An Adaptive Approach to Controlling Parameters of Evolutionary Algorithms

Aldeida Aleti

Submitted in fullment of the requirements for the degree of

Doctor of Philosophy

Swinburne University of Technology

2012

ABSTRACT

The majority of stochastic optimisation methods such as Simulated Annealing (SA), Evolutionary Algorithms (EA), Ant Colony Optimisation (ACO) and Estimation of Distribution Algorithms (EDA) have a range of adjustable parameters like learning rates, crossover probabilities, pheromone evaporation rates and weighting factors. Poor algorithm parameterisation hinders the discovery of good solutions. The parameter values required for optimal algorithm performance are known to be problem-specific, often even specific to the problem instance at hand. Practitioners often apply stochastic methods with parameter values chosen on the basis of few tuning iterations, in which various parameter settings are explored in an attempt to fine-tune the algorithm to their particular problem. Depending on the number of parameters and their plausible value ranges, investigative trials for parameter optimisations can themselves be attempts to solve a combinatorially complex problem. Moreover, it has also been established that some of the parameter values ought to vary during the search process for best algorithm performance.

Acknowledging these facts, many researchers have shifted their focus to parameter control methods, where parameter values are optimised based on algorithm performance. This thesis presents an adaptive parameter control method which redefines parameter values repeatedly based on a separate optimisation process that receives its feedback from the primary optimisation algorithm. The feedback is used for a projection of the value performing well in the future. The method uses an evaluation of the recent performance of previously applied parameter values and predicts how likely each of the parameter values is to produce optimal outcomes in the next

cycle of the algorithm. The parameter values are sampled from intervals which are adapted dynamically, a method which has proven particularly effective and outperforms all existing adaptive parameter controls significantly. To test the applicability of the approach to a real-world problem, the adaptive parameter control method is applied to a case-study from the automotive industry.

Acknowledgements

I dedicate this work to my family, Torgeir, Meli, Ari, Riza, Eva & Trond.

I am deeply grateful to my two supervisors Dr. Irene Moser and Dr. Lars Grunske, whom I was lucky to have as my mentors. I would also like to thank Indika Meedeniya, Dr. Antony Tang, and Dr. Sanaz Mostaghim for their valuable advice and help and my friends Nargiza, Tharindu and Iman for their support.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Melbourne, July 23, 2012

Aldeida Aleti

Publications

Parts of the material in this thesis have previously appeared in the following publications:

Chapter 4

* A. Aleti, "Quality Assessment of Multiobjective Optimisation Algorithms in Component Deployment", in the Doctoral Symposium, European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), pp. 39-40, ACM Digital Libraries, 2009.

* A. Aleti, L. Grunske, I. Meedeniya, I. Moser, "Let the ants deploy your software - An ACO based deployment optimisation strategy", in the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE) pp. 505 - 509, IEEE Digital Libraries, 2009.

Chapter 5

* A. Aleti and I. Meedeniya, "Component Deployment Optimisation with Bayesian Learning" , in Component Based Software Engineering (CBSE), pp. 11-20, ACM Digital Libraries, 2011.

Chapter 6

* A. Aleti and I. Moser, "Predictive Parameter Control", in Genetic and Evolutionary Computation Conference (GECCO), pp 561-568, ACM Digital Libraries, 2011.

Chapter 7

* A. Aleti, I. Moser and S. Mostaghim, "Adaptive Range Parameter Control", In

IEEE World Congress on Computational Intelligence (WCCI), to appear, 2012.

Chapter 8

* I. Meedeniya, A. Aleti, and L. Grunske, "Architecture-Driven Reliability Optimization with Uncertain Model Parameters", in Journal of Systems and Software (JSS), 2012, to appear.

* I. Meedeniya, A. Aleti, I. Avazpour and A. Amin, "Robust ArcheOpterix: Architecture Optimization of Embedded Systems under Uncertainty", International Workshop on Software Engineering for Embedded Systems (SEES), International Conference on Software Engineering (ICSE), to appear, 2012.

* I. Meedeniya, B. Zimmerova, A. Aleti and L. Grunske, "Architecture Driven Reliability and Energy Optimization for Complex Embedded Systems", in Quality of Software Architectures (QoSA), Lecture Notes in Computer Science, Vol. 6093, pp. 52-67, Springer, 2010.

* A. Aleti, Stefan Bjornander, L. Grunske, and I. Meedeniya, "Acheopterix: An extendable tool for architecture optimisation of aadl models", in Model-based Methodologies for Pervasive and Embedded Software (MOMPES), pp. 61-71, ACM and IEEE Digital Libraries, 2009.

* I. Meedeniya, A. Aleti and B. Zimmerova, "Redundancy Allocation in Automotive Systems using Multiobjective Optimisation", in the Symposium on Automotive/Avionics Systems Engineering (SAASE), 2009.

* I. Meedeniya, I. Moser, A. Aleti and L. Grunske, "Software Architecture Evaluation under Uncertainty", in Quality of Software Architectures (QoSA), pp. 85-94, ACM, 2011.

* I. Meedeniya, B. Buhnova, A. Aleti and L. Grunske, "Reliability-Driven Deployment Optimization for Embedded Systems", in Journal of Systems and Software (JSS), 84(5), pp. 835-846, 2011.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

* **FCS**: Feedback Collection Strategy
* **BEA**: Bayesian Effect Assessment
* **QAS**: Quality Attribution Strategy
* **PQA**: Predictive Quality Attribution
* **ARPS**: Adaptive Range Parameter Selection
* **PM**: Probability Matching
* **AP**: Adaptive Pursuit
* **DMAB**: Dynamic Multi-Armed Bandits
* **EA**: Evolutionary Algorithm
* **EDA**: Estimation of Distribution Algorithms
* **ES**: Evolutionary Strategies
* **GA**: Genetic Algorithm
* **GP**: Genetic Programming
* **MQAP**: Multiobjective Quadratic Assignment Problem
* **RR**: Royal Road problem
* **QAP**: Quadratic Assignment Problem
* **RAP**: Redundancy Allocation Problem
* **CDP**: Component Deployment Problem

Part I

Introduction, Background and Methodology

Chapter 1

Introduction

Due to their general applicability, stochastic optimisers such as Evolutionary Algorithms (EAs) are popular among scientists and engineers facing difficult optimisation problems. Stochastic algorithms are not expected to deliver the optimal solutions, but to provide good approximate results where exact approaches cannot be devised and optimal solutions are hard to find. Usually, these algorithms make no assumptions about the nature of optimisation problem, therefore they are also called black-box optimisation methods.

Stochastic optimisation algorithms are often used to solve combinatorial optimisation problems, where an exhaustive search of the search space is not feasible in polynomial time. A combinatorial optimisation problem can be defined as choosing the best solutions from all possible combinations of n different variables. The possible solutions may include all trees of n nodes or all possible Hamilton cycles of a complete graph with n nodes. Moreover, the objective functions are often computationally expensive and non-linear. Listing all possible solutions in order to find the best candidates is a non-deterministic polynomial-time hard (NP-hard) problem, i.e. solvable in polynomial time by a non-deterministic Turing machine.

Evolutionary Algorithms (EAs) have shown good performance in solving hard optimisation problems. They maintain a population of solutions which is evolved with the help of the genetic operators: mutation, crossover, often called recombina-

tion, selection and replacement. The mutation and crossover operators are employed to produce new solutions using candidate solutions from the population. The mutation operator creates a new solution by performing a random or guided change in one of the members of the current population, whereas the crossover operator is used to combine features of existing solutions. The selection operator is used to select the solutions that will undergo the variation procedure, which is composed of the mutation and crossover operators. Finally, the replacement operator is employed to decide which solutions should survive to the next iteration. These operators and other elements of an EA that are involved in the process of variation and selection, such as the probabilities of applying these operators, the population size, the number of offspring produced, are the algorithm parameters.

Formerly, EAs were seen as robust algorithms that exhibit approximately similar performance over a wide range of problems [66]. Hence, the use of these algorithms has increased over the years, especially in the Software Engineering domain [62, 97, 119]. The architecture design optimisation of embedded systems [21, 50, 62, 147, 193, 119] is one of the fields of the Software Engineering where the application of EAs has become almost a necessity. Architecture design entails making a multitude of decisions regarding the prospective system. Numerous stakeholders are interested in certain quality attributes of the system, which are dependent on the decisions made on the architecture [62, 97, 119], such as the allocation of the software architecture to the hardware resources [129, 4, 147] and the selection of the redundancy levels for the safety-critical components [73, 127]. The quality attributes are often in conflict with each other and solutions which are optimal according to all quality attributes may not exist. As embedded systems get larger and more complex, the task of finding high-quality architectures becomes unmanageable for a system engineer. The use of EAs in architecture optimisation helps not only make better decisions, but also reduce the time required for the decision.

In recent years, it has been acknowledged that the robustness of Evolutionary Algorithms is mostly due to the numerous parameters that these algorithms have [66, 48, 135, 13], which make the optimisation procedure flexible and efficient for any kind of problem, regardless of the search space difficulty. The settings of the parameter values, greatly affect the performance of the algorithm [135, 13, 115, 49].

Unfortunately, practitioners usually have little expertise in the Artificial Intelligence field, and they require guidance in the application of stochastic methods to their particular problems. Even after many years of research into EAs and other stochastic approaches, there are no straightforward parametrisation guidelines for interdisciplinary users of these methods. For instance, general guidelines regarding the crossover rate recommend using a value equal to 0.6 [39], 0.95 [69], or any value in the range $[0.75, 0.95]$ [160]. Similarly, DeJong [39] recommends a mutation rate equal to 0.001, whereas Grefenstette [69] argues that the mutation rate should be less than 0.05.

The reason for these conflicting guidelines, is the fact that different parameter values may be optimal for different problems. Mitchell [137] noted that it is unlikely that general principles about EA parameter configurations can be formulated, and Eiben and Smit [49] argued that setting EA parameters is an optimisation problem in itself. As a consequence, practitioners tend to choose parameter values based on few trials in which various parameter settings are explored in an attempt to fine-tune an EA to a particular problem. A conceptual framework for tuning EA parameters and an insightful analysis of state-of-the-art tuning methods is presented by Eiben and Smit [49].

However, it has been empirically and theoretically demonstrated that different parameter settings are required not only for different problem instances but also for different optimisation stages of that instance [13, 9, 179, 174, 77, 29, 180]. For instance, the 'optimal' values of mutation rate in an Evolutionary Algorithm were

studied in a set of different benchmark problems [29], where it was shown that these values depend on the state of the search space and any fixed mutation rate produced sub-optimal results. Similar works have proposed a time-dependency of certain algorithm parameters, such as the mutation rate [77, 38, 15], the crossover rate [38] and the population size [13, 115].

In essence, tuning parameter values before the optimisation process does not guarantee an optimal performance of the EA. This problem has been tackled by many researchers in the optimisation community [13, 45, 66, 55, 174, 43, 154, 134, 77, 9, 179, 28], who proposed to set the parameter values of an EA during the run, known as parameter control [45, 66, 55, 33, 37, 82, 84, 94, 114, 163, 183]. The intuitive motivation comes from the way the optimisation process unfolds from a more diffused global search, requiring parameter values responsible for the exploration of the search space, to a more focused local optimisation process, requiring parameter values which help with the convergence of the algorithm.

Different parameter control techniques exist in the literature. Following the classification by Eiben et al. [45], methods for controlling parameters of an EA perform this task in three different ways: deterministically, self-adaptively or adaptively.

Deterministic parameter control can be regarded as a variation of parameter tuning, in which several parameter settings are chosen based on preliminary experiments [133], to alleviate the performance problems of parameters that are invariate throughout the optimisation process. These changes are based on time, i.e. new values are assigned after every predetermined number of iterations. One example is decreasing the mutation rate every w iterations [57]. The problem then lies in devising such a schedule, i.e. in finding the ideal value for w. This schedule depends on the total number of iterations that the optimisation algorithm will take to converge, an information that usually is not possible to know before the run.

Another way of tackling this issue is using an optimisation procedure composed

of two levels; the top optimisation level evolves the parameters of a second level. Rechenberg [154] presents one of the first examples of this approach, by using a 'nested' Evolution Strategy (ES), where an ES is used to evolve the parameters of a lower-level ES. This is an example of *self-adaptive parameter control*. The choice of the parameter values of the top level EA still remains open. Other self-adaptive parameter control methods integrate the search for optimal parameters into the optimisation process itself - usually by encoding parameter settings into the genotype of the solution to evolve [13, 15, 53, 40]. Extending the solution size to include the parameter space obviously increases the search space and makes the search process more time-consuming [45].

In adaptive parameter control [181, 56, 33, 37, 82, 84, 94, 114, 163, 183], properties of an EA run (such as the quality of the solutions produced) are monitored and the change in the properties is used as a signal to change parameter values. The update mechanism to control the parameter values is decided by the user, rather than being part of the evolutionary cycle.

Usually, an adaptive parameter control mechanism performs four main steps: (i) it measures the change in the properties of the optimisation algorithm (*feedback collection*), such as the quality of the solutions produced, (ii) it employs the feedback from the search to infer the effect of parameter values on the successful performance of the optimisation process (*parameter effect assessment*), such as producing solutions with quality above a certain threshold (iii) it approximates the overall quality of parameter values to use in the next iteration (*parameter quality attribution*), and (iv) it adjusts the values of the parameters according to the estimated quality (*parameter value selection*). This thesis investigates each of these four steps and introduces effective ways of controlling parameter values in an adaptive way.

Current research in feedback collection mechanisms for parameter control methods focuses only on singleobjective optimisation, in which the relevant property that

indicates the performance of the algorithm is the change in fitness of the best solution. Measuring the performance of multiobjective optimisation algorithms is not as straightforward, since the output is a set of solutions which constitute a trade-off between the fitness functions, hence the selection of a feedback collection mechanism remains a challenge. With the aim of selecting an appropriate feedback collection mechanism to guide the control of parameter values in multiobjective optimisation, we investigate state-of-the-art multiobjective performance metrics. The analysis of the different multiobjective performance metrics is presented in Chapter 4 and the results of the comparison of different feedback collection strategies are discussed in Section 4.4.

Approximating the cause of the successful performance of a stochastic algorithm is challenging, since stochastic methods generally produce different results for the same parameter values [43], and the performance of the algorithm may be affected by more than one parameter value. In order to deal with the noise of stochastic systems, such as Evolutionary Algorithms, we propose a new probabilistic parameter effect assessment strategy, which is based on Bayesian Belief Networks. The Bayesian Effect Assessment (BEA) strategy is described in Chapter 5 and the validation of the the approach is presented in Section 5.5.

State-of-the-art parameter quality attribution methods derive parameter quality from recent performance, such as result quality in preceding iteration(s) (time $t-1$). One might argue that these approaches are 'one step behind', in that they propose the use of the parameter value which is optimal for the previous iteration. Ideally, we would use a setting optimised for the beginning iteration (time t). It follows that an ideal parameter quality attribution strategy would attempt to predict successful parameter values for time t based on previous performance.

The current work explores the hypothesis that using time series prediction to forecast the success of parameter values based on previous quality improves the

choice of parameters, and as a result improves the performance of the algorithm. To this end, suitable forecasting techniques are explored, with the goal of achieving adequate predictions of the quality of parameter values. Different forecasting techniques, and the analysis of the assumptions they make regarding the time-series data are discussed in Chapter 6. An empirical evaluation of the proposed Predictive Quality Attribution Strategy (PQA) is presented in Section 6.5.

The parameter value selection strategies employed in current state-of-the-art parameter control methods use predefined parameter ranges to sample the continuous values from or defined discretised choices. The quality feedback and therefore the probability of use in the next iteration is allocated to these ranges, not the actually sampled values. As the ranges are fixed, they remain sub-optimal throughout the optimisation process. Defining narrow ranges leads to more accuracy but increased combinatorial complexity. Leaving ranges wider entails a sampling inaccuracy as the actually sampled value may be far from the value whose success the range's usage probability is attributable to. Ideally, the ranges should be optimised by the parameter control process.

With the goal of handling parameter value ranges efficiently, we investigate the use of adaptive value ranges which dynamically change during the optimisation process. The proposed parameter value selection strategy is described in Chapter 7 and is validated in Section 7.3.

In essence, the goal of this thesis is to build an efficient optimisation framework which eliminates the requirement of tuning the parameters to a specific problem and that finds better solutions than state-of-the-art optimisation frameworks. Efficiency in this case refers to the following attributes for the optimisation approach:

1. Achieves good solutions with respect to the objectives being optimised compared to state-of-the-art parameter control methods.

2. Requires minimal effort in configuring the optimisation algorithm.

3. Is robust with respect to different problems being optimised: i.e. the performance of the algorithm is independent of the problem and problem instance.

With these goals in mind, the contributions of this thesis are empirically validated by using a case study from the automotive domain and a set of experiments designed to compare the proposed strategies with state-of-the-art parameter control approaches. The case study and the application of the adaptive parameter control for Evolutionary Algorithms are described in Chapter 8. The methodology used in this thesis is described in Chapter 3. The results and the analysis of the experiment on a set of benchmark optimisation problems are presented in each chapter. The conclusions of this work are presented in Chapter 9. A further discussion on the overall approach and possible ideas and directions about the future work are presented in Chapter 10.

Chapter 2

Background

2.1 Introduction

This chapter presents a general background on Evolutionary Algorithms (EA), the different parameters and methods for configuring EA parameters. First, we give a brief introduction to singleobjective and multiobjective optimisation, followed by a more in-depth description of how optimisation is performed in different classes of Evolutionary Algorithms (Genetic Algorithm, Evolution Strategies, Genetic Programming, Evolutionary Programming) and the strategy parameters that are used in each of the different algorithm classes.

Next, we present a review of state-of-the-art methods in configuring parameters of Evolutionary Algorithm, which are categorised as parameter tuning and parameter control. We analyse conflicting views in the matter of parameter tuning and parameter control and define the scope of the work presented in this thesis. Finally, we describe how various EA parameters have been configured using the three parameter control techniques - deterministic, self-adaptive and adaptive parameter control.

2.2 Optimisation

The goal of optimisation is to find the optimal solution(s) s^* among a set of solutions $S = \{s_1, s_2, ..., s_n\}$, according to the objective function $f(s)$, which describes the quality of each solution in S. The objective function $f : S \rightarrow Y$ with $Y \subset \mathbb{R}$ is a mathematical function, or algorithm for which an optimal value is sought. The domain S is the Cartesian product of domains of possible variables and the range Y of the objective function is a subset of the real numbers $\mathbb{R}$. S is the problem space, also called the search space, and Y is the objective space, or fitness space.

In many real world applications, the optimisation problem has more than one objective to optimise, and often these objectives are conflicting with each other. Multiobjective optimisation deals with conflicting objectives by optimising them simultaneously. Formally, a multiobjective optimisation problem is described as follows:

$$Optimise : F(s) = [f_1(s), f_2(s), ..., f_o(s)] \tag{2.1}$$

where s is a solution to be evaluated, f_i for $i = 1, 2, ...o$ are functions quantifying the quality of the candidate with respect to o objectives (being optimised).

Often, there is a set of conditions that the candidate solutions must satisfy, which are defined as constraints. A multiobjective optimisation problem that satisfies a set of constraints is defined as follows:

$$\begin{aligned} &Optimise : F(s) = [f_1(s), f_2(s), ..., f_o(s)] \\ &such\ that : \Omega_i(s) = 0,\ \Gamma_j(s) \leq 0 \quad i = 1, 2, ..., m,\ \ j = 1, 2, ..., g \end{aligned} \tag{2.2}$$

where Ω_i and Γ_j are the equality and inequality constraints that have to be satisfied in order for the solution to be feasible.

The objective functions are often in conflict with each other and solutions which are optimal according to all objectives may not exist. The optimal result is a group of solutions which represent different combinations of trade-off between the objectives, defined by dominance relations [195].

One solution s^* dominates another solution s if the values of all objective functions at s^* are not worse than those at s, denoted as $f_j(s) \prec f_j(s^*)$, or the value of at least one objective function at s^* is better than the value of the same objective function at variable s, represented as $f_i(s) \lhd f_i(s^*)$. The set of optimal solutions is called Pareto-optimal after work by Pareto [148], defined as follows:

Definition 1 ***Pareto optimal solution:***
A variable s^* is Pareto-optimal if there exist no other s such that $f_i(s) \prec f_i(s^*)$ for all $i = 1, ..., o$ and $f_j(s) \lhd f_j(s^*)$ for at least one j.

In other words, if there exists no other feasible variable s which would be superior in at least one objective while being no worse in all other objectives of s^*, then s^* is said to be Pareto optimal and dominate solution s.

The Pareto-optimal sets of multiobjective problem instances are most often unknown. Multiobjective optimisation approaches usually attempt to approximate the Pareto set and report *nondominated sets*, i.e. the set of solutions which are not dominated among the result solutions found by the algorithm.

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [81] are nature-inspired optimisation methods that are often used for solving NP-hard optimisation problems. They can be defined as a set of steps for quickly identifying results of high-quality in a limited amount of time. EAs are usually used in cases where exact methods are not applicable, and are associated with problems for which an optimal solution cannot be computed by an exact algorithm.

EAs maintain a population of solutions that evolve by means of the genetic operators: the variation procedure ($\mathcal{V}$), which is composed of the mutation operator ($\hat{m}$) and the crossover operator ($\hat{c}$), the selection procedure ($\mathcal{S}$), and the replacement procedure ($\mathcal{R}$), which may also be referred to as survivor selection procedure [49]. The objective is to search the solution space in order to optimise the quality functions $F : S \to \mathbb{R}$.

The main steps of a traditional Evolutionary Algorithm are given in Algorithm 1. The optimisation process starts with a set of solutions (S_0) as initial population, which can be randomly generated, created by applying heuristic rules (e.g. greedy algorithm), or provided by an expert. After the initialisation, EA evolves the population using crossover, mutation and selection operators. The crossover and mutation operators are applied according to predefined crossover and mutation rates ($\hat{c}_r, \hat{m}_r$) respectively. These operators are applied to specific solutions, which are selected according to the selection procedures ($\mathcal{S}$).

The new individuals (offspring) created by the genetic operators are added to the population. The number of offspring generated every iteration t is denoted as λ. The replacement procedure $\mathcal{R}$ selects the solutions that will survive in the next generation and removes as many individuals as required to maintain the prescribed population size μ.

Algorithm 1 Evolutionary Algorithm

procedure EA(F, μ, λ, $\mathcal{R}$, $\mathcal{V}$, $\mathcal{S}$, $\hat{m}$, $\hat{m}_r$, $\hat{c}$, $\hat{c}_r$, *stoppingCriteria*)
    $S_0 \leftarrow$ Random(μ)
    $t = 0$
    while $stoppingCriteria \neq true$ **do**
        for $i \leftarrow 1$, λ **do**
            EVALUATE(F, s_i)
        end for
        $S_t(parents) = \mathcal{S}(S_t)$
        $S'_{t+1} = \mathcal{V}(S_t(parents), \lambda, \hat{m}, \hat{m}_r, \hat{c}, \hat{c}_r)$
        $S_{t+1} = \mathcal{R}(S'_{t+1})$
        $t = t + 1$
    end while
    RETURN(S)
end procedure

Traditional EAs can be classified into four main categories: Genetic Algorithms (GAs) which are usually used for discrete problems, Genetic Programming (GP), which is used for executable expressions, Evolution Strategies (ES), which were devised to optimise the design and analysis of experiments in flexible systems and are used for continuous problems, and Evolutionary Programming (EP), which was conceived as a model to represent adaptive processes. Implementation of Evolutionary Algorithms can be found, that do not belong to any of the above categories, which may have a problem-specific representation and problem-specific operators.

All EA variations use the same principles of the evolutionary process but are adapted with different strategies. They are different in the way the solutions are encoded, the type of selection and recombination procedure and the number of population (parent, offspring). However, all four of them use the same framework of first generating a random initial population which is changed over time by means of genetic operators (variation, selection and replacement procedures). The genetic operators and other parameters of EAs are configured in a way that they ensure the exploration and exploitation of the search space, affecting the performance of the EA.

2.3.1 Genetic Algorithm

Genetic Algorithms were first devised by Bremermann [24], and later on extended by Holland [81]. They were initially developed to model and investigate organic evolution and adaptation capabilities, but they received more attention in the field of optimisation.

In general, Genetic Algorithms are well-studied and well-understood. There is a large body of literature in the GA community regarding the role and the effects of the population size [190, 47, 115], various crossover operators and rates [181, 15, 123], and mutation operators and rates [15, 123].

In Genetic Algorithms, the solutions are represented as genomes, which are encoded using an alphabet with low cardinality, e.g. binary representation, or high cardinality, e.g. integer [22] or floating point [89] representation. Goldberg [66] argues that higher order representations perform better than lower order ones. Janikow and Michalwicz [89] performed an experimental comparison of the binary and floating point representations. The results of the experiments showed that the floating point representation is faster, and gives better results than the binary representation. Furthermore, Davis [38] points out that problem parameters are usually numeric values, hence it makes more sense to represent them as numbers (integer or floating point) rather than binary strings.

The objective function determines the quality of the solution. Initially, the population is created by randomly generating μ solutions, which are then varied by using the crossover and mutation operators to create λ offspring. The first Genetic Algorithms used a single-point crossover operator, which cuts two solutions at a randomly chosen position, and exchanges the two parts of the parents, creating two offspring. Later versions of GAs include many other crossover operators, which often have more than one crossover point. De Jong [39] analysed different crossover

operators, concluding that 2-point crossover operator is the most effective one, and that increasing this number of crossover-points reduces the performance of the algorithm. A high number of crossover points may help in better exploration of the search space, however, it can disrupt the blocks with high quality in the parents.

The mutation operator is normally used with a lower rate compared to the crossover operator, which makes the recombination (i.e. the crossover of the solutions) the primary source of variation in the population. The mutation operator randomly changes the value of one or more elements of the genome. For instance, in single-point mutation, only one bit is randomly changed with a given probability. The mutation operator introduces new information into the population and is considered a random variation. Nevertheless, it is the crossover operator that is considered as the main factor for a thorough search of the possible solutions.

The selection procedure chooses the solutions (or parents) that undergo variation. For example, in binary tournament selection, two solutions are picked at random as contestants in tournaments and the solution with the highest quality is changed by using the mutation operator. The replacement procedure selects the solutions that survive to the next generation. It can be performed by first ranking the solutions and selecting the best ones, or by a fitness proportionate selection, which gives each solution a chance proportional to their fitness.

2.3.2 Evolution Strategies

Evolution Strategies (ESs) [155, 154] were initially devised to optimise the design and analysis of experiments in flexible systems. The ES used for this purpose was composed of two simple steps: (i) random variation of the controlled variables, and (ii) if the quality of the system becomes better, the changes were accepted, otherwise the system returned to the previous state. This algorithm was later called (1+1)-ES,

since it has only one parent ($\mu = 1$) and one offspring ($\lambda = 1$).

Interestingly, this Evolution Strategy with two members was probably the first example of parameter control, in which the mutation operator is adapted using the 1/5 success rule [155]. The 1/5 rule is formulated as follows:

The ratio of successful mutations to all mutations should be 1/5. For greater ratios increase the mutation points; for smaller ratios decrease the mutation points.

Rechenberg [154] added the notion of population into Evolution Strategies, by developing the $(\mu+\lambda)$-ES, in which the population size is greater than one and more than one offspring is produced every iteration. Both parents and offspring are used in the replacement procedure, in which the best of $\mu + \lambda$ solutions survive to the next generation.

The $(\mu + \lambda)$-ES is not a widely used optimisation strategy, however, it inspired Schwefel [166] to further enhance it by introducing the (μ, λ)-ES. The only difference from the previous one is that only the offspring (λ) are used in the replacement procedure, in which the best of λ survive to the next generation. The general representation of Evolution Strategies can be defined as (μ, λ, κ)-ES, where κ is a parameter which determines the maximum number of generations that an individual can survive. In the case of (μ, λ)-ES, $\kappa = 1$, whereas for $(\mu + \lambda)$-ES, κ is unlimited.

Similar to Genetic Algorithms, Evolution Strategies represent solutions with a linear data structure. ESs are usually used for numerical optimisation problems, hence the solutions are represented as real-valued vectors. Genetic Algorithms, on the other hand, are mostly used for discrete optimisation problems.

Evolution Strategies are based on the principle of strong causality, which states that small variations in the solution provoke only small variations in its fitness. Unlike Genetic Algorithms, in Evolution Strategies only the mutation operator is used. Beyer and Schwefel [19] present a comprehensive introduction to Evolution Strategies. One of the notable ES is the Matrix Adaptation-ES [74], which uses a

parameter control mechanism to adapt the mutation step-size, shape and direction.

2.3.3 Genetic Programming

The main difference between Genetic Programming (GP) and the previously discussed Evolutionary Algorithms is the representation of solutions, in which each individual encodes an executable program. Traditionally the genotype is a tree structure, which can be evaluated in a recursive manner. The leaves of the tree represent the operands, whereas the other nodes represent the operators. Due to its applicability with programming languages that are represented as trees, GP is usually used to optimise programs written in Lisp and other functional programming languages. The variation occurs by means of crossover and mutation operators similar to GAs.

2.3.4 Evolutionary Programming

Evolutionary Programming (EP) did not start as an optimisation strategy. It was initially conceived as a model to represent adaptive processes, such as finite state automata for machine learning problems [60]. More recently, it was used as an optimisation algorithm after being adapted by Fogel [59]. Despite being developed independently from each other, some researchers argue that Evolutionary Programming is based on Genetic Algorithms and Evolution Strategies, and often consider it as a special case of ES. Evolutionary Programming does not use a crossover operator.

2.4 Methods for Configuring Parameters of Evolutionary Algorithms

Following the classification introduced by Eiben et al. [45] methods for configuring parameters in Evolutionary Algorithms (EAs) fall into one of two main categories: parameter tuning and parameter control. Methods that fit in the first category find the near-optimal parameter configuration prior to the optimisation process, whereas the second category contains methods that start the optimisation process with a suboptimal parameter configuration and adapt (optimise) their values during the search process.

2.4.1 Parameter Tuning

In optimisation, approximate methods such as EAs are often used because of a lack of knowledge about the fitness landscapes of the problem at hand. However, successful applications are dependent on the usage of suitable parameter values. For previously unknown solutions, only general guidelines regarding the parameter settings are usually available [42]. As a consequence, practitioners tend to choose parameter values based on few trials in which various parameter settings are explored in an attempt to fine-tune an EA to a particular problem. As the number of training instances is usually limited, parameter quality estimates can be orders of magnitude too optimistic, which may lead to strongly impaired performance.

In more rigorous approaches, *ANOVA* or *Design of Experiments* have been used for parameter tuning [20, 192, 17, 144]. This can lead to a combinatorial explosion of parameter value combinations and requires extensive computation. For instance, the tuning of 5 parameters which can take 4 different values requires $4^5 = 1024$ different experimental designs. Considering the stochastic nature of the optimisation

algorithms, each experimental design has to be run more than once to get more reliable results.

A different method which requires less computations is the use of a racing technique proposed by Yuan and Gallagher [192] and Birattari et al. [20]. A conspicuous example of this technique is the *F-Race* algorithm [20]. *F-Race* uses a statistical test to discard poorly performing parameter values from the tuning process. First the parameter values are discretised into different ranges. At the beginning parameter values are sampled from all ranges. The method uses *Friedman two-way analysis of variance by ranks* test to discard ranges that perform significantly worse than the best range. Despite being computationally less expensive than using *ANOVA* or *Design of Experiments*, F-Race still requires a large number of runs. This is due to the initial runs that have to be performed for each initial parameter configuration before starting the discarding technique.

To solve this problem, Balaprakash et al. [16] proposed an improvement of the F-Race technique by sampling from the whole set of configurations instead of individual parameter ranges. If available, a-priori knowledge about the search space can be incorporated into the tuning technique. This method is called *Random Sampling Design F-Race* (RSD/F-Race). By discretising the parameter values into ranges, the possible number of parameter range combinations becomes less than the possible number of parameter value combinations. Nevertheless, a large number of runs is required to assess all of them, since stochastic algorithms, and especially EAs have many parameters which have to be tuned. Another problem is the decision regarding the number of ranges. If parameter values are discretised into narrow ranges, the number of possible configurations increases, increasing the combinatorial complexity of the search. Defining wider ranges reduces the number of experimental runs, but reduces the sampling accuracy.

Since exploring EA parameters is generally very time consuming and computa-

tionally expensive, De Jong [42] suggested that these parameter explorations should be carried out in a more general offline setting. The goal is to find EA parameter configurations that are optimal for a class of problems, and then use these configurations in practice whenever such a problem arises [42]. The author focused mainly in Genetic Algorithms with single-point crossover and bit mutation as genetic operators. Using a set of test functions, well performing parameter values were experimentally determined as follows:

- population size: 50
- crossover rate: 0.6
- mutation rate: 0.001
- selection strategy: elitism

However, these parameter values may not have the same performance with other problems which are different from the test functions used by De Jong. Grefenstette [70] studied the same algorithm parameters as De Jong, in two different experimental settings for the evaluation of the best parameter configurations: the offline performance of the algorithms, which reports the best parameter setting, and the online performance which considers the average performance of the system over the iterations. For the same class of problems as the ones used by De Jong [42], the following parameter settings produced the best performance:

- population size: 30 (online), 80 (offline)
- crossover rate: 0.95 (online), 0.45 (offline)
- mutation rate of 0.01(online), 0.01 (offline)
- selection strategy: elitism (online), pure replacement (offline)

Both De Jong [42] and Grefenstette [70] attempted, with conflicting results, to find the optimal parameter settings that would suit a wide range of optimisation

problems. The task to build a database of well performing parameter values for every possible problem or problem class would require an enormous effort and amount of time, and it might not even be possible.

Some other common parameter settings for the crossover rate, which were obtained by using a Genetic Algorithm are as follows: $\hat{c}_r = 0.6$ [39], $\hat{c}_r = 0.95$ [69], $\hat{c}_r \in [0.75, 0.95]$ [160].

Clearly, recommended values in the literature for the crossover rate and other parameter values vary from each other. This may confuse practitioners when they have to choose an appropriate value for their problem. One common agreement on the crossover rate is that its values should not be too low, and values which are below 0.6 are not used very often.

An insightful survey and classification of tuning methods is presented by Eiben and Smit [49]. Parameter tuning methods are categorized into sampling methods [141, 63], iterative sampling methods [141, 1], model-based methods [51, 92], iterative model-based methods [35, 17], screening methods [164, 156, 80, 98], iterative screening methods [16], meta Evolutionary Algorithms [131, 69], enhanced meta Evolutionary Algorithms [144, 143, 172] and multiobjective meta Evolutionary Algorithms [173].

Parameter tuning is important when comparing different optimisation algorithms. However, one aspect that is not considered by parameter tuning approaches is the fact that the optimisation process evolves in a dynamic way from a more diffuse global search, which requires parameter values suited for the exploration of the fitness landscape, to a more focused local optimisation process, which requires parameter values which help with the convergence of the algorithm. Hence, different algorithm configurations are optimal at different search-stages, a fact that has been demonstrated in an empirical and theoretical way by many researchers [9, 179, 174, 77].

In summary, parameter tuning has been criticised as inadequate for the following reasons:

- Parameters are not independent and must be explored systematically in combination. This is practically impossible due to the large number of combinations.
- Parameter tuning is time-consuming.
- It has been demonstrated that different values of parameters can be optimal at different stages of the search [9, 179, 174, 77]

As a result, instead of tuning the parameters before the run of the optimisation algorithm, methods for controlling them during the run were devised, referred to as parameter control.

2.4.2 Parameter Control

Parameter control addresses the requirement of finding optimal parameter configurations as the search proceeds. More specifically, it describes a process where trials start with initially suboptimal parameter values which are improved during the run-time of the algorithm.

Formally, given a set $\{\upsilon_1, ..., \upsilon_n\}$ of n algorithm parameters, where each parameter υ_i has $\{\upsilon_{i1}, ..., \upsilon_{im}\}$ values that can be discrete numbers or intervals of continuous numbers, parameter control has the task of deriving the optimal next value υ_{ij} to optimise the influence of υ_i on the performance of the algorithm. As an example, when the mutation rate υ_1 is dynamically adjusted by considering 4 intervals ($m = 4$), υ_{12} stands for a mutation rate sampled from the second interval. In the discrete case of optimising the type of mutation operator υ_2, υ_{22} stands for the second operator.

Different parameter control methods exist in the literature, which have been reviewed and classified in different ways [45, 174, 6, 37, 183, 174]. Following the classification proposed by Eiben et al. [45], parameter control methods can be grouped

into three categories: deterministic parameter control, self-adaptive parameter control and adaptive parameter control.

Deterministic parameter control

The parameter values are changed according to deterministic rules which are set a priori in a fashion similar to those used in simulating annealing. Usually the rules are based on time, i.e. a new rule is used every predetermined number of iterations, and no feedback from the search is used. One example is decreasing the mutation rate every w iterations, which was introduced by Fogarty [57]. The author observed a considerable improvement in the performance of the algorithm when compared to static settings. This improvement was achieved for an initial population where all the solutions were composed of zero bits. Similarly, Davis [38] advocates the use of a time-varying schedule for operator rates.

Hesser and Männer [77] theoretically calculated an optimal schedule for changing the mutation rate, which is as follows:

$$\hat{m}_r = \sqrt{\frac{\alpha}{\beta}} \frac{exp(\frac{-\gamma t}{2})}{\mu\sqrt{L}} \tag{2.3}$$

where α, β and γ are constants, μ is the population size and t is the iteration number or time-step. This deterministic schedule increases the mutation rate exponentially in proportion to the time-step. In other words, the mutation rate has a small value at the beginning of the optimisation process and increases over time.

Instead of using time, Bäck [11] presents a deterministic parameter control method which decreases the mutation rate using the distance to the optimal solution. The mutation rate is calculated as follows:

$$\hat{m}_r \approx \frac{1}{2(f(s) - f(s^*)) - L} \tag{2.4}$$

where $f(s)$ is the fitness of the current best solution, and $f(s^*)$ is the fitness of the optimal solution.

Bäck and Schuetz [15] improved this deterministic parameter control method by setting a minimum mutation rate value equal to $\frac{1}{L}$ if a maximum number of evaluations is achieved. This schedule starts with a mutation rate equal to 0.5.

Deterministic parameter control is difficult to accomplish since it is not obvious how to predict the number of iterations the EA will take to converge, and set a parameter control schedule accordingly. Deterministic parameter control faces similar difficulties as parameter tuning, as the parameter adaptation mechanism has to be defined a priori and does not take any notion of the actual progress of the search. One might argue that predefining intervals in the optimisation process and preassigning different parameter settings to each interval is likely to lead to suboptimal values for some problems or instances. For example, parameter reassignment intervals will ideally be shorter when optimising smaller problems, as the search progress will be faster when the problem complexity is lower.

Self-adaptive parameter control

In self-adaptive approaches [11, 53, 40, 104, 12], the parameters to be adapted are encoded in the individuals and evolved simultaneously with the traits of the solution to the problems. Individuals that have a high fitness will survive and propagate these successful parameter values which are assumed to be the reason for the good performance. An insightful review of self-adaptive methods is provided by Bäck [12].

Self-adaptive parameter control is acknowledged as one of the most effective approaches to adapting the parameters of EAs, especially when performing continuous parameter optimisation [43]. They were first applied to Evolution Strategies, which are the algorithms that commonly use self-adaptation. The investigation of the relationship between the number of crossover positions in a chromosome pair

and the population [158] is one of the earliest approaches in this area. The study shows that increasing the number of crossover points increases the scope of the exploration of the search space. Furthermore, the importance of the adaptation of the relationship between the population and the selection pressure is stressed, and recommendations regarding the self-adaptation of the crossover rates are provided. Another self-adaptive parameter control applied to Evolution Strategies is the adaptation of the mutation step size by Rechenberg [154]. Two exemplary self-adaptive approaches in Evolution Strategies are the Covariance Matrix Self-adaptation Evolution Strategy (CMSA-ES) by Kramer [103] and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) by Hansen and Ostermeier [74]. Both methods are based on the covariance computation of the differences in fitness of the best solution and the parent solution.

A notable self-adaptive parameter control method for Genetic Algorithms is the work of Bäck [11], in which the operator rates were adapted. This work was extended to include the population size [13] which was adjusted with an adaptive parameter control method. The self-adaptive method controls the mutation rate between 0.001 and 0.25, which is encoded in the genotype of each solution. The adaptation process starts with random values for the mutation rate of each individual within the predefined range [0.001,0.25]. At every iteration, first the bits that encode the mutation rate are mutated. Next, the new value of the mutation rate is employed to change the bits that encode the solutions.

Bäck et al. [13] adjusted the crossover rate in the range [0.0,1.0], by encoding the values at the end of the solutions representation, similar to the mutation rate. Initially, all crossover rates were initialised by generating random numbers in the range [0.0,1.0]. Tournament selection was employed to select the parent solutions to use in the variation procedure. A random number r was generated, which was compared with the solutions crossover rate. If r was lower than the crossover rate,

then the parent solution was selected for the crossover procedure, otherwise it was uniformly mutated. In essence, the study showed that the algorithm with dynamic parameter values performed better than static ones. However, the authors hypothesise that the main reason of the out-performance is the adaptation of the population size [13].

The adaptation of the mutation rate in a steady state Genetic Algorithm was introduced by Smith and Fogarty [174], in which the authors demonstrate that using a self-adaptive method outperforms fixed mutation rates suggested by other authors.

Since self-adaptive methods encode parameter values into the solution representation, they are generally used to adjust parameters that are applied to individual solutions, i.e. 'local' parameters, such as mutation operator, or crossover operator. Their use for adapting 'global' parameters, i.e. parameters that affect the whole population of solutions, such as population size, and selection procedure, is limited. One of the few approaches that employs a self-adaptive strategy to control a parameter that affects the whole population is the work presented by Eiben and Schut [48]. They investigate the self-adaptation of the selection procedure, in which the tournament size K is decomposed into 'local' parameters that are self-adapted and employed to establish the 'global' parameter K.

The Self-Adaptive Genetic Algorithm (SAGA) presented by Hinterding et al. [78] is similar to the approach of Eiben and Schut [48] in terms of controlling 'local' (individual-level) and 'global' (population-level) parameters by combining self-adaptive and adaptive parameter control. The mutation rate, i.e. the 'local' parameter, was encoded in each gene. Gaussian noise with a standard deviation equal to 0.013 was applied to change the mutation rate, which was used to mutate the rest of the genome in which it was encoded.

Similar to the approach presented by Eiben and Schut [48], Hinterding et al. [78] introduced an adaptive method to adjust the population size. Three different pop-

ulation sizes were maintained during the optimisation process, with initial values of 50,100, and 150. During the adaptation process, the population size was varied in the range [10,1000], based on two main rules. If the fitnesses of two populations became equal, the size of one of the populations was decreased and the size of the other population was increased. This rule is based on the assumption that the sizes of the populations are too similar if the fitnesses converge. The second rule considers the case when the fitnesses are very different, in which the population with the best fitness was used to adjust the sizes of the other two populations.

Nadi and Khader [142] propose a different approach called parameterless GA, which also belongs to the class of self-adaptive parameter control methods. A vector of probabilities is kept for each solution, where each probability represents the occurrence of the respective element on that solution. The probability of each element changes according to the effect of that element on the quality of that solution.

Instead of encoding the parameter values into the solution, Weinberg [187] and Mercer and Samson [131] use a meta-EA, where a second EA optimises the parameters of the EA used to optimise the original problem. The meta-EA approach was also used by Fogel et al. [58] to adapt the mutation rate in Evolutionary Programming (EP). The proposed approach was robust and outperformed static parameter settings. However, the parameter values of the meta-EA are still subject to optimisation.

In brief, self-adaptive parameter control methods alleviate the use of stochastic optimisation algorithms by reducing or eliminating the number of parameters that are required to be set by practitioners. However, one inherent problem faced in self-adaptive parameter control methods is that they increase the size of the search space and the complexity of the optimisation problem, since the EA has to find parameter values which facilitate the effective transmission of the genome information in addition to searching for good solutions to the problem. Another drawback

of self-adaptive control in EAs is the relative slowness of adaptation [45]. It takes time before the optimal value is found [45]. Rudolph [159] explored the theoretical underpinnings of the self-adaptation of the mutation distribution finding that this method gets caught by non-global optima with positive probability even under an infinite time horizon.

Adaptive parameter control

Adaptive parameter control describes the application of separate meta-optimisation methods which collect feedback from the optimisation process to evaluate the effect of parameter value choices and adjust the parameter values over the iterations. The feedback collection mechanism usually involves reporting the result quality.

Generally, adaptive parameter control methods keep a vector of probabilities for each parameter value υ_{ij}, denoted as $P = \{p(\upsilon_{11}), p(\upsilon_{12}), ..., p(\upsilon_{1m_1}), ..., p(\upsilon_{nm_n})\}$, which represents the selection probability for each parameter value, and a vector of the estimate of the overall quality of parameter values, denoted as $Q = \{q(\upsilon_{11}), q(\upsilon_{12}), ..., q(\upsilon_{1m_1}), ..., q(\upsilon_{nm_n})\}$. Whenever a parameter value is used, its current quality estimate $q(\upsilon_{ij})$ is updated.

At each time step, the j^{th} value of the i^{th} parameter is selected with probability $p(\upsilon_{ij})$. Its effect on the performance of the algorithm is denoted as $e(\upsilon_{ij})$. The vector of all parameter effects is equal to $\mathcal{E} = \{e(\upsilon_{11}), e(\upsilon_{12}), ..., e(\upsilon_{1m_1}), ..., e(\upsilon_{nm_n})\}$. The main goal of adaptive parameter control strategies is to adapt the vector of probabilities P such that the expected value of the cumulative effect $E[\mathcal{E}] = \sum_{i=1}^{n} e(\upsilon_{ij})$ is maximised.

The field of adaptive parameter control has been researched widely [74, 56, 181]. An insightful overview and classification of different parameter control methods is given by Eiben et al. [45]. An empirical analysis of adaptive parameter control methods is presented by Fialho et al. [56].

One of the earliest works in this area is the adaptive technique by Davis [38, 37] which rewards the crossover rates that create better solutions. More specifically, the effect of the parameter values on the performance of the algorithm is estimated as the fitness difference between a created solution and its parent(s). As a result, well performing parameter values are more likely to be selected in the next iteration.

Giger, Keller and Ermanni [65] use the best fitness frequency as a feedback collection mechanism to approximate the effect of parameter values on the performance of the algorithm. The solutions are initially divided into classes according to their fitness value. Each class contains solutions with fitness values between $[f_{min}, f_{min} + \epsilon]$, where ϵ is the range size of the fitness class. The fitness classes are employed to compute the *best fitness frequency* (bff). Every iteration t the $bff_t(s_i)$ for each solution s_i is calculated as follows:

$$bff_t(s_i) = \begin{cases} 1 & \text{if} \quad |f(s_i) - f_{best}| \leq \epsilon f_{best} \\ 0 & \text{otherwise} \end{cases}$$

where $f(s_i)$ is the fitness of a solution s_i, f_{best} is the fitness of the best solution in the population and ϵ is the width of the fitness class in which solution s_i belongs. The total bff_t of the current population is calculated as the average of the best fitness values of all solutions, given by Equation 2.5.

$$bff_t = \frac{1}{N} \sum_{i}^{\mu} bff_t(s_i) \tag{2.5}$$

A low value of the bff measure indicates that the fitness of the solutions are very similar. This may mean that the search has stagnated into a local or global optimum, or that the fitness space is flat. On the other hand, a high bff value indicates that the fitness landscape is characterised with steep uni-modal or multi-modal peak(s), in which it is very likely to improve the quality of the solutions

as the search progresses. The *best fitness frequency* is employed to calculate the adaptation rate for parameter values δ as follows:

$$\delta = \begin{cases} -\frac{\delta_{max}}{th}(bff - th) & \text{if} \quad bff \leq th \\ \frac{\delta_{max}}{th}(bff - th) & \text{otherwise} \end{cases}$$

where δ_{max} and the threshold value th control the proportion between exploration and exploitation of parameter values.

A *rank proportional mutation* is introduced in the study of Sewell et al. [167] and an extension of this work was proposed by Cervantes and Stephens [29]. The algorithm [29] uses equal value intervals for the mutation rate to sample from in the range $[\hat{m}_r(min), \hat{m}_r(max)]$. Next, the population is ranked according to the fitness function, and the mutation rate is selected according to these ranks, as shown in Equation 2.6. The better the solution is, the lower the rank.

$$\hat{m}_r(s) = \hat{m}_r(min) + (\hat{m}_r(max) - \hat{m}_r(min)) \cdot \frac{rank(s) - 1}{\mu - 1} \tag{2.6}$$

where $\hat{m}_r(s)$ is the mutation rate applied to solution s, $rank(s)$ is the rank of solution s and μ is the size of the population. The worse an individual's rank is, the higher is the effects on future values of the mutation rate used to produce that individual. As a result, the genes of low-fitness individuals have a lower probability of being propagated to the next iteration, since they become more likely to mutate as their viability decreases.

A similar approach is taken by Srinivas and Patnaik [177], who calculate the effect of the variation operators (mutation rate and crossover rate) based on the maximum fitness f_{max} and the average fitness $\overline{f}$ of the population. The crossover rate $\hat{c}_r$ and mutation rate $\hat{m}_r$ are computed as follows:

$$\hat{c}_r = \frac{k_1(f_{max} - \Delta f(s_{\hat{c}_r}))}{f_{max} - \overline{f}}, \quad \hat{m}_r = \frac{k_2(f_{max} - \Delta f(s_{\hat{m}_r}))}{f_{max} - \overline{f}} \tag{2.7}$$

where k_1 and k_2 are two constants selected in the range $[0.0, 1.0]$ and $\Delta f(s_{\hat{c}_r})$ $\Delta f(s_{\hat{m}_r})$ are the fitness gain of solutions created by applying c_r and m_r compared to their parents. The operator rates calculated by Equation 5.5 are inversely proportional to the difference between the maximum fitness and the average fitness of the population. The operator rates are increased if the difference between the maximum fitness in the population and the fitness of the solution created by applying the operator rate $(f_{max} - f)$ is low, i.e. lower values of operator rates are applied to high fitness solutions, and higher values of operator rates are used to change low fitness solutions. A low difference means that the fitnesses of the solutions are too similar, hence the operator rates are increased to introduce more diversity in the population.

Other approaches adapt multiple parameters simultaneously [94, 112, 5]. Julstrom [94] controls the mutation and crossover rates, by adapting the ratio between them based on the relative performance. Crossover and mutation operators are used individually to create different solutions, and the reward is given to each of them according to the fitness difference between the created solution and their parent(s).

Similarly, Lis and Lis [112] control three different parameters simultaneously: mutation rate, crossover rate and population size. Each of the parameters could have three possible values, which were applied to parallel algorithm instances. The populations of the parallel instances were compared with each other after a certain number of iterations. The parameter values that created the population with the best fitness were used more frequently in the next iterations.

A self-adaptive mutation rate a self-adaptive crossover rate, and an adaptive pop-

ulation size were investigated by Bäck et al. [13]. The study showed that adapting the population size significantly improves the performance of the algorithm. The authors argue that the population size should not be constant during the optimisation process.

Usually, adaptive parameter control methods [112, 94, 177, 38, 37] use the fitness improvement as an indication of the effect of parameter values on the performance of the algorithm. Compass [125] uses also a diversity measure to reward parameter values. Diversity refers to the Hamming distance in the genotype of the solutions, which computes the number of genes in which the corresponding values are different. The overall proposed feedback collection strategy uses the weighted sum of the fitness improvement and the diversity of the offspring.

Compass calculates a vector $\overrightarrow{o}_t\ (s_{v_{ij}}) = (\Delta d_t(s_{v_{ij}}), \Delta f_t(s_{v_{ij}}))$ for every parameter, where $\Delta d_t(s_{v_{ij}})$ is the diversity change in solution s caused by parameter value v_{ij} at iteration t, and $\Delta f_t(s_{v_{ij}})$ is the fitness change. Compass selects a vector $\overrightarrow{c}$ that has an angle θ defined in the interval $[0, \frac{\pi}{2}]$, which represents the desired balance between fitness and diversity. If θ has a value closer to zero, more importance is given to the diversity of the solutions, hence favouring exploration, whereas if θ has a value closer to $\frac{\pi}{2}$ the exploitation of good solutions becomes more important.

Figure 2.1 depicts an example of Compass, in which $\overrightarrow{o}_t\ (s_{v_{i1}})$ and $\overrightarrow{o}_t\ (s_{v_{i2}})$ represent the performance of the algorithm when parameter values v_{i1} and v_{i2} are applied. Vector c represents the desired trade-off between diversity and fitness, which determines the parameter value for the next iteration.

Parameter θ has to be specified by the user, who has to know how to define the compromise between obtaining good results and maintaining the diversity of the solutions. The problem is that finding the appropriate value for θ requires insight into the given problem instance, and therefore it may not be practicable. Compass was further improved into a different method by Maturana et al. [124], using Pareto

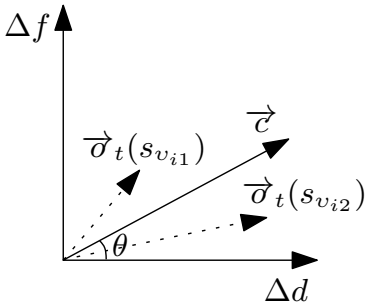


Figure 2.1: Compass.

dominance [148], which enables a trade-off between the fitness and diversity of the solutions and does not require the use of θ.

A different diversity measure is introduced by Giger et al. [65], which estimates the distribution of the solutions in the search space, calculated as the Euclidean distance of a solution $s_{v_{ij}}$ to the best solution so far s_{best}, given in Equation 2.8.

$$euc_t(s_{v_{ij}}) = \sqrt{\sum_{r=1}^{g} s_{v_{ij}}(r)} \tag{2.8}$$

where t is the current time-step (iteration), g is the number of genes of the genotype and $s_{v_{ij}}(r)$ represents the r^{th} gene in solution $s_{v_{ij}}$. The authors suggest the use of weighting factor w for each gene as follows:

$$euc_t^w(s_{v_{ij}}) = \sqrt{\sum_{r=1}^{g} w_r s_{v_{ij}}(r)} \tag{2.9}$$

The diversity measure for parameter value v_{ij} at iteration t is as follows:

$$d_t(s_{v_{ij}}) = \begin{cases} 0 & \text{if} \quad f(s_{v_{ij}}) \leq f(s_{best}) \\ (1 + euc_t(s_{v_{ij}})/euc_t(max))^{-1} & \text{otherwise} \end{cases}$$

where $euc_t(max)$ is the maximum euclidean distance between the best solution and all solutions. In essence, a balance between the best fitness frequency and the

diversity of the solutions is sought.

Most of the adaptive parameter control methods found in the literature [33, 37, 82, 84, 94, 114, 163, 183, 65] belong to the class of probability matching techniques, which update the probability vector P such that the probability of applying parameter value υ_{ij} is proportional to the quality of that parameter value, i.e. the proportion of the estimated quality $q(\upsilon_{ij})$ to the sum of all quality estimates $\sum_{s=1}^{n} q(\upsilon_{is})$.

For instance, Corne et al. [33] introduce an adaptive method to control mutation rate in an Evolution Strategy application, in which the selection probability for each parameter value is proportional to their usage frequency and the fitness distance correlation of the solutions created by employing that parameter value.

Igel and Kreutz [84] use only the fitness gain of the created solution ($f(s_{\upsilon_{ij}})$) - with respect to the fitness its parent ($f(s)$) or the fitness of the best solution so far ($f(s_{best})$) - as an indication of the quality of a parameter value $q(\upsilon_{ij})$. The selection probability for each parameter value at time step t is calculated as follows:

$$p_t(\upsilon_{ij}) = \begin{cases} \frac{w}{\sum_{r=1}^{m} q_t(\upsilon_{ir})} \sum_{r=t-\tau}^{t-1} q_r(\upsilon_{ij}) + (1-w)p_{t-1}(\upsilon_{ij}) & \text{if} \quad \sum_{r=1}^{m} q(\upsilon_{is}) > 0 \\ \frac{w}{m} + (1-w)p_{t-1}(\upsilon_{ij}) & \text{otherwise} \end{cases}$$

where m is the number of possible values for parameter υ_i, τ is the time window considered to approximate the quality of a parameter value and $w \in (0, 1]$ is the weight given to each parameter value. For w equal to 1 and τ equal to 1 the algorithm is similar to the one proposed by Hong, Wan and Chen [82].

Igel and Kreutz [84] introduce a minimum selection probability p_{min} for each parameter value, to ensure that under-performing parameter values do not disappear during the optimisation, since they may be beneficial in the later stages of the search. The selection probability for each parameter value is calculated using Equation 2.10.

$$p'_t(\upsilon_{ij}) = p_{min} + (1 - mp_{min})\frac{p_t(\upsilon_{ij})}{\sum_{r=1}^{m} p_t(\upsilon_{ir})} \tag{2.10}$$

Dividing the selection probability by the sum of selection probabilities of all possible parameter values normalises the values in the interval [0,1].

Probability matching techniques [33, 37, 82, 84, 94, 114, 163, 183, 65] have been criticised for the fact that the probability values resulting from the reward allocations poorly reflect the relative differences in algorithm performance. Values with vastly superior performance may only be differentiated by a marginal increase of the probability of being chosen in the next step.

Adaptive Pursuit (AP) [181] is a different parameter control strategy, which rewards the best performing parameter values j^* by increasing their selection probability by p_{max} as shown in Equation 2.11.

$$p_t(\upsilon_{ij}) = \begin{cases} p_{t-1}(\upsilon_{ij}) + \beta(p_{max} - p_{t-1}(\upsilon_{ij})) & \text{if } j = j^* \\ p_{t-1}(\upsilon_{ij}) + \beta(p_{min} - p_{t-1}(\upsilon_{ij})) & \text{otherwise} \end{cases} \tag{2.11}$$

where $p_{max} = 1 - (m-1)p_{min}$ and m is the number of parameter values. The Adaptive Pursuit parameter control is a greedier strategy compared to Probability Matching. Increasing the selection probabilities of the best parameter values ensures an appropriate difference in probabilities depending on experienced performance.

