a Estimation of Distribution Algorithm*, 2009.
- [48] D. F. Brown, A. B. Garmendia-Doval and J. A. W. McCall, "Markov random field modelling of royal road genetic algorithms," *International Conference on Artificial Evolution*, pp. 65-76, 2001.
- [49] L. Malago, M. Matteucci and G. Valentini, "Introducing ℓ_1 -regularized logistic regression in Markov network based EDAs," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1581-1588, 2011.
- [50] L. Malago, M. Matteucci and G. Pistone, "Stochastic natural gradient decent by estimation of empirical covariances," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 949-956, 2011.
- [51] H. Karshenas, R. Santana, C. Bielza and P. Larrañaga, "Multi-objective optimization with joint probabilistic modeling of objectives and variables," in *Evolutionary Multi-Criterion Optimization*, Springer Berlin Heidelberg, pp. 298-312, 2011.
- [52] S. Shakya and R. Santana, "An EDA based on local Markov property and ibbs sampling, pp. 475-476," in *Genetic and Evolutionary Computation (GECCO)*, 2008.

- [53] M. E. Alden, MARLEDA: Effective distribution estimation through Markov random fields, ProQuest, 2007.
- [54] A. E. I. Brownlee, J. A. W. McCall, Q. Zhang and D. F. Brown, "Approaches to selection and their effect on fitness modelling in an estimation of distribution algorithm," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 2621-2628, 2008.
- [55] P. C. Winter, G. I. Hickey and H. L. Fletcher, *Instant Notes in Genetics*, New York, New York: Springer-Verlag, 1998.
- [56] M. Munetomo and D. E. Goldberg, "Identifying linkage groups by nonlinearity/non-monotonicity detection," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 433-440, 1999.
- [57] M. Munetomo and D. E. Goldberg, "Linkage Identification by non-monotonicity detection for overlapping functions," *Evolutionary Computation*, vol. 7, no. 4, pp. 377-398, 1999.
- [58] A. E. I. Brownlee, J. A. McCall, S. K. Shakya and Q. Zhang, "Structure learning and optimisation in a markov network based estimation of distribution algorithm," in *Exploitation of linkage learning in evolutionary algorithms*, vol. 3, Chen and Yingping, Eds., Springer Berlin Heidelberg, pp. 45-69, 2010.
- [59] E. D. de Jong, R. A. Watson and D. Thierens, "On the complexity of hierarchical problem solving," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1201-1208, 2005.
- [60] M. Tsuji, M. Munetomo and K. Akama, "Linkage identification by fitness difference clustering," *Evolutionary Computation*, vol. 14, no. 4, pp. 383-409, 2006.
- [61] M. J. Streeter, "Upper bounds on the time and space complexity of optimizing additively separable functions," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 186-197, 2004.
- [62] R. B. Heckendorn and A. H. Wright, "Efficient Linkage Discovery by Limited Probing," *Evolutionary Computation*, vol. 12, no. 4, pp. 517-545, 2004.

- [63] M. Tsuji, M. Munetomo and K. Akama, "Population sizing of dependency detection by fitness difference classification," in *Workshop on Foundations of Genetic Algorithms (FOGA)*, pp. 282-299, 2005.
- [64] M. Munetomo, "Linkage identification based on epistasis measures to realize efficient genetic algorithms," in *IEEE World Congress on Computational Intelligence (WCCI)*. pp. 1332-1337, 2002.
- [65] J. Nocedal and S. J. Wright, *Numerical Optimisation*, Vol. 2, New York: Springer, 1999.
- [66] J. D. Schaffer and L. J. Eshelman, "On Crossover as an Evolutionarily Viable Strategy," *International Computer Games Association (ICGA)*, vol. 91, pp. 61-68, 1991.
- [67] S. Droste, T. Jansen and I. Wegener, "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51-81, 2002.
- [68] S. Baluja and S. Davies, "Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space," Carnegie-Mellon University Department of Computer Science, Pittsburgh, PA, 1997.
- [69] K. Deb and D. E. Goldberg, "Analyzing deception in trap functions," in *Workshop on Foundations of Genetic Algorithms (FOGA)*, pp. 93-108, 1992.
- [70] D. Thierens, "Population-based iterated local search: restricting neighborhood search by crossover," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 234-245, 2004.
- [71] D. E. Goldberg, "Genetic algorithms and Walsh functions: part i, a gentle introduction," *Complex Systems*, vol. 3, no. 2, pp. 129-152, 1989.
- [72] D. E. Goldberg, "Genetic algorithms and Walsh functions: part ii, deception and its analysis," *Complex Systems*, vol. 3, no. 2, pp. 153-171, 1989.
- [73] P. Strandmark and F. Kahl, "Pseudo-Boolean Optimization: Theory and Applications in Vision," in *Swedish Symposium on Image Analysis (SSBA)*, 2012.

