Learning, Exploiting and Benchmarking Problem Structures in Real-Valued Evolutionary Optimization

by

Tomasz Michal Oliwa

(Under the Direction of Khaled Rasheed)

Abstract

In the context of real-valued evolutionary optimization in high dimensional domains, understanding and exploiting the problem structure can lead to significant improvements in final result quality while also lowering the computational burdens by cutting down evaluation time. This dissertation presents novel approaches for linkage learning and gene sensitivity detection through machine learning methods in the real-valued domains and a proposed idea to jointly represent these measures. A surrogate-assisted perturbation-check for non-linearity that does not stress the true fitness function is introduced and various machine learning methods are employed and compared in terms of their ability to rank gene importance. Furthermore, novel surrogate-assisted crossover operators that incorporate linkage knowledge through crossover masks are defined and evaluated on synthetic fitness functions to empirically validate their utility. Finally, a new benchmark with overlapping linkage groups of increasing size is presented, which provides a platform for comparison of real-valued global optimization algorithms.

Index words: Machine Learning, Evolutionary Computation, Linkage Learning, Problem Structures, Genetic Algorithms, Benchmark

Learning, Exploiting and Benchmarking Problem Structures in Real-Valued Evolutionary Optimization

by

Tomasz Michal Oliwa

Diplom-Informatiker, University of Koblenz-Landau, Germany, 2012

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2013

©2013

Tomasz Michal Oliwa

All Rights Reserved

Learning, Exploiting and Benchmarking Problem Structures in Real-Valued Evolutionary Optimization

by

Tomasz Michal Oliwa

Approved:

Major Professor:	Khaled Rasheed
Committee:	Krzysztof J. Kochut
	Thiab R. Taha

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2013

Learning, Exploiting and Benchmarking Problem Structures in Real-Valued Evolutionary Optimization

Tomasz Michal Oliwa

May 2013

Acknowledgments

I would like to thank my main advisor, Prof. Dr. Khaled Rasheed, who introduced me to the fascinating and challenging topics of machine learning and evolutionary computation and guided me professionally towards successful research, all of which lead to multiple peer-reviewed publications and shaped this dissertation.

Furthermore, I would like to thank Prof. Dr. Krzysztof J. Kochut, a member of my advisory dissertation committee, who taught me a lot about software engineering and the proper way of leading, managing and realizing software projects, which has been very beneficial for the design and implementation of my research programming.

Also, I am thankful for the help and insight I got from Prof. Dr. Thiab R. Taha, also a member of my advisory dissertation committee.

I am very grateful for the Enterprise Information Technology Services Help Desk for their support and would like to thank its current and former staff.

My thanks also goes to Yan Li who I met at the University of Georgia, and I want to mention my gratitude to her.

Finally, I would like to thank my parents Anna and Leszek Oliwa, who have been caring for me and supporting me all my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this Chapter, the fundamental preliminaries of this dissertation are laid out by giving an introduction to continuous global optimization and its importance to the field of computer science and beyond. Evolutionary algorithms, as the chief optimization methods used, will be briefly outlined. The problem structure notions of linkage and sensitivity are then provided as well as the basic concept of machine learning in regards to the topics of this work. Following this, an overview of the proposed approach is given and the Chapter is concluded with a layout of the organization of the rest of this dissertation.

1.1 Global Optimization

In a global optimization problem, the goal is to locate the best (or a reasonable good and satisfactory) solution from a set of solutions. In mathematical terms, this means to find a global optimum (or a reasonable good and satisfactory local optimum) of a function. Such a task is usually declared as a minimization problem. Any such minimization problem can be equivalently stated as a maximization task (the minimization of the function $f(\mathbf{x})$ is equivalent to the maximization of $-f(\mathbf{x})$), hence without loss of generality, this dissertation will keep its definitions based on minimization.

1.1.1 Definition of Global Optimization

More formally, adopting the following definitions from [54], a global optimization problem for real-valued domains can be defined as:

Definition 1.

$$\begin{aligned} &minimize : f(\mathbf{x}) \\ &subject\ to : \end{aligned} \tag{1.1}$$

$$g_i(\mathbf{x}) \leq 0 \qquad i = 1, ..., p \tag{1.2}$$

$$h_j(\mathbf{x}) = 0 \qquad j = 1, ..., q \tag{1.3}$$

$$\mathbf{x}^{lower} \leq \mathbf{x} \leq \mathbf{x}^{upper} \tag{1.4}$$

where :

- $\mathbf{x} = [x_0, \ldots, x_n]$ is a real-valued vector. Its lower and upper boundaries of permissible values for the particular problem are given by $\mathbf{x}^{lower}$ and $\mathbf{x}^{upper}$, respectively.
- $f(\mathbf{x})\colon \mathcal{X} \to \mathcal{Y}$ is a function with its domain $\mathcal{X} \subseteq \mathbb{R}^n$ and its codomain (target set) $\mathcal{Y} \subseteq \mathbb{R}_{\geq 0}$. $\mathcal{X}$ (the representation space) and $\mathcal{Y}$ are restricted by representational limitations of real numbers on digital computers and $\mathcal{X}$ furthermore by the aforementioned lower and upper boundaries of $\mathbf{x}$.
- g_i and h_j are the inequality and equality constraints, if the problem defines any.

Informally, $\mathbf{x}$ can be interpreted as a possible solution to the problem expressed through the function f, and this function assigns a score (or measure of merit) from the codomain $\mathcal{Y}$ to it. In minimization problems, the score of zero (0) is commonly the best obtainable score. With a global optimization problem stated as such, additional useful definitions can be made:

Definition 2. $\mathbf{x}_0 \in \mathcal{X}$ is a called a **global optimum** with $\mathbf{y}_0 \in \mathcal{Y}$ and $f(\mathbf{x}_0) = \mathbf{y}_0$ if $f(\mathbf{x}_i) \geq f(\mathbf{x}_0)\ \forall\ \mathbf{x}_i \in \mathcal{X}$.

Definition 3. $\mathbf{x}_l \in \mathcal{X}$ is a called a **local optimum** with $\mathbf{y}_l \in \mathcal{Y}$ and $f(\mathbf{x}_l) = \mathbf{y}_l$ if there $\exists \epsilon > 0$ such that $f(\mathbf{x}_i) \geq f(\mathbf{x}_l)$ when $|x_i - x_l| < \epsilon$.

Definition 4. A function with only one global optimum (which also is the local optimum) is called a **unimodal** function, otherwise, if the function has more than one local optimum, it is called a **multimodal** function.

The notion of a fitness landscape as defined in [30] can be further introduced:

Definition 5. The **fitness landscape** is a 3-tuple $(\mathcal{X}, \mathcal{N}, f)$, where $\mathcal{N}$ defines a neighborhood relation that indicates which solutions are neighbors in the representation space.

While $\mathcal{X}$ and f are generally fixed during the process of an optimization, $\mathcal{N}$ depends on an optimization operator that establishes which solutions are reachable from any $\mathbf{x}_i$. To visualize such a fitness landscape, it has become popular to employ the analogy of a landscape with hills, valleys, peaks and plateaus. If $\mathbf{x}$ has two dimensions, these can be interpreted as x and y coordinates of a three dimensional plot, where f assigns the value on the z axis for any $\mathbf{x}$. Finding the global optimum would be akin to searching for the point of lowest elevation (for instance the deepest valley) in such a landscape.

The ruggedness of a landscape refers to the relative changes of altitude in the landscape. A flat landscape with few small hills or valleys would be considered a *smooth* landscape, whereas the existence of numerous and large peaks, valleys and abrupt changes in elevation would indicate a *rugged* landscape. While [30] suggests that (in general) problems with rugged landscapes are harder to solve (due to a more chaotic change of elevation), there are counter-examples, such as the *needle-in-a-haystack* problem [31], featuring a smooth landscape with a single (sharp) deviation.

1.2 Importance of Global Optimization

Global optimization problems are ubiquitous in a vast number of scientific and engineering domains and have been the focus of extensive research across academic disciples. For

the context of this dissertation, prominent journals include the *Evolutionary Computation* journal (MIT Press), the *IEEE Transactions on Evolutionary Computation* and the *Journal of Global Optimization* (Springer). Furthermore,important conferences related to the topics of this work and global optimization are the Genetic and Evolutionary Computation Conference (GECCO), the IEEE Congress on Evolutionary Computation (CEC), and the International Conference on Parallel Problem Solving From Nature (PPSN).

1.3 Evolutionary Algorithms

An Evolutionary Algorithm (EA) is a population-based stochastic optimization algorithm. EAs are often synonymously referred to as Evolutionary Computation (EC) algorithms, and this dissertation adopts this notion, while another descriptive term for such algorithms that has been used is metaheuristic algorithms [42] or just metaheuristics. If EC algorithms are applied for optimization purposes, it is called evolutionary optimization. In the broadest sense, the metaphor of a Darwinian natural selection, the "survival of the fittest", can be taken to symbolize the concepts of EAs. EAs operate on a population of solutions at the same time (as opposed to only having a single solution at a time), they have a stochastic nature (as opposed to deterministic algorithms) by which random variations are applied and they gradually improve solutions during the course of the optimization with no absolute guarantee of fully reaching the global optimum but with the aim to at least closely approximate it. The problem, expressed through a so called *fitness function*, serves as an environmental pressure, and solutions that can adapt to this environment (by virtue of being superior in fitness to their peers) are allowed to pass parts of themselves on to a next generation of solutions.

1.4 Problem Structures

Problem structures are notions of relationships among decision variables and the solution fitness, and linkage and sensitivity are the two problem structure means in this dissertation.

1.4.1 Linkage

The term linkage (sometimes also called epistasis) denotes interactions among decision variables in regards to the fitness. If a change of value in x_i can influence the contribution of x_j towards the solution fitness, x_i and x_j are said to be linked together. Knowledge about the existence and strength of linkage can significantly contribute to the success of optimization runs and help to understand the structure of the problem. For example, linked variables need to be simultaneously optimized together to achieve optima and should not be broken up [83] [89] and optimized separately, and on the other hand, variables that have no interactions with each other can be optimized independently.

1.4.2 Sensitivity

Sensitivity, in the context of global optimization means the direct strength of contribution of a decision variable x_i towards the fitness of its individual. Not all variables have an equal contribution, and there can be cases where some decision variables have a much stronger influence on the fitness of a solution than others. This information can be used to guide the optimization process [10], especially in cases where the dimensionality of the problem is relatively high compared to the number of truly influential variables.

1.5 Machine Learning

A descriptive quote that captures the essence of machine learning is:

> The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.
>
> — Tom Mitchell, [44]

The two major approaches in machine learning that are employed in this dissertation are:

1. **Supervised Learning**: Supervised learning methods aim to capture knowledge or find a function pertaining to the target class or target attributes usually through inductive learning.

2. **Unsupervised Learning**: With unsupervised learning, the algorithms try to detect patterns and structures within the feature space without specific target attributes.

In this dissertation, supervised and unsupervised methods are applied to autonomously construct surrogate models of the fitness function based on examples explored during an optimization. Other machine learning approaches, such as semi-supervised learning or reinforcement learning, are outside the scope of this work.

1.6 Notes on Optimization

In this dissertation, there are no restrictions placed upon f, in particular, f does not need to be differentiable. In fact, some of the functions used in the developed benchmark (for example the Weierstrass Function) are everywhere continuous but nowhere differentiable. This shows one advantage of EAs over conventional optimization methods that rely on gradient information and need derivatives to proceed with the optimization.

In all minimization problems used in this dissertation, the best achievable values (at which global optima are located) are set as 0 (zero). However, because digital computers represent the infinite real-valued domain through a finite number of bits, rounding errors of approximation occur inevitably [21]. One convention that has been established is to replace zero by an epsilon ϵ, such that if an optimization method locates a solution that has an absolute measure of merit $\leq \epsilon$, a global optimum is said to be found.

This Chapter introduced some properties with which optimization functions can be attributed, such as ruggedness, multimodality or linkage. While it seems reasonable to assume that multimodal or linkage-incorporating problems are more difficult to solve than unimodal or fully separable problems, there exist some counter-examples. In [31], a unimodal problem

is presented that despite having a basin of attraction and information provided about its single optimum within the search space remains a very challenging task for a hill climber and requires exponential time to solve; and even genetic algorithms with only mutation operators available struggle with it. In contrast to this, another problem is defined which shows extreme multimodality, yet is deemed to be easy to solve with a genetic algorithm. Similar arguments can be made regarding the interaction of genes. The existence of linkage (in the context of epistasis correlation) in a problem may well increase its solution difficulty but this is not guaranteed [59].

Finally, there is a multitude of proposed approaches for optimization outside of EC such as various hill climbing techniques or simulated annealing. According to the no free lunch theorem [93], there is no single superior algorithm, since any two optimization methods will perform equally when performance is averaged across *all* problems.

1.7 Proposed Approach and Main Contributions

This dissertation introduces novel methods to learn, exploit and benchmark problem structures in real-valued evolutionary optimization, the main contributions of this work are:

1. **Learning Problem Structures**: Both linkage (gene interactions) and sensitivity (gene importance) are considered as crucial measures of a meaningful structural representation for real-valued optimization problems. SA-LINC-R, a surrogate-assisted perturbation based linkage checker with a nonlinearity criterion is presented and empirically validated on several fitness function setups. Furthermore, the machine learning methods of neural networks, support vector machines for regression and ReliefF are employed in a novel comparison with regards to their ability to correctly rank genes in terms of importance. A proposed concept of iterative neighborhoods and a problem structure matrix idea is presented, which is envisioned to represent and characterize overlapping groups of linkage and different sensitivities.

2. **Exploitation of Learned Information**: This dissertation presents a combination of linkage-aware, informed and surrogate-assisted operators. Linkage-aware operators aim to not disrupt gene sets that have been found to possess linkage among their genes. Informed and surrogate-assisted operators create a pool of potential offspring and judge their measure of merit by an approximate model of the fitness function, to then only allow the best judged offspring to survive. Methods from the machine learning domain are used to achieve this.

3. **Benchmarking of Optimization Methods**: A newly created benchmark for real-valued optimization is presented, which combines the benefits of three contemporary benchmarks from the field of evolutionary optimization. This benchmark allows for the empirical analysis and investigation of optimization methods on problems involving linkage groups of increasing sizes with different degrees of overlap and different sensitivities.

1.8 Structure of this Dissertation

This dissertation is organized as follows:

Chapter 2

This Chapter introduces the original variants of EAs and provides a unified, contemporary view thereof for the purposes of this work.

Chapter 3

The basic concepts of machine learning relevant to this dissertation are introduced and surrogate modeling is established.

Chapter 4

Linkage and sensitivity of genes, the two chief problem structure measures studied in this work, are discussed along with established ways of detection, manifestation and

representational linkage models. The chapter also describes additively decomposable functions and perturbation-based linkage learning methods.

Chapter 5

The concept of iterative neighborhoods and a proposed idea to jointly represent linkage and sensitivity in a problem structure matrix is presented. This representation is envisioned to provide a data structure that could be analyzed with regards to a combined structural partaking of linkage and sensitivity. Furthermore, an idea to create crossover masks based on the iterative neighborhood concept is defined.

Chapter 6

Surrogate-assisted crossover operators fueled by the knowledge of the linkage problem structure in terms of partitionings are presented and their experimental setup is discussed. The surrogate-assisted linkage checker SA-LINC-R is defined and fitness function configurations are given to empirically validate it. Following this, the experimental comparison setup of the machine learning methods of neural networks, support vector machines for regression and ReliefF to detect sensitivity is given.

Chapter 7

A novel benchmark with overlapping linkage groups of increasing size for real-valued optimization is described along with its configuration and synthetic fitness function compositions.

Chapter 8

The evaluation of the surrogate-assisted perturbation-check for nonlinearity SA-LINC-R is given alongside with values of precision, recall and F-measure on several test function compositions. An experimental analysis pertaining to the comparison of sensitivity detection through machine learning methods with Spearman's ρ is shown. Furthermore, results and tests for statistical significance are presented for the introduced crossover operators.

Chapter 9

A brief summary is provided along with an outlook on future work and current limitations of the work of this dissertation.

In the appendices of this dissertation, brief instructions on surrogate modeling with Weka and statistical testing with R are provided.

Chapter 2

Contemporary Evolutionary Computation

The following Chapter portrays the field of EC and establishes its terminology and background to then define a unified, contemporary view of an EC algorithm that can be applied to global optimization. Structure, recombination and mutation operators and selection schemes are elaborated on.

2.1 EC Groundwork

To establish a basis for the following pages, the necessary terminology of EC is laid out, followed by a general workflow of an algorithm for evolutionary optimization and the components of such a system, orientated by the established definitions of EC in [17] and further inspired by the descriptions of GAs components in [54].

2.1.1 Terminology

In Table 2.1, the common terms of EC that will be used throughout this dissertation are explained. The origin of these terms is often based on notions from nature.

Table 2.1: Common Evolutionary Computation Terminology.

Term	Definition
Individual	A candidate solution for a problem that is expressed through a fitness function.
Genotype	A low-level representational view of an individual (for example $\mathbf{x}$), often as a chromosome (ordered structure of genes). It is from the domain of the fitness function.
Gene	A single problem variable (locus) in a chromosome.
Allele	A setting of a gene; in this dissertation real-valued representation is used.
Phenotype	The expressed properties of an individual, a higher-level view of what the solution stands for.
Fitness	A numerical value that gives a measure of merit to an individual; it also denotes a point in the codomain of the fitness function.
Population	A bag of individuals, often called μ.
Generation	A population at a certain point in time.
Selection Scheme	The process by which an EA selects a subset from a population, either for mating or survivor selection.
Mating Selection	This defines which individuals will undergo breeding.
Survivor Selection	This defines which individuals are carried over from one generation to the next.
Elitism	The number of best individuals to be guaranteed to survive into the next generation.
Offspring	Bag of individuals that will form the next generation, also known as λ.
Crossover	Recombination operator that recombines parts of parental chromosomes into an offspring chromosome.
Mutation	Operator that modifies the allele(s) of one or more genes.

2.1.2 A short History of EC

The history of algorithms inspired by nature and evolution is outlined in the book "Introduction to Evolutionary Computing" [17] that can be recommended as introductory material. Furthermore, "Essentials of Metaheuristics" [42] is a noteworthy set of lecture notes with algorithm descriptions that is a valuable complement. Also, a unifying framework for EC is given in "Evolutionary computation - a unified approach" [35]. A summary of EC history is provided here. Bremermann [7] and Fraser [19] were among the first to develop the notion and executable programs of evolutionary algorithms. At the same time or shortly thereafter other researchers began to perform research into optimization methods guided by the idea of Darwin's natural selection and created the following fields, which by the end of the 1990s were unified under the umbrella-term EC:

1. **Evolution Strategies**: Rechenberg [68] and Schwefel [75] created Evolution Strategies (ES), which are optimization algorithms with a real-valued representation, truncation selection [42] and possess self-adaptation abilities of their strategy parameters.

2. **Genetic Algorithms**: Genetic Algorithms (GAs) were developed by Holland [29] and later furthered by Goldberg [22], initially to study adaptive behavior [17] with binary chromosomes, fitness proportional selection and an emphasis on the crossover operator.

3. **Evolutionary Programming**: Fogel, Owens and Walsh [18] invented the field of evolutionary programming (EP) that has the goal to evolve programs expressed through finite state machines.

4. **Genetic Programming**: Koza [38] established Genetic Programming (GP), where symbolic tree-based representations of programs are evolved.

In recent years, various nature-inspired approaches that sometimes are also attributed to the field of Computational Intelligence have been considered part of EC, for instance particle

swarm optimization [36], while the field of EC also itself inspired newer methods like estimation of distribution algorithms (EDAs) [45] [39]. In this dissertation, EC methods are applied for global optimization problems with real-valued representations. Linkage and sensitivity detection mechanisms and operators guided by machine learning methods are an eclectic composition of some of the aforementioned systems. Ultimately, this work expands upon the basic concepts of the fundamental EC fields, enables linkage detection and exploitation with the help of machine learning throughout the course of an evolutionary optimization and provides a novel benchmark for real-valued evolutionary optimization.

2.2 EC Workflow

In Figure 2.1 a common workflow of an EC system that is based on the GA model is depicted and the next Sections will give a description of the constituent parts of such an EA.

2.3 Population

In an EA, the multi-set of current solutions is referred to as the *population*. EAs work by iteratively improving the fitness of the individuals of such a population through the generations. A population is a bag of individuals denoted as vectors $\mathbf{x}$ (duplicates are allowed), and its current instantiation in a generation is denoted by μ. The initialization of such a population is in most EAs based on a uniform random assignment of values to each $\mathbf{x}$ with respect to $\mathbf{x}^{lower} \leq \mathbf{x} \leq \mathbf{x}^{upper}$. An alternative to this from design experiment theory is the symmetric Latin hypercube method [95], that aims at sampling the search space in an orthogonal (evenly distributed) manner. Domain expert knowledge can be also used to seed the population with known good solutions or by creating a bias [42] of how initial solution vectors should be generated. Finally, another pre-optimizer such as a hill-climber can be run to obtain a better-than-random initial population [59]. A problem with these approaches is that it may evoke extra computational costs (which can be a serious

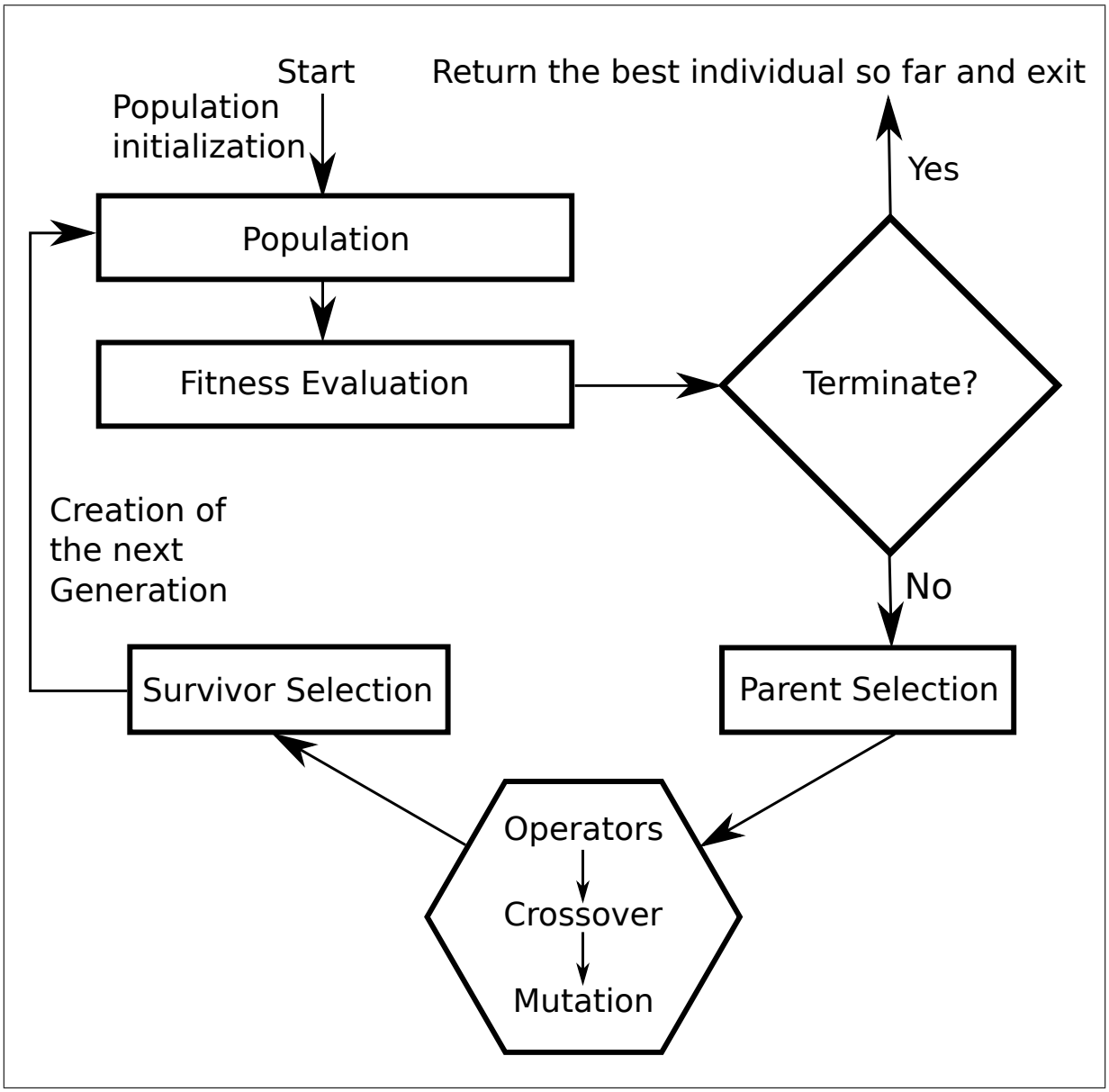

Figure 2.1: The common workflow of an EA.

problem in computationally expensive domains) and that a "typical progress curve of an evolutionary process makes it unnecessary" [17], since the quality standard of such biased solutions can be reached after a few generations using a uniform randomly initialized EA according to [17]. Furthermore, especially when seeding with initially strong solutions that vastly outperform the rest of the population, the problem of premature convergence can arise. Highly fit individuals can quickly take over the whole population, which then leads to a loss of genetic diversity, and the evolutionary optimizer is lead into a local optimum.

2.4 Fitness Evaluation

The *fitness function* assigns a quality measure to each individual. Therefore, each individual has a certain goodness according to this fitness evaluation (with respect to the fitness landscape), and the goal of an EA is to improve (in the case of this dissertation to minimize) the fitness of its individuals. The fitness function can manifest itself as a:

- **Synthetic function**: An artificial, mathematical function $f(\mathbf{x})$ that is used to benchmark EAs, showcase their optimization capabilities, and extrapolate their performance on other, unseen problems. Examples include the benchmark presented in this dissertation, the Real-Parameter Black-Box Optimization Benchmark (BBOP) [25] or the The CEC'2010 Special Session and Competition on Large-Scale Global Optimization (LSGO) [82] benchmark.

- **Real-world problem function**: Technically also expressed as a function $f(\mathbf{x})$, these functions correspond to a real-world problem.

- **Simulation**: A fitness of an individual can be the result of the individual's performance in a simulation, for instance the design and simulation of a missile inlet [64].

- **Human evaluation of the phenotype**: Here, the human judgement replaces the computational calculation of a fitness. This field is also called Interactive Evolutionary

Computation, a survey with about 250 publications in this field can be found in [81] with projects ranging from various fields including graphic, musical and artistic designs, data mining, food industry, control and robotics.

- **Actual physical response**: In evolutionary robotics [53], a control system and behavior of robots is evolved by measuring the performance of these controls via real-world feedback.

In a sense, anything that can provide a measure of quality of individuals can therefore serve as a fitness function. Of importance are the following qualities of functions: A fitness function is called a *blackbox fitness function* if the only information it can provide is a response with a certain fitness score when presented with an individual. While synthetic fitness functions and many real world functions can be investigated by looking at the implementation in the program, a simulation, a human judgement or an actual physical feedback cannot be inspected with the naked eye. A function is *computationally expensive* if there are significant time and processing costs involved in assigning a fitness to an individual, for instance if the fitness function is computed by a simulation that takes a long time to finish. A fitness function is called *noisy* if it is a relation that is not uniquely defined, which means that subsequent calculations of the fitness of the same individual can yield different fitness scores, due to noise. Dynamic fitness functions are uniquely defined, but can change the whole fitness landscape at points in time.

2.5 Termination

There are theoretical investigations of convergence of EC systems [73] [46]. A broad overview with references to bibliographies for this topic is given in [16]. For a stochastic optimization process, not only the probability of convergence is to be considered, but also the number of fitness evaluations to reach it (the speed of of convergence) [16]. For practical purposes,

it becomes necessary to define termination criteria which state when the EA will stop the optimization, the most common are:

- **Best Solution Found**: If the best possible solution has been found (for example a fitness of zero for a minimization problem with a non negative fitness function), the EA can simply stop and present this solution.
- **Fitness Functions Evaluations**: In most cases, an EA will only return an approximation of the best solution, and a typical criterion to stop the search is to pre-define a limit on the number of fitness function calls, after which the best solution (or set of solutions) will be output. Alternatively, this can be equivalently defined through the total number of generations to evolve.
- **Elapsed Time**: Another termination criterion is to provide a fixed time-frame, after which the evolution will be stopped and the best solution(s) given.
- **Stagnation**: This criterion will stop the EA if there is no improvement in the fitness of the best individual after a defined number of generations.

It is also possible to combine these criteria, for instance to set a maximum number of generations to evolve but to also stop if stagnation happens during the optimization.

2.6 Parent Selection

An EA is inspired by biological reproduction to step from one generation of individuals to the next, and parent selection mechanisms provide a way to choose a set of parents to generate new offspring. An important concept is that of *selection pressure* [4], which denotes the amount of emphasis that is put on the fitness of the individual when making a selection decision. With strong selection pressure, individuals with a better than average fitness will be selected predominantly, while a low selection pressure increases the chances for less fit

individuals to pass on their genotypic material. The following parental selection schemes are in use in EAs:

- **Trunkation Selection**: In ES, truncation selection [42] schemes such as (μ, λ) or $(\mu + \lambda)$ retain the fittest fraction of the population, this can, in the context of ES, also be seen as a survivor selection scheme [17].

- **Fitness Proportional Selection**: With the fitness proportional selection [17], each individual has a probability to be selected as a parent that corresponds to its fitness value in relation to the total fitness of the overall population. This can be expressed as probability $p(\mathbf{x}_i) = \frac{f(\mathbf{x}_i)}{\sum_{n=1}^{\mu} f(\mathbf{x}_n)}$ for maximization problems. Furthermoree, a modification called Stochastic Universal Sampling, that is described in [42], chooses highly fit individuals at least once. The disadvantages of proportional selection include the fact that individuals that possess a very high fitness are much more probable to be choosen, which then can lead to the dominance of these individuals leading to premature convergence. On the other hand, a population with little variation in the fitness of its individuals will have a very low selection pressure that can be close to being uniform random, "and having a slightly better fitness is not very "useful" to an individual [17]". Finally, information about the fitness of the whole population is needed to compute the selection probability of individuals.

- **Rank-based Selection**: In rank-based selection [17], the individuals in the population are sorted according to their fitness, and the probability of being selected as a parent is defined by the rank, rather than the actual fitness, of the individual. The actual probability can be calculated using different functions, for example an exponential or linear mapping. Rank-based selection also requires information about the fitness of the whole population.

- **Tournament Selection**: This non-parametric method [42] defines a tournament size (binary tournaments are most common). For each parent, a tournament is held where

individuals are selected uniform randomly and then the most fit among them is chosen. This selection scheme only decides which of the presented individuals is the best and does not require full knowledge of the fitness of the population, which makes it suitable for distributed computing where communication overhead has to be considered. If the tournament size is increased, so does the selection pressure, as more and more emphasis is put on letting only relatively stronger individuals become parents.

- **Restricted Tournament Selection**: Restricted tournament selection [26] combines tournament selection with local competition, so that competing individuals are likely to belong to the same niche. This aims at preserving diversity for multimodal problems.

As stated in [17], tournament selection is the most used parent selection method in GAs. It has therefore been chosen as the selection method in this dissertation.

2.7 Crossover

The crossover operator (more generally called recombination operator) is one of the two major operators in EAs (the other being mutation). In a nut-shell, it crudely models biological recombination of genetic material by mating at least two parent individuals, applying a certain recombination of their genotype and outputting one or more offspring individuals. While it can be implemented as a n-ary operator, most operators in the literature are defined as binary (two parents). It recombines the genetic material of promising parents with the hope that the resulting offspring will exceed their parents' fitness value. A probability $p_{crossover}$, which is often chosen to be within the range $[0.5, 1)$ is used to determine if two parents actually undergo crossover. The following common recombination operators are identified and outlined in [17] and have illustrations in Figures adopted from [54]:

- **n-point Crossover**: This operator has n-crossover points defined that decide if subsequent genetic material from this point on is taken from the first or second parent.

An example of the one-point crossover is shown in Figure 2.2, while Figure 2.3 presents the two-point crossover. These crossovers can be applied to any representation, be it binary, real-valued or otherwise, since they just copy and do not modify the actual alleles. The one-point crossover was one of the first crossovers of GA optimizers. As noted in [17], this crossover has a positional bias, which means that genes that are far apart from each other on the chromosome have a lower chance of being recombined together, and this can create a disturbance if they have a linkage relation.

- **Uniform Crossover**: In the uniform crossover, another probability $p_{uniform}$ is taken, which is commonly set to 0.5. In this case each gene position has an equal chance to be chosen from either of the two parents. This crossover is illustrated in Figure 2.4. This recombination operator has a distributional bias [17], by which on average half of the genetic material will come from each parent. The bias can be modified by altering $p_{uniform}$, such that one parent can get a higher probability to pass on its genes to the offspring than the other.

- **Whole Arithmetic Crossover**: The whole arithmetic crossover [17] takes a weighted sum of each allele to create an offspring. By definition, it is an operator for real-valued representations. The formula for each allele is shown in Equations 2.1 and 2.2, where w is a weight within the range $(0, 1)$, and an example with $w = 0.3$ is given in Figure 2.5. It is to note that alleles obtained through this will be always between the values of the two original parent genes.

$$offspring1_i = w * parent1_i + (1 - w) * parent2_i \tag{2.1}$$

$$offspring2_i = w * parent2_i + (1 - w) * parent1_i \tag{2.2}$$

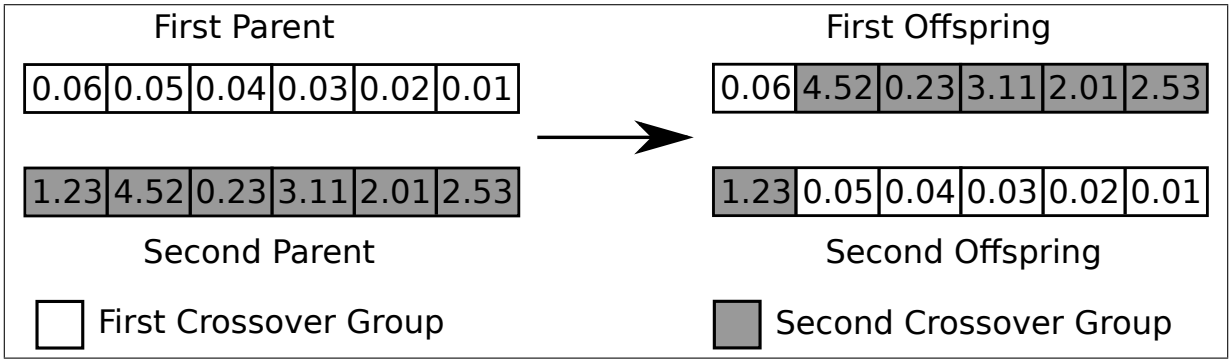

Figure 2.2: The one-point crossover applied to two parents.

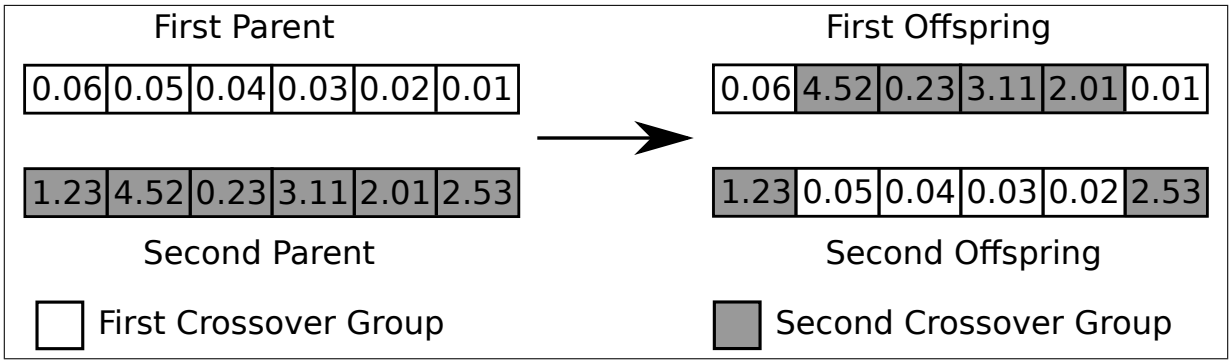

Figure 2.3: The two-point crossover applied to two parents.

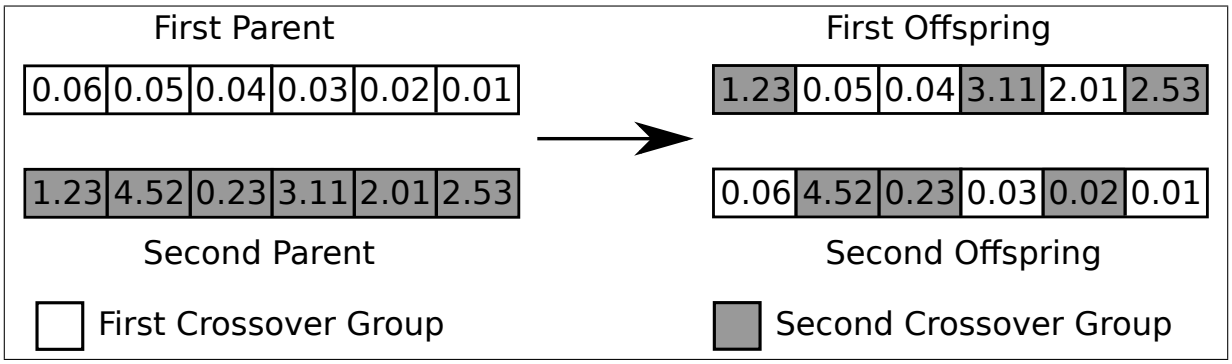

Figure 2.4: The uniform crossover applied to two parents.

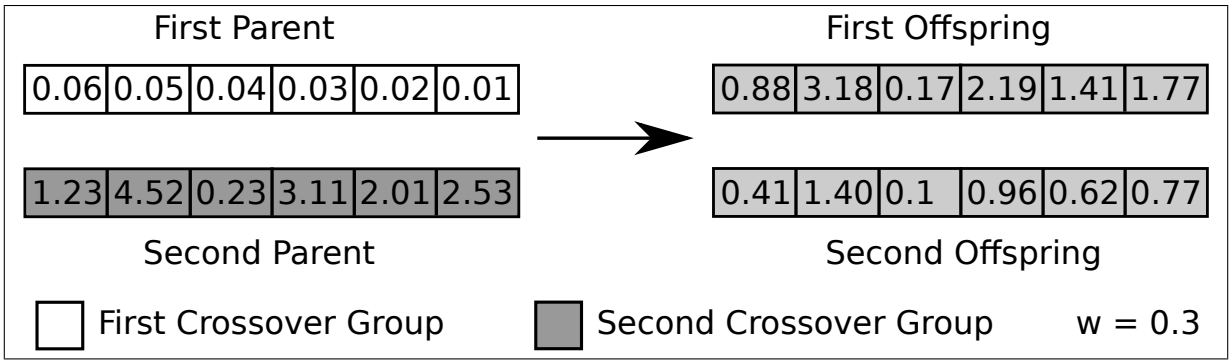

Figure 2.5: The whole arithmetic crossover applied to two parents.