Even though every parameter value is selected from time to time, in practice the Adaptive Pursuit algorithm would have to wait for a number of iterations before realising that some new parameter value has become the best one. To address this problem, the Dynamic Multi-Armed Bandit (DMAB) [56] was introduced, which completely recalculates the probabilities when a change in the effects distribution is detected by using a change detection test, in this case the statistical Page-Hinkley (PH) test. The PH test checks whether the quality of the parameter values has

changed. When a change is detected, the algorithm is restarted. As a result, DMAB can quickly identify the new best parameter value without being slowed down by old information.

2.5 Parameters to Configure

The first decision made by an EA expert that affects the performance of the algorithm is the problem representation, which is a distinguishing feature not only between different problems, but also between different classes of Evolutionary Algorithms. The main difference lies in the data structures used to represent the solutions. For example, Genetic Algorithms (GA) have a linear data structure with a fixed length over time. In Evolution Strategies (ES), the data structure does not have to be fixed over time. In Evolutionary Programming (EP) and Genetic Programming (GP) the data structures are in the form of finite state machines and parse trees, which are not linear and change their size and shape over time.

The choice of the representation of the problem clearly affects the performance of the algorithm [43]. This parameter is not easy to dynamically configure during the run of the algorithm [43]. 'Delta Coding Algorithm' [122] and 'Adaptive Representation Genetic Optimizer Technique' (ARGOT) [168] are two of the few attempts to adapt the representation of the problem online, by modifying the encoding of the function parameters. The motivation behind the 'Delta Coding Algorithm' is to ensure the trade-off between the fitness and the diversity of the solutions, whereas ARGOT adapts the representation based only on the fitness of the solutions.

Once the choice regarding the representation of the problem has been made, a number of different algorithm parameters have to be adjusted, such as:

* Population size (μ).
* Offspring number (λ).
* Selection procedure ($\mathcal{S}$).
* Replacement procedure ($\mathcal{R}$).
* Variation procedure ($\mathcal{V}$):

* Mutation operator ($\hat{m}$),
* Mutation rate ($\hat{m}_r$),
* Crossover operator ($\hat{c}$),
* Crossover rate ($\hat{c}_r$).

A comprehensive overview of the research in various parameters of Evolutionary Algorithms is presented by Eiben and Schut [48]. The study shows that the majority of the approaches were concerned with the variation procedure (approximately 46% of the publications). The second most researched topic was the population of Evolutionary Algorithms (approximately 29% of the studies). A high level description and background of the parameters of an EA that have an essential effect on the performance of the algorithm, and therefore necessitate careful configuration, is given in the next sections.

2.5.1 Population Size

Setting the population size in an EA is a difficult task. It has been argued that problems with different size and difficulty require different population sizes [116]. When the problem at hand is simple and unimodal, the ideal population size (μ) is relatively small, since not much diversity is required during the optimisation process. In this situation, the algorithm converges easily by performing a local search. One might argue that if the problem is unimodal, selecting an EA to optimise it is not a good solution. However, it is often not possible to know beforehand how the problem looks like.

On the other hand, when the problem is multimodal (has more than one optimum), or when solving a multiobjective optimisation problem, the population has to be large enough to explore all the important regions of the landscape and increase the chance of discovering the optimal set of solutions. Setting the population size is

a non-trivial task, since different problems have different ideal population sizes [115]. When there is no information about the problem and its modality, setting the population size before the optimisation process may lead to the optimisation algorithm performing sub-optimally. Lobo [115] shows that controlling the population size during the optimisation process can be beneficial in certain problems. A similar view is presented by Bäck et al. [13], who demonstrate that varying the population size significantly improves the performance of the algorithm [13].

De Jong [43] asserts that by increasing the population size, the chance of any EA finding the best solutions for any type of problem increases. His experiments show that the average performance of an Evolutionary System $(m, n = 1)$ increases when the population size is increased. Moreover, De Jong demonstrates that during the initial iterations of the optimisation process, the performance of the algorithm is highly influenced by a bigger population size, while this effect decreases over time. De Jong [43] states that this results can be generalized to any type of traditional EA, and suggests that the population should start with a large value and decrease during the later iterations.

Other studies [175, 47, 7, 115, 78] of the same nature have focused on the efficiency of controlling the population size during the run of the optimisation algorithm. An insightful review of methods for setting the population size in GAs is provided by Lobo and Lima [116], The authors also provide recommendations for further research on the adaptation of the population size, such as testing the results on problems with known requirements for appropriate population settings.

Arabas, Michalewicz and Mulawka [7] proposed a Genetic Algorithm with Varying Population Size (GAVaPs), which assigns a lifetime to each solution when it is created. The lifetime variable is reduced every iteration until it becomes zero. As a result, the solution is removed from the population. The fitness of the solutions is used to control the lifetime for each individual $(lt(s_i))$, determined using (i)

proportional allocation (lt_p), which is a fitness proportionate allocation, (ii) linear allocation (lt_l), which considers the fitness of the solution $f(s_i)$ in comparison with the best and the worst fitness in the population $f(s_{best}), f(s_{worst})$, and (iii) bilinear allocation (lt_b), which makes a trade off between proportional and linear allocation. Considering a maximisation problem, the lifetime of an individual s_i is calculated as follows:

$$\begin{aligned} lt_p(s_i) &= \arg\min((lt_{min} + \frac{(lt_{max} - lt_{min})f(s_i)}{2\overline{f}}), lt_{max}) \\ lt_l(s_i) &= lt_{min} + \frac{(lt_{max} - lt_{min})(f(s_i) - f_{worst})}{f(s_{best}) - f_{worst}} \end{aligned} \tag{2.12}$$

$$lt_b(s_i) = \begin{cases} lt_{min} + \frac{(lt_{max} - lt_{min})(f(s_i) - f(s_{worst}))}{2(\overline{f} - f(s_{worst}))} & \text{if} \quad \overline{f} \geq f(s_i) \\ \frac{lt_{min} + lt_{max}}{2} + \frac{(lt_{max} - lt_{min})(f(s_i) - \overline{f})}{2(f(s_{best}) - \overline{f})} & \text{otherwise} \end{cases}$$

where lt_{max} and lt_{min} are the maximum and minimum possible lifetime, which have to be decided before the optimisation process, and $\overline{f}$ is the average fitness of the population. The experimental studies presented by Arabas, Michalewicz and Mulawka [7] show that the linear strategy has the best performance but is the most computationally expensive. The bilinear allocation, on the other hand, is the least computationally expensive methods, however the performance is the worst. The authors argue that the proportional strategy has a moderate performance and cost.

Adaptive Population size Genetic Algorithm (APGA) [13] was introduced by Bäck, Eiben and Van der Vaart, which is and extension of the Genetic Algorithm with Varying Population Size (GAVaPs) [7]. The main differences between the two algorithms is that APGA considers a steady-state Genetic Algorithm and no lifetime is specified for the best solution.

Hinterding et al. [78] employs an adaptive mechanism to control the population size in a Genetic Algorithm (GA) with three sub-populations (50, 100 and 200 solutions), which are optimised in parallel. After a certain number of function evaluations, the fitness of the sub-populations is assessed and the size of the sub-populations is changed according to the feedback received. A lower bound of 10 and an upper bound of 1000 is maintained during the iterations.

Eiben, Marchiori and Valkó [47] introduce the Population Resizing and Fitness Improvement Genetic Algorithm (PRoFIGA), in which the population size is adjusted according to the improvement of the best fitness in the population. The population grows if the best fitness in the population increases or if the best fitness does not change for a specific number of iterations. If the best fitness in the population $f(s_{best})$ increases, the population size increases with a growth rate $\Delta\mu$ computed as:

$$\Delta\mu = \alpha(eval_{max} - eval_t)\frac{f_t(s_{best}) - f_{t-1}(s_{best})}{f_1(s_{best})} \tag{2.13}$$

where α is a parameter selected from the interval (0,1), $eval_{max}$ and $eval_t$ are the maximum number of fitness evaluations and the current evaluation number (time step t), $f_t(s_{best})$, $f_{t-1}(s_{best})$ and $f_1(s_{best})$ denote the current (time step t), the previous (time step $t-1$) and the initial best fitness values (time step 1). If the best fitness value does not improve in a certain number of evaluations, the authors propose to increase the population size by a factor y. If neither of the above situations occur, the population size is decreased. The main idea behind this approach is to use large population sizes for the exploration and small population sizes for the exploitation of the search space. The experimental study shows that adapting the population size is beneficial for the performance of the algorithm. Nevertheless, there is general agreement about the difficulty of adjusting the population size, since it interacts in

complex ways with other parameters of an EA, such as the selection and replacement procedures.

2.5.2 Selection Procedure

The selection procedure ($\mathcal{S}$) is performed when deciding on the parents which will be used to produce the offspring. The greedier the selection strategy is (i.e. the higher the demand on the quality of the solutions), the faster the algorithm converges. However, the chance of finding the optimal solution(s) may decrease, since the population is more likely to get stuck in local optima. Thus, the selection procedure should allow for the exploration of the search space, especially during the initial iterations, while focusing on the best solutions during the last iterations. This suggests that an adaptation of the selection strategy would improve the performance of the algorithm.

Two aspects of selection procedure are important for the adaptation process: the type of selection procedure and the parameters of the selection procedure. Interestingly, none of them is commonly adapted at run-time. Most of the adaptive methods for the selection procedure use the Boltzmann selection strategy [14], which uses a cooling mechanism similar to Simulated Annealing to modify the selection pressure. Bäck [10] provides a characterisation of the selection mechanisms. Goldberg and Deb [67] compared different selection mechanisms in genetic algorithms, and found that proportionate selection is significantly slower then other selection mechanisms. Linear ranking and binary tournament selection (a type of tournament selection where only two solutions are picked at a time) showed similar performance. Vajda, Eiben and Hordijk [184] compared various selection mechanisms. The study shows that using parameter control for the selection mechanism produces better results than using a constant value. The authors argue that the selection mechanism is

problem dependant, however, the self-adaptive tournament selection out-performed other methods in the majority of the problems.

Proportional selection

Proportional selection was first introduced by Holland [81]. The selection probability of each solution $s_i, i \in \mu$ is based on probabilities assigned to individuals based on their fitness ($f(s_i)$) as follows:

$$\mathcal{S}(s_i) = \frac{f(s_i)}{\sum_{j=1}^{\mu} f(s_j)} \tag{2.14}$$

The selection probability of each solution is directly proportional to its fitness and indirectly proportional to the sum of the fitnesses of all solutions. The higher the fitness of a solutions, the more probable it is for it to be selected.

Tournament selection

Tournament selection is a mechanism where solutions are picked at random as contestants in tournaments and the solution with the highest quality among the other members of the tournament is the winner (either to generate the next offspring, or as a survivor of the next generation). The tournament size provides a natural parameter for the adaptation process. A deterministic rule is presented by Vajda, Eiben and Hordijk [184] to adapt the tournament size k defined as follows:

$$k_t = \begin{cases} \frac{t(p_2 - p_1)}{1000} + p_1 & \text{if} \quad t \in [0, 1000] \\ p_2 & \text{otherwise} \end{cases}$$

where p_1 and p_2 are parameters that have to be set before the optimisation process. This deterministic rule linearly increases or decreases the tournament size from p_1 to p_2 for the initial 1000 iterations and keeps it constant for the rest of the optimisation.

Linear ranking

Linear ranking was first introduced by Grefenstette and Baker [70]. The solutions are ranked according to the fitness value, and the probability of selection is determined by Equation 2.15.

$$\mathcal{S}(s_i) = \frac{\eta^+ - (\eta^+ - \eta^-) \cdot \frac{i-1}{\mu-1}}{\mu} \tag{2.15}$$

where μ is the number of solutions, η^+ and η^- are the maximum and minimum expected values respectively, fulfilling the constraints $1 \leq \eta^+ \leq 2$ and $\eta^- = 2 - \eta^+$. These two values are used to determine the slope of the linear function. η^+ can be adapted in the interval [1,2].

Boltzmann selection

Vajda, Eiben and Hordijk [184] introduce a Boltzmann selection mechanism with a Riemann-Zeta annealing schedule (BSR), which calculates the probability of selecting a solution $\mathcal{S}(s_i)$ as follows:

$$\mathcal{S}_t(s_i) = \frac{e^{\gamma_t f(s_i)}}{\sum_{j=1}^{\mu} e^{\gamma_t f(s_j)}}, \gamma_t = \gamma_0 \sum_{k=1}^{t} \frac{1}{k^\alpha} \tag{2.16}$$

where e is the Blotzmann distribution [118], and γ_0 and α are parameters that have to be set before the optimisation. The annealing temperature γ_t is is an example of a deterministic parameter control, in which the selection pressure depends on time.

Fuzzy tournament selection

A Fuzzy Tournament Selection (FTS) was introduced by Vajda, Eiben and Hordijk [184], which is based on the adaptive method presented by Herrera and Lozano [76]. The approach adjusts the tournament size k as follows:

$$k_t = (\alpha + 0.5)k_{t-1} \tag{2.17}$$

where α is the modification parameter computed by employing the fuzzy rules that are based on the genotypic and the phenotypic diversity of the population.

2.5.3 Variation Procedure

The majority of the work in adaptive optimisation considers the variation procedure ($\mathcal{V}$) as a parameter to control. Evolutionary Algorithms build the next solutions by using two different variation procedures: the mutation operator, which uses a single parent, and the crossover operator, which is carried out using two or more parents. Mutation and crossover operators make a random variation in one or more solutions in the current population to generate new solutions, without knowing if this change is going to produce better or worse candidate solutions. The replacement procedure is in charge of accepting or not the changes made and determining the solutions that survive to the next generation.

Both operators change the current population by either introducing new information (mutation operator), or by recombining the existing information (crossover operator). Setting these two parameters greatly affects the way the search progresses, by fine-tuning the balance between exploration and exploitation of the search space.

Mutation operator

The tuning and control of the mutation rate is one of the most researched topics [15, 123, 48, 78, 9, 5]. For instance, there are several recommendations in the literature about well-performing mutation rates. However, the recommended values are for specific problems, and the mutation rate values often vary from work to work. For example, De Jong [39] recommends a mutation rate of 0.001, whereas for the same

problems Schaffer et al. [160] found that the mutation rate should be in the range $[0.005, 0.01]$.

Bremermann et al. [23] propose that the mutation rate should depend on the length of the string (i.e. the number of elements in the genotype) and should be equal to $\frac{1}{L}$ where L is the length. Smith and Fogarty [174] compared this mutation rate with different fixed mutation rates, demonstrating that the mutation rate equal to $\frac{1}{L}$ outperformed other values.

One of the first works in the adaptation of the mutation operator is the $1/5^{th}$ rule of Rechenberg [154], which controls the mutation step size of Evolution Strategies (ESs). The majority of parameter control methods focus on controlling the mutation rate [5, 9]. Often, the mutation rate has been controlled in conjuction with the crossover rate [5]. An innovative way to control multiple parameter is the work introduced by Bäck [9], where the interaction of mutation rate and the selection mechanism are studied and controlled with a self-adaptive Genetic Algorithm. Similarly, Hinterding et al. [78] uses a self-adaptive mechanism to control the mutation rate and an adaptive mechanism to control the population size.

Apart from the mutation size and mutation rate, one can control the type of mutation operator used. Different types of mutation operators exist, such as single-point mutation, transposition mutation, and uniform mutation. Single-point mutation is the random change of a single element in one of the solutions from the population to produce a new candidate solution. Transposition mutation on the other hand randomly exchanges the values of two elements in a solution to produce a new solution. In this case information is only exchanged and no new information is introduced. Finally, uniform mutation considers all the elements of the solutions for a random change according to a probability defined by a mutation rate ($\hat{m}_r$). Uniform mutation changes more than one element of the solution, introducing more new information than single-point mutation.

Crossover operator

This parameter has been extensively investigated in the context of adaptive parameter control [181, 15, 123]. The work of Davis [38] is one of the first attempts to control different crossover operators ($\hat{c}_i$), with the respective crossover rates ($\hat{c}_r(\hat{c}_i)$) in the application of a Genetic Algorithm. At every iteration, several crossover rates are applied simultaneously to create the new offspring. The effect of each crossover operator is approximated using the difference in fitness gain of the offspring with respect to the best solution in the population, called *local delta* and denoted as d_i. The rate of each the crossover operator is calculated as:

$$\hat{c}_{r_t}(\hat{c}_i) = 0.85\hat{c}_{r_t}(\hat{c}_i) + d_i \tag{2.18}$$

where t is the current iteration and 0.85 is the weight given to the value of the crossover rate from the previous iteration.

The crossover operator and its rate have also been controlled using self-adaptive parameter control mechanisms. Schaffer and Morishima [161] introduced a self-adaptive method for controlling the position in a solution where the crossover occurs, called *punctuated crossover*, which performed better than a single-point crossover operator [161].

Spears [176] uses a self-adaptive mechanism to control the type of crossover operator. An additional bit is added at the end of the solution representation, which is used to decide if a two-point or uniform crossover operator is to be used. The increased performance of the optimisation algorithm is attributed to the diversity of the variation operators, which leads to a better exploration of the search space.

Crossover rate has also been controlled in combination with other parameters such as mutation rate [5]. Another interesting multiple parameter control mechanism is offered by Lis and Lis [113]. The mutation rate, the crossover rate and the

population size of a parallel GA are controlled in an adaptive way. This approach uses predefined possible values that the parameters can have, and based on the performance of the algorithm, the successful parameter values are used for the next iteration.

Apart from the crossover rate, it is possible to control the type of crossover operator. Some options include single-point crossover and uniform crossover. Single-point crossover splits two parent solutions into two by choosing a random point and mixes the information from the parents to create two new solutions. Uniform crossover swaps the assignments between solutions. Unlike single-point crossover, the uniform crossover enables the parent chromosomes to contribute to the gene level rather than the segment level, by mixing ratio between two parents.

The two variation procedures, mutation and crossover operators, usually make a random variation in one or more solutions in the current population to generate new solutions, without knowing if this change is going to produce better or worse candidate solutions. The replacement procedure is in charge of accepting or not the changes made and determining the solutions that survive to the next generation.

2.5.4 Replacement Procedure

The two variation operators, mutation and crossover operators, which are described in Section 2.5.3, usually make a random variation in one or more solutions in the current population to generate new solutions, without knowing if this change is going to produce better or worse candidate solutions. The replacement procedure is in charge of accepting or not the changes made and determining the solutions that survive to the next generation.

The adaptation of the replacement procedure has not received as much attention as the variation operators. One of the earliest works on the effect of the replacement

procedure is the comparison of two replacement procedures by Grefenstette [69], *pure replacement*, in which the offspring substitute the current population and *elitist replacement*, where the best solutions survive to the next generation. The results of the experiments show that elitism improves the performance of the algorithm.

It has also been proven that the replacement procedure affects the diversity of the population [7], which can cause a premature convergence of the algorithm. Arabas, Michalewicz and Mulawka [7] argue that the balance between population diversity and 'replacement pressure' should be maintained, since population diversity is strongly related to the 'replacement pressure' - a high replacement pressure decreases the diversity of the solutions and vice versa. To address this problem, a Genetic Algorithm with Varying Population Size (GAVaPS) [7] was introduced, which adjusts the population size and diversity based on the fitness and the lifetime of the solutions calculated in Equation 2.12.

2.5.5 Number of Offspring

De Jong [43] argues that changing the number of offspring (λ) during the optimisation process affects the performance of the algorithm. The author performed a set of experiments using a fixed population size, in which the number of offspring was decreased over time. The results showed that in a multimodal problem a higher number of offspring results in a lower performance of the algorithm. The number of offspring and the population size seem to have opposite effects in the performance of the algorithm. One might ask if it is reasonable to control them individually; this question has yet to be answered. Jansen and De Jong [90] demonstrate in their study that the number of offspring is a difficult parameter to control.

2.6 Summary

The review of the methods concerned with setting EA parameters [33, 78, 113, 15, 123, 48, 43] shows that the way the EA parameters are configured affects the performance of the algorithm. Furthermore, empirical and theoretical evidence shows that optimal algorithm configurations are different not only for different problems and problem instances, but also for different stages of the optimisation process of the same problem instance [13, 9, 179, 174, 77].

Acknowledging these facts, a plethora of studies proposed adjusting algorithm parameters during the optimisation process: by changing parameter values according to a predefined schedule [57, 15], by encoding parameter choices into the representation of the solutions [13, 142, 154, 174, 131, 158], or by using feedback from the search [33, 37, 82, 84, 94, 114, 163, 183, 74, 56, 181].

Self-adaptive parameter control integrates the search for optimal parameters into the optimisation process itself - usually by encoding parameter settings into the genotype of the solution to evolve. Extending the solution size to include the parameter space obviously increases the search space and makes the search process more time-consuming. Moreover, by encoding parameter values into the solution representation, the search for optimal parameter values may be subject to premature convergence, i.e. convergence towards a suboptimal solution, which is a common problem faced by the search for optimal solutions [29, 132]. Based on an empirical investigation of parameter control methods, Tuson and Ross [183] suggest that for the parameter adaptation to occur in a reliable way, the adaptation mechanism should be separated from the main algorithm, i.e. the parameters should not be encoded into the representation.

Adaptive parameter control algorithms adjust parameter values in an informed way, based on the effect they have in the performance of the algorithm. Adaptive

strategies have successfully been employed to control various algorithm parameters, such as the type of crossover operator and its rate [5, 13, 181, 15, 123], the type of mutation operator and its rate [13, 15, 123, 48, 78, 9, 5], as well as the population size [13, 175, 47, 7, 115, 78]. A big advantage of adaptive parameter control algorithms when compared with self-adaptive methods is that they do not increase the search space. Furthermore, there is no co-evolution between solution space and parameter space, which removes the possibility that the search for optimal parameter values is subject to the problems associated with the search for optimal solutions, such as the premature convergence of the algorithm.

Chapter 3

Methodology

3.1 Introduction

In the previous chapter, we presented some implementations of Evolutionary Algorithms (EAs), various algorithm parameters and methods used to configure these parameters. The importance of configuring Evolutionary Algorithms is discussed throughout the different sections of the Chapter 2, and the summary of the related work (Section 2.6) highlighted the effect of parameter values on the performance of Evolutionary Algorithms and the crucial reasons for adjusting parameter values during the optimisation process.

Thus far, methods for configuring Evolutionary Algorithms have been discussed at a high level and with a wide focus. This chapter presents a more confined review of the literature, with the aim of determining the gaps in current research and defining the scope of the work presented in this thesis. This leads to the discussion of the methodological aspect of the research conducted in this thesis: the research problems, the research objectives, the research questions, and the research method used to address the research objectives and validate the solutions proposed to solve the research problems.

3.2 Research Problems

The high-level focus of this thesis is parameter configuration of Evolutionary Algorithms. Section 2.4 described and compared two different methods used to configure parameter values: parameter tuning which configures parameter values before the optimisation process, and parameter control, which dynamically adjusts parameter values during the run of the optimisation algorithm. Parameter control is more effective than parameter tuning, since different algorithm configurations may be optimal at different stages of the search [9, 179, 174, 77]. Hence, the focus of this thesis is in the adaptation of parameter values during the optimisation process.

Parameter control methods are classified into three groups as described in Section 2.4: deterministic, self-adaptive and adaptive. Deterministic parameter control changes parameter values based on a predefined schedule composed of intervals with preassigned parameter values, e.g. decreasing mutation rate by a certain amount every iteration. Using a predefined schedule is likely to lead to suboptimal values for some problems or instances, since smaller problems may require shorter intervals for the schedule, as the search progress will be faster when the problem complexity is lower. Alternatively, the search for optimal parameters can be integrated into the optimisation process itself - usually by encoding parameter settings into the genotype of the solution to evolve [15, 53, 40], i.e. self-adaptive parameter control. Extending the solution size to include the parameter space obviously increases the search space and makes the search process more time-consuming [45].

The approach proposed in this thesis falls into the category of adaptive parameter control, in which feedback from the optimisation process is collected and employed to evaluate the effect of parameter value choices and adjust the parameter values over the iterations. Adaptive parameter control does not use a predefined schedule and does not extend the solution size, which makes it a more effective way for

controlling parameter values during the optimisation process. The main steps of the whole optimisation process using adaptive parameter control are shown in Figure 3.1. These steps are shared among all existing Evolutionary Algorithms that use adaptive parameter control.

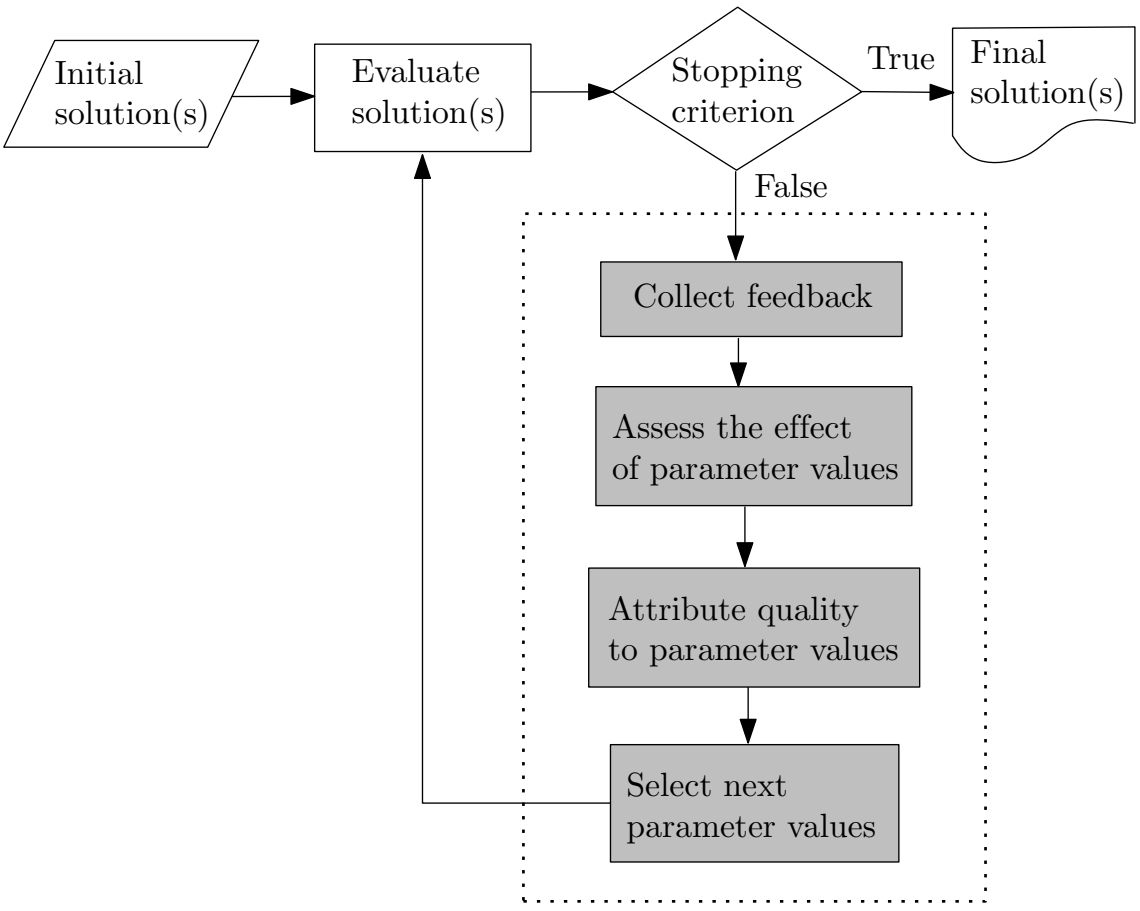


Figure 3.1: The main steps of optimisation with Evolutionary Algorithms using adaptive parameter control.

The optimisation process starts with a set of solutions (initial population) which can be randomly generated, created using expert knowledge, or constructed using some heuristic, such as a greedy algorithm. This population is evolved in iterations until a stopping criterion is achieved. The new generations of solutions are created with the help of four genetic operators: the crossover operator, the mutation operator,the selection operator and the replacement operator. The genetic operators,

and other elements of an Evolutionary Algorithms that are involved in the optimisation process, such as the probabilities of applying the genetic operators, the size of the population, the number of new solutions produced every iteration, etc. are the parameters of the algorithm which can be adapted by the parameter control method.

Every iteration, the solutions created are evaluated using the fitness function(s). The output from the evaluation process provides valuable information for the adaptive parameter control methods, since it can be used to asses the effect of the parameter values on the performance of the optimisation algorithm. This information is processed by the feedback collection strategy, which measures certain properties of an optimisation run, such as the fitness and the diversity of the solutions. The feedback collection mechanism records the change in these properties as the performance of the algorithm.

The feedback from the search is used by the parameter value effect assessment strategy to approximate the cause of the change in the properties of the solutions during the optimisation process. Parameter control methods utilise the approximated effect for a projection of the parameter values performing well to the next iterations, based on certain rules, such as the average effect in the last n iterations, instantaneous effect, the total usage frequency of parameter values, etc. This set of rules composes the parameter quality attribution strategy. The quality values assigned to the algorithm parameters are used to select the next parameter values, which are employed to create the next generation of solution(s). The selection mechanism configures the components of an Evolutionary Algorithm (variation, selection and replacement operators). The main challenge faced at this step is the trade-off that has to be made between the choice of current best configuration(s) and the search for new good settings.

The overall approach to adaptive parameter control consists of the four steps

shown by the highlighted boxes in Figure 3.1, namely the feedback collection strategy, the parameter effect assessment strategy, the parameter quality attribution strategy, and the parameter selection strategy. All four steps are executed automatically during the optimisation process. This thesis makes a contribution to each of the four steps of parameter control. In the next sections, we determine the gaps in each of the four steps of adaptive parameter control which leads to four research problems addressed in this thesis.

3.2.1 Feedback Collection Strategy

The feedback collection strategy is a distinguishing feature of adaptive parameter control, since other parameter control methods either do not consider any feedback from the search (deterministic parameter control) or the feedback is implicit (self-adaptive parameter control). The feedback is a measure of certain properties of an Evolutionary Algorithm (EA), which are an indication of the algorithm's performance. It has been shown by previous research [66, 43] that the performance of EAs is affected by algorithm parameters, which determine the search strategy. Hence, the change in algorithm properties, i.e. the change in the performance of an EA, can be used as an indication for understanding if the EA parameters are being successful.

The decision regarding the quality of the parameter values depends on the information provided by the feedback collection strategy, which makes it a very important step of adaptive parameter control. However, deciding on what performance properties of an optimisation algorithm to use as a feedback is a critical task. This is particularly challenging in multiobjective optimisation, since the output of multiobjective runs is usually a set of nondominated solutions, which make a trade-off between the fitness functions. The main challenge regarding the feedback collection

mechanism in multiobjective optimisation is assigning a single value to this set of solutions to represents the overall performance of the algorithm. Known parameter control methods use feedback collection mechanisms appropriate only for singleobjective optimisation [56, 181, 37], in which the relevant property is the change in fitness of a single best solution. These feedback collection mechanisms are not appropriate to be applied in parameter control for multiobjective optimisation. This problem leads to the following research question:

* **What is an effective feedback collection strategy to use in adaptive parameter control for multiobjective Evolutionary Algorithms?**

To answer this question, we first analyse performance aspects that should be considered in multiobjective optimisation. We employ these performance aspects to investigate metrics for measuring the performance of multiobjective optimisation algorithms, and perform an experimental evaluation of the feedback collection strategy using recommended performance metrics. The problem and recommendations regarding the multiobjective feedback collection strategy are described in Chapter 4.

3.2.2 Parameter Effect Assessment Strategy

Parameter values often have a different effect at different stages of the search [9, 179, 174, 77]. The information about the performance of the optimisation algorithm measured at every iteration can be used to determine if the parameter values have a positive effect in that iteration. This process is performed by the parameter effect assessment strategy, which derives the effect of parameter values based on the output from the feedback collection strategy. The aim is to determine successful parameter values, i.e. parameter values that have a positive effect on the performance of the optimisation algorithm.

The main difference of available parameter effect assessment methods is how they

define the 'success' of parameter values. The majority of the methods calculate the effect of parameter values using directly the improvement in the performance of the algorithm, measured as the quality difference of the generated solutions with respect to their parents [183, 191, 79, 85, 83], the overall best solution [37, 114, 65], or a specified quantile (e.g. the median) [94, 95]. Whitacre et al. [189] instead considers outlier solutions as an indication of the success of parameter values used to generate them. An outlier in this approach is a solution with a quality which is statistically different from that of other solutions.

In essence, parameter effect assessment strategies found in the literature assume that the improvement in the quality of the solutions, or the quality of the outliers, is directly related to the use of certain parameter values. For instance, if a mutation rate of 0.01 produces a solution which improves the quality of its parent by Δq, the quality improvement Δq is considered as the effect of this mutation rate. If more than one solution is created, the sum or average of the quality improvement of all solutions is assigned as the overall effect.

However, the performance of Evolutionary Algorithms is affected by more than one parameter value, hence using the performance of the algorithm directly as an indication of parameter success may be misleading. Moreover, Evolutionary Algorithms are stochastic systems, which may produce different results for the same parameter values [43]. This randomness should be taken care of by the parameter effect assessment strategy. To address this problem, the research question is formulated as follows:

- **What is an effective method for assessing the stochastic effect of EA parameters?**

We propose a probabilistic approach to measure the effect of parameter values, which is conditional on the usage of the parameter values and the successful perfor-

mance of the algorithm instances. The performance of an instance is successful if it is above a certain threshold. We define a success measure to represent the effect of parameter values, calculated as the number of successful instances that use a parameter value divided by the total number of instances that use that parameter value. This research problem, the proposed solution and the validation are discussed in Chapter 5.

3.2.3 Parameter Quality Attribution Strategy

The parameter effect assessment strategy described in Section 3.2.2 determines the effect of parameter values at every iteration, i.e. classifies each parameter value as a successful or unsuccessful choice for the current iteration. The effect of parameter values in the current iteration is used to make decisions about the parameter values to use in the next iteration. However, the effect of parameter values may not be the same for the next iteration, since different parameter values are optimal for different stages of the search [9, 179, 174, 77].

To make a better judgement on what would be a successful parameter value for the next iteration(s), current adaptive parameter control methods define a quality measure, which is calculated based on predefined rules that use the effect measured in the previous iterations. The quality of parameter values is calculated either as the average effect [83, 85] or as the best effect in the last W iterations [55]. In essence, notable parameter quality attribution mechanisms calculate the quality of parameter values from past performance (time $t-1$). One might argue that the quality of parameter values calculated by these methods represents their effect in the previous iterations. Ideally, one would use a prediction of the effect for time t based on previous performance. Acknowledging this fact, we formulate the following research question:

* **What is an effective parameter quality attribution method for projecting successful parameter values to the next iteration of the optimisation process?**

To answer this question we use a Predictive Quality Assessment (PQA) strategy that combines a measure of past performance with time series prediction to calculate the quality of parameter values. We investigate various forecasting techniques to use for time series prediction and provide recommendations on the adequacy of these methods in forecasting the quality of different parameter types. Furthermore, PQA is compared to two notable parameter quality attribution strategies - Average Quality Attribution (AQA) and Extreme Quality Attribution (EQA) - to understand if using Predictive Quality Attribution improves the performance of the algorithm. A detailed discussion of Predictive Quality Attribution and the validation of the approach are given in Chapter 6.

3.2.4 Parameter Value Selection Strategy

The final stage of adaptive parameter control is the selection of parameter values to use in the next iteration. The selection is based on the trade-off between using parameter values with high quality and exploring new parameter values. When controlling parameters with discrete values such as the type of mutation operator, the possible choices are not too many (e.g. single-point, uniform or transposition mutation operator), hence their exploration is attainable. In the case of real-valued parameter assignments, such as the mutation rate and the crossover rate, the possible values are too many to explore during the optimisation run.

Generally, parameter value selection strategies [182, 181, 56] optimise parameter values by choosing from predefined values or ranges. The quality feedback and therefore the probability of use in the next iteration is allocated to these ranges, not

the actually sampled values. Normally, parameter ranges are fixed and not optimised by the optimisation process. As a result, the number and the size of the ranges has to be decided by the user, who has to compromise either on the accuracy of the intervals or the low number of computations, i.e. low search complexity. More specifically, if narrow parameter ranges are selected, the accuracy in attributing to that interval the quality achieved by values sampled from that interval is higher. However, there are more intervals to sample from, which may lead to a non-exploration of some of the ranges or a slow identification of the ones performing well. If wider ranges are selected, the sampling inaccuracy increases since the actually sampled value may be far from the value whose success the range's usage probability is attributable to. Hence, the initial discretisation may lead to values which are suboptimal for some problems or their instances [9]. It follows that an ideal parameter value selection strategy would adapt parameter ranges during optimisation process. The research question is defined as follows:

* **What is an effective method for configuring real-valued parameters during the optimisation process?**

To address this problem we introduce the Adaptive Range Parameter Selection strategy (ARPS). ARPS dynamically changes the ranges of parameter values during the optimisation process based on their quality. Successful ranges are divided into smaller ranges to increase their selection probability and the accuracy of the parameter quality attribution strategy, whereas unsuccessful parameter value ranges are merged to decrease the selection probably. The Adaptive Range Parameter Selection strategy (ARPS) and the validation of the approach are presented in Chapter 7.

3.3 Research Method

This thesis introduces a new adaptive parameter control method which automatically adjusts parameter values of Evolutionary Algorithms during the optimisation process. The main objective is related to the ultimate goal of optimisation, i.e. answering the question of which method gives fast and high-quality solutions. More specifically, the objective is to build a parameter control method that improves choices of parameter values, which result in better algorithm performance.

To evaluate the proposed contributions of the thesis we design a set of experiments. The design of experiment describes the steps performed in order to run a set of test runs under controlled conditions to examine the performance of the EA when using Adaptive Parameter Control (APC). An experiment consists of solving a series of problem instances, some of which are well-studied benchmark problems, whereas others are obtained by using a problem generator. All problems are solved by an EA whose parameters are controlled by different adaptive parameter control approaches. Selected performance measures are used to report and compare the results from all methods. In general, performing an experimental comparison between algorithms, in particular, between Evolutionary Algorithms that use different parameter control methods implies a number of methodological questions:

- What are the experimental settings?
- Which benchmark problems and problem instances are used?
- Which benchmark parameter control methods are used?
- Which comparative measures are employed?

The design of experiments is based on the research objectives, which help in identifying the questions to ask, the hypothesis to test, the tests to run, the factors to explore, and the measures to use.

3.3.1 Experimental Settings

The validation of the approach is based on the statistical design of the experiments. Optimisation trials are treated as experiments with many degrees of freedom. In the first step of the design of an experiment we determine the design factors, which are the elements that can be changed during the experiment. These design factors or variables can be either problem-specific or algorithm-specific.

Problem-specific factors are the objective function(s), the problem instances, the constraints, etc. The problem-specific design factors are not changed during the experiments. The algorithm-specific factors are the parameters that are controlled during the optimisation process and the values of the parameters that are tuned before the run of the algorithm. The design of parameter control methods consists of the configuration of the hyperparameters, which have to be determined before the algorithm is executed. Both problem-specific and algorithm-specific design factors are determined before each experiment.

Evolutionary Algorithms are not expected to deliver exact and repeatable results, but to provide good approximate solutions where exact approaches cannot be devised. Hence, results concerning the performance of approximate algorithms such as EAs, are usually reported as mean values over repeated trials. To obtain a fair comparison, the generally accepted approach is to allow the same number of function evaluations for each trial [152]. Therefore, for the current comparison, all algorithms trials were repeated 30 times for each optimisation scheme. These values were decided after running the algorithm once for every problem and choosing the value where the quality of the solutions seemed to not improve any more. Nevertheless there are indications that all algorithms still make small but steady improvements after these numbers of evaluations.

3.3.2 Benchmark Problems

According to Lin et al. [153], differences in performance among approximate algorithms are more likely to be detected statistically if all algorithmic approaches solve the same problem instances. Along this line, four problems were chosen: the generally accepted Royal Road Problem (RRP), the Quadratic Assessment Problem (QAP), and the multiobjective Quadratic Assignment Problem (mQAP) which were especially designed for testing EAs. The problems were chosen due to their dissimilarity, which enables a more informed judgement as to the portability of the approach when applied to an EA. Moreover, all four problems are NP-hard, which makes them difficult to be solved by an exact algorithm and justifies the use of EAs to optimise them.

The problems we have selected can be categorized into two main groups: repository (benchmark) problems and randomly generated problems. The repository problems are the Quadratic Assignment Problem (QAP), the multiobjective Quadratic Assignment Problem (mQAP) and the Royal Road problem. For the Quadratic Assignment Problem (QAP) we have used problem instances published at `http://www.seas.upenn.edu/qaplib/inst.html`.

The multiobjective Quadratic Assignment Problem (mQAP) [99], which is described in Section 3.3.2, is a well-known problem and instances of considerable difficulty have been made available as benchmarks at `http://dbkgroup.org/knowles/mQAP/`. Four different problem instances are used in the experiments: one uniform triobjective instance of dimension n=30 (KC30-3fl-3uni), two triobjective problems of dimension n=30 (KC30-3fl-2rl and KC30-3fl-3rl) and a biobjective problem of dimension n=20 (KC30-2fl-5rl). The four problems are NP-hard, and their Pareto fronts are not known.

MQAP and QAP as assignment problems map N tasks to M resources. Hence

the solution representation for both problems is a simple array which describes the numbered locations and the values of the array represent the items. The problems are optimised using an Evolutionary Algorithm with string-based representation and customised crossover and mutation operators with their respective probabilities of being applied. Multipoint crossover swaps the assignments between solutions. The mutation operator changes the assigned item of a single location. As we are solving the MQAP and QAP, singularity of item occurrence is mandatory and the solutions are validated after the operators have been applied.

The Royal Road problem is optimised using an EA with string-based representation and multipoint crossover. The component deployment problem is multiobjective in nature and requires a more specialised approach. One notable multiobjective EA implementations is Deb's [41]. Its distinctive feature is nondominated sorting, which filters the population into layers of nondominated fronts and ranks the solutions according to the level of front they are a member of. These EA implementations use customised crossover and mutation operators with their respective probabilities of being applied.

The advantage in using published problem instances relies on the fact that they have already been investigated by other researchers and their characteristics are well-known. However, we also use a problem generator, which can be downloaded from `http://mercury.it.swin.edu.au/g_archeopterix/`. The problem generator produces problems by taking as input problem specific parameters. This is an advantage, since the problem specific parameters can be used to tune the difficulty of the problem, which help in offering insight into how specific components of parameter control methods deal with different problem difficulties and provide knowledge on the reasons for the performance of the different parameter control methods.

The Royal Road Problem

The Royal Road functions are parametrized landscapes, first introduced by Mitchell, Forrest, and Holland [93]. They are especially devised to demonstrate that there exist problems which are easier to solve using a Genetic Algorithm than a hill climber.

The function of the form $F : \{0,1\}^l \rightarrow \mathbb{R}$ is used to define a search task in which one wants to locate strings that produce high fitness values. The string is composed of 2^k non-overlapping contiguous sections each of length $b + g$, where b is known as the block and g is known as the gap. These sections form the first-level schemata, composed of short, low-order, highly fit schemas, which are called the building blocks. A schema s_i is a combination of ones, zeros and the special character '*' which is interpreted as 'do not care'. Lower-level schemas are used to form intermediate-level higher-fitness schemas, which in turn are combined to create even higher-fitness schemas.

A set of schemas which define the Royal Road Fitness function RRF are depicted in Figure 3.2. The set of schemas is denoted as $S = \{s_1, s_2, ...s_9\}$. Each s_i is assigned a specific coefficient c_i which is used in the fitness calculation.

$s_1 = [11111111**] \; c_1 = 8$
$s_2 = [********11111111**] \; c_1 = 8$
$s_3 = [****************11111111**] \; c_1 = 8$
$s_4 = [************************11111111**********************************] \; c_1 = 8$
$s_5 = [********************************11111111**************************] \; c_1 = 8$
$s_6 = [**11111111******************] \; c_1 = 8$
$s_7 = [**11111111********] \; c_1 = 8$
$s_8 = [**11111111] \; c_1 = 8$
$s_{optimal} = [11111111111111111111111111111111111111111111111111111111111111111]$

Figure 3.2: Example of schemas defining the RRP function.

The schemas in Figure 3.2 are created by breaking up the optimal string $s_{optimal}$ into eight building blocks. Strings are represented by using the same alphabet as schemas (0,1,*) where 0 and 1 are called defined bits, and * means 'don't care'. For

example, the pattern [0∗] requires that the first bit be set to 0 and the second bit can be either 0 or 1. A bit string x is an instance of a schema s if it follows the same pattern of that schema. For example, both [010] and [000] are instances of [$0 * 0$].

The fitness calculation of a string x proceeds in two steps, the part calculation and the bonus calculation. The overall fitness assigned to that string is the sum of these two calculations.

The *part calculation* considers each block individually and assigns a fitness according to the number of 1's. The block fitnesses are later summed to form the total fitness of the part calculation. The fitness function $RRP(x)$ for the part calculation of a string x is defined as:

$$RRP(x) = \sum_i c_i \delta_i(x) \tag{3.1}$$

where

$$\delta_i(x) = \begin{cases} 1 & \text{if} \quad x \in s_i \\ 0 & \text{otherwise} \end{cases}$$

For example, if $x = 11111111***11111111$, i.e. it corresponds to exactly two schemas, $RRP(x) = 16$.

In Holland's Royal Road function the fitness of each block is based entirely on the number of 1 bits it contains. The aim is to reward bits equal to 1 up to m^* consecutive bits. For each bit equal to 1 the block's fitness is rewarded by v. For example, a block with three $1's$ has a fitness of $3 * v$. If a block contains more than m^* 1's, but less than b 1's, it receives $-v$ for each 1 over the limit. Finally, if a block has only 1's and no 0's, it is considered to be complete. This block receives nothing from the part calculation and will be rewarded in the bonus calculation.

Table 3.1: Hollands default Royal Road problem setting

Variable	Value
k	4
b	8
g	7
m^*	4
v	0.02
u^*	1.0
u	0.3

The aim of the *bonus calculation* is to reward complete blocks and some combinations of complete blocks. Holland gives rewards for attaining a certain level. At the lowest level, rewards are given for complete blocks. If such a block exists, it receives a fitness equal to u^*. Any additional complete blocks receive a fitness of u.

In general, a Royal Road function is defined by the variables used in the part and bonus calculation (e.g. v, b, g, etc). The values used by Holland, which are also used in our experiments, are depicted in Figure 3.1.

The Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) was first introduced by Koopmans and Beckmann [101], who used this mathematical model to solve the problem of assigning economical activities to resources. In QAP, n facilities have to be allocated to n locations, such that the total cost is minimised and every resource has only one economical activity. The total cost is calculated as the flow between the facilities multiplied by the costs for placing the facilities at their respective locations. QAP is considered to be a very challenging combinatorial optimisation problem. More formally, the problem is modelled with two $n \times n$, representing the cost and the flow. The aim is to assign n utilities to n locations with minimal cost. The candidate

assignments are evaluated according to equation 3.2.

$$C = \sum_{ij}^{n} B_{ij} \cdot u_{ij} + \sum_{ij,k,l} C_{ij,k,l} \cdot u_{ik} \cdot u_{jl} \tag{3.2}$$

where

- n is the number of facilities and locations.
- B_{ik} is the cost of assigning utility i to location k
- $C_{ij,k,l}$ is the cost of the flow between neighbouring utilities (given utility i is assigned to location k and utility j is assigned to location l
- u_{ik} is 1 if utility i is assigned to location k, 0 otherwise

QAP does not allow for multiple assignments to the same location, hence solutions are subject to the following constraints: The assignments are subject to constraints regarding the

$$\sum_{j=1}^{n} u_{ij} = 1, \quad i = \{1, 2, ..., n\} \tag{3.3}$$

$$\sum_{i=1}^{n} u_{ij} = 1, \quad j = \{1, 2, ..., n\} \tag{3.4}$$

$$u_{ij} \in \{0, 1\}, \quad i, j = \{1, 2, ..., n\} \tag{3.5}$$

The Multiobjective Quadratic Assignment Problem

MQAP is the multiobjective version of the Quadratic Assignment Problem (QAP). MQAP is a difficult problem, where even relatively small problem instances (of dimension n=20) cannot be solved to optimality. The aim in mQAP is to assign n utilities to n locations with minimal cost, which is similar to singleobjective QAP. However, in mQAP, we are concerned with the flow of more than one type of item.

For instance, in a hospital layout problem, different aspects have to be considered, such as minimising the travel distance of the doctors to patient, of the patients to their rooms, of hospital visitors to the patients, and of pharmaceuticals or other equipments from storage rooms to operation or patient rooms. Similar to QAP, the overall cost of assigning all facilities to locations is evaluated according to Equation 3.6.

$$C_s = \sum_{ij}^{n} B_{ij} \cdot u_{ij} + \sum_{ij,k,l} C_{ij,k,l} \cdot u_{ik} \cdot u_{jl} \tag{3.6}$$

The mQAP is formulated as:

$$minimise: \quad \vec{C} = [C_1, C_2, ..., C_m] \tag{3.7}$$

where m is the number of objectives to be minimised.

This problem and its fitness landscapes have been carefully investigated by Knowles and Corne [99] and a set of benchmark problems are published at `http://dbkgroup.org/knowles/mQAP/`.

3.3.3 Benchmark Methods

The main goal of the thesis is to build a parameter control method that improves choices of parameter values, which result in better algorithm performance. To determine if the goal is achieved, we compare the approaches introduced in this thesis with state-of-the-art methods. For each of the four contributions, we have selected distinguished benchmark methods against which we compare the achieved results. Benchmark methods are described in the validation section of each contribution chapter.

3.3.4 Comparative Measures

In order to compare two or more Evolutionary Algorithms, suitable performance measures have to be identified. If the optimal solutions of the problems considered are known, a Success Measure (SM) can be defined as finding the optimal solutions(s). SM can be used to define a performance measure as the percentage of runs terminating with success [46].

The class of problems we consider are mostly NP-hard, and often the optimal solutions are not known. In this case, SM is not appropriate. The performance of algorithms when solving problems where the optimal solution(s) are not known can be assessed by using the computation time (CPU time) required to achieve results of certain quality. This measure is not very accurate, since it depends on different factors, such as the programmer's abilities, the hardware used, the operating system, compiler, etc. If the algorithms are implemented by different programmers, the comparison using the computation time is ill-suited since more experienced programmers are capable of implementing programs which execute faster. Using the computation time may give different results if the same experiment is executed in different computers, due to the hardware, operating system, compiler, etc.

Other more effective ways to evaluate Evolutionary Algorithms are the quality of the final solutions in a predefined number of function evaluations, called the *Mean of Solutions Quality* (MSQ) or the number of function evaluations used to achieve a certain quality, called the *Average number of Evaluations to a Solution* (AES) [46]. The AES can give a fair comparison of the algorithms if the each function evaluation takes the same amount of computation time, and that they consume most of the time of the algorithm run. The *Average number of Evaluations to a Solution* can be misleading if this is not the case.

We use the MSQ and allow for all our experiment a fixed number of function eval-

uations. This measure considers the stochastic nature of Evolutionary Algorithms, which leads to different results for different runs, by expressing the performance measure as the mean of the final quality of the solutions over a number of independent runs. Altogether 30 independent runs are performed per EA and test problem in order to restrict the influence of random effects. A different initial population is randomly created each time, and for each test problem all EAs operate on the same 30 initial populations. The MFQ can always be used to measure the performance of stochastic optimisers, since it does not require the optimal solutions. In order to check for a statistical difference in the outperformance of the optimisation schemes, results are validated using the Kolmogorov-Smirnov (KS) non-parametric test [149].

Part II

Contribution

Overview

This thesis makes four main contributions related to each of the four steps of adaptive parameter control: the feedback strategy, the effect assessment strategy, the quality assessment strategy and the parameter selection strategy.

First, we investigate different performance metrics for reporting the feedback in multiobjective optimisation problems and provide recommendations on successful strategies.

The second contribution is related to the estimation of the stochastic effect of parameter values on the performance of the optimisation algorithm. We propose a Bayesian model which approximates the relationship between the parameter values and the algorithm's performance probabilistically. This probabilistic relationship represents the effect of parameter values on the successful performance of the algorithm.

The third contribution is in the estimation of the overall quality of the parameter values. The effect of parameter values measured in the past iterations is used to predict their quality for the next iteration. The quality of parameter values is employed to select the parameter values to use in the next iteration choices, in which parameter values with a high quality are more likely to be selected.

The final contribution is related to the selection of parameter values when adapting real-valued parameters, for which the discretisation of the value intervals into ranges is required. We propose a new parameter selection strategy which adapts ranges of real-valued parameters during the optimisation process.

In essence, in this thesis we propose four strategies for performing each of the steps in adaptive parameter control: a Multiobjective Feedback Collection (MFC) strategy, presented in Chapter 4, a Bayesian Effect Assessment (BEA) strategy, described in Chapter 5, a Predictive Quality Attribution strategy (PQA), given in Chapter 6 and an Adaptive Range Parameter Selection strategy (ARPS), described in Chapter 7.

Chapter 4

Feedback Collection Strategy

4.1 Introduction

Adaptive parameter control derives the probabilities of parameter values to choose for the next iteration from the algorithm's performance, which is part of the feedback collection strategy. The main aspect of the performance of an optimisation algorithm is the current fitness of the solution(s), where the aim is to minimise/maximise its value. This is usually the way performance is measured in a singleobjective algorithm, in which the solution with the best fitness value (smallest value in minimisation problems and largest value in maximisation problems) is reported for each run of the algorithm. Hence, noting the achieved fitness as a measure of the performance of a singleobjective algorithm is agreeable. In a multiobjective problem, and in the absence of a preference for certain objectives, the optimisation process produces a set of nondominated solutions, which make a trade-off between the fitness functions. Measuring the performance of a multiobjective algorithm is not straightforward, since the output of the algorithm is more than one value. The success of measuring the performance of a multiobjective optimisation algorithm relies on the ability to express the quality of multiple data points with a single value to use as a feedback for the adaptive parameter control method.

State-of-the-art adaptive parameter control methods typically consider singleob-

jective optimisation problems and use the fitness of the solutions ($f(s)$) as an indication of the performance of the algorithm [47, 56, 181, 37, 38, 94, 84, 7]. In typical approaches the fitness is employed directly, i.e. a high fitness indicates a good performance of the algorithm. These feedback collection collection mechanisms are not suitable to use in adaptive parameter control for multiobjective optimisation algorithms, which usually produce more than one solution as output. Recommendations regarding appropriate feedback collection mechanisms to use in adaptive parameter control for multiobjective optimisation are not available. Acknowledging these facts, in this chapter, we investigate the following research question:

* **What is an effective feedback collection strategy to use in adaptive parameter control for multiobjective optimisation?**

With the aim of selecting an adequate performance metric for the feedback collection strategy to use in adaptive parameter control for multiobjective Evolutionary Algorithms, we discuss different performance aspects in multiobjective optimisation. These performance aspects are employed to examine state-of-the-art performance metrics for multiobjective optimisation algorithms and to provide recommendations for successful strategies to use in multiobjective feedback estimation. This step is a precondition for the following main contribution chapters, which employ the information from the feedback collection strategy to decide on the successful applications of parameter values and adjust them accordingly.

4.2 Performance Aspects in Multiobjective Optimisation

In singleobjective optimisation, the output of an optimisation run is a single solution, which has the best objective value (i.e. fitness of the solution). Hence, the performance of the algorithm can be directly assessed by using the achieved fitness of the best solution, which pertains to the quality of that solution. In multiobjective optimisation, the output of the optimisation process is the set of nondominated solutions that guarantee a trade-off among the objective values. As a result, multiple data points should be used to calculate the performance of a multiobjective optimisation algorithm. In this section, we discuss the quality aspects of nondominated solutions that should be considered when measuring the performance of the algorithm.

Zitzler et al. [195] suggest that a performance metric for multiobjective optimisation algorithms should consider three aspects: (i) the distance of the nondominated solutions to the Pareto front, (ii) the distribution of the nondominated solutions, and (iii) the extent of the obtained nondominated front.

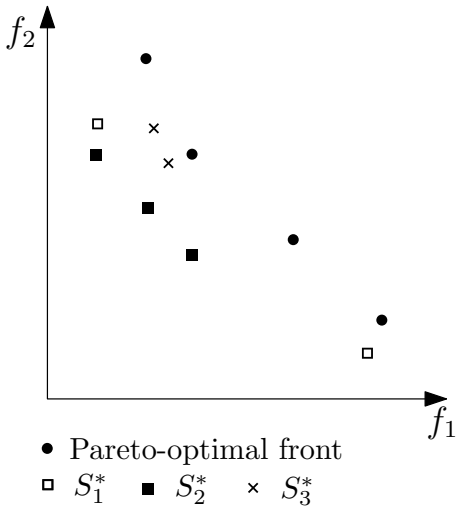


Figure 4.1: Properties of nondominated sets.

The first criterion pertains to the *fitness* of a nondominated set. The closer the nondominated solutions to the Pareto front, the better they are. The other two properties measure how the solutions are distributed, which describes the *diversity* of the solutions. An even distribution of the solutions and a large coverage of the Pareto front is desirable.

Consider the outcome of three hypothetical optimisation algorithms in a maximisation problem and the Pareto-optimal front depicted in Figure 4.1. The three nondominated sets differ from each other with respect to three performance aspects. The nondominated set S_2^* has a more uniform distribution of the solutions than the other nondominated sets, the nondominated set S_3^* has a smaller distance to the Pareto-optimal front, i.e. has a better fitness, whereas the nondominated set S_1^* has a larger extent, i.e. has a better coverage property.