- [74] K. Smyth, H. H. Hoos and T. Stützle, "Iterated robust tabu search for MAX-SAT," in *Advances in Artificial Intelligence*, pp. 129-144, 2003.
- [75] J. Ryan, "The depth and width of local minima in discrete solution spaces," *Discrete Applied Mathematics*, vol. 56, no. 1, pp. 75-82, 1995.
- [76] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, no. 5, 2014.
- [77] C. García-Martínez, F. J. Rodríguez and M. Lozano, "Arbitrary function optimisation with metaheuristics," *Soft Computing*, vol. 16, no. 12, pp. 2115-2133, 2012.
- [78] C. R. Reeves, "Direct Statistical Estimates of GA Landscape Properties," in *Workshop on Foundations of Genetic Algorithms (FOGA)*, pp. 91-107, 2001.
- [79] F. Glover and M. Laguna, *Tabu Search, Volume 1*, Springer Science & Business Media, 1998.
- [80] C. R. Reeves and T. Yamada, "Genetic algorithms, path relinking, and the flowshop sequencing problem," *Evolutionary Computation*, vol. 6, no. 1, pp. 45-60, 1998.
- [81] D. R. Hains, D. Whitley and A. E. Howe, "Revisiting the big valley search space structure in the TSP," *Journal of the Operational Research Society*, vol. 62, no. 2, pp. 305-312, 2011.
- [82] J. He, T. Chen and X. Yao, "On the Easiest and Hardest Fitness Functions," *IEEE Transactions on Evolutionary Computation*, vol. PP, no. 99, pp. 1-15, 2014.
- [83] S. Droste and T. Jansen, "Upper and lower bounds for randomized search heuristics in black-box optimization," *Theory of computing systems*, vol. 39, no. 4, pp. 525-544, 2006.
- [84] S. Droste, "Not all linear functions are equally difficult for the compact genetic algorithm," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 679-686, 2005.

- [85] G. R. Harik, F. G. Lobo and K. Sastry, "Linkage learning via probabilistic modeling in the extended compact genetic algorithm (EGCA)," in *Scalable optimization via probabilistic modeling, studies in computational intelligence*, vol. 33, M. Pelikan, K. Sastry and E. Cantú-Paz, Eds., Berlin, Springer Berlin Heidelberg, pp. 39-61, 2006.
- [86] L. Kallel, B. Naudts and C. R. Reeves, "Properties of fitness functions and search landscapes," in *Theoretical aspects of evolutionary computing*, pp. 175-206, 2001.
- [87] R. Santana, P. Larrañaga and J. A. Lozano, "Challenges and open problems in discrete EDAs," Department of Computer Science and Artificial Intelligence, University of the Basque Country, 2007.
- [88] Y. P. Chen, T. L. Yu, K. Sastry and D. E. Goldberg, "A survey of linkage learning techniques in genetic and evolutionary algorithms," 2007.
- [89] Y. P. Chen, *Extending the scalability of linkage learning genetic algorithms: theory and practice*, 2004.
- [90] T. L. Yu, K. Sastry and D. E. Goldberg, "Linkage learning, overlapping building blocks, and systematic strategy for scalable recombination," in *Genetic and Evolutionary Computation Conference (GECCO)*. pp. 1217-1224, 2005.
- [91] P. Pošík and S. Vaníček, "Parameter-less Local Optimizer with Linkage Identification for Deterministic Order-k Decomposable Problems," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 577-584, 2011.
- [92] E. Radetic and M. Pelikan, "Spurious dependencies and EDA scalability," in *Genetic and Evolutionary Computation Conference (GECCO)*. pp. 303-310, 2010.
- [93] I. H. F. E. Witten, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2010.
- [94] H. Mühlenbein, T. Mahnig and A. O. Rodriguez, "Schemata, distributions and graphical models in evolutionary optimization," *Journal of Heuristics*, vol. 5, no. 2, pp. 215-247, 1999.