2.8 Mutation

This operation is most commonly performed after an offspring has been created and has the goal to introduce new genetic material into the population by slightly altering an individual. A probability $p_{mutationrate}$, which is often set to a low value, determines if mutation is applied. Heuristics on how to set this parameter range from low fixed settings of $p_{mutationrate} = 0.01$ or $p_{mutationrate} = \frac{1}{\mathbf{x}_{length}}$ [3], to adaptive settings such as the *1/5 rule* (the ratio of fitness improving mutations to all occurred mutations in a generation should be $\frac{1}{5}$) [68] to self-adaptation of mutation rates [17], which are part of the chromosome and evolve alongside the individuals. Some frequently applied mutation operators for real-valued representations, that are described in [17], are (accompanied with Figures inspired from [54]):

- **Non-uniform mutation following a Distribution**: This mutation operator draws a value from a distribution (for instance the Gaussian or Cauchy distribution) with a mean of zero and a standard deviation equal to $p_{mutationrate}$ for each gene and modifies the gene by adding this value, an example is given in Figure 2.6. Bounded Uniform Convolution [42] is a variant of this where $p_{mutationrate}$ decides if a gene is mutated.

- **Uniform mutation**: For each gene x_i a decision is made if it should be mutated or not according to $p_{mutationrate}$. Then, within an allowable $\mathbf{x}^{lower} \leq \mathbf{x} \leq \mathbf{x}^{upper}$, the

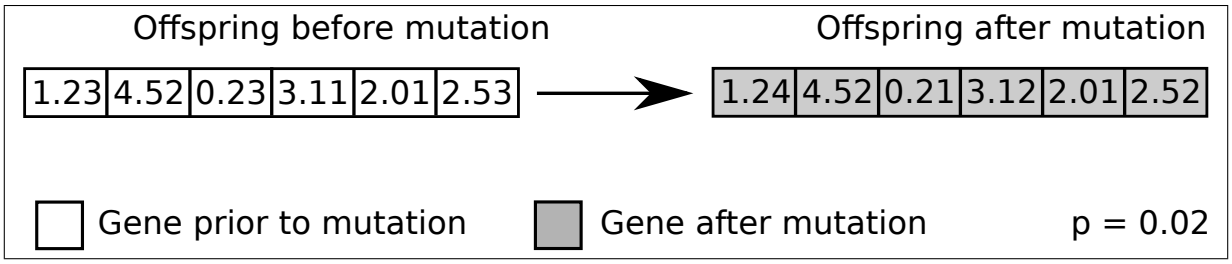

Figure 2.6: The non-uniform mutation applied to an offspring.

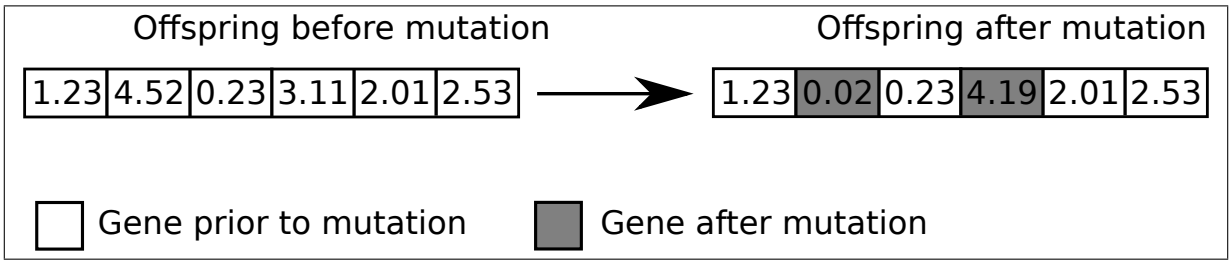

Figure 2.7: The uniform mutation applied to an offspring.

gene x_i is set with a new allele uniformly randomly drawn from within the permissible range. This is shown in Figure 2.7.

2.9 Survivor Selection

The second selection mechanism other than parent selection in an EA is the survivor selection, which defines what individuals from μ (the parent generation) and λ (all created offspring) at generation t will form the population in generation $(t+1)$. The population size of an EA is fixed, and therefore it has to be decided which individuals survive. Two major selection schemes described in [17] are:

- **Steady-State Selection**: With steady-state selection, only a small number of best individuals from λ replace an equal number of individuals from μ. Determining which individuals to remove from the parent population is a decision problem within itself. Some examples are to remove the worst individual(s), or to remove below average

individuals that provide the least genetic diversity for the whole population. Steady-state selection is considered to be more exploitative than generational selection, since fit parents can stay and influence the population for a longer time [42].

- **Generational Selection**: In a generational selection, the size of μ is equal to λ, and after every generation, the group of λ individuals becomes the new generation, while the original μ individuals are removed. The scheme of *elitism* can be used to guarantee that the best k number of individuals (usually $k = 1$) from μ will survive and be carried over to the next generation. Without elitism, the best fitness in a population might get worse from one generation to the next if all offspring are worse then the best parent(s), this can prevent the convergence in some GAs [73].

Chapter 3

Machine Learning

After introducing the concepts of machine learning for supervised and unsupervised learning tasks, this Chapter defines surrogate modeling, which is the application of machine learning methods for fitness approximation in the field of EC. Machine learning in general and particularly surrogate models are essential elements of this dissertation, and their components are briefly described the following pages.

3.1 Supervised Learning

A large part of machine learning research is concerned with inductive learning, that is, to empirically discover rules, concepts or decisions based on existing examples and then extrapolate from this knowledge somehow to be able to make meaningful decisions (classifications or predictions) for unseen examples. "Machine Learning" [44] is a seminal work that covers both theoretical and practical sides of the field. Another book which places more emphasis on application (and the implementation thereof in the open source data mining software Weka[1]) is "Data Mining: Practical Machine Learning Tools and Techniques" [92]. A huge bulk of machine learning algorithms stem from the supervised learning contingent of the field. In supervised learning, the data that a machine learning algorithm is investigating consists of a number of instances, each having a certain number of features (or sometimes

[1] http://www.cs.waikato.ac.nz/ml/weka/

Table 3.1: Play Outside data.

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSE	no
sunny	80	90	TRUE	no
overcast	83	86	FALSE	yes
rainy	70	96	FALSE	yes
rainy	68	80	FALSE	yes
rainy	65	70	TRUE	no

Table 3.2: New Play Outside data instance.

Outlook	Temperature	Humidity	Windy	Play
sunny	82	75	TRUE	?

called attributes). In addition, each instance also includes a target attribute (label or a basis for a target function) that describes this instances characteristics. The following example dataset in Table 3.1 from the Weka datasets exemplifies this. Each row in this dataset can stand for the conditions of a particular day and a decision to play outside or not. Each instance is described by four features and a target attribute, that characterizes that instance. The task of a machine learning method would be to study this dataset and to form a basis for decisions if outside play should happen or not given a new, unseen instance such as the instance in Table 3.2.

In Table 3.1, the target class is binary (a "yes" or "no" decision), therefore such a concept learning can be seen as a classification task, where instances such as the one in Table 3.2 have to be labeled (or classified) appropriately. In other problems, called regression problems, the target attribute is real-valued, and therefore a machine learning method can be viewed as approximating (or modeling) the underlying function that formed the data instances.

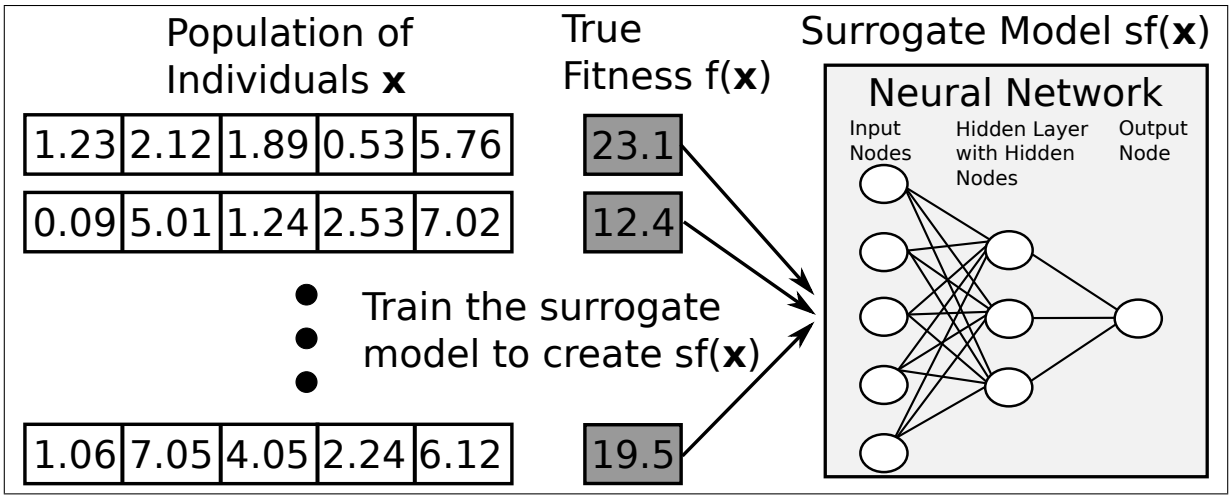

Figure 3.1: Training a surrogate model.

3.2 Surrogate Models

To apply supervised machine learning methods for EC, the population (or possibly a multi-set collection of every individual encountered so far) can be viewed as a dataset with each individual being an instance thereof. Each specific gene x_i would represent a specific feature and its allele the attribute value. The fitness of the individual can be then interpreted as the target attribute. With this approach, suddenly a large collection of machine learning algorithms are directly applicable to any global optimization problem. This approach is known as surrogate-assisted evolution (SAE). Figure 3.1 shows how a population is given as input to a surrogate model learner (a neural network) to learn the approximation of the fitness function $sf(\mathbf{x})$ based on all individuals and the true fitness function $f(\mathbf{x})$. Figure 3.2 sketches a fitness approximation through $sf(\mathbf{x})$. A survey and discussion on SAE with function approximator examples of neural networks, quadratic least squares, support vector machines and kriging (also known as Gaussian processes) can be found in [34]. Contemporary frameworks which implement an adaptive variety of surrogate models and surrogate control have been empirically studied in [41].

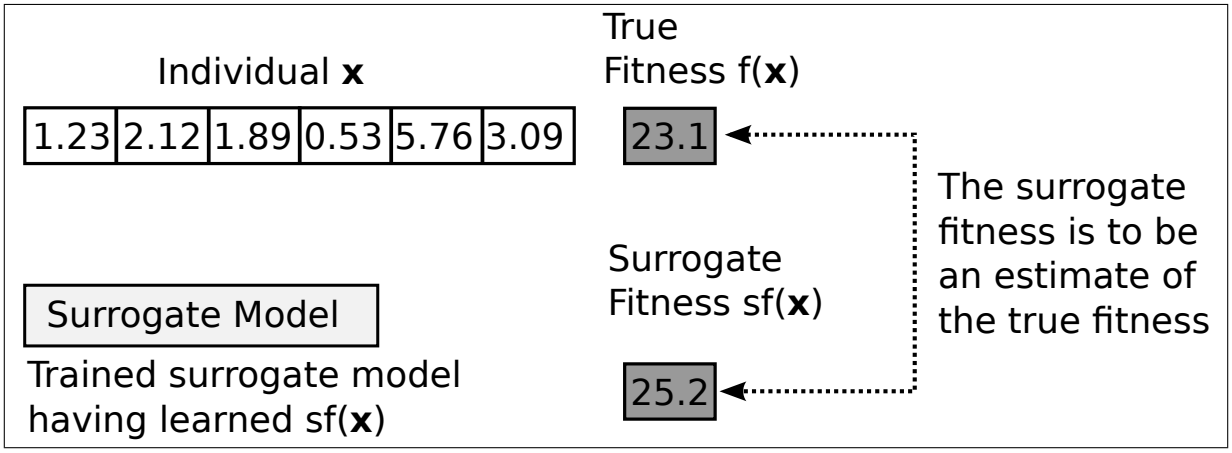

Figure 3.2: A surrogate model as a function approximator.

Chapter 4

Linkage and Sensitivity

The two main approaches to represent or assess problem structure in this dissertation are linkage and sensitivity of genes. Linkage and sensitivity learning and detection techniques are discussed and representational models of interacting groups of genes, so called linkage groups, are presented.

4.1 Dimensionality Reduction

Dimensionality reduction, a methodology commonly used in machine learning and optimization, aims at reducing or transforming the features of a problem while still retaining their usefulness in the context of the problem. The motivations for doing so include:

- The combinatorial explosion of complexity with an increasing dimensionality puts a computational burden on algorithms making large-scale problems very difficult to tackle or even intractable. Such an exponential increase in complexity is also known as the "curse of dimensionality" [5].
- Even a single random binary feature that serves as noise to a decision tree learner can degrade its performance up to 10% [92].
- The features may need to be transformed into a different format (numerical, ordinal, binary) to meet a machine learning algorithm's input requirements.

- While technically the opposite of dimensionality reduction, the introduction of a new feature based on multiple existing features can be benefitial to the learner, for instance an "age" attribute [92] can be helpful even though its value can be derived from other date attributes.

In EC, problem structure detection can be thought of as a step towards dimensionality reduction, where successful analysis of the structural dependencies and importance of genes leads to mechanisms to meaningfully exploit linkage groups (reduced subsets of the whole chromosome) and gene sensitivities to significantly lower the overall number of function evaluations to reach an acceptable solution quality.

4.2 Linkage

Learning linkage is an active topic of research in EC. Two books that provide a fairly recent overview on learning, exploitation and applications are "Exploitation of Linkage Learning in Evolutionary Algorithms" [11] and "Linkage in Evolutionary Computation" [12]. Another helpful collection of mostly perturbation-based approaches of linkage learning is presented in [89]. Definition 6 gives an intuitive way to understand the concept of linkage:

Definition 6 (Linkage)**.** If a change of value in gene x_i can influence the contribution of gene x_j towards the solutions fitness, x_i and x_j interact and are said to be **linked** together.

A linkage relation is therefore an interdependence between genes in regards to the fitness, and is often considered to be symmetric, i.e. if gene x_i is linked to gene x_j, then the opposite is also true. An important concept to describe certain functions that have been a focus of linkage learning research is that of an additively decomposed function (ADF) [47], sometimes also called additively decomposable function. An ADF adopting the formalism of [46] is provided in Definition 7.

Definition 7 (Additively Decomposed Function)**.**

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}_{s_i}) \tag{4.1}$$

where $S_1, \ldots, S_m$, $S_i \subseteq 1, \ldots, n$ are index sets, f_i are functions only applied to the genes of S_i. These are defined as $\mathbf{x}_{s_i}$. Furthermore, S_i can be called the scope of f_i.

ADFs may involve overlapping, such that two functions f_a and f_b would have a gene in common in their scopes. In [89] it is stated that many real-world problems are "loosely decomposable" through such an ADF with a degree of overlapping. This concept of ADFs has been applied in the creation of the benchmark proposed in this dissertation.

Linkage can be used to investigate separability of functions. A notion of separability is defined in [94], where a fitness function $f(\mathbf{x})$ is deemed to possess separability under the following conditions: The function $f(\mathbf{x})$ is fully separable, if Equation 4.2 implies Equation 4.3 $\forall k \in \{0, .., n\}$ and $\mathbf{x}$ and $\mathbf{x}$**'** are both $\in \mathcal{X}$, and $\forall\ \mathbf{y} \in \mathcal{X}$.

$$\left.\begin{aligned} \mathbf{x} &= \{x_0, .., x_k, .., x_n\} \\ \mathbf{x}\text{'} &= \{x_0, .., x_k^{'}, .., x_n\} \end{aligned}\right\} \quad f(\mathbf{x}) < f(\mathbf{x}\text{'}) \tag{4.2}$$

$$\left.\begin{aligned} \mathbf{y} &= \{y_0, .., x_k, .., y_n\} \\ \mathbf{y}\text{'} &= \{y_0, .., x_k^{'}, .., y_n\} \end{aligned}\right\} \quad f(\mathbf{y}) < f(\mathbf{y}\text{'}) \tag{4.3}$$

However, this definition only holds without modifications for non-noisy and non-dynamic (changing) fitness landscapes. Informally, if x_k is perturbed into $x_k^{'}$ (all other genes being equal), and this leads to a decrease in fitness, then for all other individuals $\in \mathcal{X}$ the same needs to hold. If it is not the case, another gene's allele is necessarily influencing the fitness (and therefore these two genes are *linked*) and full separability cannot be claimed.

The question arises why and how this might be important for global optimization tasks involving EAs. In [89] it is stated that:

> In genetic algorithms, it is essential to process building blocks (BBs) effectively through genetic recombination such as crossovers. To apply simple GAs, tight linkage of BBs is necessary in encoding strings because simple genetic operators such as one-point crossovers should disrupt BBs when they are encoded sparsely over a string.
>
> — Miwako Tsuji and Masaharu Munetomo, [89]

With black-box optimization problems it is unforeseeable which genes share interactions with each other as the scopes S_i of the functions that this problem might be based upon are unknown. Furthermore, even if some of the groups of genes which have interactions are tightly encoded as neighbours on a chromosome, a linkage-blind operator can still disrupt linkage groups most of the time. To remedy this, linkage learning approaches need to solve (at least) the following problems:

1. Which genes are linked to each other and how is their linkage characterized?
2. What representational model can be used to express linkage structures?
3. How can this structural information about linkage be incorporated into the optimization process?

What follows is a non-exhaustive overview of influential and current linkage learning methods and representational models in EC.

4.3 Representational Models

With regards to linkage, it has been stated that:

> The goal of linkage learning, while ambitious, is only a small part of the more general goal of representation learning.
>
> — Georges Harik, [27]

Detecting interactions among genes needs to be coupled a method to represent this information in a meaningful way in the context of the problem. The extended compact GA (ECGA) is a type of EDA that defines such a structure through a Marginal Product Model (MPM), where probability distributions over more than one gene are modeled, which was shown to be very beneficial on non-overlapping sum of binary trap functions. Essentially, groups of interacting genes can be seen by the MPM as members of a partition of the chromosome. With the MPM, two groups of interacting variables are merged into one if a dependency between theses groups can be detected. Every gene $\{x_i\}$ belongs to exactly one linkage group (or forms its own one-sized group if it has no epistasis with other genes). One downside of such a model is that if there is only a tiny dependency among two otherwise unrelated groups, they will be merged regardless. In a drastic scenario, if the problem consists of several large linkage groups that only share one variable with each other, the MPM will represent this problem as one giant block of genes and not provide any other structural information, as described in [85].

Another way to represent an interaction structure is through a dependency structure matrix (DSM). This has been done by the dependency structure matrix (driven) genetic algorithm (DSMGA) [97] [96]. The DSM is an adjacency matrix, where the DSM_{ij} entry represents a measure of the interaction between genes x_i and x_j. In the case of a binary matrix this defaults to 0 as no interaction and 1 as interaction present. The DSMGA calculates its matrix through a mutual information measure and defines the following subproblem for each problem that it is solving: What is the optimal clustering of the DSM such that "nodes within a cluster are maximally dependent and clusters are minimally interacting" [97].

The Bayesian Optimization Algorithm (BOA) [61] is a type of an EDA that uses Bayesian networks to build a model of the structural composition of the problem by defining the relationships (edges) among variables (nodes) in a Bayesian network. It is capable of modeling overlapping interaction structures and has been extended to create the hierarchical Bayesian optimization algorithm (hBOA) [60], which defines Bayesian networks with decision graphs

and uses niching. An investigation of linkage learning in EDAs with a focus on the hBOA and problems involving parity functions is provided in [13].

One of the challenges that EDAs typically face is the time requirement for model building, especially if done repeatedly. ClusterMI [15], a model building method, has been proposed to alleviate this with ideas inspired from both the ECGA and the DSMGA. Furthermore, while the aforementioned approaches have been originally developed focusing on binary representations of the chromosomes, there exist extensions such as the real-coded ECGA (rECGA) [40] or rBOA [1].

Tree representations for linkage structures have also been applied. A recent variant is the Linkage Tree Genetic Algorithm (LTGA) [84] [86]. It uses the information theory based entropy and hierarchical clustering to build a tree-structure of linkage bottom-up and has been implemented for binary representations. A comparison of LTGA modifications is shown in [23]. In the LTGA, a hierarchical tree is constructed in a bottom-up manner by merging genes that have epistasis with each other. When two genes are merged into a node, this node is treated as a new entity that might have dependencies with other genes or other nodes. In this tree, the lower a node is located, the stronger the dependency is present. However, it is not possible to directly model overlapping linkage groups [23], since child nodes are completely merged into a parent node.

Recently, the concept of Linkage Neighbors (LN) has been proposed [6] for binary chromosomes, where linkage is calculated through a likelihood-ratio test and each gene is associated with its own family of subsets (the genes that are in its neighborhood). This allows a set of genes to be logically connected to each gene and aims to represent overlapping or transitivity relations between genes. Also, the notion of a Multiscale Linkage Neighbors (MLN) model is introduced that assigns multiple linkage sets per gene.

Linkage representation models do not necessarily correspond to an additive decomposition of the fitness function. In [85], the three models MPM, LN and LTGA are described and empirically compared in the context of a family of subsets. An illustrative example in Figure

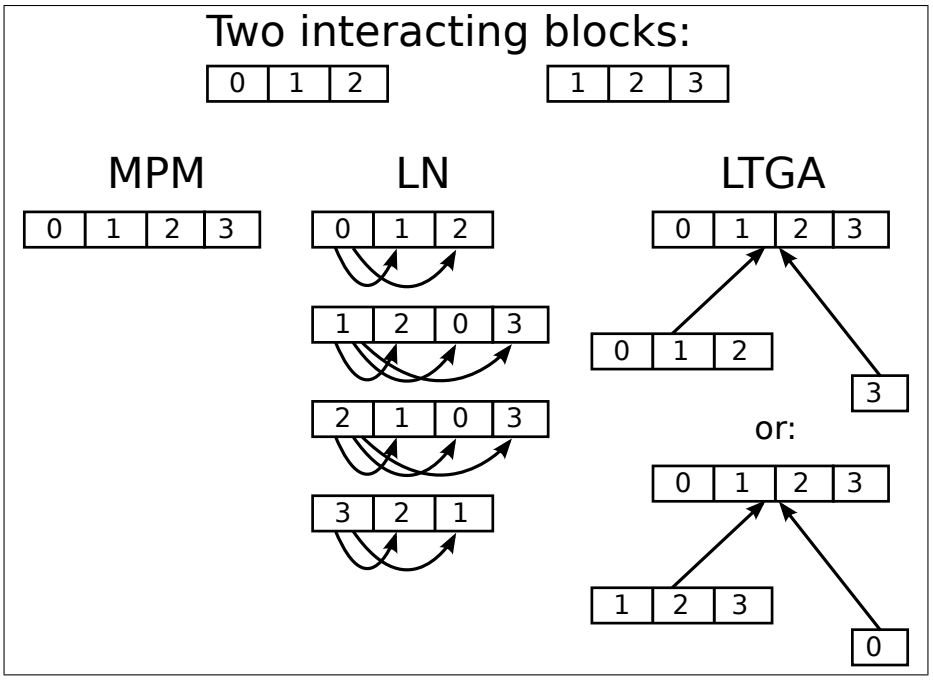

Figure 4.1: Two building blocks represented through the MPM, LN and LTGA models based on the family of subsets.

4.1 depicts this, where the two building blocks $\{x_0, x_1, x_2\}$ and $\{x_1, x_2, x_3\}$ are represented.

The results in [85] show that, in the context of family of subsets, using a predetermined model on a NK-landscape and MAXCUT fitness functions leads to worse results than the usage of a predetermined linkage tree model, which cannot model overlapping accurately. This can initially seem surprising since the predetermined model "is an exact copy of the underlying structure of the additively decomposable fitness function" [85]. Furthermore, a learned linkage tree (which may include mistakes about the detected interactions) shows an even greater performance increase compared to a predetermined linkage tree.

Finally, in [59] an empirical investigation with hBOA and a GA about the relationship of epistasis correlation and problem difficulty on nearest-neighbor NK landscapes for binary representations is carried out. While the results show that epistasis correlation is not a sole indicator to judge problem difficulty, it can be said that:

> Nonetheless, given our current understanding of problem difficulty, there is no doubt that introducing epistasis increases the potential of a problem to be difficult.
>
> — Martin Pelikan, [59]

This adds to the need for more testing, benchmarking and theoretical research in this area, especially with complex overlapping linkage structures and in particular with real-valued representations, as these problem characteristics are common in many real-world problems. The benchmark provided in this dissertation takes a step to establish a common test-ground for real-valued problems with overlapping linkage of varying size.

4.4 Existing Perturbation-based Linkage Learners

This Section describes some perturbation-based linkage learning algorithms [89] and comments on their strengths and drawbacks as a preparation for SA-LINC-R, the perturbation-based surrogate-assisted linkage check idea in this dissertation. However, it should be noted that methods that detect initial linkage based on pair-wise checks, regardless of being of perturbation-based or EDA nature, will encounter difficulties with functions that do not show lower-order dependencies but express a higher order dependency among some variables, such as the parity function [13].

4.4.1 Linkage Identification by Nonlinearity Check

One of the earliest linkage detection mechanisms for genetic algorithms with direct gene-pair investigations is Linkage Identification by Nonlinearity Check (LINC) [49] [89]. It is a perturbation-based method, defined on a binary representation (the perturbations being bit-flips), which uses non-linearity as a linkage criterion. The method calculates for each unique gene pair the quantities shown in Equations 4.4, 4.5 and 4.6:

$$\Delta f_i(\mathbf{x}) = f(...\neg x_i...) - f(...x_i...) \tag{4.4}$$

$$\Delta f_j(\mathbf{x}) = f(...\neg x_j...) - f(...x_j...) \tag{4.5}$$

$$\Delta f_{ij}(\mathbf{x}) = f(...\neg x_i \neg x_j...) - f(...x_i x_j...) \tag{4.6}$$

If for any individual $\in \mathcal{X}$ Equation 4.7 is true, then the corresponding genes x_i and x_j are said to be linked to each other in a non-linear fashion. Otherwise, there exists only a linear dependency among these genes.

$$|\Delta f_{ij}(\mathbf{x}) - (\Delta f_i(\mathbf{x}) + \Delta f_j(\mathbf{x}))| > \epsilon \tag{4.7}$$

The constant of ϵ is used instead of an absolute zero to counter floating-point irregularities on digital computers as described in Chapter 1.

- **Strengths**: LINC is relatively easy to implement and provides a direct detection mechanism for non-linear interactions on any gene-pair in the neighborhood of the individual.

- **Drawbacks**: In [48] it is shown that LINC fails to detect proper linkage for sum of trap functions with an exponent and is "vulnerable to nonlinearity of overall fitness functions" [48]. Also, non-linearity as criterion is not always sufficient to decide that two genes should be optimized together, for instance the Sphere function can be made non-linear yet still independently optimizable on each dimension. An example with the onemax Function is given in [58]. Furthermore, $\mathcal{O}(l^2)$ perturbation-checks per individual need to be performed for a complete linkage picture, where l is the dimensionality of the problem. This means three extra calls to the fitness function per linkage pair. Furthermore, the authors recommend performing this check on every individual of the population, since if LINC fails to detect linkage, it does not exclude the posibility of epistasis between two genes in another part of the search space. This makes LINC

potentially very computationally expensive. Finally, this method is only defined for binary representation.

4.4.2 LINC for Real-Coded GAs

Linkage Identification by Nonlinearity Check for Real-Coded Genetic Algorithms (LINC-R) [83] is an adaption of LINC to real-valued representations. The perturbations from Equations 4.4, 4.5 and 4.6 are not binary flips but instead uniformly randomly drawn Δx_i and Δx_j that are added to the corresponding genes for a perturbation (within allowable constraints) check.

- **Strengths**: LINC-R can directly check a gene pair for non-linearity interaction and is as straight-forward to implement as LINC. In [83] this approach is shown to be superior on synthetic fitness functions to a linkage learner that calculates correlations to estimate linear dependencies.

- **Drawbacks**: Just like LINC, LINC-R does not provide a guarantee that if a linkage check fails, no linkage between the checked genes exists. The perturbation is performed by adding a uniformly drawn value (within permissible bounds of the chromosome). If all four points $(x_i, x_j, x_i + \Delta, x_j, x_i, x_j + \Delta, x_i + \Delta, x_j + \Delta)$ are outside of the non-linear region, LINC-R will fail to detect the interaction. Hence, to increase the chances of finding linkage, more than one individual needs to undergo a linkage check, which brings the computational costs to the same level as LINC.

4.4.3 Linkage Identification by non-Monotonicity Detection

In [50], Linkage Identification by non-Monotonicity Detection (LIMD) is presented. It performs perturbation checks with the goal to detect linkage based on non-monotonicity. For each gene-pair check, Equations 4.4, 4.5 and 4.6 are calculated but the criterion to decide if linkage exists is given in Equations 4.8 and 4.9.

$$\text{if } (\Delta f_i(\mathbf{x}) > 0 \text{ and } \Delta f_j(\mathbf{x}) > 0) \text{ then } (\Delta f_{ij}(\mathbf{x}) > \Delta f_i(\mathbf{x}) \text{ and } \Delta f_{ij}(\mathbf{x}) > \Delta f_j(\mathbf{x})) \quad (4.8)$$

$$\text{if } (\Delta f_i(\mathbf{x}) < 0 \text{ and } \Delta f_j(\mathbf{x}) < 0) \text{ then } (\Delta f_{ij}(\mathbf{x}) < \Delta f_i(\mathbf{x}) \text{ and } \Delta f_{ij}(\mathbf{x}) < \Delta f_j(\mathbf{x})) \quad (4.9)$$

Non-monotonicity is argued to be a more reliable criterion for linkage than non-linearity, and that "the non-linearity condition is sometimes too strict to optimize real-world problems" [89]. This is exemplified on a fitness function which consists of the sum of binary trap functions as shown in Equations 4.10 and 4.11. LINC will detect one single linkage group for all genes of $\mathbf{x}$ (because the whole fitness function is raised to the power of two and therefore a non-linear interaction is occurring for all genes), whereas LIMD is shown to capture the 5-bit trap functions as separate groups.

$$f(\mathbf{x}) = [\sum_{i=1}^{10}(f_i(u_i)]^2 \quad (4.10)$$

$$f_i(u_i) = \begin{cases} 4 - u_i, & \text{if } 0 \leq u_i \leq 4 \\ 5, & \text{if } u_i = 5 \end{cases} \quad (4.11)$$

In [51], the authors extend LIMD with a measure to model overlapping linkage groups called tightness detection (TD) that is shown in Equation 4.12. $n1(x_i, x_j)$ refers to the number of linkage groups where both x_i and x_j occur, and $n2(x_i, x_j)$ is the number of such groups where either x_i or x_j (XOR) are included. The TD measure will be [0..1] for any pair of genes and denotes their tightness or closeness.

$$TD(x_i, x_j) = \frac{n1(x_i, x_j)}{n1(x_i, x_j) + n2(x_i, x_j)} \quad (4.12)$$

- **Strengths**: LIMD checks for non-monotonicity, which can capture dependencies that are hidden behind non-linearity. The addition of TD provides an estimate of the tightness of genes in overlapping linkage scenarios.

- **Drawbacks**: Like LINC and LINC-R, three extra perturbation checks are necessary to investigate a gene pair. It is important to note that the authors write "It is still not clear on what class of problems the LINC and the LIMD procedures outperform the other methods ..." [50]. Furthermore, this was only developed for binary representations.

4.4.4 Dependency Detection by Fitness Clustering

In [88], Dependency Detection by Fitness Clustering (D^5) and its extension to the real-valued domain, rD^5 and presented. An overview is also given in [89]. This method combines a perturbation-based approach with estimation of distribution methods to calculate linkage. Only $\mathcal{O}(l)$ extra fitness calculations per individual are required, but just as with LINC and its derivatives, the use of a population that is large and varied enough to capture dependencies is recommended.

D^5 works as follows: To find which genes interact with gene x_i, $\Delta f_i(\mathbf{x})$ is calculated for each individual. The value of the fitness difference is taken as the individuals classification attribute and the population of perturbed individuals is clustered based on it. Then, in a bottom-up approach using the notion of entropy, linkage groups for x_i are created. This process is then repeated for all genes. rD^5 quantizes (discretizes) the domain of the fitness function to apply the same technique. An extension to D^5 for the binary domain is presented in [87], where it is applied together with the Apriori data mining algorithm and is found to require a lower population size to obtain comparable results to the original D^5.

- **Strengths**: (r)D^5 requires only $\mathcal{O}(l)$ extra fitness calls per individual. It provides means to calculate linkage for binary and real-valued scenarios.

- **Drawbacks**: The size of linkage groups has to be provided beforehand to the algorithm as a parameter, otherwise D^5 needs to be called successively until a satisfactory result is found.

4.4.5 Inductive Linkage Identification

Inductive Linkage Identification (ILI) [32] shares a similarity with D^5 in that it requires $\mathcal{O}(l)$ extra fitness calculations per individual, and also needs a set of individuals to operate on. Just like D^5, to find which genes are linked to gene x_i, ILI performs a $\Delta f_i(\mathbf{x})$ (a binary bit flip is performed as perturbation) for each individual in the population and uses the value of this fitness difference as a classification attribute. However, here, a decision tree (ID3) [44] is formed with gene x_i forced to be the root node, and then every gene that shows up in the tree as a node is taken to be a linked gene to gene x_i. The rationale of ILI is that since the classification attribute is a fitness difference on a particular perturbed gene, and this gene is furthermore taken as the root node, only genes that interact with it can come into consideration as further classification nodes. This method can detect overlapping structures. This was shown on overlapping binary trap-functions.

- **Strengths**: ILI requires only $\mathcal{O}(l)$ extra fitness calls per individual and can represent overlapping linkage structures.

- **Drawbacks**: ILI only works with binary representation, and unlike LINC it does not provide a numerical indication of the strength of the linkage; it only gives a binary decision about it.

4.5 Sensitivity

The term sensitivity denotes the impact of a parameter or variable on an outcome or measure of merit, and Definition 8 states this in the context of EAs.

Definition 8 (Sensitivity)**. Sensitivity** is defined as the direct strength of contribution or effect of a gene x_i towards the fitness of the individual.

In this dissertation, it can be considered as the importance of a gene in regards to the fitness. This can be viewed as similar to the statistical term *main effect* of a variable, which has been investigated for GAs in [69] through the ANOVA method.

An example of sensitivity, similar to the one in [54], can be given as follows: The fitness function given in Equation 4.13 is the three dimensional Schwefel 1.2 Function (and will be investigated further in Chapter 6). A possible individual could be $\mathbf{x} = [2.0, 2.0, 2.0]$, and its fitness calculation is shown in Equation 4.14. Perturbation operations on x_0 and on x_2 with a $\Delta = 1.0$ are shown in Equations 4.15 and 4.16, respectively. The fitness scores indicate that the same Δ on x_2 had less of an effect in the change of the fitness of the individual than the perturbation on x_0. Therefore, it can be said that in this situation, the sensitivity of x_0 is higher than that of x_2.

$$f(\mathbf{x}) = x_0^2 + (x_0 + x_1)^2 + (x_0 + x_1 + x_2)^2 \tag{4.13}$$

$$2.0^2 + (2.0 + 2.0)^2 + (2.0 + 2.0 + 2.0)^2 = 56 \tag{4.14}$$

$$\mathbf{3.0}^2 + (\mathbf{3.0} + 2.0)^2 + (\mathbf{3.0} + 2.0 + 2.0)^2 = 83 \tag{4.15}$$

$$2.0^2 + (2.0 + 2.0)^2 + (2.0 + 2.0 + \mathbf{3.0})^2 = 69 \tag{4.16}$$

4.5.1 An Epistasis Measure Based on the Analysis of Variance for the Real-coded Representation in Genetic Algorithms

The algorithm in [10] is able to calculate sensitivity (called the main effect) and also interactions of genes of real-valued GAs by applying the analysis of variance (ANOVA) method on a population. The work focuses on the sensitivity detection and its exploitation with an adaptive mutation operator. It is shown to give an estimate of interaction on three benchmark functions. In [9], the authors extend this work with a linkage aware crossover operator.

- **Strengths**: This approach does not need extra fitness functions to calculate the interactions of genes and also gives an indication of the importance of the impact of each gene.
- **Drawbacks**: This method has only been tested on three non-overlapping fitness functions, and results for effective linkage detection are only available for three dimensions with gene values between $[-1, 1]$.

4.6 Conclusion

Linkage learning is a well studied concept in EC, and the sensitivity of genes has also been considered an important factor in the context of GAs. They can help to understand the problem structure and the importance of decision variables, represented through genes. Furthermore, the presented detection techniques can help to improve the quality of solutions and detect structures within problems. Such notions of a problem structure can also provide valuable insight about the search space. However, many of these methods have been primarily tested on binary representations, and the combination of linkage and sensitivity is a topic that requires further investigation. Finally, to empirically compare some of the aforementioned approaches on problems with overlapping linkage groups of varying size, a novel benchmark has been created and is defined in this dissertation.

Chapter 5

Ideas of a Joint Representation of Problem Structure

This Chapter provides an idea of a joint representation of problem structure in EC using the present linkage and sensitivity knowledge. First, the concept of iterative neighborhoods is discussed, in which the neighborhoods start with a LN model and are incrementally build up from direct interactions towards a MPM neighborhood. Following this, a proposed way of obtaining crossover masks is shown. Finally, the notion of a problem structure matrix idea is presented, which is envisioned to combine linkage information and sensitivity, and an example is shown in which the iterative linkage neighborhood with a reference to a distance metric and the sensitivity of genes are combined.

5.1 Iterative Linkage Neighborhoods

This Section details the concept of iterative linkage neighborhoods. They are based on the notion of LN, where each variable defines its own neighborhood of linked genes. By stepping through transitive relationships among linked genes, similar to the chain of subproblems [62], these neighborhoods are built as a possible foundation for a problem structure matrix and its distance metric.

5.1.1 Preparation

Iterative linkage neighborhoods are one way to represent the interaction structures that form the problem's representational model; they are, in essence, the sets composed through transitive steps of pairwise interactions. To build them, at first a pair-wise linkage checker such as LINC-R needs to calculate pairwise dependencies among genes and store them in for instance a $n \times n$ two-dimensional dependency matrix M (with n being the dimensionality), where the (i, j) entry stands for a possible interaction of genes (x_i, x_j)[1]. This is shown in pseudo-code in Figure 5.1. Such a matrix has been previously used to store linkage of genes of a GA in [97] and [96].