In essence, the key question when selecting a performance metric for multiobjective optimisation is how to summarize the nondominated solutions with a single value. The underlying idea is to quantify the quality of nondominated solutions, which describes the performance of the optimisation algorithm. We investigate state-of-the-art performance metric for multiobjective optimisation with respect to the performance properties described in this section. The goal is to select an appropriate performance measure that quantifies the change in the properties of nondominated fronts and accurately guides the parameter control strategy.

4.3 Multiobjective Performance Metrics

Performance metrics for multiobjective optimisation have been widely studied in the optimisation community. A review of state-of-the-art performance metrics for multiobjective optimisation is depicted in Table 4.1. The majority of performance metrics assign a nondominated set a value which represents a certain property, such as diversity or fitness. They are usually referred to as unary measures, since they consider only one set of solutions. Binary metrics, such as binary hypervolume indicator [197, 198] and the binary epsilon indicator [198] can be used to assign property measures to pairs of nondominated sets.

Unary measures assess the performance of multiobjective optimisation algorithms by using the fitness of the nondominated solutions only, the diversity of the results exclusively or both fitness and diversity. In terms of the diversity property, unary metrics calculate the total number of solutions in a nondominated set (*front occupation indicator* [185]), the spread of the solutions (*front spread indicator* [194]) or the relative distance between solutions (*spacing indicator* [165]). However, the diversity property alone is not very useful if the fitness of the solutions is unkown. The solutions in the nondominated set may have a high diversity, however the ultimate goal of optimisation is to find solutions that have a good fitness as well.

Unary fitness metrics found in the literature measure the distance of the nondominated solutions to the Pareto-optimal front, such as the *proximity indicator* [185] and the *generation distance* [8]. The optimal solutions often are not known, especially when solving previously unknown NP-hard problems. This is particularly true when using parameter control techniques to adapt algorithm parameters during the optimisation of problems which most probably have not been optimised beforehand, and whose Pareto-optimal front is not known. Hence, performance measures which calculate the quality of nondominated sets with respect to the Pareto-optimal front

Table 4.1: Performance metrics for multiobjective optimisation algorithms.

Metric	**Property**	**Description**	**Comment**
Cluster [109]	Diversity	Numbers of clusters in solution space represent diversity.	Unary measure. Very sensitive to the parameters.
Binary Hypervolume [197]	Diversity, fitness	Measures the size of the dominated area in the objective space.	Unary/binary measure.
Spacing [165]	Diversity	Measures the range variance of neighbouring solutions in the nondominated set.	Unary measure. Fails if all the solutions are located in the same point.
Chi-square-like deviation [178]	Diversity	Divides the whole nondominated region into sub-regions.	Unary measure. Fails in multiobjective optimisation and is sensitive to the division.
Δ [41]	Diversity	Measures Euclidean distances between solutions and two additional extreme solutions.	Unary measure. Requires some prior knowledge of the Pareto front.
Front Spread [194]	Diversity	Indicates the size of the objective space covered by an approximation set.	Unary measure. Does not consider the number of points.
Front Occupation [185]	Fitness	Indicates the number of points available in the nondominated set.	Unary measure. Does not measure how far these points are from each other.
Generation Distance [8]	Fitness	Euclidean distance of the nondominated solutions to the Pareto front.	Binary measure. Requires prior knowledge of the Pareto Front.
Proximity [185]	Fitness, diversity	Measures the coverage of the Pareto front.	Binary measure. Requires some prior knowledge of the Pareto Front.
Average best weight combination [52]	Fitness	Measures the desirable properties of a set using user preferences.	Unary measure. Nonconvex points of a set are not considered.
Binary Epsilon	Fitness.	Measures the minimum factor between two nondominated solutions	Binary measure. Works only for comparing nondominated solutions.

are a poor choice, in particular when used as a feedback collection mechanism in adaptive parameter control methods.

Other unary metrics employ user preferences to evaluate the quality of nondominated points. One example is the *average best weight combination* [52]. The user specifies a goal and feasibility vector $(g, c) \in G$, where G is the set of all preference vectors. Each goal value g_i specifies a user preference for the value of fitness function f_i and c_i specifies the limit beyond which solutions are of no interest. In a minimisation problem, the quality of a nondominated point s_i^* is calculated using the goal vector as follows:

$$q(s_i^*) = \begin{cases} \sum_j^k w_{j=1} f_j(s_i^*) & \text{if} \quad f_j(s_i^*) \leq g_j \\ \sum_{j=1}^k w_j g_j \max(c_j, g_j) & g_j \leq f_j(s_i^*) \leq c_j \\ \sum_{j=1}^k w_j^2 g_j \max(c_j, g_j) & \text{otherwise} \end{cases}$$

where w_j represents the weight for objective function f_j. This metric does not require the Pareto-optimal front to calculate the fitness of the solutions. However, the *average best weight combination* requires that the user defines the goal and feasibility vector, which is different for every problem and objective functions being optimised. This limits the applicability of the metric to problems in which a goal and feasibility vector does not exist or cannot be generated. Furthermore, in order to use this method as a feedback collection strategy in parameter control the goal and feasibility vector has to be defined for every iteration, which is impractical.

Despite the variety of performance metrics in the literature, the majority of these methods do not indicate how well a multiobjective optimisation algorithm performs with respect to both diversity and fitness properties. The proximity indicator [185] is one of the few examples of performance metrics that measure both fitness and diversity of nondominated solutions. However, this metric assumes that the Pareto-

optimal front is known or can be generated.

Zitzler et al. [196] investigated different unary and binary performance metrics for multiobjective optimisation. The goal of the study was to understand if it is possible to conclude from a performance metric that a nondominated set is undoubtedly better than another nondominated set, and to what extent that metric helps in making precise statements about how much better it is. The study concludes that there exist no unary performance metric capable of achieving this and their use is discouraged. In essence, the authors argue that binary performance metrics are superior to unary ones.

One of the recommended metrics is the *coverage measure* [197], which is a binary performance metric usually used for comparison and interpretation of results from different runs or algorithms. The coverage measure calculates the number of solutions in a results set that are dominated by the other results set, and reports it as a percentage. In the maximisation problem shown in Figure 4.2, the coverage measures for nondominated sets S_1^* and S_2^* are $C(S_1^*, S_2^*) = 0.75$ and $C(S_2^*, S_1^*) = 0.25$ respectively.

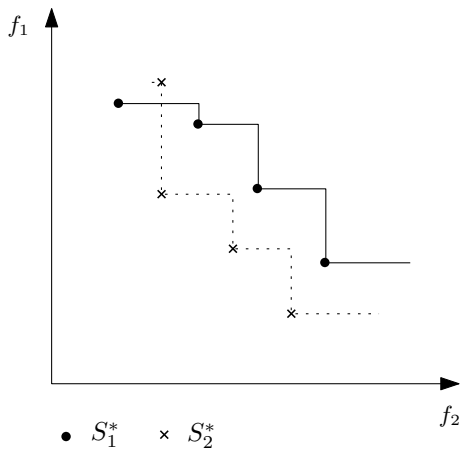


Figure 4.2: Coverage measure.

Despite being an accurate metric in measuring if a nondominated set is better

then another one [196] *coverage measure* does not calculate how much better are the solutions.

Two other metrics recommended by this study are the binary hypervolume indicator [197, 198] and the binary epsilon indicator [198]. Both metrics are capable of detecting whether a nondominated set is better than another nondominated set. The binary hypervolume indicator and the binary epsilon indicator provide additional information about how much better a nondominated set is compared to the other. Instead of using the Pareto-optimal front as a reference set to measure the quality of nondominated solutions, both metrics assign quality measures to pairs of nondominated sets, hence the term binary. In the case of binary epsilon indicator, quality refers to the fitness of the solutions, whereas for the binary hypervolume indicator quality is measured as the diversity and the fitness of the solutions. The binary hypervolume indicator considers the dominated area (volume or hyperspace) in the objective space as a measure of the quality of the nondominated set. The epsilon indicator calculates the smallest value that is needed to add to the objective values of all solutions in one nondominated set, such that it dominates the other nondominated set. These two metrics are described in more details in the next sections.

4.3.1 Binary Hypervolume Indicator

The binary hypervolume indicator ($\hat{h}$) [197, 198] uses the dominated part of the objective hyperspace to measure the difference between two nondominated solutions with respect to diversity and fitness. It was first devised by Zitzler and Thiele [197], defined as the hypervolume in the objective space that is dominated by one of the nondominated sets but not by the other, e.g. Figure 4.3.

Formally, the hypervolume indicator can be defined by using volumes of poly-

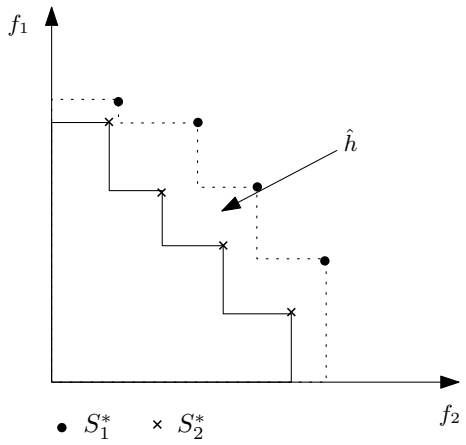


Figure 4.3: The binary hypervolume indicator is the area (volume or hyperspace) between two nondominated sets. It is measured as the difference of the hypervolume values of the two sets.

topes [197]. Let $F = \{f_1, f_2, ..., f_o\}$ be the set of objective functions, and $S^* = \{s_1^*, s_2^*, ...s_n^*\}$ be the set of nondominated solutions. For each axis in the objective space F and for each nondominated solution s_i^* there exists a hyperplane perpendicular to the axis and passing through the point $(f_1(x_i), f_2(x_i), ..., f_o(x_i))$. A polytope p_i is formed by the intersection of the hyperplanes that arise out of s_i^* along with the axes. The hypervolume indicator $\hat{h}(S^*)$ calculates the area, volume or hyperspace enclosed by the union of the polytopes $p_1, p_2, ..., p_n$. For instance, in the two-dimensional case, i.e. when optimising two objectives functions f_1 and f_2, each polytope p_i represents a rectangle defined by the origin (0.0) and the data point $(f_1(s_i^*), f_2(s_i^*))$.

The binary hypervolume indicator has shown to be very sensitive to small improvements of the nondominated solutions [198]. For instance, if a nondominated set dominates another nondominated set, the binary hypervolume indicator will reflect this difference of the quality of the two nondominated solutions [198].

4.3.2 Binary Epsilon Indicator

The binary epsilon indicator (ϵ) [198] calculates the difference by which one nondominated set of solutions is worse than another nondominated set with respect to all objective functions. More specifically, the value of ϵ is equal to the smallest value that is needed to add to each objective value of the solutions in one nondominated set, such that it dominates all solutions in the other nondominated set. An example of the ϵ-indicator in a maximisation problem is shown in Figure 4.4.

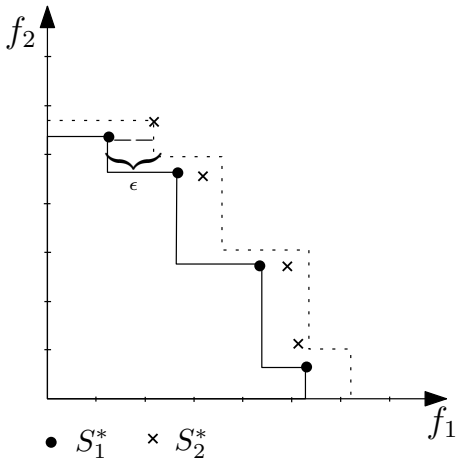


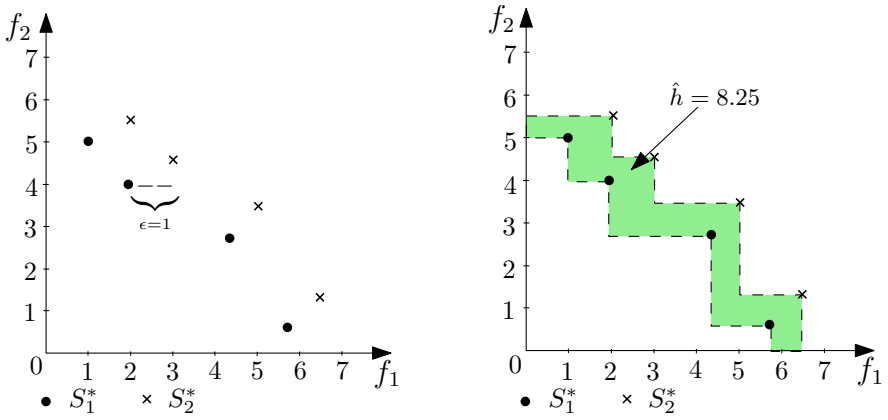
Figure 4.4: The binary epsilon indicator is the smallest value that is needed to add to the objective values of one nondominated set, such that it dominates all points in the other nondominated set.

Formally, in a maximisation problem, given two sets of nondominated solutions S_1^* and S_2^*, the objective vector $F(S_1^*)$ is said to ϵ-dominate the objective vector $F(S_2^*)$, denoted as $S_1^* \succeq_\epsilon S_2^*$, if and only if $\forall f \in F, s_1^* \in S_1^*, s_2^* \in S_2^* : \epsilon + f(s_1^*) \geq f(s_2^*)$.

4.3.3 Example

To illustrate the difference between these two metrics consider the results from a maximisation problem shown in Figure 4.5, in which both the binary hypervolume and the binary epsilon indicators are employed to measure the change in the quality

of the solutions after an optimisation run. Figure 4.5a and 4.5b depict the same nondominated solutions, where S_1^* is the initial nondominated set and S_2^* is the output of the optimisation process, i.e. the optimised nondominated solutions.

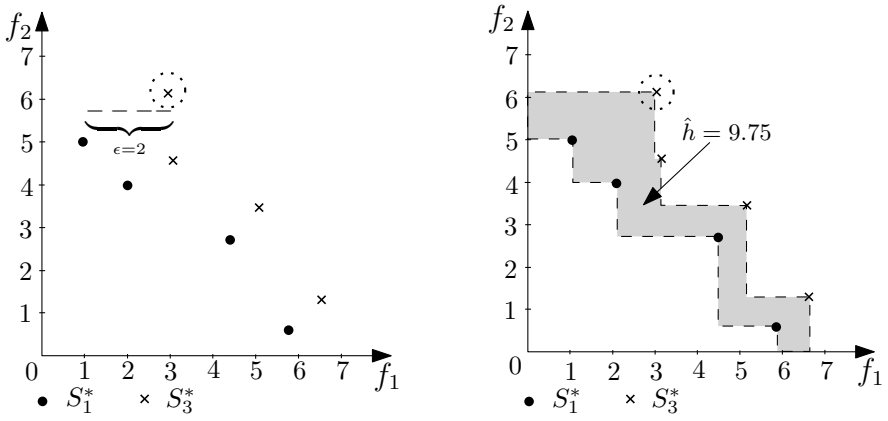


(a) Binary epsilon. (b) Binary hypervolume.

Figure 4.5: Example of binary hypervolume and binary epsilon indicators.

The quality improvement is measured using the binary epsilon indicator, graphically shown in Figure 4.5a and the binary hypervolume indicator depicted in Figure 4.5b. The binary hypervolume indicator, represented by the shaded area in Figure 4.5b, is equal to 8.25, and the binary epsilon indicator, represented by the biggest difference among the objective values of the two nondominated sets in Figure 4.5a, is equal to 1.

Figure 4.6 shows a different scenario. The nondominated set S_1^* is similar to the one showed in Figure 4.5, whereas the nondominated set S_3^* is similar to S_2^*, but has one solution (the solution highlighted with a circle) which is better in quality, i.e. both fitness values are bigger. The binary hypervolume indicator in this case is equal to 9.75 and the binary epsilon indicator is 2. Comparing the two examples in Figure 4.5 and 4.6, it can be observed that by improving the fitness of a single solutions the binary epsilon indicator value doubled, i.e. the binary hypervolume value of the nondominated set in Figure 4.6b is 100% more than the binary hyper-

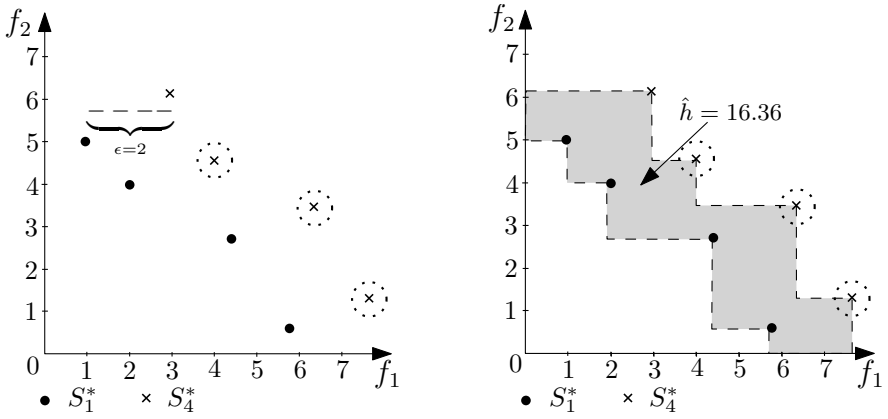


(a) Binary epsilon. (b) Binary hypervolume.

Figure 4.6: Example of binary hypervolume and binary epsilon indicators.

volume value of the nondominated set in Figure 4.5. The value of the binary epsilon indicator, on the other hand, increased by only 18%.

Lets consider a third scenario depicted in Figure 4.7, in which the circled solutions in the nondominated set S_4^* are better in quality compared to the solutions of the nondominated set S_3^*.



(a) Binary epsilon. (b) Binary hypervolume.

Figure 4.7: Example of binary hypervolume and binary epsilon indicators.

In this example, the binary hypervolume indicator is equal to 16.36, i.e. approximately 68% better than the original value in the example shown in Figure 4.6. The binary epsilon indicator instead is the same as the original value ($\epsilon = 2$) despite the

fact that three solutions in S_4^* have better quality.

This example shows that the binary hypervolume indicator is more sensitive to the diversity of the solutions compared to the binary epsilon indicator. On the other hand, the binary epsilon indicator is more sensitive to improvements in the quality of the solutions. In summary, the binary hypervolume indicator measures both fitness and diversity of nondominated sets, whereas the binary epsilon indicator is affected more by the fitness property.

4.4 Validation

Both binary hypervolume indicator and binary epsilon indicator are possible choices for the multiobjective feedback collection strategy. In this section, we investigate the use of these two metrics as a feedback collection mechanism for parameter control in multiobjective optimisation. The main objective is to select an appropriate feedback collection strategy that improves the search for high-quality solutions. We employ the binary hypervolume indicator and the binary epsilon indicator as feedback collection mechanisms for parameter control, and compare the outcomes of the optimisation runs using these feedback collection strategies. In essence, in this section, we aim at finding an answer to the following research question:

* **What is an effective feedback collection strategy to use in adaptive parameter control for multiobjective Evolutionary Algorithms?**

To compare the results we use dominance ranking, proposed by Knowles et al. [100]. Dominance ranking is based on ranking the nondominated sets by means of the dominance relation described in Section 2.2. In essence, each nondominated set generated by one optimisation scheme is assigned a rank representing the number of nondominated sets generated from the other optimisation scheme that are not dominated by it. The lower the rank, the better the nondominated set. A non-parametric statistical test is used to verify if the rank distributions for the two optimisation schemes are significantly different or not.

4.4.1 Experimental Settings

Both performance assessment methods (binary hypervolume indicator and binary epsilon indicator) are used as a feedback collection mechanism during parameter control in a multiobjective Evolutionary Algorithm. We use a quality-proportionate selection mechanism to select the parameter values for the next iteration. Quality-

proportionate selection assigns each parameter value a selection probability proportional to their quality. For the purpose of this experiment, the quality of the parameter values is equal to the performance of the algorithm instance measured by the two performance assessment methods.

The crossover and mutation operators and their rates are probably the most conspicuous control parameters to optimise in stochastic optimisation [13]. Hence, for the benefit of these experiments the crossover and mutation rates were varied. For the crossover and mutation rates we use different value intervals to sample from. Preliminary trials have shown that a cardinality of two intervals or levels with ranges of $\{[0.6, 0.799], [0.8, 1.0]\}$ produced the best results among several cardinalities with even spreads between 0.2 and 1 for crossover rate, and two intervals of $\{[0.0010, 0.25], [0.2505, 0.5]\}$ for mutation rate.

We use six instances of the multiobjective Quadratic Assignment Problem obtained from the database of benchmark problems published at `http://dbkgroup.org/knowles/mQAP/`: a uniform biobjective instance of dimension n=20 (KC20-2fl-3uni), a uniform triobjective instance of dimension n=30 (KC30-3fl-2uni), two triobjective real instances of dimension n=30 (KC30-3fl-2rl and KC30-3fl-3rl), two biobjective real problem of dimension n=10 (KC10-2fl-5rl) and n=20 (KC20-2fl-5rl). All problems are NP-hard, and their Pareto fronts are not known.

The solution representation of the Quadratic Assignment problem is an array which describes the numbered locations and the values of the array represent the items. The problem is optimised using an Evolutionary Algorithm with string-based representation and customised crossover and mutation operators with their respective probabilities of being applied. Multipoint crossover swaps the assignments between solutions. The mutation operator changes the assigned item of a single location.

Every optimisation instance is granted 30000 function evaluations. To obtain a

fair comparison among the different effect assessment schemes we repeat each run 30 times, and report the results as boxplots. Furthermore, to check for a statistical difference of the results, the different effect assessment schemes of the optimisation methods are validated using the Kolmogorov-Smirnov (KS) nonparametric test [149].

4.4.2 Results

The means and standard deviations of the dominance rankings of the 30 repeated trials are reported in Table 4.2. The means show a significant difference between the result groups of the binary $\hat{h}$ indicator and the binary ϵ indicator.

Table 4.2: The means, standard deviations (SD) and the Kolmogorov-Smirnov test values of the 30 runs of each problem instance using different feedback collection strategies.

	Mean		**SD**		**KS test**	
Problem	$\hat{h}$	ϵ	$\hat{h}$	ϵ	d	p
KC10-2fl-1rl	**3.233**	4.600	0.12	0.30	0.3667	0.026
KC10-2fl-5rl	**3.667**	4.967	0.66	0.44	0.4333	0.005
KC20-2fl-5rl	**5.000**	6.300	0.50	0.09	0.3529	0.021
KC20-2fl-3uni	**13.41**	18.45	0.60	0.22	0.3793	0.022
KC30-3fl-2rl	**2.667**	4.500	0.88	0.30	0.3333	0.047
KC30-3fl-2uni	**5.500**	8.900	0.95	0.73	0.4000	0.011

The results show that means of the binary hypervolume indicator are above the respective values of the binary epsilon indicator. The least benefit the binary hypervolume indicator provides for the triobjective problem instance KC30-3fl-2rl. This is an inherent feature of the binary hypervolume indicator, whose accuracy decreases with the increase of the number of the objectives.

The results of a statistical analysis, depicted in Table 4.2, clearly show a significant difference between the result groups of the binary hypervolume indicator and the binary epsilon indicator.

All KS tests, used for establishing differences between independent datasets under the assumption that they are not normally distributed, result in a confirmation of the null hypothesis with a minimum d-value of 0.3333 at a 95% confidence level. Hence we conclude that the superior performance of the feedback collection mechanism using the binary hypervolume indicator is statistically significant.

4.5 Summary

In this chapter we investigated different feedback collection mechanisms for multiobjective Evolutionary Algorithms and performed an empirical evaluation of two recommended metrics: the binary hypervolume indicator and the binary epsilon indicator. When employed by the feedback collection mechanism, the binary hypervolume indicator produced better results quality compared to the binary epsilon indicator.

The calculation of binary hypervolume indicator becomes computationally expensive with the number of objectives. Previous works recommend that binary hypervolume indicator should be used with problems that have less than four objectives. The binary epsilon indicator, on the other hand, is easy to calculate. In the next chapters, we employ the binary hypervolume indicator as a feedback collection mechanism in adaptive parameter control for multiobjective Evolutionary Algorithms.

Chapter 5

Parameter Effect Assessment Strategy

5.1 Introduction

The performance of the optimisation algorithm, measured as described in Chapter 4, is used as evidence to approximate the effect of parameter values. This step determines the relationship between the solution space and the parameter space.

The most common way of assessing the effect of parameters is by using the improvement in specific properties of the generated solutions with respect to a reference solution s_{ref}. This reference point can be either the parents of the generated solutions [94, 183, 191, 79, 85, 83, 84, 56, 177], the best solution so far [37, 114, 65, 47], or a specified quantile, e.g. the median [94, 95]. For example, if a mutation rate 0.01 is applied to create a solution s' from its parent s, the effect of the mutation rate on the performance of the algorithm is equal to the fitness gain $\Delta f = f(s') - f(s)$.

In the approach of Srinivas and Patnaik [177], the operator rates are inversely proportional to the difference between the maximum fitness and the average fitness of the population. A low difference is associated with solutions with similar fitness, hence the operator rates are increased to introduce more diversity in the population. As a result, lower values of operator rates are applied to high-fitness solutions, and

higher values of operator rates are used to change low-fitness solutions.

In essence, state-of-the-art parameter effect assessment methods associate specific properties of an EA with particular parameters, and vary the parameter settings according to the changes in these properties. However, Evolutionary Algorithms are stochastic systems, which may produce different results quality for the same parameter values [43]. Moreover, the performance of the algorithm can be affected by more than one parameter value. Hence using the performance improvement reported by the feedback strategy directly as an indication of the effect of parameter values may be misleading. The uncertainty regarding the real cause of the algorithm performance improvement and the stochastic nature of the EAs should be taken care of by the parameter effect assessment strategy, which leads us to the following research question:

- **What is an effective method for assessing the stochastic effect of EA parameters?**

To accommodate the stochastic nature of EAs we employ statistical tools which approximate the effect of parameter values based on conditional probabilities. We model the relationship between the performance of the algorithm and parameter values as a Bayesian Belief Network, and define a probabilistic effect measure called Bayesian Effect Assessment (BEA). Instead of using the performance improvement of the EA directly, the Bayesian Effect Assessment Strategy is conditional on the usage of the parameter values, as a result calculating their relative effect on the performance of the optimisation algorithm. The same parameter values are applied to more than one solution and a success measure is defined, which counts the number of times a parameter value was used and the number of times the solutions using this value were successful. Success is defined as producing solutions with feedback above a certain threshold. The threshold value covers the uncertainty of EA outcomes.

5.2 Bayesian Effect Assessment Strategy

In our approach, the optimisation process is carried out in $X = \{x_1, x_2, ..., x_k\}$, k parallel algorithm instances. Each algorithm instance evolves independently and every iteration reports to an algorithm manager, which coordinates all algorithm instances. At the start of the optimisation process, the algorithm manager randomly assigns each instance different parameter values. However, the same parameter value can be assigned to more than one instance. Figure 5.1 illustrates this process for n parameters ($\{\upsilon_1, ..., \upsilon_n\}$).

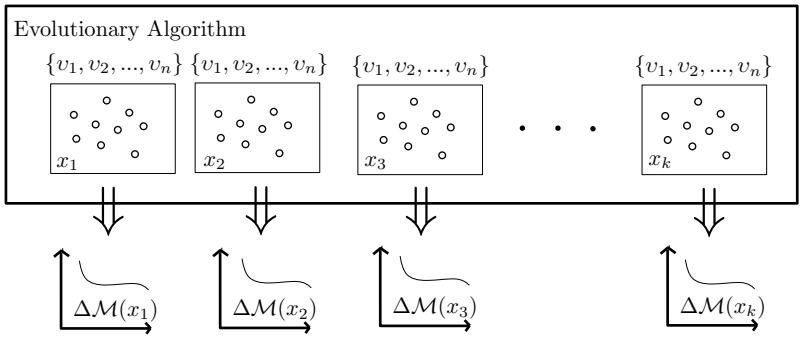


Figure 5.1: Assessing the effect of parameters with parallel instances of the optimisation algorithm

Each algorithm instance has the same number of solutions, in the range $[1, \mu]$, which can also be subject to optimisation. During every iteration, new solutions are created in each algorithm instance by using the parameter values assigned to them. At the end of each iteration, the algorithm instances report the performance that was achieved during that iteration to the algorithm manager.

We denote the set of performance metrics used to measure the performance of an Evolutionary Algorithm as $\mathcal{M} = \{m_1, m_2, ..., m_p\}$. At each iteration, the achieved performance for each algorithm instance is measured by calculating the improvement in the properties of the current solutions, measured by these metrics, with respect to the properties of the solutions in the previous iteration, as shown in Equation 5.1.

$$\Delta\mathcal{M}(x_i) = \mathcal{M}(x_i^t) - \mathcal{M}(x_i^{t-1}) \tag{5.1}$$

We denote the effect of parameter values on this performance change as e. This performance difference determines if the parameter value was successful, denoted as e^+, or unsuccessful, denoted as $e = e^-$. Success is defined as producing solutions with performance values above a certain threshold th. Given an algorithm instance x_i, the effect of parameter values used in that instance is calculated as:

$$e(x_i) = \begin{cases} e^+ & \text{if} \quad \Delta\mathcal{M}(x_i) > th \\ e^- & otherwise \end{cases}$$

In other words, if the change in the performance metric of an algorithm instance in the last iteration, i.e. the difference between current (time t) $\mathcal{M}(x_i^t)$ and the value of the performance metric at the previous iteration $\mathcal{M}(x_i^{t-1})$ is above th, that instance is deemed successful ($e(x_i) = e^+$), which also means that the parameter values used to produce the solutions are considered successful.

The value of the threshold th determines the greediness of the algorithm. For smaller values of th (e.g. 5% of the best solutions) the focus of parameter control is narrowed to the absolute best-performing parameter values. Larger values of th (e.g. 50% of the best solutions) allow a better exploration of parameter values with a moderate performance. The effect of this hyperparameter is investigated in Section 5.4.

The successful or unsuccessful performance of a parallel instance x_i can be affected by different factors. First, Evolutionary Algorithms have more than one parameter that can affect their performances, which makes it difficult to reward specific parameters according to their merits. Second, Evolutionary Algorithms are stochastic solvers, which return different results for the same parameter settings.

An ideal parameter effect assessment strategy would consider these two kinds of uncertainties. Hence, we propose to use Bayesian Belief Network (BBN) [44] as a probabilistic parameter effect assessment strategy.

BBNs measure cause-effect relationships between variables in a probabilistic way. They are considered to be effective tools in modelling systems which do not have complete information about the variables and the available data is stochastic or not completely available.

BBNs are represented as directed acyclic graphs (DAGs) which have probability labels for each node as a representation of the probabilistic knowledge. The graph is defined as a triplet (V, L, P) [44]. $V = (v_1, v_2, ...v_n)$ is a set of n variables represented by nodes of the DAG. We think of each variable as an event which consists of a finite set of mutually exclusive states. It is also possible to represent continuous variables, representing a numerical value such as crossover probability, by discretising them into a number of intervals.

L is the set of links that denote the causal relationship among the variables V, modelled as directed arcs between nodes of the DAG. The directed links between variables represent dependence relationships. A link from a variable v_1 to variable v_2 indicates that v_2 depends on the parent node v_1, which is denoted as $\pi(v_2)$. It can also be explained as a causal relationship where v_1 is the cause of the event v_2. The lack of a link between two nodes means that the nodes are not dependent on each other. This relationship is represented graphically in Figure 5.2.

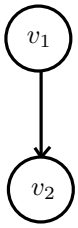


Figure 5.2: An example of a Bayesian Belief Network.

We use BBNs to model the relationship between algorithm parameters and their effect on the performance of the algorithm. The effect on the algorithm is represented by a child node with the name e, which takes two values: e^+ for a successful effect and e^- for an unsuccessful effect. The child node e has many parent nodes which represent the algorithm parameters denoted as $\{\upsilon_1, \upsilon_2, ..., \upsilon_n\}$, as shown in Figure 5.3. Each node υ_i can take different values, which we denote similarly to parameter values (ranges) as $\{\upsilon_{i1}, ..., \upsilon_{im_i}\}$.

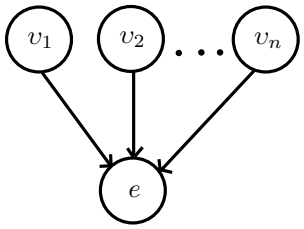


Figure 5.3: A BBN modelling the relationship between algorithm parameters and the performance of the algorithm.

Each of the parent nodes υ_i is annotated with probabilities $p(\upsilon_{ij})$, as shown in Figure 5.4. These represent the prior probabilities, which can be calculated as classical probabilities, to represent the probability that a certain parameter value will occur, or as Bayesian probabilities, to represent a belief that a certain parameter value will be used. In the first iteration, we assign equal prior probabilities to each value of the parameters, calculated as:

$$p(\upsilon_{ij}) = \frac{1}{m_i} \tag{5.2}$$

where m_i is the total number of values or intervals of parameter υ_i. This represents our belief in how frequently a certain parameter value should be used. These prior probabilities are used to select the parameter values to assign to the algorithm instances in the first iteration.

The probability that the child node e attains a certain value, in our case e^+ or

e^-, depends on the probabilities of the parent nodes, i.e. the probabilities of the parameter values. This relationship is measured as a conditional probability, i.e. the probability of an event occurring given that another event has already happened, defined as: $P = \{p(e \mid \pi(e)) \mid \pi(e) \in V\}$, where $\pi(e)$ are the parents of node v.

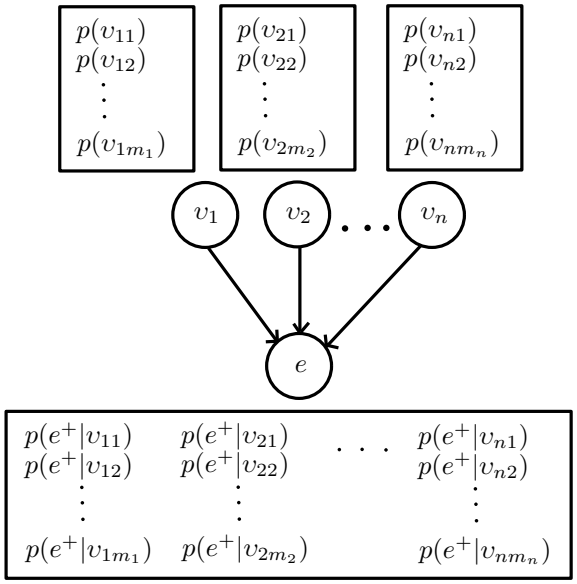


Figure 5.4: A Bayesian Belief Network annotated with conditional probabilities employed by the parameter effect assessment strategy.

To calculate the conditional probabilities, every iteration, we establish the number of times the value has been used, denoted as $n(v_{ij})$ and the number of times the value led to a success $n(v_{ij} \wedge e^+)$ (the number of successful instances that use that parameter value). Then we calculate the conditional probabilities for each parameter value by using Equation 5.3. These conditional probabilities represent the success rate of each parameter value in the current iteration.

$$p(e^+|v_{ij}) = \frac{n(v_{ij} \wedge e^+)}{n(v_{ij})} \tag{5.3}$$

where $n(v_{ij} \wedge e^+)$ is the successful attempts that use v_{ij} and $n(v_{ij})$ is the number of times v_{ij} was used. The conditional probabilities of all parameter values $v_{ij}, i \in$

$n, j \in m$ describe the likelihood of the causes of the successful performance of the algorithm instances. Conditional probabilities, are calculated for all parameter values resulting in the annotated graph shown in Figure 5.4.

The initial structure of the BBN and the conditional probabilities are learnt based on the data from the k parallel optimisation instances. Algorithm 2 shows the steps of the process.

Algorithm 2 Bayesian Effect Assessment Strategy

```
1: procedure EAS
2:     for all parameter υ_i, i ∈ n do
3:         for all algorithm instance x_s, s ∈ k do
4:             for all parameter value υ_ij, j ∈ m do
5:                 if υ_ij is used in x_s then
6:                     n(υ_ij) = n(υ_ij) + 1
7:                     if ΔM(x_s) > th then
8:                         n(υ_ij ∧ e+) = n(υ_ij ∧ e+) + 1
9:                     end if
10:                end if
11:            end for
12:        end for
13:    end for
14:    for all υ_i, i ∈ n do
15:        for all υ_ij, j ∈ m do
16:            p(e+|υ_ij) = n(υ_ij ∧ e+) / n(υ_ij)
17:        end for
18:    end for
19: end procedure
```

This probabilistic effect assessment strategy calculates the relative success rate of each parameter value with respect to other parameter values. As a result, the calculated conditional probabilities deal with the uncertainties that arise from the stochastic nature of Evolutionary Algorithms by measuring the effect of parameters probabilistically, instead of using directly the performance difference, as other state-of-the-art methods.

5.3 Example

We illustrate the parameter effect assessment procedure described in Algorithm 2 with an artificial data set, which has three categorical variables: two controlled parameters (υ_1, υ_2), each of them having two intervals: $\upsilon_{11}, \upsilon_{12}$ and $\upsilon_{21}, \upsilon_{22}$ respectively, and the performance increase of the algorithm e, which can take two different values: e^+ and e^-. As mentioned in Section 5, when the performance of the algorithm is above a certain threshold, we consider that instance successful, denoted as $e = e^+$, otherwise it is deemed unsuccessful, denoted as $e = e^-$.

Suppose that we have 10 parallel algorithm instances, denoted as $\{x_1, x_2, ...x_{10}\}$. Each instance uses parameter values for the two controlled parameters (υ_1 and υ_2) selected from different intervals as shown in Figure 5.5. The dots symbolically represent the solution sets of the instances.

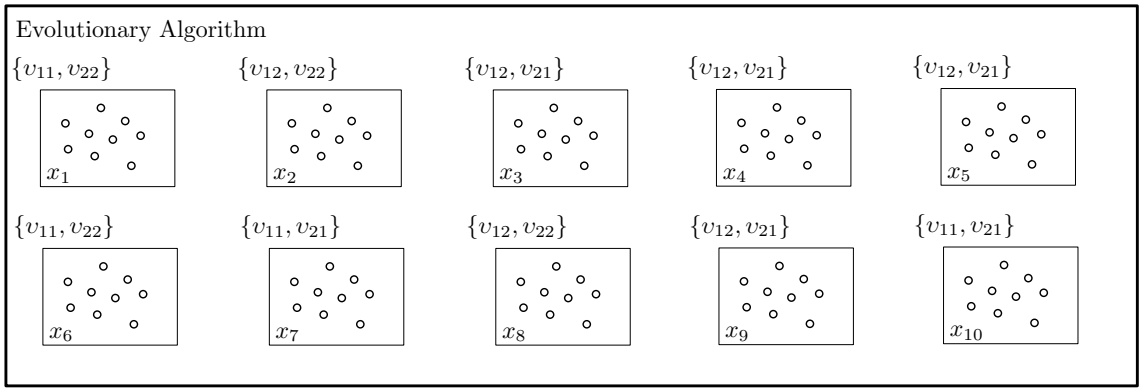


Figure 5.5: Algorithm instances with parameter values selected from different intervals.

After running the EA with the different instances, the performance of each instance is assessed separately. Suppose that instances $\{x_1, x_2, x_7, x_{10}\}$ are deemed successful as shown in Figure 5.6

The same results are written in a table format as depicted in Table 5.1.

Using the data in Table 5.1, we calculate the frequency of each parameter value in the 10 instances. Results are shown in Table 5.2.

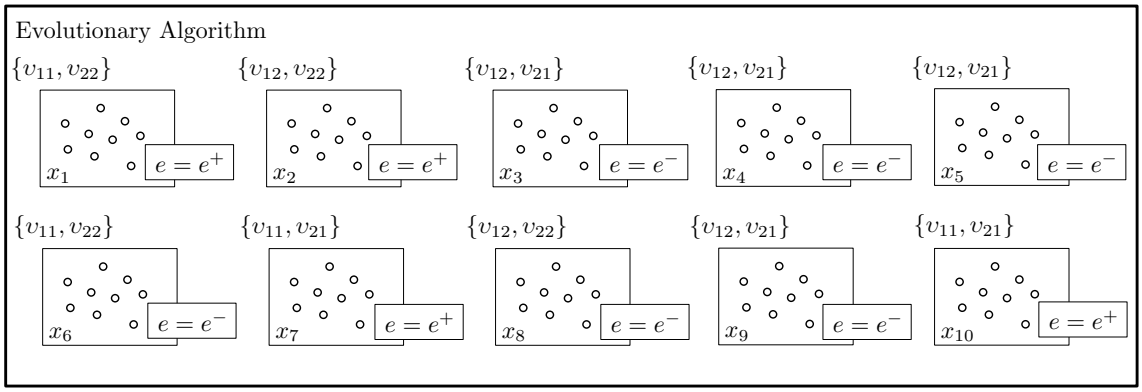


Figure 5.6: Successful algorithm instances.

Table 5.1: Performance of algorithm instances with different values for parameters v_1 and v_2.

Instance	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
v_1	v_{11}	v_{12}	v_{12}	v_{12}	v_{12}	v_{11}	v_{11}	v_{12}	v_{12}	v_{11}
v_2	v_{22}	v_{22}	v_{21}	v_{21}	v_{21}	v_{22}	v_{21}	v_{22}	v_{21}	v_{21}
e	e^+	e^+	e^-	e^-	e^-	e^-	e^+	e^-	e^-	e^+

Table 5.2: Frequencies of parameter values.

$n(v_{11}) = 4$
$n(v_{12}) = 6$
$n(v_{21}) = 6$
$n(v_{22}) = 4$

The next step involves calculating the frequency of each parameter value in the successful and unsuccessful instances. Results are shown in Table 5.3.

Table 5.3: Frequencies of parameter values in the successful and unsuccessful instances.

$n(v_{11} \wedge e^+) = 3$	$n(v_{11} \wedge e^-) = 1$
$n(v_{12} \wedge e^+) = 1$	$n(v_{12} \wedge e^-) = 5$
$n(v_{22} \wedge e^+) = 2$	$n(v_{21} \wedge e^-) = 4$
$n(v_{21} \wedge e^+) = 2$	$n(v_{22} \wedge e^-) = 2$

Looking at the frequencies of parameter values in the successful algorithm instances, one may think that parameter values υ_{21} and υ_{22} have had the same effect in the successful performance of the algorithm since they all have equal numbers in the successful class. However, these numbers do not consider the times these parameter values were unsuccessful. The conditional probabilities calculated in the final step of the parameter effect assessment strategy using Equation 5.3 incorporate this information. Results are shown in Table 5.4.

Table 5.4: The conditional probabilities of parameter values.

$p(e^+\|\upsilon_{11}) = 3/4$	$p(e^-\|\upsilon_{11}) = 1/4$
$p(e^+\|\upsilon_{12}) = 1/6$	$p(e^-\|\upsilon_{12}) = 5/6$
$p(e^+\|\upsilon_{21}) = 1/3$	$p(e^-\|\upsilon_{21}) = 4/6$
$p(e^+\|\upsilon_{22}) = 1/2$	$p(e^-\|\upsilon_{22}) = 2/4$

The data in Table 5.4 shows that although parameter values $\upsilon_{22}, \upsilon_{21}$ have the same frequency in the successful algorithm instances, they do not have the same success rates, i.e. their effect on the successful performance of the algorithm is not the same.

Similarly, the conditional probabilities help in deriving conclusions about the causes of the successful or unsuccessful performance of the algorithm instances. For example, the success rate of parameter value υ_{11} is $p(e^+|\upsilon_{11}) = 3/4$, which is higher than $p(e^+|\upsilon_{12}) = 1/6$. This means that parameter value υ_{11} has a higher probability of being the cause of the successful performance of the algorithm instances compared to υ_{12}.

5.4 Analysis of the Threshold Value

The Bayesian Effect Assessment strategy described in Section 5.2 uses a threshold value th to classify the algorithm instances into groups of high- and low-quality and determine if a parameter value has been successful. The value of th determines the greediness of the parameter effect assessment strategy. For instance, classifying only the 10% best performing algorithm instances successful (a threshold value equal to the 10th percentile), results in a fast convergence to the best parameter values. A larger value of th (e.g. the 30th percentile) allows the exploration of parameter values with a moderate performance. In this section, we investigate how this hyper-parameter effects the performance of the optimisation algorithm.

5.4.1 Experimental Settings

To examine the effect of the threshold on the performance of the optimisation algorithm we test four different values: 10th percentile, 20th percentile, the average and the median performance gain. For the benefit of these experiments, the crossover and mutation rates were varied. For the crossover and mutation rates we use different value intervals to sample from, with a cardinality of two intervals with ranges of $\{[0.6, 0.799], [0.8, 1.0]\}$ and two intervals of $\{[0.0010, 0.25], [0.2505, 0.5]\}$ for the mutation rate.

For the optimisation process, we use problem instances of the Quadratic Assignment Problem (TAI30B, STE36B), the multiobjective Quadratic Assignment Problem (KC30-3fl-1rl, KC30-3fl-2rl), and the Royal Road Problem. For the multiobjective Quadratic Assignment Problem instances we record the final hypervolume indicator value as a performance indicator. Different QAP instances have different scales of the fitness values. To make the results comparable, we normalise the fitness function as follows:

$$f_{norm}(s) = \frac{f_{max} - f(s)}{f_{max} - f_{min}} \tag{5.4}$$

where f_{max} is the maximum value and f_{min} is the minimum value that the fitness function f can take. Since QAP is a minimisation problem, the normalisation of the results converts it into a maximisation problem. Hence, the normalised fitness of the results increases over time.

The solution representation of the Quadratic Assignment problem is an array which describes the numbered locations and the values of the array represent the items. The problem is optimised using an Evolutionary Algorithm with string-based representation and customised crossover and mutation operators with their respective probabilities of being applied. Multipoint crossover swaps the assignments between solutions. The mutation operator changes the assigned item of a single location.

5.4.2 Results

The means and standard deviations of the 30 runs for each threshold value are listed in Table 5.5, which clearly show a significant difference between the result groups of the method using the median and the methods using the average, 10th, and 20th percentiles. The mean performance of the method using the median as a threshold value is consistently above the means of the other methods for all problems.

As the threshold value using the median performance gain consistently outperforms the three other threshold values, we employ the Kolmogorov-Smirnov (KS) non-parametric test [149] to check for a statistical difference. The 30 hypervolume indicators and normalised fitness values of the repeated trials for each of the problem instances were submitted to the KS analysis.

The median performance gain was compared to the other three threshold values,

Table 5.5: The means and the standard deviations of the optimisation schemes using different threshold values for the Bayesian Effect Assessment (BEA) strategy.

	Mean			
Problem	10%	20%	Average	Median
Royal Road	7.10E-03	7.07E-03	7.06E-03	**7.35E-03**
TAI30B	0.5145	0.5120	0.5170	**0.5354**
STE36B	0.7860	0.8302	0.8277	**0.8558**
KC30-3fl-1rl	0.1425	0.1428	0.1441	**0.1464**
KC30-3fl-2rl	0.6004	0.6003	0.6009	**0.6072**
	Standard Deviation			
Problem	10%	20%	Average	Median
Royal Road	1.360E-03	1.372E-03	1.381E-03	1.442E-03
TAI30B	1.541E-02	1.323E-02	1.463E-02	2.608E-02
STE36B	6.790E-03	3.140E-02	3.383E-02	6.696E-03
KC30-3fl-1rl	3.936E-03	3.809E-03	3.727E-03	2.809E-03
KC30-3fl-2rl	3.903E-03	7.831E-03	6.768E-03	6.554E-03

Table 5.6: The Kolmogorov-Smirnov test for comparing the 30 runs of the optimisation schemes using different threshold values for the Bayesian Effect Assessment (BEA) strategy.

	Median vs. 10%		**Median vs. 20%**		**Median vs. Average**	
Problem	d	p	d	p	d	p
Royal Road	0.4333	0.005	0.4667	0.002	0.4667	0.002
TAI30B	0.5000	0.023	0.5625	0.007	0.5000	0.023
STE36B	1.0000	0.000	0.5000	0.023	0.5625	0.007
KC30-3fl-1rl	0.4891	0.014	0.5625	0.003	0.4375	0.037
KC30-3fl-2rl	0.5375	0.013	0.4917	0.030	0.6167	0.003

with a null hypothesis of no difference between the performances (Median vs. 10%, Median vs. 20% and Median vs. Average). The KS tests resulted in a rejection of the null hypothesis with a minimum d-value of 0.4375 at a 95% confidence level (see Table 5.6). Hence, we conclude that the superior performance of the Evolutionary Algorithm using the median as a threshold value is statistically significant.

These results indicate that using the median as a threshold value for the Bayesian Effect Assessment (BEA) strategy produces better results quality and is more robust than using the other threshold values. Hence, we employ the median performance gain as a threshold value in the experiments presented in the rest of the thesis.

5.5 Validation

The Bayesian Effect Assessment Strategy (BEA) introduced in Section 5.2 measures the effect of parameter values based on a success measure instead of using the quality gain of the generated solutions directly. We investigate if the application of the Bayesian Effect Assessment Strategy (BEA) to parameter control improves the choice of the parameters and if its use results in better algorithm performance. We compare results quality achieved by using BEA with a method that uses directly the quality gain of the solutions. In essence, the research question investigated in this section is the following:

- **What is an effective method for assessing the stochastic effect of EA parameters?**

5.5.1 Benchmark Effect Assessment Strategy

As a benchmark parameter effect assessment strategy we employ the approach of Srinivas and Patnaik (SP) [177], which calculates the effect of the variation operators (mutation rate and crossover rate) based on the maximum fitness f_{max} and the average fitness $\overline{f}$ of the population. The crossover rate $\hat{c}_r$ and mutation rate $\hat{m}_r$ are computed as follows:

$$\hat{c}_r = \frac{k_1(f_{max} - \Delta f(s_{\hat{c}_r}))}{f_{max} - \overline{f}}, \quad \hat{m}_r = \frac{k_2(f_{max} - \Delta f(s_{\hat{c}_r}))}{f_{max} - \overline{f}} \tag{5.5}$$

where k_1 and k_2 are two constants selected in the range $[0.0, 1.0]$ and $\Delta f(s_{\hat{c}_r})$ is the fitness gain of solutions compared to their parents. The operator rates calculated by Equation 5.5 are inversely proportional to the difference between the maximum fitness and the average fitness of the population. A low difference means that the

fitnesses of the solutions are too similar, hence the operator rates are increased to introduce more diversity in the population. Furthermore, the operator rates are increased if the difference between the maximum fitness in the population and the fitness of the solution created by applying the operator rate ($f_{max} - f$) is low, i.e. lower values of operator rates are applied to high fitness solutions, and higher values of operator rates are used to change low fitness solutions.

5.5.2 Experimental Settings

Bayesian Affect Assessment (BEA) is compared to the parameter effect assessment strategy introduced by Srinivas and Patnaik (SP), in which the reference points are the maximum and the average fitness of the solutions. For the benefit of these experiments, the crossover and mutation rates were varied. For the crossover and mutation rates we use different value intervals to sample from, with a cardinality of two intervals with ranges of $\{[0.6, 0.799], [0.8, 1.0]\}$ for crossover rate and $\{[0.0010, 0.25], [0.2505, 0.5]\}$ for mutation rate.

For the optimisation process, we use problem instances from the Quadratic Assignment Problem (BUR26A, BUR26B and BUR26E) the multiobjective Quadratic Assignment Problem (KC30-3fl-3uni, KC20-2fl-5rl, KC30-3fl-2rl and KC30-3fl-3rl), and the Royal Road Problem. Every instance is granted 90000 function evaluations. To obtain a fair comparison among the different parameter effect assessment schemes we repeat each run 30 times, and report the results as boxplots.

Furthermore, to check for a statistical difference of the results, the different parameter effect assessment schemes of the optimisation methods are validated using the Kolmogorov-Smirnov (KS) non-parametric test [149]. For each parameter effect assessment scheme, we record the final hypervolume indicator value for the multiobjective problem instances and the normalised fitness (Equation 5.4) value for the

singleobjective problems, and compare the achieved results.

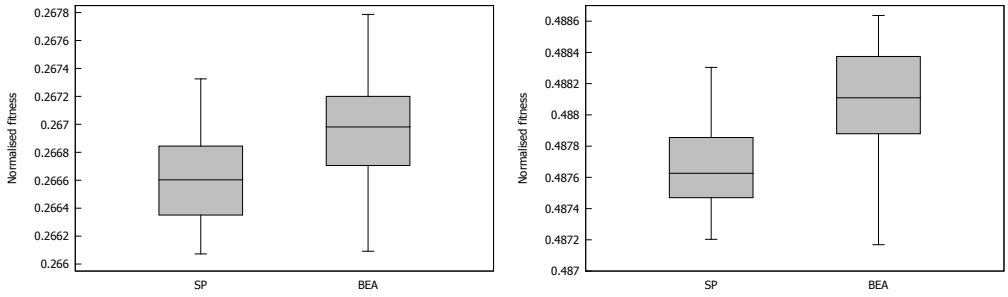
5.5.3 Results

The 30 results of the repeated trials are presented as boxplots in Figure 5.7. The empirical results are not normally distributed, but the mean and 25th percentile of the Bayesian Effect Assessment (BEA) method are consistently above the respective values of the benchmark.

The means and standard deviations of the two parameter effect assessment schemes are listed in Table 5.7. The mean performance of the Bayesian Effect Assessment (BEA) strategy is consistently above the average of the benchmark parameter effect assessment method (SP) which uses the fitness gain with respect to the best and average fitnesses.

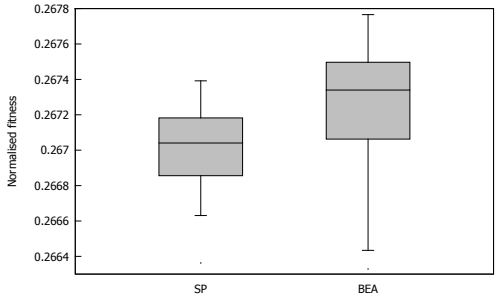
To understand if the obtained difference is not due to chance, the results of the optimisation methods with the two parameter effect assessment schemes are validated using the Kolmogorov-Smirnov (KS) non-parametric test [149], which checks for a statistical significance in the outperformance. The 30 values of the fitness functions and hypervolume indicators of the repeated trials for each problem instance were submitted to the KS analysis and results are shown in Table 5.7. BEA was compared to the approach of Srinivas and Patnaik (SP), with a null hypothesis of no difference between the performances.

All KS tests, used for establishing differences between independent datasets under the assumption that they are not normally distributed, result in a rejection of the null hypothesis with a minimum d-value of 0.3448 at a 95% confidence level. Hence we conclude that the superior performance of the Bayesian Effect Assessment method is statistically significant.

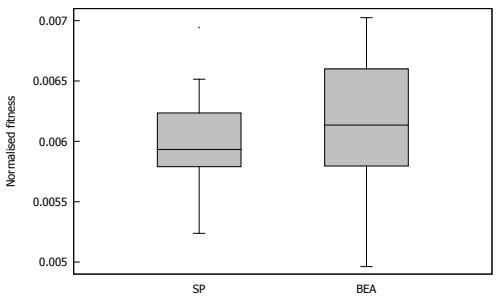


(a) BUR26A

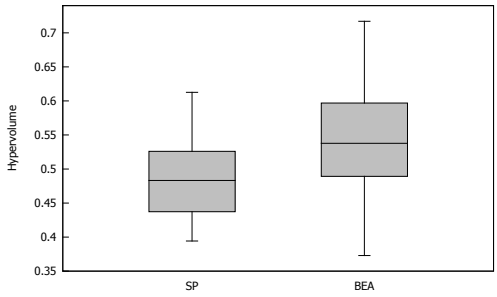
(b) BUR26B



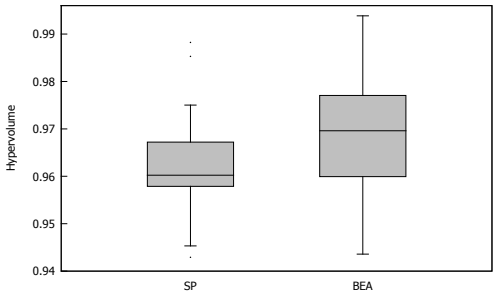
(c) BUR26E



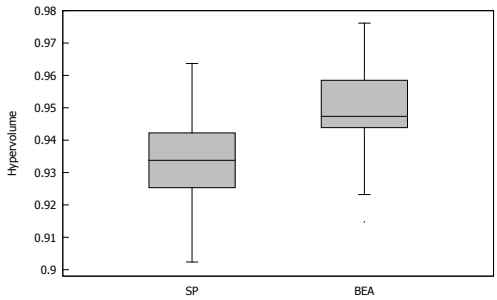
(d) Royal Road



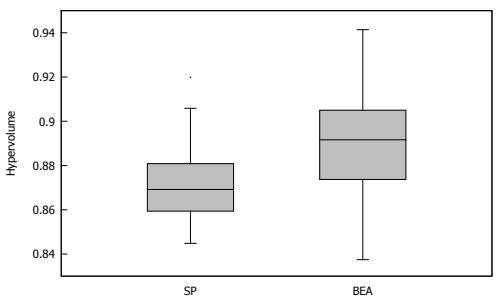
(e) KC30-3fl-3uni



(f) KC20-2fl-5rl



(g) KC30-3fl-2rl



(h) KC30-3fl-3rl

Figure 5.7: Boxplots of the 30 trials of the two parameter effect assessment schemes.

Table 5.7: The means, standard deviations and Kolmogorov-Smirnov test values of fitness functions for the 30 runs using different parameter effect assessment schemes.