- [95] A. Ochoa, M. R. Soto and R. Santana, "The edge incident model," in *Symposium on Artificial Intelligence (CIMAFA)*. pp. 352–359, 1999.
- [96] R. Santana, P. Larrañaga and J. A. Lozano, "Interactions and dependencies in estimation of distribution algorithms," in *IEEE Congress on Evolutionary Computation (CEC)*. pp. 1418-1425, 2005.
- [97] M. R. Soto and A. Ochoa, "A factorized distribution algorithm based on polytrees," in *IEEE Congress on Evolutionary Computation (CEC)*. pp. 232–237, 2000.
- [98] A. Rogers and A. Prügel-Bennett, "A Solvable Model Of A Hard Optimisation Problem," in *Theoretical aspects of evolutionary computing*, pp. 207-221, 2001.
- [99] C. R. Reeves, "Predictive measures for problem difficulty," in *IEEE World Congress on Computational Intelligence (CEC)*, 1999.
- [100] S. Roman, *Advanced Linear Algebra with Applications: Volume 1: Vector Spaces and Groups*, CRC, 1993.
- [101] J. J. Sylvester, "Thoughts on inverse orthogonal matrices, simultaneous sign successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers.,," *Philosophical Magazine*, vol. 34, pp. 461-475, 1867.
- [102] B. J. Fino and V. R. Algazi, "Unified matrix treatment of the fast Walsh-Hadamard transform," *IEEE Transactions on Computers*, Vols. C-25, no. 11, pp. 1142-1146, 1976.
- [103] B. Doerr and C. Winzen, "Playing Mastermind with constant-size memory," *Theory of Computing Systems*, vol. 55, no. 4, pp. 658-684, 2011.
- [104] H. Kargupta and B. Park, "Gene expression and fast construction of distributed evolutionary representation," *Evolutionary Computation*, vol. 9, no. 1, pp. 43-69, 2001.

- [105] E. Cantú-Paz and D. E. Goldberg, "Are multiple runs of genetic algorithms better than one?," in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 801-812, 2003.
- [106] C. Echegoyen, A. Mendiburu, R. Santana and J. A. Lozano, "On the taxonomy of optimization problems under estimation of distribution algorithms," *Evolutionary computation*, vol. 21, no. 3, pp. 471-495, 2013.
- [107] C. Echegoyen, R. Santana, A. Mendiburu and J. A. Lozano, "Comprehensive characterization of the behaviors of estimation of distribution algorithms," *Theoretical Computer Science*, 2015.
- [108] D. L. Whitley, A. M. Sutton and A. E. Howe, "Understanding Elementary Landscapes," in *Genetic and Evolutionary Computation Conference (GECCO 2008)*, pp. 585-592, 2008.
- [109] H. H. Hoos and T. Stützle, "SAT 2000," I. P. Gent, H. V. Maaren and T. Walsh, Eds., SATLIB is available online at www.satlib.org, IOS Press, pp. 283-292, 2000.
- [110] R. C. Prim, "Shortest connection networks and some generalizations.," *Bell Systems Technical Journal*, vol. 36, no. 6, pp. 1389-1401, 1957.