The discussed linkage neighborhood approach requires only such an interaction matrix as input, so that various linkage detection mechanisms from the literature can be applied, as long as they produce M. It should be noted, however, that the choice of the linkage criterion is of fundamental importance for the creation of the problem structure. A non-linear criterion can retrieve different interaction answers than a non-monotonicity check [50] and furthermore, a non-linear dependency as a sufficient basis for non-separability of genes might not always correspond to the right choice with regards to optimization [58].

Based on the calculated matrix M through the algorithm in Figure 5.1, the direct neighborhood, which is a structure that keeps a record of which genes are directly (via M) interacting, is calculated. This bears a strong resemblance to the LN model [6] where each gene is assigned a set of genes that is has direct dependencies with. The algorithm is given in Figure 5.2.

5.1.2 1st, 2nd Neighborhoods and Beyond

Having created the direct neighborhood, 1st, 2nd, and further neighborhoods can be now calculated iteratively. These neighborhoods are incrementally created by an iterative expansive approach to increase the scope of interactions that each gene participates in through the

[1] This matrix is a symmetric matrix.

```
1: x ← current individual
2: M ← empty n × n 2D-matrix
3:
4: for each genepair x_i, x_j ∈ x do
5:    M[i][j] = M[j][i] ← dependency(x_i, x_j) ▷ for instance LINC-R or LIMD check
6: end for
```

Figure 5.1: Pairwise dependency calculation.

```
1: M ← n × n 2D-matrix with pairwise dependencies
2: DN ← empty direct neighborhood
3: ϵ ← small value close to 0, i.e. 0.000001
4:
5: for each genepair x_i, x_j ∈ x do
6:    if M[i][j] > ϵ then
7:       DN_{x_i} ← x_j
8:       DN_{x_j} ← x_i
9:    end if
10: end for
```

Figure 5.2: Direct Neighborhood preparation.

idea of transitivity. The 1st neighborhood is a direct mapping from the direct neighborhood adopting the LN notation, in which the first element of a list corresponds to the gene under investigation and the remaining elements the genes that it interacts with. For instance, if x_i is (based on M) interacting with x_a, x_b, x_c, x_d, the notation of x_i's 1st neighborhood, $1stN_{x_i}$, would be: $[i, a, b, c, d]$. The algorithm that generates the 1st neighborhood is presented in Figure 5.3. The 2nd neighborhood is iteratively generated based on the input from the 1st neighborhood. The algorithm for this is described in pseudo code in Figure 5.4. A 2nd neighborhood of gene x_i contains all genes from its 1st neighborhood plus all genes in which x_i is part of their 1st neighborhood. For example, if x_i's 1st neighborhood, $1stN_{x_i}$, is $[i, a, b, c, d]$, and x_a's 1st neighborhood, $1stN_{x_a}$, is $[a, b, i, j]$, then x_i's 2nd neighborhood, $2ndN_{x_i}$, is $[i, a, b, c, d, j]$.

This procedure can be iteratively continued, to generate a k-neighborhood from a (k-1) neighborhood using the algorithm in Figure 5.4 and substituting the 2nd neighborhood with a

```
1: DN ← direct neighborhood
2: 1stN ← empty 1st neighborhood
3:
4: for each gene x_i ∈ DN do
5:    1stN_{x_i} ← x_i ▷ the gene itself
6:    for each gene x_j ∈ DN_{x_i} do
7:       1stN_{x_i} ← x_j
8:    end for
9: end for
```

Figure 5.3: 1st neighborhood calculation.

```
1: DN ← direct neighborhood
2: 1stN ← 1st neighborhood
3: 2ndN ← empty 2nd neighborhood
4:
5: for each gene x_i ∈ DN do
6:    2ndN_{x_i} ← x_i ▷ the gene itself
7:    for each group 1stN_{x_j} do
8:       if x_i ∈ 1stN_{x_j} then
9:          2ndN_{x_i} ← 1stN_{x_j} ▷ add all genes of this group to x_i's 2nd neighborhood
10:      end if
11:   end for
12: end for
```

Figure 5.4: 2nd neighborhood calculation.

k-neighborhood and the 1st neighborhood with the (k-1) neighborhood. If there is no change (no genes are added to any neighborhoods) in such step, the calculation of neighborhoods can be stopped. At this point, this last neighborhood will then contain a representational model that corresponds to the MPM. In a way, the iterative linkage neighborhoods approach can be viewed as incrementally approaching the MPM from direct, pairwise dependencies through transitivity and allowing more and more overlapping as the neighborhoods grow.

At this step, two views on these neighborhoods can be adopted, a list-view and a set-view.

- **List-view**: Each k-neighborhood is a list of integer lists, where the number of these integer lists is equal to number of genes in the chromosome. Each such list adopts the

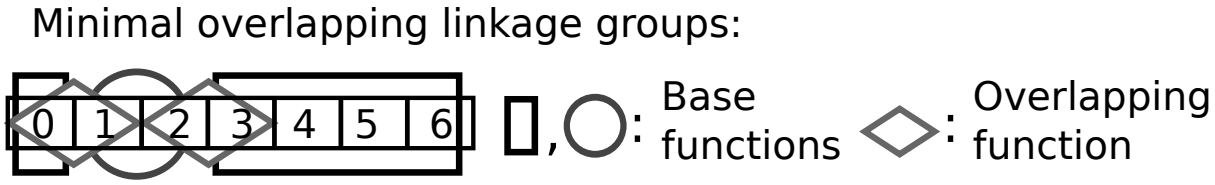

Figure 5.5: Example structure of linkage.

Table 5.1: Dependency Matrix Example.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6
x_0	-	1	0	0	0	0	0
x_1	1	-	1	0	0	0	0
x_2	0	1	-	1	0	0	0
x_3	0	0	1	-	1	1	1
x_4	0	0	0	1	-	1	1
x_5	0	0	0	1	1	-	1
x_6	0	0	0	1	1	1	-

LN notation of a neighborhood, such that $[i, ...]$ defines the neighborhood of x_i and lists all genes that belong to it.

- **Set-view**: In the set view, each k-neighborhood is a set of integer sets. This can be achieved by applying set properties on a k-neighborhood under a list view.

To provide an example with seven genes and overlapping among linkage neighborhoods, Table 5.1 shows M for a function that is visualized in Figure 5.5. Table 5.2 then gives the 1st neighborhood, Table 5.3 the 2nd neighborhood and finally Table 5.4 the final 3rd neighborhood, all under the list view. Generating a 4th neighborhood would not provide any changes compared to the 3rd neighborhood, hence this is where the calculation stops. If a pairwise linkage detection checker would only apply the MPM to represent linkage, the result would be a list such as in Table 5.4, where every gene is linked to every other gene. Table 5.5 gives a view on the neighborhoods as sets, here, the 3rd neighborhood would not include any new set ($[0, 1, 2, 3, 4, 5, 6]$ is already included in the 2nd neighborhood), so the calculation stops.

Table 5.2: 1st Neighborhood in List-view Example.

	1st Neighborhood in List-view
x_0	$[0, 1]$
x_1	$[1, 0, 2]$
x_2	$[2, 1, 3]$
x_3	$[3, 2, 4, 5, 6]$
x_4	$[4, 3, 5, 6]$
x_5	$[5, 3, 4, 6]$
x_6	$[6, 3, 4, 5]$

Table 5.3: 2nd Neighborhood in List-view Example.

	2nd Neighborhood in List-view
x_0	$[0, 1, 2]$
x_1	$[1, 0, 2, 3]$
x_2	$[2, 0, 1, 3, 4, 5, 6]$
x_3	$[3, 1, 2, 4, 5, 6]$
x_4	$[4, 2, 3, 5, 6]$
x_5	$[5, 2, 3, 4, 6]$
x_6	$[6, 2, 3, 4, 5]$

Table 5.4: 3rd Neighborhood in List-view Example.

	3rd Neighborhood in List-view
x_0	$[0, 1, 2, 3, 4, 5, 6]$
x_1	$[1, 0, 2, 3, 4, 5, 6]$
x_2	$[2, 0, 1, 3, 4, 5, 6]$
x_3	$[3, 0, 1, 2, 4, 5, 6]$
x_4	$[4, 0, 1, 2, 3, 5, 6]$
x_5	$[5, 0, 1, 2, 3, 4, 6]$
x_6	$[6, 0, 1, 2, 3, 4, 5]$

Table 5.5: Neighborhoods in the Set-view Example.

	Neighborhoods in Set-view
1st Neighborhood	$[0, 1]$, $[1, 0, 2]$, $[2, 1, 3]$, $[3, 2, 4, 5, 6]$, $[4, 3, 5, 6]$
2nd Neighborhood	$[0, 1, 2]$, $[1, 0, 2, 3]$, $[0, 1, 2, 3, 4, 5, 6]$, $[1, 2, 3, 4, 5, 6]$, $[2, 3, 4, 5, 6]$

5.2 Identification of Crossover Masks

Crossover masks partition a chromosome into disjoint subsets. One of the simplest examples of a crossover mask is the one-point crossover as introduced in Chapter 2. This operator defines a mask that separates a chromosome into two parts, before and after the crossover point. Crossover masks have been shown to be useful in linkage exploitation with linkage trees [23], MPM or LN representations [86], [85]. The idea goes back to the realization that it is beneficial to keep genes that have meaningful interactions together during optimization, such as with the BB-wise crossover [97]. A possible draft of determining such valuable partitionings that is inspired by these crossover mask implementations and can be performed through the iterative neighborhoods concept is as follows:

- **Iterative Linkage Neighborhoods Conjunction**: This approach recombines the iterative linkage neighborhoods under the set-view. Figure 5.6 presents the algorithm for this and in Figure 5.7, a visualization of the construction of a partitioning with this method is shown. Every time a substructure is picked, as shown in Line 4 of Figure 5.7, its genes are removed from the $AllN$ set, this is visible in Line 5. In Line 6, the chosen substructure is added as disjoint subset to the partitioning. The rationale behind this method is to generate small, linkage preserving partitionings of the chromosome in scenarios where distinct linkage groups with some overlapping among them exist that would be all merged in the MPM model, as demonstrated in [85]. Furthermore, genes that do not interact with each other in any neighborhood do not appear within the same group in the crossover mask, while the optimizer is given varying possibilities of subsets of genes that have at least some degree of interactions as explorative opportunities.

1: $AllN \leftarrow$ Set of all neighborhood substructures $\triangleright$ 1st, 2nd, ...
2: $partitioning \leftarrow \emptyset$ $\triangleright$ empty partitioning set
3: **while** $AllN$ includes at least one gene **do**
4: $\quad substructure \leftarrow randomPick(AllN)$ $\triangleright$ randomly pick a substructure
5: $\quad remove(AllN, substructure)$ $\triangleright$ remove all genes of substructure from $AllN$
6: $\quad partitioning \leftarrow substructure$ $\triangleright$ add substructure to partitioning
7: **end while**

Figure 5.6: Iterative Linkage Neighborhoods Conjunction.

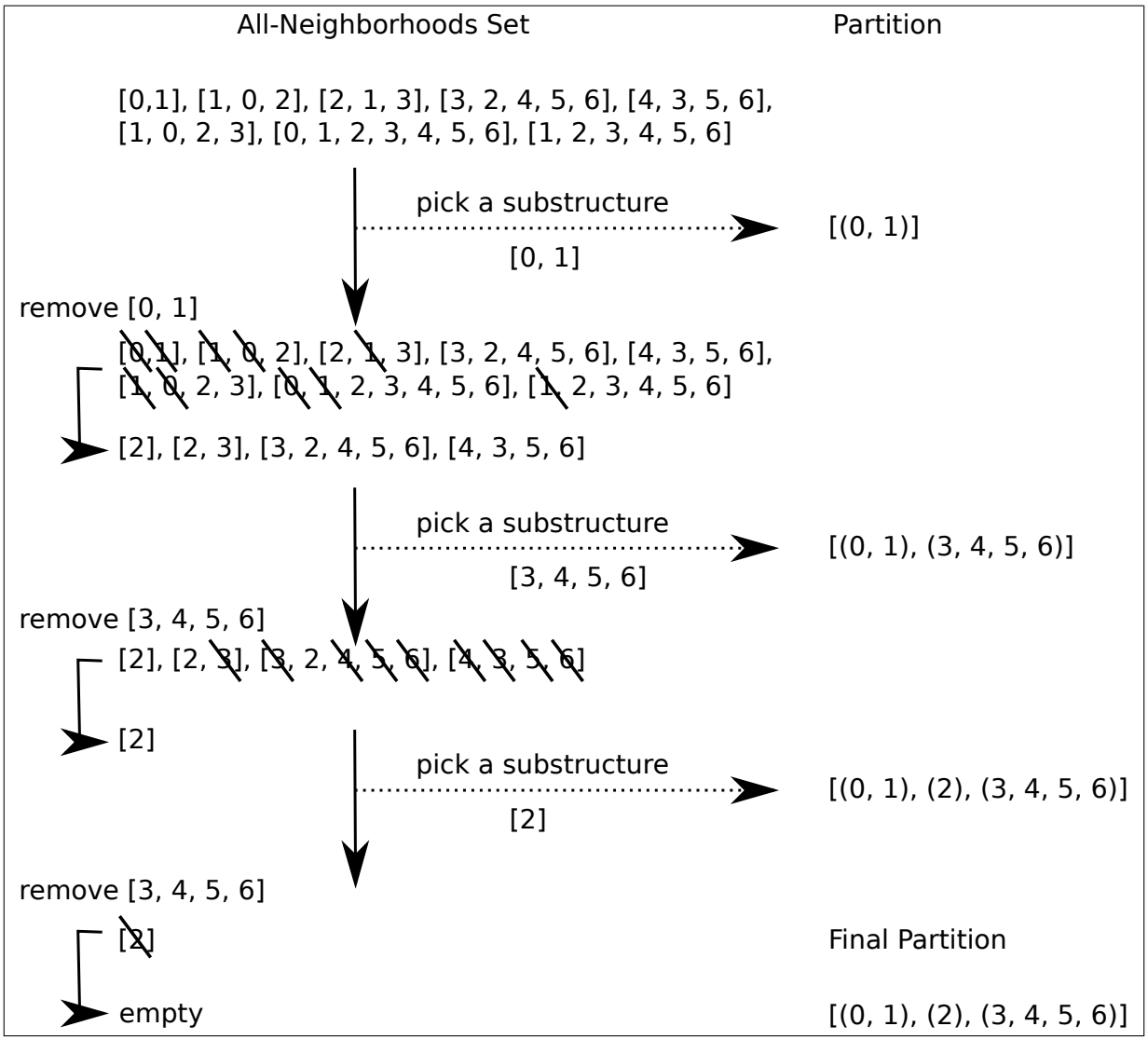

Figure 5.7: Iterative Linkage Neighborhoods Conjunction example.

5.3 Problem Structure Matrix

Linkage and sensitivity can both provide important insights about the problems tackled by EAs. It is proposed that a meaningful representation of such problem structure measures can be achieved through a joint combination. This Section presents the first idea of a problem structure matrix which combines information about the interaction of genes, expressed as linkage with a distance metric, as well as their contribution towards the fitness function, defined as gene sensitivity. The proposed matrix can be seen as an extension of the idea of the dependency structure matrix [97], and could be envisioned to allow numerical ways of comparing substructures, which are sets of genes, and to be subsequently used to discover valuable crossover masks.

5.3.1 Generating the Problem Structure Matrix

The work in [37] introduces binary test problems with *iff* constraints (if and only if) that are defined through an a priori filled out adjacency matrix. Weights (i, j) in this matrix correspond to the linkage between genes x_i and x_j. The four matrices shown follow a pattern of having weights such as $\frac{1}{2}, \frac{1}{8}, \frac{1}{32}$ or $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$, where the weaker weights are obtained by dividing a weight by a constant; such an idea for a distance metric allows to model weaker, transitive interactions. Furthermore, [28] and [62] advance the notion of a distance metric as a soft measure of interactions, where variables separated by greater distance in a graph have a weaker interaction.

In [97] and [96], the concept of a dependency structure matrix is applied to represent interactions for GAs. However, this matrix excludes the information about sensitivity of genes and only specifies existing linkage found by a linkage checking algorithm. This stands in opposition to the findings in [69] and [77] that the main effect plays an important role in understanding the problem and the additional findings in [10], where detection and exploitation of sensitivity with regards to GAs results in notable improvements of performance. A

Table 5.6: PSM - Filling in the sensitivities.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6
x_0	$S_{\mathbf{x}_0}$						
x_1		$S_{\mathbf{x}_1}$					
x_2			$S_{\mathbf{x}_2}$				
x_3				$S_{\mathbf{x}_3}$			
x_4					$S_{\mathbf{x}_4}$		
x_5						$S_{\mathbf{x}_5}$	
x_6							$S_{\mathbf{x}_6}$

novel comparison of three machine learning based sensitivity detection methods is presented in this dissertation which adds to the variety of possible detection approaches.

The proposed representation, called the problem structure matrix PSM, is designed to contain both sensitivity and linkage information, such as given by the iterative neighborhoods method. However other ways of including the interaction information gained from model builds presented in Chapter 4 are also possible.

A concept of the PSM will be created step-by-step based on the structure in Figure 5.5 and its iterative neighborhoods to provide insight on this idea[2].

In a first step, the sensitivity of each gene $S_{\mathbf{x}_i}$ is inserted in its cells x_i, x_i of the PSM as shown in Table 5.6. Ideas for a numerical score include the obtained ranks from sensitivity detection methods with a transformation or scaling function (for instance linear) applied or raw weights of the strength of contribution if extractable from the sensitivity detection method, such as through the ANOVA [10].

In the next steps, the linkage information can be entered. In this example, it is taken from the iteratively generated neighborhoods (using the list-view) from the problem displayed in Figure 5.5, and Table 5.7 shows the entries of all 1st neighborhood interactions.

For any subsequent neighborhood level, the amount of interaction could be reduced through an applied function to give an emphasis on the importance of closer interactions.

[2]Being a symmetric matrix, for brevity's sake only the upper half is displayed in the examples.

Table 5.7: PSM - Filling in the 1st neighborhood.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6
x_0	$S_{\mathbf{x}_0}$	$1stN$					
x_1		$S_{\mathbf{x}_1}$	$1stN$				
x_2			$S_{\mathbf{x}_2}$	$1stN$			
x_3				$S_{\mathbf{x}_3}$	$1stN$	$1stN$	$1stN$
x_4					$S_{\mathbf{x}_4}$	$1stN$	$1stN$
x_5						$S_{\mathbf{x}_5}$	$1stN$
x_6							$S_{\mathbf{x}_6}$

Table 5.8: PSM - Filling in the 2nd neighborhood.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6
x_0	$S_{\mathbf{x}_0}$	$1stN$	$2ndN$				
x_1		$S_{\mathbf{x}_1}$	$1stN$	$2ndN$			
x_2			$S_{\mathbf{x}_2}$	$1stN$	$2ndN$	$2ndN$	$2ndN$
x_3				$S_{\mathbf{x}_3}$	$1stN$	$1stN$	$1stN$
x_4					$S_{\mathbf{x}_4}$	$1stN$	$1stN$
x_5						$S_{\mathbf{x}_5}$	$1stN$
x_6							$S_{\mathbf{x}_6}$

Such a distance metric corresponds to ideas in [28], where in the context of hBOA, interaction strengths of binary decision variables that decrease with increasing distance in a graph are investigated and also in [62], where a decreasing weight added to the edges (which stand for interactions) in the graph is considered. Table 5.8 shows how the entries from the 2nd neighborhoods are incorporated.

Finally, Table 5.9 gives the final PSM for this problem by including the information from the 3rd neighborhoods.

5.3.2 Discussion of Problem Structure Matrix Analyses

The PSM contains the joint measures of sensitivity and linkage about a problem. This can be denoted by *structural partaking* of sets of genes in relation to the overall problem. There are scoring and distance metric techniques that have been considered before for problem

Table 5.9: PSM - Filling in the 3rd neighborhood.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6
x_0	$S_{\mathbf{x}_0}$	$1stN$	$2ndN$	$3rdN$	$3rdN$	$3rdN$	$3rdN$
x_1		$S_{\mathbf{x}_1}$	$1stN$	$2ndN$	$3rdN$	$3rdN$	$3rdN$
x_2			$S_{\mathbf{x}_2}$	$1stN$	$2ndN$	$2ndN$	$2ndN$
x_3				$S_{\mathbf{x}_3}$	$1stN$	$1stN$	$1stN$
x_4					$S_{\mathbf{x}_4}$	$1stN$	$1stN$
x_5						$S_{\mathbf{x}_5}$	$1stN$
x_6							$S_{\mathbf{x}_6}$

matrices such as [62] and [15]. In [37], the concept of modularity is discussed, which refers to subsets of nodes which have a larger linkage density with each other than with nodes outside of this subset, and one way to apply structural partaking can be to view it as the extension of it with added information about the strength of contribution of genes towards the fitness. If a quantitative measure of subsets can be given, a next step can be to apply such a scoring to possible subset combinations with the goal to obtain partitionings with a high measure of structural partaking.

The number of all possible partitionings of a chromosome can be obtained through the n-th Bell number [72] (B_n), where n is the dimensionality of the chromosome. Equation 5.1 shows a recursive formula to calculate Bell numbers.

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \tag{5.1}$$

Looking at the first Bell numbers $1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, ...$ it becomes clear that an exhaustive search for the best (in terms of maximized structural partaking) partitions is impractical. However, if a scoring measure is defined, this task can be clearly stated as an optimization (maximization) problem that an EC method or any other suitable optimizer can tackle. The problem can be therefore formulated as finding the partition(s) that maximize the structural partaking. In [96], a similar problem (using a

dependency structure matrix that only represents interactions) is viewed as a clustering task and various techniques to cluster into meaningful groups are presented. A model building technique with clustering called clusterMI [15] has also been applied to find best partitionings, inspired by the ECGA and the DSMGA. Furthermore, an idea to assess modularity through Q values is discussed in [37]. However, the dimensionality of the problem has to be taken into account, since it involves a whole subsidiary optimization run that can involve additional computational overhead.

Finally, a reasonable heuristic to cut down the search time in the PSM for these important partitions can be to include substructures from the iteratively generated neighborhoods as starting points of the search. One direct application of these valuable partitionings in terms of an EC optimizer is that of a crossover mask, or, in broader terms suitable for other optimization techniques, a dimensionality reduction to apply improvements on only one substructure at a time, while the other substructures are fixed.

Chapter 6

Machine Learning aided Discovery and Exploitation of Problem Structure

This Chapter introduces methods to discover the problem structure measures of linkage and sensitivity through machine learning and further proposes a method to exploit detected interaction groups through crossover masks and specialized linkage-aware operators. Four such surrogate-assisted informed crossover operators are defined that are designed to work in real-valued evolutionary optimization; and their implementations and rationales are presented. Furthermore, a novel surrogate-assisted perturbation-check for linkage, SA-LINC-R, is described and its experimental setup is given. Finally, three machine learning methods that detect sensitivity of genes are introduced and their experimental configuration is shown.

6.1 Operators

One immediate application of learned partitionings through detected linkage are crossover masks for recombination operators. In the following, four such recombination operators are introduced, they have been previously published in [57] to be employed with a partitioning under the MPM generated through linkage learning.

6.1.1 Informed Surrogate-Assistance

Informed operators were introduced in [67]. Their idea is to create a pool of promising offspring (as opposed to only one or two with conventional operators for GAs), and then rank every offspring in this pool according to its fitness to finally only choose the best one or two individuals to be taken as true offspring. To avoid the computational burden on extra fitness evaluations (each offspring needs to have its own fitness associated with it), informed operators implement surrogate models to estimate the fitness of their potential offspring. By doing so, no calls to the true fitness function are necessary, while the operators are still able to return offspring that are expected to be superior or at least as good as other members from the offspring pool. Whenever the domain provides enough training examples to allow such a fitness approximator to be created, it is strongly recommended to do so, as it can save evaluation time and provide significant improvements to the final result quality [67]. The following four crossover operators are all implemented as informed operators (and therefore can use a surrogate model), but they are also capable of only producing one or two offspring in case it is not possible or desired to apply a fitness approximator.

6.1.2 Informed Linkage Group Line Crossover

The informed linkage group line (ILGL) crossover is based on the line and the double line crossover methods described in [64]. The line crossover connects the two parents with a line, extends this line from both sides by twice its length (the length of the segment between the two parents) and picks a point with uniform probability from its total length. The reason behind this is that many Engineering design fitness functions exhibit thin but good (in terms of fitness) regions of ellipsoidal space. Those hyper-ellipsoids [64] are targeted by this operator. With explicit problem structure information available that is expressed as a crossover mask gained from a linkage learning algorithm, the ILGL crossover improves upon this by applying separate line crossovers according to the provided partitioning knowledge.

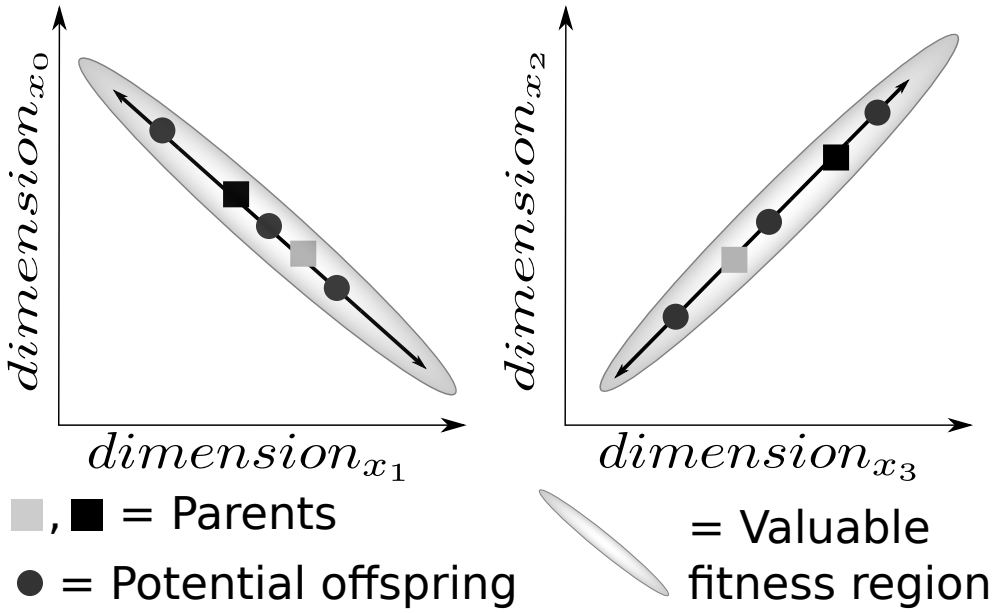

Figure 6.1: Example of the ILGL crossover with two parents with genes x_1 and x_2 in one problem structure group (one partition) and genes x_3 and x_4 in another partition. If the chromosome consists of four genes, the partitioning here is $[(0, 1), (2, 3)]$. Potential offspring on the extended lines are created inside the ellipsoidal regions of higher fitness, each group of genes exploring their own ellipsoidal region.

The algorithm for the ILGL crossover is given in Figure 6.2. Each group (single genes form a unique additional group) receives a ratio r drawn uniformly from $[-2, 3]$, as described in Figure 6.2. This means that while the connected line between the parents extends the connection length for every group by two times, each group will be steered independently by a different r, keeping linked parameters together. If a newly calculated gene value is outside of its lower or upper bound, a repair function corrects this (by setting it to the lower or upper bound, respectively). This operator is particularly helpful in fitness regions of ellipsoidal space as shown in Figure 6.1. The creation of an offspring from its parents can be seen in an illustrative example in Figure 6.3.

1: $p1, p2 \leftarrow parentSelection(currentPopulation)$
2: $offspring \leftarrow$ empty offspring
3: $partitioning \leftarrow$ structural partition $\triangleright$ based on linkage information
4: **for all** $group \in partitioning$ **do**
5: $\quad r \leftarrow getRandom(-2, 3)$
6: $\quad$ **for all** $gene \in group$ **do**
7: $\quad\quad newGene = r * p1_{gene} + (1 - r) * p2_{gene}$
8: $\quad\quad offspring_{gene} \leftarrow newGene$
9: $\quad$ **end for**
10: **end for**

Figure 6.2: The ILGL crossover algorithm.

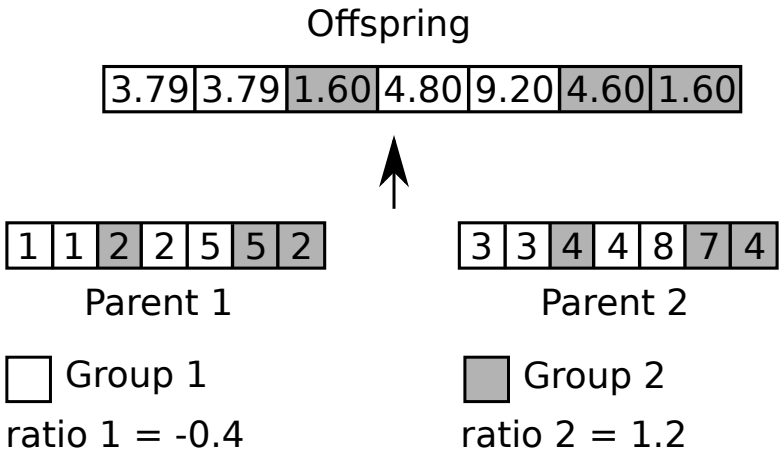

Figure 6.3: Example of calculating an offspring with the ILGL crossover.

6.1.3 Informed Guided Linkage Group Crossover

The informed guided linkage group (IGLG) crossover is based on the guided crossover [66]. It has a greedy nature and aims at exploiting the search landscape with the goal to converge to an optimum that the population is close to.

The first parent $parent1$ is picked through the regular binary tournament selection. To get the second parent $parent2$, a value $mutualFitness(x, parent1)$ is computed for every other individual $\mathbf{x}$ in the current population. It serves as a rough approximation of the gradient and is calculated as follows:

$$mutualFitness(x, y) = \frac{((fitness(x) - fitness(y))^2}{euclideanDistance(x, y)^2} \tag{6.1}$$

The individual that has the highest mutual fitness with $parent1$ is chosen as $parent2$, and if $parent2$ has a higher fitness than $parent1$, they are swapped. As in [64], during the last 5% of the evolutionary iterations, the best individual of the generation is taken to be $parent2$ (except if $parent1$ is the best, in which case the second best individual is chosen). Also, with Equation 6.2 the variable $stepSize$ is calculated using the maximal number of generations and the current generation number. $stepSize$ will decrease as the generations progress.

$$stepSize = 0.75 * \frac{maxGens. - currentGen.}{maxGens.} + 0.25 \tag{6.2}$$

Finally, for every gene pair a ratio variable r is calculated by picking a value uniformly randomly from $[1 - 0.2 * stepSize, 1 + stepSize]$. The algorithm for IGLG is given in Figure 6.4. Each group is guided by its own ratio variable r (similar to the ILGL crossover) so that genes that form meaningful groups can exploit the search space in the same direction, and a repair function ensures that upper and lower bounds of gene values are maintained. The creation of an offspring with this operator is shown in an example in Figure 6.5.

1: $p1, p2 \leftarrow guidedParentSelection(currentPopulation)$
2: $stepSize \leftarrow calculateStepSize()$
3: $offspring \leftarrow$ empty offspring
4: $partitioning \leftarrow$ structural partition $\triangleright$ based on linkage information
5: **for all** $group \in partitioning$ **do**
6: $\quad r \leftarrow getRandom(1 - 0.2 * stepSize, 1 + stepSize)$
7: $\quad$ **for all** $gene \in group$ **do**
8: $\quad\quad newGene = r * p1_{gene} + (1 - r) * p2_{gene}$
9: $\quad\quad offspring_{gene} \leftarrow newGene$
10: $\quad$ **end for**
11: **end for**

Figure 6.4: The IGLG crossover algorithm.

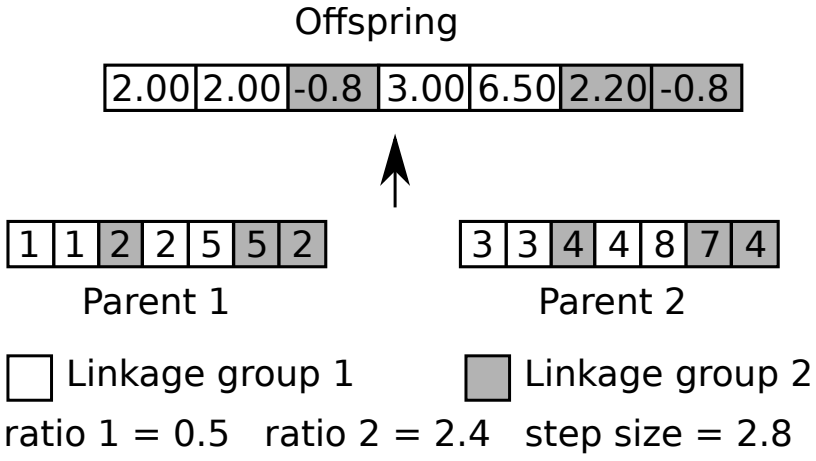

Figure 6.5: Example of calculating an offspring with the IGLG crossover.

6.1.4 Informed Linkage Inter Group Crossover

The motivation behind informed linkage inter group (ILIG) crossover goes back to the idea that mixing of above average fit subcomponents of solutions (building blocks) leads to successful individuals, and genes forming such valuable subcomponents (based on a problem structure metric) need to be kept together during optimization [52]. This has been also formulated as "Linkage learning in genetic algorithms (GAs) is the identification of building blocks to be conserved under crossover" [27]. This crossover therefore functions essentially as the BB-wise [97] recombination operator and has been named ILIG with regards to the usage of crossover masks gained from a linkage detection mechanism in combination with employing a surrogate-model to create and rank a pool of offspring. The ILIG crossover applies the rationale of the uniform crossover with problem structure groups of interacting genes as basic units (to get from the same parent). The motivation is to keep discovered substructures from being disrupted while recombining parent groups with the goal to discover better recombinations. This operator has been used in the past in early linkage learning approaches and is a natural extension of basic GA recombinations. An algorithm for this operator is given in Figure 6.6. An illustration is also provided in Figure 6.7. The group of single genes (each forming its own substructure) is treated by a regular uniform crossover, as there is no further information available about which of these genes should be grouped together during recombination.

6.1.5 Informed Linkage Inter-Intra Group Crossover

The informed linkage inter-intra group (ILIIG) crossover consists of a slight modification of the ILIG operator. It is a two-stage crossover. During the first stage, it mixes substructures of two parents in their offspring just like the ILIG crossover. In the second stage, it uses a shuffling of the gene values inside the substructures after they have been recombined into a new offspring. This allows for an intra group gene exchange, while still keeping the values

1: $p1, p2 \leftarrow parentSelection(currentPopulation)$
2: $offspring1 \leftarrow$ empty offspring
3: $offspring2 \leftarrow$ empty offspring
4: $partitioning \leftarrow$ structural partition $\triangleright$ based on linkage information
5: **for all** $group \in partitioning$ **do**
6: $r \leftarrow getRandomBoolean()$
7: **if** $r == true$ **then**
8: $offspring1_{group} \leftarrow p1_{group}$
9: $offspring2_{group} \leftarrow p2_{group}$
10: **else**
11: $offspring1_{group} \leftarrow p2_{group}$
12: $offspring2_{group} \leftarrow p1_{group}$
13: **end if**
14: **end for**

Figure 6.6: The ILIG crossover algorithm.

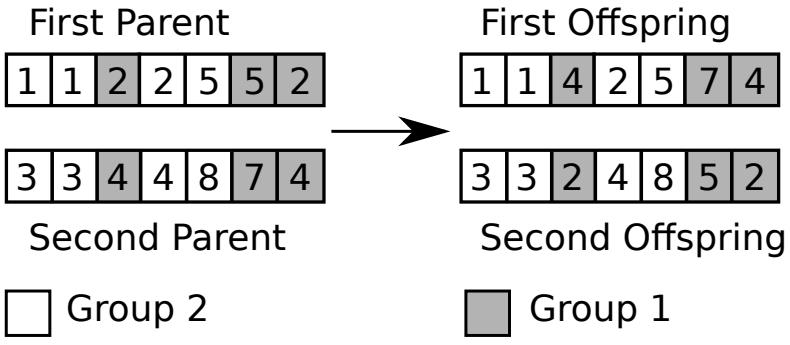

Figure 6.7: Example of calculating an offspring with the ILIG crossover.

```
1: p1, p2 ← parentSelection(currentPopulation)
2: offspring1 ← empty offspring
3: offspring2 ← empty offspring
4: partitioning ← structural partition ▷ based on the amount of structural partaking
5: for all group ∈ partitioning do
6:    r ← getRandomBoolean()
7:    if r == true then
8:       offspring1_group ← shuffle(p1_group)
9:       offspring2_group ← shuffle(p2_group)
10:   else
11:      offspring1_group ← shuffle(p2_group)
12:      offspring2_group ← shuffle(p1_group)
13:   end if
14: end for
```

Figure 6.8: The ILIIG crossover algorithm.

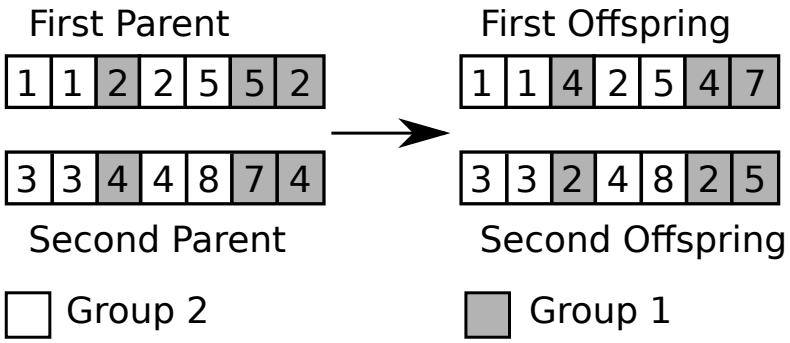

Figure 6.9: Example of calculating an offspring with the ILIIG crossover.

of a linkage group together. Figure 6.9 depicts this in an example. A repair function similar to the one used in the ILGL crossover makes sure that no upper or lower boundaries are violated. The algorithm is given in Figure 6.8. As with the ILIG crossover, the uniform crossover is applied to single gene groups, giving these genes the maximal possibility to mix. It should be noted that this crossover is very disruptive in terms of allele settings. Therefore, if positions of optima differ strongly and are spread out, it should not be applied.

6.2 A Surrogate-Assisted and Informed Linkage Exploiting Genetic Algorithm

In [57], the surrogate-assisted and informed linkage exploiting genetic algorithm (SAILEGA) is presented and will be outlined in the following pages. It combines the three fields of linkage learning through LINC-R, surrogate modeling through the machine learning algorithms from Weka[1] [24] and informed operators. The operators are designed in a modular way that makes it easy to re-implement them without the informed or the surrogate-assisted components, making them potential candidates for other linkage-aware frameworks.

SAILEGA's[2] structure and general work flow is depicted in Figure 6.10. This Section presents its steps in detail.