	Mean		**Standard Deviation**		**KS test**	
Problem	BEA	SP	BEA	SP	d	p
Royal Road	**6.15E-03**	5.97E-03	5.478E-04	3.616E-04	0.3448	0.048
BUR26A	**0.2669**	0.2666	4.794E-04	3.998E-04	0.4381	0.006
BUR26B	**0.4881**	0.4876	4.409E-04	3.673E-04	0.5517	0.000
BUR26E	**0.2672**	0.2670	4.474E-04	2.491E-04	0.4828	0.001
KC20-2fl-5rl	**0.9687**	0.9619	1.315E-02	1.053E-02	0.3793	0.022
KC30-3fl-2rl	**0.9488**	0.9339	1.484E-02	1.639E-02	0.5517	0.000
KC30-3fl-3rl	**0.8893**	0.8702	2.385E-02	1.453E-02	0.4483	0.004
KC30-3fl-3uni	**0.5505**	0.4852	8.653E-02	5.921E-02	0.3793	0.022

5.6 Summary

This chapter presented a Bayesian Effect Assessment (BEA) strategy which measures the effect of parameter values based on conditional probabilities (success rates). Unlike state-of-the-art adaptive parameter control methods, in which the improvement of the quality of the solutions reported by the feedback strategy is considered as the effect of parameter values, BEA attempts to infer the cause of the improvement and to what extent the change in the quality of the solutions is affected by the parameter values. We performed a set of experiments on the Royal Road Problem, the Quadratic Assignment Problem and the multiobjective Quadratic Assignment Problem, in which Bayesian Effect Assessment produced better algorithm performance compared to using directly the performance improvement of the EA reported by the feedback strategy.

Chapter 6

Parameter Quality Attribution Strategy

6.1 Introduction

The previous chapter derived the effect of parameter values based on the performance of the optimisation algorithm. The effect, assessed at every iteration, provides new information regarding the quality of the parameter values, the estimation of which constitutes the Parameter Quality Attribution Strategy. The recorded effect of parameter values is the input to the quality estimation rule, which updates its internal state with the newly assessed effect for a projection of the performance of the value in the future. The estimated quality of the parameter values is employed to make an 'educated guess' about the probability of the parameter values producing high quality results in the next iteration.

State-of-the-art parameter quality attribution methods use a predefined time window W (the W last iterations) to approximate the quality of parameter values for the preceding iteration $t-1$. Igel et al. [85] use this time window (adaptation cycle) to estimate the quality of parameter values as the average effect on algorithm performance. Fialho et al. [55, 54, 56] use an Extreme value-based Quality Attribution (EQA) strategy, which employs the maximum effect in W iterations to

approximate the quality of parameter values in the preceding iteration $t-1$. The Extreme value-based Quality Attribution strategy uses a normalisation scheme, which divides parameter effects by the best effect achieved so far by any parameter value.

An Extreme value-based Quality Attribution strategy is also presented by Whitacre et al. [189], in which statistical tests are employed to detect outliers, i.e. parameter values that produce exceptionally good solutions. A parameter value is considered an outlier if it produces a solution which is statistically not expected in the population. The parameter effect assessment strategy calculates the probability that a solution is an outlier. The sum of all outlier probabilities of the solutions produced by a parameter value represents its quality. The main idea is to reward parameter values that generate solutions which are rare but very fit compared to the other members of the population. Other parameter values that generate frequent solutions with small improvements are not regarded as important and thus they are not used in the next iteration.

In general, adaptive parameter control methods [33, 85, 55, 189] use the derived parameter quality from recent or past performance, i.e. $q_{t-1}(\upsilon_{ij})$, to select the parameter values to use in the next iteration (time t). This is accomplished by considering the average (AQA) or the extreme (EQA) effects of parameter values in the preceding W iteration(s). One might argue that these approaches are 'one step behind', in that they represent the parameter value which is optimal for the previous iteration (time $t-1$). Ideally, we would use a setting optimised for the beginning iteration (time t). It follows that an ideal parameter control method would attempt to predict successful parameter values for time t based on previous performance. Acknowledging this fact, we investigate the following research question:

* **What is an effective parameter quality attribution method for projecting successful parameter values to the next iteration of the optimisation process?**

Given the problem with deriving parameter values based on very recent, but indeed past, performance, we use a method of Predictive Quality Attribution (PQA) that combines a measure of past performance with time series prediction to predict the performance of parameter values. PQA records the effect (success rates) of every parameter value for each iteration. Instead of applying the effect in the last iteration(s), i.e. time $t-1$, directly as the quality for the parameter values, PQA applies time series prediction as a method of forecasting the quality (time t) to be applied in the parameter control strategy. We investigate if using Predictive Quality Attribution (PQA) improves the algorithm performance, by comparing the results with two state-of-the-art quality attribution strategies: Average Quality Attribution (AQA) and Extreme Quality Attribution (EQA).

6.2 Predictive Quality Attribution

Predictive Quality Attribution (PQA) applies forecasting techniques to predict the quality of parameter values to be applied in the parameter control strategy. The effect of parameter values (success rates) is recorded for each iteration. The time series data for a single parameter value υ_{ij} is recorded as shown in Equation 6.1:

$$p_1(e^+|\upsilon_{ij}), p_2(e^+|\upsilon_{ij}), ..., p_{t-1}(e^+|\upsilon_{ij}) \tag{6.1}$$

where $t-1$ is the current iteration and $p_s(e^+|\upsilon_{ij})$ is the effect (success) of the parameter value υ_{ij} at iteration s, which is calculated using Equation 5.3.

The success history of each parameter value/range, given by the time series in Equation 6.1, is used to forecast the success rate of that parameter value/range for the next iterations. Forecasting necessitates the use of analytical methods to predict the future values of a system, i.e. the success rates of parameter values given the evidence from the behaviour of this system in the past. In Predictive Quality Attribution, this process requires the identification of the relationship between future (predicted) success rates of parameter values and the past success rates.

Forecasting methods abound in the literature, ranging from simple regression analysis to more complex and sophisticated techniques such as Autoregressive Moving Average models. The forecasting techniques we explore are Linear Regression (LR), Simple Moving Average (SMA), Exponentially-Weighted Moving Average (EWMA), and Autoregressive Moving Average (ARIMA).

Linear Regression (LR) fits a linear equation to the observed data. This model works on the assumption that the success rates have a linear relationship with time. LR is one of the simplest forecasting models, yet it is a powerful model, provided that the data conforms to the assumptions that this model makes. If the time-series is not linear, other models such as Simple Moving Average (SMA), Exponentially

Weighted Moving Average (EWMA) or Autoregressive Integrated Moving Average (ARIMA) can be used to forecast the next parameter values.

The purpose of employing a Simple Moving Average model is to measure trends by smoothing the data using the mean of the success rates. As a result, the trend in the data is extrapolated into a forecast. The assumption is that prior trend movements will continue. EWMA on the other hand focuses mostly on recent success rates and captures shorter trends in the data. As a result, an EWMA is expected to reduce any lags in the simple moving average such that the fitting line is closer to the real values of the success rates compared to a simple moving average. One of the disadvantages of EWMA is that it is prone to sudden changes in the success rates, which means that it may easily be affected by noise. In order to deal with this problem, one could increase the length of the moving average term or switch from an EWMA to an SMA. The ARIMA model is a combination of moving average models and a special case of the linear regression models. The ARIMA model arguably is more powerful the LR, SMA and EWMA, however, because of its power and flexibility, ARIMA is a complex technique.

Using a more complex model may not necessarily lead to better results, since all models make assumptions and have their limitations. Hence, different forecasting methods are explored and the accuracy level in the forecast of each one using the time series data obtained from the optimisation process is measured. We check the assumptions made by each forecasting technique (e.g. linearity and normality of errors) to make a judgement of the suitability of the model.

6.2.1 Linear Regression

The linear regression model is one of the simplest forecasting models used. Linear regression builds a linear mathematical model of past observation that can be used

to forecast future values of a discrete time-series. This model makes an assumption about the linearity of the data and normality of the errors distribution.

The assumption of the errors of linear models being normally distributed is often justified by using the central limit theorem, which states that the sum of a sufficiently large number of independent random variables (from any kind of distributions) is normally distributed. If the success rates of parameter values occur randomly and somewhat independently, the variations in the data can be expected to be approximately normal in distribution. This assumption facilitates the calculation of the coefficient, by minimising the mean squared error, which is easy to calculate.

Regression models are based on the idea of the *regression to the mean*, i.e. if the time step is x standard deviations from the mean, the model predicts that the next value of the success rate of a parameter value will be $r \cdot x$ standard deviations from the mean of the time series, where r is the correlation coefficient between time and success rates. This is an indication of the extent to which the linear model can be used to predict the deviation of the success rate of a parameter value $p(e^+|\upsilon_{ij})$ from the mean when the deviation of time t from the mean is known.

In essence, the correlation coefficient measures how strongly the success rate of a parameter value and time are related to each other in a linear way, by using a scale of -1 to $+1$. The correlation coefficient is positive if both variables vary on the same side of the respective means, i.e. the success rate increases with time, and negative if the vary on opposite sides of their means. If the success rate varies independently of the number of iterations, the correlation coefficient is equal to zero.

Before computing the correlation coefficient, the current success rate and time-step are standardised by expressing them in units of standard deviations from their own mean. The standardised success rate $\mathcal{P}_{t-1}(e^+|\upsilon_{ij})$ of parameter value υ_{ij} at time-step $t-1$ is calculated using the time-series in Equation 6.1 as follows:

$$\mathcal{P}_{t-1}(e^+|\upsilon_{ij}) = \frac{p_{t-1}(e^+|\upsilon_{ij}) - \overline{p}_{t-1}(e^+|\upsilon_{ij})}{\sigma(p_{t-1}(e^+|\upsilon_{ij}))} \tag{6.2}$$

where $\overline{p}_{t-1}(e^+|\upsilon_{ij})$ is the mean of the success rates in the last $t-1$ iterations and $\sigma(p_{t-1}(e^+|\upsilon_{ij}))$ is its standard deviation at time $t-1$. The standardised time is calculated in Equation 6.3.

$$\tau = \frac{t - \overline{t}}{\sigma(t)} \tag{6.3}$$

The correlation coefficient r is equal to the average of the product of the standardized success rate and time calculated as follows:

$$\begin{aligned} r &= \overline{\mathcal{P}(e^+|\upsilon_{ij}) \cdot \tau} \\ &= \frac{\tau_1\mathcal{P}_1(e^+|\upsilon_{ij}) + \tau_2\mathcal{P}_2(e^+|\upsilon_{ij}) + ... + \tau_{t-1}\mathcal{P}_{t-1}(e^+|\upsilon_{ij})}{t-1} \end{aligned} \tag{6.4}$$

where $t-1$ is the total number of observations. The linear model for predicting $\hat{p}_t(e^+|\upsilon_{ij})$ from t is equal to:

$$\frac{\hat{p}_t(e^+|\upsilon_{ij}) - \overline{p}_t(e^+|\upsilon_{ij})}{\sigma p_t(e^+|\upsilon_{ij})} = r \cdot \frac{t - \overline{t}}{\sigma t} \tag{6.5}$$

This equation is usually expressed with constant terms as follows:

$$\hat{p}_t(e^+|\upsilon_{ij}) = a + bt \tag{6.6}$$

where:

$$b = r\frac{\sigma p_t(e^+|\upsilon_{ij})}{\sigma t} \tag{6.7}$$

$$a = \overline{p}_t(e^+|\upsilon_{ij}) - b(\overline{t}) \tag{6.8}$$

where a and b are the two coefficients of the linear equation used for predicting the next success rates. They depend only on the mean and standard deviations of $p(e^+|\upsilon_{ij})$ and t, and the correlation coefficient of the two variables. The best fit of the model is calculated by minimising the mean squared error ($\overline{\epsilon^2}$) calculated in Equation 6.9.

$$\overline{\epsilon^2} = \frac{1}{t}\sum_{s=1}^{t}(p_s(e^+|\upsilon_{ij}) - \hat{p}_s(e^+|\upsilon_{ij})) \tag{6.9}$$

6.2.2 Simple Moving Average

Due to the stochastic nature of approximate optimisation methods, the performance of the algorithm can display random variation with time. Therefore, a smoothing method is used to cancel, or at least reduce, the effect due to random variation. Averaging and Exponential Smoothing Models, such as the Simple Moving Average (SMA) model reduce the noise in the data. These models make an assumption on the local stationarity of the time series and expect a slow-varying mean. This is because these models calculate a moving, i.e. local, average, which smoothes out the spikes in the original time series and use this average to predict the next value of the time series. As a result, a smoothed model of the original time series is produced.

A simple moving average model uses equal weights for all the data points in the time series as follows:

$$\hat{p}_t(e^+|\upsilon_{ij}) = \frac{p_{t-1}(e^+|\upsilon_{ij}) + p_{t-2}(e^+|\upsilon_{ij}) + ... + p_{t-k}(e^+|\upsilon_{ij})}{k} \tag{6.10}$$

where k is the 'width' of the moving average and denotes the number of observations used to compute the average. The forecast $\hat{p}_t(e^+|\upsilon_{ij})$ is equal to the average of the last k data points in the time series, and is centred at $t - \frac{k+1}{2}$. This means that the forecast will tend to lag behind turning points in the data by $\frac{k+1}{2}$ time steps.

For example, if the size of the window is equal to 3, the forecasts will take 3 time steps to approach the real observations.

The value of k has to be adjusted in order to fit the model to the time series. If k is close to 1, the model is equivalent to a random walk. If k is sufficiently large, the model becomes equivalent to the mean model, which considers the mean of all data point in the time series. Similar to linear regression, the best fit of the model (i.e. appropriate value for k) is obtained by minimizing the forecast errors on average.

6.2.3 Exponentially-Weighted Moving Average

In the linear regression and simple moving average models described in the previous sections, the past observations are weighted equally. Intuitively, older data should have less weight than recent.

Exponentially-Weighted Moving Average is a particular type of smoothing technique which gives exponentially decreasing weights to past observations, i.e. feedback taken from the current iteration is given more importance in forecasting the next parameter values than the earlier observations. The forecast of the future effect of parameter values is derived from the weighted sum of all observations.

Mathematically, the Exponentially-Weighted Moving Average is the convolution of the data points with an exponentially smoothing weighting function defined as:

$$\begin{aligned}\hat{p}_t(e^+|v_{ij}) =& \alpha p_{t-1}(e^+|v_{ij}) + \alpha(1-\alpha)p_{t-2}(e^+|v_{ij}) \\ &+\alpha(1-\alpha)^2 p_{t-3}(e^+|v_{ij}) + ... + \alpha(1-\alpha)^{k+1} p_{t-(k+1)}(e^+|v_{ij})\end{aligned} \tag{6.11}$$

where α is the smoothing factor in the range (0,1). From Equation 6.11, it can be observed that the exponents of the data series decrease exponentially, approaching zero but never reaching it. The smoothing factor α controls the closeness of the

interpolated value to the most recent observation.

Large values of α reduce the level of smoothing. To determine the value of α we use the least-squares method, for which the mean of squared errors, given in Equation 6.9, is minimised. Alternatively, α may be expressed in terms of t time steps, calculated as:

$$\alpha = \frac{2}{t+1} \tag{6.12}$$

For example, if we have 19 data points, α will be equal to 0.1. The half-life of the weights (the interval over which the weights decrease by a factor of two) is approximately $N/2.8854$ (within 1% if $N > 5$).

6.2.4 Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) models are the general class of models for forecasting time series. The moving average models described in the previous sections are all special cases of the ARIMA models. In general ARIMA models are appropriate when the time series data shows a stable trend with time and does not have many outliers. ARIMA can perform better than exponential smoothing techniques, however it requires a certain number of observations, and the correlation between past observations should be stable. The ARIMA model is composed of three parts, which are calculated individually: the Integrated (I) part, the Autoregressive part (AR) and the Moving Average part (MA).

The integrated part $I(d)$ is used if the data shows evidence of non-stationarity, i.e. the time series is not stable with time and exhibits some trend, and d is the number of non-seasonal differences. The stationarity of the time series is measured using the stationarity in mean and variance. The first step in the identification of an appropriate ARIMA model starts by identifying the value for d, i.e. the order(s)

of differencing needed to make the time series stationary, which can be one of the followings:

- If there is a constant trend, i.e. zero mean, $d = 0$.
- For a linear trend, i.e. linear growth behaviour, $d = 1$.
- Otherwise, if the trend is quadratic, i.e. quadratic growth behaviour, $d = 2$.

The process of differencing subtracts each observation at time t from the the observation at time $t - 1$. For $d = 0$, this transformation is performed only once to the time series, which is also referred to as 'first differencing'. As a result, the trend of a linear growth behaviour is eliminated. At this point, we have fitted a random walk model, which predicts the first difference of the time series to be constant. When this process is repeated, the time series is 'second differenced' and so on.

The $AR(p)$ model is used to describe the correlation between the current value of the time series and p past values. The order p of the AR model describes the number of lags (past values) included in the model prediction, i.e. the number of the autoregressive terms. For instance, when applied to our application, $AR(1)$ means that the prediction of the success rate $\hat{p}_t(e^+|\upsilon_{ij})$ at time t is correlated with the immediately preceding value, i.e. $p_{t-1}(e^+|\upsilon_{ij})$ in the time series. This can be written as:

$$\hat{p}_t(e^+|\upsilon_{ij}) = \phi_1 p_{t-1}(e^+|\upsilon_{ij}) + \epsilon_t \tag{6.13}$$

where ϕ_1 is the first-lag (lag 1) autoregressive parameter, and ϵ_t is the error, which is assumed to be random and normally distributed. If the estimated value of parameter ϕ_1 is 0.25, the current value of the time series is related to 25% of its value 1 time-step ago. The current success rate $p_t(e^+|\upsilon_{ij})$ in the time series can be related to more than one past value. For instance:

$$\hat{p}_t(e^+|\upsilon_{ij}) = \phi_1 p_{t-1}(e^+|\upsilon_{ij}) + \phi_2 p_{t-2}(e^+|\upsilon_{ij}) + \epsilon_t \tag{6.14}$$

In this case, the success rate of the parameter value υ_{ij} at time t is related to two immediately past success rates, $p_{t-1}(e^+|\upsilon_{ij})$ and $p_{t-2}(e^+|\upsilon_{ij})$, by considering also the random error ϵ_t.

The Moving Average ($MA(q)$) part of the ARIMA model considers the time series as an unevenly weighted moving average of random errors ϵ_t, capturing the deviation of the current success rate as a finite weighted sum of q previous deviations of the success rates, plus a random error, i.e. the current success rate of parameter values is related only to the errors that happened in the past lags, rather than to the preceding success rates as in the AR model. As an example, the first order $MA(1)$ means that the error of the current value is directly correlated with the error of the immediate past observation given by Equation 6.15.

$$\hat{p}_t(e^+|\upsilon_{ij}) = \epsilon_t - \theta_1 \epsilon_{t-1} \tag{6.15}$$

where ϵ_t, ϵ_{t-1} are the errors (or residuals) at times t and $t-1$, and θ_1 is the first order moving average coefficient. Similar to the AR model, the MA model can be extended to a higher order model in order to include more than just the error of the immediate past observation.

6.3 Characteristics of the Parameter Effects

Each of the forecasting models described in Section 6.2 makes assumptions about the data, such as linearity and normality of the distribution of errors. In this section, we check whether the assumptions about the time series data hold to make a judgement of the suitability of the models. The time series represents the effect of parameter values during the optimisation process. The effect of the parameter values is calculated using Equation 5.3. The data examination is performed by employing statistical tests.

There are four principal assumptions which justify the use of linear regression models for purposes of prediction: *linearity of the relationship between dependent and independent variables*, *normality of the error distribution*, *independence of the errors* (no serial correlation), and *homoscedasticity*, which refers to the constant variance independent of time, of the errors (i) versus time and (ii) versus the predictions.

Simple Moving Average and Exponentially-Weighted Moving Average are not sensitive to normality assumptions. However, in order for SMA and EWMA to be appropriate models, two assumptions are made. First, both EWMA and SMA assume that the time series is *stationary*, which means that the mean and standard deviation of the time series do not change with time. Furthermore, both models assume that the variance around the mean remains constant with time, i.e. the model assumes *homoscedasticity* of the time series.

In order to achieve best results using the ARIMA approach, five assumptions have to be met. The first is the generally accepted threshold of 50 data points. This is a significant obstacle for the parameter control method, since it implies that the model will start giving appropriate results only after 50 iterations. The other four assumptions of ARIMA models are *homoscedasticity*, *stationarity*, *independence* and *normality* of residuals.

6.3.1 Experimental Settings

For the purpose of this analysis we test six parameters: mutation rate, crossover rate, mutation operator, crossover operator, population size and mating pool size. For the mutation rate, crossover rate, population size and mating pool size we perform the experiments twice: with a cardinality of two ranges and with a cardinally of four ranges with an even spread within a predefined interval. The aim of using different cardinalities is to investigate if using ranges with different size results in different statistical properties.

Table 6.1: Ranges/values of parameters.

Parameter	**Ranges/values**
Mutation rate - 2 ranges	[0.001,0.249], [0.25,0.49]
Mutation rate - 4 ranges	[0.001,0.1249], [0.125,0.249], [0.25,0.3749], [0.375,0.49]
Crossover rate - 2 ranges	[0.6,0.79], [0.8,0.99]
Crossover rate - 4 ranges	[0.6,0.69], [0.7,0.79], [0.8,0.89], [0.9,0.99]
Population size - 2 ranges	[20,60], [61,100]
Population size - 4 ranges	[20,40], [41,60], [61,80], [81,100]
Mating pool size - 2 ranges	[0.1,0.39], [0.4,0.69]
Mating pool size - 4 ranges	[0.1,0.249], [0.25,0.39], [0.4,0.549], [0.55,0.69]
Mutation operator	Single-point, Uniform
Crossover operator	Single-point, Uniform

To investigate the characteristics of the data, we run ten different experiments. In every experiment, one of the parameters is the design factor and the other parameters are kept constant. In each experiment, every parameter range/value is applied to 10 different instances which run in parallel. Every instance is granted 15000 function evaluations. Every 150 function evaluations the effect of the parameter ranges, i.e. the success rate approximated by Equation 5.3, is recorded. In total, each experiment produces 100 data points for each parameter range.

Approximate algorithms are not expected to deliver exact and repeatable results.

Therefore, for the current data analysis, all algorithms trials were repeated 30 times for each parameter value.

6.3.2 Statistical Analysis

Linearity

In order to justify the use of the linear regression model described in Section 6.2.1, we test the success rates of parameter values (calculated using Equation 5.3) for linearity. Linearity means that the relationship between the success rates and time is a straight line. To check for a statistical significance of the linearity of the data, we run the optimisation algorithms 30 times for each case. After fitting a linear model to the data, we report the p-value which indicates the significance of the regression. A $p - \text{value} < 0.05$ implies that the time series is linear with all data points having equal influence on the fitted line. The hypothesis is formulated as follows:

- H_0: The time series is linear.
- H_1: The time series is not linear.

Normality

The calculation of the confidence intervals for the model coefficients in linear regression is based on the assumptions of normally distributed errors. In Predictive Quality Attribution, normality of errors means that most of the success rates of parameter values are expected to fall close to the regression line and very few to fall far from the line. Since the estimation of model coefficients is based on the minimisation of squared error, the presence of extreme data points can exert a disproportionate influence on the estimates. For example, if the data contains large outliers, the error distribution becomes skewed.

To check for the normality assumption, we use the Kolmogorov-Smirnov (KS)

test [64] to tests the null hypothesis that the errors of the success rates are normally distributed. The null hypothesis is rejected if the p-value is smaller that 0.05, which means that the data is not normally distributed. The hypothesis is formulated as follows:

- H_0: The data follows a normal distribution
- H_1: The data does not follow a normal distribution

Independence

Linear regression models make the assumption that the error terms are independent. One example of the violation of this assumption is autocorrelation, which occurs if each error term in the time series is related to its immediate predecessor (ϵ_i is related to ϵ_{i-1}) defined as follows:

$$\epsilon_t = \rho\epsilon_{t-1} + \mu_t \tag{6.16}$$

where ρ is the coefficient of the first order autocorrelation defined in the interval $[-1, 1]$, and the constant μ_t values are assumed to be independent. If ρ is equal to zero, the errors are not correlated. The Durbin-Watson (DW) statistical test, measures the correlation between the error terms and their immediate predecessors as follows:

$$D = \frac{\sum_{i=1}^{t}(\epsilon_i - \epsilon_{i-1})^2}{\sum_{i=1}^{t}\epsilon_i^2} \tag{6.17}$$

The D statistic tests the null hypothesis that the errors from a least-squares regression are not autocorrelated. The values of D vary in the interval $[0, 4]$. For independent error terms, the value of D is expected to be close to 2. The value of D tends to be smaller than 2 for positive autocorrelation between the terms, and greater than 2 for negative autocorrelation.

- H_0: The the autocorrelation of the errors is 0.
- H_1: The autocorrelation of the errors is less than 0.

The null hypothesis is rejected if p-value is less than 0.05.

Homoscedasticity

Homoscedasticity refers to the assumption that the success rates of parameter values exhibit a similar variance with time. If homoscedasticity is violated, it is difficult to measure the true standard deviation of the residuals, which can result in confidence intervals that are too wide or too narrow. In particular, if the variance of the errors increases with time, confidence intervals for predictions will tend to be unrealistically narrow. Heteroscedasticity may also have the effect of placing too much weight on a small subset of the data, i.e. the subset where the error variance is the largest when estimating coefficients.

Residuals are tested for homoscedasticity using the Breusch-Pagan test, which regresses square residuals to independent variables. The null hypothesis for the Breusch-Pagan test is homosedasticity. The alternate hypothesis is that the error variance varies with a set of regressors.

- H_0: The data exhibit a similar variance with time.
- H_1: The error variance varies with a set of regressors.

If the p-value is less than 0.05, the null hypothesis is rejected, which means that the data is not homoscedastic.

Stationarity

A stationary time series fluctuates around a constant long-term mean and has a constant standard deviation independent of time. In the application of the Predictive Quality Attribution, this means that it does not matter when in time the success

rates are observed. If the time series is non-stationary, it can be transformed into stationary by differencing. To check for the stationarity of the success rates of parameter values we use the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) [108] test, which tests the null hypothesis that the data is stationary, formulated as follows:

- H_0: The data is stationary.
- H_1: The data is not stationary.

If p-value is less than 0.05, the null hypothesis is rejected, which means that the data is not stationary.

6.3.3 Results

The time series data from the optimisation process was tested against the assumptions of linearity, normality, independence, homoscedasticity, and stationarity using the statistical tests described in the precious sections. The aim was to investigate if it is conform these assumptions, which indicates the adequacy of using the forecasting techniques (LR, SMA, EWMA and ARIMA) to model and predict the effect of parameter values in the future iterations. The summary of the assumptions that each of the considered forecasting models makes about the data is sumarised in Table 6.2.

Table 6.2: Assumptions of the forecasting models.

	Linearity	Normality	Independence	Homoscedasticity	Stationarity
LR	+	+	+	+	-
SMA	-	-	-	+	+
EWMA	-	-	-	+	+
ARIMA	-	+	+	+	+

Table 6.3 summarises the results from the statistical tests. The characteristics of the success rates of parameter values (linearity, normality, etc.) are listed based on the assumptions of the forecasting models described in Table 6.2. The percentages represent the runs among the 30 trials in which H_0 was not rejected. We use these results to derive conclusions regarding the appropriateness of using forecasting techniques to predict the success rates of the parameter values.

The detailed results are presented in the Appendix. The linearity test results are shown in Tables 10.1-10.12 for the experiment with two ranges/values and in Tables 10.61-10.76 for the experiment with four ranges. The results from the Kolmogorov-Smirnov (KS) test are shown in Tables 10.13-10.24 and Tables 10.77-10.92. The results of the Durbin-Watson (DW) statistical test are shown in Ta-

bles 10.25-10.36 and Tables 10.93-10.108. The results from the Breusch-Pagan (BP) test are shown in Tables 10.37-10.48 and Tables 10.109-10.124. Finally, the results of the KwiatkowskiPhillipsSchmidtShin test, are shown in Tables 10.49-10.60 and Tables 10.125-10.140.

Table 6.3: Characteristics of the success rates of six algorithm parameters with different values or ranges. The percentages represent the confirmation of H_0 among the 30 trials.

Parameter	**Linear**	**Normal**	**Independent**	**Homoscedastic**	**Stationary**
Mutation rate [0.0010,0.25]	97%	100%	80%	90%	97%
Mutation rate [0.2505,0.5]	100%	100%	70%	97%	100%
Mutation rate [0.001,0.124]	97%	97%	97%	90%	80%
Mutation rate [0.125,0.249]	100%	100%	100%	80%	63%
Mutation rate [0.25,0.3749]	100%	100%	97%	90%	87%
Mutation rate [0.375,0.49]	100%	90%	97%	87%	80%
Crossover rate [0.6,0.799]	100%	100%	83%	90%	93%
Crossover rate [0.8,1.0]	100%	100%	83%	97%	93%
Crossover rate [0.6,0.69]	100%	100%	93%	83%	80%
Crossover rate [0.7,0.79]	100%	100%	97%	90%	73%
Crossover rate [0.8,0.89]	100%	100%	97%	87%	80%
Crossover rate [0.9,0.99]	100%	100%	100%	83%	93%
Population size [20,60]	100%	100%	100%	**53**%	**13**%
Population size [61,100]	**33**%	73%	100%	77%	**13**%
Population size [20,40]	100%	97%	100%	**40**%	**10**%
Population size [41,60]	100%	100%	100%	**40**%	**3**%
Population size [61,80]	100%	100%	100%	**57**%	**17**%
Population size [81,100]	100%	100%	100%	**53**%	**47**%
Mating pool [0.1,0.4]	100%	100%	77%	93%	97%
Mating pool [0.41,0.7]	100%	100%	80%	87%	100%
Mating pool [0.1,0.249]	100%	100%	90%	80%	63%
Mating pool [0.25,0.39]	100%	100%	97%	83%	97%
Mating pool [0.4,0.549]	100%	100%	100%	80%	70%
Mating pool [0.55,0.69]	100%	100%	97%	80%	77%
Single-point mutation	100%	100%	70%	93%	97%
Uniform mutation	100%	100%	77%	93%	93%
Single-point crossover	100%	100%	77%	87%	93%
Uniform crossover	100%	100%	77%	97%	93%

In general, the time series data is linear with normally distributed errors. The lowest number of normally distributed residuals resulted from the experiment with the population size in the range [61,100], in which the null hypothesis was rejected in 27% of the cases. The population size in this range is also the only parameter range that was not linear in the majority of the cases (the null hypothesis was rejected in 77% of the runs). It follows that the use of linear regression models to forecast the success rates of the population size in this range is not appropriate.

We employed the Durbin-Watson (DW) statistical test to verify the assumption of no correlation between the error terms of the time series (independence of the errors). The error terms of the success rates of parameter values are independent for the majority of the cases, with the lowest percentage for the mutation rate in the ranges [0.2505,0.5] (70% of the runs have independent errors). Nevertheless, this does not limit the applicability of the Linear Regression and ARIMA models to forecast the effect of this parameter range.

The Breusch-Pagan (BP) test was used to measure the homosedasticity of the time series data. All forecasting methods require that the time series data is homoscedastic. In general, this assumption is true for the majority of the time series data, i.e. the success rates (effects of parameter values) are highly homoscedastic for the majority of the parameter values/ranges. The lowest percentages of homoscedastic runs are in the experiments with the ranges [20,40] and [41,60] of the population size, in which only 40% of the runs are homoscedastic (see column headlined 'Homoscedasticity' in Table 6.3). Similar results were observed in the other ranges of the population size, which indicate that forecasting the effect of the population size may not be appropriate.

The Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test verifies the null hypothesis that the data is stationary. The assumption regarding the stationarity of the time-series data is made by three of the forecasting models shown in Table 6.2: SMA,

EWMA and ARIMA. It can be observed that stationarity is a common feature in the success rates (effects) of the single-point mutation, uniform mutation, single-point crossover and uniform crossover operators. Similarly, in some of the ranges of the mutation rate ([0.0010,0.25] and [0.2505,0.5]), crossover rate ([0.6,0.799], [0.8,1.0] and [0.9,0.99]) and mating pool size ([0.1,0.4], [0.41,0.7] and [0.25,0.39]) the null hypothesis was not rejected in more than 90% of the runs with the mating pool size in the range [0.41,0.7] and mutation rate in the range [0.2505,0.5] being the most stationary processes (100% of the runs are stationary).

However, the crossover rate in the range [0.7,0.79], the mutation rate in the range [0.125,0.249] and the mating pool size in the ranges [0.1,0.249], [0.4,0.549] and [0.55,0.69] have a moderately low number of stationary runs. Furthermore, less than 50% of the runs in all ranges of the population size are stationary. Population size in the range [41,60] has the lowest percentage of stationary runs, with only 3% of the runs being stationary. These results indicate that stationarity, which is a prerequisite of SMA, EWMA and ARIMA, does not fit perfectly the time series data we investigated. The time series can be transformed into stationary by differencing, however that would require extra computations. Instead, Linear Regression does not make any assumption about the stationarity of the time series (see Table 6.2), hence may be a better choice than SMA, EWMA and ARIMA to predict the nonstationary effect of the parameter values.

Table 6.4 shows recommendations on which models to use for forecasting the success rates of various parameter values. The '+' sign means that the time series data of the parameter value follows all assumptions made by the forecasting model, hence the model can be used to predict the future effect of that parameter value. On the other hand, the '$\sim$' sign indicates that the time series data of the parameter value partially follows the assumptions made by the forecasting model, hence the use of the forecasting model to predict its effect is discouraged.

Table 6.4: Recommendation for using forecasting models to predict the success rates of different parameter values chosen from different ranges.

Parameter	**LR**	**SMA**	**EWMA**	**ARIMA**
Mutation rate [0.0010,0.25]	+	+	+	+
Mutation rate [0.2505,0.5]	+	+	+	+
Mutation rate [0.001,0.1249]	+	+	+	+
Mutation rate [0.125,0.249]	+	+	+	+
Mutation rate [0.25,0.3749]	+	+	+	+
Mutation rate [0.375,0.49]	+	+	+	+
Crossover rate [0.6,0.799]	+	+	+	+
Crossover rate [0.8,1.0]	+	+	+	+
Crossover rate [0.6,0.69]	+	+	+	+
Crossover rate [0.7,0.79]	+	+	+	+
Crossover rate [0.8,0.89]	+	+	+	+
Crossover rate [0.9,0.99]	+	+	+	+
Population size [20,60]	∼	∼	∼	∼
Population size [61,100]	∼	∼	∼	∼
Population size [20,40]	∼	∼	∼	∼
Population size [41,60]	∼	∼	∼	∼
Population size [61,80]	∼	∼	∼	∼
Population size [81,100]	∼	∼	∼	∼
Mating pool size [10,40]	+	+	+	+
Mating pool size [41,70]	+	+	+	+
Mating pool size [0.1,0.249]	+	+	+	+
Mating pool size [0.25,0.39]	+	+	+	+
Mating pool size [0.4,0.549]	+	+	+	+
Mating pool size [0.55,0.69]	+	+	+	+
Single-point mutation	+	+	+	+
Uniform mutation	+	+	+	+
Single-point crossover	+	+	+	+
Uniform crossover	+	+	+	+

It can be observed that any of the forecasting techniques can be used to predict the success rates of mutation rate, crossover rate, mating pool size, mutation operator and crossover operator. However, we found that the success rates of the population size do not meet the assumptions of the forecasting techniques that we investigated. As a result, none of these forecasting techniques are recommended for

use with this parameter. In the next sections we investigate if forecasting the effect of the population size has a negative effect on the performance of the Evolutionary Algorithm.

6.4 Analysis of the Predictive Quality Attribution Methods

In the previous section, we investigated various characteristics of the effects (success rates) of parameter values, such as linearity, normality of the residuals, and stationarity. The statistical tests showed that the time series data of all EA parameters, except the population size, conform to the assumptions made by the forecasting models. Recommendations on appropriate forecasting techniques for various parameter values were presented in Table 6.4.

In this section, we analyse the parameter quality attribution strategy using different forecasting techniques - Linear Regression (LR), Simple Moving Average (SMA), Exponentially Weighted Moving Average (EWMA) and Autoregressive Integrated Moving Average (ARIMA) - to predict the success rates of parameter values. The aim is to understand which of the forecasting models is the best choice for the parameter quality attribution strategy.

6.4.1 Experimental Settings

For the purpose of this experiment we consider five parameters: the mutation rate, crossover rate, mating pool size, crossover operator, and mutation operator. The parameter values or ranges used in this experiment are similar to the ones used in the previous section, given in Table 6.1. Population size is not controlled based on the outcome of the statistical tests, which showed that the effects of this parameter do not conform to the assumptions made by the forecasting models.

We perform 30 runs for each of the optimisation schemes using different parameter quality attribution strategies and record the results for 9000 function evaluations. The results are presented as the mean of 30 runs over 55 iteration.

We employ the optimisation schemes to solve the Quadratic Assignment Problem (BUR26E), the multiobjective Quadratic Assignment Problem (KC30-3fl-2rl) and the Royal Road Problem. The problem instances are downloaded from the online repository of benchmark problems at `http://www.seas.upenn.edu/qaplib/`. Different QAP instances have different scales of the fitness values. To make the results comparable, we normalise the fitness function as follows:

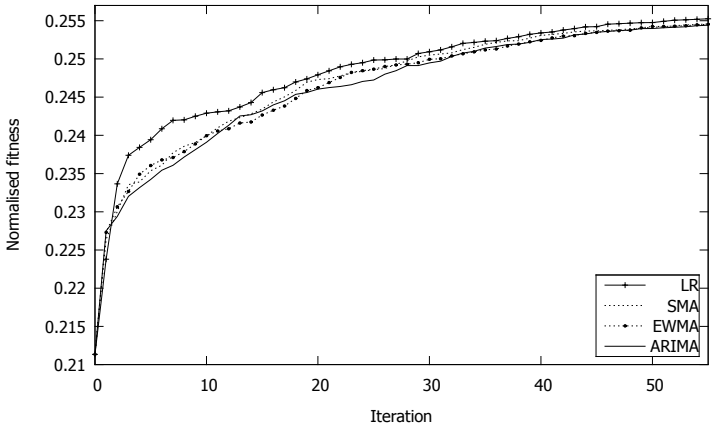
$$f_{norm}(s) = \frac{f_{max} - f(s)}{f_{max} - f_{min}} \tag{6.18}$$

where f_{max} is the maximum value and f_{min} is the minimum value that the fitness function f can take. Since QAP is a minimisation problem, the normalisation of the results converts it into a maximisation problem. Hence the normalised fitness of the results increases over time.

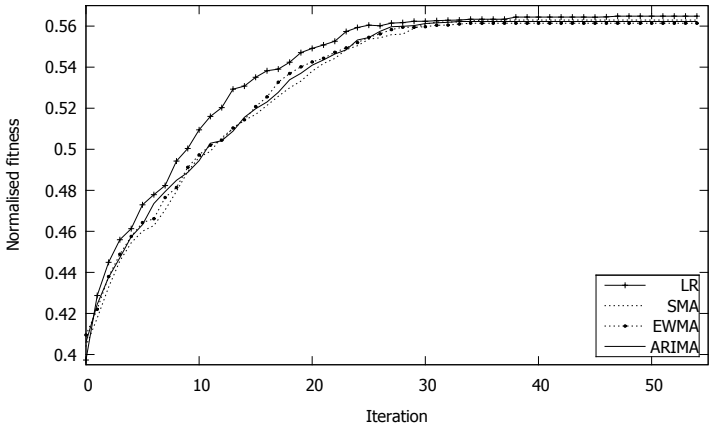
6.4.2 Results

The mean hypervolume growths of the 30 runs of the Evolutionary Algorithm using different forecasting techniques as parameter quality attribution strategy are presented in Figure 6.1. In the optimisation of QAP, mQAP and the Royal Road problem, the Evolutionary Algorithm using Linear Regression (LR) as a quality attribution strategy produced better results quality than the three other forecasting methods, especially in the initial iterations. The difference between the performances of the parameter quality attribution strategies becomes smaller as the optimisation progresses. However, Linear Regression consistently outperforms SMA, EWMA and ARIMA in all iterations.

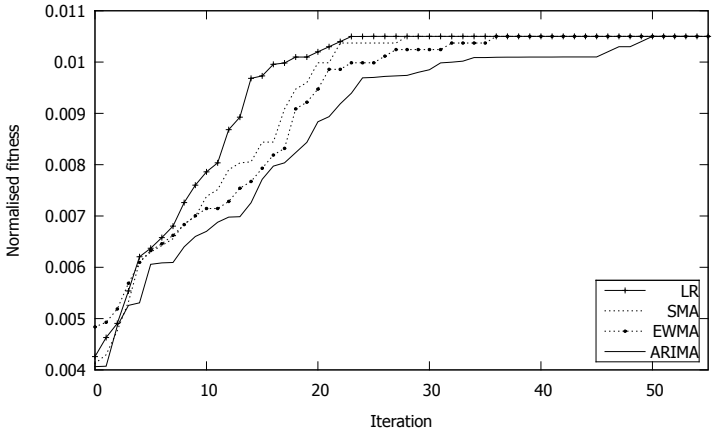
The optimisation scheme using Autoregressive Moving Average (ARIMA) as a quality attribution scheme does not perform very well in the initial iterations, especially in the experiments with the Royal Road problem. This is an inherent



(a) BUR26E



(b) KC30-3fl-2rl



(c) Royal Road

Figure 6.1: Performance improvement of the EA using different forecasting techniques to solve the Quadratic Assignment Problem BUR26E, the multiobjective Quadratic Assignment Problem KC30-3fl-2rl, and the Royal Road problem.

problem of ARIMA, which requires more data points than the other forecasting techniques to forecast the next values accurately. The parameter quality attribution strategy using ARIMA archives good results towards the end of the optimisation process.

In summary, the optimisation scheme using Linear Regression (LR) to predict the effect of the parameter values performed better than SMA, EWMA and ARIMA. Linear Regression is one of the simplest forecasting models, yet it is a powerful model, provided that the data conforms to the assumptions that this model makes. All parameter ranges that we used in this experiment meet the assumptions made by the Linear Regression.

On the other hand, certain ranges of mutation rate, crossover rate and mating pool size meet the stationarity assumption, which is a prerequisite of SMA, EWMA and ARIMA (see Table 6.3), only in a moderate number of runs. As a result, these models may not fit the time series data we are controlling perfectly. This explains the poor performance of SMA, EWMA and ARIMA models when compared with Linear Regression. Hence, in the rest of the thesis, we use Linear Regression to predict the effect of EA parameter values/ranges.

6.5 Validation

The Predictive Quality Attribution (PQA) strategy approximates the quality of parameter values by combining past performance with time series prediction. PQA records the success rates of every parameter value for each iteration and uses a forecasting technique to predict the success rate for the next iteration. The research question addressed in this section is formulated as follows:

* **What is an effective parameter quality attribution method for projecting successful parameter values to the next iteration of the optimisation process?**

To answer this research question we investigate if using the Predictive Quality Attribution is more beneficial than using the average or extreme quality attribution schemes.

6.5.1 Benchmark Parameter Quality Attribution Strategies

We compare the proposed parameter quality attribution strategy to two notable quality attribution strategies. The first method is the Average Quality Attribution (AQA), proposed by Igel et al. [85], which uses time window (adaptation cycle) W to approximate the quality of parameter values as the average effect on algorithm performance. The Average Quality Attribution (AQA) is given in Equation 6.19.

$$q_{t-1}(\upsilon_{ij}) = \frac{1}{N_W^{\upsilon_{ij}}(S)} \sum_{s=t-W-1}^{t-1} e_s(\upsilon_{ij}) \tag{6.19}$$

where $e_s(\upsilon_{ij})$ is the effect of parameter value υ_{ij}, assessed as the fitness difference between the solution produced by applying parameter value υ_{ij} and its parent. The sum of the effect values of υ_{ij} in the last W iterations (i.e. $\sum_{s=t-W-1}^{t-1} e_s(\upsilon_{ij})$) is divided by the total number of solutions created using that parameter value ($N_W^{\upsilon_{ij}}(S)$)

normalising the quality of υ_{ij}.

The second approach is the Extreme value-based Quality Attribution (EQA) strategy, proposed by Fialho et al. [55, 54, 56]. EQA employs the maximum effect in W iterations to approximate the quality of parameter values in the preceding iteration $t-1$ as follows:

$$q_{t-1}(\upsilon_{ij}) = \arg\max_{s=t-W-1,\ldots,t} \frac{e_s(\upsilon_{ij})}{e(\upsilon_{i*})} \tag{6.20}$$

where the effect $e_s(\upsilon_{ij})$ of parameter value υ_{ij} at each time-step s is measured as the difference in fitness of the offspring created by using parameter value υ_{ij} and its parent(s). The extreme value-based quality attribution strategy uses a normalisation scheme, which divides parameter effects by the best effect $e(\upsilon_{i*})$ achieved so far by any parameter value.

6.5.2 Experimental Settings

The Predictive Quality Attribution (PQA), Average Quality Attribution (AQA) and Extreme Quality Attribution (EQA) strategies are used to control parameters of an Evolutionary Algorithm, which is employed to optimise the Royal Road Problem, described in Section 3.3.2, four instances of the Quadratic Assignment Problem (CHR20A, BUR26E, TAI30A, and STE36B), described in Section 3.3.2 and three instances of the multiobjective Quadratic Assignment Problem (KC10-2fl-5rl, KC30-3fl-1rl, and KC30-3fl-2rl), described in Section 3.3.2.

For the benefit of this experiment we control the mutation rate, the crossover rate, the mutation operator, the crossover operator, the population size and the mating pool size with values/ranges depicted in Table 6.1.

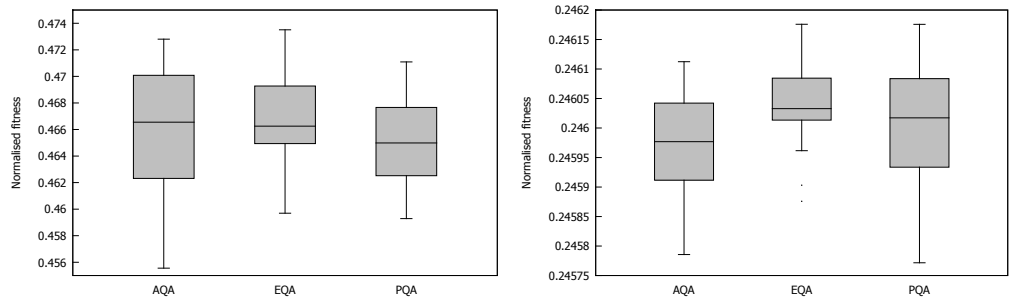
The experimental study in Section 6.3 demonstrated that the effect of the population size during the optimisation time-steps does not adhere the assumptions

made by the forecasting techniques. Hence, we conduct two sets of experiment: in the first experiment we control only the population size, whereas in the second experiment we control the mutation rate, the crossover rate, the mutation operator, the crossover operator, and the mating pool size. The aim of the first experiment is to understand if using PQA to approximate the effect of the population size has a negative effect on the performance of the algorithms, since this parameter does not meet the assumptions made by the forecasting models. The second set of experiments is performed to examine the performance of the PQA when used to predict the quality of parameter values which conform to the assumptions of the forecasting techniques.

Every instance is granted 90000 function evaluations. To obtain a fair comparison among the parameter quality attribution schemes, we repeat each run 30 times, and report the results as boxplots. To check for a statistical difference of the results, the parameter quality attribution schemes of the optimisation methods are validated using the Kolmogorov-Smirnov (KS) non-parametric test [149]. The final solutions of each optimisation scheme are recorded and the results are compared by using the normalised fitness for the singleobjective problems and the hypervolume indicator (described in Section 3.3.4) for the multiobjective problem.

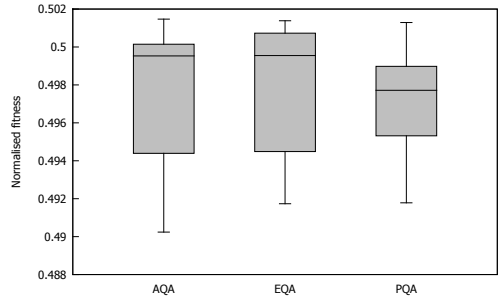
6.5.3 Results

In the first experiment, only the population size was controlled using three parameter quality attribution schemes: AQA, EQA, and PQA. Figure 6.2 depicts the results of the 30 runs for each problem instance and parameter quality attribution scheme. The Predictive Quality Attribution (PQA) is slightly outperformed by the Extreme Quality Attribution (EQA) strategy in some of the problems. These results indicate that using forecasting techniques to predict the quality of the population choices for

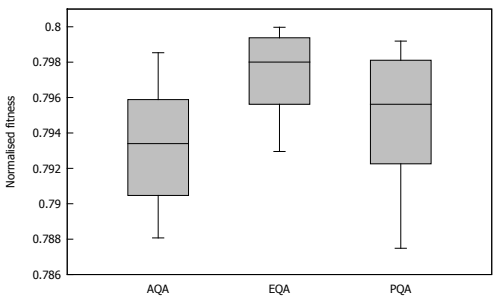


(a) CHR20A

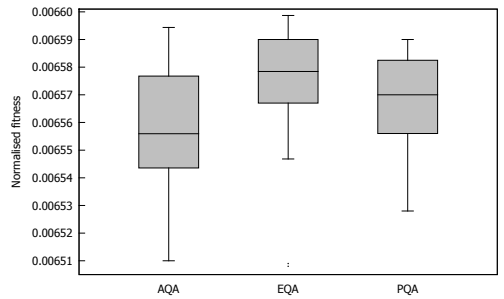
(b) BUR26E



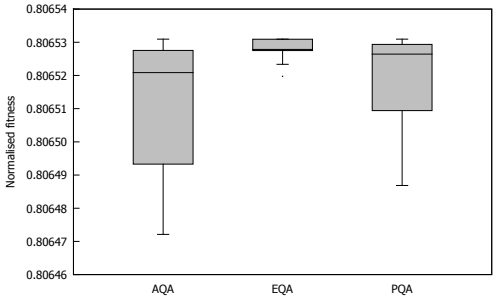
(c) TAI30B



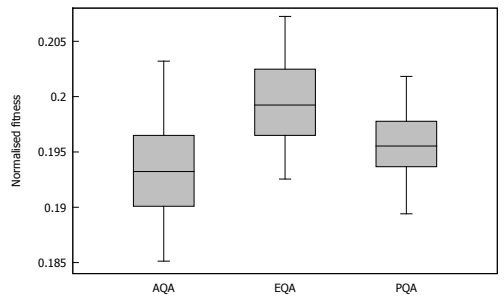
(d) STE36B



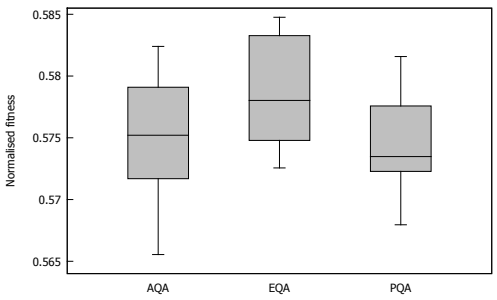
(e) Royal Road



(f) KC10-2fl-5rl



(g) KC30-3fl-1rl



(h) KC30-3fl-2rl

Figure 6.2: Performance improvement of the EA using different parameter quality attribution strategies for controlling population size to solve the QAP, the mQAP and the Royal Road problem

the future iterations may not always be beneficial.

Table 6.5: The means and standard deviations for the 30 runs of each problem instance using different parameter quality attribution schemes: Average Quality Attribution (AQA), Extreme Quality Attribution (EQA) and Predictive Quality Attribution (PQA).

	Mean			**Standard Deviation**		
Problem	AQA	EQA	PQA	AQA	EQA	PQA
CHR20A	0.4660	**0.4669**	0.4651	4.85E-03	3.51E-03	4.26E-03
BUR26E	0.2460	**0.2461**	0.2459	8.92E-05	6.88E-05	1.31E-04
TAI30A	0.4977	**0.4982**	0.4962	3.57E-03	3.23E-03	5.94E-03
STE36B	0.7934	**0.8164**	0.7945	3.29E-03	4.22E-02	3.64E-03
Royal Road	6.56E-03	**6.57E-03**	6.56E-03	2.14E-05	2.27E-05	1.82E-05
KC10-2fl-5rl	0.8065	0.8065	0.8065	2.74E-05	3.30E-06	2.74E-05
KC30-3fl-1rl	0.1936	**0.1993**	0.1974	5.68E-03	4.53E-03	6.74E-03
KC30-3fl-2rl	0.5754	**0.5785**	0.5745	4.92E-03	4.14E-03	3.67E-03

The means and standard deviations of the three parameter quality attribution schemes are shown in Table 6.5. It can be observed that the mean performance of the Predictive Quality Attribution (PQA) is usually not higher then the other two parameter quality attribution schemes. To check for a statistical significance of the difference in the performances of the parameter quality attribution schemes we submitted the results from the 30 runs to the Kolmogorov-Smirnov (KS) non-parametric test [149], with the null hypothesis that there is no significant difference between the result sets.

The null hypothesis was not rejected in any of the KS tests (see Table 6.6), which means that the superior performances of the Average Quality Attribution (AQA) and Extreme Quality Attribution (EQA) methods compared to the Predictive Quality Attribution (PQA) strategy are not statistically significant.

This indicates that using PQA to forecast the effect of teh population size does not have a negative effect in the performance of the algorithm, despite the fact that

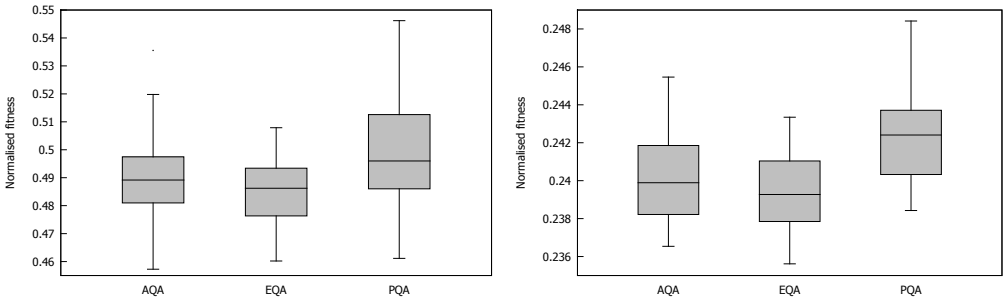
Table 6.6: The Kolmogorov-Smirnov test values for the 30 runs of each problem instance using different parameter quality attribution schemes: Average Quality Attribution (AQA), Extreme Quality Attribution (EQA) and Predictive Quality Attribution (PQA).

	PQA vs. AQA		**PQA vs. EQA**	
Problem	d	p	d	p
CHR20A	0.2000	0.537	0.2667	0.200
BUR26E	0.2000	0.537	0.3000	0.109
TAI30A	0.3750	0.162	0.3750	0.162
STE36B	0.2500	0.633	0.3750	0.162
Royal Road	0.2667	0.200	0.2667	0.200
KC10-2fl-5rl	0.3750	0.162	0.3750	0.162
KC30-3fl-1rl	0.3500	0.121	0.3542	0.113
KC30-3fl-2rl	0.2000	0.890	0.4000	0.136

this parameter does not meet the assumptions made by the forecasting models. As a result, we expect that when new EA parameters are controlled, prediction can be used without a negative effect in the performance of the algorithm.

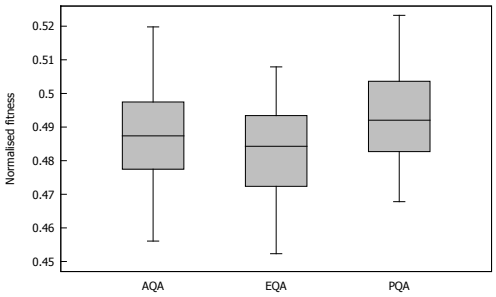
In the second experiment, the mutation rate, crossover rate, mating pool size, mutation operator and crossover operator were controlled. The 30 results of the repeated trials are presented as boxplots in Figure 6.3. The empirical results are not normally distributed, but the mean and 25th percentile of the Predictive Quality Attribution (PQA) method are consistently above the respective values of the benchmarks.

The means and standard deviations are listed in Table 6.7, which show a significant difference between the result groups of AQA, EQA and PQA. The mean performance of PQA is consistently above the averages of the benchmarks. The performance of EQA is worse than the other two parameter quality attribution strategies, which is quite the opposite from what the results from the experiment with the population size showed.