6.2.1 Clustering

Step 1 in Figure 6.10 consists of copying the current generation of individuals into the population storage. This storage has the properties of a set and holds all unique encountered individuals so far. Every generation, the population storage is clustered with X-means [63]. X-means clustering is based on the Bayesian Information Criterion and is a computationally fast extension of the K-means algorithm. It allows to specify a lower and upper bound of the number of clusters, and computes the clustering with the ideal cluster size according to its metric and a maximum number of preset iterations. SAILEGA sets these bounds as follows:

1. The lower bound is fixed as two, defining a minimum of two clusters.

2. To calculate the unknown parameters with a second-order polynomial approximator, Equation 6.3 shows the minimum necessary individuals i_{basic} [34], where n is the chromosome length. Following the advice of [65], it is slightly increased to $i_{increased}$ as

[1]The Weka software can be downloaded from: http://www.cs.waikato.ac.nz/ml/weka/

[2]The algorithm is build on top of the Watchmaker Framework 0.7.1, which can be obtained from: http://watchmaker.uncommons.org/

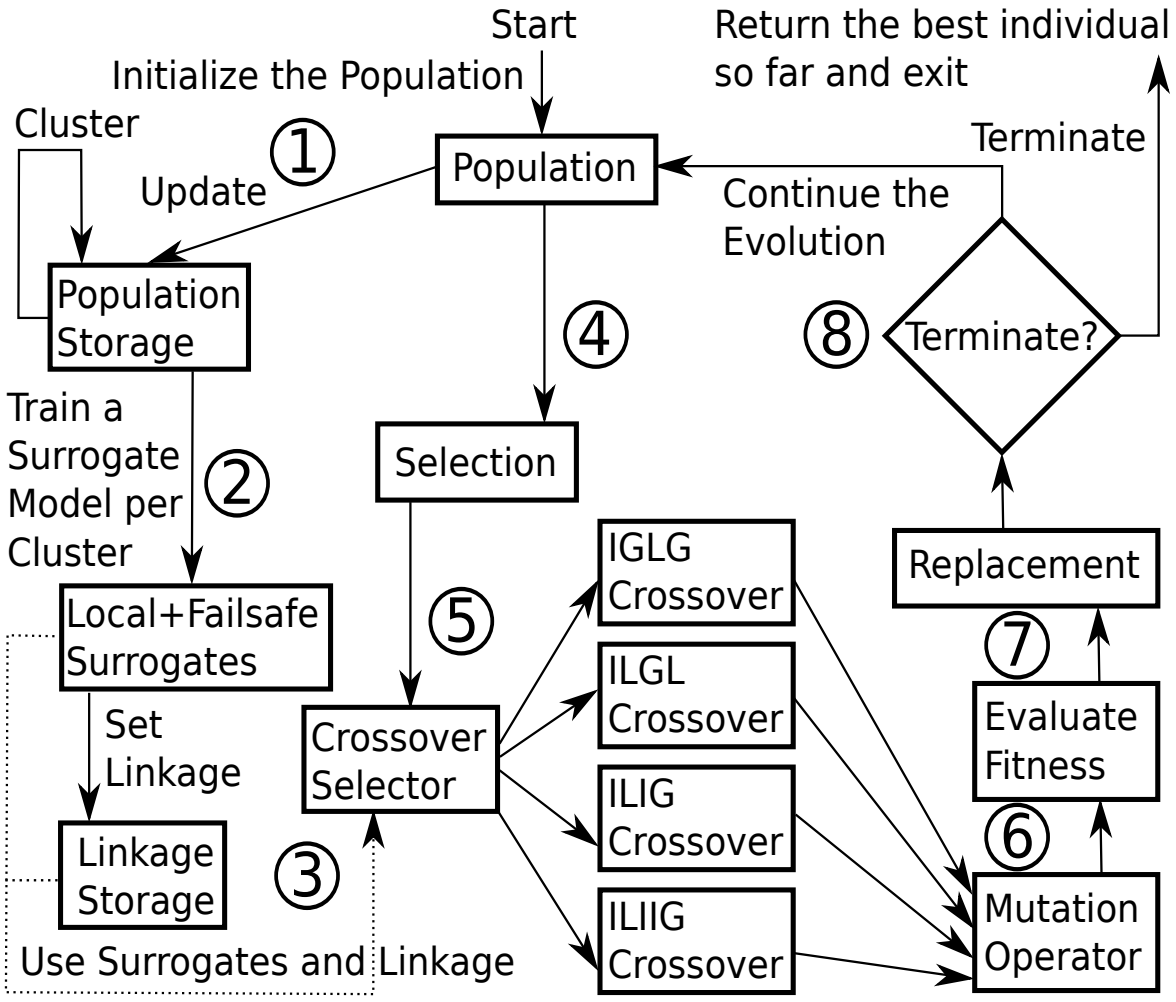

Figure 6.10: Overview of SAILEGA's structure and its work flow.

shown in Equation 6.4 as minimum size. This yields Equation 6.5 with $popStore_{size}$ as the number of individuals in the population storage and *upper* as upper bound. The rationale behind this is to gain a reasonable size for each cluster even if the upper bound is chosen and a similar size for all clusters is assumed.

$$i_{basic} = \frac{(n+1) * (n+2)}{2} \tag{6.3}$$

$$i_{increased} = (n+1) * (n+2) * 0.75 \tag{6.4}$$

$$upper = \frac{popStore_{size}}{i_{increased}} \tag{6.5}$$

6.2.2 Surrogate Models

A study of fitness approximation in EC [34] suggests that there are benefits to using local surrogate models instead of a single global one. While this can be taken to an extreme by creating a custom surrogate model for each individual [70] based on its i_{basic} k-nearest neighbors from Equation 6.3, a middle ground is chosen by creating surrogate models for each cluster given by X-means, that is represented by at least one member of the current population. Clusters without representation within the current population do not become eligible for surrogate training which cuts down computation time as only useful surrogates will be created. The second rationale for such clustering is that, especially in multi-objective problems, regions with good fitness might be located far apart from each other, and therefore having a dedicated surrogate model for each of those regions where parts of the populations gather can add to the model's prediction accuracy.

To make this happen, in Step 2 of Figure 6.10 first a mapping between cluster numbers and sets of individuals eligible to become training sets for surrogate models is created. The algorithm for this is given is Figure 6.11. Then, a mapping between the eligible instance sets, accessible via their cluster numbers, and the corresponding surrogate models is created. In case there is at least one individual in the population that is a member of a cluster that did

1: $pop \leftarrow currentPopulation$
2: $store \leftarrow populationStore$
3: $map_{CI} \leftarrow$ map of clusternumber to instances
4: $clusterSet \leftarrow xMeans(store, lower, upper)$
5: **for all** clusters $c \in clusterSet$ **do**
6: **if** $size(c) \geq i_{increased}$ **then**
7: **if** $\exists\ x \in pop\ with\ c_{number}$ **then**
8: $map_{CI} \leftarrow (c_{number}, y_{c_{number}})\ \forall\ y_{c_{number}} \in store$
9: **end if**
10: **end if**
11: **end for**

Figure 6.11: Cluster number to sets of instances mapping.

1: $pop \leftarrow currentPopulation$
2: $store \leftarrow populationStore$
3: $map_{CI} \leftarrow$ map of clusternumber to instances
4: $map_{CS} \leftarrow$ map of clusternumber to surrogate models
5: **for all** $c_{number} \in map_{CI}$ **do**
6: $instances \leftarrow y_{c_{number}} \in map_{CI}$
7: $map_{CS} \leftarrow (c_{number}, buildSurrogate(instances))$
8: **end for**
9: **if** $\exists\ x \in pop\ with\ x_{c_{number}} \notin map_{CS}$ **then**
10: $instances \leftarrow y_{failsafe} \in map_{CI}$
11: $buildFailSafe(instances)$
12: **end if**

Figure 6.12: Cluster number to surrogate model mapping.

not receive a surrogate, a failsafe surrogate trained on the last two generations is constructed. This algorithm is depicted in Figure 6.12.

SAILEGA uses Weka implementations of support vector machines for regression [79] with the Pearson VII function-based universal kernel (PUK) [90] as surrogate model. The PUK allows for robust surrogate models without having to specify or test any kernel exponents or kernel types, making it suitable for creating accurate fitness approximators without any prior knowledge of the fitness landscape.

6.2.3 Crossover Operators and Selector

Four recombination operators have been implemented in SAILEGA, among which two (informed linkage group line crossover the informed guided linkage group crossover) are extensions of informed but linkage unaware crossovers from GADO [64], and they have been described previously in this Chapter. In the absence of any dependencies among genes (none of the genes is linked), SAILEGA's operators automatically work like their linkage unaware counterparts, making them applicable to a wide range of fitness functions. To choose a crossover operator for a mating between two parents, Step 5 of Figure 6.10 includes a crossover selector. The idea of this crossover selector is taken from [64]. It maintains a variable *elapsedGens* that expresses the proportion of generations elapsed so far; its value is 0 at the beginning of the evolutionary run and 1 at the end. Also, a variable $guidedFactor$ that is fixed to 0.25 is defined. To choose a crossover operator for two parents, the crossover selector proceeds as follows:

1. With a probability of $elapsedGens * guidedFactor$, the informed guided linkage group crossover is chosen. As the evolutionary search progresses, the probability of using this operator increases.

2. Otherwise, with a probability of 0.5, the informed linkage group line crossover is selected, or the informed linkage inter group crossover with a probability of 0.25, or the informed linkage inter-intra group crossover with a probability of 0.25.

The selector follows GADO's guideline, as the informed guided linkage group crossover is a greedy operator and should be used more during the later (exploitation) phase of the search, and less during the early (exploration) phase, where it could lead to premature convergence to a local optimum.

6.2.4 Configuration and Experimental Setup of SAILEGA

To provide comparative results, three SAILEGA configurations are defined as follows:

Table 6.1: SAILEGA Configuration.

Operator/Parameter	Setting
Representation	Real-valued genome
Population Size	chromosomeLength^2
Linkage Learning	(SA)LINC-R, (SA)LIMD
Clustering	X-means
Surrogate Model	Set of SVMs with PUK kernel
Recombination	ILGL, ILIG, ILIIG, IGLG
Informed Offspring Pool	20, (following setting in [67])
Recombination prob.	80%
Mutation	Gaussian Mutation
Mutation prob.	0.02
Parent Selection	Binary Tournament Selection
Survivor Selection	Generational
Elitism	Yes (1)
Initialization	Random (Mersenne Twister[43])
Termination Condition	After 50 generations

- Configuration $SAILEGA_{default}$, using surrogate models, a summary can be seen in Table 6.1.
- Configuration $SAILEGA_{noSu}$, calling only the true fitness function for any evaluations to show the maximal potential of SAILEGAs novel operators.
- Configuration $SAILEGA_{1point}$, having only the 1-point crossover as recombination operator.

The three configurations are run 50 times on composite functions, where the base functions are given in Table 6.2 and their compositions in Table 6.3.

6.3 Surrogate-Assisted Linkage Check

In [56], a surrogate-assisted perturbation check for non-linearity (SA-LINC-R) is introduced. It performs a perturbation-based LINC-R check (as described in Chapter 4), but calls a

Table 6.2: SAILEGA Fitness Functions.

Function	Definition
F_1: Sphere$_4$	$\sum_{i=1}^{D}(x_i+4)^2$
F_2: Sphere$_{-4}$	$\sum_{i=1}^{D}(x_i-4)^2$
F_3: Rosenbrock	$\sum_{i=2}^{D}(100(x_1-x_i)^2+(x_i-1)^2)$
F_4: Schwefel 1.2	$\sum_{i=1}^{D}(\sum_{j=1}^{i}x_j)^2$
F_5: Rastrigin	$10D+\sum_{i=1}^{D}(x_i^2-10cos(2\pi x_i))$

Table 6.3: SAILEGA Function Compositions.

Config	Function^{Dim}
$SAILEGA_{default}$	$F_1^7+F_2^7$
$SAILEGA_{noSu}$	$F_1^7+F_2^7$
$SAILEGA_{1point}$	$F_1^7+F_2^7$
$SAILEGA_{default}$	$F_3^6+F_2^8$
$SAILEGA_{noSu}$	$F_3^6+F_2^8$
$SAILEGA_{1point}$	$F_3^6+F_2^8$
$SAILEGA_{default}$	$F_4^4+F_5^4+F_2^6$
$SAILEGA_{noSu}$	$F_4^4+F_5^4+F_2^6$
$SAILEGA_{1point}$	$F_4^4+F_5^4+F_2^6$

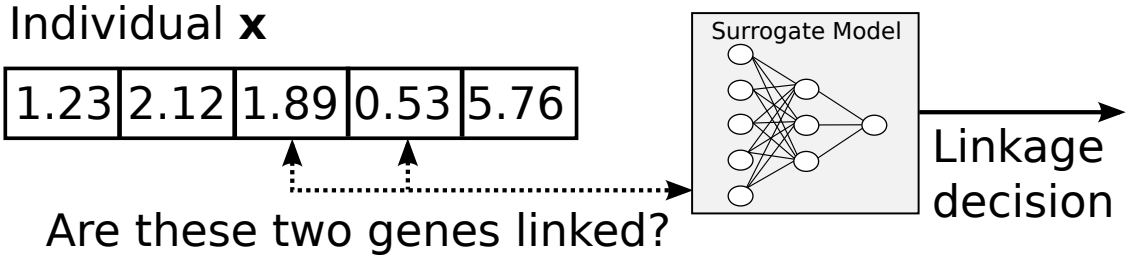

Figure 6.13: SA-LINC-R pairwise checking illustration.

surrogate-model for the evaluation of the three additional individuals that need to be created per gene-pair investigation. Figure 6.13 depicts such an operation in an illustrative example. A pseudo-code of SA-LINC-R is given in Figure 6.14. Line 4 initializes the empty linkage matrix LM that stores the numerical value of a SA-LINC-R check for genes x_i and x_j in position LM_{ij}. In Line 5, an empty list $LMList_{gen}$ is created that will store all LM matrices created during a generation (a matrix LM is created for each individual). Line 6 refers to an empty current mean linkage matrix, which will contain the means of all matrices in $LMList_{gen}$. For every individual, its LM is generated in Line 10 with the help of the surrogate model, and in Line 12 this completed LM is added to the list of all linkage matrices of this generation $LMList_{gen}$. Following this, in Line 14 a mean matrix of all matrices from $LMList_{gen}$ is created, the MLM_{gen}. Finally, in Line 15, the global mean linkage matrix MLM_{global} is updated with the MLM_{gen}. The MLM_{global} is itself a matrix created from the means of all previous MLM_t, for t from 1 to gen. At the end of every generation, the MLM_{global} is created anew, incorporating the latest MLM_{gen} and reflecting all previous and current encountered linkage information.

Each generation, the updated MLM_{global} is then used as the basis for a binary dependency structure matrix DSM, which is created by applying Weka's implementation of the k-means clustering method [24] with a fixed cluster size of two on the values of the MLM_{global}, this is shown in Equation 6.6. The DSM gives, for each generation, the accumulated binary view of existing pair-wise interactions (based on the non-linearity criterion).

1: $pop \leftarrow$ current population
2: $gen \leftarrow$ current generation
3: $surrogate \leftarrow$ trained surrogate model
4: $LM \leftarrow$ empty linkage matrix
5: $LMList_{gen} \leftarrow$ empty list of linkage matrices of gen
6: $MLM_{gen} \leftarrow$ empty mean linkage matrix
7: $MLM_{global} \leftarrow$ global mean linkage matrix
8: **for all** individuals $\mathbf{x} \in pop$ **do**
9: **for all** gene pairs x_i, x_j **do**
10: $LM_{ij} \leftarrow$ SA-LINC-R($\mathbf{x}, x_i, x_j$)
11: **end for**
12: $LMList_{gen} \leftarrowtail LM$ ▷ add the LM to the list of matrices $LMList_{gen}$
13: **end for**
14: $MLM_{gen} \leftarrow$ createMeanMatrix(LML_{gen})
15: update(MLM_{global}, MLM_{gen})

Figure 6.14: The SA-LINC-R check during a generation.

$$DSM(x_i,\, x_j) = \begin{cases} 1 & \text{if } (x_i,\, x_j) \in cluster_{highCentroid} \\ 0 & \text{if } (x_i,\, x_j) \in cluster_{lowCentroid} \end{cases} \tag{6.6}$$

While a check with LINC-R will cost additional fitness evaluations, unless the fitness function is noisy, it will return correct information about the non-linearity of the problem, provided that at least one of the alleles in each check is in a non-linear region of the fitness function. With a perturbation test done through a surrogate as function approximator, estimation errors are bound to occur. When the surrogate model is not accurate enough, a check for a gene pair that does not have any interactions in common might still claim existing linkage. Therefore it is imperative that the surrogate model is adequately trained to predict the fitness of the perturbed individuals. While using the technique from Equation 6.6 helps to smoothen the values and has been found to be necessary to obtain reliable results, the drawback of using k-means clustering as described is that if the function consists of two or more subfunctions that feature interactions on disjoint sets of genes, and one of these subfunctions yields substantialy higher pairwise linkage scores than the other(s), then

the linkage scores obtained on the other subfunctions will be incorrectly clustered into the non-linked group with the low centroid.

Neural networks will be used as surrogate models for this experiment, a focus on neural networks as function approximators is given in [33]. To increase the general approximation quality beyond the presented setup here, the work in [34] can be studied, where an overview of various surrogate models in EC is given, and in [41] where adaptive surrogate model frameworks are discussed.

Furthermore, when estimating the fitness through a surrogate, a calibration technique as described in [56] can be used, where the fitness of the original individual is used to modify the fitness of the three extra individuals per gene pair check, however depending on the domain, this technique can degrade the approximator's accuracy.

The base functions used to empirically evaluate the capabilities of this approach are given in Table 6.5. They are used to create the configurations shown in Table 6.6. A visualization of the configurations can be seen in Figure 6.15. Each configuration will be run for 50 times and a neural network with its settings shown in Table 6.4 will be used as surrogate model. The range of each dimension is fixed to $[0, 1]$. The configurations are kept in a low dimensionality with a small range and no overlapping to demonstrate the principles of this method. For more challenging or complex functions with a higher dimensionality and range, this method needs to be adapted with a more accurate surrogate modeling technique to be able to reliably detect the linkage.

6.4 Detecting Sensitivity through Machine Learning

In the following sub-sections, three approaches from machine learning to detect sensitivity in real-valued optimization and a setup for a novel empirical comparison will be presented. The approaches perform the detection through a neural network, a support vector machine for regression and a feature selection approach called ReliefF.

Table 6.4: Neural Network Linkage Configuration.

Parameter	Setting
Weka Classifier	Multilayer Perceptron
Hidden Layer Size	1
Hidden Nodes	(Number of genes + 1)
Learning Rate	0.3
Momentum	0.2
Decay	True
Normalize	True
Training Time	500
Calibration	True

Table 6.5: Linkage Detection Base Fitness Functions.

Function	Definition
F_1: Sphere	$\sum_{i=1}^{D} x_i^2$
F_2: Schwefel 1.2	$\sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$
F_3: Rosenbrock	$\sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$
F_4: Ackley	$-20exp\big(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\big) - exp\big(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i^2)\big) + 20 + e$

Table 6.6: Linkage Detection Composite Fitness Functions.

Configuration	Definition	Population	Max. Gen.	Range
1	$F_2(x_0, x_1, x_2) + F_1(x_3, x_4, x_5)$	100	10	$[0, 1]$
2	$F_3(x_0, x_1, x_2) + F_1(x_3, x_4, x_5)$	100	10	$[0, 1]$
3	$F_4(x_0, x_1, x_2) + F_1(x_3, x_4, x_5)$	100	10	$[0, 1]$
4	$F_2(x_0, x_1, x_2) + F_2(x_3, x_4, x_5)$	100	10	$[0, 1]$
5	$F_3(x_0, x_1, x_2) + F_3(x_3, x_4, x_5)$	100	10	$[0, 1]$
6	$F_4(x_0, x_1, x_2) + F_4(x_3, x_4, x_5)$	200	10	$[0, 1]$
7	$F_2(x_0, x_1) + F_2(x_2, x_3) + F_2(x_4, x_5)$	100	10	$[0, 1]$
8	$F_3(x_0, x_1) + F_3(x_2, x_3) + F_3(x_4, x_5)$	100	10	$[0, 1]$
9	$F_4(x_0, x_1) + F_4(x_2, x_3) + F_4(x_4, x_5)$	200	10	$[0, 1]$

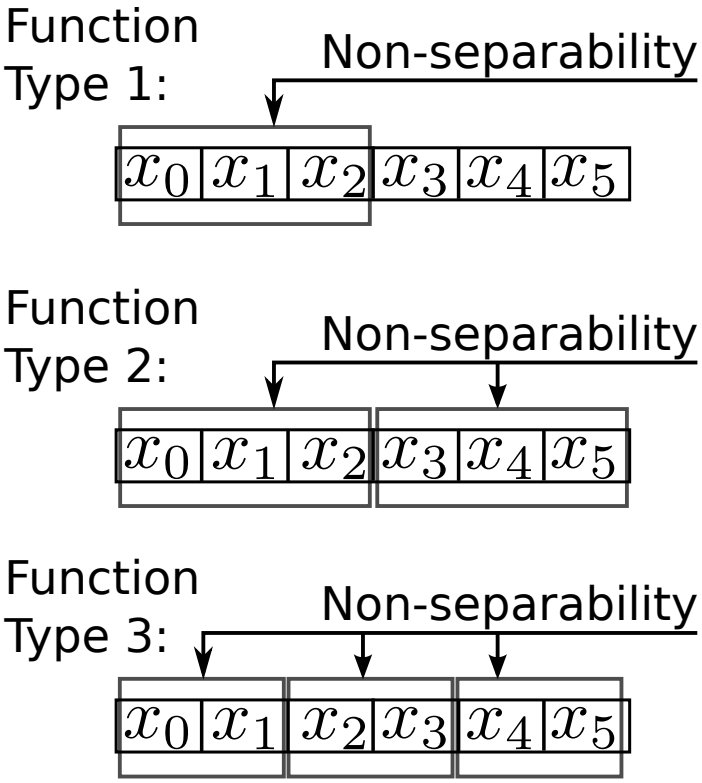

Figure 6.15: Visualization of the function setup of Table 6.6.

6.4.1 Detecting Sensitivity with a Neural Network

Introduced in [55] and applied in [54] with a GA using a specialized mutation operator, a way to detect sensitivity through machine learning methods is presented that is here part of a new empirical comparison. A neural network architecture with one hidden layer implemented in Weka is trained as a surrogate model and approximates the fitness of the individuals. Figure 6.16 (based on [54]) shows how an individual is given as the input to a neural network for training to create a surrogate model using the true fitness as the target attribute. The proposed way to induce sensitivity of a gene x_i is to use all its distinct weight paths starting from the corresponding input node and reaching all the way to the output node. An example of all such distinct weight paths for the gene x_4 (input node 5) is given is Figure 6.17 (based on [54]).

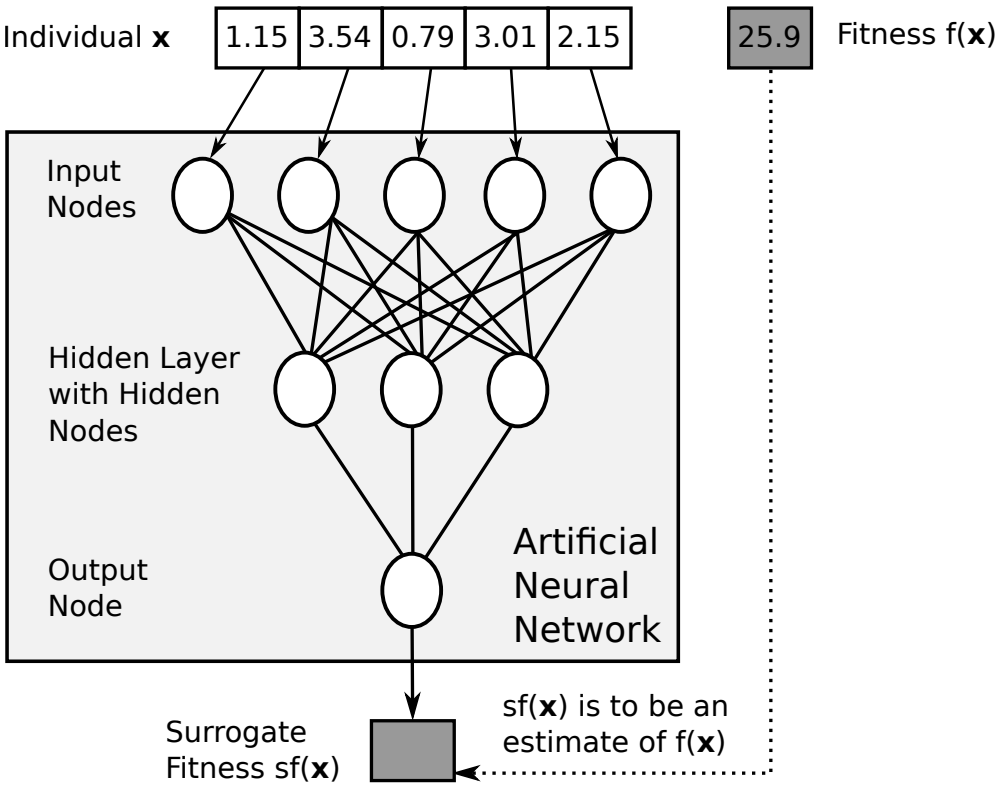

Figure 6.16: Training a neural network with an individual.

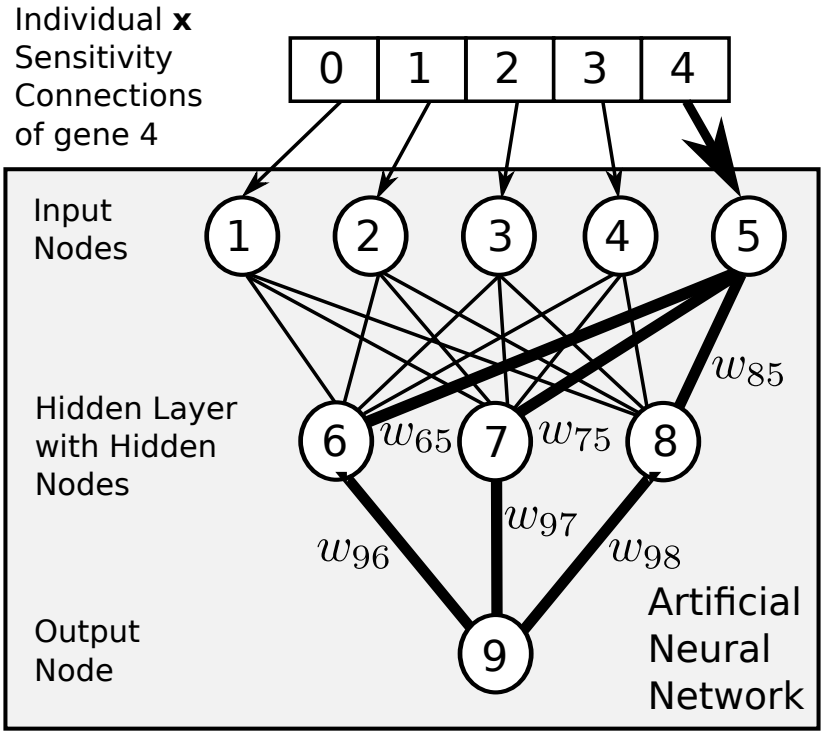

Figure 6.17: Distinct weight paths of gene x_4.

The sensitivity of a gene can be calculated with this approach as given in Equation 6.7, in which z stands for the output node and k represents the hidden nodes. In Figure 6.17, only the marked weights will be used for the calculation of the sensitivity of x_4 (in neural network literature, it is common to describe a weight from node x to node y as w_{yx}).

$$Sensitivity(x) = |\sum_{k \in hiddenLayer} w_{kx} * w_{zk}| \tag{6.7}$$

For example, the calculation for obtaining the sensitivity of x_4 as shown in Figure 6.17 is given in Equation 6.8.

$$Sensitivity(x_4) = |w_{65} * w_{96} + w_{75} * w_{97} + w_{85} * w_{98}| \tag{6.8}$$

6.4.2 Detecting Sensitivity with Support Vector Machines

Support vector machines have been developed for regression tasks [14]. An introduction and overview is provided in [80]. Weka's implementation of these is called SMOreg, and an example of measuring the strength of genes on several synthetic fitness functions with a SMOreg approach has been given in [55]. The idea behind this is also described as:

> Another possibility is to use an algorithm that builds a linear model-for example, a linear support vector machine-and ranks the attributes based on the size of the coefficients.
>
> — Ian H. Witten and Eibe Frank and Mark A. Hall, [92]

Although a support vector machine that builds a linear model through its kernel can have deficits in terms of its predictive power for the fitness when used as a surrogate model on non-linear fitness functions, the primary goal here is to capture the raw strengths of contribution of the genes.

6.4.3 Detecting Sensitivity through ReliefF

In machine learning, feature selection aims to select a useful subset of variables (features) from all variables. This helps to discard variables that add no explanatory benefit to the dataset but just increase the dimensionality of the problem and to reduce the problem's general dimensionality by selecting only the most important features for further machine learning tasks. ReliefF for regression tasks [71] is one such approach which aims to detect the strength of contribution of each attribute by continually sampling instances and performing a comparison of how well an attribute can categorize instances in close proximity of the sampled. For regression problems, as noted in [71], a probability is used to make a decision on membership to the same or different class based on the real-valued target class. Weka's implementation of ReliefF can act jointly with the Ranker search method for attribute evaluation, which "sorts attributes by their individual evaluations" [92]. With the idea from Chapter 3 that a population can be viewed as a dataset, such a feature selection technique becomes directly applicable to identify the most influential genes.

6.4.4 Configuration and Experimental Setup of Sensitivity Detection

To provide empirical evidence and comparisons for the results of the aforementioned machine learning based sensitivity detection approaches, the functions in Table 6.7 are used as fitness functions for an EA. They have already been investigated for epistasis detection in [10]. In the Increasing Sphere Function, the higher the number of a gene, the higher its sensitivity. This is realized by the function's exponent. The Schwefel 1.2 Function features interactions of genes in addition to a varying sensitivity. An example of this is shown in Figure 6.18. The dimensionality of the experiments is set to 4, 16 and 32 with the two ranges $[0, 1]$ and $[0, 100]$. The goal of the experiment is to detect the sensitivities of genes in the first generation. The full function setup is provided in Table 6.8. The configurations of the Weka algorithms of

Table 6.7: Sensitivity Fitness Functions.

Function	Definition
F_1: Increasing Sphere	$\sum_{i=1}^{D} 1.5^{i-1} x_i^2$
F_2: Schwefel 1.2	$\sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$

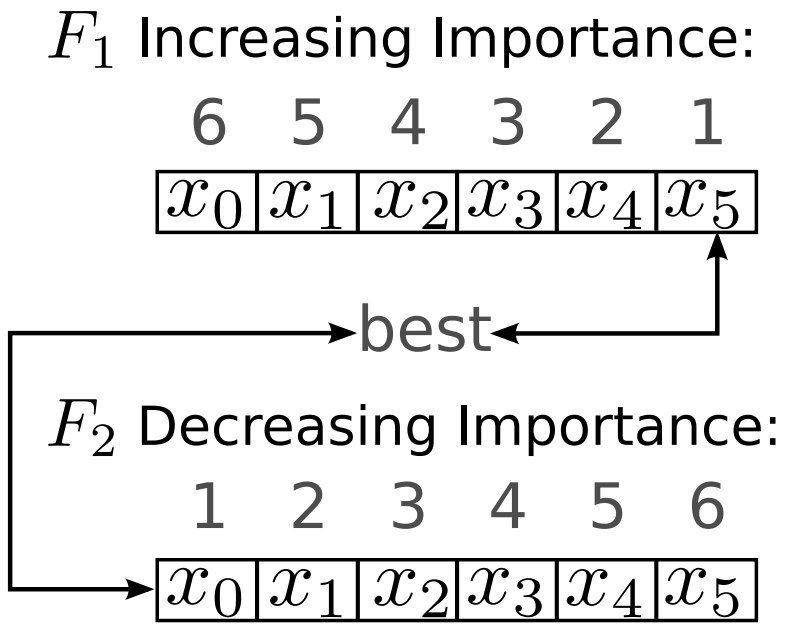

Figure 6.18: Example of increasing and decreasing rankings of genes.

neural networks, ReliefF and SMOreg are given in Tables 6.9, 6.10 and 6.11 and have been chosen based on sensible Weka defaults. Each configuration is run 50 times, with the results being ranking averages, Spearman's ρ as well as two different probability measures of the rankings.

Table 6.8: Sensitivity Fitness Function Configurations.

Configuration	Function	Dimension	Population	Range
1	F_1: Increasing Sphere	4	50	$[0, 1]$
2	F_1: Increasing Sphere	4	50	$[0, 100]$
3	F_1: Increasing Sphere	16	500	$[0, 1]$
4	F_1: Increasing Sphere	16	500	$[0, 100]$
5	F_1: Increasing Sphere	32	1000	$[0, 1]$
6	F_1: Increasing Sphere	32	1000	$[0, 100]$
7	F_2: Schwefel 1.2	4	50	$[0, 1]$
8	F_2: Schwefel 1.2	4	50	$[0, 100]$
9	F_2: Schwefel 1.2	16	500	$[0, 1]$
10	F_2: Schwefel 1.2	16	500	$[0, 100]$
11	F_2: Schwefel 1.2	32	1000	$[0, 1]$
12	F_2: Schwefel 1.2	32	1000	$[0, 100]$

Table 6.9: Neural Network Sensitivity Configuration.

Parameter	Setting
Weka Classifier	Multilayer Perceptron
Hidden Layer Size	1
Hidden Nodes	(Number of genes + 1) / 2
Learning Rate	0.3
Momentum	0.2
Decay	True
Normalize	True
Training Time	500

Table 6.10: ReliefF Sensitivity Configuration.

Parameter	Setting
Weka Attribute Evaluator	ReliefF
Number of neighbors	10
Sigma	2
Search Method	Ranker

Table 6.11: SMOreg Sensitivity Configuration.

Parameter	Setting
Weka Classifier	SMOreg
Kernel	Polynomial Kernel
Kernel Exponent	1
Kernel Cache Size	250007
RegOptimizer	RegSMOImproved

Chapter 7

Benchmark for Problem Structures

This Chapter introduces three recent EC benchmarks for real-valued representations and identifies their limitations in regards to being able to exhibit the behavior of EAs on problems with complex and overlapping linkage structures. Following this, a novel benchmark suite is presented that mends these shortcomings. The benchmark and contents of this Chapter are to appear in GECCO 2013. As benchmarks are important to indicate the quality of EAs over different problem classes, it is critical to cover situations where some genes share interactions with each other - a situation that often occurs in real-world problems. This benchmark provides the possibility to quantitatively assess the performance of various optimization techniques on problems with overlapping linkage groups of varying sizes, and is not only limited to evolutionary algorithms.

7.1 Contemporary Benchmarks

The purpose of public benchmarks in fields related to global optimization such as EC is to:

- Give a possibility to empirically evaluate an EA's capabilities on a broad set of established problem classes and to inspect its optimization abilities and deficiencies on problems with certain features.
- Provide the means for statistical tests to compare different EAs on well-studied problems and to make statements about their statistically significant differences [2].

- Enable extrapolations of the EA's performance on new, unseen problems (potentially from real-world domains) that share similarities with the synthetic benchmark fitness functions.

This improves upon the practice of testing new EAs on only one problem or with publicly inaccessible functions and gives a common testing ground of assembled, well defined functions. The two major contemporary benchmarks for real-valued evolutionary optimization are:

- The Real-Parameter Black-Box Optimization Benchmarking (BBOP) [25] alongside with the COmparing Continuous Optimisers (COCO) platform[1].
- The CEC'2010 Special Session and Competition on Large-Scale Global Optimization (LSGO) [82] benchmark[2].

Furthermore, an extension of the LSGO benchmark, in this dissertation called LSGO-extended, was presented in [74] and contains overlapping composite functions.

7.1.1 BBOP

The BBOP benchmark from GECCO 2012 consists of two parts, a noise-free and a noisy set of problems. Only the noise-free part is considered in this dissertation. The 24 functions for real-valued optimization that it is composed of can be, under the MPM, classified as:

1. Fully separable.
2. Fully non-separable.

A visualization of this is shown in Figure 7.1. There is, however, an emphasis on other functional properties such as ruggedness, deceptiveness, symmetry and (multi)modality of

[1] http://coco.gforge.inria.fr/doku.php
[2] http://staff.ustc.edu.cn/~ketang/cec2012/cec2012lsgo.htm

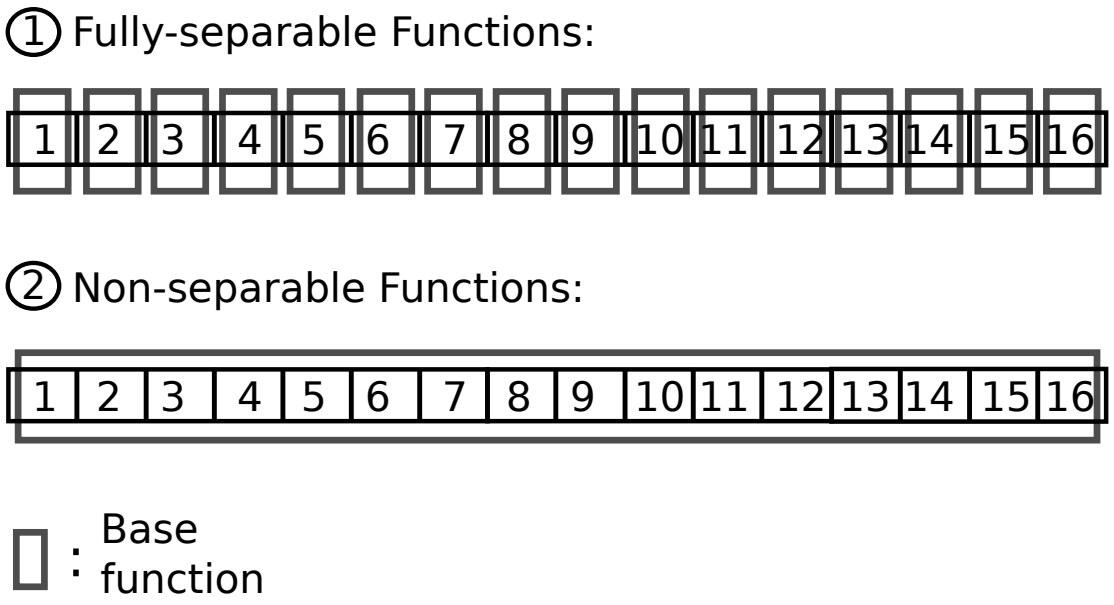

Figure 7.1: Illustrations of MPM separability from the BBOP benchmark.

different degrees. The benchmark also allows shifting the global optimum of the function individually per dimension and implements a rotation technique to make a separable function fully non-separable. The Katsuura, Weierstrass and Sharp Ridge Functions have been taken from here to supplement the novel benchmark of this dissertation with a more varied function set.

7.1.2 LSGO

In contrast to this, linkage among genes is a main focus in the LSGO benchmark, which includes 20 problems for large-scale global optimization. Although multiple linkage groups that comprise the chromosome are offered here, the group sizes remain constant throughout the chromosome and no function includes overlapping linkage groups. Furthermore, these problems are composite functions constructed from six base functions, thereby offering less diversity of the remaining function properties than the BBOP benchmark. These six functions are the Sphere, Elliptic, Schwefel 1.2, Rosenbrock's, Rastrigin's and Ackley's Function.

Similar to the BBOP benchmark, the LSGO also offers a shifting of the global optimum. It also gives a permutation vector to randomize the order of the genes on the chromosome to prevent any positional bias. With the exception of the Elliptic Function[3], all other functions from LSGO are included in the benchmark given in this dissertation. Figure 7.2 displays the separability configurations of the LSGO benchmark and shows an overview of the different fitness function compositions[4] in this benchmark. As can be seen, it includes functions that are:

1. Fully separable.
2. Single-group m-non-separable, where m denotes a size of a group of genes that form a non-separable linkage group, while the rest of the genes are evaluated by a fully separable function.
3. $\frac{D}{2m}$-group non-separable-m, where same-sized linkage groups are formed on the first half of the chromosome, while the remaining part is evaluated by a fully separable function.
4. $\frac{D}{m}$-group non-separable-m, where same-sized linkage groups are formed on the length of the whole chromosome.
5. Fully non-separable.

None of these setups includes overlapping linkage groups, and, the size of the non-separable groups within a chromosome does not change.

7.1.3 LSGO-extended

The LSGO-extended benchmark is composed of the same six base functions as LSGO. However it has been modified to allow an overlapping of non-separable groups. Its setup is shown

[3]This function is a variant of the Sphere Function with a non-constant sensitivity of the genes.

[4]The dimensionality of LSGO is by default 1000 and its linkage group size is 50, this has been lowered in the Figure for better visibility without loss of generality.

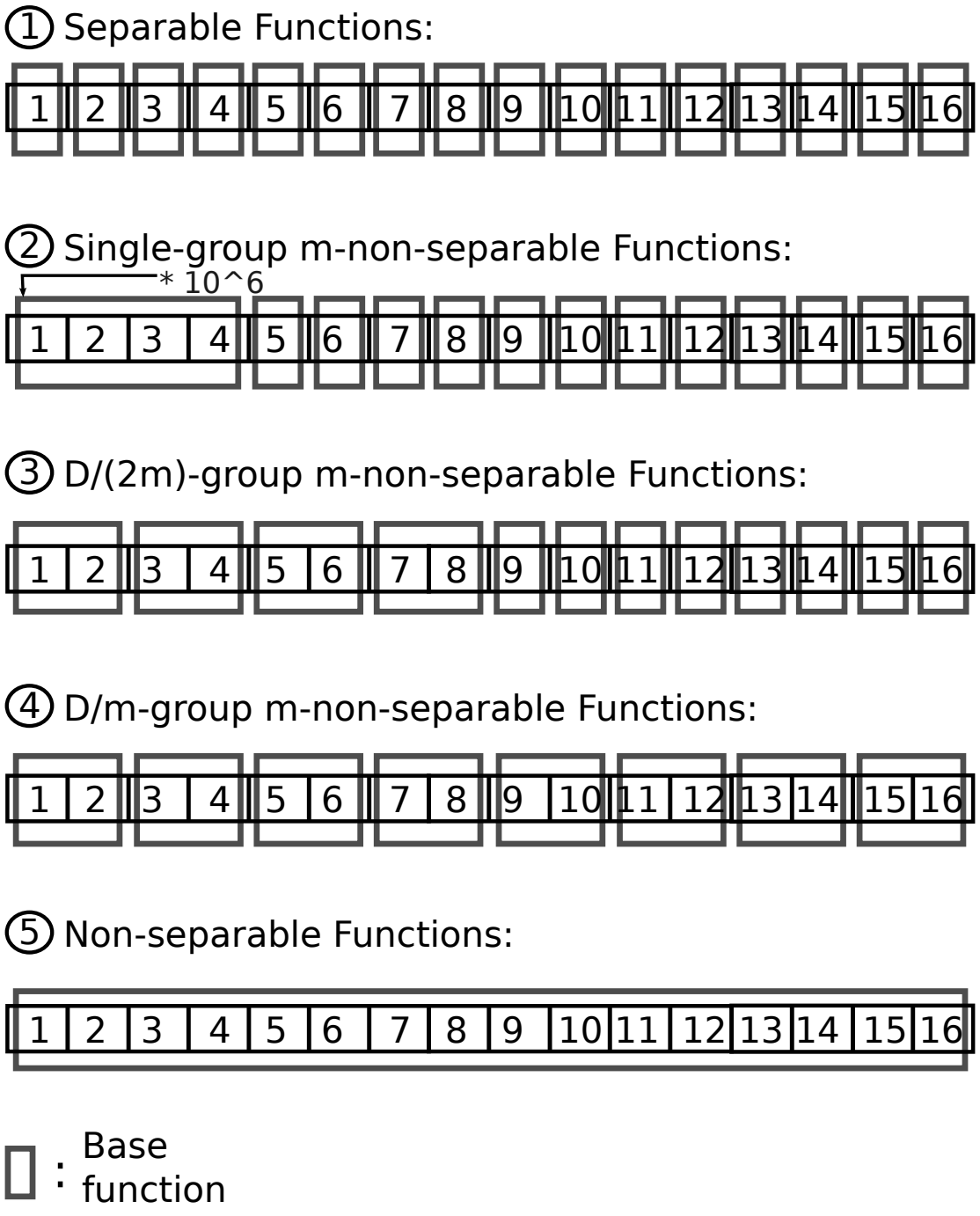

Figure 7.2: Illustrations of separability from the LGSO benchmark.

in Figure 7.3 where the composite fitness functions can be of the following form:

1. Two fully non-separable and non-overlapping functions that split the chromosome in half. Viewed logically, these genes can either be subsequent to each other as shown in the first example (so that a one-point crossover that splits a chromosome with even length in the middle would perfectly divide the two linkage groups) or *spliced*, which assigns successive genes in an alternating way to the two different fitness functions.

2. Two fully non-separable and non-overlapping functions that split the chromosome in half, and in addition to that, a number of genes (determined by the parameter *ov*) evaluated on the first fitness function that also are evaluated on the second. The spliced version is omitted from Figure 7.3.

3. Two fully non-separable and non-overlapping functions that split the chromosome in half, with *ov* genes from the first fitness function evaluated on the second function and, additionally, $\frac{1}{10}$ genes at the end of the chromosome (also denoted to as *sep*) that are only evaluated by a fully separable function. Again, the spliced version is omitted from Figure 7.3.

Although this benchmark introduces overlapping linkage groups for its fitness functions, the following shortcomings can be identified:

- Splicing is supposed to prevent a bias of position, where linked groups of genes are (logically) always next to each other and a linkage-blind optimizer might still unintentionally exploit this, with for instance the application of the aforementioned one-point crossover example. However, the LSGO benchmark offers a permutation vector, which completely randomizes the logical positioning order of the genes, and splicing is only a special case thereof.

- The linkage groups have the same size, they either split the chromosome in half ($\frac{D}{2}$) or they split it in half minus the fully separable part $\frac{D-sep}{2}$.

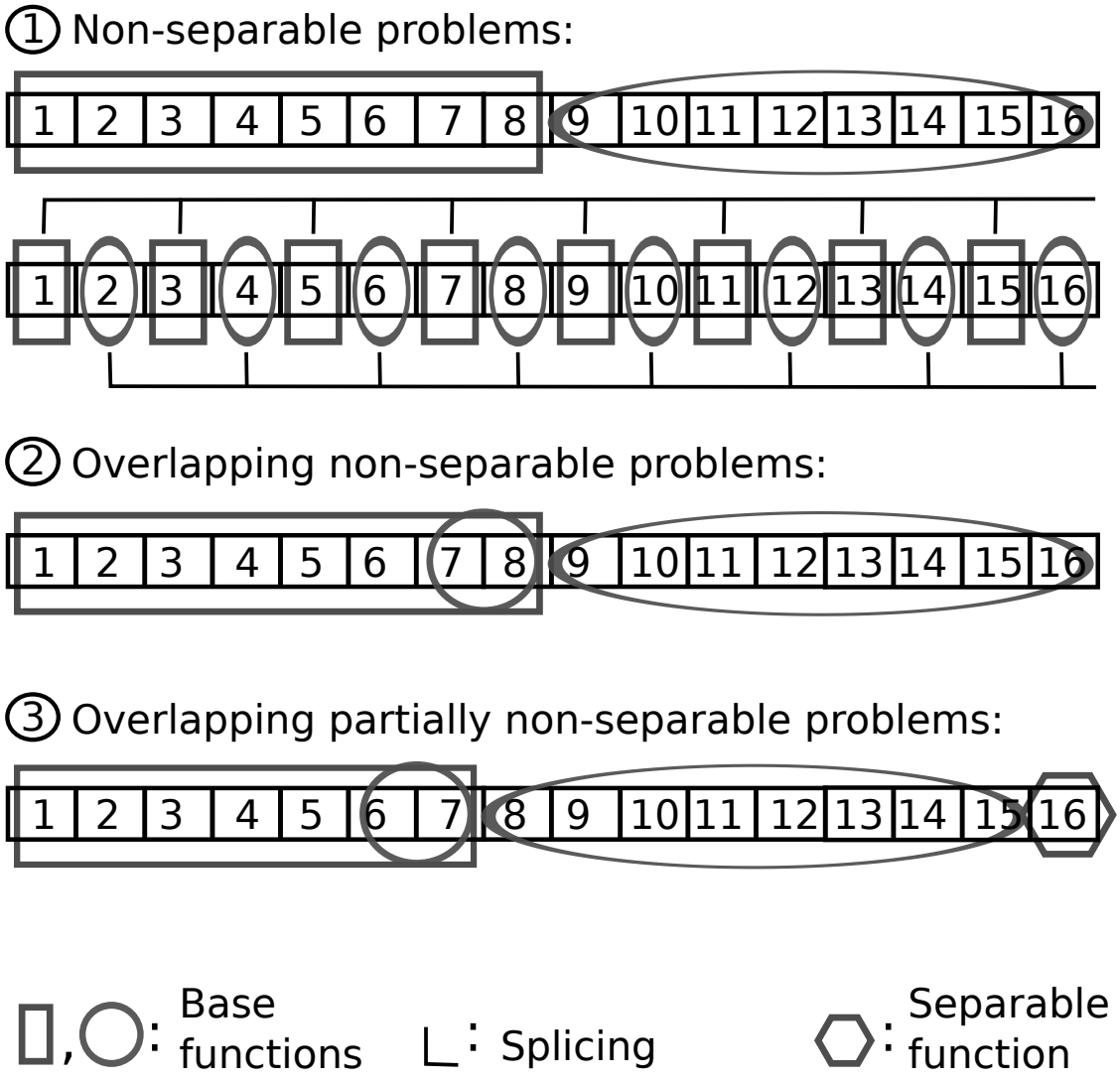

Figure 7.3: Illustrations of separability from the LSGO-extended benchmark.

- While there is overlapping through *ov* and a fully separable part with *sep*, there are always only two linkage groups, each group evaluated on a non-separable fitness function.

- *ov* and *sep* are fixed constants, so that the amount of overlapping is either zero or always *ov* throughout the whole benchmark run.

- As with the original LSGO, only six base functions are included, which limits the expressiveness of this benchmark compared to the BBOP setup.

These deficiencies are addressed in the novel benchmark of this dissertation.

7.2 Overlapping Variable Linkage Benchmark

This Sections presents a novel overlapping variable linkage benchmark for real-valued optimization, introduces its templates, eight base and 33 composite fitness functions and discusses their configurations.

7.2.1 Templates

The Overlapping Variable Linkage Benchmark (OVLB) includes three templates that are applied to create its composite benchmark functions, these are shown in Figure 7.4. The templates are:

1. **Non-overlapping linkage groups**: Composite functions of this type are divided into linkage groups of increasing size, where the increase is determined by the powers of two (the sizes of the linkage groups are $1, 2, 4, 8, 16, 32, ...$). Two different non-separable fitness functions are applied in an alternating fashion to evaluate these groups and their results are added to the overall fitness of the individual. Similar to the LSGO-extended benchmark, $\frac{1}{10}$ of the genes of the chromosome are not evaluated on either of these

two functions, but on a fully separable function, and are therefore independent. The rationale for this template is to represent a class of real-world problems that have separate groups of interacting genes (for instance a group of control parameters that experience interactions with each other but do not share any epistasis with other groups of control parameters) with different sizes. In addition, a relatively small number of genes do not have any interactions, adding to the problem as a whole.

2. **Minimal overlapping linkage groups**: These functions are laid out in the same way as the functions from **(1.)**. However, overlapping is added to increase the problem's structure complexity. For each neighboring linkage group, the two adjacent genes of each group are partaking in an extra evaluation through a non-separable fitness function, establishing a transitive relation between the two groups. In the MPM representation, this would mean that all linkage groups that have such a minimal overlapping would be grouped together as one giant interacting group.

3. **Increasing overlapping linkage groups**: As with **(2.)**, after the setup of the composite and fully separable functions is complete (so that the template looks like **(1.)**), overlapping is added. However, in this template, the overlapping linkage groups increase linearly in size $(2, 4, 6, 8, 10, 12, 14, ...)$. Smaller linkage groups are therefore interacting with all or a considerable amount of their genes with other linkage groups. This effect diminishes with the increase of linkage group sizes, which follow the powers of two, so that large linkage groups only interact directly with a relatively small number of genes with distinct groups of linked genes. For example, the linkage groups with size 32 and 64 interact on 10 genes with each other through an extra applied non-separable fitness function on them, while the linkage groups of size 256 and 512 only interact on 16 genes. With this, an increase in overlapping is simulated while at the same time larger linkage groups are less affected by direct interactions.

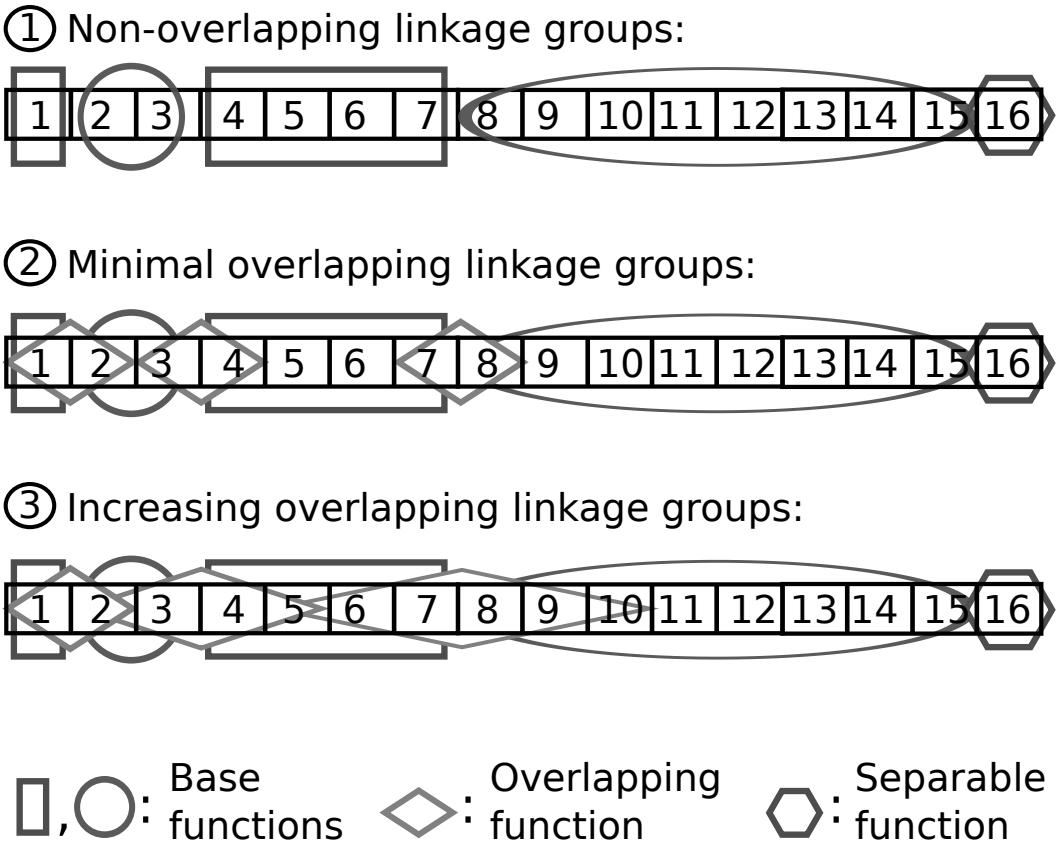

Figure 7.4: Illustrations for the novel overlapping variable linkage benchmark.

The algorithm for the non-overlapping linkage groups is shown in Figure 7.5, while the algorithm for the minimal and increasing overlapping linkage groups is given in Figure 7.6. The only difference between these templates is the amount of overlapping that is occurring among the individual linkage groups, specified through the variable *overlapping*.

7.2.2 Features

Other features of the OVLB, that have been borrowed from the LSGO benchmark, include:

- **Shift Vector**: A shift vector is created as a constant at the instantiation time of an OVLB function and shifts the global optimum uniformly randomly within allowable boundaries. This vector is accessed in the fitness function at evaluation time. The shift

```
1: result ← 0
2: i ← 0
3: separable ← separable genes
4: individual ← current individual
5: groupLengths ← linkage group lengths
6: kernel ← fitnessFunctions ▷ contains F1 − F8
7: for i = 0; i ≤ groupLengths.size; i++ do
8:    if i % 2 == 0 then
9:       result += kernel.f1(individual, groupLengths_i)
10:   else
11:      result += kernel.f2(individual, groupLengths_i)
12:   end if
13: end for
14: result += kernel.fSeparable(individual, separable)
15: return result
```

Figure 7.5: The fitness function selector without overlapping.

```
1: result ← 0
2: i ← 0
3: overlapping ← overlapping genes
4: separable ← separable genes
5: individual ← current individual
6: groupLengths ← linkage group lengths
7: kernel ← fitnessFunctions ▷ contains F1 − F8
8: for i = 0; i ≤ groupLengths.size; i++ do
9:    if i % 2 == 0 then
10:      result += kernel.f1(individual, groupLengths_i)
11:   else
12:      result += kernel.f2(individual, groupLengths_i)
13:   end if
14: end for
15: result += kernel.fOverlap(individual, overlapping)
16: result += kernel.fSeparable(individual, separable)
17: return result
```

Figure 7.6: The fitness function selector with overlapping.

vector does not change after creation, so that the global (shifted) optimum for each gene stays the same for the whole optimization run. For example, if $\mathbf{x} = [x_1, \ldots, x_n]$ is an individual and $\mathbf{o} = [o_1, \ldots, o_n]$ is the (shifted) global optimum, then $\mathbf{z} = \mathbf{x} - \mathbf{o} = [z_1, \ldots, z_n]$ is a vector that is evaluated on the OVLB function. Without such a shifting, global optima can be typically located at the alleles of 0.0.

- **Permutation Vector**: A permutation vector is also created as a constant at the instantiation time of an OVLB function and contains the permuted indices of the chromosome's genes. As an example, the typical indices of a chromosome are $[0, 1, 2, \ldots, n]$, and this permuted vector is a shuffled version (random permutation) thereof. This allows the removal of positional bias from an OVLB function, and while linkage groups are situated next to each other on the genotype, logically (at the time of the fitness function evaluation), they are shuffled throughout the chromosome. Shifting and permuting a chromosome with regards to the optimization is shown in Figure 7.7

- **Rotation Matrix**: A random coordinate rotation technique directly taken from the LSGO benchmark is applied to create non-separable versions of separable functions.

The shifting and permuting operations can be disabled by providing a vector of zeros as the shift vector (so that the global optimum for each gene does not move) and a vector with integer numbers from 1 to n as the permutation vector, so that the genotypical (real) and the logical (view of the fitness function) location of the genes are the same.

7.2.3 Synthetic Base Functions

The following synthetic base functions are applied on the defined templates in this Chapter. They are provided in their mathematical forms in Table 7.1, followed by 3-D surface plots made with gnuplot. Those functions are further described in [25] or [82]. The functions are:

1. F_1, the Sphere Function is a unimodal function that is fully separable. It is relatively simple to optimize compared to the other base functions, and is used to evaluate 1/10

Table 7.1: Synthetic Base Functions.

Function	Definition
F_1: Sphere	$\sum_{i=1}^{D} x_i^2$
F_2: Schwefel 1.2	$\sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$
F_3: Rosenbrock	$\sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$
F_4: Rastrigin	$\sum_{i=1}^{D} (x_i^2 - 10cos(2\pi x_i) + 10)$
F_5: Ackley	$-20exp\big(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\big) - exp\big(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i^2)\big) + 20 + e$
F_6: Weierstrass	$\sum_{i=1}^{D} (\sum_{k=0}^{20} (1/2^k cos(2\pi 3^k (x_i + 1/2)))) - D\sum_{k=0}^{20} (1/2^k cos(\pi 3^k))$
F_7: Katsuura	$\frac{10}{D^2}\prod_{i=1}^{D}\big(1 + i\sum_{j=1}^{32}\frac{\lvert 2^j x_i - \lfloor 2^j x_i \rfloor\rvert}{2^j}\big)^{10/D^{1.2}} - \frac{10}{D^2}$
F_8: Sharp Ridge	$x_1^2 + 100\sqrt{\sum_{i=2}^{D} x_i^2}$

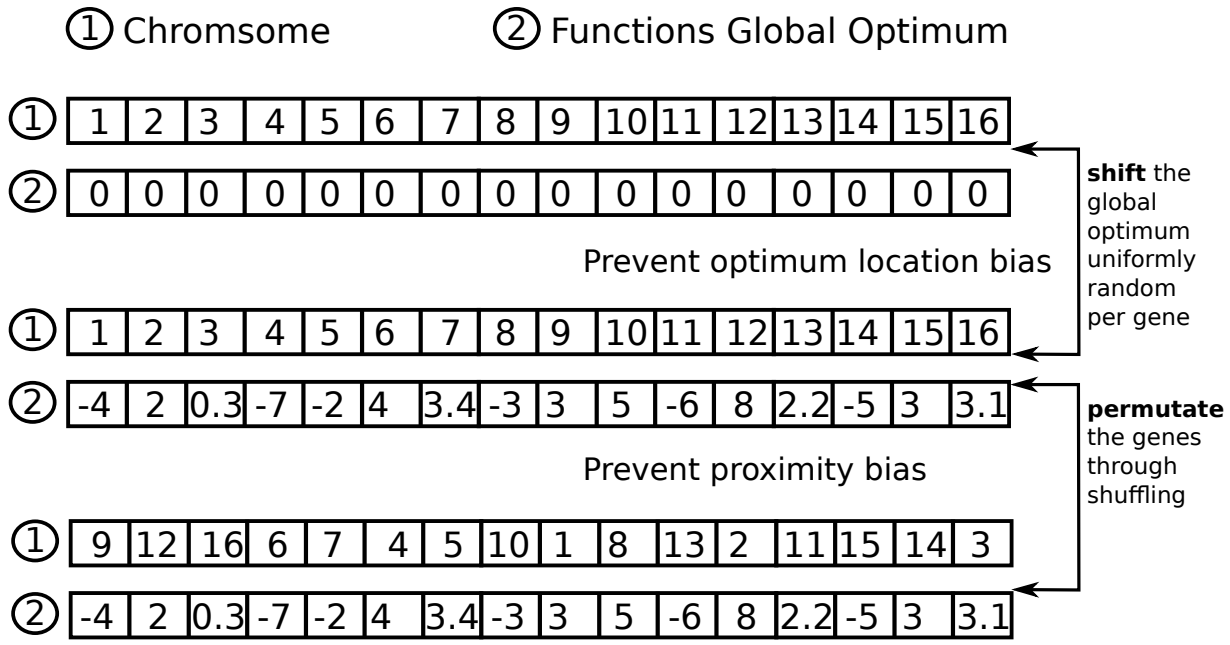

Figure 7.7: Illustration of shifting the global optimum and the permutation of access to the genes. This happens only at the beginning of the optimization run and stays fixed until the end of the optimization of the particular function.

of the genes of a chromosome in cases where neither of the two base functions that are applied to create the linkage groups are originally separable (prior to the application of the random coordinate rotation technique of the LSGO benchmark). 3-D surface plots for the Sphere Function are shown for the range $[-1, 1]$ in Figure 7.8 and for the range $[-100, 100]$ in Figure 7.9.

2. F_2, the Schwefel 1.2 Function, defined in [76], is a unimodal function that is fully non-separable. From its definition it can be inferred that in contrast to F_1, the sensitivity of genes differs. 3-D surface plots for the Schwefel 1.2 Function are shown for the range $[-1, 1]$ in Figure 7.10 and for the range $[-100, 100]$ in Figure 7.11.

3. F_3, the Rosenbrock Function, is a multimodal function in higher dimensions (4∼30[78]) that is non-separable. It is one of the most widely used optimization benchmark functions. The global minimum of this function is located inside a narrow valley of

the search space, and therefore this problem "often serves as a test case for premature convergence" [78]. 3-D surface plots for the Rosenbrock Function are shown for the range $[-1, 1]$ in Figure 7.12 and for the range $[-100, 100]$ in Figure 7.13.

4. F_4: the Rastrigin Function is a highly multimodal function (its modality increases exponentially in relation to the dimensionality [82]) that is fully separable. It bears similarity to F_1, but has the cosine function applied to it. 3-D surface plots for the Rastrigin Function are shown for the range $[-1, 1]$ in Figure 7.14 and for the range $[-100, 100]$ in Figure 7.15.

5. F_5: the Ackley Function is a multimodal function (also including the usage of the cosine function as F_4), which has many local optima spread out across the search space and one global optimum. This function is separable (but not additively separable). 3-D surface plots for the Ackley Function are shown for the range $[-1, 1]$ in Figure 7.16 and for the range $[-100, 100]$ in Figure 7.17.

6. F_6: the Weierstrass Function is a multimodal and separable function, and has more than one unique global optimum [25]. It is famous for being a continuous function that is nowhere differentiable. 3-D surface plots for the Weierstrass Function are shown for the range $[-1, 1]$ in Figure 7.18 and for the range $[-100, 100]$ in Figure 7.19.

7. F_7: the Katsuura Function is highly multimodal, separable, and just like F_6 continuous on all points but nowhere differentiable [25]. 3-D surface plots for the Katsuura Function are shown for the range $[-1, 1]$ in Figure 7.20 and for the range $[-100, 100]$ in Figure 7.21.

8. F_8: the Sharp Ridge Function defines a non-differentiable sharp ridge that "needs to be followed" [25] to reach the global optimum. 3-D surface plots for the Sharp Ridge Function are shown for the range $[-1, 1]$ in Figure 7.22 and for the range $[-100, 100]$ in Figure 7.23.

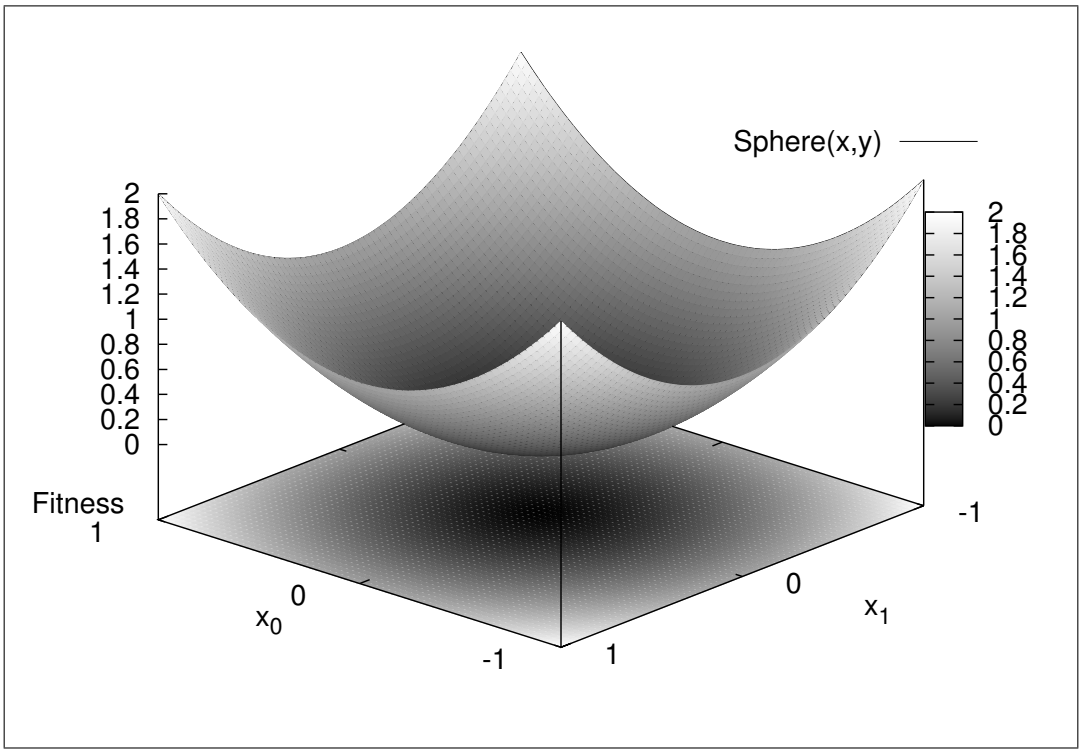

Figure 7.8: 3-D plot of F_1, the Sphere Function, range $[-1, 1]$.

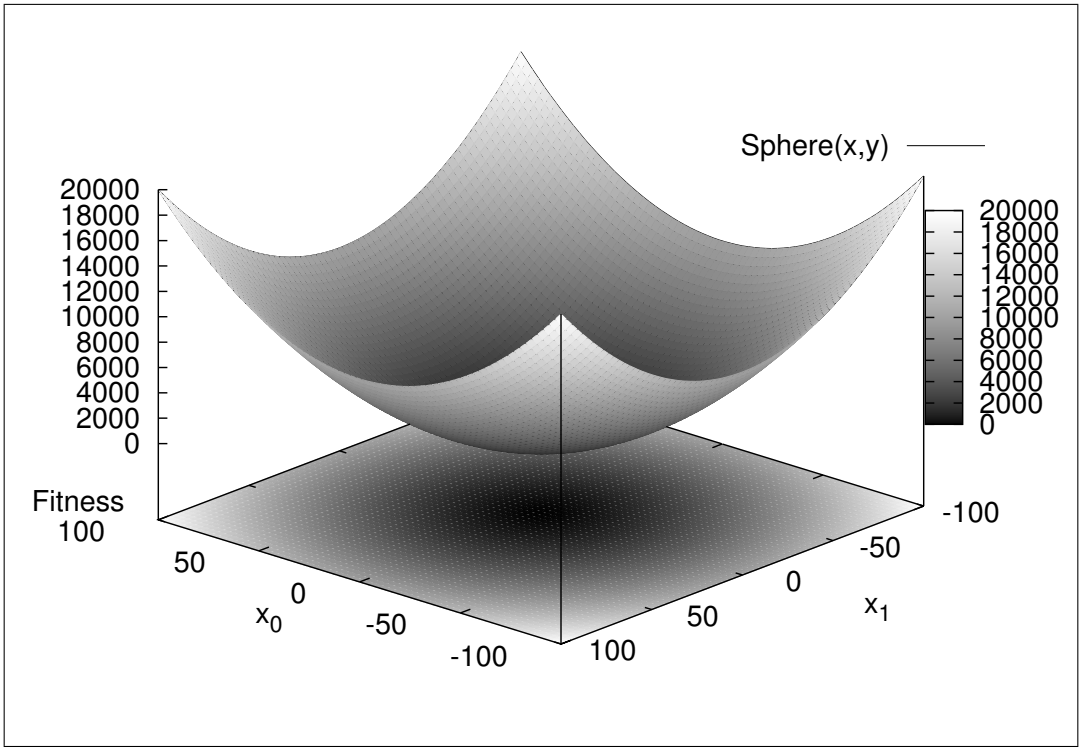

Figure 7.9: 3-D plot of F_1, the Sphere Function, range $[-100, 100]$.

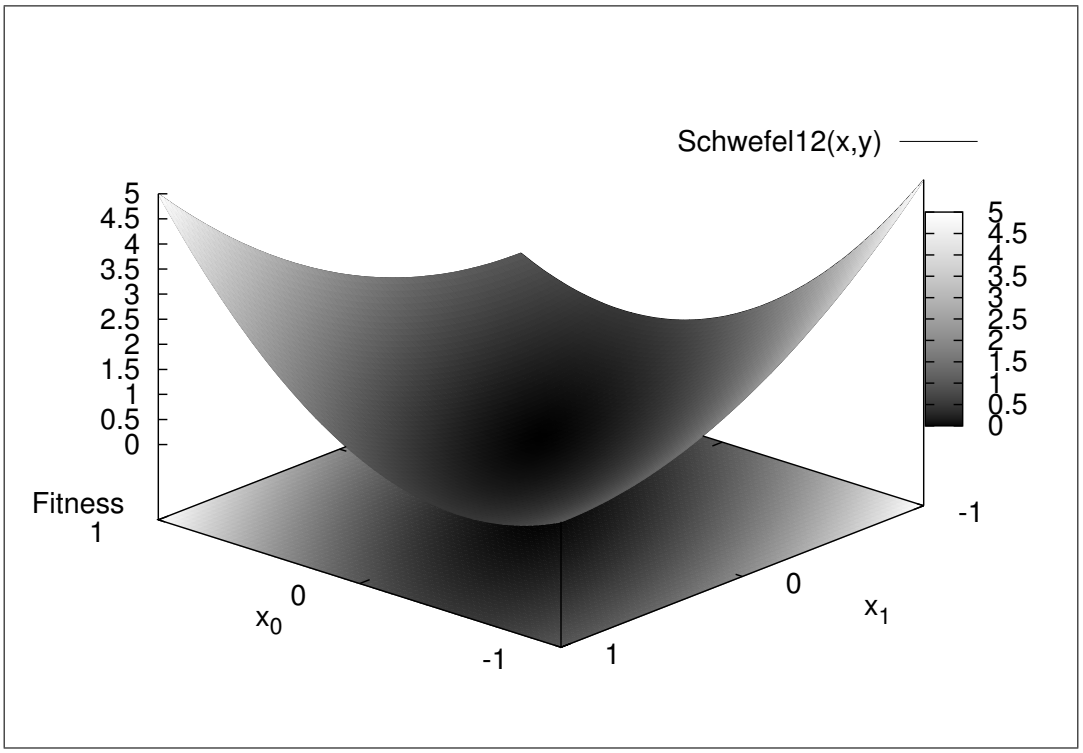

Figure 7.10: 3-D plot of F_2, the Schwefel 1.2 Function, range $[-1, 1]$.

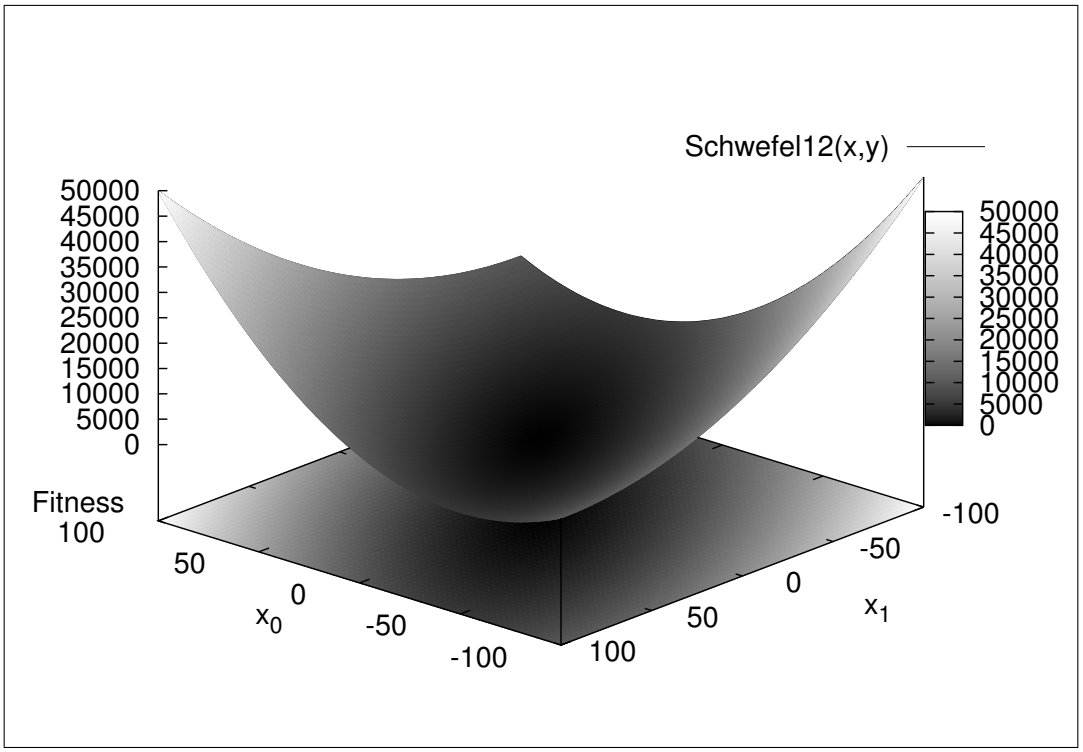

Figure 7.11: 3-D plot of F_2, the Schwefel 1.2 Function, range $[-100, 100]$.