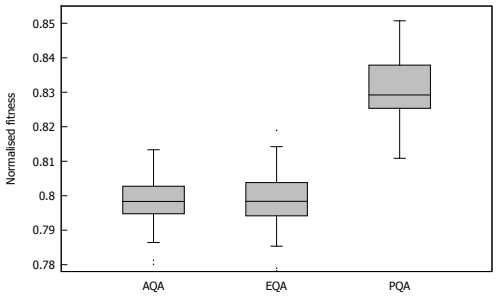


(a) CHR20A

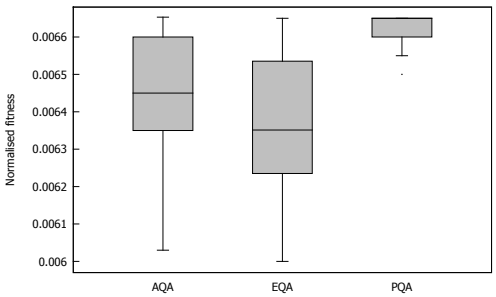
(b) BUR26E



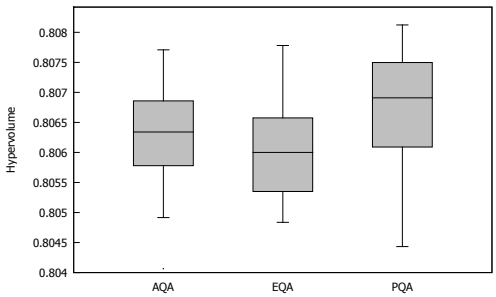
(c) TAI30A



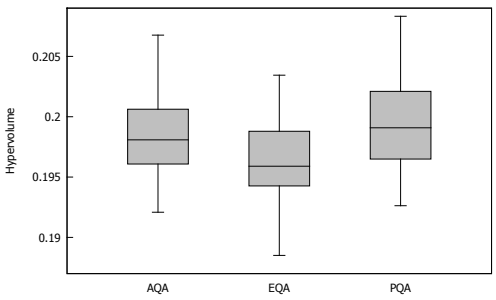
(d) STE36B



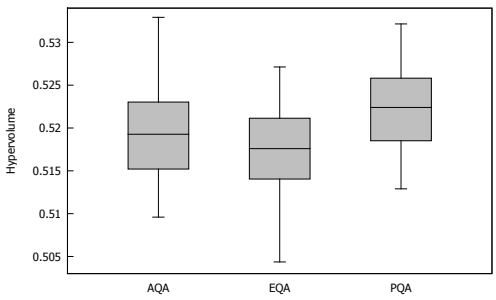
(e) Royal Road



(f) KC10-2fl-5rl



(g) KC30-3fl-1rl



(h) KC30-3fl-2rl

Figure 6.3: Boxplots of the 30 trials of the three different parameter quality attribution schemes used with an Evolutionary Algorithm optimising the QAP, the mQAP and the Royal Road problem.

Table 6.7: The means and standard deviations for the 30 runs of each problem instance using different parameter quality attribution schemes: Average Quality Attribution (AQA), Extreme Quality Attribution (EQA) and Predictive Quality Attribution (PQA).

	Mean			**Standard Deviation**		
Problem	AQA	EQA	PQA	AQA	EQA	PQA
CHR20A	0.4909	0.4845	**0.4972**	1.627E-02	**1.306E-02**	1.920E-02
BUR26E	0.2416	0.2395	**0.2422**	2.049E-03	**1.893E-03**	2.365E-03
TAI30A	0.4909	0.4845	**0.4982**	1.627E-02	**1.306E-02**	1.870E-02
STE36B	0.7978	0.7984	**0.8308**	**7.823E-03**	9.038E-03	8.704E-03
Royal Road	0.0064	0.0063	**0.0072**	1.537E-04	6.3723E-03	**1.473E-03**
KC10-2fl-5rl	0.8062	0.8061	**0.8067**	1.040E-03	8.284E-04	**9.555E-04**
KC30-3fl-1rl	0.1984	0.1961	**0.1995**	**3.255E-03**	3.974E-03	3.806E-03
KC30-3fl-2rl	0.5199	0.5175	**0.5223**	7.312E-03	5.439E-03	**4.821E-03**

The gap between result qualities widens in favour of PQA as the problem difficulty increases. The smallest instances of the QAP (CHR20A) and mQAP (KC10-2fl-5rl) can be assumed to be the least challenging, and there the results are not as clearly in favour of Predictive Quality Assignment. The Royal Road problem and the larger instances of QAP (STE36B) and mQAP (KC30-3fl-1rl,KC10-2fl-5rl) are clearly solved to better quality using PQA, as they are more complex than the smaller problem instances.

To check for a statistical significance of the results we used the Kolmogorov-Smirnov (KS) non-parametric test [149]. PQA was compared with AQA and EQA with a null hypothesis of no difference between the performances. The 30 values of the hypervolume indicators of the repeated trials for each problem instance were submitted to the KS analysis [149] and results are shown in Tables 6.8. All KS tests, used for establishing differences between independent datasets under the assumption that they are not normally distributed, result in a rejection of the null hypothesis with a minimum d-value of 0.2333 at a 95% confidence level. Hence

Table 6.8: The Kolmogorov-Smirnov test values for the 30 runs of each problem instance using different parameter quality attribution schemes: Average Quality Attribution (AQA), Extreme Quality Attribution (EQA) and Predictive Quality Attribution (PQA).

	PQA vs. AQA		**PQA vs. EQA**	
Problem	d	p	d	p
CHR20A	0.2333	0.042	0.3333	0.045
BUR26E	0.3000	0.048	0.3000	0.049
TAI30A	0.3294	0.046	0.3517	0.039
STE36B	0.9667	0.000	0.9667	0.000
Royal Road	0.6333	0.000	0.7333	0.000
KC10-2fl-5rl	0.3333	0.045	0.4000	0.011
KC30-3fl-1rl	0.2897	0.016	0.4333	0.005
KC30-3fl-2rl	0.3433	0.035	0.4333	0.005

we conclude that the superior performance of the Predictive Quality Attribution method is statistically significant.

6.6 Summary

This chapter presented a new predictive method for estimating the probability of a parameter value producing a certain level of quality in the next iterations. The effect of parameter values were recorded at every iteration and the Predictive Quality Attribution (PQA) strategy was employed to forecast the effect of parameter values in the next iteration. Four forecasting techniques were investigated: the Linear Regression (LR), the Simple Moving Average (SMA), the Exponentially-Weighted Moving Average (EWMA), and the Autoregressive Moving Average (ARIMA).

We have examined the suitability of each forecasting technique in predicting the effect of six EA parameters: the mutation rate, the crossover rate, the population size, the mating pool size, the type of crossover operator and the type of mutation operator. We have found that the population size does not conform to the assumptions made by the forecasting techniques, hence it may not be appropriate to predict its effect for the future iterations. However, the results from the experiments have shown that using the PQA strategy to control the population size does not have a negative effect on the performance of the EA. Based on the experiments with the population size, we argue that if EA parameters are controlled without prior testing for statistical properties, PQA can be used without a negative effect in the performance of the algorithm.

Despite being one of the simplest forecasting strategies, Linear Regression has proven to be the most effective model to predict the effect of EA parameters. In the experimental studies we have used the PQA method with Linear Regression to adjust the mutation rate, crossover rate, mating pool size, mutation operator and crossover operator throughout the optimisation process carried out using different problem instances. The trials have demonstrated that PQA outperforms the parameter quality attribution methods currently considered most successful.

Chapter 7

Parameter Value Selection Strategy

7.1 Introduction

When controlling algorithm parameters, the probability of selecting parameter values from a particular interval depends on how suitable, i.e. successful, that particular parameter value has proven in the previous iterations. In the early stages of the search, there is no knowledge which parameter values perform well. At every step, when evidence becomes available, the control method faces a dilemma: using the parameter value that has the highest assigned quality or exploring new parameter values. The trade-off that has to be made between the choice of using currently best parameter values and exploring new settings has already been addressed by state-of-the-art adaptive parameter control methods, which refer to it as the 'exploration vs. exploitation dilemma' [182, 181, 56]. The main task is to ensure the right balance between using currently well performing parameter values and exploring unseen ones to be adopted in the future iterations. If more priority is given to the exploitation of good parameter settings, the strategy of the parameter control method focuses predominately on the short-term performance of the algorithm. If it opts for the exploration of other parameter values, the aim is to achieve long-term performance.

Current parameter value selection strategies handle the 'exploration vs. exploitation dilemma' in different ways. Probability Matching (PM) [182, 181] and Adaptive Pursuit (AP) [182, 181] control the exploration of underperforming parameter values by assigning a lower threshold on the selection probabilities p_{min}. In PM, the projected probability of a parameter value providing good quality results at the next time step is based on a running average of past rewards (quality). The minimum probability p_{min} is enforced on values which do not receive rewards in order to maintain a non-zero probability. Adaptive Pursuit (AP) [181] was conceived with the goal of improving the performance of PM by ensuring an appropriate difference in probabilities depending on experienced performance. AP establishes the respective rewards for the parameter values used, but only applies the maximum reward to the value of the best-performing algorithm instance. All other values have their probabilities of future use diminished. Similar to PM, a non-zero probability is enforced as a minimum probability p_{min}, which ensures the exploration of underperforming parameter values.

Unlike PM and AP the Dynamic Multi-Armed Bandit (DMAB) [56, 123] focuses on the usage frequency of parameter values. The selection mechanism of DMAB controls the trade-off between exploitation, which favours the parameter values with best effect on the algorithm performance, and exploration, which favours the parameter values that have not been used that frequently. The selection probability of a parameter value is inversely proportionate to its usage frequency in the past. Furthermore, a change detection test is employed to check whether the quality distribution of the parameter value $q(\upsilon_{ij})$ has changed. When a change is detected, the empirical quality is re-initialised. As a result, DMAB identifies the new best parameter value without being slowed down by old information.

In essence, parameter control methods found in the literature select the next parameter values based on different rules which ensure the exploration vs. exploita-

tion trade-off. These rules are composed of two parts. The first part is concerned with the exploitation of parameter values, and assigns a selection probability to parameter values based on their quality. The second part deals with underperforming or previously unused parameter values, which are assigned a minimum selection probability (p_{min}) for exploration purposes.

One aspect of parameter exploration that has not been covered by current research is the choice for real-valued parameter assignments, such as the mutation rate and the crossover rate. Usually, parameter control methods [191, 78, 29, 84, 56, 182, 181] discretise the choices for parameter assignments beforehand and apply the feedback from the algorithm to the preselected choices. Based on insights offered by existing research [9] it is likely that the initial discretisation will provide values which are suboptimal for some problems or their instances. Defining narrow ranges leads to more accuracy but increased combinatorial complexity. Leaving ranges wider entails a sampling inaccuracy as the actually sampled value may be far from the value whose success the range's usage probability is attributable to. Ideally, the ranges should be optimised by the parameter control process, which leads us to the following research question:

* **What is an effective method for configuring real-valued parameters during the optimisation process?**

To address this question, we introduce an Adaptive Range Parameter Selection (ARPS) strategy, which explores the parameter-values space in an efficient way, by dynamically changing the parameter intervals during the optimisation process. The parameter control method described in this section uses an efficient way to reduce the number of observations required to explore new values of parameters and ensures a balanced trade-off between exploitation and exploration of parameter values. To validate the approach, we investigate if using dynamic ranges improves the choice of parameter values and as a result achieves better solution quality.

7.2 Adaptive Range Parameter Selection

Adaptive Range Parameter Selection (ARPS) adjusts parameter value ranges as the optimisation process progresses. After each iteration, the best-performing range is halved, whereas the worst-performing is merged with the worse-performing of its neighbours. This technique was first conceived for the context of parallel computing [139] but has never been applied to the dynamic adjustment of parameter ranges. Mostaghim [139, 140] uses a self-organized Invasive Parallel Optimization (SIPO) technique to solve optimization problems in parallel. The algorithm starts with one resource and automatically divides the optimisation task stepwise into smaller tasks which are assigned to more resources. We extend this technique to the adaptation of the parameter ranges of Evolutionary Algorithms.

The technique used to dynamically adapt parameter ranges is illustrated with the help of a single parameter υ_i. Figure 7.1 shows how the parameter values are initially divided into two ranges. The first range υ_{i1} is defined by its minimum and maximum values $[\upsilon_{i1}(min), \upsilon_{i1}(max)]$, and the second range υ_{i2} represents the set of values that lie within $[\upsilon_{i2}(min), \upsilon_{i2}(max)]$.

At the beginning of the search, both ranges have equal success rates, denoted as the conditional probabilities $p(e^+|\upsilon_{i1})$ and $p(e^+|\upsilon_{i2})$, with e denoting the expectation and e^+ denoting a successful outcome given the usage of a value from range υ_{ij} for the parameter υ_i. This value is approximated as the ratio of the number of times the usage of the value υ_{ij} was successful and the number of times it was used, denoted $\frac{n(\upsilon_{ij} \wedge e^+)}{n(\upsilon_{ij})}$ in the algorithmic listing 2. In Figure 7.1, an equal height of two ranges represents the equality of the probabilities of both ranges to be selected for use in the next iteration.

After applying the parameter values sampled from the ranges for the optimisation process, the conditional probabilities of each interval are recalculated based on their

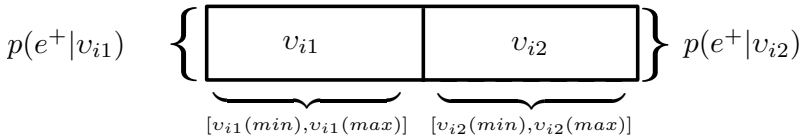


Figure 7.1: Parameter υ_i is initially divided into two equal ranges. An equal height of two ranges represents the equality of the probabilities of both ranges to be selected for use in the next iteration.

usage and performance in the latest iteration.

Assuming that the new conditional probabilities have the proportions shown in Figure 7.2, the success rate of the first interval, i.e. $p(e^+|\upsilon_{i1})$, is greater than that of the second interval ($p(e^+|\upsilon_{i2})$).

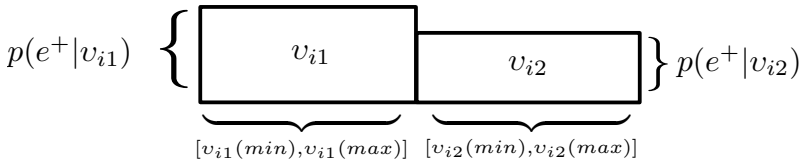


Figure 7.2: The new success rates of the levels of parameter υ_i after running the algorithm.

The adaptive range selection strategy divides the level with the highest success rate into two new levels, denoted as $p(e^+|\upsilon_{i1_1})$ and $p(e^+|\upsilon_{i1_2})$, as shown in Figure 7.3.

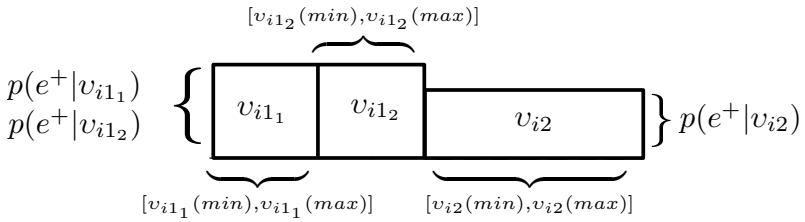


Figure 7.3: The level with the highest success rate is divided into two.

The conditional probabilities of both new levels are equal to the conditional probability of the level they were created from. That is:

$$p(e^+|\upsilon_{i1_1}) = p(e^+|\upsilon_{i1})$$

$$p(e^+|\upsilon_{i1_2}) = p(e^+|\upsilon_{i1})$$

As a result, the most successful interval is refined into smaller intervals, and the selection probability of the values that lie within these ranges is increased. This increases the exploitation of the intervals and the exploration of new values within the intervals.

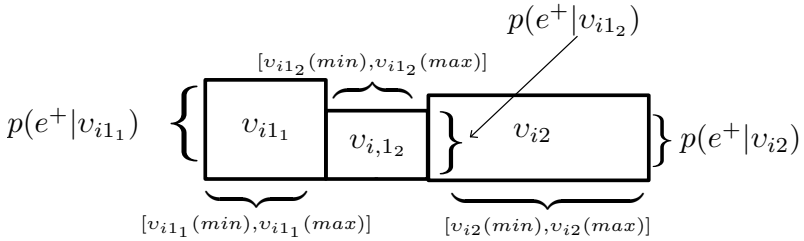


Figure 7.4: The new success rates adjusting the selection probabilities of the values of parameter υ_i after running the algorithm.

The Adaptive Range Parameter Selection Strategy merges the worst performing range υ_{i1_2} in Figure 7.4, with the worse-performing neighbouring range. In this case, the range υ_{i1_2} has been merged with the range υ_{i2} forming the new value range υ'_{i2} as shown in Figure 7.5.

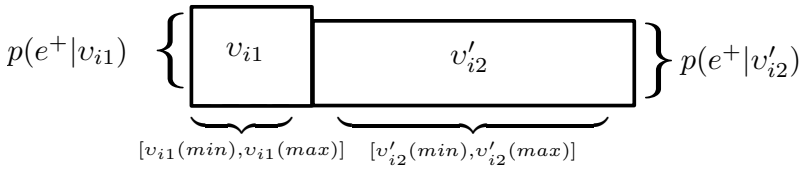


Figure 7.5: The ranges with the lowest success rates are merged to form a new range, denoted as υ'_{i2}.

The selection probability of the new range is set equal to the higher selection probability of the two ranges. The values that lie within the newly formed range υ'_{i2}

have a lower probability of being selected when compared to the values of the previous range υ_{i2}. As a result, the selection probability of under-performing parameter values is decreased, however there is still a possibility that they may be selected. In the future, if the value range υ'_{i2} shows a successful performance, the Adaptive Range Parameter Selection strategy will divide it again into two different ranges. As a result the number of the parameter ranges is regulated during the optimization process dynamically. The algorithmic listings 3 and 4 demonstrate how the adaptive parameter control applies the dynamic changes to the parameter ranges.

Algorithm 3 The algorithm for finding the best and worst performing ranges.

```
    procedure BestAndWorst(υ_i)
2:      for all parameter υ_i, i ∈ n do
            p(υ_i,best) = 0.0
4:          p(υ_i,worst) = 1.0
            for all parameter value υ_ij ∈ m do
6:              p(e+|υ_ij) = QualityAssessmentStrategy(υ_ij)
                if p(e+|υ_ij) > p(υ_i,best) then
8:                  p(υ_i,best) ← p(e+|υ_ij)
                    υ_i,best ← υ_ij
10:             end if
                if p(e+|υ_ij) < p(υ_i,worst) then
12:                 p(υ_i,worst) ← p(e+|υ_ij)
                    υ_i,worst ← υ_ij
14:             end if
            end for
16:     end for
        AdjustLevels(υ_i,best, υ_i,worst)
18: end procedure
```

Every parameter υ_i has a number of ranges υ_{ij}. After each iteration, we investigate the parameter's range's success rate using the quality assessment strategy introduced in Chapter 6. The variable $p(\upsilon_{i,best})$ then holds the best ratio of all ranges of parameter υ_i, and $\upsilon_{i,best}$ points to the best-performing range of this parameter. Analogously, $p(\upsilon_{i,worst})$ stores the success rate of the worst-performing range of parameter υ_i. The best range $\upsilon_{i,best}$ is subsequently split into $\upsilon_{i,best1}$ and

$\upsilon_{i,best2}$, both of which are assigned the raw probability value of $p(\upsilon_{i,best})$. Similarly, the worst range $\upsilon_{i,worst}$ is expanded to cover the worse-performing of its neighbours $\upsilon_{i,worst+1}$, and the new range is assigned the raw probability value $p(\upsilon_{i,worst})$ of the worst-performing range.

Algorithm 4 Adjusting ranges according to the best and worst performing ones.

procedure AdjustLevels($\upsilon_{i,best}, \upsilon_{i,worst}$)
  $range \leftarrow \upsilon_{i,best}(max) - \upsilon_{i,best}(min)$
  $\upsilon_{i,best_1}(min) \leftarrow \upsilon_{i,best}(min)$
  $\upsilon_{i,best_1}(max) \leftarrow \upsilon_{i,best}(min) + range/2$
  $\upsilon_{i,best_2}(min) \leftarrow \upsilon_{i,best}(max) - range/2$
  $\upsilon_{i,best_2}(max) \leftarrow \upsilon_{i,best}(max)$
  $p(e^+|\upsilon_{i,best_1}) \leftarrow p(\upsilon_{i,best})$
  $p(e^+|\upsilon_{i,best_2}) \leftarrow p(\upsilon_{i,best})$
  if $p(e^+|\upsilon_{i,worst+1}) < p(e^+|\upsilon_{i,worst-1})$ **then**
    $\upsilon_{i,worst+1}(min) = \upsilon_{i,worst}(min)$
  end if
  if $p(e^+|\upsilon_{i,worst+1}) > p(e^+|\upsilon_{i,worst-1})$ **then**
    $\upsilon_{i,worst-1}(min) = \upsilon_{i,worst}(max)$
  end if
end procedure

Algorithm 5 Fitness proportionate selection.

procedure FitnessProportionateSelection($p(\upsilon_{i1}), ..., p(\upsilon_{im})$)
  $sumCP = 0.0$
  for all $j \leftarrow 1, m$ **do**
    $sumCP += p(\upsilon_{ij})$
    $cumulative_{ij} = sumCP$
  end for
  $number =$ Random($[0, sumCP]$)
  for all $j \leftarrow 1, m$ **do**
    if $number < cumulative_{ij}$ **then**
      return j
    end if
  end for
end procedure

Adaptive parameters are set for an iteration at a time, sampling the respective values probabilistically from the distribution obtained from the parameter value se-

lection strategy. A fitness-proportionate selection mechanism is applied to choose the next parameter configurations, described in the algorithm listing 5. The predicted selection probabilities are used to associate a probability of selection with each parameter configuration. This is achieved by dividing the predicted success rate of every parameter value by the total success rates of all other values of that parameter, thereby normalising them to 1. Then a random selection is made.

While candidate parameter values with a higher success rate will be less likely to be eliminated, there remains a probability that they may be. With this kind of fitness proportionate selection there is a chance that some weaker solutions may survive the selection process; this is an advantage, as though a parameter value may be not as successful, it may include some component which could prove useful in the following stages of the search.

7.3 Validation

The Adaptive Range Parameter Selection (ARPS) strategy described in this thesis adjusts the range sizes of continuous parameter values as the optimisation process progresses. Notable parameter value selection strategies such as Probability Matching (PM) [181], Adaptive Pursuit (AP) [181] and Dynamic Multi-Armed Bandit (DMAB) [56] use static ranges or specific values. In this section we investigate if using dynamic ranges improves the choice of parameter values and as a result achieves better solution quality. This motivates the following research question:

* **What is an effective method for configuring real-valued parameters during the optimisation process?**

To answer the research question posed in this chapter, we have performed a set of experiments in which Adaptive Range Parameter Selection (ARPS) is compared with three parameter value selection strategies: Probability Matching (PM) [181], Adaptive Pursuit (AP) [181] and Dynamic Multi-Armed Bandit (DMAB) [56].

7.3.1 Benchmark Parameter Value Selection Strategies

The main characteristic that distinguishes different adaptive parameter value selection methods is how they explore the parameter space, i.e. how they calculate the probability vector $P = \{p(\upsilon_{11}), p(\upsilon_{12}), ..., p(\upsilon_{1m_1}), ..., p(\upsilon_{nm_n})\}$ over time.

Vector P represents the selection probability for each parameter value, estimated based on the parameter qualities $Q = \{q(\upsilon_{11}), q(\upsilon_{12}), ..., q(\upsilon_{1m_1}), ..., q(\upsilon_{nm_n})\}$. The quality of a parameter value is approximated from the effect of the parameter values on the performance of the algorithm. The vector of all parameter effects is denoted as $\mathcal{E} = \{e(\upsilon_{11}), e(\upsilon_{12}), ..., e(\upsilon_{1m_1}), ..., e(\upsilon_{nm_n})\}$. The main goal of adaptive parameter control strategies is to adapt the vector of probabilities P such that the expected value of the cumulative effect $E[\mathcal{E}] = \sum_{i=1}^{n} e(\upsilon_{ij})$ is maximised.

Probability Matching

Probability Matching (PM) [181] uses reinforcement learning to project the probability of a parameter value performing well based on the previous performance of an algorithm that used this value. The projected probability of a value providing good quality results at the next time step is based on a running average of past rewards. Rewards are allocated on the basis of the outcome of the optimisation process the parameter value was used in.

In essence, PM updates the probability vector P such that the probability of applying parameter value υ_{ij} is proportional to the quality of that parameter value, i.e. the proportion of the estimated quality $q_t(\upsilon_{ij})$ to the sum of all quality estimates $\sum_{s=1}^{n} q_t(\upsilon_{is})$.

The quality estimate $q(\upsilon_{ij})$ is updated using an additive relaxation mechanism with adaptation rate α, $(0 < \alpha \leq 1)$. When α increases, more weight is given to the current quality. The update rule for the quality of parameters is as follows:

$$q_t(\upsilon_{ij}) = q_{t-1}(\upsilon_{ij}) + \alpha(e_{t-1}(\upsilon_{ij}) - q_{t-1}(\upsilon_{ij})) \tag{7.1}$$

The selection probability for each parameter is the estimated as follows:

$$p_t(\upsilon_{ij}) = \frac{q_t(\upsilon_{ij})}{\sum_{s=1}^{n} q_t(\upsilon_{is})} \tag{7.2}$$

This selection strategy may lead to the loss of some parameter values, since parameter values will no longer be selected when their quality $q(\upsilon_{ij})$ becomes zero. This property is unwelcome, since the optimisation process is a non-stationary environment, which means that the parameter value may become valuable again in the future stages. To remedy this problem, i.e. ensure that parameter values do not get lost, Thierens [181] enforced a minimum selection probability p_{min} for every

parameter value. The new equation for selection probabilities is as follows:

$$p_t(\upsilon_{ij}) = p_{min} + (1 - n * p_{min}) \frac{q_t(\upsilon_{ij})}{\sum_{s=1}^{n} q_t(\upsilon_{is})} \tag{7.3}$$

where p_{min} is a minimum probability enforced on values which do not receive rewards in order to maintain a non-zero probability. The general motivation of a minimum value is the assumption that parameter values which do not perform well at present might be optimal in the future.

If a parameter does not perform well, its reward is not updated. As a result, the quality estimate of this parameter converges to zero, which in turn makes the selection probability of that parameter equal to p_{min}. On the other hand, if only one parameter receives a reward for a long time, the selection probability of this parameter will be $p_{min} + 1 - np_{min}$, which is equal to the maximum selection probability p_{max}. The main steps of Probability Matching method are described in Algorithm 6.

Algorithm 6 Probability Matching (PM).

```
1: procedure PM(P, Q, n, p_min, α)
2:     for i ← 1, n do
3:         for j ← 1, m do
4:             p(υ_ij) = 1/m
5:             q(υ_ij) = 1.0
6:         end for
7:     end for
8:     while finalCriterion == False do
9:         for i ← 1, n do
10:            j = FitnessProportionateSelection(p(υ_i1), ..., p(υ_im))
11:            e(υ_ij) = CalculateEffect(υ_ij)
12:            q(υ_ij) = q(υ_ij) + α(e(υ_ij) − q(υ_ij))
13:            p(υ_ij) = p_min + (1 − n * p_min) q(υ_ij) / Σ_{s=1}^{n} q(υ_is)
14:        end for
15:    end while
16: end procedure
```

PM has been criticised for the fact that the probability values resulting from the reward allocations poorly reflect the relative differences in algorithm performance

when using the values due to the relaxation mechanism described by Equation 7.1. Values with vastly superior performance may only be differentiated by a marginal increase of the probability of being chosen in the next step.

Adaptive Pursuit

Adaptive Pursuit (AP) [181] was conceived with the goal of improving the performance of PM by ensuring an appropriate difference in probabilities depending on experienced performance. After an iteration of the optimisation process, AP establishes the respective rewards for the parameter values used, but only applies the maximum reward to the value of the best-performing algorithm instance. All other values have their probabilities of future use diminished. A non-zero probability is enforced as a minimum probability. The best parameter value of the current iteration is estimated as follows:

$$j^* = \arg\max_{j=1,\ldots,m} q(\upsilon_{ij}) \tag{7.4}$$

where $q(\upsilon_{ij})$ is approximated in the same way as for PM using Equation 7.1. The selection probability of the next parameter values is obtained as follows:

$$p_t(\upsilon_{ij}) = \begin{cases} (1-\beta)p_{t-1}(\upsilon_{ij}) + \beta & \text{if } j = j^* \\ (1-\beta)p_{t-1}(\upsilon_{ij}) & \text{otherwise} \end{cases} \tag{7.5}$$

where β is the learning rate, which controls the greediness of the 'winner-take-all' strategy. If a parameter value is continuously the best-performing one, its selection probability will converge to one, whereas the probabilities of the other parameter values will converge to zero. This is not suitable for non-stationary environments, such as Evolutionary Algorithms, where the effect of the parameter values changes over time. In order not to lose underperforming parameter values, the selection rule

in Equation 7.6 is modified with a minimum and maximum selection probability as follows:

$$p_t(\upsilon_{ij}) = \begin{cases} p_{t-1}(\upsilon_{ij}) + \beta(p_{max} - p_{t-1}(\upsilon_{ij})) & \text{if } j = j^* \\ p_{t-1}(\upsilon_{ij}) + \beta(p_{min} - p_{t-1}(\upsilon_{ij})) & \text{otherwise} \end{cases} \tag{7.6}$$

under the constraint:

$$p_{max} = 1 - (m-1)p_{min} \tag{7.7}$$

where m is the total number of parameter values. This constraint ensures that the sum of the probabilities of all parameter values is equal to 1. In order to ensure this constraint, p_{min} should be less than $\frac{1}{m}$. A recommended value [181] for p_{min} is $\frac{1}{2m}$, which ensures that the best parameter value is selected half the time, while the other half is allocated to all other parameter values.

Similar to the PM parameter value selection strategy, in the AP approach, every parameter is selected proportionally to the probability vector P, and the quality vector Q is updated accordingly. The main steps of the Adaptive Pursuit parameter adaptation strategy are described in Algorithm 7.

Dynamic Multi-Armed Bandit

The Dynamic Multi-Armed Bandit (DMAB) [56, 123], similar to AP and PM, aims at finding an optimal parameter value selection strategy which maximises a cumulative effect along time. DMAB, in the same way as other methods, keeps an empirical quality $q(\upsilon_{ij})$ with every parameter value, based on the effect the parameter value has on the performance of the algorithm. Unlike other methods, DMAB records the number of times a parameter is employed, denoted as $n(\upsilon_{ij})$, which is used in the calculation of the best parameter as follows:

Algorithm 7 Adaptive Pursuit (AP).

procedure AP($P, Q, p_{min}, \alpha, \beta$)
  $p_{max} = 1 - (n-1)p_{min}$
  for $i \leftarrow 1$, n **do**
    for $j \leftarrow 1$, m **do**
      $p(\upsilon_{ij}) = \frac{1}{m}$
      $q(\upsilon_{ij}) = 1.0$
    end for
  end for
  while finalCriterion == False **do**
    $j = \text{FitnessProportionateSelection}(p(\upsilon_{i1}), ..., p(\upsilon_{im}))$
    $e(\upsilon_{ij}) = \text{CalculateEffect}(\upsilon_{ij})$
    $q(\upsilon_{ij}) = q(\upsilon_{ij}) + \alpha(e(\upsilon_{ij}) - q(\upsilon_{ij}))$
    $j^* = \arg\max_{s=1...n} q(\upsilon_{is})$
    $p(\upsilon_{ij^*}) = p(\upsilon_{ij^*}) + \beta(p_{max} - p(\upsilon_{ij^*}))$
    for $s \leftarrow 1$, m **do**
      if $s \neq j^*$ **then**
        $p(\upsilon_{is}) = p(\upsilon_{is}) + \beta(p_{min} - p(\upsilon_{is}))$
      end if
    end for
  end while
end procedure

$$\arg\max_{j=1,...,m} \left(q(\upsilon_{ij}) + C\sqrt{\frac{2log\sum_s^m n(\upsilon_{is})}{n(\upsilon_{ij})}} \right) \tag{7.8}$$

where C is a hyperparameter, which serves as a scaling factor. This selection mechanism is called the Upper Confidence Bound (UCB) and controls the trade-off between exploitation (the left term in Equation 7.8), which favours the parameter values with best effect on the algorithm performance, and exploration (the right term in Equation 7.8), which favours the parameter values that have not been used that frequently.

The parameter value selection strategy in Equation 7.8 considers a static environment (the unknown effect probability of any parameter is fixed along time), whereas the optimisation process is intrinsically dynamic (the quality of any operator is bound to vary along evolution). Even though every operator is selected from

Algorithm 8 Dynamic Multi-Armed Bandit (DMAB).

procedure DMAB(P, Q, N, C, γ)
  for $i \leftarrow 1$, n **do**
    for $j \leftarrow 1$, m **do**
      $p(v_{ij}) = \frac{1}{m}$
      $q(v_{ij}) = 1.0$
      $n(v_{ij}) = 0$
      $\hat{m}(v_{ij}) = 0.0$
      $M(v_{ij}) = 0.0$
    end for
  end for
  while finalCriterion == False **do**
    for $i \leftarrow 1$, n **do**
      for $j \leftarrow 1$, m **do**
        if $n(v_{ij}) == 0$ **then**
          $s =$FitnessProportionateSelection$(p(v_{i1}), ..., p(v_{im}))$
          $n(v_{is}) += 1$
          $e(v_{is}) =$CalculateEffect(v_{is})
          $q(v_{is}) = \left(\frac{(n(v_{is})-1)q(v_{is})+e(v_{is})}{n(v_{is})}\right)$
          $\hat{m}(v_{is}) = m(v_{is}) + (e(v_{is}) - q(v_{is}) + \delta)$
          $M(v_{is}) = \arg\max_{k=1,...,t} \left(|\hat{m}(v_{ij})^k|\right)$
          if $M(v_{ij}) - |\hat{m}(v_{is})| > \gamma$ **then**
            Restart(DMAB(n, C, γ))
          end if
        else
          $j^* = \arg\max_{j=1,...,m} \left(q(v_{ij}) + C\sqrt{\frac{2log\sum_s^m n(v_{is})}{n(v_{ij})}}\right)$
          $n(v_{ij^*}) += 1$
          $e(v_{ij^*}) =$CalculateEffect(v_{ij^*})
          $q(v_{ij^*}) = \left(\frac{(n(v_{ij^*})-1)q(v_{ij^*})+e(v_{ij^*})}{n(v_{ij^*})}\right)$
          $m(v_{ij^*}) = m(v_{ij^*}) + (e(v_{ij^*}) - q(v_{ij^*}) + \delta)$
          $M(v_{ij^*}) = \arg\max_{k=1,...,t} \left(|\hat{m}(v_{ij^*})^k|\right)$
          if $M(v_{ij^*}) - |\hat{m}(v_{ij^*})| > \gamma$ **then**
            Restart(DMAB(P, Q, N, C, γ))
          end if
        end if
      end for
    end for
  end while
end procedure

time to time, in practice UCB would wait too long before realising that some new parameter value has become the best-performing. To address this limitation, DMAB completely recalculates the probabilities when a change in the effects distribution is detected using a change detection test, in this case the statistical Page-Hinkley (PH) test. The PH test checks whether the quality distribution of the parameter value has changed. When a change is detected, the algorithm is restarted, i.e. the empirical quality $q(v_{ij})$ and confidence intervals are re-initialised. As a result, DMAB can quickly identify the new best parameter value without being slowed down by old information. Formally, the DMAB calculates the average reward over the last time steps for each parameter value, which is equal to the quality of that parameter value estimated as follows:

$$q(v_{ij}) = \Big(\frac{(n(v_{ij}) - 1)q(v_{ij}) + e(v_{ij})}{n(v_{ij})}\Big) \tag{7.9}$$

Then it obtains the difference between the instant and the average effect, by incorporating a tolerance factor δ:

$$\hat{m}(v_{ij}) = \hat{m}(v_{ij}) + (e(v_{ij}) - q(v_{ij}) + \delta) \tag{7.10}$$

Next it calculates the maximum value of $m(v_{ij})$ in the past iterations as:

$$M(v_{ij}) = \arg\max_{k=1,\ldots,t} \left(|\hat{m}(v_{ij}^k)|\right) \tag{7.11}$$

Finally, the PH test is triggered when the difference between the maximum value $M(v_{ij})$ in the past and its current value $|\hat{m}(v_{ij})|$ is greater than some user-defined threshold γ.

The PH test is thus parametrized by γ, which controls the sensitivity of the test to false alarms, and δ, which enforces the robustness of the test in slowly varying

environments. Following guidelines from the original work [55], δ is kept fixed to 0.15. The main steps of the algorithm for the Dynamic Multi-Armed Bandit are shown in Algorithm 8.

7.3.2 Experimental Settings

The Adaptive Range Parameter Selection Strategy is compared with three successful algorithms representative of adaptive parameter value selection strategies: Probability Matching (PM) [181], Adaptive Pursuit (AP) [181] and Dynamic Multi-Armed Bandit (DMAB) [56]. A general description of these methods is given in Section 3.3.3. A comprehensive description and comparison of the three adaptive methods is given by Fialho et al. [56].

All adaptive algorithms involve hyper-parameters, which have to be tuned depending on the optimisation problem at hand. This defines another optimisation problem, which can become computationally expensive if we attempt an exhaustive exploration of the search space. We used recommendations from Thierens [181], Fialho et al. [56] and DaCosta et al. [36] when tuning the values of hyper-parameters for the adaptive algorithms, which are depicted in Table 7.1.

Table 7.1: Hyper-parameters of the three adaptive methods: Dynamic Multi-Armed Bandit (DMAB), Adaptive Pursuit (AP) and Probability Matching (PM).

Benchmark	**Hyperparameter**	**Value**	**Description**
DMAB	ς	0.5	scaling factor
DMAB	γ	100	PH threshold
AP,PM	p_{min}	0.1	minimum selection probability
AP,PM	α	0.8	adaptation rate
AP,PM	β	0.8	adaptation rate

The four optimisation schemes are used to optimise ten problem instances which

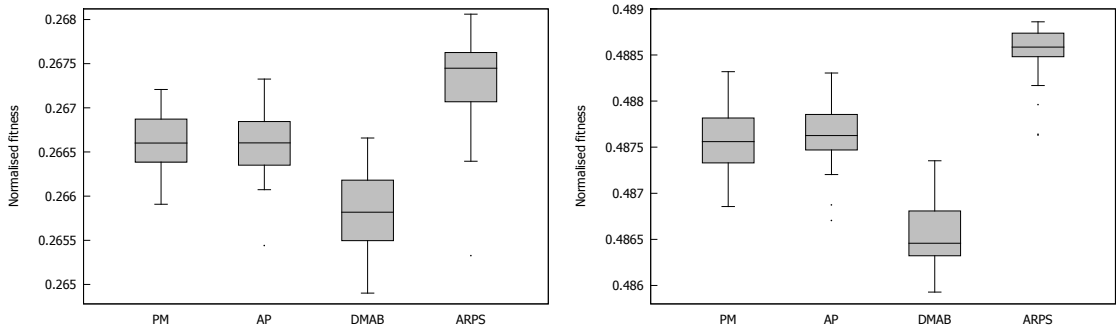
vary in size and difficulty: four instances of the multiobjective Quadratic Assignment Problem (KC30-3fl-3uni, KC20-2fl-5rl, KC30-3fl-2rl and KC30-3fl-3rl), the Royal Road problem and five instances of the Quadratic Assignment Problem (BUR26A, BUR26B, BUR26E, TAI30B and STE36B). The RR function is being used here as a benchmark to match the Genetic Algorithm, whose parameters are being optimised for the experimental results presented. It has also been used by Fialho, Schoenauer and Sebag [56], whose results are being used for the comparison.

We perform 30 trials for each problem and optimisation scheme. The final solutions of each optimisation scheme are recorded and the results are compared using the normalised fitness for the singleobjective problems and the hypervolume indicator, described in Section 3.3.4, for the multiobjective problems. We use the Kolmogorov-Smirnov (KS) nonparametric test [149] to check for the statistical significance of the final results.

For the benefit of these experiments, only the crossover and mutation rates were varied. For the crossover and mutation rates we use different value intervals to sample from, with a cardinality of two intervals or levels with ranges of $\{[0.6, 0.799], [0.8, 1.0]\}$ produced the best results among several cardinalities with even spreads between 0.6 and 1 for the crossover rate, and two intervals with ranges of $\{[0.001, 0.25], [0.2505, 0.5]\}$ between 0.001 and 0.5 for the mutation rate.

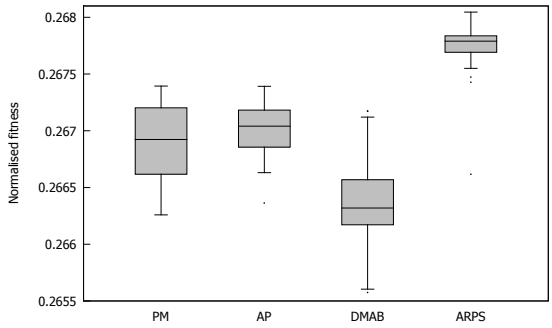
7.3.3 Results

The 30 results of the repeated trials are presented as boxplots in Figure 7.6 and 7.7 for the Quadratic Assignment Problem (QAP), the Royal Road problem (RR) and the multiobjective Quadratic Assignment Problem (mQAP). The empirical results are not normally distributed, but the mean and 25th percentile of ARPS are consistently above the respective values of the benchmark approaches.

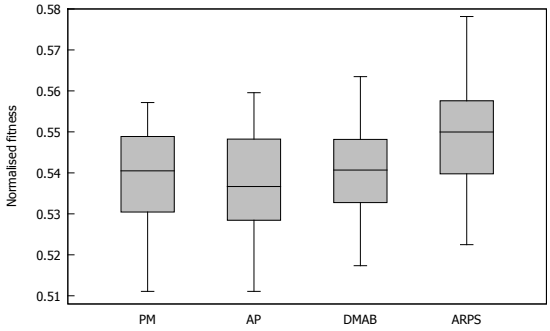


(a) BUR26A

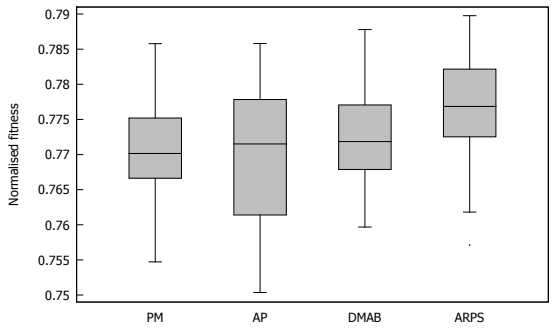
(b) BUR26B



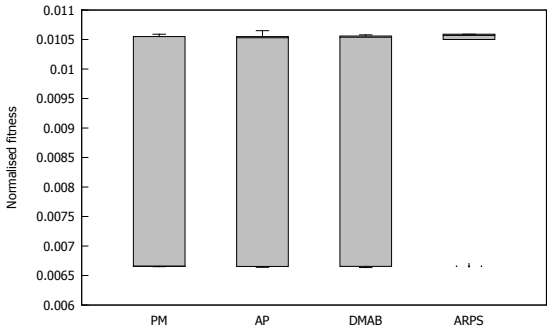
(c) BUR26E



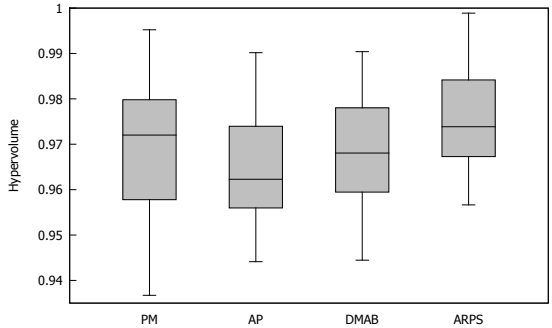
(d) TAI30B



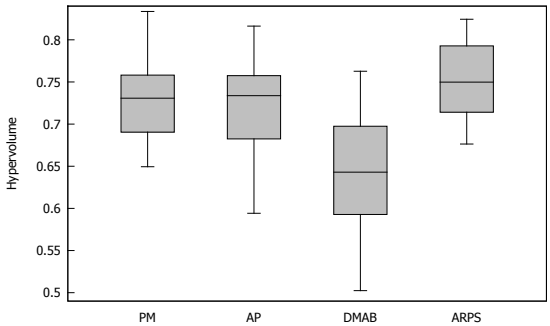
(e) STE36B



(f) Royal Road

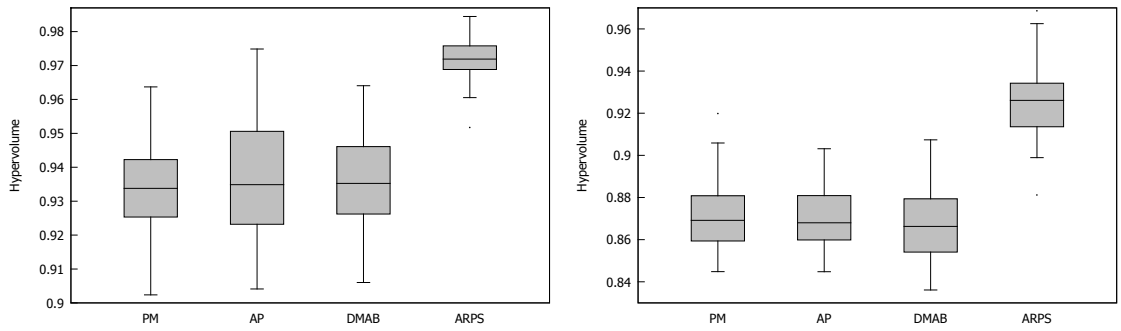


(g) KC20-2fl-5rl



(h) KC30-3fl-3uni

Figure 7.6: Boxplots of the 30 trials of the four different parameter value selection schemes used with an Evolutionary Algorithm optimising the QAP, RR and mQAP.



(a) KC30-3fl-2rl (b) KC30-3fl-3rl

Figure 7.7: Boxplots of the 30 trials of the four different parameter value selection schemes used with an Evolutionary Algorithm optimising the multiobjective Quadratic Assignment Problem.

The means and standard deviations are listed in Table 7.2, which clearly show a significant difference between the result groups of ARPS and the benchmarks. The mean performance of ARPS is consistently above the averages of the benchmark approaches. The standard deviation of ARPS is relatively high in the singleobjective Quadratic Assignment Problem, but lower than the respective values of the other parameter value selection strategies in the multiobjective Quadratic Assignment Problem instances.

The gap between result qualities widens in favour of ARPS as the problem difficulty increases. The biobjective mQAP of dimension n=20 (KC20-2fl-5rl) can be assumed to be the least challenging, and there the results are not as clearly in favour of ARPS. The triobjective mQAP instances of dimension n=30 (KC30-3fl-2rl and KC30-3fl-3rl) are clearly solved to better quality using ARPS, as they are more complex than the smaller mQAP instances.

As our method consistently outperforms the three other parameter value selection schemes, we employ the Kolmogorov-Smirnov (KS) non-parametric test [149] to check for a statistical difference. The 30 hypervolume indicators of the repeated trials for each of the problem instances were submitted to the KS analysis. ARPS

Table 7.2: The means and standard deviations of the 30 runs of each problem instance using different parameter value selection schemes.

	Mean			
Problem	AP	PM	DMAB	ARPS
Royal Road	0.0086	0.0085	0.0093	**0.0096**
BUR26A	0.2666	0.2666	0.2666	**0.2672**
BUR26B	0.4876	0.4876	0.4865	**0.4882**
BUR26E	0.2670	0.2669	0.2664	**0.2674**
TAI30B	0.5294	0.5384	0.5407	**0.5483**
STE36B	0.7704	0.7704	0.7730	**0.7773**
KC20-2fl-5rl	0.9641	0.9665	0.9684	**0.9768**
KC30-3fl-2rl	0.9364	0.9339	0.9360	**0.9720**
KC30-3fl-3rl	0.9429	0.9417	0.9251	**0.9530**
KC30-3fl-3uni	0.7253	0.7284	0.6468	**0.7553**
	Standard deviation			
Problem	AP	PM	DMAB	ARPS
Royal Road	1.985E-03	8.5381E-03	1.835E-03	**1.702E-03**
BUR26A	3.998E-04	**3.398E-04**	4.410E-04	1.032E-03
BUR26B	3.673E-04	**3.401E-04**	3.564E-04	6.286E-04
BUR26E	**2.491E-04**	3.189E-04	4.233E-04	6.325E-04
TAI30B	1.272E-02	1.233E-02	**1.178E-02**	1.423E-02
STE36B	9.796E-03	**7.578E-03**	7.938E-03	9.659E-03
KC20-2fl-5rl	1.190E-02	1.561E-02	1.124E-02	**1.123E-02**
KC30-3fl-2rl	1.759E-02	1.639E-02	1.516E-02	**7.423E-03**
KC30-3fl-3rl	1.268E-02	1.198E-02	2.062E-02	**1.127E-02**
KC30-3fl-3uni	5.110E-02	4.552E-02	5.751E-02	**4.500E-02**

was compared to the other three parameter value selection schemes, with a null hypothesis of an insignificant difference between the performances (ARPS vs. DMAB, ARPS vs. AP and ARPS vs. PM). The results of the tests are shown in Table 7.3.

All KS tests, used for establishing that there is no difference between independent datasets under the assumption that they are not normally distributed, result in a

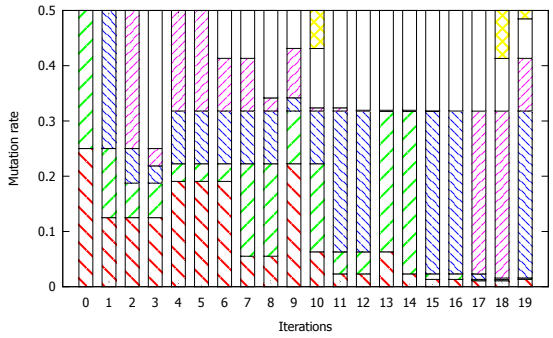
rejection of the null hypothesis with a minimum d-value of 0.3000 at a 95% confidence level. Hence we conclude that the superior performance of ARPS is statistically significant.

Table 7.3: The Kolmogorov-Smirnov test values of the 30 runs of each problem instance using different parameter value selection schemes.

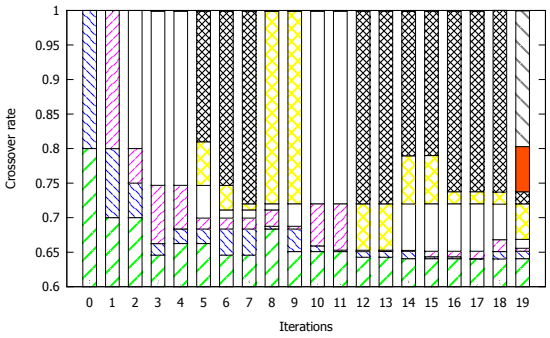
	ARPS vs. DMAB		**ARPS vs. AP**		**ARPS vs. PM**	
Problem	d	p	d	p	d	p
Royal Road	0.4483	0.004	0.4138	0.009	0.4138	0.009
BUR26A	0.6207	0.000	0.8621	0.000	0.6539	0.000
BUR26B	0.6552	0.000	0.9310	0.000	0.6552	0.000
BUR26E	0.6552	0.000	0.7586	0.000	0.6552	0.000
TAI30B	0.3667	0.026	0.4000	0.011	0.3667	0.026
STE36B	0.4000	0.011	0.3000	0.049	0.4667	0.002
KC20-2fl-5rl	0.3640	0.036	0.4406	0.006	0.3963	0.050
KC30-3fl-2rl	0.8966	0.000	0.8966	0.000	0.5310	0.000
KC30-3fl-3rl	0.6897	0.000	0.3793	0.022	0.4828	0.001
KC30-3fl-3uni	0.6897	0.000	0.3631	0.043	0.3308	0.040

The difference in performance seems more pronounced in the trials using the Royal Road problem, the Quadratic Assignment Problem and the bigger instances of the multiobjective Quadratic Assignment problem (KC30-3fl-2rl, KC30-3fl-3rl). The least benefit ARPS provides for the smallest instance of the multiobjective Quadratic Assignment problem (KC20-2fl-5rl). This is an 'easier' instance to solve, hence the algorithm performance can be expected to be more robust to parameter settings. Nonetheless, the Kolmogorov-Smirnov test (Table 4.2) finds a significantly superior performance of ARPS compared to other parameter value selection methods on all ten problems.

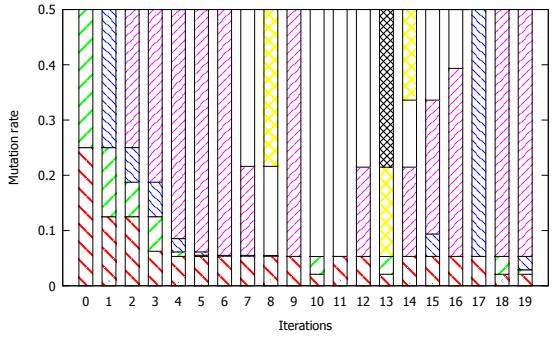
In ARPS, the change in the ranges of the intervals of crossover and mutation rates during 20 iterations is depicted in Figures 7.8-7.9. At the beginning of the optimisation process, all intervals are equal. The selection of the parameter values



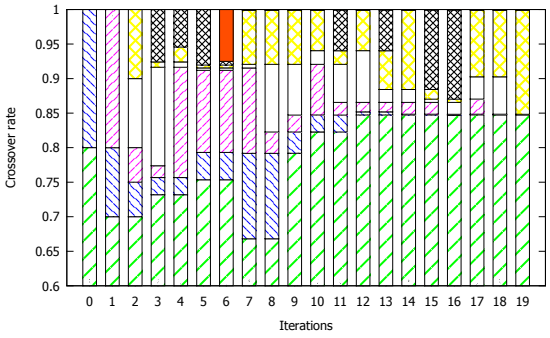
(a) Mutation rate (Royal Road).



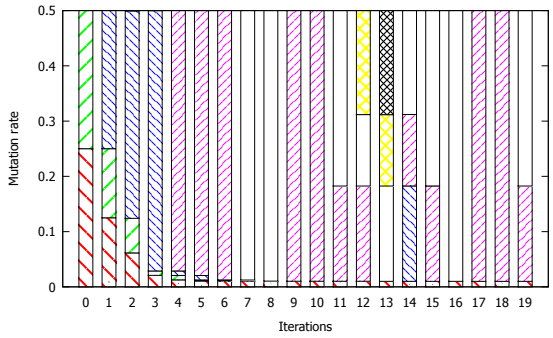
(b) Crossover rate (Royal Road).



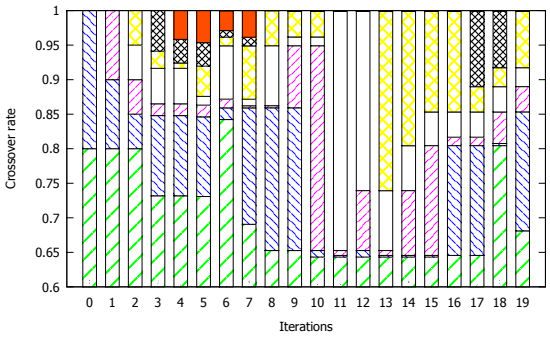
(c) Mutation rate (KC20-2fl-5rl).



(d) Crossover rate (KC20-2fl-5rl).

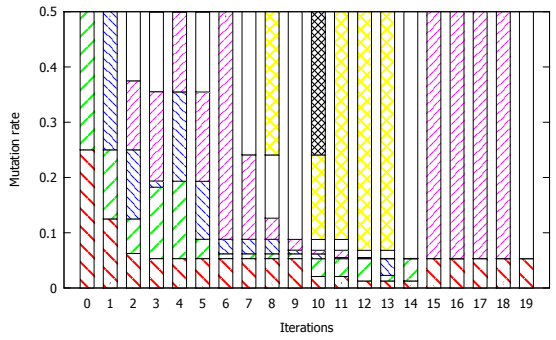


(e) Mutation rate(KC30-3fl-3uni).

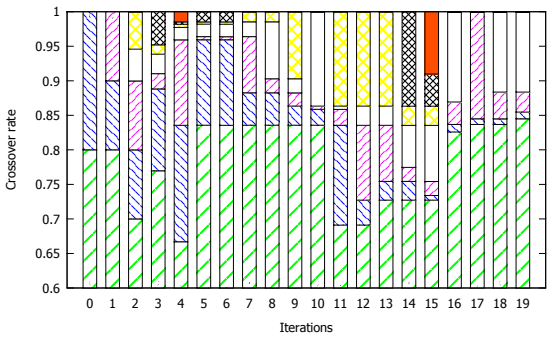


(f) Crossover rate (KC30-3fl-3uni).

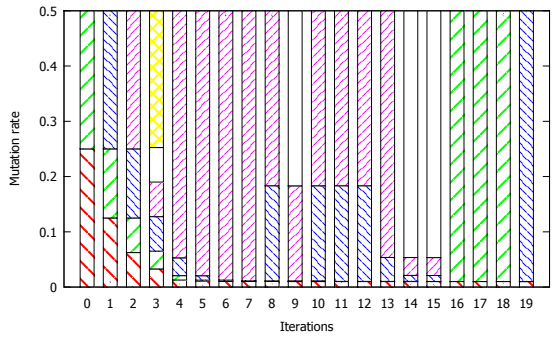
Figure 7.8: Change of parameter ranges for the optimisation of the Royal Road Problem and the multiobjective Quadratic Assignment Problem with an EA that uses ARPS during 20 iteration.



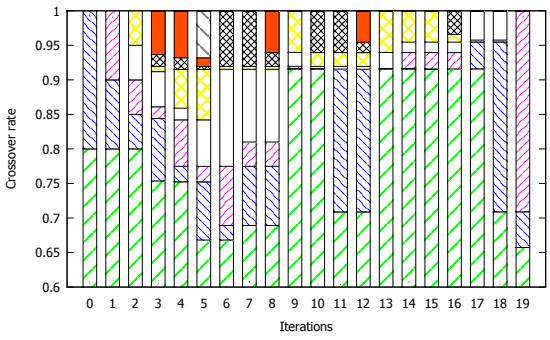
(a) Mutation rate (BUR26A).



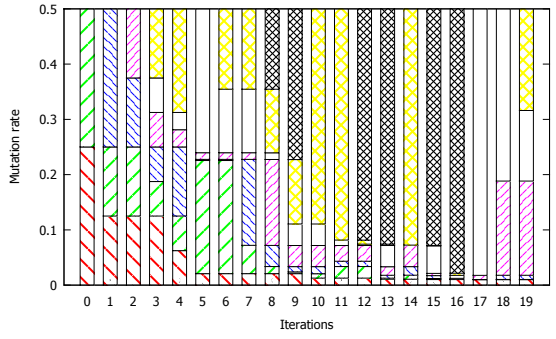
(b) Crossover rate (BUR26A).