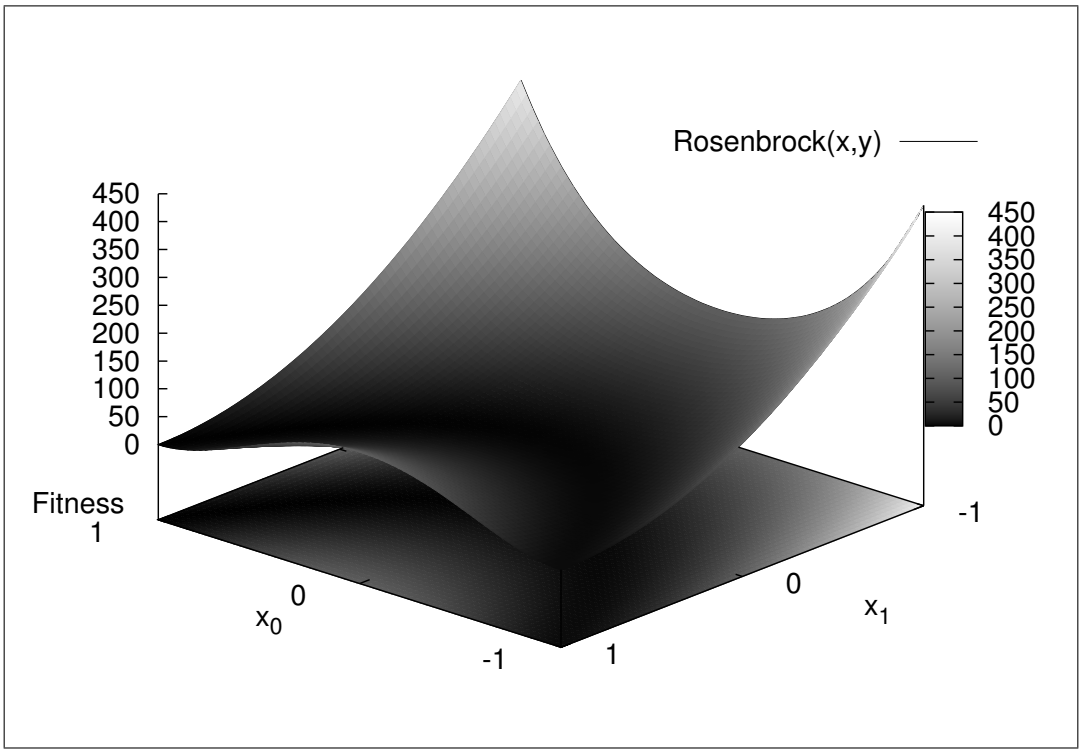

Figure 7.12: 3-D plot of F_3, the Rosenbrock Function, range $[-1, 1]$.

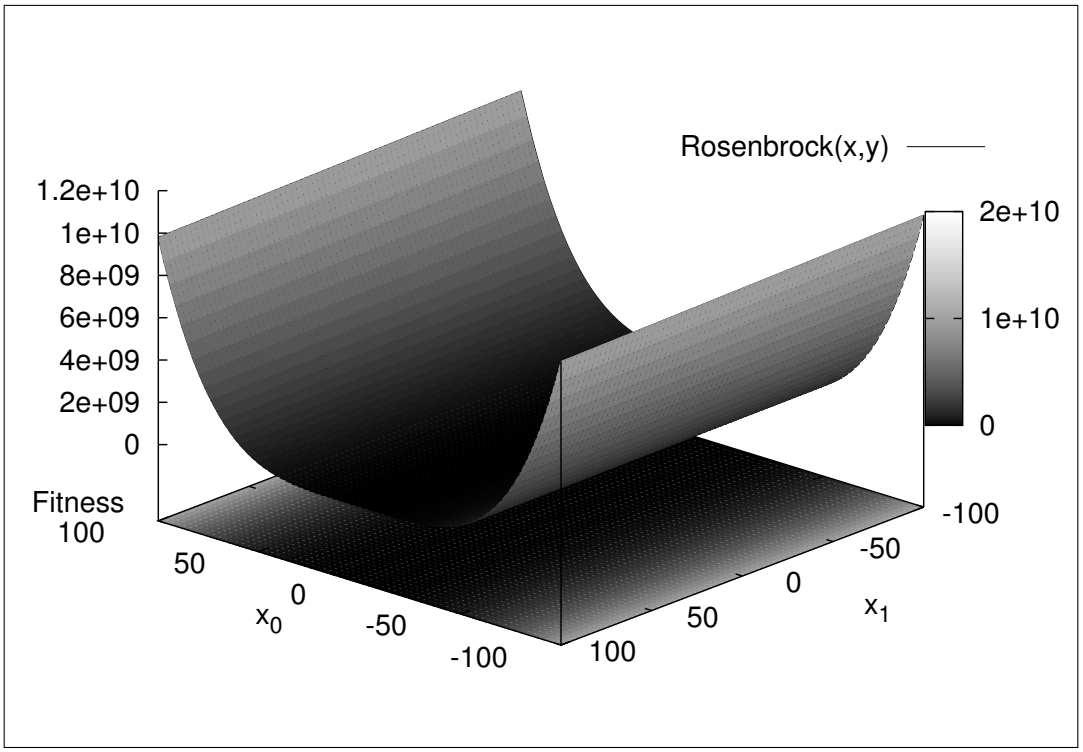

Figure 7.13: 3-D plot of F_3, the Rosenbrock Function, range $[-100, 100]$.

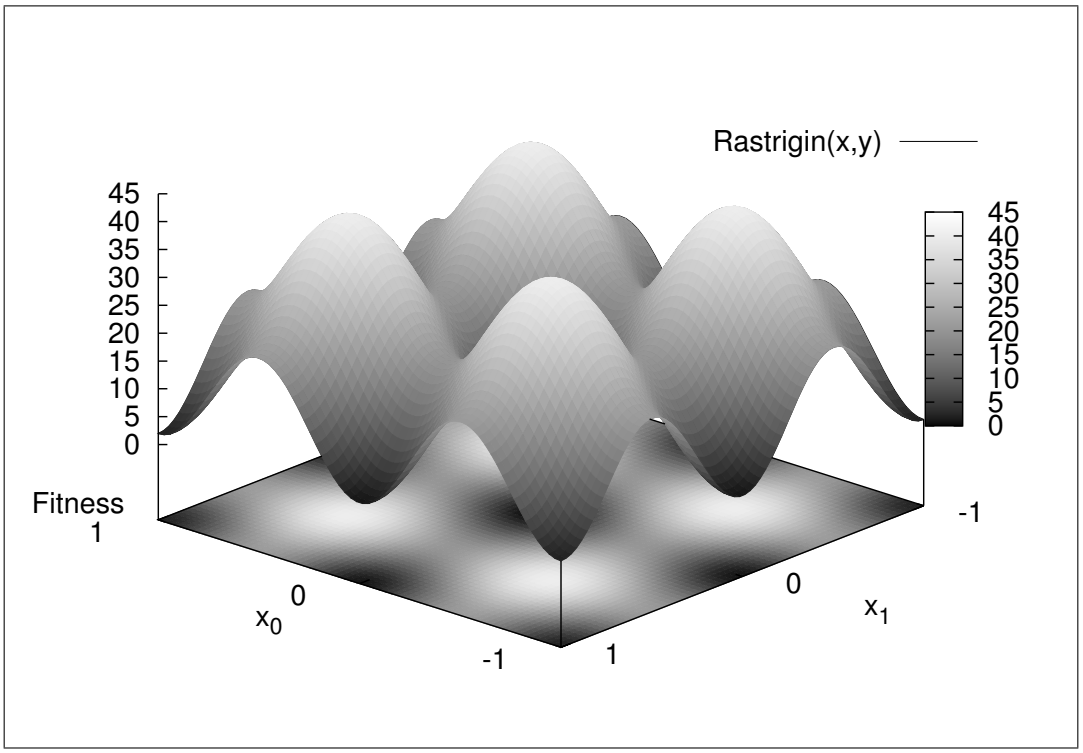

Figure 7.14: 3-D plot of F_4, the Rastrigin Function, range $[-1, 1]$.

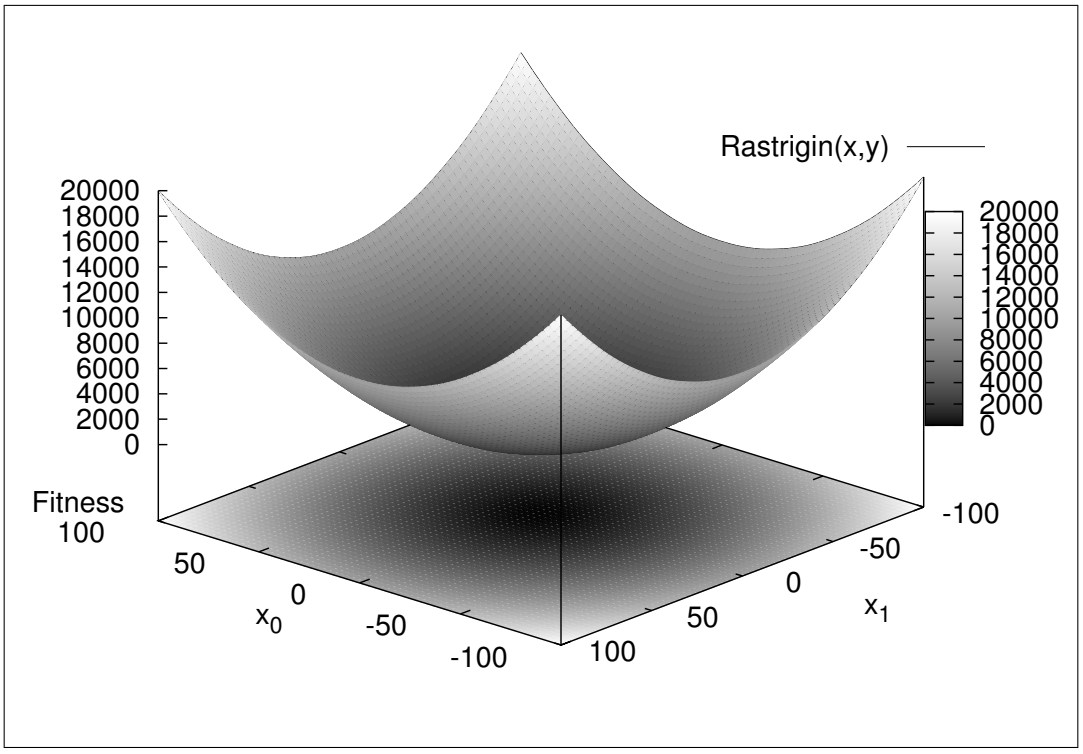

Figure 7.15: 3-D plot of F_4, the Rastrigin Function, range $[-100, 100]$.

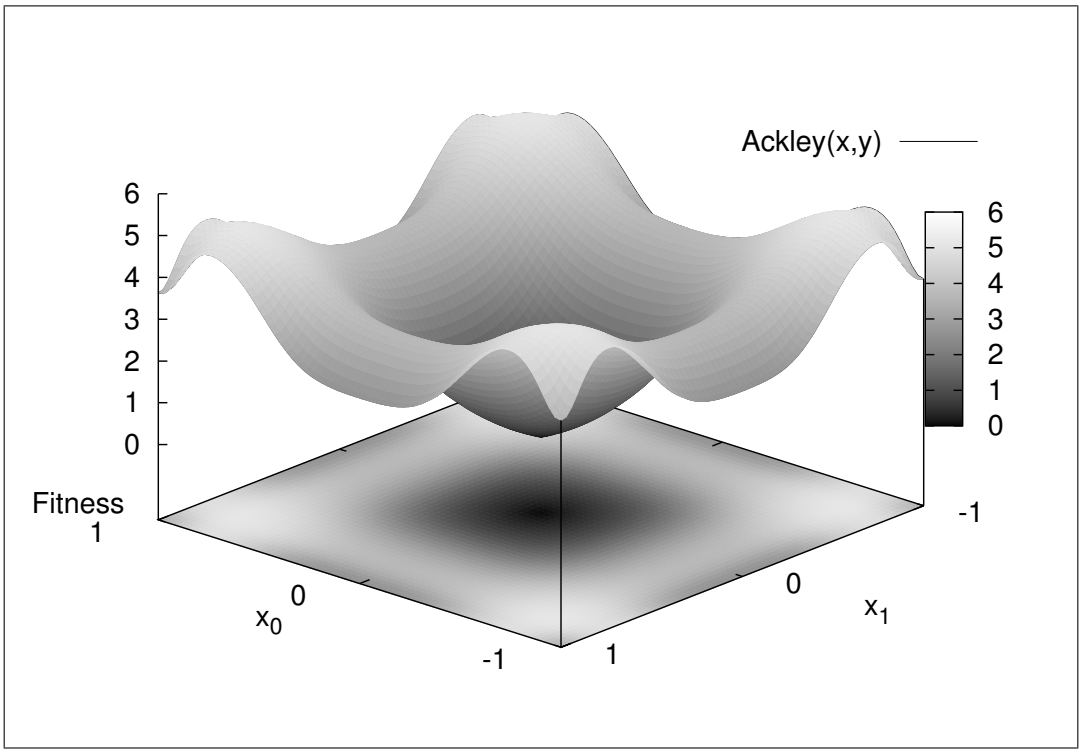

Figure 7.16: 3-D plot of F_5, the Ackley Function, range $[-1, 1]$.

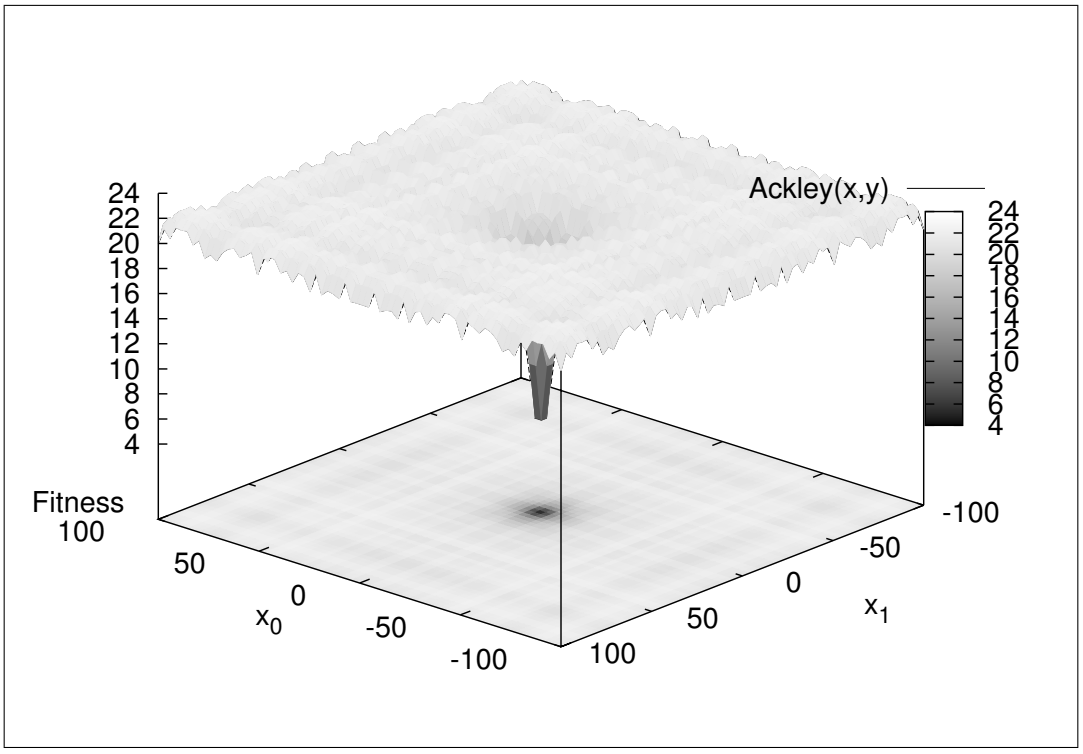

Figure 7.17: 3-D plot of F_5, the Ackley Function, range $[-100, 100]$.

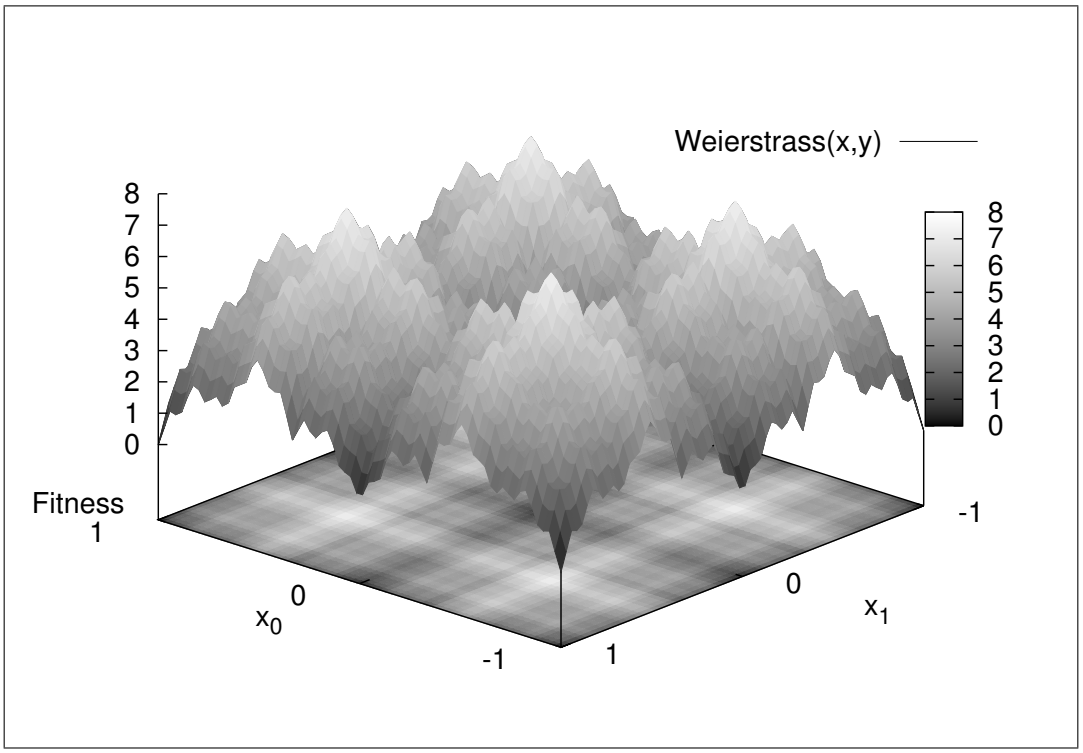

Figure 7.18: 3-D plot of F_6, the Weierstrass Function, range $[-1, 1]$.

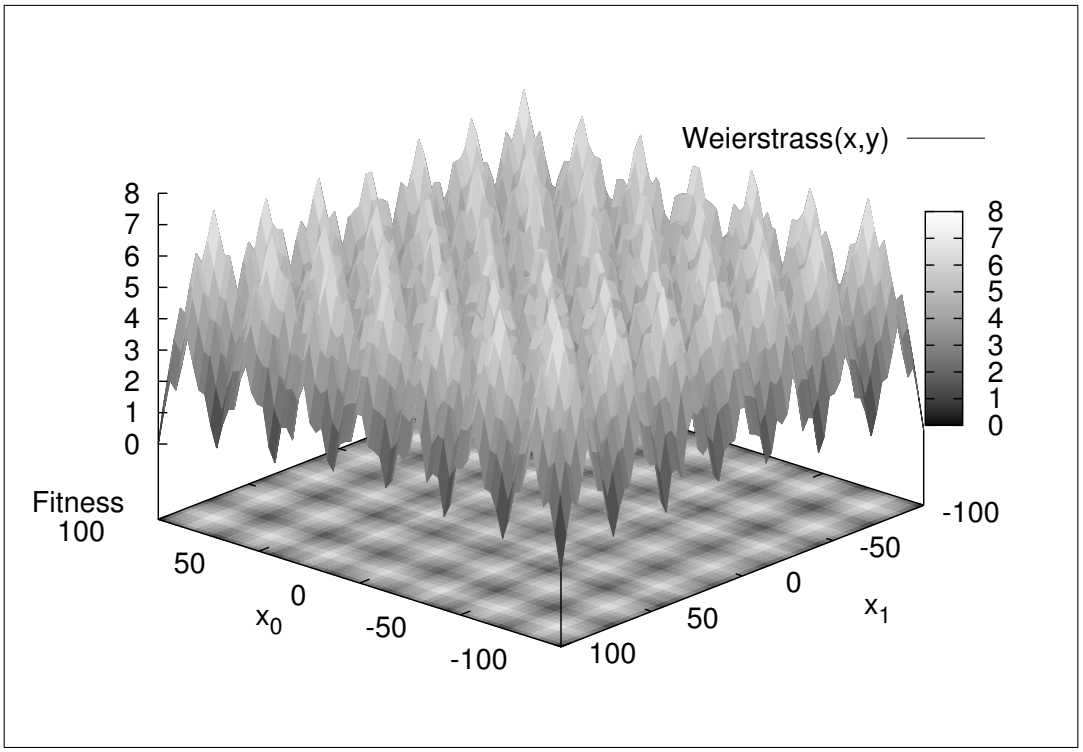

Figure 7.19: 3-D plot of F_6, the Weierstrass Function, range $[-100, 100]$.

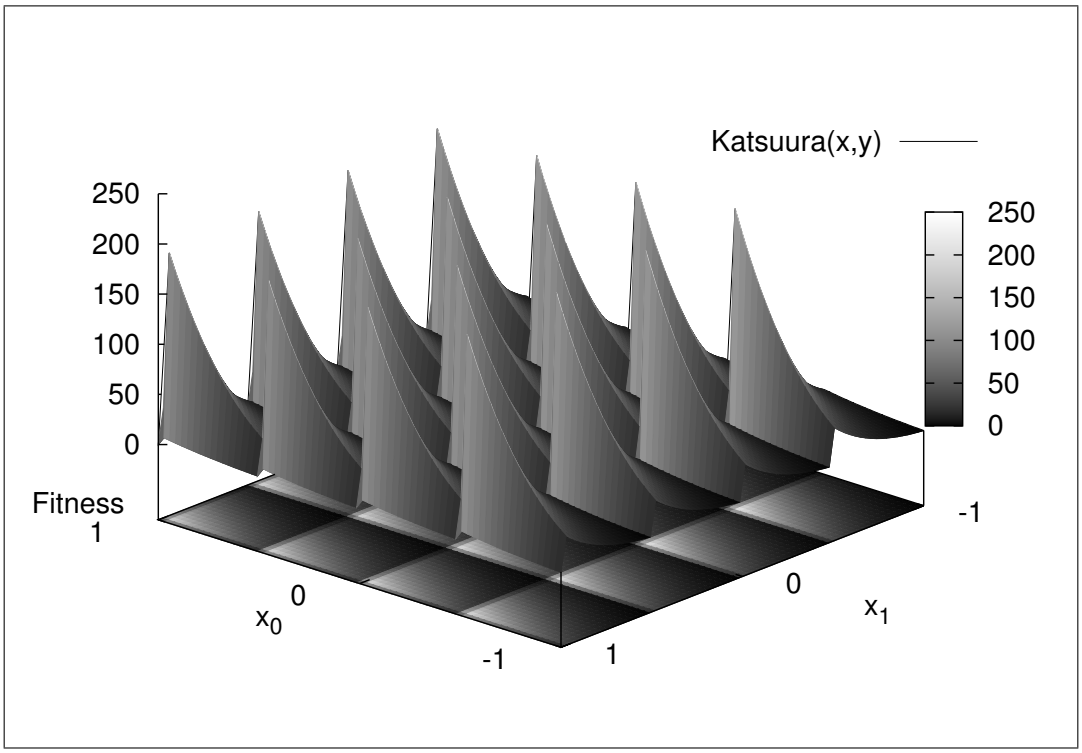

Figure 7.20: 3-D plot of F_7, the Katsuura Function, range $[-1, 1]$.

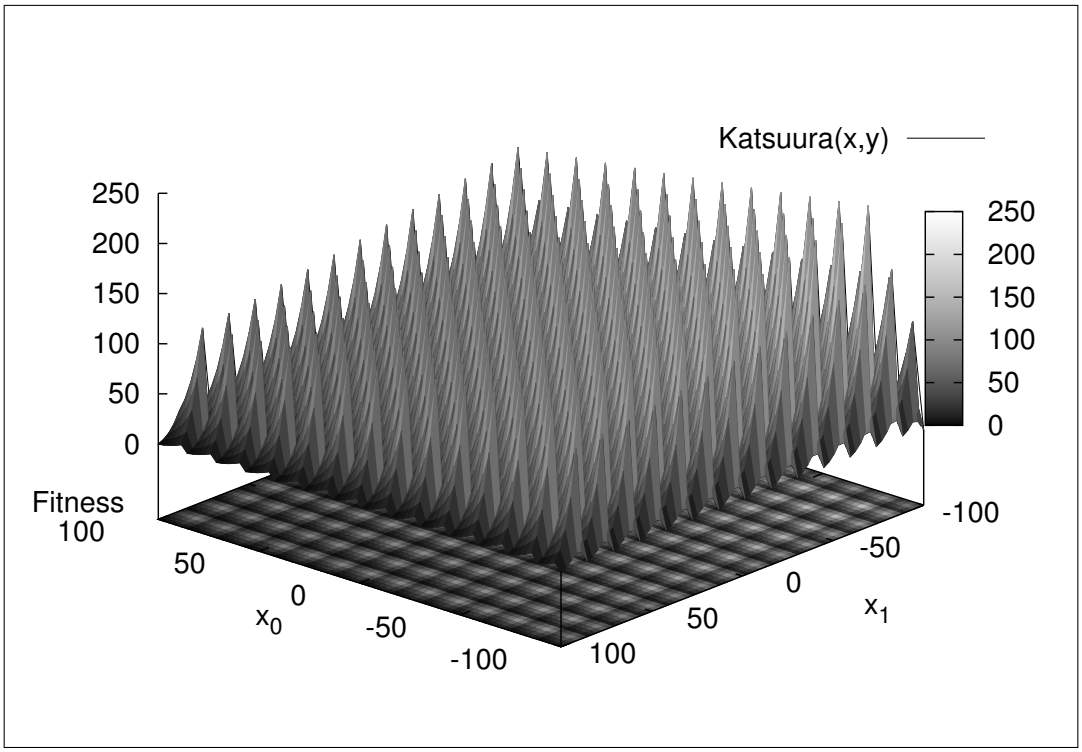

Figure 7.21: 3-D plot of F_7, the Katsuura Function, range $[-100, 100]$.

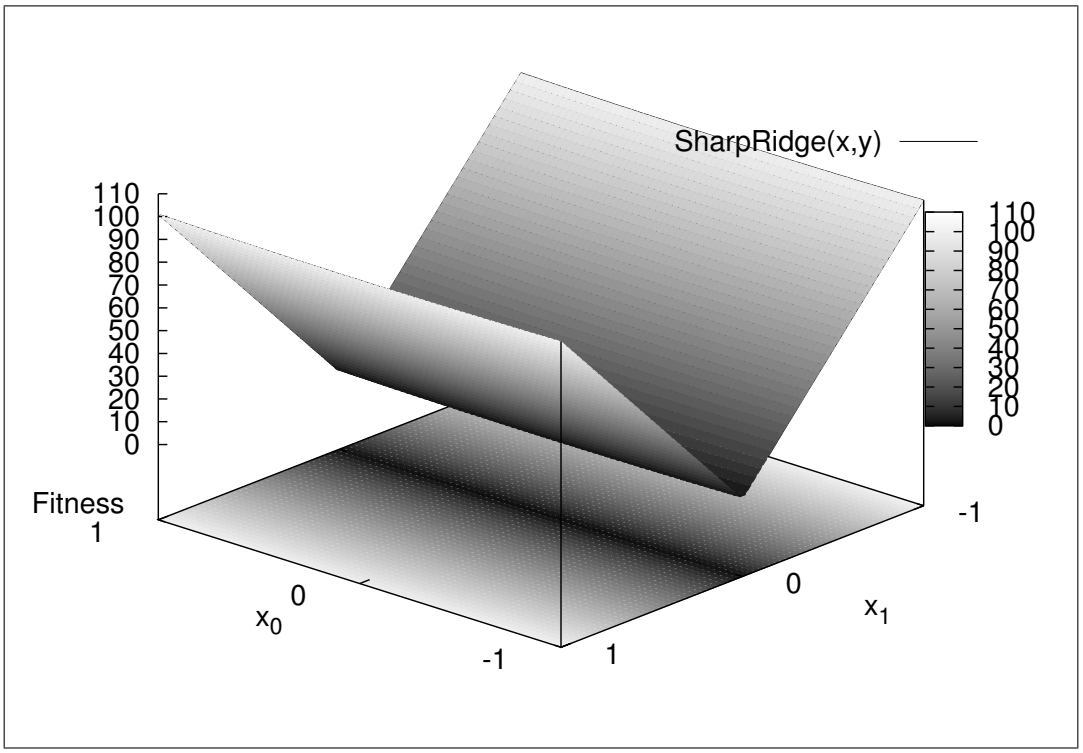

Figure 7.22: 3-D plot of F_8, the Sharp Ridge Function, range $[-1, 1]$.

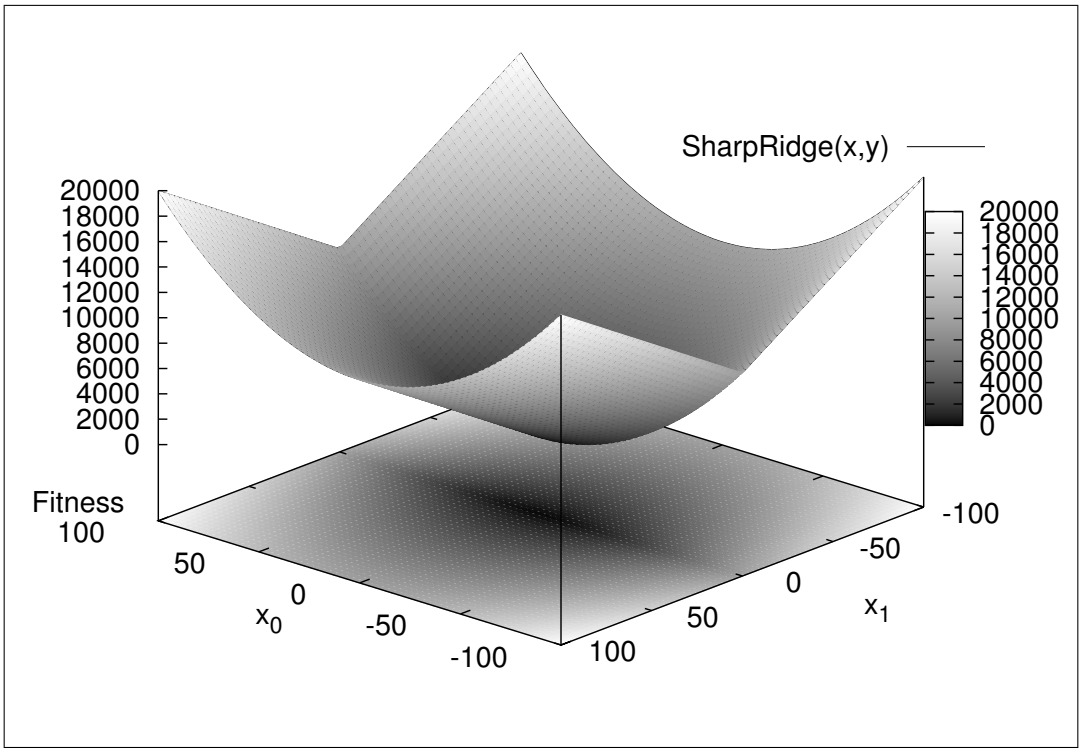

Figure 7.23: 3-D plot of F_8, the Sharp Ridge Function, range $[-100, 100]$.

Table 7.2: Composition of the Fitness Functions.

Functions	Base-1	Base-2	Separable	If Overlap
C_1, C_{12}, C_{23}	F_2	F_3	F_1	F_3
C_2, C_{13}, C_{24}	F_4	F_5	F_4	F_3
C_3, C_{14}, C_{25}	F_4	F_6	F_4	F_3
C_4, C_{15}, C_{26}	F_4	F_7	F_4	F_3
C_5, C_{16}, C_{27}	F_4	F_8	F_4	F_3
C_6, C_{17}, C_{28}	F_5	F_6	F_5	F_3
C_7, C_{18}, C_{29}	F_5	F_7	F_5	F_3
C_8, C_{19}, C_{30}	F_5	F_8	F_5	F_3
C_9, C_{20}, C_{31}	F_6	F_7	F_6	F_3
C_{10}, C_{21}, C_{32}	F_6	F_8	F_6	F_3
C_{11}, C_{22}, C_{33}	F_7	F_8	F_1	F_3

Table 7.2 shows the application of the base functions from Table 7.1 after being fit into the aforementioned templates as follows:

1. **Non-overlapping linkage groups**: $C_1 - C_{11}$.

2. **Minimal overlapping linkage groups**: $C_{12} - C_{22}$.

3. **Increasing overlapping linkage groups**: $C_{23} - C_{33}$.

F_2 and F_3, in their role as base functions, are only paired together and form the functions C_1, C_{12}, C_{23}. These two functions are the only ones which are naturally fully non-separable. Furthermore, F_3 is used as an overlapping function. $F_4 - F_8$ appear in every possible permutation. These functions have been made non-separable into the defined linkage sizes with the aforementioned random coordinate rotation technique. With 8 base functions forming 33 composite fitness functions, the OVLB benchmark strives to keep a balance between a reasonable function size in terms of computability, and variety of diverse function properties and combinations with regards to providing a reasonable encompassing benchmark of functions with overlapping linkage groups of different sizes. While the no free lunch theorem [93]

states that no algorithm will be better than another averaged over *all* possible problems, it can be of tremendous benefit to know the performance, the strengths and the weaknesses of a particular optimization algorithm on some or all of the OVLB functions to extrapolate this onto real-world problems with overlapping linkage groups of varying size that have similar functional properties.

7.2.4 Parameter and Configuration

The OVLB has one parameter to be set a priori, which is the desired number of base linkage groups. Through this, the dimensionality of all composite functions $C_1 - C_{33}$ is determined as well as the actual overlap amount. Starting from zero and up to and including the desired number of base linkage groups, the powers of two with this as exponent are calculated; these numbers determine the size of the base linkage groups. Furthermore, after all linkage group sizes are calculated, $\lfloor \frac{1}{10} * D \rfloor$ expresses the number of genes on which only the separable function may be evaluated. To give an example, if the parameter value is set to 4, a benchmark will be created with the dimensionality of 16, with base linkage groups of sizes $1, 2, 4, 8$ (each having its distinct and disjoint group of genes) accounting for 15 genes and the one extra gene, which will be evaluated by the separable function. Table 7.3 shows, for each parameter value, the dimensionality of the part of the chromosome evaluated by the base functions, the dimensionality of the separable part and the final dimension D.

After the templates have been filled in and the composite functions $C_1 - C_{33}$ have been set up, the OVLB applies the same code as the LSGO to create a shift vector in order to shift the global optimum for each gene and also creates a permutation vector to create a shuffled access to the genes from inside the composite fitness functions. By doing so, positional bias is excluded (otherwise, genes in close proximity could be always assumed to have higher probability to interact with each other) and the global optimum can be anywhere within allowable boundaries while still operating with well studied and understood base fitness functions. In addition to that, the same rotation matrix technique from the LSGO is used to

Table 7.3: OVLB Benchmark Dimensionalities.

Parameter	Base Function 1 + Base Function 2	Separable Function	Final D
1	1	0	1
2	3	0	3
3	7	0	7
4	15	1	16
5	31	3	34
6	63	6	69
7	127	12	139
8	255	25	280
9	511	51	562
10	1023	102	1125

create non-separable versions of separable functions in order to fill the templates. To be able to repeat the experiments, a fixed seed for the random number generator can be provided, so that the same templates will appear every time the benchmark is recreated.

The proposed and pre-defined settings for the OVLB are:

1. **Problem Range**: $[-100, 100]$

2. **Maximal number of fitness evaluations**: 3E+6 for each composite fitness function C_i when parameter value 10 is chosen ($D = 1125$), similar to the LSGO benchmark.

3. **Independent Runs**: 50 runs are performed per fitness function C_i, each run recording the mean and the standard deviation of the best solutions found every generation.

4. **Comparison of two optimizers**: As proposed in [20], non-parametric test methods such as the Wilcoxon rank sum test should be applied to answer the question "Is algorithm A statistically significantly better than B on C_i of the OVLB?" based on the data gathered from a sufficient number of independent runs of each optimizer.

Chapter 8

Experimental Results and Analysis

In this Chapter, the results and analysis of the experimental setups for the surrogate-assisted perturbation check for nonlinearity (SA-LINC-R), the sensitivity detection through machine learning methods and the application of the presented surrogate-assisted crossover operators are given. The linkage results include precision, recall, F-measure and an empirically computed probability of discovered interactions of gene pairings for each generation. In terms of the detection of influential genes, Spearman's rank correlation coefficient towards the true ranking and both the ranking averages as well as two probabilities denoting the appearance of genes in their correct place are provided. The results of the evolutionary optimization runs are given along with plots and tests for statistical significance.

8.1 Linkage Detection

This Section contains the linkage detection results of the SA-LINC-R approach described in Chapter 6. To give an impression of how k-means clusters a matrix with interaction results obtained from the perturbation checks, first an example is given to visualize this.

8.1.1 Clustering the DSM

To give an overview of how k-means can smoothen the results in the MLM_{global} and provide a binary linkage decision, Table 8.1 shows an example of a MLM_{global} calculated from a run

Table 8.1: MLM_{global} on an example run with the Configuration Function 1 in generation ten.

	x_0	x_1	x_2	x_3	x_4	x_5
x_0	-	**0.41**	**0.26**	0.05	0.06	0.05
x_1	-	-	**0.22**	0.04	0.05	0.04
x_2	-	-	-	0.04	0.04	0.04
x_3	-	-	-	-	0.06	0.05
x_4	-	-	-	-	-	0.03
x_5	-	-	-	-	-	-

Table 8.2: DSM created from the MLM_{global} in Table 8.1 through k-means clustering.

	x_0	x_1	x_2	x_3	x_4	x_5
x_0	-	**1**	**1**	0	0	0
x_1	-	-	**1**	0	0	0
x_2	-	-	-	0	0	0
x_3	-	-	-	-	0	0
x_4	-	-	-	-	-	-
x_5	-	-	-	-	-	-

on Configuration Function 1 from generation ten, where the first half of the chromosome is evaluated on the Schwefel 1.2 Function, and the second half on the Sphere Function. Table 8.2 shows the resulting DSM. The genes x_0, x_1 and x_2 all have interactions with each other, while the remaining genes do not. As long as the estimation for linkage is significantly higher than the estimation for genes that do not interact, k-means will create the DSM accordingly. However, one limitation of this approach is that if the fitness function consists of two (or more) different subfunctions, each having linkage for all their genes, but one subfunction provides substantially stronger interaction values from SA-LINC-R than the others, then the genes that are evaluated on the subfunctions with the weaker linkage results would be classified as non-interacting.

8.1.2 Linkage Detection Results

Precision, recall and F-measure are measures from the field of information retrieval [92] that are used quantify the performance of information retrieval algorithms. With regards to linkage learning, these measures have been applied in [8] and their notion is adopted here in the same fashion as shown in Equations 8.1, 8.2 and 8.3.

$$Precision = \frac{True\ interactions\ discovered}{Total\ interactions\ discovered} \tag{8.1}$$

$$Recall = \frac{True\ interactions\ discovered}{Total\ true\ interactions\ existing} \tag{8.2}$$

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{8.3}$$

Precision provides a way to express how much meaningful linkage has been discovered in relation to all detected linkage (including wrongfully detected interactions), and recall can be viewed to describe how much of the existing linkage of a problem has been found. The F-measure combines precision and recall into a single measure. All three measures have the range $[0, 1]$, with 1 being the best, and the F-measure value of 1 can be only achieved if both precision and recall are equal to 1.

In addition to providing averages of these measures for each generation with every configuration, probability results for every possible gene-pair check are given that tell how often (based on the 50 runs) a particular interaction was detected. For instance, Table 8.3 has a value of 0.98 for combination x_0x_2 in generation 2. This means that this interaction was detected in 49 out of 50 runs.

Probability results for Configurations 1, 2 and 3 (half the chromosome is evaluated on a non-separable function, the other half on the separable Sphere function) are presented in Tables 8.3, 8.5 and 8.7. The Tables containing precision, recall and the F-measure for these

configurations are 8.4, 8.6 and 8.8. On all three configurations, the F-measure is either close to or equal to one after a few generations. Configuration 3 includes the Ackley Function, which initially poses a harder challenge, as can be seen in the first generation, where an F-measure of only 0.665 is obtained, and especially the precision of 0.567 contributes to this low value. Subsequent generations are able to improve the F-measure to 1.

Probability results for Configurations 4, 5 and 6 and the information retrieval measures can be found in Tables 8.9, 8.11, 8.13 and Tables 8.10, 8.12 and 8.14, respectively. Although both the Rosenbrock and Ackley Configurations do not result in a perfect F-measure, the values of 0.988 and 0.989 in generation 10 get very close.

Finally, Tables 8.15, 8.17 and 8.19 show the probabilities for Configurations 7, 8 and 9, and Tables 8.16, 8.18 and 8.20 show the corresponding precision, recall and F-measure values. While interactions from the Schwefel 1.2 Function are found with an F-measure of 1 from the third generation onward, this measure goes close to 1 (precision, recall and F-measure are all above 0.95) but does not quite reach it for Configurations 8 and 9.

Table 8.3: Linkage Probabilities in Dimension 6 for Configuration 1.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	1.00	1.00	0.00	0.00	0.00	0.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	1.00	0.98	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
True	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0

Table 8.4: Precision, Recall and F-measure results for Configuration 1.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	1.000	0.980	0.988	0.048
2	1.000	0.993	0.996	0.028
3	1.000	1.000	1.000	0.000
4	1.000	1.000	1.000	0.000
5	1.000	1.000	1.000	0.000
6	1.000	1.000	1.000	0.000
7	1.000	1.000	1.000	0.000
8	1.000	1.000	1.000	0.000
9	1.000	1.000	1.000	0.000
10	1.000	1.000	1.000	0.000

Table 8.5: Linkage Probabilities in Dimension 6 for Configuration 2.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.98	0.04	0.00	0.00	0.04	0.98	0.00	0.00	0.06	0.02	0.02	0.02	0.00	0.02	0.00
2	1.00	0.08	0.00	0.00	0.06	1.00	0.02	0.00	0.06	0.02	0.02	0.06	0.02	0.02	0.00
3	1.00	0.02	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00
4	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00
5	1.00	0.02	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00
6	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
True	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Table 8.6: Precision, Recall and F-measure results for Configuration 2.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.948	0.980	0.977	0.097
2	0.931	1.000	0.951	0.140
3	0.987	1.000	0.992	0.040
4	0.993	1.000	0.996	0.028
5	0.987	1.000	0.992	0.040
6	1.000	1.000	1.000	0.000
7	1.000	1.000	1.000	0.000
8	1.000	1.000	1.000	0.000
9	1.000	1.000	1.000	0.000
10	1.000	1.000	1.000	0.000

Table 8.7: Linkage Probabilities in Dimension 6 for Configuration 3.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.78	0.88	0.10	0.12	0.14	0.82	0.08	0.20	0.24	0.14	0.20	0.22	0.24	0.26	0.38
2	0.86	0.92	0.08	0.06	0.10	0.82	0.10	0.06	0.08	0.02	0.06	0.06	0.12	0.12	0.22
3	0.92	0.92	0.02	0.00	0.00	0.82	0.02	0.02	0.02	0.00	0.00	0.00	0.00	0.04	0.02
4	0.98	0.96	0.00	0.00	0.00	0.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
5	1.00	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	1.00	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
True	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0

Table 8.8: Precision, Recall and F-measure results for Configuration 3.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.567	0.827	0.665	0.160
2	0.803	0.867	0.791	0.154
3	0.976	0.887	0.913	0.139
4	0.995	0.967	0.975	0.086
5	1.000	0.993	0.996	0.028
6	1.000	1.000	1.000	0.000
7	1.000	0.993	0.996	0.028
8	1.000	1.000	1.000	0.000
9	1.000	1.000	1.000	0.000
10	1.000	1.000	1.000	0.000

Table 8.9: Linkage Probabilities in Dimension 6 for Configuration 4.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.96	0.84	0.20	0.18	0.04	0.72	0.12	0.08	0.02	0.04	0.04	0.00	1.00	0.84	0.76
2	1.00	0.98	0.06	0.02	0.00	0.94	0.02	0.02	0.00	0.00	0.00	0.00	1.00	1.00	0.96
3	1.00	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
4	1.00	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
5	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
6	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
7	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
8	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
9	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
10	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
True	1	1	0	0	0	1	0	0	0	0	0	0	1	1	1

Table 8.10: Precision, Recall and F-measure results for Configuration 4.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.886	0.853	0.866	0.230
2	0.983	0.980	0.980	0.047
3	1.000	0.997	0.998	0.013
4	1.000	0.997	0.998	0.013
5	1.000	1.000	1.000	0.000
6	1.000	1.000	1.000	0.000
7	1.000	1.000	1.000	0.000
8	1.000	1.000	1.000	0.000
9	1.000	1.000	1.000	0.000
10	1.000	1.000	1.000	0.000

Table 8.11: Linkage Probabilities in Dimension 6 for Configuration 5.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.90	0.16	0.10	0.18	0.10	0.72	0.20	0.30	0.20	0.10	0.22	0.08	0.84	0.08	0.80
2	0.88	0.10	0.06	0.14	0.04	0.84	0.06	0.16	0.10	0.04	0.10	0.06	0.88	0.04	0.86
3	0.90	0.04	0.02	0.04	0.00	0.88	0.02	0.10	0.06	0.00	0.00	0.02	0.92	0.02	0.96
4	0.96	0.00	0.00	0.00	0.00	0.96	0.00	0.04	0.04	0.00	0.00	0.00	1.00	0.00	0.98
5	0.98	0.00	0.00	0.00	0.00	0.96	0.00	0.02	0.02	0.00	0.00	0.00	0.98	0.00	0.96
6	1.00	0.00	0.00	0.00	0.00	0.98	0.00	0.02	0.00	0.00	0.00	0.00	0.98	0.00	0.96
7	1.00	0.00	0.00	0.00	0.00	0.96	0.00	0.02	0.00	0.00	0.00	0.00	0.98	0.00	0.96
8	0.96	0.00	0.00	0.00	0.00	0.96	0.00	0.02	0.00	0.00	0.00	0.00	0.98	0.00	0.98
9	0.98	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.98
10	0.98	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.98
True	1	0	0	0	0	1	0	0	0	0	0	0	1	0	1

Table 8.12: Precision, Recall and F-measure results for Configuration 5.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.705	0.815	0.739	0.216
2	0.843	0.865	0.835	0.176
3	0.940	0.915	0.912	0.148
4	0.984	0.975	0.976	0.063
5	0.992	0.970	0.977	0.073
6	0.996	0.980	0.985	0.056
7	0.996	0.975	0.983	0.059
8	0.996	0.970	0.979	0.072
9	1.000	0.980	0.988	0.054
10	1.000	0.980	0.988	0.054

Table 8.13: Linkage Probabilities in Dimension 6 for Configuration 6.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.62	0.56	0.24	0.20	0.22	0.66	0.34	0.38	0.30	0.32	0.18	0.30	0.56	0.58	0.58
2	0.72	0.80	0.28	0.12	0.12	0.80	0.24	0.22	0.28	0.24	0.10	0.24	0.74	0.72	0.72
3	0.82	0.84	0.10	0.10	0.08	0.88	0.10	0.06	0.06	0.14	0.06	0.14	0.82	0.76	0.78
4	0.86	0.84	0.02	0.04	0.00	0.86	0.06	0.04	0.04	0.08	0.04	0.04	0.86	0.88	0.84
5	0.90	0.92	0.00	0.02	0.00	0.90	0.02	0.02	0.02	0.08	0.04	0.00	0.86	0.90	0.86
6	0.96	0.94	0.00	0.02	0.00	0.98	0.00	0.00	0.00	0.04	0.00	0.00	0.88	0.92	0.86
7	0.98	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.02	0.00	0.00	0.94	0.96	0.92
8	0.96	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.94	0.92
9	0.94	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.98	0.96
10	0.96	1.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.98	1.00	0.96
True	1	1	0	0	0	1	0	0	0	0	0	0	1	1	1

Table 8.14: Precision, Recall and F-measure results for Configuration 6.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.625	0.593	0.578	0.163
2	0.757	0.750	0.727	0.132
3	0.876	0.817	0.832	0.140
4	0.940	0.857	0.890	0.120
5	0.969	0.890	0.922	0.089
6	0.991	0.923	0.951	0.071
7	0.997	0.963	0.977	0.064
8	1.000	0.963	0.979	0.053
9	1.000	0.973	0.985	0.040
10	1.000	0.980	0.989	0.037

Table 8.15: Linkage Probabilities in Dimension 6 for Configuration 7.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.88	0.04	0.04	0.08	0.02	0.04	0.04	0.06	0.02	0.88	0.10	0.06	0.00	0.00	0.90
2	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.98
3	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
4	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
5	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
6	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
7	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
8	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
9	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
10	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00
True	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1

Table 8.16: Precision, Recall and F-measure results for Configuration 7.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.888	0.887	0.872	0.181
2	1.000	0.987	0.992	0.040
3	1.000	1.000	1.000	0.000
4	1.000	1.000	1.000	0.000
5	1.000	1.000	1.000	0.000
6	1.000	1.000	1.000	0.000
7	1.000	1.000	1.000	0.000
8	1.000	1.000	1.000	0.000
9	1.000	1.000	1.000	0.000
10	1.000	1.000	1.000	0.000

Table 8.17: Linkage Probabilities in Dimension 6 for Configuration 8.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.84	0.10	0.18	0.12	0.08	0.18	0.18	0.14	0.16	0.86	0.12	0.16	0.12	0.20	0.94
2	0.94	0.02	0.02	0.02	0.00	0.08	0.06	0.04	0.04	0.98	0.04	0.04	0.06	0.04	1.00
3	0.96	0.00	0.00	0.02	0.00	0.04	0.02	0.00	0.02	0.94	0.00	0.02	0.00	0.02	0.98
4	0.98	0.00	0.00	0.00	0.00	0.02	0.02	0.00	0.02	0.96	0.00	0.00	0.00	0.02	0.98
5	0.98	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.02	0.98	0.00	0.00	0.00	0.00	0.98
6	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.98	0.00	0.00	0.00	0.00	0.98
7	0.96	0.02	0.02	0.00	0.00	0.02	0.02	0.00	0.02	1.00	0.00	0.00	0.00	0.00	0.98
8	0.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	1.00	0.00	0.00	0.00	0.00	0.98
9	0.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	1.00	0.00	0.00	0.00	0.00	0.98
10	0.96	0.02	0.04	0.00	0.00	0.02	0.04	0.00	0.02	0.98	0.00	0.00	0.00	0.00	0.98
True	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1

Table 8.18: Precision, Recall and F-measure results for Configuration 8.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.708	0.880	0.755	0.217
2	0.924	0.973	0.937	0.137
3	0.968	0.960	0.958	0.090
4	0.982	0.973	0.973	0.068
5	0.990	0.980	0.982	0.054
6	0.995	0.980	0.985	0.051
7	0.984	0.980	0.977	0.075
8	0.995	0.980	0.985	0.051
9	0.995	0.980	0.985	0.051
10	0.976	0.973	0.968	0.085

Table 8.19: Linkage Probabilities in Dimension 6 for Configuration 9.

Gen.	x_0x_1	x_0x_2	x_0x_3	x_0x_4	x_0x_5	x_1x_2	x_1x_3	x_1x_4	x_1x_5	x_2x_3	x_2x_4	x_2x_5	x_3x_4	x_3x_5	x_4x_5
1	0.82	0.26	0.32	0.34	0.34	0.22	0.26	0.28	0.24	0.88	0.26	0.30	0.30	0.22	0.74
2	0.90	0.10	0.10	0.18	0.20	0.14	0.12	0.14	0.14	0.94	0.10	0.18	0.20	0.14	0.94
3	0.88	0.08	0.10	0.08	0.08	0.08	0.08	0.04	0.04	0.92	0.06	0.12	0.10	0.12	0.90
4	0.88	0.06	0.06	0.02	0.00	0.00	0.06	0.00	0.00	0.96	0.00	0.00	0.02	0.04	0.92
5	0.88	0.02	0.04	0.00	0.02	0.02	0.04	0.00	0.02	0.94	0.00	0.00	0.02	0.02	0.90
6	0.82	0.04	0.04	0.00	0.02	0.04	0.04	0.00	0.02	0.94	0.00	0.00	0.02	0.00	0.90
7	0.92	0.02	0.04	0.02	0.04	0.04	0.04	0.02	0.04	0.96	0.00	0.00	0.02	0.00	0.90
8	0.96	0.02	0.04	0.00	0.02	0.04	0.02	0.00	0.02	0.98	0.00	0.00	0.02	0.00	0.90
9	0.96	0.02	0.02	0.00	0.00	0.00	0.02	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.94
10	0.96	0.02	0.00	0.00	0.00	0.02	0.02	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.96
True	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1

Table 8.20: Precision, Recall and F-measure results for Configuration 9.

Gen.	Precision mean	Recall mean	F-measure mean	F-measure std.dev.
1	0.482	0.813	0.582	0.205
2	0.707	0.927	0.775	0.182
3	0.861	0.900	0.847	0.183
4	0.942	0.920	0.917	0.123
5	0.954	0.907	0.916	0.122
6	0.951	0.887	0.902	0.125
7	0.945	0.927	0.923	0.117
8	0.961	0.947	0.946	0.103
9	0.987	0.960	0.968	0.075
10	0.985	0.973	0.975	0.062

8.2 Sensitivity Detection

The average ranking results were calculated as follows: Each configuration was run independently for 50 times and 50 individual rankings were obtained. For each variable, its ranks from each detection run were added and then divided by 50 to give an average rank that is displayed in the tables. It should be noted that while this method provides an overview of how, on average, the three machine learning methods were judging each variable, it smoothens the results such that for example even if the most influential variable was not always identified as such (with rank 1), it suffices that on average it was ranked as the best. The means and standard deviations of the Spearman's rank correlation coefficient for each of the configurations have been therefore calculated to measure how close the obtained rankings from the 50 runs came to the true, known ranking of gene sensitivities. In addition to that, the tables provide a probability that is based on how often a gene appeared in its correct ranking, and a second probability that also allows neighboring ranks per gene to be counted as right position to fully display the detection of gene sensitivities.

8.2.1 Sensitivity Averages

Tables 8.21 and 8.22 show the results of the final ranking averages on functions F_1 and F_2 with dimensionality four. The ordering is sorted by the average descending order of strength of contribution of the gene towards the fitness of the chromosome, and equals to its average rank. In this low dimensionality, all methods, on average, discovered the sensitivities perfectly. In Tables 8.23 and 8.24, the results of the final ranking averages on functions F_1 and F_2 with dimensionality 16 are shown. The neural network approach is able to almost reliably capture the sensitivity information, followed by the SMOreg approach. Especially with the ReliefF results with F_1 it can be seen that as the sensitivity ranking gets larger (and therefore the genes are getting less influential), the precision of the approach decreases - a trend that is even more visible in dimension 32. Finally, Tables 8.25 and 8.26 give the

Table 8.21: Sensitivity Detection Results in Dimension 4 for F_1.

Config.	Approach	Average Ranking (descending order of strength)
	True Ranking	$3, 2, 1, 0$
1	Neural Network	$3, 2, 1, 0$
	ReliefF	$3, 2, 1, 0$
	SMOreg	$3, 2, 1, 0$
2	Neural Network	$3, 2, 1, 0$
	ReliefF	$3, 2, 1, 0$
	SMOreg	$3, 2, 1, 0$

Table 8.22: Sensitivity Detection Results in Dimension 4 for F_2.

Config.	Approach	Average Ranking (descending order of strength)
	True Ranking	$0, 1, 2, 3$
7	Neural Network	$0, 1, 2, 3$
	ReliefF	$0, 1, 2, 3$
	SMOreg	$0, 1, 2, 3$
8	Neural Network	$0, 1, 2, 3$
	ReliefF	$0, 1, 2, 3$
	SMOreg	$0, 1, 2, 3$

results of the final ranking averages on functions F_1 and F_2 with dimensionality 32. The neural network misplaces 0 and 1 in Table 8.26 Configuration 12. It can also be observed that in this dimensionality, the rankings of mid and low influential genes are hard to distinguish for all three approaches, whereas the genes with the strongest sensitivity influence can be mostly reliably detected and ranked.

Table 8.23: Sensitivity Detection Results in Dimension 16 for F_1.

Config.	Approach	Average Ranking (descending order of strength)
	True Ranking	$15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0$
3	Neural Network	$15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0$
	ReliefF	$15, 14, 13, 12, 11, 10, 9, 6, 0, 2, 4, 3, 8, 1, 5, 7$
	SMOreg	$15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 4, 1, 5, 2, 0, 3$
4	Neural Network	$15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0$
	ReliefF	$15, 14, 13, 12, 11, 10, 9, 3, 8, 4, 6, 5, 1, 2, 0, 7$
	SMOreg	$15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 3, 5, 1, 2, 4, 0$

Table 8.24: Sensitivity Detection Results in Dimension 16 for F_2.

Config.	Approach	Average Ranking (descending order of strength)
	True Ranking	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$
9	Neural Network	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$
	ReliefF	$1, 0, 2, 4, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 14$
	SMOreg	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$
10	Neural Network	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$
	ReliefF	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$
	SMOreg	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$

Table 8.25: Sensitivity Detection Results in Dimension 32 for F_1.

Config.	Approach	Average Ranking (descending order of strength)
	True Ranking	31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
5	Neural Network	31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 17, 9, 1, 18, 0, 6, 2, 7, 19, 14, 11, 8, 16, 10, 4, 12, 15, 3, 5, 13
	ReliefF	31, 30, 29, 28, 27, 26, 6, 24, 2, 13, 19, 11, 18, 25, 12, 22, 9, 8, 14, 4, 21, 17, 7, 23, 15, 20, 0, 1, 3, 16, 10, 5
	SMOreg	31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 19, 4, 15, 6, 5, 9, 3, 14, 17, 18, 16, 12, 20, 13, 0, 1, 2, 8, 10, 7, 11
6	Neural Network	31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 19, 12, 20, 0, 18, 11, 9, 14, 5, 8, 15, 4, 16, 1, 13, 7, 10, 6, 2, 3, 17
	ReliefF	31, 30, 29, 28, 27, 26, 25, 24, 5, 17, 11, 7, 22, 1, 16, 9, 19, 6, 14, 20, 2, 21, 0, 4, 23, 3, 10, 8, 15, 12, 13, 18
	SMOreg	31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 19, 21, 15, 4, 11, 8, 14, 20, 16, 1, 17, 3, 5, 0, 2, 10, 9, 18, 12, 6, 13, 7

Table 8.26: Sensitivity Detection Results in Dimension 32 for F_2.

Config.	Approach	Average Ranking (descending order of strength)
	True Ranking	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
11	Neural Network	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
	ReliefF	2, 1, 0, 4, 5, 6, 7, 9, 10, 8, 11, 13, 12, 14, 15, 16, 17, 19, 18, 20, 22, 21, 23, 24, 25, 28, 27, 26, 30, 29, 31
	SMOreg	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
12	Neural Network	1, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
	ReliefF	0, 1, 2, 4, 5, 3, 6, 8, 9, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 22, 24, 25, 26, 28, 27, 29, 31, 30
	SMOreg	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31

8.2.2 Spearman's ρ and Sensitivity Probabilities

Spearman's rank correlation coefficient, also known as Spearman's ρ, is described in [91]. It is a non-parametric test that can be used to investigate the "correlation between two ranked variables" [91]. In the context of this work, it can be applied to determine how close a sensitivity ranking of genes by a machine learning method compares to the true ranking (the true ranking, in these experiments, is known beforehand). If no ties in the true and observed rankings are present, the calculation of Spearman's rank correlation coefficient between the two rankings given in variables x and y can be done as shown in Equation 8.4.

$$\rho = 1 - \frac{6 \sum_{i-1}^{n} d_i^2}{n(n^2 - 1)} \; where \; d_i = x_i - y_i. \tag{8.4}$$

The range of ρ is $[-1, 1]$, with 1 being a perfect match between the detected and the true sensitivity ranking. Table 8.27 shows the mean and standard deviation of ρ (it was calculated for each of the 50 independent runs) for the neural network, the SMOreg and the ReliefF results on F_1 for both dimensionalities. In dimension 4, the neural network approach obtains a ρ of 1 and 0.988, outperforming the SMOreg and ReliefF methods. As the dimensionality is increased to 16, the neural network again receives the best ρ values of 0.931 and 0.934, but this means that a perfect ranking was not always found. ReliefF already shows a considerably weaker performance with ρ values of 0.646 and 0.690. This trend continues in the results for dimensionalities 32, where ReliefF only achieves a ρ of 0.416 and 0.425. While the highest ρ values again belong to the neural network approach. It can be seen that with the values of 0.694 and 0.706, the detected rankings show inaccuracies.

The results for Function F_2 are given in Table 8.28. In general, the three methods had more success to detect the sensitivities than on F_1, as can be seen with all ρ values. In the dimension 4, the neural network slightly outperforms the SMOreg with a ρ of 0.992 compared to 0.988, and the ReliefF method already drops to a ρ of 0.904. Similar trends can be seen in dimensionalities 16 and 32, with the neural network having the same or very

Table 8.27: Spearman's ρ results for F_1.

Dim.	Range	Neural Network		SMOreg		ReliefF	
		mean	std. dev.	mean	std. dev.	mean	std. dev.
4	[0,1]	**1.000**	0.000	0.992	0.040	0.984	0.055
16	[0,1]	**0.931**	0.039	0.898	0.044	0.646	0.133
32	[0,1]	**0.694**	0.083	0.678	0.090	0.416	0.121
4	[0,100]	**0.988**	0.048	0.976	0.066	0.980	0.061
16	[0,100]	**0.934**	0.036	0.904	0.042	0.690	0.139
32	[0,100]	**0.706**	0.069	0.682	0.084	0.425	0.103

Table 8.28: Spearman's ρ results for F_2.

Dim.	Range	Neural Network		SMOreg		ReliefF	
		mean	std. dev.	mean	std. dev.	mean	std. dev.
4	[0,1]	**0.992**	0.040	0.988	0.048	0.904	0.116
16	[0,1]	**0.998**	0.002	**0.998**	0.003	0.900	0.054
32	[0,1]	**0.999**	0.001	**0.999**	0.001	0.819	0.039
4	[0,100]	0.988	0.048	**0.992**	0.040	0.960	0.081
16	[0,100]	**0.999**	0.002	0.997	0.003	0.900	0.050
32	[0,100]	**0.999**	0.001	**0.999**	0.001	0.838	0.050

close ρ when compared to the SMOreg method, and ReliefF which obtain noticeably lower (but still reasonably high) ρ values.

Another way of looking at the obtained ranking results is to count, for each of the 50 runs of an algorithm on a function, how many times a gene appeared in its correct ranking. For example, looking at F_2 in dimensionality four, the correct ranking in the descending order of strength of contribution should be $0, 1, 2, 3$, indicating a sensitivity order for the genes x_0, x_1, x_2, x_3. If in an example the neural network approach classified gene x_0 in 48 out of 50 times as the best gene, but for two times, it misplaced it as possibly second or even third best, then it can be said that with a probability of $\frac{48}{50} = 0.96$ the gene was ranked correctly. This probability is denoted as P_{NN_0}. Furthermore, a more lenient measure, P_{NN_1}, can be

adopted which allows a gene to be ranked with either its correct true rank or a neighboring rank. If gene x_0 was ranked in its true place correctly for 48 times, and ranked as second most influential gene for 2 times, then its probability to appear correctly is $\frac{50}{50} = 1$. For the most contributing and least contributing gene only one neighboring place is allowed under this measure (second most contributing and penultimate contributing), while the other genes are allowed to be misplaced one rank below or above. This provides a straightforward way of judging the accuracy of the sensitivity detection methods, and in the following tables, the probabilities for all runs on all genes are provided. A higher value in the tables means a stronger probability to detect the gene in its correct rank and is therefore more desirable (higher is better).

Table 8.29 gives the probabilities for F_1 in dimensionality four, and Table 8.30 shows the probabilities for F_2 in the same dimensionality. While the neural network calculated all rankings correctly in F_1, even with the stricter P_{NN_0} measure, ReliefF and SMOreg already show performance differences in the least influential genes. The opposite happens in F_2, where the most influential genes are the hardest to identify. A reason for this can be that there is no interaction among variables in F_1, only an exponent that increases subsequently with the genes, such that the difference between the most influential genes is bigger than that of the least influential ones. F_2, on the other hand, features linkage between its genes, and the more influential a gene is, the more interactions are present between this genes and the others.

In Table 8.31, results of the probabilities for F_1 in dimensionality 16 are presented, and Table 8.32 contains the probabilities for F_2 in dimensionality 16. Here the performance differences between the neural network and ReliefF are clearly visible, especially in P_{NN_1}, where the probabilities for ReliefF degrade quickly. It can be said that on F_2, except for a few most contributing genes, both the neural network and the SMOreg provide a correct ranking based on the probabilities, whereas ReliefF struggles with all genes rankings.

Finally, Table 8.33 has the probabilities for F_1 in dimensionality 32, and Table 8.34 holds

the probabilities for F_2 in dimensionality 32. After approximately half the dimensionality of F_1, the ranking performance of all three methods looses its expressive power, giving a different picture than the previous ranking averages. The exponent in F_1 increases with the number of dimensions, and with 32 genes it could be concluded that the contribution of the strongest genes vastly outperforms the contributions of remaining genes such that the algorithms have problems to distinguish between the ranks of these less influential genes. However, especially with the neural network approach and followed by the SMOreg, the most influential genes in F_1 and a large number of the least influential genes in F_2 are classified with their correct rank with a high probability.

It can be said that these approaches can be used to detect and rank the genes with the high sensitivities on functions like F_1, where the gene contributions are not blurred through interactions. On functions like F_2, where no exponential increase of contribution takes place and genes are interacting more strongly if they feature a higher importance, these approaches were able to detect the least and medium influential genes very well and especially for the neural network and SMOreg, the strong interacting genes were found with a reliable probability. In practice, while an absolute perfect ranking is desirable, it is already beneficial to know the genes approximate ranking, for example to know that a gene is either the most influential or the second most influential.

With obtained sensitivities through machine learning methods, several possibilities to employ this knowledge are open. In [55], a crossover operator is presented that focuses on the most influential genes, while [10] gives the idea of a mutation operator with variable mutation rates based on the genes importance.

Table 8.29: Sensitivity Probabilities in Dimension 4 for F_1.

Config.	Prob.	x_3	x_2	x_1	x_0
1	P_{NN_0}	1.00	1.00	1.00	1.00
	$P_{ReliefF_0}$	1.00	1.00	0.92	0.92
	P_{SMOreg_0}	1.00	0.98	0.96	0.98
	P_{NN_1}	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	1.00	1.00	1.00	1.00
	P_{SMOreg_1}	1.00	1.00	1.00	1.00
2	P_{NN_0}	1.00	1.00	0.94	0.94
	$P_{ReliefF_0}$	1.00	0.98	0.90	0.92
	P_{SMOreg_0}	1.00	1.00	0.88	0.88
	P_{NN_1}	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	1.00	1.00	1.00	1.00
	P_{SMOreg_1}	1.00	1.00	1.00	1.00

Table 8.30: Sensitivity Probabilities in Dimension 4 for F_2.

Config.	Prob.	x_0	x_1	x_2	x_3
7	P_{NN_0}	0.96	0.96	1.00	1.00
	$P_{ReliefF_0}$	0.66	0.60	0.86	0.92
	P_{SMOreg_0}	0.94	0.94	1.00	1.00
	P_{NN_1}	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	1.00	1.00	1.00	0.92
	P_{SMOreg_1}	1.00	1.00	1.00	1.00
8	P_{NN_0}	0.94	0.94	1.00	1.00
	$P_{ReliefF_0}$	0.88	0.80	0.92	1.00
	P_{SMOreg_0}	0.96	0.96	1.00	1.00
	P_{NN_1}	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	1.00	1.00	1.00	1.00
	P_{SMOreg_1}	1.00	1.00	1.00	1.00

Table 8.31: Sensitivity Probabilities in Dimension 16 for F_1.

Config.	Prob.	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
3	P_{NN_0}	1.00	1.00	1.00	1.00	1.00	0.96	0.92	0.66	0.40	0.18	0.16	0.12	0.14	0.10	0.18	0.22
	$P_{ReliefF_0}$	1.00	1.00	1.00	0.74	0.28	0.10	0.04	0.08	0.08	0.04	0.08	0.06	0.10	0.06	0.12	0.12
	P_{SMOreg_0}	1.00	1.00	1.00	1.00	0.98	0.94	0.80	0.62	0.22	0.22	0.12	0.10	0.12	0.14	0.12	0.08
	P_{NN_1}	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.86	0.82	0.50	0.42	0.38	0.38	0.42	0.56	0.44
	$P_{ReliefF_1}$	1.00	1.00	1.00	0.88	0.48	0.30	0.26	0.16	0.14	0.42	0.26	0.16	0.36	0.22	0.34	0.26
	P_{SMOreg_1}	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.86	0.52	0.52	0.46	0.32	0.36	0.48	0.40	0.24
4	P_{NN_0}	1.00	1.00	1.00	1.00	1.00	0.98	0.80	0.68	0.46	0.22	0.22	0.28	0.24	0.18	0.18	0.22
	$P_{ReliefF_0}$	1.00	1.00	1.00	0.86	0.24	0.22	0.14	0.08	0.10	0.14	0.14	0.04	0.06	0.02	0.02	0.12
	P_{SMOreg_0}	1.00	1.00	1.00	1.00	1.00	0.90	0.66	0.48	0.30	0.14	0.16	0.20	0.14	0.20	0.12	0.08
	P_{NN_1}	1.00	1.00	1.00	1.00	1.00	1.00	0.90	0.94	0.84	0.58	0.50	0.52	0.48	0.52	0.56	0.42
	$P_{ReliefF_1}$	1.00	1.00	1.00	0.96	0.56	0.54	0.30	0.18	0.24	0.48	0.30	0.32	0.26	0.38	0.26	0.26
	P_{SMOreg_1}	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.82	0.70	0.46	0.32	0.46	0.34	0.44	0.38	0.28

Table 8.32: Sensitivity Probabilities in Dimension 16 for F_2.

Config.	Prob.	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
9	P_{NN_0}	0.58	0.48	0.86	0.98	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_0}$	0.26	0.16	0.16	0.20	0.22	0.12	0.26	0.24	0.20	0.32	0.18	0.32	0.30	0.16	0.30	0.36
	P_{SMOreg_0}	0.68	0.64	0.74	0.86	0.94	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	P_{NN_1}	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	0.58	0.70	0.52	0.50	0.48	0.36	0.74	0.64	0.56	0.68	0.52	0.62	0.58	0.58	0.82	0.36
	P_{SMOreg_1}	0.92	0.98	0.94	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10	P_{NN_0}	0.70	0.60	0.86	0.98	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_0}$	0.26	0.18	0.18	0.12	0.10	0.20	0.32	0.32	0.28	0.22	0.24	0.20	0.30	0.22	0.32	0.42
	P_{SMOreg_0}	0.62	0.52	0.66	0.90	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	P_{NN_1}	1.00	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	0.50	0.60	0.50	0.46	0.48	0.44	0.60	0.66	0.64	0.58	0.54	0.64	0.62	0.58	0.82	0.42
	P_{SMOreg_1}	0.92	1.00	0.94	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 8.33: Sensitivity Probabilities in Dimension 32 for F_1.

Config.	Prob.	x_{31}	x_{30}	x_{29}	x_{28}	x_{27}	x_{26}	x_{25}	x_{24}	x_{23}	x_{22}	x_{21}	x_{20}	x_{19}	x_{18}	x_{17}	x_{16}
	P_{NN_0}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.88	0.58	0.24	0.24	0.06	0.08	0.06	0.06	0.02
	$P_{ReliefF_0}$	1.00	1.00	1.00	0.86	0.26	0.08	0.04	0.00	0.02	0.08	0.00	0.04	0.02	0.02	0.04	0.00
5	P_{SMOreg_0}	1.00	1.00	1.00	1.00	1.00	1.00	0.90	0.72	0.42	0.26	0.14	0.04	0.02	0.06	0.02	0.08
	P_{NN_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.54	0.40	0.16	0.16	0.14	0.18	0.10
	$P_{ReliefF_1}$	1.00	1.00	1.00	0.98	0.54	0.26	0.12	0.06	0.06	0.14	0.12	0.06	0.14	0.06	0.10	0.04
	P_{SMOreg_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.90	0.68	0.46	0.32	0.14	0.12	0.10	0.20	0.16
		x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	P_{NN_0}	0.02	0.06	0.06	0.02	0.04	0.08	0.08	0.02	0.02	0.04	0.04	0.02	0.08	0.02	0.06	0.04
	$P_{ReliefF_0}$	0.12	0.06	0.06	0.00	0.02	0.04	0.00	0.00	0.06	0.04	0.02	0.00	0.00	0.04	0.06	0.04
5	P_{SMOreg_0}	0.10	0.06	0.04	0.06	0.04	0.06	0.02	0.00	0.04	0.02	0.04	0.04	0.06	0.04	0.04	0.04
	P_{NN_1}	0.08	0.16	0.10	0.10	0.06	0.16	0.16	0.18	0.10	0.14	0.14	0.10	0.12	0.12	0.10	0.08
	$P_{ReliefF_1}$	0.14	0.10	0.16	0.02	0.08	0.12	0.10	0.08	0.16	0.10	0.10	0.10	0.16	0.10	0.22	0.08
	P_{SMOreg_1}	0.22	0.18	0.08	0.22	0.12	0.18	0.14	0.06	0.16	0.16	0.16	0.08	0.10	0.16	0.14	0.08
Config.	Prob.	x_{31}	x_{30}	x_{29}	x_{28}	x_{27}	x_{26}	x_{25}	x_{24}	x_{23}	x_{22}	x_{21}	x_{20}	x_{19}	x_{18}	x_{17}	x_{16}
	P_{NN_0}	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.84	0.68	0.30	0.08	0.06	0.00	0.06	0.02	0.06
	$P_{ReliefF_0}$	1.00	1.00	1.00	0.88	0.32	0.04	0.06	0.06	0.06	0.02	0.02	0.00	0.04	0.00	0.06	0.02
6	P_{SMOreg_0}	1.00	1.00	1.00	1.00	1.00	0.94	0.86	0.68	0.44	0.14	0.08	0.06	0.08	0.06	0.04	0.06
	P_{NN_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.68	0.38	0.26	0.12	0.22	0.08	0.16
	$P_{ReliefF_1}$	1.00	1.00	1.00	1.00	0.60	0.20	0.16	0.14	0.10	0.14	0.12	0.14	0.08	0.08	0.08	0.14
	P_{SMOreg_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.86	0.82	0.44	0.20	0.12	0.20	0.18	0.06	0.16
		x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	P_{NN_0}	0.00	0.08	0.06	0.04	0.08	0.08	0.02	0.08	0.06	0.08	0.04	0.10	0.00	0.04	0.04	0.06
	$P_{ReliefF_0}$	0.00	0.00	0.02	0.06	0.08	0.08	0.02	0.08	0.02	0.10	0.04	0.02	0.02	0.02	0.08	0.00
6	P_{SMOreg_0}	0.10	0.06	0.06	0.06	0.04	0.00	0.06	0.02	0.10	0.04	0.08	0.06	0.12	0.02	0.10	0.04
	P_{NN_1}	0.14	0.16	0.16	0.18	0.22	0.14	0.12	0.16	0.22	0.18	0.22	0.18	0.10	0.16	0.14	0.12
	$P_{ReliefF_1}$	0.10	0.06	0.10	0.12	0.16	0.18	0.08	0.12	0.06	0.18	0.10	0.14	0.12	0.08	0.16	0.02
	P_{SMOreg_1}	0.20	0.20	0.10	0.12	0.16	0.12	0.12	0.08	0.22	0.14	0.16	0.14	0.26	0.12	0.16	0.04

Table 8.34: Sensitivity Probabilities in Dimension 32 for F_2.

Config.	Prob.	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
	P_{NN_0}	0.42	0.36	0.34	0.36	0.66	0.72	0.84	0.98	0.92	0.86	0.88	0.96	1.00	0.98	0.98	1.00
	$P_{ReliefF_0}$	0.14	0.14	0.04	0.16	0.10	0.08	0.10	0.06	0.02	0.06	0.06	0.02	0.10	0.04	0.04	0.06
11	P_{SMOreg_0}	0.48	0.28	0.42	0.42	0.46	0.72	0.80	0.80	0.86	0.88	0.94	0.98	0.98	1.00	1.00	1.00
	P_{NN_1}	0.78	0.84	0.78	0.76	0.96	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	0.24	0.36	0.28	0.34	0.20	0.18	0.28	0.22	0.16	0.22	0.18	0.12	0.22	0.12	0.16	0.12
	P_{SMOreg_1}	0.86	0.80	0.82	0.86	0.86	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
		x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}	x_{29}	x_{30}	x_{31}
	P_{NN_0}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_0}$	0.08	0.08	0.04	0.12	0.16	0.02	0.12	0.08	0.02	0.10	0.08	0.08	0.02	0.22	0.14	0.20
11	P_{SMOreg_0}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	P_{NN_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	0.12	0.18	0.24	0.24	0.32	0.08	0.20	0.14	0.20	0.28	0.22	0.24	0.18	0.50	0.32	0.20
	P_{SMOreg_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Config.	Prob.	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
	P_{NN_0}	0.26	0.20	0.36	0.48	0.76	0.60	0.76	0.94	0.90	0.92	0.94	0.96	1.00	1.00	1.00	0.96
	$P_{ReliefF_0}$	0.06	0.04	0.12	0.08	0.06	0.12	0.08	0.08	0.06	0.06	0.04	0.06	0.04	0.08	0.08	0.08
12	P_{SMOreg_0}	0.42	0.26	0.30	0.38	0.62	0.64	0.64	0.88	0.92	0.96	1.00	0.96	0.96	1.00	1.00	1.00
	P_{NN_1}	0.70	0.92	0.92	0.82	0.96	0.98	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	0.26	0.30	0.34	0.20	0.26	0.26	0.14	0.16	0.22	0.18	0.20	0.30	0.16	0.30	0.22	0.28
	P_{SMOreg_1}	0.72	0.92	0.86	0.84	0.92	0.94	0.94	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00
		x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}	x_{29}	x_{30}	x_{31}
	P_{NN_0}	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_0}$	0.04	0.02	0.02	0.04	0.04	0.10	0.12	0.04	0.08	0.08	0.04	0.10	0.04	0.10	0.10	0.18
12	P_{SMOreg_0}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	P_{NN_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	$P_{ReliefF_1}$	0.14	0.20	0.14	0.16	0.18	0.24	0.18	0.16	0.22	0.20	0.20	0.38	0.20	0.38	0.52	0.18
	P_{SMOreg_1}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

8.3 SAILEGA Runs

$SAILEGA_{default}$, $SAILEGA_{noSu}$ and $SAILEGA_{1point}$ were run for 50 times on each composite fitness function to provide statistically meaningful results.