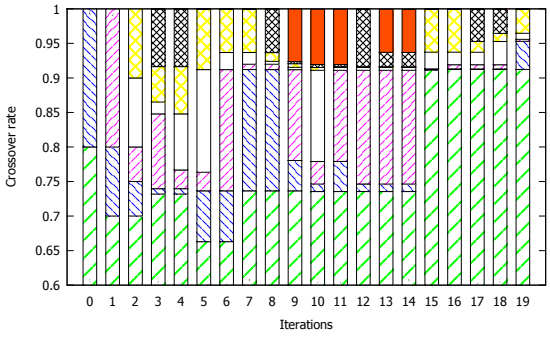
(c) Mutation rate (BUR26B).



(d) Crossover rate (BUR26B).



(e) Mutation rate (BUR26E).



(f) Crossover rate (BUR26E).

Figure 7.9: Change of parameter ranges for the optimisation of the Quadratic Assignment Problem with an EA that uses ARPS during 20 iteration.

is based on the assigned probabilities. The bigger the interval becomes, the smaller is the chance of the values in that interval to be selected. Accordingly, the most successful values for each iteration are to be placed in the smallest interval. We can clearly see that the behaviour of the adaptive range parameter value selection is different for different problems and different parameters.

From the bar diagrams we can see that the smaller of the multiobjective Quadratic Assignment Problem instances (KC20-2fl-5rl) depicted in Figure 7.8c and 7.8d are best optimised with a very small mutation rate throughout the process, whereas the RR problem (Figure 7.8a and 7.8b) seems to require slightly higher mutation rates (approx. 0.2) at the start but toward the end of the process the level ranges are not as focussed. A different observation can be made regarding the optimal mutation rates for the QAP instances (Figure 7.9); there, the most successful mutation rates are clearly very low at the end of the optimisation process.

The levels of crossover rate develop quite differently compared to mutation rate. Higher rates are often more successful towards the end of the optimisation process which runs somewhat contrary to popular opinion that crossover rates should decrease towards the end of the optimisation process so as not to disturb solutions with high quality. For some problems, both crossover rates from the upper third and from the lower third of the overall range seem beneficial at the same time. The mutation/crossover range analysis shows that high-performing ranges are sometimes absorbed (merged) into very large intervals, making it difficult for the algorithm to re-establish small, promising areas within the range. There may be a potential for further optimisation of the range adaptation in this respect.

7.4 Summary

In this chapter we presented a new parameter value selection strategy called Adaptive Range Parameter Selection (ARPS). ARPS, different from state-of-the-art parameter value selection strategies, uses dynamic ranges for parameter values which are adjusted as the optimisation process progresses. The method rewards best-performing parameter ranges by dividing them into two new ranges and assigning each of them the same selection probability as the parent range. As a result, the selection probability of all values within that range is doubled. On the other hand, the worst performing range is merged with one of the neighbouring ranges, reducing its selection probability, but not discarding it.

According to our knowledge, the best-performing approaches with the same functionality are AP, PM and DMAB. The new approach clearly outperforms the other parameter control methods. As the problem difficulty increases, so does the difference in result quality produced by ARPS compared to the benchmark approaches. The mutation/crossover range analysis shows that high-performing ranges are sometimes absorbed (merged) into very large intervals, making it difficult for the algorithm to re-establish small, promising areas within the range. There may be a potential for further optimisation of the range adaptation in this respect.

Part III

Practical Validation

Chapter 8

Architecture Optimisation in Embedded Systems

8.1 Introduction

Embedded software plays a significant role in performing a variety of tasks in automotive, avionic, medical, defence, railway, and telecommunication systems [146, 150]. The design and development of today's embedded systems is a complex task, involving several decisions regarding the architecture of the system. Examples of architecture-level decisions include the deployment of software components to hardware platform and allocating redundancies for software and hardware elements. The decisions made during architecture design have significant implications for the quality of the final system [62, 97, 119]. For instance, the decision of running two components which implement safety-critical functions on the same hardware resource may lead to a violation of safety requirements due to common cause failures, i.e. both components will fail because of the same cause. Usually, this scenario is avoided. Another example is the allocation of redundant components, which is a widely used method for the reliability improvement. Last but not least, selecting the suitable components to perform specific tasks can affect system quality attributes, such as performance, reliability, and security. Due to the life- and mission-critical nature

of the majority of these systems, quality attributes such as availability, reliability, performance, safety and security are very important.

The ever-increasing complexity of software systems introduces a big challenge for systems engineers, who have to choose from a growing number of design options resulting in a design space that is beyond the human capabilities of understanding, making the architecture design problem a challenging task [146, 150, 73]. The requirement for automating the task of architecture design in embedded systems has been recognised by previous research [25, 150, 21, 119, 34], and a plethora of architecture optimisation approaches have been developed, such as Evolutionary Algorithms (EA) [97, 193, 119, 147]. To handle the complexity of the task, the optimisation approaches limit the variability of architectural decisions, optimising the architecture from a particular point of view, such as component deployment optimisation [126, 86, 18, 50, 62, 97, 193, 119, 147] and redundancy allocation optimisation [111, 193, 107, 106, 127]. The use of Evolutionary Algorithms for the architecture optimisation of embedded systems has steadily grown over the years [21, 50, 62, 147, 193, 119], proving that EAs can successfully be applied to solving complex combinatorial problem spaces [188]. It has also been observed that EAs are robust algorithms, producing result qualities with low deviation [157].

The main reasons why Evolutionary Algorithms are often applied in the Software Engineering domain is their ability to solve any complex problem if it is configured in the appropriate way. However, this is not an easy task due to the large number of parameter values and combinations that have to be configured. This task becomes even more difficult because of the non-linear ways EA parameters interact [43]. Exploring all the parameter configurations is very time consuming and computationally expensive [43]. It follows that finding the right EA configuration for a specific problem is an optimisation problem in its own that practitioners have to face.

The requirement for configuring optimisation algorithms in the Software Engineering domain is only one example of practitioners having to do experts' tasks. This requirement extends beyond the Software Engineering domain, however we use the architecture optimisation problem to test the applicability of the proposed optimisation approach to a real world application. We apply the Evolutionary Algorithm with the Adaptive Parameter Control (APC) to two optimisation problems in architecture design and use a case-study from the automotive industry to demonstrate the results.

8.2 Architecture Design and Optimisation

The architecture of an embedded system represents a model or an abstraction of the real elements of the system and their properties, such as software components, hardware units, interactions of software components, and communications between hardware units. Formally, we define the software elements as a set of software components, denoted as $\mathcal{C} = \{c_1, c_2, ..., c_n\}$, where $n \in \mathbb{N}$. The software components are considered as black boxes [91], i.e. not modifiable and with unknown internal structure, but with a description of externally visible parameters. The software components interact to implement a set of *services*, which define the functional units accessed by the user of the system. Figure 8.1 depicts an example of four components and their interactions.

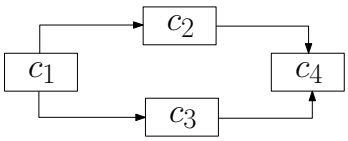


Figure 8.1: Software elements and interactions.

Each *service* is initiated in one software component (with a given probability), and during its execution uses many other components (possibly shared with other services), connected via communication links. For each service, the links are assigned with a transition probability. This view follows the Kubat model [105], who expresses the software architecture as a Discrete-Time Markov Chain (DTMC), where vertices represent the execution of software components, and arcs enumerate the probability of transferring the execution flow to the next component. The DTMC model is used to calculate specific quality attributes of the system. The model depicted in Figure 8.2 is constructed by using the software elements shown in Figure 8.1.

The hardware architecture is composed of a distributed set of hardware hosts, denoted as $\mathcal{H} = \{h_1, h_2, ..., h_m\}$, where $m \in \mathbb{N}$, with different capacities of memory,

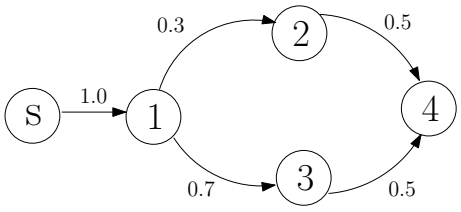


Figure 8.2: Software elements and interactions.

processing power, access to sensors and other peripherals. The hardware hosts are connected via network links denoted as $\mathcal{N} = \{n_1, n_2, ...n_s\}$. An example of three hardware hosts and the network that is used for communication is depicted in Figure 8.3.

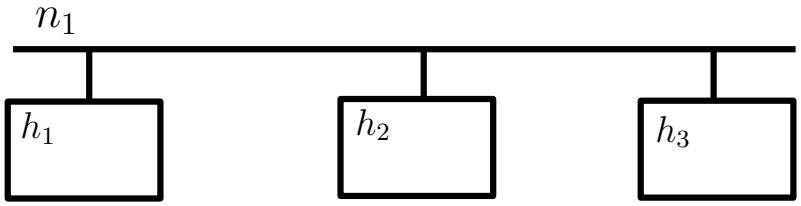


Figure 8.3: Hardware elements and communication network.

Given the set of architectural elements, a series of decisions have to be taken in order to obtain the final architecture of the embedded system, such as how to deploy the software components to the hardware resources called Component Deployment Problem (CDP), and how to assign redundancy levels to safety-critical components, called the Redundancy Allocation Problem (RAP). Current research [62, 97, 119] shows that these decisions have significant implications for the likelihood that the system achieves the desired quality attributes. In general, quality attributes are non-functional characteristics of a system, the combination of which constitutes the overall quality of the system as defined by the IEEE 1061 standard [88]. Examples of quality attributes are reliability, safety, availability, cost, energy consumption and performance. For every design decision, several quality attributes are considered, which due to their conflicting nature, are optimised simultaneously.

8.2.1 Redundancy Allocation Optimisation

Component redundancy allocation is a widely used reliability improvement technique for dependable embedded systems [73, 127]. The Redundancy Allocation Problem (RAP) has shown to be NP-hard [30], hence most of the approaches in the literature use stochastic algorithms to optimise this problem.

Coit et al. [31] solve the Redundancy Allocation Problem (RAP) by using functionally similar components such that if one component fails, the redundant part performs the required functionality without a system failure. The optimisation problem is defined as the minimisation of cost while satisfying a user defined system reliability level, which is handled as a constraint. The problem is optimised by using a Genetic Algorithm. Kulturel-Konak et al. [106] use a Tabu Search with a penalty function for infeasible solutions, which allows the exploration of infeasible regions. Ant Colony Optimisation (ACO) is another optimisation technique used to solve the redundancy allocation problem by Liang et al. [110], who employ an elitist strategy to preserve good solutions and a mutation operator to search in unexplored areas of the search space. The enhanced ACO performs better compared to an ACO without the mutation operator and the elitism.

All the mentioned approaches model the redundancy allocation problem as a singleobjective problem, which minimises cost while satisfying the predefined reliability criteria. In contrast, Grunske [71] addressed the redundancy allocation problem as a multiobjective problem to optimise reliability and minimise weight. We follow a similar approach, since the allocation of redundancy levels affects conflicting objectives, which have to be optimised simultaneously.

For the redundancy allocation, we use the *hot spare* design topology, in which all component redundancies are active at the same time, mimicking the execution of the original component. With the NMR extension the system employs a decision

mechanism in the form of majority voting when multiple replicas deliver their results to the entry gate of a component. In this configuration, each component with its parallel replicas is regarded as a *subsystem*.

Formally, an architecture alternative for the redundancy allocation problem is denoted as ra. The set of all redundancy allocation candidates ra is denoted as $RA = \{ra \mid ra : \mathcal{C} \rightarrow N_{RA}\}$, where $\mathcal{N}_{RA} = \{n \mid 0 \leq n \leq nMax, n \in \mathbb{N}_0\}$ delimits the redundancy level of a component. Note that, since $\mathcal{C}$ and $\mathcal{N}$ are finite, RA is also finite. A visual representation of an architecture alternative for the Redundancy Allocation Problem is depicted in Figure 8.4.

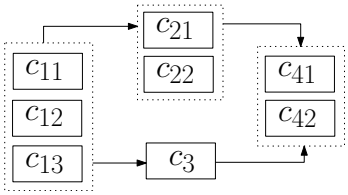


Figure 8.4: Redundancy Allocation Problem.

The redundancy allocation problem is optimised by using an EA with a specialised solution encoding which maps a redundancy level to each software component. This problem has many quality-related aspects and is therefore modelled as a multiobjective problem.

Solution representation

In order to apply an Evolutionary Algorithm to RAP, each solution is encoded as $ra_i = [ra_i(c_1), ra_i(c_2), ..., ra_i(c_n)]$, where $ra_i(c_j)$ represents the redundancy level for component c_j in the architecture alternative ra_i.

Genetic operators

The crossover and mutation operators are used to find new architecture alternatives for RAP. The crossover operator creates two new solutions $ra'_i, ra'_j \in R$ from

two parents $ra_i = [ra_i(c_1), ra_i(c_2), ..., ra_i(c_n)]$ and $ra_j = [ra_j(c_1), ra_j(c_2), ..., ra_j(c_n)]$ coming from existing population by recombining the redundancy levels, i.e. for a randomly selected index k: $ra'_i = [ra_i(c_1), ..., ra_i(c_{k-1}), ra_j(c_k), ..., ra_j(c_n)]$ and $ra'_j = [ra_j(c_1), ..., ra_j(c_{k-1}), ra_i(c_k), ..., ra_j(c_n)]$.

Similarly, a single-point mutation produces a new solution ra'_i from existing ra_i by switching the redundancy levels of two software components, i.e. for randomly selected k, l: $ra'_i = [ra_j(c_1), ..., ra_j(c_l), ..., ra_j(c_k), ..., ra_j(c_n)]$ while the original is $ra_i = [ra_i(c_1), ..., ra_j(c_k), ..., ra_j(c_l), ..., ra_j(c_n)]$.

8.2.2 Component Deployment Optimisation

The Component Deployment Problem (CDP) refers to the allocation of software components to the hardware nodes, and the assignment of inter-component communications to network links. The way the components are deployed affects many aspects of the final system, such as the processing speed of the software components, how much hardware is used or the reliability of the execution of different functionalities [129, 4, 147, 136, 126, 120, 169, 62], which constitute the quality attributes of the system. Some of the methods focus on the satisfaction of the constraints or user requirements [96, 27, 121], others aim at finding optimal deployments or at least candidates that are near-optimal [126, 169, 62, 145, 21], often in combination with the given constraints [136, 126, 62, 21, 117]. In our approach, conflicting quality attributes are considered which are optimised simultaneously. Moreover, constraints are checked for satisfaction in every generated solution.

From an optimisation perspective, the component deployment problem is similar to the generalized Quadratic Assignment Problem (gQAP), where there is no restriction that one location can accommodate only a single equipment. Formally, the component deployment problem is defined as $D = \{d \mid d : \mathcal{C} \rightarrow \mathcal{H}\}$, where D

is the set of all functions assigning components to hardware resources. One possible solution for the component deployment problem of the software and hardware architectures introduced in the previous section is depicted in Figure 8.5.

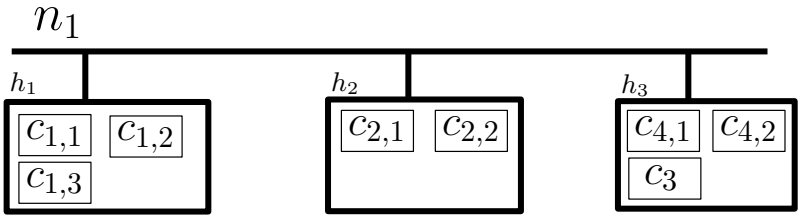


Figure 8.5: Component Deployment Problem.

Solution representation

The component deployment problem uses an EA with a specialised solution encoding which maps software components to hardware resources. Each architecture alternative of the component deployment problem is encoded for the optimisation process as $d_i = [d_i(c_1), d_i(c_2), ..., d_i(c_n)]$, where $d_i(c_j)$ represents the hardware host used to deploy component c_j in the deployment alternative d_i.

Genetic operators

The genetic operators are applied to this representation of the solutions in order to generate new architecture alternatives. The crossover operator combines the allocation lists of two solutions. Formally, crossover creates two new solutions $d'_i, d'_j \in D$ from two parents $d_i = [d_i(c_1), d_i(c_2), ..., d_i(c_n)]$ and $d_j = [d_i(c_1), d_i(c_2), ..., d_i(c_n)]$ coming from existing population by recombining the allocation of components, i.e. for a random k: $d'_i = [d_i(c_1), d_i(c_2), ..., d_i(c_{k-1}), d_j(c_k), ..., d_j(c_n)]$ and $d'_j = [d_j(c_1), d_j(c_2), ..., d_j(c_{k-1}), d_i(c_k), ..., d_i(c_n)]$.

The mutation operator exchanges the host allocations of two randomly chosen components. Formally, mutation produces a new solution d'_i from existing d_i by switching the mapping of two components, i.e. for randomly selected k, l:

$d'_i = [d_i(c_1), d_i(c_2), ..., d_i(c_k)..., d_i(c_l), ..., d_i(c_n)]$ while the original solution is $d_i = [d_i(c_1), d_i(c_2), ..., d_i(c_l)..., d_i(c_k), ..., d_i(c_n)]$. The problem definition does not allow for duplications, and a repair operation follows the crossover/mutation move.

8.3 ArcheOpterix

To solve the architecture design problem in embedded systems we have implemented an open source framework called ArcheOpterix [3] (`http://mercury.it.swin.edu.au/g_archeopterix/`). ArcheOpterix is a generic platform for modelling, evaluating and optimising embedded systems. The main modules of ArcheOpterix are shown in Figure 8.6. The current implementation supports several quality attributes (service reliability [129, 130], response time [127], data communication overhead [3, 4, 5], cost [127], data transmission reliability [3, 4, 5], energy consumption [128], and scheduling length [138]), different optimisation algorithms (Non-dominated Sorting Genetic Algorithm-II [41] and Pareto-Ant Colony Optimisation [4]) and various parameter control mechanisms (Probability Matching (PM) [181], Adaptive Pursuit (AP) [181] and Dynamic Multi-Armed Bandit (DMAB) [56], Predictive Parameter Control [5], etc.).

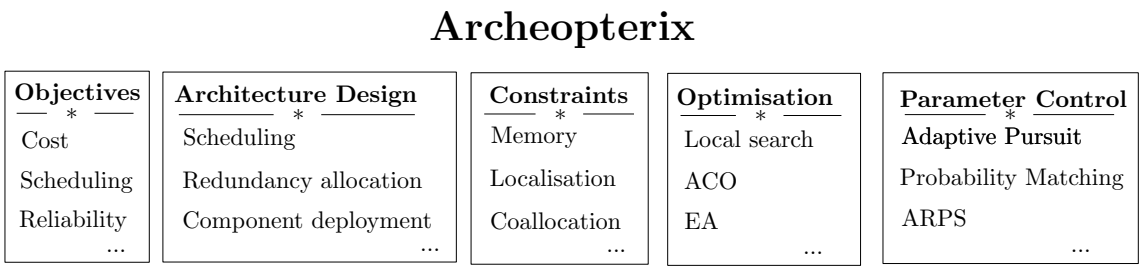


Figure 8.6: ArcheOpterix framework.

The architecture design decisions supported in ArcheOpterix are component deployment, redundancy allocation, etc. Problem-specific constraints are implemented to check for the feasibility of produced solutions, such as localisation constraint [129, 3, 4, 138, 5], collocation [129, 3, 4, 138], memory [129, 3, 4, 138, 5], and redundancy levels [127, 128]. ArcheOpterix supports distributed processing in grid and cloud platforms and parallel optimisation algorithms.

8.4 Case-Study: Automotive Embedded Systems

Today more than 80% of innovations in a car come from software systems. Software adds distinguishing features to car models and allows hardware to be reused. With the increase of the number of functions performed by software, the automotive embedded systems are becoming more complex with many design options to choose from. For instance, a simple power train control application has 3488 possible component realisations by instantiating different algorithms and their variants. A typical car consists of around 80 electronic fittings. Simple 'yes, no' decisions for each function yield approximately 2^{80} variants to be ordered and produced for a car. Thus there is a large amount of variants that should be handled technically [26], a task that can take years for the system designer.

Both the software and the hardware architecture form the embedded architecture of an automotive system, the design of which involves a variety of manual and automatic decisions. Component deployment and redundancy allocation are two important design decisions that can be automated. The way the architecture of automotive systems is designed affects different quality attributes of the system [62, 96, 119], such as safety, security, availability, reliability, maintainability, performance and temporal correctness [72, 151], which often are conflicting with each other.

Given the conflicting quality attributes and the design options which grow exponentially with the number of components, the design space extends beyond the human capabilities of understanding. To deal with the complexity of today's automotive systems, automatic methods are required, which can support engineers in exploring the design space and finding good solutions; a problem that has already been recognised by the industry [25, 151]. We demonstrate the application of the proposed adaptive parameter control for Evolutionary Algorithms to the design of an automotive embedded system.

8.4.1 Hardware Architecture

In the automotive industry, an existing hardware topology is usually used, because car models remain the same through parts of their lifetimes. The hardware is used to run software components which form the basis of the increasingly sophisticated functionality of contemporary cars. The hardware model of an automotive embedded system is composed of a distributed set of Electronic Control Units (ECUs), which have different capacities of memory, processing power, access to sensors, etc. Automotive embedded systems closely interact with the physical environment, typically via sensors and actuators. All the hardware modules (ECUs, sensors, actuators) are connected through communication buses, which are shared among the hardware units, such as Ethernet, CAN, or FlexRay. The hardware modules and the communication buses, as depicted in Figure 8.7, form the hardware architecture of an automotive system.

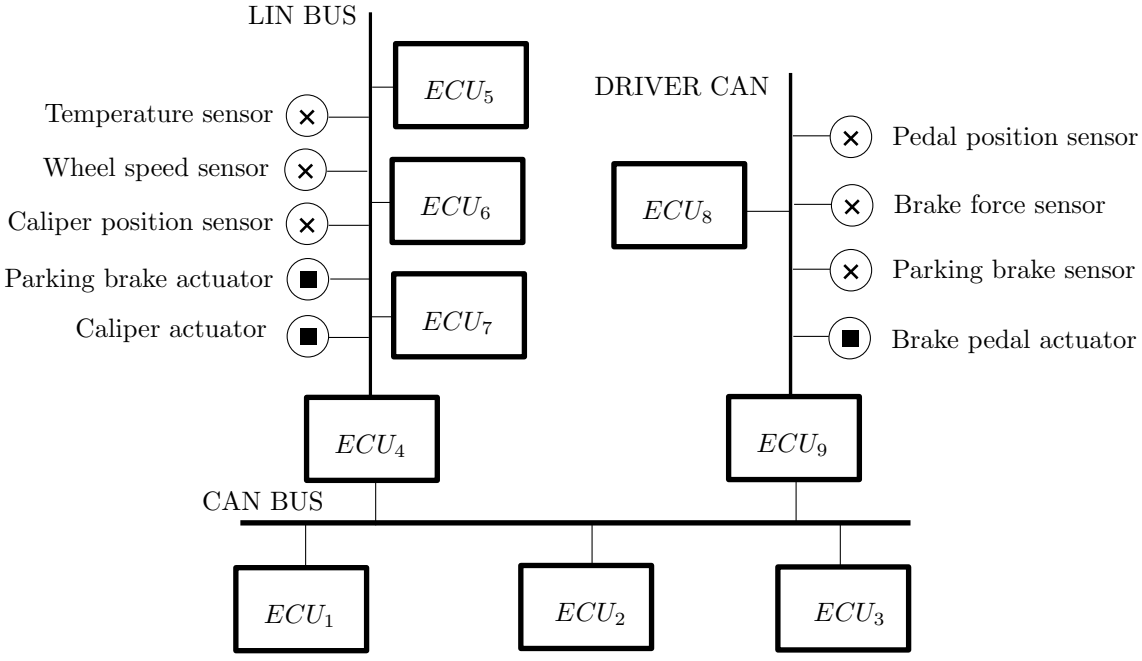


Figure 8.7: The hardware architecture.

Many types of buses can be present, having different characteristics of data rates and reliability. For example, for the Brake-by-wire (BBW) system a highly reliable bus is used due to the safety-critical nature of this system, whereas a less reliable bus may be sufficient for multi-media streaming. Each ECU has access to different sensors and actuators, which impose localisation constraints for the software components. In other words, software components that read from a specific sensor or write to a particular actuator, have to be deployed to an ECU that has access to the sensor or actuator.

The hardware elements are annotated with different properties required for the calculation of the quality attributes. The communication among the hardware hosts is achieved by the network links, which have different characteristics of data rates, reliability, etc. Figure 8.7 depicts three different network links: a LIN bus, a Driver CAN and a CAN bus. A description of the properties of hardware units and network links are depicted in Table 8.1.

Table 8.1: Properties of the hardware elements.

Annotation	**Definition**	**Description**
$ps(h_i)$	$\mathcal{H} \rightarrow \mathbb{N}$	Processing speed of the host ($MIPS$).
$mh(h_i)$	$\mathcal{H} \rightarrow \mathbb{N}$	Memory capacity of host h_i.
λ	$\mathcal{N} \rightarrow \mathbb{R}$	Failure rate of the network link.
bw	$\mathcal{N} \rightarrow \mathbb{N}$	the bandwidth of a network link.
tt	$\mathcal{N} \rightarrow \mathbb{N}$	the transfer time for a link per execution.

8.4.2 Software Architecture

The software layer of an automotive embedded system consists of a high number of lightweight components, representing the logical blocks of system functionality (typically in a low-level programming language). This forms the software architecture

of the system. Automotive software has very diverse functionality, ranging from entertainment software to safety-critical, real-time control software. We illustrate the software architecture of an automotive system by using the Brake-by-wire (BBW) service, which is a real-time system that performs safety-critical tasks.

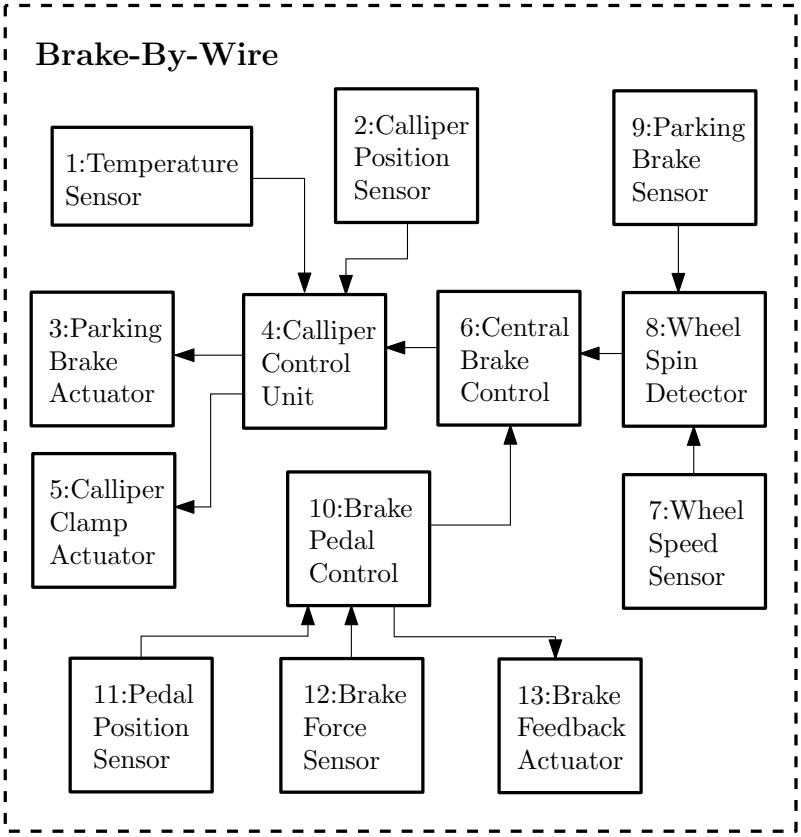


Figure 8.8: The software architecture of the Brake-by-wire system.

BBW technology is a recent innovation in the automotive industry, which replaces the traditional mechanical and hydraulic control systems with electronic control systems. A diagram of the BBW software model is shown in Figure 8.8, where each box represents a software component and the connections among them correspond to the interactions. The direction of the arrows shows the sequence of the execution of the components.

A BBW pedal is usually equipped with several sensors which provide information

about the driver's brake request. The *Pedal Position Sensor* (PPS) and the *Brake Force Sensor* (BFS) measure the force applied by the driver to the brakes and the current position of the brakes. Upon measurement of the driver's brake request, the brake demand is sent to the *Brake Pedal Control* (BPC) via the communication network. The BPC transfers the brake request to the *Central Brake Control* (CBC), which generates four independent brake commands and sends them to the *Calliper Control Unit* (CCU) in each wheel. These commands are usually in the form of four clamp forces that are generated between each of the four brake discs and their corresponding brake pads. Each CCU includes a *Calliper Clamp Actuator* (CCA) to clamp the brake pad toward the braking disc and a *Parking Brake Actuator*. The CCU also processes additional inputs from several sensors, such as the *Temperature Sensor* (TS) and the *Calliper Position Sensor*, which regulate the brake command execution.

There are also different sensors and a controller to tune the actual brake force to the desired clamp force received from the *Central Brake Control* (CBC), such as the *Parking Brake Sensor* (PBS) and the *Wheel Speed Sensor* (WSS). After the brake request has been applied, a message is generated by the *Brake Feedback Actuator* (BFA) which notifies the driver. A description of the properties of software elements is given in Table 8.2.

8.4.3 Configuration of the Case-study

The property of the hardware and software elements described in Tables 8.1 and 8.2 are configured using values from a real world applications of an automotive embedded system. The case-study was built with inputs from the Australian Automotive Industry via the cooperate research centre (AutoCRC), and Volvo Sweden. The configuration of the hardware and software architecture are given in Tables 8.3-8.7.

Table 8.2: Properties of the software elements.

Annotation	**Definition**	**Description**
$ext(c_i)$:	$\mathcal{C} \rightarrow \mathbb{N}$	Estimated time taken for a single execution of component c_i.
$q_0(c_i)$	$\mathcal{C} \rightarrow \mathbb{R}$	Mean execution initialisation probability for component c_i.
$mc(c_i)$	$\mathcal{C} \rightarrow \mathbb{N}$	Memory size required to run component c_i.
$\lambda(c_i)$	$\mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}$	Failure rate of component c_i.
$cost(c_i)$	$C \rightarrow \mathbb{N}$	The price associated with a single component; specified in (\$)s.
$p(c_j, c_i)$	$\mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}$	Transition probability from component c_j to component c_i.
$es(c_j, c_i)$	$\mathcal{C} \times \mathcal{C} \rightarrow \mathbb{N}$	Message (event) size transferred from component c_j to component c_i.
$cf(c_i, c_j)$	$\mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}$	Communication frequency between components c_i and c_j.
$lr(c_i, h_j)$	$\mathcal{C} \times \mathcal{H} \rightarrow \{0, 1\}$	Localisation restrictions. The restriction $lr(c_i, h_j) = 1$ if component c_i has to be deployed to hardware host h_j, otherwise $lr(c_i, h_j)$ is equal to 0.
$cr(c_i, c_j)$	$\mathcal{C} \times \mathcal{C} \rightarrow \{0, 1\}$	Collocation restrictions. Restriction $cr = 1$ if c_i has to be deployed in the same hardware unit as c_j, and $cr = 0$ if there is no such restriction.

The software architecture is deployed to the hardware architecture described in Section 8.4.1, to realise the BBW system. This process involves two main design steps which are optimised individually, described in the following sections. First a redundancy level is decided for each component. Then the software components and their redundancies are allocated to the ECUs. In each step, different quality attributes are optimised.

Table 8.3: Values of hardware nodes for the BBW system.

Host	**mh**	**ps**	**fr**	**lr**
ECU_1	512	4	0.000400	$0,0,1,1,1,1,0,0,0,1,0,0,1$
ECU_2	1024	6	0.000400	$0,0,1,1,1,1,0,0,0,1,0,0,1$
ECU_3	512	2	0.000020	$0,0,1,1,1,1,0,0,0,1,0,0,1$
ECU_4	1024	2	0.000100	$1,1,1,0,1,0,1,1,0,0,0,0,0$
ECU_5	510	11	0.000800	$1,1,1,0,1,0,1,1,0,0,0,0,0$
ECU_6	1024	11	0.000200	$1,1,1,0,1,0,1,1,0,0,0,0,0$
ECU_7	1024	8	0.000060	$1,1,1,0,1,0,1,1,0,0,0,0,0$
ECU_8	1024	7	0.000040	$0,0,0,0,0,0,0,0,1,1,1,1,1$
ECU_9	1024	8	0.000800	$0,0,0,0,0,0,0,0,1,1,1,1,1$

Table 8.4: Values of communication links for the BBW system.

Link	fr	bw	tt	r
Lin bus	3.00E-05	128	10	0.9
Driver CAN	1.20E-04	64	2	0.8
CAN bus	4.00E-05	64	4	0.8

8.4.4 Redundancy Allocation Optimisation

Redundancy Allocation (RA) determines the number of redundancies that have to be implemented for each software components. RA is one of the well-known system reliability improvement techniques [110, 32, 107, 73]. Reliability is one of the crucial aspects that should be considered when designing embedded architectures of dependable, safety critical systems such as in the automotive domain [2]. However, employing redundant components can have a negative influence in the other quality attributes. Employing multiple redundancies in software level adds communication and processing overhead, as well as requirements for additional hardware (e.g. sensors). This problem has been addressed mostly in component based software ar-

Table 8.5: Values of software components for BBW system.

Component		mc	q_0	ext	cost
c_1	Temperature Sensor (TS)	64	0.108	12	25
c_2	Calliper Position Sensor (CPS)	128	0.182	10	30
c_3	Parking Brake Actuator (PBA)	64	0	2	20
c_4	Calliper Control Unit (CCU)	512	0	20	30
c_5	Calliper Clamp Actuator (CCA)	256	0	5	40
c_6	Central Brake Control (CBC)	1024	0	10	30
c_7	Wheel Speed Sensor (WSS)	64	0.1	12	35
c_8	Wheel Spin Detector (WSD)	128	0.2	8	20
c_9	Parking Brake Sensor (PBS)	128	0.1	4	30
c_{10}	Brake Pedal Control (BPC)	512	0	6	35
c_{11}	Pedal Position Sensor (PPS)	64	0.13	10	30
c_{12}	Brake Force Sensor (BFS)	128	0.18	10	35
c_{13}	Brake Feedback Actuator (BFA)	64	0	4	25

Table 8.6: Values of interactions between components in the BBW system.

c_i	c_0	c_1	c_3	c_3	c_5	c_7	c_6	c_8	c_9	c_{10}	c_{11}	c_9
c_j	c_3	c_3	c_2	c_4	c_3	c_5	c_7	c_7	c_5	c_9	c_9	c_{12}
p	1	1	0.3	0.7	1	1	1	1	0.6	1	1	0.4
cl	2	2	2	2	2	1	2	1	2	1	2	2
ext	0.2	0.3	0.1	0.4	0.3	0.2	0.6	0.4	0.2	0.3	0.5	0.4
es	10	12	12	10	12	6	12	6	12	8	10	6
cf	1	1	0.3	0.7	1	1	1	1	0.6	1	1	0 .4

Table 8.7: Values of collocation restrictions between components in the BBW system.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}
c_1	0	0	0	0	0	0	0	0	0	0	0	0	0
c_2	0	0	0	0	0	0	0	0	0	0	0	0	0
c_3	0	0	0	0	0	0	0	0	0	0	0	0	0
c_4	0	0	0	0	0	0	0	0	0	0	0	0	0
c_5	0	0	0	0	0	0	0	0	0	0	0	0	0
c_6	0	0	0	0	0	0	0	0	0	0	0	0	0
c_7	0	0	0	0	0	0	0	1	0	0	0	0	0
c_8	0	0	0	0	0	0	1	0	0	0	0	0	0
c_9	0	0	0	0	0	0	0	0	0	0	0	0	0
c_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0
c_{11}	0	0	0	0	0	0	0	0	0	0	0	1	0
c_{12}	0	0	0	0	0	0	0	0	0	0	1	0	0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0

chitecture development, where the trade-offs between cost and reliability were investigated. Apart from the cost, when redundancy is employed in automotive systems, additional overheads incur into the response time. Response time is also a critical and important aspect of automotive systems, not only in safety critical sub-systems but also in user interaction sub-systems, such as the navigation system [2]. Hence, to quantify the quality of a single redundancy allocation architecture ra, we define three objective functions $F : RA \rightarrow \mathbb{R}^2$, where $F(ra) = (rt(ra), r(ra), cost(ra))$ s.t. $rt(ra)$ is the the response time of ra, defined by Equation 8.1, $r(ra)$ denotes the reliability (probability of failure-free operation) of ra, defined in Equation 8.7 and $cost(ra)$ is the total cost of the system with the redundancy levels assigned to components.

Response time

The response time has already been the focus of an extensive research in the embedded systems [169, 61, 87, 170]. In the prediction of response time for each redundancy allocation candidate, we use the DTMC model based approach presented in [170]. We describe the system behaviour as a Markov model with probabilities of execution transfer between components together with probabilities of execution initialisation at each component. The response time rt for a redundancy allocation candidate ra is calculated as follows:

$$rt(ra) \; = \; \sum_{i \in n} ext(c_i) \cdot exv(c_i) \tag{8.1}$$

where $ext : C \rightarrow \mathbb{N}$ is the estimated time taken for a single execution of a component, and $exv : C \rightarrow \mathbb{R}$ quantifies the expected number of executions of a component during the system execution. This can be computed by solving the following equation [105]:

$$exv(c_i) = q_0(c_i) + \sum_{j \in \setminus} (exv(c_j) \cdot p(c_j, c_i)) \tag{8.2}$$

where $p(c_j, c_i)$ denotes the transition probability from component c_j to component c_i, and $q_0(c_i)$ represents the mean execution initialisation probability for component c_i.

Reliability

We employ a well-established method of reliability estimation presented by Kubat [105, 68]. In estimating the reliability of a single component, we assume that failure of a component has an exponential distribution [171], which is characterised by failure rate parameter λ. First, we calculate the *reliability of a single component*

c_i per visit as define by Shatz et al. [171]:

$$r(c_i) = e^{-\lambda(c_i) \cdot ext(c_i)} \tag{8.3}$$

When the redundancy levels are employed, the reliability of a component with its replicas connected in parallel for the architectural alternative ra is computed as:

$$r(c_{i,rep}) = 1 - (1 - r(c_i))^{ra(c_i)+1} \tag{8.4}$$

Similarly, the reliability of transferring a message from component c_i to component c_j is calculated as follows:

$$r(c_i, c_j) = e^{-\lambda(c_i,c_j) \cdot tt(c_i,c_j)} \tag{8.5}$$

where $tt(c_i, c_j)$ is the transfer time for a link per execution $\lambda(c_i, c_j)$ is the failure rate in the communication:

In consideration of a redundancy allocation ra, the presence of multiple senders increases the reliability (due to the tolerance against commission and value failures), which is calculated as follows:

$$r(c_{i,rep}, c_j) = 1 - (1 - r(c_i, c_j))^{ra(c_i)+1} \tag{8.6}$$

The reliabilities of individual system elements (subsystems and links) per a single visit are used to compute the reliability of the system execution based on the expected number of executions [68, 105]:

$$r(ra) \approx \prod_{i \in n} (r(c_{i,rep}))^{exv(c_i)} \tag{8.7}$$

Cost

The cost of the system for each architecture alternative is evaluated as the sum of the costs of individual components and the respective redundancies as follows:

$$cost(ra) \;=\; \sum_{i \in n} cost(c_i) \cdot (ra(c_i) + 1) \tag{8.8}$$

8.4.5 Component Deployment Optimisation

Another challenge for the automotive industry remains the deployment function which relates hardware to software. The hardware, software and the deployment function should be a concrete realisation of the logical architecture, which describes the interaction between the logical components. The software components run on ECUs and communicate by bus systems. The deployment process determines the ECU to which each software component is allocated. Generally, the same platform can be used for different models and different products at the same time, by integrating new functionality in an iterative process [186], which makes the implementation of new innovative functions the main focus of the automotive electronics [162]. The decisions regarding the deployment architecture of a car affect the quality of the final systems. For example, consider the two deployment architectures shown in Figure 8.9.

In the first deployment architecture, frequently interacting software components have been deployed to the same hardware resource, which result in an architecture with a lower communication overhead compared to the second deployment architecture. However, the first deployment architecture has a longer scheduling length due to the sequential execution of the components that require information from each other.

To quantify the quality of a single deployment architecture d, we define three

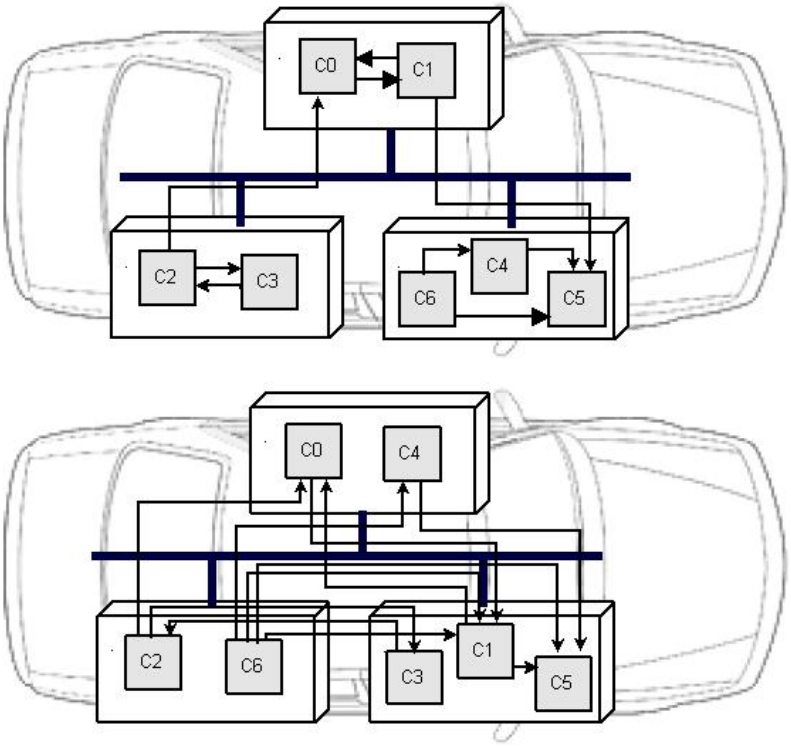


Figure 8.9: The software architecture of Brake-by-wire system.

objective functions $F : D \rightarrow \mathbb{R}^3$, where $F(d) = (sl(d), co(d), dtr(d)$ s.t. $sl(d)$ is the quantification of the scheduling length of d, defined by Equation 8.9, $co(d)$ denotes the communication overhead of d, defined in Equation 8.10 and $dtr(d)$ denotes the data transmission reliability defined in Equation 8.11.

Scheduling length

The scheduling length (sl) describes the average time for a hardware unit (i.e. ECU) to complete the round-robin processing of all its assigned components. ST is given by

$$sl(d) = \frac{1}{m} \cdot \sum_{j=1}^{m} \left(\frac{\sum_{c \in C_{h_j}} ext(c)}{ps(h_j)} \right). \tag{8.9}$$

where C_{h_j} is the set of components deployed to the hardware host (ECU) h_j, $ext(c)$ is the estimated time taken for a single execution of the component c and $ps(h_j)$ is the processing speed of ECU h_j.

ECUs are assumed to have a fixed and deterministic scheduling, which is a technology used with Time Triggered Architectures (TTA) in embedded systems in order to maintain the predictability of internal behaviour [102, 75]. With this scheduling strategy, when the software components are allocated to ECUs, a fixed schedule is determined. Each component is given a specific time frame, which it requires to complete the execution. The schedule is done in a round-robin fashion, i.e. each component is allocated its own time frame in a circular fashion. The sum of all the execution time slots in the ECU is called *scheduling length*.

Communication Overhead

In embedded systems with their constrained hardware resources, repeated transmissions between software components are discouraged. The Communication Overhead (CO) [126] objective attempts to enforce minimal data communication for a given set of components and system parameters. As a network- and deployment-dependent metric, the overall communication overhead of the system is used to quantify this aspect. This metric was first formalised by Medvidovic and Malek [126].

$$\begin{aligned} co(d) = \sum_{i=1}^{n} \sum_{j=1}^{n} cf(c_i, c_j) \cdot nd(d(c_i), d(c_j)) + \\ \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{cf(c_i, c_j) \cdot es(c_i, c_j)}{bw(d(c_i), d(c_j)) \cdot r(d(c_i), d(c_j))} \end{aligned} \tag{8.10}$$

where $es : C \times C \rightarrow \mathbb{N}$ is the component event size, with $es(c_i, c_j) = 0$ if $c_i = c_j$ or there is no event occurring, $cf : C \times C \rightarrow \mathbb{R}$ is the communication frequency between c_i and c_j, $bw : H \times H \rightarrow \mathbb{N}$ is the network bandwidth, with $bw(h_i, h_j) = 0$ if $h_i = h_j$ or there is no network connection between h_i and h_j, and $nd : H \times H \rightarrow \mathbb{N}$ is the network delay, with $nd(h_i, h_j) = 0$ if $h_i = h_j$ or there is no network connection between h_i and h_j.

Data Transmission Reliability

In a deployment architecture, the communication of components that belong to the same service and that are deployed in different ECUs is supported by the network. Normally, they have to exchange frequent messages with each other, which makes the reliability of the data transmission in the network a crucial quality attribute. Data transmission reliability is especially important in a real-time embedded system, since important decisions are taken based on the data transmitted through the communication links. Following the definition introduced by Malek [120], data transmission reliability is calculated as follows:

$$dtr(d) = \sum_{i=1}^{n} \sum_{j=1}^{n} cf(c_i, c_j) \cdot r(d(c_i), d(c_j)) \tag{8.11}$$

where $cf(c_i, c_j)$ is the communication frequency between components c_i and c_j, and $r(d(c_i), d(c_j))$ is the reliability of the communication link between the hardware resources where c_i and c_j are deployed, which is calculated by using Equation 8.5.

Constraints

Not all deployment architectures represent feasible alternatives. For instance, placing all components into a single host is not feasible due to memory constraints. Hence the optimisation process should check if every new created solution satisfies

the set of constraints. The set of constraints Ω is defined independently of the quality functions. Currently, we consider three constraints $\Omega = \{mem, loc, colloc\}$, where mem is the memory constraint, loc denotes the localisation constraint and $colloc$ is the collocation constraint.

Memory constraint: Processing units have limited memory, which enforces a constraint on the possible components that can be deployed in each ECU. Formally, let $d^{-1} : \mathcal{H} \rightarrow \mathcal{C}_h$ denote the inverse relation to $d \in D$, i.e. $d^{-1}(\mathcal{H}) = \{C_h \in C \mid d(C_h) = h\}$. Then the memory constraint $mem : D \rightarrow \{true, false\}$ is defined as follows:

$$mem(d) \; = \; \forall h \in \mathcal{H} : \sum_{C_h \in d^{-1}(h)} mc(C_h) \; \leq \; mh(h) \tag{8.12}$$

where $mc(C_h)$ is the total memory required to run the set of components deployed to the hardware host h, and $mh(h)$ is the available memory in host h. In other words, this constraint does not allow any deployment solution which exceeds the available memory in the hardware resources.

Localisation constraints: Processing units have access to different sensors and actuators which are used by the software components. For instance, the Brake Force Sensor (BFS) software component reads information from the respective sensor. As a result it has to be deployed to an ECU which can communicate with that sensor (i.e. is connected via the network or the sensor is built into it). The availability of a specific sensor in a hardware host, restricts the allocation of the software component that uses the sensor to that particular host. This constraint is called localisation constraint, denoted $loc : D \rightarrow \{true, false\}$ and is defined as follows:

$$loc(d) = \forall c \in C : (h \in lr(c) \; \Rightarrow \; d(c) \neq h) \tag{8.13}$$

where $lr(c)$ is the list of localisation restrictions.

Collocation constraints: Some of the components should not be allocated in the same ECU. For example, a software component should not be allocated in the same ECU as its redundancies, so that if one of the ECUs fails the other one can still perform the required task. This is called collocation constraint, denoted as $colloc : D \rightarrow \{true, false\}$, and restricts the allocation of two software components to two different hosts. Collocation constraint is calculated as follows:

$$colloc(d) = \forall c \in C : (h \in cr(c_i, c_j) \Rightarrow d(c_i) \neq d(c_j) \tag{8.14}$$

where $cr(c_i, c_j)$ is the matrix of collocation restrictions.

8.4.6 Results of the Optimisation Process

The presented case-study is a comparatively small segment of the actual automotive architecture optimisation problem. Despite this fact, the possible number of candidate architectures is still too large to search with an exact algorithm, i.e. $9^{13} \approx 2.54 \cdot 10^12$ options for the redundancy allocation problem, and even larger for the component deployment problem. To optimise both architecture optimisation problems we employed an Evolutionary Algorithm with the adaptive parameter control strategy proposed in this thesis which was compared to an EA with tuned parameter values. For the purpose of this experiment we consider five parameters: mutation rate, crossover rate, mating pool size, crossover operator, mutation operator. The ranges/values of parameters controlled in this experiment are given in Table 8.8.

The tuning of the static parameter values was performed following recommendations of Smit and Eiben [172]. We use a Sequential Parameter Optimisation (SPO) [17], which tests each parameter combination using several runs. To decrease the number of tests required, we employ a racing technique, which uses a

Table 8.8: Ranges of parameters.

Parameter	**Controlled range/value**
Mutation rate	[0.0010,0.5]
Crossover rate	[0.6,1.0]
Mating pool size	[0.1,0.7]
Mutation operator	Single-point, Uniform
Crossover operator	Single-point, Uniform

variable number of runs depending on the performance of the parameter configuration. Parameter configurations are tested against the best configuration so far, using at least the same number of function evaluations as employed for the best configuration. Five different design points are selected for mutation rate, crossover rate and mating pool size from the ranges depicted in Figure 8.8. The results from the tuning process are shown in Table 8.9.

Table 8.9: Tuned parameter values.

Parameter	Component deployment	Redundancy allocation
Mutation rate	0.1	0.2
Crossover rate	0.6	0.8
Population size	100	100
Mating pool size	0.6	0.6
Mutation operator	Single-point	Uniform
Crossover operator	Single-point	Uniform

The execution of the algorithm was set to 10 000 candidate evaluations, and performed under on a dual-core 2.26 GHz processor computer. The algorithm took 92 seconds for the 10 000 function evaluations and generated 231 approximate solutions, four of which are depicted in Table 8.10.

Looking at the results, we notice that the second solution (ra_2) has the highest reliability among the four. However, its cost and response time are also the highest.

Table 8.10: Redundancy Solutions.

Sol	ra_1	ra_2	ra_3	ra_4
$ra(c_1)$	2	2	0	0
$ra(c_2)$	1	2	0	0
$ra(c_3)$	2	2	0	1
$ra(c_4)$	2	2	0	1
$ra(c_5)$	1	2	0	0
$ra(c_6)$	1	2	0	0
$ra(c_7)$	1	2	0	0
$ra(c_8)$	2	2	0	0
$ra(c_9)$	1	2	1	0
$ra(c_{10})$	1	1	0	0
$ra(c_{11})$	1	2	0	0
$ra(c_{12})$	1	2	0	0
$ra(c_{13})$	2	2	1	0
cost	890	1120	440	435
rt(ra)	56.48	62.96	41.15	43.27
r(ra)	0.996099	0.996347	0.793488	0.920808

The first solution (ra_1) is lower in cost and has a smaller response time compared to the second solution, however its reliability is also lower. The two last solutions (ra_3 and ra_4) have the lower cost and response time than the first two solutions, but their reliability values are much lower.

Both solutions ra_3 and ra_4 have only two components with one redundancy level assigned (solution ra_3 has components c_9 and c_{13}, whereas solutions ra_4 has components c_3 and c_4). Interestingly, the reliability of the ra_4 is much higher than the respective value of ra_3 ($0.920808 > 0.793488$). The response time and the cost of solution ra_4 are also higher than ra_3, although the difference between respective

values is not big.

Table 8.11: Deployment Solutions.

Sol	d_1	d_2	d_3	d_4
$d(c_1)$	6	6	6	6
$d(c_2)$	6	6	6	6
$d(c_{3,1})$	3	6	6	2
$d(c_{3,2})$	6	6	6	6
$d(c_{4,1})$	3	2	2	2
$d(c_{4,1})$	3	2	2	2
$d(c_5)$	3	2	6	6
$d(c_6)$	3	2	2	2
$d(c_7)$	7	6	6	6
$d(c_8)$	7	6	6	6
$d(c_9)$	9	9	9	9
$d(c_{10})$	8	9	9	9
$d(c_{11})$	8	9	9	9
$d(c_{12})$	8	9	9	9
$d(c_{13})$	8	9	9	9
sl(d)	4.2186	1.95538	1.91329	1.93013
co(d)	65.2804	74.8092	94.6799	82.8092
dtr(d)	0.95569	0.95117	0.94121	0.94548

At this stage, the system designer would have to select the architecture representing the redundancy levels for all component according to their preferences. For the purpose of this case-study, we select solution ra_4 as the final software architecture with the redundancy levels, which has to be deployed in to the hardware architecture. Similar to the redundancy allocation optimisation problem, we employ the adaptive Evolutionary Algorithm to optimise the deployment of the software archi-

tecture (solution ra_4) to the hardware architecture (Figure 8.7). The Evolutionary Algorithm reported eight nondominated deployment architectures after the optimisation process. We show four of these deployment architectures in Figure 8.11. The optimised deployment architectures make a trade-off among the quality attributes of response time, communication overhead and data transmission reliability. The system architects have to decide on which architecture to use depending on their preferences.

Table 8.12: The means, standard deviations and Kolmogorov-Smirnov test values of hypervolume indicators for the 30 runs of each problem instance using different optimisation schemes.

Mean		
Problem	**Tuned**	**Parameter control**
Redundancy allocation	0.2617	**0.2619**
Component deployment	0.2618	**0.2620**
Standard Deviation		
Problem	**Tuned**	**Parameter control**
Redundancy allocation	9.770E-05	1.173E-04
Component deployment	1.179E-04	1.083E-04
KS test		
Problem	**d**	**p**
Redundancy allocation	0.3284	0.049
Component deployment	0.3667	0.026

To understand the benefits of using the approach proposed in this thesis for the architecture optimisation problem, we compared the outcomes of an Evolutionary Algorithm using parameter control and tuned parameter values. We performed 30 runs for each optimisation scheme. The mean and standard deviation of the 30 hypervolume values for each optimisation scheme are reported in Table 8.12. The

Evolutionary Algorithm using the adaptive parameter control consistently outperforms the EA with tuned parameter values.

The different optimisation methods are validated using the Kolmogorov-Smirnov (KS) non-parametric test [149] in order to check for a statistical difference in the outperformance. The 30 values of the hypervolume indicators were submitted to the KS analysis which result in a confirmation of the outperformance of the adaptive EA with a minimum d-value of 0.3284 at a 95% confidence level. Hence we conclude that the superior performance of the proposed Adaptive Parameter Control is statistically significant.

8.5 Summary

Architecture optimisation in embedded systems is an example of optimisation problems that are faced in the industry. This chapter has demonstrated the applicability of the adaptive parameter control for Evolutionary Algorithms in this domain. We have presented a set of optimisation runs which have shown that using our approach to control parameters of an Evolutionary Algorithm achieves better results quality than using tuned parameter settings. Most importantly, the reduction in the number of the EA parameters that have to be set by a practitioner facilitates the transfer of EAs into industrial settings, where practitioners do not have any knowledge on how to tune algorithm parameters.

Part IV

Conclusions and Future Work

Chapter 9

Conclusions

This thesis presented an adaptive parameter control method for adjusting parameters of Evolutionary Algorithms during the optimisation process. The main contributions of the thesis are in the four stages of adaptive parameter control: the feedback collection strategy, the parameter effect assessment strategy, the parameter quality attribution strategy and the parameter value selection strategy.

Adaptive parameter control approaches for Evolutionary Algorithms adjust parameter values based on the algorithm's performance, which is part of the feedback collection strategy. While many feedback collection strategies for singleobjective optimisation problems have been presented in the past, there are no recommendations in the literature regarding effective feedback collection strategies for multiobjective problems. In multiobjective optimisation runs, and in the absence of a preference for certain objectives, a set of nondominated solutions, which make a trade-off between the fitness functions, are produced. The challenge faced in assessing the performance of a multiobjective optimisation algorithm relies on the ability to express the quality of multiple solutions with a single value to employ as feedback for the adaptive parameter control method.

Distinguished multiobjective performance metrics were investigated in Chapter 4 and recommended metrics were applied as a feedback collection strategy in adaptive parameter control for multiobjective EAs. An empirical evaluation of two different

feedback collection strategies - the binary hypervolume indicator and the binary epsilon indicator - was performed and recommendations regarding appropriate multiobjective performance metrics for feedback collection strategies were provided. The results from a set of experiments on multiobjective problem instances demonstrated that using the binary hypervolume indicator as a feedback collection strategy leads the algorithm to finding solutions of better quality compared to the binary epsilon indicator.

The output of the feedback collection strategy was employed to approximate the effect of parameter values on the performance of the Evolutionary Algorithm. Due to the stochastic nature of Evolutionary Algorithms, and the presence of more than one parameter, the approximation of parameter effects has to accommodate the ambiguity regarding the real cause of the algorithm's performance.

To cope with this kind of uncertainty, a probabilistic measure of the effect of parameter values, called Bayesian Effect Assessment (BEA), was introduced in Chapter 5. BEA approximates the effect of each parameter value as the times a parameter value was successful divided by the times a parameter value was used, where success is defined as producing results with qualities above a certain threshold value. Unlike state-of-the-art approaches, in which the performance of the EA reported by the feedback strategy is considered as the effect of parameter values, BEA attempts to infer the extent to which the algorithm's performance is affected by the parameter values. The analysis of the threshold value on different problem instances demonstrated that using the median performance of the algorithm instances for the BEA produces better results quality and is more robust compared to the 10th, 20th and 40th percentiles. In the experimental studies, BEA outperformed other prominent parameter effect assessment methods in different problem instances.

To make an informed judgement on what would be a successful parameter value for the next iteration(s), we derived the Predictive Quality Attribution (PQA) strat-

egy, which combines the parameter effects measured in the past with time series prediction. As described in Chapter 6, PQA attempts to predict the effect that parameter values may have in the next iteration by using forecasting techniques. Four forecasting techniques were investigated: the Linear Regression (LR), the Simple Moving Average (SMA), the Exponentially-Weighted Moving Average (EWMA), and the Autoregressive Moving Average (ARIMA).

To verify the suitability of each forecasting technique for predicting the effect of EA parameters, statistical tests were applied to the effects of the mutation rate, crossover rate, population size, mating pool size, crossover operator and mutation operator. It was observed the mutation rate, crossover rate, mating pool size, crossover operator and mutation operator followed the assumptions made by the forecasting models, such as linearity, normality of the residuals, stationarity, homosedasticity, and independence of the residuals, hence any of these techniques can be used to predict the success rates of these parameters.

The population size, on the other hand, did not conform to the assumptions made by the forecasting techniques, hence it may not be appropriate to predict its effect for the future iterations. However, the results from the experiments showed that using the Predictive Quality Attribution strategy to control the population size does not have a negative effect on the performance of the Evolutionary Algorithm. Based on the experiments with the population size, we argue that if EA parameters are controlled without prior testing for statistical properties, Predictive Quality Attribution can be used without a negative effect in the performance of the algorithm.

Despite being one of the simplest forecasting strategies, Linear Regression proved to be the most effective model to predict the effect of Evolutionary Algorithm parameters. In the experimental studies we the Predictive Quality Attribution (PQA) method with Linear Regression was used to adjust the mutation rate, crossover

rate, mating pool size, mutation operator and crossover operator throughout the optimisation process carried out using different problem instances. The Predictive Quality Attribution (PQA) strategy was compared to two prominent parameter quality assessment strategies: the Average Quality Attribution (AQA) strategy and the Extreme Quality Attribution (EQA) strategy. The trials demonstrated that PQA outperforms the parameter quality attribution methods currently considered most successful. Based on these results, it can be asserted that predicting the effects of parameter values is favourable for the algorithm's performance.

The projected parameter quality was used to select the parameter values for the following cycle. Parameter value selection strategies are typically devised to handle the balance between exploration and exploitation required to sample promising areas of the parameter space. In real-valued parameter assignments, such as the mutation rate and the crossover rate, choices for parameter assignments are discretised beforehand and the feedback from the algorithm is applied to the preselected choices. It has been observed that the initial discretisation of parameter values may provide values which are suboptimal for some problems or their instances.

To address this problem, the Adaptive Range Parameter Selection (ARPS) strategy, introduced in Chapter 7, adjusts the ranges of continuous parameter values as the optimisation process progresses. The algorithm starts with two ranges for each parameter interval. At every iteration, ARPS rewards the best-performing parameter ranges by dividing them into two new ranges and assigning each of them the same selection probability as the parent range, whereas the worst performing ranges are merged with one of the neighbouring ranges.

According to our knowledge, the best-performing approaches with equivalent functionality are AP, PM and DMAB. The experimental studies showed that the new approach clearly outperforms the other parameter value selection methods. The adaptation of parameter value ranges increases the sampling accuracy, since the

more successful a parameter value range is, the smaller the interval becomes, which can explain the superior performance of ARPS compared to other state-of-the-art parameter selection strategies. Furthermore, unlike other adaptive parameter value selection methods, ARPS does not have any hyperparameters that require tuning before the optimisation process.

Through using ARPS, we observed that different parameter ranges were optimal for different problems and different stages of the optimisation process. The smaller problems were best optimised with a low mutation rate throughout the process, whereas the more difficult problems required slightly higher mutation rates at the start but toward the end of the process the value ranges were not as focussed. Furthermore, the ranges of the crossover rate developed quite differently compared to the mutation rate. Higher crossover rates were often more successful towards the end of the optimisation process, which runs somewhat contrary to popular opinion that crossover rates should decrease towards the end of the optimisation process so as not to disturb solutions with high quality.

The applicability of the adaptive parameter control for configuring Evolutionary Algorithms in a real world problem was demonstrated in Chapter 8. Two problems from the Software Engineering domain - the component deployment problem and the redundancy allocation problem - were introduced. The Evolutionary Algorithm with adaptive parameter control was applied to optimising these two problems in a case study from the automotive industry and the results were compared to the outcome from an EA with prior tuning of its parameter. The adaptive Evolutionary Algorithm produced results with better quality than the EA with static parameter settings.

To conclude the adaptive parameter control strategy presented in this thesis, within its acknowledged limitations, is an effective method for adjusting parameters of Evolutionary Algorithms. One might argue that although parameter configura-

tion seems to be a very important factor of any algorithm, the parameter space is considerably smaller than the search space of all relevant NP-hard problems and the number of parameters is relatively small. So contrary to NP-hard optimisation problems, where 'solving by hand' is entirely inappropriate, 'choosing parameters by hand' is possible.

However, the use of parameter values that remain constant over the optimisation process has been observed in previous studies to achieve suboptimal results. This was also demonstrated by the case study in Chapter 8. Parameters of an EA can have different optimal values at different stages of the search. Thus, when an optimisation algorithm is tuned prior to the optimisation stage, the selected parameters at the end of this process are not necessarily optimal. This suggests that controlling parameters dynamically during the optimisation process is likely to lead to better outcomes. Hence, the proposed method of adapting parameters during the optimisation process is a more suitable option for the parameter configuration process. Furthermore, by reducing the number of parameters that need to be set in Evolutionary Algorithms, the transfer of these algorithms into industrial settings, where practitioners do not have any knowledge about EA parameters, is facilitated.

Chapter 10

Future Work

The adaptive parameter control method introduced in this thesis used feedback from the optimisation process to derive the effect of parameter values on the performance of the algorithm. Two performance aspects were considered: the fitness and the diversity of the solutions. In the future, we intend to investigate other performance aspects of the optimisation algorithm, such as the ability of the solutions to evolve, i.e. the ability to produce solutions with better quality, which can be used to measure the *parameter productivity*.

We presented an experimental validation of the approach in which multiple parameters were controlled. The adjustment of parameter values was based on the information about their effect on the performance of the algorithm. The experiments conducted thus far are not exhaustive. There is ample opportunity for future work and experimentation. In particular, it would be interesting to more thoroughly investigate the use of information about possible correlations of the values of different parameters as well as the potential integration of such information into the adaptive parameter control strategy.

The parameter value selection strategy introduced in Chapter 7 proved successful in controlling continuous ranges of parameter values. The analysis showed that high-performing ranges were sometimes absorbed (merged) into very large intervals, making it difficult for the algorithm to re-establish small, promising areas within the

range. There may be a potential for further optimisation of the range adaptation in this respect. Furthermore, the use of adaptive ranges for tuning continuous EA parameters is another interesting area that requires investigation.

The application of the proposed method to other stochastic algorithms is a priority. We are particularly interested in using adaptive parameter control for the on-the-fly optimisation of a cooling schedule for an implementation of Simulated Annealing, which is always a challenging problem for practitioners, as this parameter cannot rely on a predefined static value.

Finally, it is unlikely to know if any optimiser, with or without parameter control is better than another over all problems. Trials with other problems and problem instances will be needed to explore the capabilities of the adaptive parameter control.

Part V

Appendix

Characteristics of the Effect of Parameter Values - Experiments with Two Ranges/Values

Linearity

Table 10.1: Linearity test of mutation rate in the range [0.001,0.249].

Mutation rate in the range [0.001,0.249]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	52.24	1.12e-10	Do not reject H_0	16	5.05	0.02676	Do not reject H_0
2	19.13	3.06e-05	Do not reject H_0	17	8.14	0.00528	Do not reject H_0
3	30.30	3.01e-07	Do not reject H_0	18	5.97	0.01631	Do not reject H_0
4	15.39	0.000162	Do not reject H_0	19	13.36	0.00041	Do not reject H_0
5	29.92	3.50e-07	Do not reject H_0	20	23.82	4.14e-06	Do not reject H_0
6	16.36	0.000104	Do not reject H_0	21	14.45	0.000250	Do not reject H_0
7	18.58	3.89e-05	Do not reject H_0	22	25.26	2.28e-06	Do not reject H_0
8	18.73	3.65e-05	Do not reject H_0	23	30.67	2.60e-07	Do not reject H_0
9	13.08	0.000474	Do not reject H_0	24	14.34	0.000263	Do not reject H_0
10	27.99	7.52e-07	Do not reject H_0	25	35.63	3.90e-08	Do not reject H_0
11	24.15	3.62e-06	Do not reject H_0	26	13.60	0.000372	Do not reject H_0
12	19.15	3.03e-05	Do not reject H_0	27	26.63	1.30e-06	Do not reject H_0
13	36.84	2.48e-08	Do not reject H_0	28	23.87	4.05e-06	Do not reject H_0
14	2.479	0.11861	Reject H_0	29	27.65	8.64e-07	Do not reject H_0
15	5.058	0.026767	Do not reject H_0	30	11.08	0.001233	Do not reject H_0

Table 10.2: Linearity test of mutation rate in the range [0.25,0.49].

Mutation rate in the range [0.25,0.49]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	18.2	4.61e-05	Do not reject H_0	16	28.06	7.32e-07	Do not reject H_0
2	17.42	6.53e-05	Do not reject H_0	17	17.61	6.01e-05	Do not reject H_0
3	14.90	0.000203	Do not reject H_0	18	26.4	1.43e-06	Do not reject H_0
4	14.87	0.000206	Do not reject H_0	19	15.01	0.000194	Do not reject H_0
5	7.98	0.005729	Do not reject H_0	20	21.96	9.08e-06	Do not reject H_0
6	14.24	0.000276	Do not reject H_0	21	42.22	3.51e-09	Do not reject H_0
7	23.69	4.36e-06	Do not reject H_0	22	20.51	1.68e-05	Do not reject H_0
8	16.77	8.72e-05	Do not reject H_0	23	19.05	3.17e-05	Do not reject H_0
9	16.21	0.000112	Do not reject H_0	24	22.96	5.94e-06	Do not reject H_0
10	18.58	3.91e-05	Do not reject H_0	25	30.01	3.37e-07	Do not reject H_0
11	17.78	5.54e-05	Do not reject H_0	26	16.38	0.000104	Do not reject H_0
12	10.73	0.001456	Do not reject H_0	27	9.259	0.003014	Do not reject H_0
13	8.80	0.003783	Do not reject H_0	28	11.95	0.000812	Do not reject H_0
14	27.15	1.05e-06	Do not reject H_0	29	12.61	0.000592	Do not reject H_0
15	28.06	7.32e-07	Do not reject H_0	30	8.448	0.004527	Do not reject H_0

Table 10.3: Linearity test of crossover rate in the range [0.6,0.79].

Crossover rate [0.6,0.79]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	16.85	8.42e-05	Do not reject H_0	16	13.37	0.000415	Do not reject H_0
2	53.45	7.57e-11	Do not reject H_0	17	13.84	0.000332	Do not reject H_0
3	14.99	0.000195	Do not reject H_0	18	14.61	0.000233	Do not reject H_0
4	5.86	0.017293	Do not reject H_0	19	16.19	0.000113	Do not reject H_0
5	16.66	9.16e-05	Do not reject H_0	20	9.02	0.003384	Do not reject H_0
6	16.47	9.99e-05	Do not reject H_0	21	12.88	0.000520	Do not reject H_0
7	30.82	2.45e-07	Do not reject H_0	22	18.41	4.21e-05	Do not reject H_0
8	20.37	1.79e-05	Do not reject H_0	23	37.66	1.83e-08	Do not reject H_0
9	7.64	0.006822	Do not reject H_0	24	20.72	1.54e-05	Do not reject H_0
10	22.50	7.21e-06	Do not reject H_0	25	23.82	4.14e-06	Do not reject H_0
11	36.78	2.54e-08	Do not reject H_0	26	6.07	0.015432	Do not reject H_0
12	9.92	0.002167	Do not reject H_0	27	15.74	0.000139	Do not reject H_0
13	16.68	9.07e-05	Do not reject H_0	28	15.00	0.000194	Do not reject H_0
14	13.27	0.000434	Do not reject H_0	29	20.72	1.53e-05	Do not reject H_0
15	13.37	0.000415	Do not reject H_0	30	9.03	0.003364	Do not reject H_0

Table 10.4: Linearity test of crossover rate in the range [0.8,0.99].

Crossover rate in the range [0.8,0.99]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	44.73	1.44e-09	Do not reject H_0	16	17.81	5.48e-05	Do not reject H_0
2	7.81	0.006243	Do not reject H_0	17	11.30	0.001108	Do not reject H_0
3	23.24	5.28e-06	Do not reject H_0	18	27.62	8.75e-07	Do not reject H_0
4	29.97	3.43e-07	Do not reject H_0	19	11.66	0.000931	Do not reject H_0
5	19.61	2.49e-05	Do not reject H_0	20	36.12	3.24e-08	Do not reject H_0
6	7.99	0.005708	Do not reject H_0	21	38.02	1.60e-08	Do not reject H_0
7	14.78	0.000215	Do not reject H_0	22	27.99	7.54e-07	Do not reject H_0
8	20.61	1.61e-05	Do not reject H_0	23	22.82	6.29e-06	Do not reject H_0
9	27.38	9.65e-07	Do not reject H_0	24	14.40	0.000256	Do not reject H_0
10	45.16	1.24e-09	Do not reject H_0	25	24.25	3.46e-06	Do not reject H_0
11	6.65	0.011409	Do not reject H_0	26	31.64	1.78e-07	Do not reject H_0
12	16.95	8.05e-05	Do not reject H_0	27	13.86	0.000330	Do not reject H_0
13	21.72	1.00e-05	Do not reject H_0	28	20.25	1.88e-05	Do not reject H_0
14	7.54	0.007172	Do not reject H_0	29	13.70	0.000356	Do not reject H_0
15	17.81	5.48e-05	Do not reject H_0	30	12.39	0.000657	Do not reject H_0

Table 10.5: Linearity test of population size in the range [20,60].

Population size in the range [20,60]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	169.4	5.24e-23	Do not reject H_0	16	122.7	6.39e-19	Do not reject H_0
2	132.3	7.97e-20	Do not reject H_0	17	224.5	5.49e-27	Do not reject H_0
3	88.72	2.40e-15	Do not reject H_0	18	109.6	1.29e-17	Do not reject H_0
4	211.8	3.89e-26	Do not reject H_0	19	166.1	9.77e-23	Do not reject H_0
5	182.1	5.39e-24	Do not reject H_0	20	167.3	7.73e-23	Do not reject H_0
6	154.8	8.21e-22	Do not reject H_0	21	79.03	3.33e-14	Do not reject H_0
7	122.1	7.23e-19	Do not reject H_0	22	122.7	6.37e-19	Do not reject H_0
8	176.3	1.50e-23	Do not reject H_0	23	158.7	3.87e-22	Do not reject H_0
9	124.1	4.80e-19	Do not reject H_0	24	176.4	1.47e-23	Do not reject H_0
10	173.5	2.46e-23	Do not reject H_0	25	205.6	1.04e-25	Do not reject H_0
11	88.69	2.42e-15	Do not reject H_0	26	91.24	1.24e-15	Do not reject H_0
12	283.4	1.53e-30	Do not reject H_0	27	157.7	4.69e-22	Do not reject H_0
13	107.6	2.05e-17	Do not reject H_0	28	180.7	6.96e-24	Do not reject H_0
14	235.7	1.03e-27	Do not reject H_0	29	120.7	1.01e-18	Do not reject H_0
15	122.7	6.39e-19	Do not reject H_0	30	169.9	4.81e-23	Do not reject H_0

Table 10.6: Linearity test of population size in the range [61,100].

Population size in the range [61,100]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	9.826	0.00227	Do not reject H_0	16	3.922	0.05047	Reject H_0
2	5.18	0.02504	Do not reject H_0	17	0.143	0.70544	Reject H_0
3	13.62	0.00036	Do not reject H_0	18	4.392	0.03869	Do not reject H_0
4	0.767	0.38309	Reject H_0	19	0.519	0.47299	Reject H_0
5	0.172	0.67886	Reject H_0	20	0.213	0.64494	Reject H_0
6	0.350	0.55505	Reject H_0	21	25.63	1.96e-06	Do not reject H_0
7	4.149	0.04436	Do not reject H_0	22	5.673	0.019179	Do not reject H_0
8	0.966	0.32799	Reject H_0	23	0.298	0.58636	Reject H_0
9	5.861	0.01733	Do not reject H_0	24	0.939	0.33482	Reject H_0
10	0.202	0.65351	Reject H_0	25	0.094	0.75895	Reject H_0
11	7.343	0.00795	Do not reject H_0	26	0.966	0.32797	Reject H_0
12	2.543	0.11401	Reject H_0	27	0.096	0.75659	Reject H_0
13	16.66	9.17e-05	Do not reject H_0	28	1.699	0.19538	Reject H_0
14	0.015	0.9011	Reject H_0	29	3.766	0.055179	Reject H_0
15	3.922	0.05047	Reject H_0	30	0.833	0.36358	Reject H_0

Table 10.7: Linearity test of mating pool size in the range [0.1,0.39].

Mating pool size in the range [0.1,0.39]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	9.54	0.002613	Do not reject H_0	16	9.05	0.003337	Do not reject H_0
2	12.83	0.000534	Do not reject H_0	17	8.77	0.003841	Do not reject H_0
3	19.35	2.78e-05	Do not reject H_0	18	15.48	0.000156	Do not reject H_0
4	7.33	0.007991	Do not reject H_0	19	13.72	0.000351	Do not reject H_0
5	21.01	1.35e-05	Do not reject H_0	20	11.00	0.001279	Do not reject H_0
6	33.75	7.94e-08	Do not reject H_0	21	15.56	0.000150	Do not reject H_0
7	9.74	0.002371	Do not reject H_0	22	12.61	0.000591	Do not reject H_0
8	6.75	0.010817	Do not reject H_0	23	7.13	0.008868	Do not reject H_0
9	4.9	0.029186	Do not reject H_0	24	16.87	8.35e-05	Do not reject H_0
10	13.22	0.000444	Do not reject H_0	25	10.46	0.001664	Do not reject H_0
11	5.132	0.025703	Do not reject H_0	26	14.73	0.000220	Do not reject H_0
12	13.69	0.000357	Do not reject H_0	27	30.42	2.87e-07	Do not reject H_0
13	11.22	0.001151	Do not reject H_0	28	13.77	0.000343	Do not reject H_0
14	7.28	0.008208	Do not reject H_0	29	26.21	1.54e-06	Do not reject H_0
15	9.05	0.003337	Do not reject H_0	30	22.72	6.57e-06	Do not reject H_0

Table 10.8: Linearity test of mating pool size in the range [0.4,0.69].

Mating pool size in the range [0.4,0.69]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	39.32	9.97e-09	Do not reject H_0	16	37.05	2.29e-08	Do not reject H_0
2	21.32	1.19e-05	Do not reject H_0	17	40.52	6.45e-09	Do not reject H_0
3	33.80	7.79e-08	Do not reject H_0	18	10.14	0.001943	Do not reject H_0
4	39.34	9.89e-09	Do not reject H_0	19	32.36	1.35e-07	Do not reject H_0
5	12.56	0.000605	Do not reject H_0	20	12.18	0.000726	Do not reject H_0
6	18.05	4.94e-05	Do not reject H_0	21	30.92	2.36e-07	Do not reject H_0
7	12.35	0.000670	Do not reject H_0	22	8.96	0.003493	Do not reject H_0
8	16.11	0.000117	Do not reject H_0	23	24.73	2.84e-06	Do not reject H_0
9	37.97	1.63e-08	Do not reject H_0	24	39.62	8.92e-09	Do not reject H_0
10	23.15	5.49e-06	Do not reject H_0	25	29.03	4.98e-07	Do not reject H_0
11	18.85	3.46e-05	Do not reject H_0	26	28.11	7.17e-07	Do not reject H_0
12	36.95	2.38e-08	Do not reject H_0	27	23.34	5.07e-06	Do not reject H_0
13	21.28	1.21e-05	Do not reject H_0	28	20.09	2.02e-05	Do not reject H_0
14	13.67	0.000361	Do not reject H_0	29	13.62	0.000368	Do not reject H_0
15	37.05	2.29e-08	Do not reject H_0	30	12.76	0.000551	Do not reject H_0

Table 10.9: Linearity test of single-point mutation operator.

Single-point mutation							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	43.24	2.44e-09	Do not reject H_0	16	15.66	0.000144	Do not reject H_0
2	33.64	8.28e-08	Do not reject H_0	17	12.27	0.000695	Do not reject H_0
3	13.00	0.000493	Do not reject H_0	18	12.61	0.000592	Do not reject H_0
4	16.13	0.000116	Do not reject H_0	19	4.37	0.03907	Do not reject H_0
5	25.99	1.69e-06	Do not reject H_0	20	10.67	0.001506	Do not reject H_0
6	35.69	3.81e-08	Do not reject H_0	21	24.18	3.57e-06	Do not reject H_0
7	24.25	3.46e-06	Do not reject H_0	22	21.47	1.11e-05	Do not reject H_0
8	19.14	3.04e-05	Do not reject H_0	23	19.30	2.84e-05	Do not reject H_0
9	11.86	0.000844	Do not reject H_0	24	27.19	1.04e-06	Do not reject H_0
10	29.55	4.05e-07	Do not reject H_0	25	7.06	0.009178	Do not reject H_0
11	20.58	1.63e-05	Do not reject H_0	26	14.32	0.000266	Do not reject H_0
12	22.33	7.74e-06	Do not reject H_0	27	29.07	4.90e-07	Do not reject H_0
13	15.59	0.000148	Do not reject H_0	28	13.09	0.000473	Do not reject H_0
14	51.58	1.40e-10	Do not reject H_0	29	35.20	4.57e-08	Do not reject H_0
15	15.66	0.000144	Do not reject H_0	30	15.72	0.000139	Do not reject H_0

Table 10.10: Linearity test of uniform mutation operator.

Uniform mutation							
Run	f	p	Conclusion	Run	f	p	Conclusion
	17.12	7.46e-05	Do not reject H_0	16	14.27	0.000272	Do not reject H_0
2	14.70	0.000223	Do not reject H_0	17	40.44	6.65e-09	Do not reject H_0
3	24.65	2.93e-06	Do not reject H_0	18	38.35	1.42e-08	Do not reject H_0
4	23.35	5.05e-06	Do not reject H_0	19	9.93	0.002157	Do not reject H_0
5	17.14	7.40e-05	Do not reject H_0	20	41.58	4.41e-09	Do not reject H_0
6	4.352	0.039569	Do not reject H_0	21	17.72	5.72e-05	Do not reject H_0
7	13.31	0.000426	Do not reject H_0	22	21.71	1.00e-05	Do not reject H_0
8	39.04	1.10e-08	Do not reject H_0	23	13.90	0.000324	Do not reject H_0
9	20.56	1.64e-05	Do not reject H_0	24	28.51	6.12e-07	Do not reject H_0
10	11.73	0.000901	Do not reject H_0	25	14.14	0.000289	Do not reject H_0
11	19.99	2.10e-05	Do not reject H_0	26	29.84	3.61e-07	Do not reject H_0
12	13.63	0.000366	Do not reject H_0	27	12.19	0.000722	Do not reject H_0
13	27.66	8.61e-07	Do not reject H_0	28	27.06	1.09e-06	Do not reject H_0
14	3.294	0.072612	Do not reject H_0	29	8.74	0.003898	Do not reject H_0
15	14.27	0.000272	Do not reject H_0	30	17.69	5.78e-05	Do not reject H_0

Table 10.11: Linearity test of single-point crossover operator.

Single-point crossover							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	26.06	1.64e-06	Do not reject H_0	16	21.61	1.05e-05	Do not reject H_0
2	43.54	2.19e-09	Do not reject H_0	17	24.12	3.66e-06	Do not reject H_0
3	18.69	3.72e-05	Do not reject H_0	18	27.96	7.62e-07	Do not reject H_0
4	23.87	4.05e-06	Do not reject H_0	19	9.104	0.003256	Do not reject H_0
5	26.85	1.19e-06	Do not reject H_0	20	30.48	2.81e-07	Do not reject H_0
6	21.54	1.08e-05	Do not reject H_0	21	40.05	7.63e-09	Do not reject H_0
7	22.94	5.99e-06	Do not reject H_0	22	7.640	0.006829	Do not reject H_0
8	15.76	0.001376	Do not reject H_0	23	21.46	1.12e-05	Do not reject H_0
9	25.74	1.87e-06	Do not reject H_0	24	11.28	0.001116	Do not reject H_0
10	13.84	0.000332	Do not reject H_0	25	12.90	0.000516	Do not reject H_0
11	10.53	0.001608	Do not reject H_0	26	10.67	0.001501	Do not reject H_0
12	15.59	0.000148	Do not reject H_0	27	29.93	3.49e-07	Do not reject H_0
13	34.85	5.23e-08	Do not reject H_0	28	17.45	6.43e-05	Do not reject H_0
14	15.75	0.000138	Do not reject H_0	29	16.59	9.46e-05	Do not reject H_0
15	21.61	1.05e-05	Do not reject H_0	30	8.281	0.004926	Do not reject H_0

Table 10.12: Linearity test of uniform crossover operator.

Uniform crossover							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	31.24	2.09e-07	Do not reject H_0	16	13.11	0.000469	Do not reject H_0
2	6.034	0.015803	Do not reject H_0	17	21.88	9.35e-06	Do not reject H_0
3	15.32	0.000168	Do not reject H_0	18	23.04	5.74e-06	Do not reject H_0
4	17.74	5.66e-05	Do not reject H_0	19	12.24	0.000706	Do not reject H_0
5	12.59	0.000598	Do not reject H_0	20	13.82	0.000336	Do not reject H_0
6	18.70	3.71e-05	Do not reject H_0	21	14.10	0.000294	Do not reject H_0
7	12.60	0.000595	Do not reject H_0	22	32.52	1.27e-07	Do not reject H_0
8	44.41	1.61e-09	Do not reject H_0	23	20.15	1.97e-05	Do not reject H_0
9	23.30	5.14e-06	Do not reject H_0	24	46.13	8.89e-10	Do not reject H_0
10	21.05	1.33e-05	Do not reject H_0	25	8.295	0.004892	Do not reject H_0
11	34.03	7.13e-08	Do not reject H_0	26	20.45	1.72e-05	Do not reject H_0
12	25.55	2.03e-06	Do not reject H_0	27	13.03	0.000486	Do not reject H_0
13	10.55	0.001591	Do not reject H_0	28	18.54	3.97e-05	Do not reject H_0
14	32.90	1.09e-07	Do not reject H_0	29	30.91	2.37e-07	Do not reject H_0
15	13.11	0.000469	Do not reject H_0	30	27.13	1.06e-06	Do not reject H_0

Normality

Table 10.13: Kolmogorov-Smirnov test of mutation rate in the range [0.001,0.249].

Mutation rate in the range [0.001,0.249]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0648	0.7748	Do not reject H_0	16	0.0546	0.9131	Do not reject H_0
2	0.0567	0.8904	Do not reject H_0	17	0.0559	0.8998	Do not reject H_0
3	0.0738	0.627	Do not reject H_0	18	0.0449	0.9834	Do not reject H_0
4	0.0738	0.627	Do not reject H_0	19	0.0586	0.866	Do not reject H_0
5	0.0495	0.9589	Do not reject H_0	20	0.0734	0.6338	Do not reject H_0
6	0.1036	0.2228	Do not reject H_0	21	0.0457	0.9798	Do not reject H_0
7	0.0785	0.5482	Do not reject H_0	22	0.0433	0.9886	Do not reject H_0
8	0.0767	0.5785	Do not reject H_0	23	0.0794	0.5347	Do not reject H_0
9	0.0525	0.934	Do not reject H_0	24	0.0452	0.982	Do not reject H_0
10	0.0826	0.4835	Do not reject H_0	25	0.0771	0.5718	Do not reject H_0
11	0.0703	0.6847	Do not reject H_0	26	0.0773	0.5683	Do not reject H_0
12	0.0627	0.8082	Do not reject H_0	27	0.0557	0.9015	Do not reject H_0
13	0.062	0.819	Do not reject H_0	28	0.0579	0.8756	Do not reject H_0
14	0.1029	0.2288	Do not reject H_0	29	0.0812	0.505	Do not reject H_0
15	0.0729	0.641	Do not reject H_0	30	0.0432	0.655	Do not reject H_0

Table 10.14: Kolmogorov-Smirnov test of mutation rate in the range [0.25,0.49].

Mutation rate in the range [0.25,0.49]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0697	0.6955	Do not reject H_0	16	0.0514	0.944	Do not reject H_0
2	0.103	0.2281	Do not reject H_0	17	0.0556	0.9022	Do not reject H_0
3	0.071	0.673	Do not reject H_0	18	0.0661	0.7542	Do not reject H_0
4	0.083	0.4778	Do not reject H_0	19	0.0553	0.906	Do not reject H_0
5	0.0889	0.3911	Do not reject H_0	20	0.0697	0.6952	Do not reject H_0
6	0.0643	0.7838	Do not reject H_0	21	0.0652	0.7688	Do not reject H_0
7	0.0615	0.8252	Do not reject H_0	22	0.066	0.7565	Do not reject H_0
8	0.0752	0.6028	Do not reject H_0	23	0.0569	0.8877	Do not reject H_0
9	0.0636	0.794	Do not reject H_0	24	0.0512	0.9458	Do not reject H_0
10	0.0676	0.73	Do not reject H_0	25	0.0617	0.823	Do not reject H_0
11	0.0587	0.8648	Do not reject H_0	26	0.0462	0.9775	Do not reject H_0
12	0.0638	0.7914	Do not reject H_0	27	0.0834	0.4715	Do not reject H_0
13	0.0637	0.7927	Do not reject H_0	28	0.0678	0.727	Do not reject H_0
14	0.0726	0.6472	Do not reject H_0	29	0.0558	0.9	Do not reject H_0
15	0.0398	0.9959	Do not reject H_0	30	0.0748	0.89	Do not reject H_0

Table 10.15: Kolmogorov-Smirnov test of crossover rate in the range [0.6,0.79].

Crossover rate in the range [0.6,0.79]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0648	0.7748	Do not reject H_0	16	0.1034	0.2245	Do not reject H_0
2	0.0567	0.8904	Do not reject H_0	17	0.0779	0.5587	Do not reject H_0
3	0.0563	0.8942	Do not reject H_0	18	0.0549	0.9102	Do not reject H_0
4	0.0482	0.9673	Do not reject H_0	19	0.0855	0.4405	Do not reject H_0
5	0.0697	0.6955	Do not reject H_0	20	0.0816	0.4996	Do not reject H_0
6	0.0778	0.56	Do not reject H_0	21	0.0587	0.8647	Do not reject H_0
7	0.0782	0.5538	Do not reject H_0	22	0.0502	0.9533	Do not reject H_0
8	0.0532	0.9281	Do not reject H_0	23	0.0439	0.9869	Do not reject H_0
9	0.0712	0.6707	Do not reject H_0	24	0.0354	0.9993	Do not reject H_0
10	0.038	0.9978	Do not reject H_0	25	0.0816	0.4997	Do not reject H_0
11	0.0905	0.3695	Do not reject H_0	26	0.0486	0.9648	Do not reject H_0
12	0.0682	0.7212	Do not reject H_0	27	0.068	0.7233	Do not reject H_0
13	0.0866	0.4239	Do not reject H_0	28	0.0532	0.9275	Do not reject H_0
14	0.0628	0.8063	Do not reject H_0	29	0.0483	0.9664	Do not reject H_0
15	0.05	0.955	Do not reject H_0	30	0.0433	0.8561	Do not reject H_0

Table 10.16: Kolmogorov-Smirnov test of crossover rate in the range [0.8,0.99].

Crossover rate in the range [0.8,0.99]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0507	0.9499	Do not reject H_0	16	0.0791	0.5389	Do not reject H_0
2	0.051	0.9474	Do not reject H_0	17	0.0484	0.9656	Do not reject H_0
3	0.0556	0.9029	Do not reject H_0	18	0.1001	0.2569	Do not reject H_0
4	0.0818	0.4963	Do not reject H_0	19	0.0699	0.6917	Do not reject H_0
5	0.0572	0.8841	Do not reject H_0	20	0.0529	0.9303	Do not reject H_0
6	0.079	0.5408	Do not reject H_0	21	0.0564	0.8937	Do not reject H_0
7	0.0912	0.3607	Do not reject H_0	22	0.0419	0.9921	Do not reject H_0
8	0.0577	0.878	Do not reject H_0	23	0.0678	0.7263	Do not reject H_0
9	0.0431	0.9893	Do not reject H_0	24	0.0632	0.801	Do not reject H_0
10	0.0782	0.5544	Do not reject H_0	25	0.0726	0.6475	Do not reject H_0
11	0.0862	0.4293	Do not reject H_0	26	0.0545	0.9148	Do not reject H_0
12	0.0512	0.9454	Do not reject H_0	27	0.0857	0.437	Do not reject H_0
13	0.0459	0.9792	Do not reject H_0	28	0.079	0.5403	Do not reject H_0
14	0.0581	0.8719	Do not reject H_0	29	0.0559	0.8996	Do not reject H_0
15	0.0537	0.9231	Do not reject H_0	30	0.0339	0.8291	Do not reject H_0

Table 10.17: Kolmogorov-Smirnov test of population size in the range [20,60].

Population size in the range [20,60]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0462	0.9775	Do not reject H_0	16	0.0579	0.8737	Do not reject H_0
2	0.0699	0.6911	Do not reject H_0	17	0.0491	0.9611	Do not reject H_0
3	0.0420	0.9918	Do not reject H_0	18	0.0550	0.9090	Do not reject H_0
4	0.0614	0.8269	Do not reject H_0	19	0.0532	0.9273	Do not reject H_0
5	0.0663	0.7499	Do not reject H_0	20	0.0554	0.9045	Do not reject H_0
6	0.0684	0.7167	Do not reject H_0	21	0.0516	0.9416	Do not reject H_0
7	0.0915	0.3567	Do not reject H_0	22	0.0611	0.8302	Do not reject H_0
8	0.0617	0.8227	Do not reject H_0	23	0.0446	0.9840	Do not reject H_0
9	0.0672	0.7365	Do not reject H_0	24	0.0619	0.8185	Do not reject H_0
10	0.0758	0.5928	Do not reject H_0	25	0.0742	0.6186	Do not reject H_0
11	0.0376	0.9980	Do not reject H_0	26	0.0657	0.7600	Do not reject H_0
12	0.0331	0.9997	Do not reject H_0	27	0.0461	0.9780	Do not reject H_0
13	0.0551	0.9075	Do not reject H_0	28	0.0635	0.7948	Do not reject H_0
14	0.0445	0.9844	Do not reject H_0	29	0.0628	0.8050	Do not reject H_0
15	0.0579	0.8737	Do not reject H_0	30	0.0470	0.9733	Do not reject H_0

Table 10.18: Kolmogorov-Smirnov test of population size in the range [61,100].

Population size in the range [61,100]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0473	0.9720	Do not reject H_0	16	0.0502	0.9530	Do not reject H_0
2	0.0563	0.8936	Do not reject H_0	17	0.1552	0.0149	Reject H_0
3	0.0615	0.8255	Do not reject H_0	18	0.0947	0.3161	Do not reject H_0
4	0.1182	0.1155	Do not reject H_0	19	0.1379	0.0418	Reject H_0
5	0.1493	0.0215	Reject H_0	20	0.1598	0.0111	Reject H_0
6	0.1265	0.0766	Do not reject H_0	21	0.0853	0.4415	Do not reject H_0
7	0.0419	0.9920	Do not reject H_0	22	0.0439	0.9866	Do not reject H_0
8	0.0937	0.3283	Do not reject H_0	23	0.1339	0.0519	Do not reject H_0
9	0.0653	0.7664	Do not reject H_0	24	0.1337	0.0527	Do not reject H_0
10	0.1457	0.0267	Reject H_0	25	0.1619	0.0097	Reject H_0
11	0.0863	0.4280	Do not reject H_0	26	0.1343	0.0508	Do not reject H_0
12	0.0703	0.6853	Do not reject H_0	27	0.1589	0.0118	Reject H_0
13	0.0595	0.8530	Do not reject H_0	28	0.0998	0.2595	Do not reject H_0
14	0.1870	0.0016	Reject H_0	29	0.0467	0.9751	Do not reject H_0
15	0.0502	0.9530	Do not reject H_0	30	0.1285	0.0691	Do not reject H_0

Table 10.19: Kolmogorov-Smirnov test of mating pool size in the range [0.1,0.39].

Mating pool size in the range [0.1,0.39]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0608	0.8350	Do not reject H_0	16	0.0487	0.9638	Do not reject H_0
2	0.0741	0.6214	Do not reject H_0	17	0.0394	0.9963	Do not reject H_0
3	0.0832	0.4738	Do not reject H_0	18	0.0693	0.7008	Do not reject H_0
4	0.0487	0.9636	Do not reject H_0	19	0.0548	0.9109	Do not reject H_0
5	0.0338	0.9996	Do not reject H_0	20	0.0655	0.7630	Do not reject H_0
6	0.0538	0.9218	Do not reject H_0	21	0.0435	0.9879	Do not reject H_0
7	0.0804	0.5178	Do not reject H_0	22	0.0601	0.8455	Do not reject H_0
8	0.0580	0.8730	Do not reject H_0	23	0.0480	0.9680	Do not reject H_0
9	0.0581	0.8717	Do not reject H_0	24	0.0796	0.5293	Do not reject H_0
10	0.0706	0.6803	Do not reject H_0	25	0.0728	0.6418	Do not reject H_0
11	0.0847	0.4512	Do not reject H_0	26	0.0454	0.9812	Do not reject H_0
12	0.0971	0.2884	Do not reject H_0	27	0.0829	0.4789	Do not reject H_0
13	0.0460	0.9782	Do not reject H_0	28	0.0966	0.2941	Do not reject H_0
14	0.0533	0.9264	Do not reject H_0	29	0.0579	0.8749	Do not reject H_0
15	0.0487	0.9638	Do not reject H_0	30	0.0892	0.3863	Do not reject H_0

Table 10.20: Kolmogorov-Smirnov test of mating pool size in the range [0.4,0.69].

Mating pool size in the range [0.4,0.69]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0572	0.8835	Do not reject H_0	16	0.0951	0.3112	Do not reject H_0
2	0.0382	0.9975	Do not reject H_0	17	0.0513	0.9446	Do not reject H_0
3	0.0424	0.9908	Do not reject H_0	18	0.0470	0.9736	Do not reject H_0
4	0.0506	0.9500	Do not reject H_0	19	0.0543	0.9165	Do not reject H_0
5	0.0636	0.7928	Do not reject H_0	20	0.0639	0.7890	Do not reject H_0
6	0.0719	0.6575	Do not reject H_0	21	0.0774	0.5658	Do not reject H_0
7	0.0860	0.4318	Do not reject H_0	22	0.0801	0.5222	Do not reject H_0
8	0.0748	0.6098	Do not reject H_0	23	0.1102	0.1670	Do not reject H_0
9	0.0488	0.9630	Do not reject H_0	24	0.0496	0.9575	Do not reject H_0
10	0.0387	0.9971	Do not reject H_0	25	0.0566	0.8905	Do not reject H_0
11	0.0581	0.8717	Do not reject H_0	26	0.0569	0.8864	Do not reject H_0
12	0.0395	0.9962	Do not reject H_0	27	0.0761	0.5878	Do not reject H_0
13	0.0830	0.4771	Do not reject H_0	28	0.0453	0.9814	Do not reject H_0
14	0.0529	0.9306	Do not reject H_0	29	0.0665	0.7469	Do not reject H_0
15	0.0951	0.3112	Do not reject H_0	30	0.0564	0.8926	Do not reject H_0

Table 10.21: Kolmogorov-Smirnov test of single-point mutation operator.

Single-point mutation							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0561	0.8967	Do not reject H_0	16	0.0613	0.8280	Do not reject H_0
2	0.0560	0.8980	Do not reject H_0	17	0.1113	0.1590	Do not reject H_0
3	0.0619	0.8188	Do not reject H_0	18	0.0698	0.6924	Do not reject H_0
4	0.0479	0.9686	Do not reject H_0	19	0.0726	0.6465	Do not reject H_0
5	0.0406	0.9946	Do not reject H_0	20	0.0668	0.7425	Do not reject H_0
6	0.0441	0.9860	Do not reject H_0	21	0.0589	0.8612	Do not reject H_0
7	0.0488	0.9630	Do not reject H_0	22	0.0770	0.5726	Do not reject H_0
8	0.0501	0.9541	Do not reject H_0	23	0.0538	0.9209	Do not reject H_0
9	0.0528	0.9316	Do not reject H_0	24	0.0759	0.5903	Do not reject H_0
10	0.0545	0.9146	Do not reject H_0	25	0.0704	0.6830	Do not reject H_0
11	0.0548	0.9111	Do not reject H_0	26	0.0572	0.8834	Do not reject H_0
12	0.0698	0.6931	Do not reject H_0	27	0.0708	0.6767	Do not reject H_0
13	0.0752	0.6026	Do not reject H_0	28	0.0576	0.8782	Do not reject H_0
14	0.0680	0.7227	Do not reject H_0	29	0.0355	0.9992	Do not reject H_0
15	0.0613	0.8280	Do not reject H_0	30	0.0839	0.4636	Do not reject H_0

Table 10.22: Kolmogorov-Smirnov test of uniform mutation operator.

Uniform mutation							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0857	0.4368	Do not reject H_0	16	0.0503	0.9522	Do not reject H_0
2	0.0521	0.9375	Do not reject H_0	17	0.0707	0.6773	Do not reject H_0
3	0.0402	0.9952	Do not reject H_0	18	0.0416	0.9927	Do not reject H_0
4	0.0689	0.7073	Do not reject H_0	19	0.0720	0.6553	Do not reject H_0
5	0.0556	0.9026	Do not reject H_0	20	0.0496	0.9573	Do not reject H_0
6	0.0691	0.7050	Do not reject H_0	21	0.0702	0.6857	Do not reject H_0
7	0.0721	0.6539	Do not reject H_0	22	0.0496	0.9575	Do not reject H_0
8	0.0557	0.9014	Do not reject H_0	23	0.0596	0.8523	Do not reject H_0
9	0.1058	0.2022	Do not reject H_0	24	0.0582	0.8705	Do not reject H_0
10	0.0545	0.9140	Do not reject H_0	25	0.0707	0.6776	Do not reject H_0
11	0.0414	0.9931	Do not reject H_0	26	0.0608	0.8353	Do not reject H_0
12	0.1122	0.1529	Do not reject H_0	27	0.0719	0.6580	Do not reject H_0
13	0.0628	0.8051	Do not reject H_0	28	0.0575	0.8793	Do not reject H_0
14	0.0834	0.4713	Do not reject H_0	29	0.0466	0.9754	Do not reject H_0
15	0.0503	0.9522	Do not reject H_0	30	0.0553	0.9055	Do not reject H_0

Table 10.23: Kolmogorov-Smirnov test of single-point crossover operator.

Single-point crossover							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0585	0.8671	Do not reject H_0	16	0.0547	0.9122	Do not reject H_0
2	0.0703	0.6850	Do not reject H_0	17	0.0739	0.6249	Do not reject H_0
3	0.0321	0.9998	Do not reject H_0	18	0.0467	0.9749	Do not reject H_0
4	0.0690	0.7067	Do not reject H_0	19	0.1035	0.2230	Do not reject H_0
5	0.0720	0.6566	Do not reject H_0	20	0.0599	0.8475	Do not reject H_0
6	0.0564	0.8928	Do not reject H_0	21	0.0565	0.8920	Do not reject H_0
7	0.0665	0.7481	Do not reject H_0	22	0.0626	0.8081	Do not reject H_0
8	0.0823	0.4875	Do not reject H_0	23	0.0549	0.9097	Do not reject H_0
9	0.0515	0.9427	Do not reject H_0	24	0.0619	0.8196	Do not reject H_0
10	0.0533	0.9264	Do not reject H_0	25	0.0774	0.5655	Do not reject H_0
11	0.0572	0.8829	Do not reject H_0	26	0.0601	0.8450	Do not reject H_0
12	0.0763	0.5844	Do not reject H_0	27	0.0446	0.9841	Do not reject H_0
13	0.0525	0.9335	Do not reject H_0	28	0.0853	0.4427	Do not reject H_0
14	0.0778	0.5585	Do not reject H_0	29	0.0489	0.9620	Do not reject H_0
15	0.0547	0.9122	Do not reject H_0	30	0.0507	0.9490	Do not reject H_0

Table 10.24: Kolmogorov-Smirnov test of uniform crossover operator.

Uniform crossover							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.0794	0.5327	Do not reject H_0	16	0.0555	0.9037	Do not reject H_0
2	0.0733	0.6340	Do not reject H_0	17	0.0674	0.7331	Do not reject H_0
3	0.0425	0.9907	Do not reject H_0	18	0.0494	0.9592	Do not reject H_0
4	0.0462	0.9772	Do not reject H_0	19	0.0480	0.9682	Do not reject H_0
5	0.0451	0.9820	Do not reject H_0	20	0.0553	0.9049	Do not reject H_0
6	0.0445	0.9846	Do not reject H_0	21	0.0518	0.9403	Do not reject H_0
7	0.0673	0.7343	Do not reject H_0	22	0.0712	0.6702	Do not reject H_0
8	0.0470	0.9737	Do not reject H_0	23	0.0385	0.9973	Do not reject H_0
9	0.0627	0.8076	Do not reject H_0	24	0.0613	0.8274	Do not reject H_0
10	0.0663	0.7502	Do not reject H_0	25	0.0696	0.6968	Do not reject H_0
11	0.0686	0.7123	Do not reject H_0	26	0.0337	0.9996	Do not reject H_0
12	0.0664	0.7485	Do not reject H_0	27	0.0648	0.7750	Do not reject H_0
13	0.0432	0.9886	Do not reject H_0	28	0.0662	0.7529	Do not reject H_0
14	0.0452	0.9817	Do not reject H_0	29	0.0649	0.7733	Do not reject H_0
15	0.0555	0.9037	Do not reject H_0	30	0.0901	0.3750	Do not reject H_0

Independence

Table 10.25: Durbin-Watson test of mutation rate in the range [0.001,0.249].

Mutation rate in the range [0.001,0.249]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.922	0.6889	Do not reject H_0	16	2.0333	0.4784	Do not reject H_0
2	2.0577	0.4287	Do not reject H_0	17	2.1064	0.3350	Do not reject H_0
3	1.9792	0.582	Do not reject H_0	18	2.1513	0.2572	Do not reject H_0
4	2.3169	0.0684	Do not reject H_0	19	2.3273	0.0617	Do not reject H_0
5	2.2703	0.1054	Do not reject H_0	20	2.1301	0.2927	Do not reject H_0
6	2.2058	0.1773	Do not reject H_0	21	2.3106	0.0727	Do not reject H_0
7	2.2081	0.1743	Do not reject H_0	22	1.964	0.6114	Do not reject H_0
8	2.3095	0.0735	Do not reject H_0	23	2.4884	0.0088	Reject H_0
9	2.0352	0.4746	Do not reject H_0	24	2.3713	0.0387	Reject H_0
10	2.4107	0.0245	Reject H_0	25	2.2373	0.1391	Do not reject H_0
11	1.8952	0.7346	Do not reject H_0	26	2.4143	0.0235	Reject H_0
12	2.0502	0.4437	Do not reject H_0	27	1.8163	0.8467	Do not reject H_0
13	2.3629	0.0425	Reject H_0	28	2.246	0.1295	Do not reject H_0
14	2.1995	0.1856	Do not reject H_0	29	1.9508	0.6365	Do not reject H_0
15	2.4781	0.0102	Reject H_0	30	2.1065	0.733	Do not reject H_0

Table 10.26: Durbin-Watson test of mutation rate [0.25,0.49].

Mutation rate in the [0.25,0.49]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.9795	0.5815	Do not reject H_0	16	2.5333	0.0045	Reject H_0
2	2.1044	0.3387	Do not reject H_0	17	1.9433	0.6504	Do not reject H_0
3	2.2896	0.0887	Do not reject H_0	18	2.3049	0.0768	Do not reject H_0
4	1.9209	0.6908	Do not reject H_0	19	2.3288	0.0608	Do not reject H_0
5	2.3946	0.0297	Reject H_0	20	2.0996	0.3476	Do not reject H_0
6	2.0155	0.5104	Do not reject H_0	21	2.3023	0.0787	Do not reject H_0
7	2.3873	0.0323	Reject H_0	22	2.3541	0.0467	Reject H_0
8	2.1375	0.2801	Do not reject H_0	23	2.4525	0.0145	Reject H_0
9	2.6308	0.0008	Reject H_0	24	2.261	0.1143	Do not reject H_0
10	2.0122	0.517	Do not reject H_0	25	2.6994	0.0002	Reject H_0
11	2.0645	0.4152	Do not reject H_0	26	1.8137	0.8497	Do not reject H_0
12	2.2037	0.1800	Do not reject H_0	27	2.1607	0.2423	Do not reject H_0
13	1.9337	0.6679	Do not reject H_0	28	2.1945	0.1924	Do not reject H_0
14	2.0646	0.415	Do not reject H_0	29	2.4044	0.0264	Reject H_0
15	2.3705	0.0391	Reject H_0	30	2.1034	0.3901	Do not reject H_0

Table 10.27: Durbin-Watson test of crossover rate in the range [0.6,0.79].

Crossover rate in the range [0.6,0.79]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.277	0.0994	Do not reject H_0	16	2.2178	0.1621	Do not reject H_0
2	1.9583	0.6222	Do not reject H_0	17	2.0295	0.4825	Do not reject H_0
3	1.8781	0.7619	Do not reject H_0	18	2.2184	0.1613	Do not reject H_0
4	2.1583	0.2460	Do not reject H_0	19	2.079	0.3868	Do not reject H_0
5	2.4978	0.0077	Reject H_0	20	2.3013	0.0795	Do not reject H_0
6	2.2443	0.1314	Do not reject H_0	21	2.1898	0.1989	Do not reject H_0
7	1.7936	0.872	Do not reject H_0	22	2.1002	0.3464	Do not reject H_0
8	2.3236	0.0640	Do not reject H_0	23	2.4647	0.0123	Reject H_0
9	2.0227	0.4961	Do not reject H_0	24	2.7302	0.0001	Reject H_0
10	2.0111	0.5191	Do not reject H_0	25	2.468	0.0117	Reject H_0
11	2.0641	0.4160	Do not reject H_0	26	2.2937	0.0854	Do not reject H_0
12	2.268	0.1076	Do not reject H_0	27	2.1461	0.2658	Do not reject H_0
13	2.1054	0.3369	Do not reject H_0	28	2.135	0.2843	Do not reject H_0
14	2.4104	0.0246	Reject H_0	29	2.1896	0.1992	Do not reject H_0
15	2.2816	0.0953	Do not reject H_0	30	2.102	0.421	Do not reject H_0

Table 10.28: Durbin-Watson test of crossover rate in the range [0.8,0.99].

Crossover rate in the range [0.8,0.99]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.267	0.1085	Do not reject H_0	16	2.3276	0.0615	Do not reject H_0
2	2.3986	0.0283	Reject H_0	17	2.3452	0.0513	Do not reject H_0
3	1.9097	0.7104	Do not reject H_0	18	2.1845	0.2064	Do not reject H_0
4	2.3529	0.0473	Reject H_0	19	2.0683	0.4077	Do not reject H_0
5	2.4927	0.0083	Reject H_0	20	2.2217	0.1572	Do not reject H_0
6	2.1352	0.2840	Do not reject H_0	21	2.3377	0.0555	Do not reject H_0
7	1.9345	0.6665	Do not reject H_0	22	2.2445	0.1311	Do not reject H_0
8	1.9668	0.606	Do not reject H_0	23	2.1119	0.325	Do not reject H_0
9	2.1071	0.3338	Do not reject H_0	24	2.3175	0.0680	Do not reject H_0
10	2.0365	0.4719	Do not reject H_0	25	2.009	0.5234	Do not reject H_0
11	2.38	0.0351	Reject H_0	26	2.1191	0.312	Do not reject H_0
12	2.131	0.2911	Do not reject H_0	27	2.0603	0.4235	Do not reject H_0
13	2.2093	0.1727	Do not reject H_0	28	2.4104	0.0246	Reject H_0
14	2.2729	0.1030	Do not reject H_0	29	2.2218	0.1571	Do not reject H_0
15	2.2953	0.0841	Do not reject H_0	30	2.130	0.3497	Do not reject H_0

Table 10.29: Durbin-Watson test of population size in the range [20,60].

Population size in the range [20,60]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.8205	0.8416	Do not reject H_0	16	2.2726	0.1033	Do not reject H_0
2	2.0895	0.3666	Do not reject H_0	17	1.6196	0.9782	Do not reject H_0
3	2.0029	0.5355	Do not reject H_0	18	1.6083	0.9811	Do not reject H_0
4	1.8848	0.7514	Do not reject H_0	19	1.5926	0.9845	Do not reject H_0
5	2.0361	0.4726	Do not reject H_0	20	1.9947	0.5516	Do not reject H_0
6	1.5324	0.9932	Do not reject H_0	21	1.5265	0.9938	Do not reject H_0
7	2.2983	0.0817	Do not reject H_0	22	1.9699	0.6000	Do not reject H_0
8	2.1322	0.2890	Do not reject H_0	23	1.9294	0.6756	Do not reject H_0
9	1.8792	0.7602	Do not reject H_0	24	1.8986	0.7290	Do not reject H_0
10	1.4955	0.9961	Do not reject H_0	25	2.0559	0.4323	Do not reject H_0
11	1.9466	0.6442	Do not reject H_0	26	2.3309	0.0595	Do not reject H_0
12	1.9106	0.7088	Do not reject H_0	27	2.0736	0.3971	Do not reject H_0
13	1.4059	0.9991	Do not reject H_0	28	1.882	0.7558	Do not reject H_0
14	1.7981	0.8673	Do not reject H_0	29	1.6811	0.95595	Do not reject H_0
15	2.2726	0.1033	Do not reject H_0	30	1.0905	1	Do not reject H_0

Table 10.30: Durbin-Watson test of population size in the range [61,100].

Population size in the range [61,100]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.7918	0.8739	Do not reject H_0	16	2.2033	0.1805	Do not reject H_0
2	1.8339	0.8248	Do not reject H_0	17	1.4813	0.9969	Do not reject H_0
3	1.8063	0.8582	Do not reject H_0	18	2.0181	0.5052	Do not reject H_0
4	1.8102	0.8537	Do not reject H_0	19	2.1889	0.2002	Do not reject H_0
5	1.8266	0.8341	Do not reject H_0	20	1.7614	0.9029	Do not reject H_0
6	1.9293	0.6759	Do not reject H_0	21	1.9566	0.6255	Do not reject H_0
7	2.0971	0.3522	Do not reject H_0	22	1.9768	0.5867	Do not reject H_0
8	2.2004	0.1844	Do not reject H_0	23	1.9811	0.5784	Do not reject H_0
9	1.9052	0.7179	Do not reject H_0	24	1.9832	0.5742	Do not reject H_0
10	1.5417	0.9922	Do not reject H_0	25	1.7468	0.9150	Do not reject H_0
11	2.0459	0.4525	Do not reject H_0	26	1.7189	0.9348	Do not reject H_0
12	1.7391	0.9209	Do not reject H_0	27	1.8708	0.7731	Do not reject H_0
13	1.6364	0.9734	Do not reject H_0	28	1.7180	0.9354	Do not reject H_0
14	1.7637	0.9009	Do not reject H_0	29	2.0201	0.5012	Do not reject H_0
15	2.2033	0.1805	Do not reject H_0	30	1.1891	1.0	Do not reject H_0

Table 10.31: Durbin-Watson test of mating pool size in the range [0.1,0.39].

Mating pool size in the range [0.1,0.39]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.3354	0.0568	Do not reject H_0	16	2.0756	0.3933	Do not reject H_0
2	1.8966	0.7323	Do not reject H_0	17	2.0332	0.4786	Do not reject H_0
3	1.7529	0.9100	Do not reject H_0	18	1.9406	0.6554	Do not reject H_0
4	2.1685	0.2301	Do not reject H_0	19	2.0223	0.4967	Do not reject H_0
5	2.3997	0.0280	Reject H_0	20	2.3446	0.0517	Do not reject H_0
6	2.1311	0.2910	Do not reject H_0	21	2.1886	0.2006	Do not reject H_0
7	2.1907	0.1977	Do not reject H_0	22	2.3831	0.0339	Reject H_0
8	2.3286	0.0609	Do not reject H_0	23	2.4491	0.0151	Reject H_0
9	2.1796	0.2135	Do not reject H_0	24	2.0708	0.4026	Do not reject H_0
10	2.1157	0.3180	Do not reject H_0	25	2.0454	0.4534	Do not reject H_0
11	2.3609	0.0434	Reject H_0	26	2.058	0.4281	Do not reject H_0
12	2.1768	0.2176	Do not reject H_0	27	1.7931	0.8725	Do not reject H_0
13	2.4537	0.0142	Reject H_0	28	2.0393	0.4661	Do not reject H_0
14	2.5887	0.0018	Reject H_0	29	2.7384	0.0001	Reject H_0
15	2.0756	0.3933	Do not reject H_0	30	1.8624	0.7854	Do not reject H_0

Table 10.32: Durbin-Watson test of mating pool size in the range [0.4,0.69].