The outcomes of the experiments are given in Table 8.35 together with the mean of the best individuals from the last generation and their standard deviation calculated from the 50 runs. $SAILEGA_{default}$ and $SAILEGA_{noSu}$ perform distinctly better than $SAILEGA_{1point}$ on all the tested functions when looking at the lower mean of the final individuals and also their standard deviation. These results can be interpreted that the linkage learning and exploitation effort payed off in terms of better final result quality. Furthermore, $SAILEGA_{default}$ and $SAILEGA_{noSu}$ both obtained a very close final mean fitness on the composite function $F_1^7 + F_2^7$, indicating the ability of SAILEGAs operators to work well in this non-linked fitness scenario. Also, Table 8.35 also gives the results of an one-tailed t-test to compare $SAILEGA_{default}$ and $SAILEGA_{noSu}$ against $SAILEGA_{1point}$. The p-value on all three functions for all comparisons is below the threshold for statistical significance (0.01 level), showing that $SAILEGA_{default}$ and $SAILEGA_{noSu}$ performed significantly better than $SAILEGA_{1point}$.

To visualize the performance during the generations, Figures 8.1 and 8.2 provide examples of logarithmically scaled plots of the average fitness development through the generations.

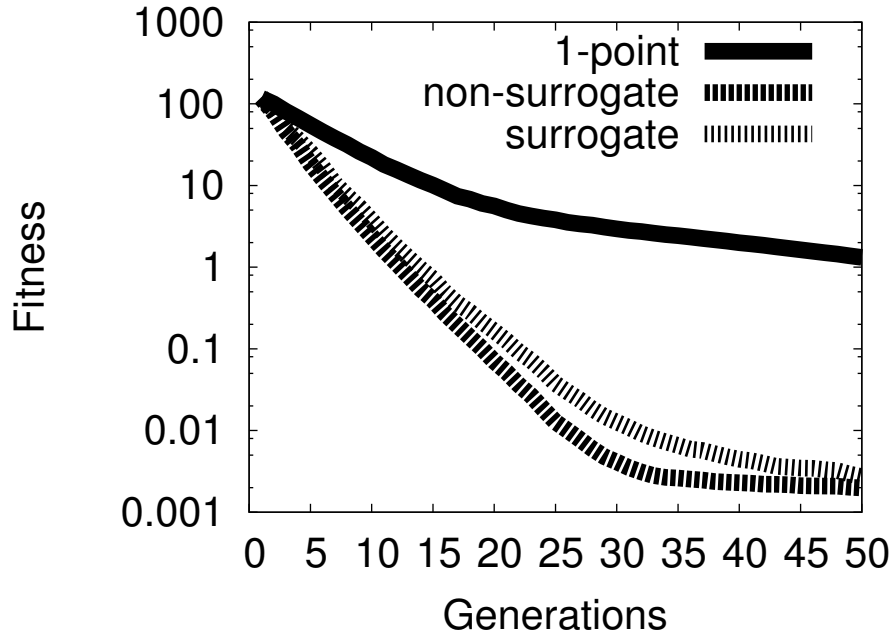

Figure 8.1: Fitness averages on $F_1^7 + F_2^7$.

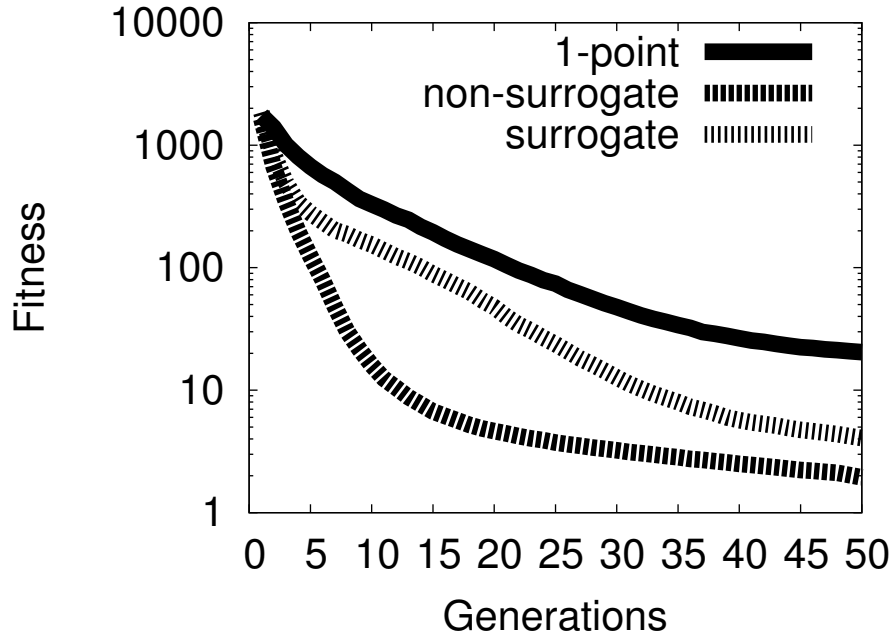

Figure 8.2: Fitness averages on $F_3^6 + F_2^8$.

Table 8.35: SAILEGA Minimization Results.

Config	Function^{Dim}	mean	std. dev.	p-value
$C_{default}$	$F_1^7 + F_2^7$	0.0027	7.68e-4	1.168e-10
C_{noSu}	$F_1^7 + F_2^7$	0.002	0.00036	1.15e-10
C_{1point}	$F_1^7 + F_2^7$	1.32	1.17	-
$C_{default}$	$F_3^6 + F_2^8$	4.06	3.75	2.911e-10
C_{noSu}	$F_3^6 + F_2^8$	1.94	2.27	7.383e-12
C_{1point}	$F_3^6 + F_2^8$	20.44	14.99	-
$C_{default}$	$F_4^4 + F_5^4 + F_2^6$	2.03	1.43	2.567e-10
C_{noSu}	$F_4^4 + F_5^4 + F_2^6$	0.08	0.2	$<$ 2.2e-16
C_{1point}	$F_4^4 + F_5^4 + F_2^6$	4.86	2.42	-

Chapter 9

Conclusion

This Chapter provides a summary of the work presented in this dissertation and includes a review of the main contributions and conclusive remarks about the machine learning aided detection methods for linkage and sensitivity, the exploitation through novel surrogate-assisted operators and finally the benchmarking of overlapping linkage groups in real-valued evolutionary optimization. Current limitations of the developed methods are laid out and future work to mend them is discussed.

9.1 Summary

In this dissertation, the perturbation-based linkage checking method SA-LINC-R is presented, which employs surrogate models to discover interactions among genes without having to call the true fitness function and results on several synthetic fitness functions are presented that provide precision, recall and the F-measure. In addition to that, probabilities for every possible gene-pair check are explored to show how often a specific interaction pair was found. Results show that SA-LINC-R is a promising method to detect interactions if true perturbation costs significantly outweigh the computational expenses of training accurate and precise surrogate models.

Furthermore, an empirical evaluation and comparison including Spearman's ρ for detecting sensitivities through the machine learning methods of neural networks, ReliefF and

support vector machines for regression is given. The neural network method results show that it is a viable approach to discover sensitivities while providing a surrogate model capable of approximating any continuous function at the same time (given a large enough hidden layer).

Means to represent problem structure in real-valued evolutionary optimization are discussed. Iterative linkage neighborhoods are built up with only pair-wise dependency information and a method attempting to extract crossover masks from them is shown. Furthermore, a problem structure matrix idea is envisioned that aims to give a data structure for describing an optimization problem's joint composition of linkage and sensitivity.

The four introduced surrogate-assisted and informed operators are applied through the SAILEGA method, where they show a large performance and final result improvement over a linkage-unaware GA setup. These operators can be integrated into other optimization frameworks, as long as meaningful crossover masks are provided that partition the chromosome with regards to the problem structure.

Finally, a benchmark suite for overlapping linkage groups of increasing size has been defined which fills a gap in the current field of evolutionary computation benchmarks for real-valued optimization problems. It combines benefits of three contemporary benchmarks BBOP, LSGO and LSGO-extended and can be applied with any global optimization algorithm capable of handling real-valued representations. This novel benchmark can be a valuable component in empirically judging the performance and quality differences of optimization methods on well studied fitness functions.

9.2 Review of Contributions

The main contributions of this work are:

1. **Surrogate Model aided Learning of Problem Structure**: Interactions among genes as well as the importance of genes with regards to the fitness are meaningful

measures to characterize a problems structure. The novel machine learning based method of linkage detection, SA-LINC-R, is presented, which extends LINC-R by applying a surrogate model to perform the perturbation checks on the extra individuals. This method is especially useful in cases where the true fitness evaluation is computationally expensive. Three methods for detecting sensitivity, using a neural network, a support vector machine for regression and ReliefF, are presented and empirically compared to gauge their performance. Like SA-LINC-R, no additional fitness evaluations are needed for all three methods, and the neural network method has been found to be the most accurate on the investigated functions. These concepts and their experimental setups are given in Chapter 6, while Chapter 8 contains the corresponding experimental results. Furthermore, iterative linkage neighborhoods are presented and an algorithm aiming to generate crossover masks from them is shown.

2. **Exploitation of Problem Structure**: In Chapter 6, four novel recombination operators are presented that are capable of exploiting the discovered structural information of linkage through crossover masks. By virtue of this, the operators do not disrupt important substructures that need to be optimized together, and are furthermore surrogate-assisted and informed which means that they employ machine learning methods to approximate the fitness function and create a pool of individuals that is ranked by the approximate fitness. Chapter 8 provides the results along with statistical tests for significance for the application of these operators on synthetic fitness functions indicating a large performance improvement.

3. **Overlapping Linkage Benchmark of Increasing Size**: Chapter 7 presents a novel benchmark for real-valued global optimization that focuses on overlapping linkage groups of increasing size, governed by the number of the powers of two. This benchmark contains 33 composite fitness functions and draws from the strengths of three contemporary benchmarks for evolutionary algorithms, BBOP, LSGO and LSGO-extended.

It allows researchers to systematically compare and evaluate optimization algorithms on well-studied and well-defined functions in the context of overlapping groups of dependent genes of different sizes.

9.3 Limitations and Future Work

The presented operators are designed to operate with two parents to produce a pool of offspring - a common setting in EC. However, the concept of optimal mixing allows the use of a new parent for each linkage group investigated. In the future, this can be adapted to correspond to optimal mixing of the proposed substructures and implemented for the defined operators.

Linkage learning with the help of surrogate models can be of benefit in cases where the true cost of a fitness evaluation is the major computational component of an optimization. A crucial measure of the success of such learning is an accurate fitness approximator. The presented setup could be extended by other methods and techniques from the field of surrogate modeling such as a comprehensive surrogate model management system, integration of other techniques such as kriging or the use of parallel computing concepts.

Based on the conclusions about the sensitivity detection through machine learning methods, it becomes harder to detect the perfect ranking of genes as the dimensionality increases. However, a combination of learning techniques (such as ensemble learning) might provide additional precision for this task.

The presented idea of the problem structure matrix requires future work to identify a proper scheme to assign sensible linkage and sensitivity values based on individual problems and to measure them according to well-established metrics.

An improvement for the novel overlapping linkage benchmark is to extend it to include functions that feature noise. Noisy optimization is a challenging field of optimization, in which the fitness function is not guaranteed to return the same fitness value for the same

individual at different times as it includes random error terms. This effect happens in some real-world problems, especially when measurements of the physical reality are part of the fitness function calculation and is worthy to be studied in an extension of this work.

Bibliography

[1] Chang Wook Ahn, R.S. Ramakrishna, and David E. Goldberg. Real-coded bayesian optimization algorithm. In Jose A. Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea, editors, *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 51–73. Springer Berlin Heidelberg, 2006.

[2] Anne Auger, Nikolaus Hansen, and Marc Schoenauer. Benchmarking of continuous black box optimization algorithms. *Evolutionary Computation*, 20(4):481, 2012.

[3] Thomas Bäck. Optimal mutation rates in genetic search. In Stephanie Forrest, editor, *ICGA*, pages 2–8. Morgan Kaufmann, 1993.

[4] Thomas Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *International Conference on Evolutionary Computation*, pages 57–62, 1994.

[5] Richard Bellman. *Dynamic Programming*. Dover Publications, March 2003.

[6] Peter A.N. Bosman and Dirk Thierens. Linkage neighbors, optimal mixing and forced improvements in genetic algorithms. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 585–592, New York, NY, USA, 2012. ACM.

[7] H. J. Bremermann. Optimization through evolution and recombination. In M. C. Yovits, G. T. Jacobi, and G. D. Golstine, editors, *Proceedings of the Conference on Self-Organizing Systems – 1962*, pages 93–106, Washington, DC, 1962. Spartan Books.

[8] Alexander E.I. Brownlee, John A.W. McCall, Siddhartha K. Shakya, and Qingfu Zhang. Structure learning and optimisation in a markov network based estimation of distribution algorithm. In Ying-ping Chen, editor, *Exploitation of Linkage Learning in Evolutionary Algorithms*, volume 3 of *Evolutionary Learning and Optimization*, pages 45–69. Springer Berlin Heidelberg, 2010.

[9] K. Y. Chan, C. K. Kwong, H. Jiang, M. E. Aydin, and T. C. Fogarty. A new orthogonal array based crossover, with analysis of gene interactions, for evolutionary algorithms and its application to car door design. *Expert Syst. Appl.*, 37(5):3853–3862, May 2010.

[10] Kit Yan Chan, Mehmet Emin Aydin, and Terence C. Fogarty. An epistasis measure based on the analysis of variance for the real-coded representation in genetic algorithms. In *IEEE Congress on Evolutionary Computation (1)*, pages 297–304. IEEE, 2003.

[11] Ying-ping Chen, editor. *Exploitation of Linkage Learning in Evolutionary Algorithms*, volume 3 of *Evolutionary Learning and Optimization*. Springer, 2010.

[12] Ying-Ping Chen and Meng-Hiot Lim, editors. *Linkage in Evolutionary Computation*, volume 157 of *Studies in Computational Intelligence*. Springer, 2008.

[13] David Coffin and Robert E. Smith. Linkage learning in estimation of distribution algorithms. In Ying-ping Chen and Meng-Hiot Lim, editors, *Linkage in Evolutionary Computation*, volume 157 of *Studies in Computational Intelligence*, pages 141–156. Springer Berlin Heidelberg, 2008.

[14] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support vector regression machines. In Michael Mozer, Michael I. Jordan, and Thomas Petsche, editors, *NIPS*, pages 155–161. MIT Press, 1996.

[15] Thyago S.P.C. Duque and David E. Goldberg. Clustermi: Building probabilistic models using hierarchical clustering and mutual information. In Ying-ping Chen, editor, *Exploitation of Linkage Learning in Evolutionary Algorithms*, volume 3 of *Evolutionary Learning and Optimization*, pages 123–137. Springer Berlin Heidelberg, 2010.

[16] A. E. Eiben and G. Rudolph. Theory of evolutionary algorithms: a bird's eye view. *Theor. Comput. Sci.*, 229(1-2):3–9, November 1999.

[17] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[18] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, Chichester, WS, UK, 1966.

[19] A. S. Fraser. Simulation of genetic systems by automatic digital computers. I. Introduction. *Australian Journal of Biological Science*, 10:484–491, 1957.

[20] Salvador Garca, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: acase study onthecec2005 special session onreal parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009.

[21] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991.

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

[23] Brian W. Goldman and Daniel R. Tauritz. Linkage tree genetic algorithms: variants and analysis. In *Proceedings of the fourteenth international conference on Genetic and*

evolutionary computation conference, GECCO '12, pages 625–632, New York, NY, USA, 2012. ACM.

[24] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[25] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions. Technical report, Technical Report RR-6829, INRIA, 2010.

[26] Georges Harik. Finding multimodal solutions using restricted tournament selection. In Larry J. Eshelman, editor, *ICGA*, pages 24–31. Morgan Kaufmann, 1995.

[27] Georges Harik. Linkage learning via probabilistic modeling in the ecga. Technical report, IlliGAL Technical Report 99010, Illinois Genetic Algorithm Laboratory, 1999.

[28] Mark Hauschild and Martin Pelikan. Enhancing efficiency of hierarchical boa via distance-based model restrictions. In Günter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *PPSN*, volume 5199 of *Lecture Notes in Computer Science*, pages 417–427. Springer, 2008.

[29] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[30] Wim Hordijk. A measure of landscapes. *Evol. Comput.*, 4(4):335–360, December 1996.

[31] Jeffrey Horn and David E. Goldberg. Genetic algorithm difficulty and the modality of fitness landscapes. In L. Darrell Whitley and Michael D. Vose, editors, *FOGA*, pages 243–269. Morgan Kaufmann, 1994.

[32] Yuan-Wei Huang and Ying-ping Chen. On the detection of general problem structures by using inductive linkage identification. In *Proceedings of the 11th Annual conference*

on Genetic and evolutionary computation, GECCO '09, pages 1853–1854, New York, NY, USA, 2009. ACM.

[33] Y. Jin, M. Hüsken, M. Olhofer, and B. Sendhoff. Neural networks for fitness approximation in evolutionary optimization. In Y. Jin, editor, *Knowledge Incorporation in Evolutionary Computation*, pages 281–305. Springer, Berlin, 2004.

[34] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.*, 9(1):3–12, 2005.

[35] Kenneth A. De Jong. *Evolutionary computation - a unified approach*. MIT Press, 2006.

[36] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4. IEEE, November 1995.

[37] Susan Khor. Linkage structure and genetic evolutionary algorithms. In Ying-ping Chen, editor, *Exploitation of Linkage Learning in Evolutionary Algorithms*, volume 3 of *Evolutionary Learning and Optimization*, pages 3–23. Springer Berlin Heidelberg, 2010.

[38] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[39] Pedro Larrañaga and José A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.

[40] Minqiang Li, David E. Goldberg, Kumara Sastry, and Tian-Li Yu. Real-coded ecga for solving decomposable real-valued optimization problems. In Ying-ping Chen and Meng-Hiot Lim, editors, *Linkage in Evolutionary Computation*, volume 157 of *Studies in Computational Intelligence*, pages 61–86. Springer Berlin Heidelberg, 2008.

[41] Dudy Lim, Yaochu Jin, Yew-Soon Ong, and Bernhard Sendhoff. Generalizing surrogate-assisted evolutionary computation. *IEEE Trans. Evolutionary Computation*, 14(3):329–355, 2010.

[42] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[43] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[44] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[45] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer Berlin Heidelberg, 1996.

[46] Heinz Mühlenbein. Convergence theorems of estimation of distribution algorithms. In Siddhartha Shakya and Roberto Santana, editors, *Markov Networks in Evolutionary Computation*, volume 14 of *Adaptation, Learning, and Optimization*, pages 91–108. Springer Berlin Heidelberg, 2012.

[47] Heinz Mühlenbein, Thilo Mahnig, and Alberto Ochoa Rodriguez. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):215–247, 1999.

[48] M. Munetomo. Linkage identification based on epistasis measures to realize efficient genetic algorithms. In *Proceedings of the Evolutionary Computation on 2002. CEC*

'02. Proceedings of the 2002 Congress - Volume 02, CEC '02, pages 1332–1337, Washington, DC, USA, 2002. IEEE Computer Society.

[49] Masaharu Munetomo and David E. Goldberg. Identifying linkage by nonlinearity check. Technical report, IlliGAL Report 98012, Illinois Genetic Algorithm Laboratory, 1998.

[50] Masaharu Munetomo and David E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 433–440. Morgan Kaufmann Publishers, Inc, 1999.

[51] Masaharu Munetomo and David E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evol. Comput.*, 7(4):377–398, December 1999.

[52] Masaharu Munetomo and David E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4):377–398, 1999.

[53] Andrew L. Nelson, Gregory J. Barlow, and Lefteris Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robot. Auton. Syst.*, 57(4):345–370, April 2009.

[54] Tomasz Oliwa. A machine learning approach for sensitivity inference and exploitation in genetic algorithms. Diploma thesis, Universität Koblenz-Landau, March 2012.

[55] Tomasz Oliwa and Khaled Rasheed. A machine learning approach for sensitivity inference in genetic algorithms. In Hamid R. Arabnia, Ray R. Hashemi, and Ashu M. G. Solo, editors, *GEM*, pages 36–41. CSREA Press, 2010.

[56] Tomasz Oliwa and Khaled Rasheed. A surrogate-assisted linkage inference approach in genetic algorithms. In Natalio Krasnogor and Pier Luca Lanzi, editors, *GECCO*, pages

997–1004. ACM, 2011. doi>10.1145/2001576.2001712 http://doi.acm.org/10.1145/2001576.2001712.

[57] Tomasz Oliwa and Khaled Rasheed. A surrogate-assisted and informed linkage aware ga. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, GECCO Companion '12, pages 1467–1468, New York, NY, USA, 2012. ACM. doi>10.1145/2330784.2330994 http://doi.acm.org/10.1145/2330784.2330994.

[58] Martin Pelikan. *Bayesian optimization algorithm: from single level to hierarchy*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2002.

[59] Martin Pelikan. Analysis of epistasis correlation on nk landscapes with nearest-neighbor interactions. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1013–1020, New York, NY, USA, 2011. ACM.

[60] Martin Pelikan and David. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001*, pages 511–518. Morgan Kaufmann, 2001.

[61] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. Linkage problem, distribution estimation, and bayesian networks. Technical report, IlliGAL Technical Report 98013, Illinois Genetic Algorithm Laboratory, 1999.

[62] Martin Pelikan and Mark Hauschild. Distance-based bias in model-directed optimization of additively decomposable problems. In Terence Soule and Jason H. Moore, editors, *GECCO*, pages 273–280. ACM, 2012.

[63] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In Pat Langley, editor, *ICML*, pages 727–734. Morgan Kaufmann, 2000.

[64] Khaled Rasheed. *GADO: A genetic algorithm for continuous design optimization.* Ph.d. thesis, Rutgers University, 1998.

[65] Khaled Rasheed. An incremental-approximate-clustering approach for developing dynamic reduced models for design optimization. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 986–993, 2000.

[66] Khaled Rasheed and Haym Hirsh. Guided crossover: A new operator for genetic algorithm based optimization. In *In Proceedings of the Congress on Evolutionary Computation*, pages 1535–1541, 1997.

[67] Khaled Rasheed and Haym Hirsh. Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models. In L. Darrell Whitley, David E. Goldberg, Erick Cantú-Paz, Lee Spector, Ian C. Parmee, and Hans-Georg Beyer, editors, *GECCO*, pages 628–635. Morgan Kaufmann, 2000.

[68] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, 1973.

[69] Colin Reeves and Christine Wright. An experimental design perspective on genetic algorithms. In *Foundations of Genetic Algorithms 3*, pages 7–22. Morgan Kaufmann, 1995.

[70] Rommel G. Regis and Christine A. Shoemaker. Local function approximation in evolutionary algorithms for the optimization of costly functions. *IEEE Trans. Evolutionary Computation*, 8(5):490–505, 2004.

[71] Marko Robnik-Sikonja and Igor Kononenko. An adaptation of relief for attribute estimation in regression. In Douglas H. Fisher, editor, *ICML*, pages 296–304. Morgan Kaufmann, 1997.

[72] Gian-Carlo Rota. The Number of Partitions of a Set. *The American Mathematical Monthly*, 71(5):498–504, May 1964.

[73] G. Rudolph. Convergence analysis of canonical genetic algorithms. *Trans. Neur. Netw.*, 5(1):96–101, January 1994.

[74] Eman Sayed, Daryl Essam, and Ruhul A. Sarker. Dependency identification technique for large scale optimization problems. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[75] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen*. Birkhäuser Verlag, Basel, 1977.

[76] Hans-Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[77] Dong-Il Seo, Yong-Hyuk Kim, and Byung-Ro Moon. New entropy-based measures of gene significance and epistasis. In *Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII*, GECCO'03, pages 1345–1356, Berlin, Heidelberg, 2003. Springer-Verlag.

[78] Yun-Wei Shang and Yu-Huang Qiu. A note on the extended rosenbrock function. *Evol. Comput.*, 14(1):119–126, March 2006.

[79] Shirish Krishnaj Shevade, S. Sathiya Keerthi, Chiranjib Bhattacharyya, and K. R. K. Murthy. Improvements to the smo algorithm for svm regression. *IEEE Trans. Neural Netw. Learning Syst.*, 11(5):1188–1193, 2000.

[80] Alexander J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[81] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capacities of ec optimization and human evaluation. In *IEEE*, volume 89, pages 1275–1296, 2001.

[82] Kē Táng, Xiǎodōng Lǐ, Ponnuthurai Nagaratnam Suganthan, Zhènyǔ Yáng, and Thomas Weise. Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization. Technical report, University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL): Héféi, Ānhuī, China, January 8, 2010.

[83] Masaru Tezuka, Masaharu Munetomo, and Kiyoshi Akama. Linkage identification by nonlinearity check for real-coded genetic algorithms. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors, *GECCO (2)*, volume 3103 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2004.

[84] Dirk Thierens. The linkage tree genetic algorithm. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *PPSN (1)*, volume 6238 of *Lecture Notes in Computer Science*, pages 264–273. Springer, 2010.

[85] Dirk Thierens and Peter Bosman. Predetermined versus learned linkage models. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 289–296, New York, NY, USA, 2012. ACM.

[86] Dirk Thierens and Peter A.N. Bosman. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 617–624, New York, NY, USA, 2011. ACM.

[87] C. K. Ting, W. M. Zeng, and T. C. Lin. Linkage discovery through data mining. *IEEE Comput. Intell. Mag.*, 5:10–13, 2010.

[88] Miwako Tsuji. *Designing Genetic Algorithm Based on Exploration and Exploitation of Gene Linkage*. PhD thesis, Hokkaido University, Sapporo, Japan, 2007.

[89] Miwako Tsuji and Masaharu Munetomo. Linkage analysis in genetic algorithms. In Lakhmi C. Jain, Mika Sato-Ilic, Maria Virvou, George A. Tsihrintzis, Valentina Emilia Balas, and Canicious Abeynayake, editors, *Computational Intelligence Paradigms*, volume 137 of *Studies in Computational Intelligence*, pages 251–279. Springer, 2008.

[90] B. Üstün, W.J. Melssen, and L.M.C. Buydens. Facilitating the application of support vector regression by using a universal pearson VII function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81(1):29–40, March 2006.

[91] Dennis D. Wackerly, William Mendenhall III, and Richard L. Scheaffer. *Mathematical Statistics with Applications*. Duxbury Advanced Series, sixth edition, 2002.

[92] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[93] David Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1):67–82, 1997.

[94] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985 – 2999, 2008. Nature Inspired Problem-Solving.

[95] Kenny Q. Ye, William Li, and Agus Sudjianto. Algorithmic construction of optimal symmetric latin hypercube designs. *JOURNAL OF STATISTICAL PLANNING AND INFERENCES*, 90:145–159, 2000.

[96] Tian-Li Yu, David E. Goldberg, Kumara Sastry, Cláudio F. Lima, and Martin Pelikan. Dependency structure matrix, genetic algorithms, and effective recombination. *Evolutionary Computation*, 17(4):595–626, 2009.

[97] Tian-Li Yu, David E. Goldberg, Ali Yassine, and Ying-Ping Chen. Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm. In Erick Cantú-Paz, James A. Foster, Kalyanmoy Deb, Lawrence Davis, Rajkumar Roy, Una-May O'Reilly, Hans-Georg Beyer, Russell K. Standish, Graham Kendall, Stewart W. Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitchell A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasa Jonoska, and Julian F. Miller, editors, *GECCO*, volume 2724 of *Lecture Notes in Computer Science*, pages 1620–1621. Springer, 2003.

Appendix A

Surrogate Modeling with Weka

Weka is a popular machine learning and data mining open-source (free software under the GPL) suite written in Java. In the following pages, code examples on how to create surrogate models for a population of an EA for real-valued optimization will be presented as well as an implementation of population clustering.

A.1 Weka Preliminaries

Weka[1] comes in two editions, the (stable) book edition (at the time of writing, weka-3-6-9), and the developer edition (at the time of writing, weka-3-7-9), in this context it is assumed that the developer edition is used on a GNU/Linux system with an installed Java environment. A very helpful resource with many examples and their source codes is the Weka wiki[2], and a short guide on training a neural network that these pages are inspired by is included in [54].

After obtaining and unpacking the Weka package "weka-3-7-9.zip", the suite can be started by running "weka.jar" in a shell such as Bash as follows (omitting the $ symbol):

```
$ java -jar weka.jar
```

[1] Weka is available at: http://www.cs.waikato.ac.nz/ml/weka/
[2] The Wiki is available at: http://weka.wikispaces.com/

The weka source code resides in the file "weka-src.jar", and, once extracted, is accessible under "../weka-3-7-9/weka-src/src/main/java/weka/". The Weka source code needs to be accessible (for imports of its classes) to allow the following code examples to function. Furthermore, Weka uses a special format for its data instances, the Attribute-Relation File Format (ARFF). To employ Weka on a population of an EA, the population needs to be transformed into this format. For demonstration purposes, a bare bones definition of an individual from a real-valued EA is defined in Listing A.1, the class *Individual* contains the genes as well as the fitness. Listing A.2 shows a method that has a list of individuals as input and creates an Instances object in the ARFF format for further Weka processing.

```
1  public class Individual {
2
3      private double[] genes;
4      private double fitness;
5
6      public Individual(double[] genes, double fitness) {
7          this.genes = genes;
8          this.fitness = fitness;
9      }
10
11     public double[] getGenes() {
12         return genes;
13     }
14
15     public double getFitness() {
16         return fitness;
17     }
```

```
18 }
```

Listing A.1: A bare bones implementation of an individual.

```
1     public static Instances createInstances(List<Individual>
          population) {
2
3         int popSize = population.size();
4         int chromLength = population.get(0).getGenes().length;
5
6         // Creating the attributes object
7         ArrayList<Attribute> atts = new ArrayList<Attribute>();
8
9         // All genes are added as attributes
10        for (int i = 0; i < chromLength; i++) {
11            atts.add(new Attribute("gene " + i));
12        }
13
14        // As last attribute, the fitness is added
15        atts.add(new Attribute("fitness"));
16
17        // The instances object is created
18        Instances data = new Instances("population", atts,
              popSize);
19
20        // Every instance is added to the data object
21        double[] vals;
22        for (int i = 0; i < popSize; i++) {
```

```
23                vals = new double[data.numAttributes()];
24                vals = Arrays.copyOf(population.get(i).getGenes(),
                      chromLength + 1);
25                // The fitness is added
26                vals[data.numAttributes() - 1] = population.get(i).
                      getFitness();
27                data.add(new DenseInstance(1.0, vals));
28            }
29
30            // The class attribute (fitness) is set
31            data.setClassIndex(data.numAttributes() - 1);
32
33            return data;
34        }
```

Listing A.2: Creating the Weka instances.

A.2 Fitness Approximation

In Listing A.3, a method is shown that creates a support vector machine for regression (SMOreg) and trains it on an object of data instances, such as the object created in Listing A.2. The SMOreg can be set with desired options (in this example, the exponent "2.0" is specified for the kernel). The SMOreg is then trained on the data instances. Finally, in Listing A.4, an example is given of how to approximate the fitness of a new individual by letting the trained SMOreg classify the instance and store the estimated fitness in the result variable.

```
1  public AbstractClassifier buildClassifier(Instances data) {
```

```
2       SMOreg smoREG = new SMOreg();
3       String[] options = new String[2];
4       options[0] = "-K";
5       options[1] = "weka.classifiers.functions.supportVector.
            PolyKernel -C 250007 -E 2.0";
6       try {
7           smoREG.setOptions(options);
8           smoREG.buildClassifier(data);
9       } catch (Exception e) {
10          // perform Exception handling
11      }
12      return smoREG;
13  }
```

Listing A.3: Training a support vector machine for regression.

```
1  Instances population = ... // create population instances
2  Instances newInstance = ... // create instance to classify
3
4  AbstractClassifier classifier = buildClassifier(population);
5  double result = 0;
6
7  try {
8      result = classifier.classifyInstance(newInstance.
           firstInstance());
9  } catch (Exception e) {
10     // perform Exception handling
```

```
11 }
```

Listing A.4: Classifying a new instance.

A.3 Clustering

Weka also contains clustering algorithms that can be applied on a population of an EA (which has been used to create an Instances object as shown in Listing A.2). Clustering is useful in situations where a partitioning of the current population (or even all or a subset of all encountered individuals) is desired, for instance to train a surrogate model on each of the clusters to then obtain a set of local surrogates. A method that performs a clustering of instances and returns an array of cluster assignments in the order of individuals in the instances object is shown in Listing A.5.

```
1  public static double[] clusterInstances(Instances data, int
       numberOfClusters) {
2      SimpleKMeans kMeans = new SimpleKMeans();
3      ClusterEvaluation eval = new ClusterEvaluation();
4      try {
5          data.setClassIndex(-1); // necessary for clustering
6          kMeans.setNumClusters(numberOfClusters);
7          kMeans.setPreserveInstancesOrder(true);
8          kMeans.buildClusterer(data);
9          eval.setClusterer(kMeans);
10         eval.evaluateClusterer(new Instances(data));
11     } catch (Exception ex) {
12         // perform Exception handling
13     }
```

```
14        return eval.getClusterAssignments();
15    }
```

Listing A.5: Classifying a new instance.

Appendix B

Statistical Tests with R

R is a language (free software under the GPL) for statistical computing, modeling and analysis. Examples of how to use R's capabilities to calculate the mean and standard deviation and how to perform the one-tailed t-test and the Wilcoxon rank sum test on the outputs of EAs are presented here.

B.1 R Preliminaries

The R[1] language is a dynamically typed interpreted language which contains a large collection of statistical methods. For the following examples, it is assumed that R is installed and used on a GNU/Linux system. The R interpreter is started from a shell such as Bash as follows (omitting the $ symbol):

```
$ R
```

B.2 Statistical Tests with R

If an EA is run for multiple times on the same fitness function with the task of global optimization (and provided that the seed for the random number generator is not fixed), the

[1] R is available at: http://www.r-project.org/

final best individuals found are often different from each other. To compare the performance of two different optimizers on the same fitness function, the means and corrected standard deviations of the best final individuals for those two algorithms can be calculated to provide a first impression of the performances.

Furthermore, one-tailed t-tests have been used in the past to check for a certain statistical significance (one direction) of two datasets, which in the context of evolutionary optimization aims to answer: Given a fitness function for a minimization task and a list of final fitness scores of runs of optimizer A on this function and also a list of final fitness scores of optimizer B on this function, did optimizer A perform statistically significantly (with a confidence level) better than B?

For the one-tailed t-test, the two lists with the final results need to contain a reasonable representative number of independent results drawn from a normal distribution. With regards to EAs, lists that have been used consist of, for example, an adequate number of final fitness scores of the optimizers, where each run was performed independently on the fitness function. However, in [20] it is recommended to perform non-parametric tests such as the Wilcoxon rank sum test to compare the final results of EAs instead. The rationale is that the conditions for parametric tests (like the t-test) are not always fulfilled.

In a first example (in which, for the sake of brevity, only five final fitness scores per optimizer are shown) in Listing B.1, it can be assumed that optimizers A and B have been each run five times on a minimization problem and their results are stored in the variables $resultA$ and $resultB$. After calculating their means and standard deviations, a first thought might be that optimizer A did not outperform optimizer B. A one-tailed t-test is then performed to test if $resultA$ is statistically significantly better (in terms of minimization) than $resultB$ under a certain significance level, for example 0.05. Looking at the p-value of 0.9992, it can be said that this is not the case and that the null hypothesis cannot be rejected. It cannot be said that $resultA$ is statistically significantly better (in terms of minimization) than $resultB$. The Wilcoxon rank sum test provides a p-value of 1, leading to the same

conclusion.

```
> resultA <- c(2, 2.4, 2.1, 1.9, 2.5)
> resultB <- c(1, 1.5, 1.2, 0.7, 1.6)
> mean(resultA)
[1] 2.18
> mean(resultB)
[1] 1.2
> sd(resultA)
[1] 0.2588436
> sd(resultB)
[1] 0.3674235
>
> t.test(resultA, resultB, alternative="less")

        Welch Two Sample t-test

data:  resultA and resultB
t = 4.8757, df = 7.186, p-value = 0.9992
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
    -Inf 1.35933
sample estimates:
mean of x mean of y
     2.18      1.20
>
> wilcox.test(resultA, resultB, alternative="less")

```

```
27            Wilcoxon rank sum test
28
29 data:  resultA and resultB
30 W = 25, p-value = 1
31 alternative hypothesis: true location shift is less than 0
```

Listing B.1: Running statistical tests with R, first example.

Listing B.2 shows a second example, where optimizer A2 has been run on the same fitness function and obtained its results stored in the variable *resultA*2. After computing the means and standard deviations, it might initially look like optimizer A2 outperformed optimizer B, since the mean and standard deviation of *resultA*2 are both lower than their counterparts of *resultB*. However, when performing the one-tailed t-test and the Wilcoxon rank sum test to determine if *resultA*2 is statistically significantly better (in terms of minimization) than *resultB*, the obtained p-values of 0.4397 and 0.5 indicate that this is not the case and that the null hypothesis cannot be rejected. Therefore, it cannot be claimed that optimizer A2 outperformed optimizer B.

```
1  > resultA2 <- c(0.95, 1.21, 1.4, 1.52, 0.75)
2  > resultB <- c(1, 1.5, 1.2, 0.7, 1.6)
3  > mean(resultA2)
4  [1] 1.166
5  > mean(resultB)
6  [1] 1.2
7  > sd(resultA2)
8  [1] 0.3169069
9  > sd(resultB)
10 [1] 0.3674235
11 >
```

```
12 > t.test(resultA2, resultB, alternative="less")
13
14         Welch Two Sample t-test
15
16 data:  resultA2 and resultB
17 t = -0.1567, df = 7.831, p-value = 0.4397
18 alternative hypothesis: true difference in means is less than 0
19 95 percent confidence interval:
20       -Inf 0.3706399
21 sample estimates:
22 mean of x mean of y
23     1.166      1.200
24
25 >
26 > wilcox.test(resultA2, resultB, alternative="less")
27
28         Wilcoxon rank sum test
29
30 data:  resultA2 and resultB
31 W = 12, p-value = 0.5
32 alternative hypothesis: true location shift is less than 0
```

Listing B.2: Running statistical tests with R, second example.