Mating pool size in the range [0.4,0.69]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.3969	0.0289	Reject H_0	16	2.14	0.2758	Do not reject H_0
2	2.0756	0.3933	Do not reject H_0	17	2.0626	0.4189	Do not reject H_0
3	2.0603	0.4235	Do not reject H_0	18	2.0240	0.4934	Do not reject H_0
4	2.1835	0.2079	Do not reject H_0	19	2.242	0.1338	Do not reject H_0
5	2.4221	0.0213	Reject H_0	20	2.2978	0.0821	Do not reject H_0
6	2.1411	0.2740	Do not reject H_0	21	2.1646	0.2361	Do not reject H_0
7	2.1877	0.2019	Do not reject H_0	22	2.2617	0.1135	Do not reject H_0
8	2.0864	0.3725	Do not reject H_0	23	2.4625	0.0126	Reject H_0
9	2.3240	0.0637	Do not reject H_0	24	2.2054	0.1777	Do not reject H_0
10	2.5440	0.0038	Reject H_0	25	2.2341	0.1427	Do not reject H_0
11	2.3598	0.0439	Reject H_0	26	2.2324	0.1446	Do not reject H_0
12	2.3956	0.0293	Reject H_0	27	2.1007	0.3455	Do not reject H_0
13	2.2264	0.1516	Do not reject H_0	28	2.0423	0.46	Do not reject H_0
14	2.3083	0.0743	Do not reject H_0	29	2.3077	0.0748	Do not reject H_0
15	2.14	0.2758	Do not reject H_0	30	1.9046	0.7189	Do not reject H_0

Table 10.33: Durbin-Watson test of single-point mutation operator.

Single-point mutation operator							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.5002	0.0075	Reject H_0	16	2.3574	0.0451	Reject H_0
2	1.9763	0.5877	Do not reject H_0	17	2.1747	0.2208	Do not reject H_0
3	2.4367	0.0177	Reject H_0	18	2.4931	0.0083	Reject H_0
4	2.297	0.0827	Do not reject H_0	19	1.9087	0.7119	Do not reject H_0
5	2.3848	0.0333	Reject H_0	20	2.1698	0.2281	Do not reject H_0
6	2.0710	0.4023	Do not reject H_0	21	2.4009	0.0276	Reject H_0
7	2.2762	0.1001	Do not reject H_0	22	2.1679	0.2310	Do not reject H_0
8	2.3601	0.0438	Reject H_0	23	2.0859	0.3734	Do not reject H_0
9	2.2815	0.0954	Do not reject H_0	24	2.1141	0.3209	Do not reject H_0
10	2.2517	0.1236	Do not reject H_0	25	2.1308	0.2915	Do not reject H_0
11	1.9748	0.5906	Do not reject H_0	26	2.2104	0.1712	Do not reject H_0
12	2.1604	0.2427	Do not reject H_0	27	2.3365	0.0562	Do not reject H_0
13	2.0945	0.3571	Do not reject H_0	28	1.7105	0.9401	Do not reject H_0
14	1.8462	0.8083	Do not reject H_0	29	2.0029	0.5354	Do not reject H_0
15	2.3574	0.0451	Reject H_0	30	2.3922	0.0305	Reject H_0

Table 10.34: Durbin-Watson test of uniform mutation operator.

Uniform mutation operator							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.1562	0.2493	Do not reject H_0	16	2.3508	0.0484	Reject H_0
2	2.2342	0.1425	Do not reject H_0	17	2.3039	0.0775	Do not reject H_0
3	2.0863	0.3726	Do not reject H_0	18	2.2308	0.1464	Do not reject H_0
4	2.0481	0.4480	Do not reject H_0	19	2.1309	0.2912	Do not reject H_0
5	2.2172	0.1627	Do not reject H_0	20	2.2138	0.1669	Do not reject H_0
6	2.3317	0.0590	Do not reject H_0	21	2.3532	0.0472	Reject H_0
7	2.3434	0.0523	Do not reject H_0	22	2.2019	0.1823	Do not reject H_0
8	2.1526	0.2551	Do not reject H_0	23	2.0687	0.4068	Do not reject H_0
9	2.3865	0.0326	Reject H_0	24	2.2968	0.0829	Do not reject H_0
10	2.37	0.0393	Reject H_0	25	1.9931	0.5547	Do not reject H_0
11	1.9043	0.7195	Do not reject H_0	26	2.3034	0.0779	Do not reject H_0
12	2.1993	0.1858	Do not reject H_0	27	2.3407	0.0538	Do not reject H_0
13	2.3611	0.0433	Reject H_0	28	1.9797	0.5811	Do not reject H_0
14	2.2081	0.1742	Do not reject H_0	29	2.3667	0.0407	Reject H_0
15	2.3508	0.0484	Reject H_0	30	2.0268	0.4878	Do not reject H_0

Table 10.35: Durbin-Watson test of single-point crossover operator.

Single-point crossover operator							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.0695	0.4052	Do not reject H_0	16	2.4373	0.0176	Reject H_0
2	1.9365	0.6629	Do not reject H_0	17	2.3813	0.0346	Reject H_0
3	2.2308	0.1464	Do not reject H_0	18	2.3369	0.0559	Do not reject H_0
4	2.4873	0.0090	Reject H_0	19	2.0675	0.4092	Do not reject H_0
5	1.9645	0.6104	Do not reject H_0	20	2.2374	0.1389	Do not reject H_0
6	2.3362	0.0563	Do not reject H_0	21	2.6411	0.0007	Reject H_0
7	2.2094	0.1725	Do not reject H_0	22	1.9928	0.5554	Do not reject H_0
8	2.1009	0.3450	Do not reject H_0	23	2.0201	0.5012	Do not reject H_0
9	1.9808	0.5788	Do not reject H_0	24	2.352	0.0478	Reject H_0
10	2.2299	0.1474	Do not reject H_0	25	1.9761	0.5881	Do not reject H_0
11	2.3453	0.0513	Do not reject H_0	26	2.0497	0.4447	Do not reject H_0
12	2.2975	0.0823	Do not reject H_0	27	2.1904	0.1980	Do not reject H_0
13	2.1017	0.3436	Do not reject H_0	28	2.3135	0.0707	Do not reject H_0
14	2.4381	0.0174	Reject H_0	29	1.7589	0.9051	Do not reject H_0
15	2.4373	0.0176	Reject H_0	30	2.1057	0.3362	Do not reject H_0

Table 10.36: Durbin-Watson test of uniform crossover operator.

Uniform crossover operator							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.2804	0.0963	Do not reject H_0	16	2.4239	0.0208	Reject H_0
2	2.1893	0.1996	Do not reject H_0	17	2.1741	0.2216	Do not reject H_0
3	2.356	0.0458	Reject H_0	18	2.3265	0.0622	Do not reject H_0
4	2.2508	0.1245	Do not reject H_0	19	2.2098	0.1720	Do not reject H_0
5	1.8401	0.8167	Do not reject H_0	20	2.4727	0.0110	Reject H_0
6	1.8895	0.7438	Do not reject H_0	21	2.1139	0.3212	Do not reject H_0
7	1.9936	0.5538	Do not reject H_0	22	2.1442	0.2687	Do not reject H_0
8	2.3103	0.0729	Do not reject H_0	23	2.0560	0.4320	Do not reject H_0
9	2.2535	0.1217	Do not reject H_0	24	2.3911	0.0309	Reject H_0
10	2.4607	0.0129	Reject H_0	25	2.1389	0.2776	Do not reject H_0
11	2.2955	0.0839	Do not reject H_0	26	2.1231	0.3049	Do not reject H_0
12	2.1472	0.2639	Do not reject H_0	27	2.1914	0.1967	Do not reject H_0
13	2.5735	0.0024	Reject H_0	28	2.2458	0.1297	Do not reject H_0
14	1.9969	0.5473	Do not reject H_0	29	2.0441	0.4562	Do not reject H_0
15	2.4239	0.0208	Reject H_0	30	2.1410	0.2741	Do not reject H_0

Homoscedasticity

Table 10.37: Breusch-Pagan test of mutation rate in the range [0.001,0.249].

Mutation rate in the range [0.001,0.249]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.7161	0.3974	Do not reject H_0	16	0.2784	0.5977	Do not reject H_0
2	8e-04	0.9777	Do not reject H_0	17	0.0644	0.7997	Do not reject H_0
3	0.003	0.9565	Do not reject H_0	18	0.1186	0.7305	Do not reject H_0
4	0.5097	0.4753	Do not reject H_0	19	0.2949	0.5871	Do not reject H_0
5	0.002	0.9646	Do not reject H_0	20	2.1412	0.1434	Do not reject H_0
6	0.5466	0.4597	Do not reject H_0	21	0.4476	0.5035	Do not reject H_0
7	0.9482	0.3302	Do not reject H_0	22	3.1018	0.0782	Do not reject H_0
8	0.705	0.4011	Do not reject H_0	23	3.1795	0.0745	Do not reject H_0
9	0.1291	0.7194	Do not reject H_0	24	1.5713	0.21	Do not reject H_0
10	0.0092	0.9236	Do not reject H_0	25	1.8452	0.1743	Do not reject H_0
11	9.2751	0.0023	Reject H_0	26	0.601	0.4382	Do not reject H_0
12	4.5713	0.0325	Reject H_0	27	0.0578	0.81	Do not reject H_0
13	0.2524	0.6154	Do not reject H_0	28	1.9137	0.1666	Do not reject H_0
14	6.3624	0.0116	Reject H_0	29	1.558	0.2120	Do not reject H_0
15	1.2282	0.2678	Do not reject H_0	30	1.237	0.54	Do not reject H_0

Table 10.38: Breusch-Pagan test of mutation rate in the range [0.25,0.49].

Mutation rate in the range [0.25,0.49]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.2945	0.5873	Do not reject H_0	16	3.0084	0.0828	Do not reject H_0
2	0.028	0.8671	Do not reject H_0	17	5.6779	0.0171	Reject H_0
3	2.335	0.1265	Do not reject H_0	18	2.8278	0.0926	Do not reject H_0
4	0.2021	0.653	Do not reject H_0	19	0.19	0.663	Do not reject H_0
5	0.6695	0.4132	Do not reject H_0	20	0.3189	0.5723	Do not reject H_0
6	0.0384	0.8446	Do not reject H_0	21	0.3926	0.531	Do not reject H_0
7	0.2158	0.6423	Do not reject H_0	22	0.8963	0.3438	Do not reject H_0
8	0.5312	0.4661	Do not reject H_0	23	0.203	0.6523	Do not reject H_0
9	0.2339	0.6287	Do not reject H_0	24	0.0416	0.8384	Do not reject H_0
10	0.0303	0.8619	Do not reject H_0	25	0.2973	0.5856	Do not reject H_0
11	0.1394	0.7089	Do not reject H_0	26	0.4412	0.5066	Do not reject H_0
12	0.4068	0.5236	Do not reject H_0	27	1.37	0.2418	Do not reject H_0
13	1.191	0.2751	Do not reject H_0	28	0.3282	0.5667	Do not reject H_0
14	0.2365	0.6268	Do not reject H_0	29	1.8697	0.1715	Do not reject H_0
15	0.1505	0.6981	Do not reject H_0	30	1.141	0.624	Do not reject H_0

Table 10.39: Breusch-Pagan test of crossover rate in the range [0.6,0.79].

Crossover rate in the range [0.6,0.79]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.1456	0.7028	Do not reject H_0	16	1.0213	0.3122	Do not reject H_0
2	0.0015	0.9687	Do not reject H_0	17	0.697	0.4038	Do not reject H_0
3	5.4211	0.0199	Reject H_0	18	0.4682	0.4938	Do not reject H_0
4	4.4225	0.0354	Reject H_0	19	1.637	0.2007	Do not reject H_0
5	0.3194	0.572	Do not reject H_0	20	1.0663	0.3018	Do not reject H_0
6	5.4369	0.0197	Reject H_0	21	0.2984	0.5849	Do not reject H_0
7	0.0128	0.91	Do not reject H_0	22	1.1708	0.2792	Do not reject H_0
8	0.3236	0.5695	Do not reject H_0	23	0.3138	0.5753	Do not reject H_0
9	0.8673	0.3517	Do not reject H_0	24	0.0273	0.8688	Do not reject H_0
10	0.3258	0.5682	Do not reject H_0	25	0.0639	0.8004	Do not reject H_0
11	0.0029	0.9569	Do not reject H_0	26	0.0471	0.8282	Do not reject H_0
12	0.5076	0.4762	Do not reject H_0	27	0.4202	0.5168	Do not reject H_0
13	0.668	0.4138	Do not reject H_0	28	0.0233	0.8786	Do not reject H_0
14	0.3872	0.5338	Do not reject H_0	29	1.3116	0.2521	Do not reject H_0
15	0.0614	0.8043	Do not reject H_0	30	2.345	0.356	Do not reject H_0

Table 10.40: Breusch-Pagan test of crossover rate in the range [0.8,0.99].

Crossover rate in the range [0.8,0.99]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.0542	0.8159	Do not reject H_0	16	3.2637	0.07083	Do not reject H_0
2	0.3268	0.5675	Do not reject H_0	17	0.0361	0.8493	Do not reject H_0
3	0.4154	0.5192	Do not reject H_0	18	0.2242	0.6359	Do not reject H_0
4	0.3426	0.5583	Do not reject H_0	19	0.0233	0.8788	Do not reject H_0
5	0.0191	0.8901	Do not reject H_0	20	0.0278	0.8676	Do not reject H_0
6	2.3709	0.1236	Do not reject H_0	21	0.0122	0.912	Do not reject H_0
7	2.1015	0.1472	Do not reject H_0	22	0.4973	0.4807	Do not reject H_0
8	1.7854	0.1815	Do not reject H_0	23	0.6478	0.4209	Do not reject H_0
9	0.9477	0.3303	Do not reject H_0	24	0.0554	0.814	Do not reject H_0
10	0.185	0.6671	Do not reject H_0	25	1.2135	0.2706	Do not reject H_0
11	1.4093	0.2352	Do not reject H_0	26	1.1878	0.2758	Do not reject H_0
12	3.9162	0.0478	Reject H_0	27	1.0879	0.2969	Do not reject H_0
13	0.8315	0.3618	Do not reject H_0	28	1.0018	0.3169	Do not reject H_0
14	3.1222	0.0772	Do not reject H_0	29	0.8453	0.3579	Do not reject H_0
15	0.0419	0.8377	Do not reject H_0	30	0.481	0.4325	Do not reject H_0

Table 10.41: Breusch-Pagan test of population size in the range [20,60].

Population size in the range [20,60]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	13.090	0.0002	Reject H_0	16	10.356	0.0012	Reject H_0
2	0.0989	0.7530	Do not reject H_0	17	2.9409	0.0863	Do not reject H_0
3	3.4772	0.0622	Do not reject H_0	18	3.6097	0.0574	Do not reject H_0
4	2.3964	0.1216	Do not reject H_0	19	2.5269	0.1119	Do not reject H_0
5	6.5948	0.0102	Reject H_0	20	8.5337	0.0034	Reject H_0
6	1.6781	0.1951	Do not reject H_0	21	5.3809	0.0203	Reject H_0
7	0.3247	0.5687	Do not reject H_0	22	1.8074	0.1788	Do not reject H_0
8	3.2901	0.0696	Do not reject H_0	23	2.6732	0.1020	Do not reject H_0
9	8.8222	0.0029	Reject H_0	24	7.8227	0.0051	Reject H_0
10	3.0785	0.0793	Do not reject H_0	25	7.6996	0.0055	Reject H_0
11	0.4228	0.5155	Do not reject H_0	26	0.33747	0.5612	Do not reject H_0
12	4.9534	0.0260	Reject H_0	27	1.4928	0.2217	Do not reject H_0
13	4.8224	0.0280	Reject H_0	28	9.2236	0.0023	Reject H_0
14	2.1302	0.1444	Do not reject H_0	29	4.061	0.0438	Reject H_0
15	10.356	0.0012	Reject H_0	30	12.649	0.0003	Reject H_0

Table 10.42: Breusch-Pagan test of population size in the range [61,100].

Population size in the range [61,100]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	9.612	0.0019	Reject H_0	16	1.2278	0.2678	Do not reject H_0
2	6.8403	0.0089	Reject H_0	17	0.7396	0.3897	Do not reject H_0
3	0.5887	0.4429	Do not reject H_0	18	2.5679	0.1090	Do not reject H_0
4	0.0377	0.8459	Do not reject H_0	19	0.0935	0.7597	Do not reject H_0
5	8.165	0.0042	Reject H_0	20	2.5829	0.1080	Do not reject H_0
6	1.7546	0.1853	Do not reject H_0	21	0.2781	0.5979	Do not reject H_0
7	0.1423	0.7059	Do not reject H_0	22	0.6002	0.4384	Do not reject H_0
8	6.4462	0.0111	Reject H_0	23	0.2698	0.6034	Do not reject H_0
9	0.0005	0.9807	Do not reject H_0	24	0.4891	0.4843	Do not reject H_0
10	5.3302	0.0209	Reject H_0	25	0.406	0.0012	Reject H_0
11	0.0118	0.9131	Do not reject H_0	26	3.7338	0.0533	Do not reject H_0
12	1.6253	0.2023	Do not reject H_0	27	0.2354	0.6275	Do not reject H_0
13	2.0733	0.1499	Do not reject H_0	28	5.3274	0.0209	Reject H_0
14	0.1458	0.7025	Do not reject H_0	29	0.1261	0.7224	Do not reject H_0
15	1.2278	0.2678	Do not reject H_0	30	2.2908	0.1301	Do not reject H_0

Table 10.43: Breusch-Pagan test of mating pool size in the range [0.1,0.39].

Mating pool size in the range [0.1,0.39]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.0064	0.936	Do not reject H_0	16	1.4181	0.2337	Do not reject H_0
2	0.0189	0.8905	Do not reject H_0	17	2.4437	0.1180	Do not reject H_0
3	0.2558	0.6129	Do not reject H_0	18	0.0358	0.8498	Do not reject H_0
4	0.1194	0.7296	Do not reject H_0	19	0.0033	0.9536	Do not reject H_0
5	1.5124	0.2187	Do not reject H_0	20	4.0217	0.0449	Reject H_0
6	1.1070	0.2927	Do not reject H_0	21	1.8383	0.1751	Do not reject H_0
7	1.8092	0.1786	Do not reject H_0	22	1.8862	0.1696	Do not reject H_0
8	1.7134	0.1905	Do not reject H_0	23	1.3204	0.2505	Do not reject H_0
9	0.1897	0.6631	Do not reject H_0	24	0.4817	0.4876	Do not reject H_0
10	0.0019	0.9649	Do not reject H_0	25	0.0451	0.8317	Do not reject H_0
11	0.4543	0.5002	Do not reject H_0	26	0.4532	0.5007	Do not reject H_0
12	0.0459	0.8302	Do not reject H_0	27	1.2058	0.2721	Do not reject H_0
13	0.2735	0.6009	Do not reject H_0	28	0.8445	0.3581	Do not reject H_0
14	0.4849	0.4862	Do not reject H_0	29	6.9999	0.0081	Reject H_0
15	1.4181	0.2337	Do not reject H_0	30	1.4090	0.2352	Do not reject H_0

Table 10.44: Breusch-Pagan test of mating pool size in the range [0.4,0.69].

Mating pool size in the range [0.4,0.69]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.0065	0.9356	Do not reject H_0	16	0.6197	0.4311	Do not reject H_0
2	0.0094	0.9227	Do not reject H_0	17	0.4426	0.5058	Do not reject H_0
3	0.3282	0.5666	Do not reject H_0	18	0.0017	0.9663	Do not reject H_0
4	0.0943	0.7587	Do not reject H_0	19	4.4251	0.0354	Reject H_0
5	0.0867	0.7684	Do not reject H_0	20	2.9845	0.0840	Do not reject H_0
6	2.7568	0.0968	Do not reject H_0	21	0.8207	0.3649	Do not reject H_0
7	1.5263	0.2166	Do not reject H_0	22	0.2832	0.5945	Do not reject H_0
8	0.3320	0.5644	Do not reject H_0	23	4.7721	0.0289	Reject H_0
9	0.0067	0.9344	Do not reject H_0	24	0.9976	0.3178	Do not reject H_0
10	1.7472	0.1862	Do not reject H_0	25	1.6783	0.1951	Do not reject H_0
11	6.5343	0.0105	Reject H_0	26	3.5149	0.0608	Do not reject H_0
12	0.8482	0.3570	Do not reject H_0	27	0.0045	0.9461	Do not reject H_0
13	5.426	0.0198	Reject H_0	28	0.0079	0.9289	Do not reject H_0
14	0.0208	0.8851	Do not reject H_0	29	0.3733	0.5411	Do not reject H_0
15	0.6197	0.4311	Do not reject H_0	30	0.5890	0.4427	Do not reject H_0

Table 10.45: Breusch-Pagan test of single-point mutation operator.

Single-point mutation operator							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.5117	0.4743	Do not reject H_0	16	0.7563	0.3844	Do not reject H_0
2	0.7591	0.3836	Do not reject H_0	17	1.9970	0.1576	Do not reject H_0
3	2.2215	0.1361	Do not reject H_0	18	0.843	0.3585	Do not reject H_0
4	0.1096	0.7406	Do not reject H_0	19	1.666	0.1966	Do not reject H_0
5	0.0478	0.8268	Do not reject H_0	20	0.7589	0.3836	Do not reject H_0
6	0.0870	0.7679	Do not reject H_0	21	7.4996	0.0061	Reject H_0
7	0.2191	0.6396	Do not reject H_0	22	4.7802	0.0287	Reject H_0
8	0.8504	0.3564	Do not reject H_0	23	0.0541	0.8159	Do not reject H_0
9	1.2928	0.2555	Do not reject H_0	24	0.2934	0.5879	Do not reject H_0
10	0.0104	0.9187	Do not reject H_0	25	0.4866	0.4854	Do not reject H_0
11	0.2719	0.6020	Do not reject H_0	26	1.9203	0.1658	Do not reject H_0
12	0.0048	0.9443	Do not reject H_0	27	0.5266	0.4680	Do not reject H_0
13	0.0590	0.8080	Do not reject H_0	28	1.0866	0.2972	Do not reject H_0
14	1.5915	0.2071	Do not reject H_0	29	0.0776	0.7805	Do not reject H_0
15	0.7563	0.3844	Do not reject H_0	30	2.8308	0.0924	Do not reject H_0

Table 10.46: Breusch-Pagan test of uniform mutation operator.

Uniform mutation operator							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	2.3662	0.1239	Do not reject H_0	16	0.5939	0.4408	Do not reject H_0
2	0.0001	0.9889	Do not reject H_0	17	0.0944	0.7586	Do not reject H_0
3	0.901	0.3425	Do not reject H_0	18	0.0272	0.8689	Do not reject H_0
4	0.4615	0.4969	Do not reject H_0	19	0.3765	0.5394	Do not reject H_0
5	0.0006	0.9799	Do not reject H_0	20	4.1127	0.0425	Reject H_0
6	0.4388	0.5077	Do not reject H_0	21	1.7360	0.1876	Do not reject H_0
7	0.0057	0.9393	Do not reject H_0	22	0.0833	0.7728	Do not reject H_0
8	2.793	0.0946	Do not reject H_0	23	0.2069	0.6491	Do not reject H_0
9	0.0448	0.8323	Do not reject H_0	24	0.3275	0.5671	Do not reject H_0
10	1.3098	0.2524	Do not reject H_0	25	1.3987	0.2369	Do not reject H_0
11	1.1042	0.2933	Do not reject H_0	26	0.3859	0.5344	Do not reject H_0
12	0.2095	0.6471	Do not reject H_0	27	1.4508	0.2284	Do not reject H_0
13	0.1438	0.7045	Do not reject H_0	28	4.5723	0.0324	Reject H_0
14	0.7593	0.3835	Do not reject H_0	29	2.0851	0.1487	Do not reject H_0
15	0.5939	0.4408	Do not reject H_0	30	1.8479	0.1740	Do not reject H_0

Table 10.47: Breusch-Pagan test of single-point crossover operator.

Single-point crossover operator							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	1.7009	0.19217	Do not reject H_0	16	0.0476	0.8271	Do not reject H_0
2	1.1979	0.27373	Do not reject H_0	17	0.0507	0.8218	Do not reject H_0
3	0.1627	0.68665	Do not reject H_0	18	0.7155	0.3976	Do not reject H_0
4	1.3330	0.24827	Do not reject H_0	19	0.1886	0.6640	Do not reject H_0
5	0.2524	0.61539	Do not reject H_0	20	3.8588	0.0494	Reject H_0
6	1.6388	0.20049	Do not reject H_0	21	0.0245	0.8754	Do not reject H_0
7	0.1778	0.67326	Do not reject H_0	22	0.3246	0.5688	Do not reject H_0
8	0.0384	0.84444	Do not reject H_0	23	0.7035	0.4015	Do not reject H_0
9	0.0640	0.80021	Do not reject H_0	24	0.5040	0.4777	Do not reject H_0
10	0.3937	0.53031	Do not reject H_0	25	1.1501	0.2835	Do not reject H_0
11	4.7457	0.02937	Reject H_0	26	1.6573	0.1979	Do not reject H_0
12	0.6882	0.40676	Do not reject H_0	27	0.0029	0.9567	Do not reject H_0
13	0.0140	0.90573	Do not reject H_0	28	3.8658	0.0492	Reject H_0
14	0.2066	0.64939	Do not reject H_0	29	0.8842	0.3470	Do not reject H_0
15	0.0476	0.82717	Do not reject H_0	30	6.152	0.0131	Reject H_0

Table 10.48: Breusch-Pagan test of uniform crossover operator.

Uniform crossover operator							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.0553	0.81406	Do not reject H_0	16	0.1785	0.67264	Do not reject H_0
2	0.5092	0.47546	Do not reject H_0	17	1.7755	0.1827	Do not reject H_0
3	0.0786	0.77911	Do not reject H_0	18	1.3373	0.24751	Do not reject H_0
4	6.8922	0.00865	Reject H_0	19	3.4326	0.06392	Do not reject H_0
5	0.0366	0.84827	Do not reject H_0	20	0.7026	0.40191	Do not reject H_0
6	0.4571	0.49897	Do not reject H_0	21	0.4510	0.50183	Do not reject H_0
7	1.2284	0.26773	Do not reject H_0	22	0.5453	0.46022	Do not reject H_0
8	0.3459	0.55642	Do not reject H_0	23	0.0025	0.95987	Do not reject H_0
9	0.0028	0.95762	Do not reject H_0	24	2.1037	0.14694	Do not reject H_0
10	0.8992	0.34300	Do not reject H_0	25	0.0109	0.91651	Do not reject H_0
11	0.0420	0.83751	Do not reject H_0	26	3.4221	0.06433	Do not reject H_0
12	3.2174	0.07286	Do not reject H_0	27	0.6643	0.41501	Do not reject H_0
13	3.4863	0.06188	Do not reject H_0	28	0.7637	0.38217	Do not reject H_0
14	0.1154	0.73401	Do not reject H_0	29	0.9381	0.33276	Do not reject H_0
15	0.1785	0.67264	Do not reject H_0	30	0.0424	0.83676	Do not reject H_0

Stationarity

Table 10.49: KPSS test of mutation rate in the range [0.001,0.249].

Mutation rate in the range [0.001,0.249]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
Mutation rate in the range [0.001,0.249]							
1	0.0541	> 0.1	Do not reject H_0	16	0.0396	> 0.1	Do not reject H_0
2	0.0452	> 0.1	Do not reject H_0	17	0.0374	> 0.1	Do not reject H_0
3	0.0414	> 0.1	Do not reject H_0	18	0.0453	> 0.1	Do not reject H_0
4	0.0412	> 0.1	Do not reject H_0	19	0.1634	0.0355	Reject H_0
5	0.051	> 0.1	Do not reject H_0	20	0.0955	> 0.1	Do not reject H_0
6	0.0405	> 0.1	Do not reject H_0	21	0.0638	> 0.1	Do not reject H_0
7	0.0395	> 0.1	Do not reject H_0	22	0.0408	> 0.1	Do not reject H_0
8	0.0342	> 0.1	Do not reject H_0	23	0.0318	> 0.1	Do not reject H_0
9	0.038	> 0.1	Do not reject H_0	24	0.0665	> 0.1	Do not reject H_0
10	0.0199	> 0.1	Do not reject H_0	25	0.0338	> 0.1	Do not reject H_0
11	0.0538	> 0.1	Do not reject H_0	26	0.0477	> 0.1	Do not reject H_0
12	0.0548	> 0.1	Do not reject H_0	27	0.1436	0.0544	Do not reject H_0
13	0.0469	> 0.1	Do not reject H_0	28	0.0687	> 0.1	Do not reject H_0
14	0.1498	> 0.1	Do not reject H_0	29	0.0669	> 0.1	Do not reject H_0
15	0.0383	> 0.1	Do not reject H_0	30	0.0579	> 0.1	Do not reject H_0

Table 10.50: KPSS test of mutation rate in the range [0.25,0.49].

Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
Mutation rate in the range [0.25,0.49]							
1	0.0979	> 0.1	Do not reject H_0	16	0.063	> 0.1	Do not reject H_0
2	0.0672	> 0.1	Do not reject H_0	17	0.051	> 0.1	Do not reject H_0
3	0.0762	> 0.1	Do not reject H_0	18	0.0992	> 0.1	Do not reject H_0
4	0.0436	> 0.1	Do not reject H_0	19	0.0607	> 0.1	Do not reject H_0
5	0.0301	> 0.1	Do not reject H_0	20	0.0453	> 0.1	Do not reject H_0
6	0.1386	0.0636	Do not reject H_0	21	0.1322	0.0754	Do not reject H_0
7	0.0452	> 0.1	Do not reject H_0	22	0.0415	> 0.1	Do not reject H_0
8	0.0258	> 0.1	Do not reject H_0	23	0.0302	> 0.1	Do not reject H_0
9	0.0805	> 0.1	Do not reject H_0	24	0.1061	> 0.1	Do not reject H_0
10	0.0422	> 0.1	Do not reject H_0	25	0.0322	> 0.1	Do not reject H_0
11	0.1038	> 0.1	Do not reject H_0	26	0.0443	> 0.1	Do not reject H_0
12	0.0373	> 0.1	Do not reject H_0	27	0.0593	> 0.1	Do not reject H_0
13	0.0398	> 0.1	Do not reject H_0	28	0.029	> 0.1	Do not reject H_0
14	0.0673	> 0.1	Do not reject H_0	29	0.0287	> 0.1	Do not reject H_0
15	0.0865	> 0.1	Do not reject H_0	30	0.037	> 0.1	Do not reject H_0

Table 10.51: KPSS test of crossover rate in the range [0.6,0.79].

Crossover rate in the range [0.6,0.79]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.1741	0.026	Reject H_0	16	0.0743	> 0.1	Do not reject H_0
2	0.0375	> 0.1	Do not reject H_0	17	0.0494	> 0.1	Do not reject H_0
3	0.0375	> 0.1	Do not reject H_0	18	0.1129	> 0.1	Do not reject H_0
4	0.0529	> 0.1	Do not reject H_0	19	0.0316	> 0.1	Do not reject H_0
5	0.0467	> 0.1	Do not reject H_0	20	0.1617	0.0369	Reject H_0
6	0.0322	> 0.1	Do not reject H_0	21	0.0745	> 0.1	Do not reject H_0
7	0.0459	> 0.1	Do not reject H_0	22	0.0515	> 0.1	Do not reject H_0
8	0.0543	> 0.1	Do not reject H_0	23	0.0275	> 0.1	Do not reject H_0
9	0.0855	> 0.1	Do not reject H_0	24	0.0413	> 0.1	Do not reject H_0
10	0.0442	> 0.1	Do not reject H_0	25	0.0262	> 0.1	Do not reject H_0
11	0.0769	> 0.1	Do not reject H_0	26	0.049	> 0.1	Do not reject H_0
12	0.0603	> 0.1	Do not reject H_0	27	0.0352	> 0.1	Do not reject H_0
13	0.0401	> 0.1	Do not reject H_0	28	0.0447	> 0.1	Do not reject H_0
14	0.0637	> 0.1	Do not reject H_0	29	0.0588	> 0.1	Do not reject H_0
15	0.0295	> 0.1	Do not reject H_0	30	0.0348	> 0.1	Do not reject H_0

Table 10.52: KPSS test of crossover rate in the range [0.8,0.99].

Crossover rate in the range [0.8,0.99]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.0637	> 0.1	Do not reject H_0	16	0.0424	> 0.1	Do not reject H_0
2	0.1348	0.0706	Do not reject H_0	17	0.0736	> 0.1	Do not reject H_0
3	0.0343	> 0.1	Do not reject H_0	18	0.049	> 0.1	Do not reject H_0
4	0.0332	> 0.1	Do not reject H_0	19	0.0329	> 0.1	Do not reject H_0
5	0.0423	> 0.1	Do not reject H_0	20	0.0483	> 0.1	Do not reject H_0
6	0.0688	> 0.1	Do not reject H_0	21	0.046	> 0.1	Do not reject H_0
7	0.0221	> 0.1	Do not reject H_0	22	0.0383	> 0.1	Do not reject H_0
8	0.0375	> 0.1	Do not reject H_0	23	0.0428	> 0.1	Do not reject H_0
9	0.0464	> 0.1	Do not reject H_0	24	0.157	0.0408	Reject H_0
10	0.0372	> 0.1	Do not reject H_0	25	0.0189	> 0.1	Do not reject H_0
11	0.0377	> 0.1	Do not reject H_0	26	0.0515	> 0.1	Do not reject H_0
12	0.0462	> 0.1	Do not reject H_0	27	0.0786	> 0.1	Do not reject H_0
13	0.0648	> 0.1	Do not reject H_0	28	0.0397	> 0.1	Do not reject H_0
14	0.0746	> 0.1	Do not reject H_0	29	0.1793	0.0237	Reject H_0
15	0.0634	> 0.1	Do not reject H_0	30	0.0586	> 0.1	Do not reject H_0

Table 10.53: KPSS test of population size in the range [20,60].

Population size in the range [20,60]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.3125	0.01	Reject H_0	16	0.1583	0.0397	Reject H_0
2	0.1636	0.0352	Reject H_0	17	0.2241	0.01	Reject H_0
3	0.3420	0.01	Reject H_0	18	0.5096	0.01	Reject H_0
4	0.5167	0.01	Reject H_0	19	0.3007	0.01	Reject H_0
5	0.1607	0.0377	Reject H_0	20	0.1666	0.0327	Reject H_0
6	0.3367	0.01	Reject H_0	21	0.3414	0.01	Reject H_0
7	0.0720	> 0.1	Do not reject H_0	22	0.3053	0.01	Reject H_0
8	0.2176	0.01	Reject H_0	23	0.2107	0.0119	Reject H_0
9	0.1058	> 0.1	Do not reject H_0	24	0.2884	0.01	Reject H_0
10	0.2820	0.01	Reject H_0	25	0.2098	0.0123	Reject H_0
11	0.1990	0.0163	Reject H_0	26	0.1053	> 0.1	Do not reject H_0
12	0.2092	0.0125	Reject H_0	27	0.2647	0.01	Reject H_0
13	0.4038	0.01	Reject H_0	28	0.1438	0.0539	Do not reject H_0
14	0.1619	0.0366	Reject H_0	29	0.2134	0.0109	Reject H_0
15	0.1583	0.0397	Reject H_0	30	0.4566	0.01	Reject H_0

Table 10.54: KPSS test of population size in the range [61,100].

Population size in the range [61,100]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.2021	0.0151	Reject H_0	16	0.1941	0.018	Reject H_0
2	0.4157	0.01	Reject H_0	17	0.3148	0.01	Reject H_0
3	0.1437	0.0542	Do not reject H_0	18	0.2723	0.01	Reject H_0
4	0.3633	0.01	Reject H_0	19	0.1282	0.082	Do not reject H_0
5	0.3332	0.01	Reject H_0	20	0.2656	0.01	Reject H_0
6	0.3105	0.01	Reject H_0	21	0.2337	0.01	Reject H_0
7	0.1916	0.0191	Reject H_0	22	0.3399	0.01	Reject H_0
8	0.3179	0.01	Reject H_0	23	0.2252	0.01	Reject H_0
9	0.1711	0.0290	Reject H_0	24	0.2610	0.01	Reject H_0
10	0.2000	0.0159	Reject H_0	25	0.2975	0.01	Reject H_0
11	0.0802	> 0.1	Do not reject H_0	26	0.1348	0.070	Do not reject H_0
12	0.2009	0.0156	Reject H_0	27	0.3139	0.01	Reject H_0
13	0.3477	0.01	Reject H_0	28	0.2260	0.01	Reject H_0
14	0.2594	0.01	Reject H_0	29	0.1933	0.018	Reject H_0
15	0.1941	0.0181	Reject H_0	30	0.3252	0.01	Reject H_0

Table 10.55: KPSS test of mating pool size in the range [0.1,0.39].

Mating pool size in the range [0.1,0.39]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.0599	> 0.1	Do not reject H_0	16	0.0452	> 0.1	Do not reject H_0
2	0.0447	> 0.1	Do not reject H_0	17	0.0584	> 0.1	Do not reject H_0
3	0.0442	> 0.1	Do not reject H_0	18	0.0259	> 0.1	Do not reject H_0
4	0.0846	> 0.1	Do not reject H_0	19	0.1728	0.0276	Reject H_0
5	0.0304	> 0.1	Do not reject H_0	20	0.0743	> 0.1	Do not reject H_0
6	0.0759	> 0.1	Do not reject H_0	21	0.0222	> 0.1	Do not reject H_0
7	0.0666	> 0.1	Do not reject H_0	22	0.0396	> 0.1	Do not reject H_0
8	0.0618	> 0.1	Do not reject H_0	23	0.0306	> 0.1	Do not reject H_0
9	0.0879	> 0.1	Do not reject H_0	24	0.0608	> 0.1	Do not reject H_0
10	0.0558	> 0.1	Do not reject H_0	25	0.0312	> 0.1	Do not reject H_0
11	0.0863	> 0.1	Do not reject H_0	26	0.0826	> 0.1	Do not reject H_0
12	0.1276	0.0839	Do not reject H_0	27	0.0450	> 0.1	Do not reject H_0
13	0.0411	> 0.1	Do not reject H_0	28	0.0749	> 0.1	Do not reject H_0
14	0.0920	> 0.1	Do not reject H_0	29	0.0265	> 0.1	Do not reject H_0
15	0.0452	> 0.1	Do not reject H_0	30	0.0952	> 0.1	Do not reject H_0

Table 10.56: KPSS test of mating pool size in the range [0.4,0.69].

Mating pool size in the range [0.4,0.69]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.0544	> 0.1	Do not reject H_0	16	0.0611	> 0.1	Do not reject H_0
2	0.0451	> 0.1	Do not reject H_0	17	0.0432	> 0.1	Do not reject H_0
3	0.0267	> 0.1	Do not reject H_0	18	0.0406	> 0.1	Do not reject H_0
4	0.0212	> 0.1	Do not reject H_0	19	0.0352	> 0.1	Do not reject H_0
5	0.0375	> 0.1	Do not reject H_0	20	0.0253	> 0.1	Do not reject H_0
6	0.0573	> 0.1	Do not reject H_0	21	0.0235	> 0.1	Do not reject H_0
7	0.0489	> 0.1	Do not reject H_0	22	0.0580	> 0.1	Do not reject H_0
8	0.0672	> 0.1	Do not reject H_0	23	0.0332	> 0.1	Do not reject H_0
9	0.0254	> 0.1	Do not reject H_0	24	0.0451	> 0.1	Do not reject H_0
10	0.0566	> 0.1	Do not reject H_0	25	0.0778	> 0.1	Do not reject H_0
11	0.0438	> 0.1	Do not reject H_0	26	0.0636	> 0.1	Do not reject H_0
12	0.0538	> 0.1	Do not reject H_0	27	0.0236	> 0.1	Do not reject H_0
13	0.0512	> 0.1	Do not reject H_0	28	0.0256	> 0.1	Do not reject H_0
14	0.0839	> 0.1	Do not reject H_0	29	0.0185	> 0.1	Do not reject H_0
15	0.0611	> 0.1	Do not reject H_0	30	0.0643	> 0.1	Do not reject H_0

Table 10.57: KPSS test of single-point mutation operator.

Single-point mutation operator							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.0371	> 0.1	Do not reject H_0	16	0.0254	> 0.1	Do not reject H_0
2	0.0439	> 0.1	Do not reject H_0	17	0.0255	> 0.1	Do not reject H_0
3	0.0364	> 0.1	Do not reject H_0	18	0.0707	> 0.1	Do not reject H_0
4	0.0487	> 0.1	Do not reject H_0	19	0.0960	> 0.1	Do not reject H_0
5	0.0356	> 0.1	Do not reject H_0	20	0.0378	> 0.1	Do not reject H_0
6	0.0514	> 0.1	Do not reject H_0	21	0.0581	> 0.1	Do not reject H_0
7	0.0626	> 0.1	Do not reject H_0	22	0.0299	> 0.1	Do not reject H_0
8	0.0379	> 0.1	Do not reject H_0	23	0.0496	> 0.1	Do not reject H_0
9	0.0716	> 0.1	Do not reject H_0	24	0.0463	> 0.1	Do not reject H_0
10	0.0783	> 0.1	Do not reject H_0	25	0.0382	> 0.1	Do not reject H_0
11	0.0683	> 0.1	Do not reject H_0	26	0.0571	> 0.1	Do not reject H_0
12	0.0771	> 0.1	Do not reject H_0	27	0.0800	> 0.1	Do not reject H_0
13	0.0372	> 0.1	Do not reject H_0	28	0.0294	> 0.1	Do not reject H_0
14	0.2071	0.0133	Reject H_0	29	0.1288	0.0817	Do not reject H_0
15	0.0254	> 0.1	Do not reject H_0	30	0.0486	> 0.1	Do not reject H_0

Table 10.58: KPSS test of uniform mutation operator.

Uniform mutation operator							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.0471	> 0.1	Do not reject H_0	16	0.0369	> 0.1	Do not reject H_0
2	0.0550	> 0.1	Do not reject H_0	17	0.1324	0.0750	Do not reject H_0
3	0.0863	> 0.1	Do not reject H_0	18	0.2167	0.01	Reject H_0
4	0.1620	0.0366	Reject H_0	19	0.0278	> 0.1	Do not reject H_0
5	0.0275	> 0.1	Do not reject H_0	20	0.0305	> 0.1	Do not reject H_0
6	0.0290	> 0.1	Do not reject H_0	21	0.0612	> 0.1	Do not reject H_0
7	0.0443	> 0.1	Do not reject H_0	22	0.0800	> 0.1	Do not reject H_0
8	0.0972	> 0.1	Do not reject H_0	23	0.0263	> 0.1	Do not reject H_0
9	0.0587	> 0.1	Do not reject H_0	24	0.0556	> 0.1	Do not reject H_0
10	0.0992	> 0.1	Do not reject H_0	25	0.0540	> 0.1	Do not reject H_0
11	0.1226	0.0931	Do not reject H_0	26	0.0551	> 0.1	Do not reject H_0
12	0.0538	> 0.1	Do not reject H_0	27	0.0411	> 0.1	Do not reject H_0
13	0.0278	> 0.1	Do not reject H_0	28	0.0449	> 0.1	Do not reject H_0
14	0.0517	> 0.1	Do not reject H_0	29	0.0437	> 0.1	Do not reject H_0
15	0.0369	> 0.1	Do not reject H_0	30	0.0952	> 0.1	Do not reject H_0

Table 10.59: KPSS test of single-point crossover operator.

Single-point crossover operator							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.0471	> 0.1	Do not reject H_0	16	0.0369	> 0.1	Do not reject H_0
2	0.0550	> 0.1	Do not reject H_0	17	0.1324	0.0750	Do not reject H_0
3	0.0863	> 0.1	Do not reject H_0	18	0.2167	0.01	Reject H_0
4	0.1620	0.0366	Reject H_0	19	0.0278	> 0.1	Do not reject H_0
5	0.0275	> 0.1	Do not reject H_0	20	0.0305	> 0.1	Do not reject H_0
6	0.0290	> 0.1	Do not reject H_0	21	0.0612	> 0.1	Do not reject H_0
7	0.0443	> 0.1	Do not reject H_0	22	0.0800	> 0.1	Do not reject H_0
8	0.0972	> 0.1	Do not reject H_0	23	0.0263	> 0.1	Do not reject H_0
9	0.0587	> 0.1	Do not reject H_0	24	0.0556	> 0.1	Do not reject H_0
10	0.0992	> 0.1	Do not reject H_0	25	0.0540	> 0.1	Do not reject H_0
11	0.1226	0.0931	Do not reject H_0	26	0.0551	> 0.1	Do not reject H_0
12	0.0538	> 0.1	Do not reject H_0	27	0.0411	> 0.1	Do not reject H_0
13	0.0278	> 0.1	Do not reject H_0	28	0.0449	> 0.1	Do not reject H_0
14	0.0517	> 0.1	Do not reject H_0	29	0.0437	> 0.1	Do not reject H_0
15	0.0369	> 0.1	Do not reject H_0	30	0.0952	> 0.1	Do not reject H_0

Table 10.60: KPSS test of uniform crossover operator.

Uniform crossover operator							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.0471	> 0.1	Do not reject H_0	16	0.0369	> 0.1	Do not reject H_0
2	0.0550	> 0.1	Do not reject H_0	17	0.1324	0.0750	Do not reject H_0
3	0.0863	> 0.1	Do not reject H_0	18	0.2167	0.01	Reject H_0
4	0.1620	0.0366	Reject H_0	19	0.0278	> 0.1	Do not reject H_0
5	0.0275	> 0.1	Do not reject H_0	20	0.0305	> 0.1	Do not reject H_0
6	0.0290	> 0.1	Do not reject H_0	21	0.0612	> 0.1	Do not reject H_0
7	0.0443	> 0.1	Do not reject H_0	22	0.0800	> 0.1	Do not reject H_0
8	0.0972	> 0.1	Do not reject H_0	23	0.0263	> 0.1	Do not reject H_0
9	0.0587	> 0.1	Do not reject H_0	24	0.0556	> 0.1	Do not reject H_0
10	0.0992	> 0.1	Do not reject H_0	25	0.0540	> 0.1	Do not reject H_0
11	0.1226	0.0931	Do not reject H_0	26	0.0551	> 0.1	Do not reject H_0
12	0.0538	> 0.1	Do not reject H_0	27	0.0411	> 0.1	Do not reject H_0
13	0.0278	> 0.1	Do not reject H_0	28	0.0449	> 0.1	Do not reject H_0
14	0.0517	> 0.1	Do not reject H_0	29	0.0437	> 0.1	Do not reject H_0
15	0.0369	> 0.1	Do not reject H_0	30	0.0952	> 0.1	Do not reject H_0

Characteristics of the Effect of Parameter Values - Experiments with Four Ranges

Linearity

Table 10.61: Linearity test of the mutation rate in the range [0.001,0.1249].

Mutation rate in the range [0.001,0.1249]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	32.846	1.12e-07	Do not reject H_0	16	66.422	1.28e-12	Do not reject H_0
2	80.321	2.33e-14	Do not reject H_0	17	13.253	0.000438	Do not reject H_0
3	38.693	1.25e-08	Do not reject H_0	18	11.611	0.000955	Do not reject H_0
4	69.39	5.31e-13	Do not reject H_0	19	31.005	2.29e-07	Do not reject H_0
5	2.2139	0.14001	Reject H_0	20	25.067	2.47e-06	Do not reject H_0
6	14.837	0.00021	Do not reject H_0	21	140.10	1.56e-20	Do not reject H_0
7	18.542	3.97e-05	Do not reject H_0	22	30.601	2.68e-07	Do not reject H_0
8	24.678	2.90e-06	Do not reject H_0	23	53.118	8.44e-11	Do not reject H_0
9	17.914	5.25e-05	Do not reject H_0	24	34.739	5.46e-08	Do not reject H_0
10	78.77	3.58e-14	Do not reject H_0	25	15.733	0.000139	Do not reject H_0
11	26.398	1.43e-06	Do not reject H_0	26	77.022	5.86e-14	Do not reject H_0
12	34.194	6.72e-08	Do not reject H_0	27	43.198	2.48e-09	Do not reject H_0
13	71.126	3.18e-13	Do not reject H_0	28	27.556	8.99e-07	Do not reject H_0
14	23.906	4.00e-06	Do not reject H_0	29	66.002	1.46e-12	Do not reject H_0
15	66.422	1.28e-12	Do not reject H_0	30	22.658	6.76e-06	Do not reject H_0

Table 10.62: Linearity test of the mutation rate in the range [0.125,0.249].

Mutation rate in the range [0.125,0.249]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	188.91	1.68e-24	Do not reject H_0	16	63.696	2.94e-12	Do not reject H_0
2	85.539	5.62e-15	Do not reject H_0	17	243.31	3.47e-28	Do not reject H_0
3	96.7	3.06e-16	Do not reject H_0	18	219.34	1.21e-26	Do not reject H_0
4	140.92	1.31e-20	Do not reject H_0	19	124.60	4.23e-19	Do not reject H_0
5	218.96	1.29e-26	Do not reject H_0	20	123.61	5.27e-19	Do not reject H_0
6	72.126	2.38e-13	Do not reject H_0	21	73.109	1.79e-13	Do not reject H_0
7	92.699	8.52e-16	Do not reject H_0	22	160.86	2.59e-22	Do not reject H_0
8	78.174	4.24e-14	Do not reject H_0	23	86.149	4.77e-15	Do not reject H_0
9	125.98	3.13e-19	Do not reject H_0	24	113.39	5.36e-18	Do not reject H_0
10	116.75	2.47e-18	Do not reject H_0	25	222.43	7.59e-27	Do not reject H_0
11	139.71	1.68e-20	Do not reject H_0	26	90.811	1.39e-15	Do not reject H_0
12	153.96	9.74e-22	Do not reject H_0	27	87.665	3.18e-15	Do not reject H_0
13	93.809	6.40e-16	Do not reject H_0	28	107.78	2.01e-17	Do not reject H_0
14	135.22	4.30e-20	Do not reject H_0	29	148.55	2.82e-21	Do not reject H_0
15	63.696	2.94e-12	Do not reject H_0	30	101.80	8.58e-17	Do not reject H_0

Table 10.63: Linearity test of the mutation rate in the range [0.25,0.3749].

Mutation rate in the range [0.25,0.3749]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	132.04	8.43e-20	Do not reject H_0	16	53.159	8.33e-11	Do not reject H_0
2	47.902	4.84e-10	Do not reject H_0	17	70.133	4.26e-13	Do not reject H_0
3	95.07	4.63e-16	Do not reject H_0	18	82.28	1.36e-14	Do not reject H_0
4	137.81	2.50e-20	Do not reject H_0	19	77.717	4.82e-14	Do not reject H_0
5	43.564	2.18e-09	Do not reject H_0	20	32.381	1.34e-07	Do not reject H_0
6	138.23	2.29e-20	Do not reject H_0	21	97.052	2.80e-16	Do not reject H_0
7	79.76	2.72e-14	Do not reject H_0	22	84.97	6.55e-15	Do not reject H_0
8	94.147	5.87e-16	Do not reject H_0	23	76.557	6.68e-14	Do not reject H_0
9	70.566	3.75e-13	Do not reject H_0	24	66.602	1.21e-12	Do not reject H_0
10	77.23	5.52e-14	Do not reject H_0	25	174.75	2.00e-23	Do not reject H_0
11	49.532	2.78e-10	Do not reject H_0	26	117.36	2.14e-18	Do not reject H_0
12	90.895	1.36e-15	Do not reject H_0	27	99.44	1.54e-16	Do not reject H_0
13	106.67	2.62e-17	Do not reject H_0	28	74.816	1.09e-13	Do not reject H_0
14	51.158	1.61e-10	Do not reject H_0	29	63.636	2.99e-12	Do not reject H_0
15	53.159	8.33e-11	Do not reject H_0	30	156.15	6.37e-22	Do not reject H_0

Table 10.64: Linearity test of the mutation rate [0.375,0.49].

Mutation rate in the range [0.25,0.3749]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	66.543	1.24e-12	Do not reject H_0	16	131.46	9.56e-20	Do not reject H_0
2	151.23	1.65e-21	Do not reject H_0	17	97.335	2.61e-16	Do not reject H_0
3	198.27	3.50e-25	Do not reject H_0	18	155.27	7.54e-22	Do not reject H_0
4	43.224	2.46e-09	Do not reject H_0	19	140.91	1.31e-20	Do not reject H_0
5	111.07	9.24e-18	Do not reject H_0	20	151.98	1.43e-21	Do not reject H_0
6	53.958	6.41e-11	Do not reject H_0	21	123.68	5.20e-19	Do not reject H_0
7	148.72	2.72e-21	Do not reject H_0	22	98.336	2.03e-16	Do not reject H_0
8	92.603	8.73e-16	Do not reject H_0	23	158.23	4.27e-22	Do not reject H_0
9	121.53	8.38e-19	Do not reject H_0	24	69.121	5.75e-13	Do not reject H_0
10	190.82	1.21e-24	Do not reject H_0	25	83.9	8.75e-15	Do not reject H_0
11	169.82	4.89e-23	Do not reject H_0	26	101.55	9.13e-17	Do not reject H_0
12	74.602	1.16e-13	Do not reject H_0	27	107.35	2.23e-17	Do not reject H_0
13	127.98	2.02e-19	Do not reject H_0	28	94.808	4.95e-16	Do not reject H_0
14	159.82	3.15e-22	Do not reject H_0	29	109.69	1.27e-17	Do not reject H_0
15	131.46	9.56e-20	Do not reject H_0	30	57.046	2.36e-11	Do not reject H_0

Table 10.65: Linearity test of the crossover rate in the range [0.6,0.69].

Crossover rate [0.6,0.69]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	97.87	2.28e-16	Do not reject H_0	16	127.6	2.16e-19	Do not reject H_0
2	94.54	5.31e-16	Do not reject H_0	17	75.25	9.67e-14	Do not reject H_0
3	92.08	1.00e-15	Do not reject H_0	18	179.6	8.35e-24	Do not reject H_0
4	52.67	9.76e-11	Do not reject H_0	19	86.02	4.93e-15	Do not reject H_0
5	82.97	1.12e-14	Do not reject H_0	20	142.2	9.99e-21	Do not reject H_0
6	93.23	7.43e-16	Do not reject H_0	21	154.4	8.90e-22	Do not reject H_0
7	119.4	1.34e-18	Do not reject H_0	22	58.37	1.54e-11	Do not reject H_0
8	75.37	9.36e-14	Do not reject H_0	23	38.02	1.60e-08	Do not reject H_0
9	100.3	1.21e-16	Do not reject H_0	24	52.04	1.20e-10	Do not reject H_0
10	38.32	1.43e-08	Do not reject H_0	25	104.8	4.05e-17	Do not reject H_0
11	73.09	1.79e-13	Do not reject H_0	26	127.7	2.14e-19	Do not reject H_0
12	138.9	1.99e-20	Do not reject H_0	27	38.43	1.38e-08	Do not reject H_0
13	70.21	4.16e-13	Do not reject H_0	28	37.67	1.82e-08	Do not reject H_0
14	98.36	2.01e-16	Do not reject H_0	29	94.59	5.23e-16	Do not reject H_0
15	127.6	2.16e-19	Do not reject H_0	30	82.18	1.39e-14	Do not reject H_0

Table 10.66: Linearity test of the crossover rate in the range [0.7,0.79].

Crossover rate [0.7,0.79]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	60.12	8.91e-12	Do not reject H_0	16	66.61	1.21e-12	Do not reject H_0
2	112.7	6.23e-18	Do not reject H_0	17	70.06	4.35e-13	Do not reject H_0
3	48.95	3.38e-10	Do not reject H_0	18	60.13	8.88e-12	Do not reject H_0
4	87.47	3.35e-15	Do not reject H_0	19	22.66	6.74e-06	Do not reject H_0
5	90.61	1.46e-15	Do not reject H_0	20	158.7	3.88e-22	Do not reject H_0
6	73.23	1.72e-13	Do not reject H_0	21	93.17	7.53e-16	Do not reject H_0
7	71.92	2.52e-13	Do not reject H_0	22	42.13	3.62e-09	Do not reject H_0
8	139.1	1.90e-20	Do not reject H_0	23	127.4	2.29e-19	Do not reject H_0
9	141.7	1.10e-20	Do not reject H_0	24	143.9	7.14e-21	Do not reject H_0
10	67.28	9.95e-13	Do not reject H_0	25	117.0	2.29e-18	Do not reject H_0
11	145.8	4.84e-21	Do not reject H_0	26	105.5	3.48e-17	Do not reject H_0
12	36.64	2.67e-08	Do not reject H_0	27	107.8	1.97e-17	Do not reject H_0
13	98.98	1.72e-16	Do not reject H_0	28	82.75	1.19e-14	Do not reject H_0
14	171.8	3.38e-23	Do not reject H_0	29	38.46	1.36e-08	Do not reject H_0
15	66.61	1.21e-12	Do not reject H_0	30	130.5	1.15e-19	Do not reject H_0

Table 10.67: Linearity test of the crossover rate in the range [0.8,0.89].

Crossover rate [0.8,0.89]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	61.57	5.65e-12	Do not reject H_0	16	59.84	9.73e-12	Do not reject H_0
2	127.4	2.26e-19	Do not reject H_0	17	149.1	2.55e-21	Do not reject H_0
3	57.97	1.75e-11	Do not reject H_0	18	104.1	4.92e-17	Do not reject H_0
4	134.7	4.72e-20	Do not reject H_0	19	124.6	4.16e-19	Do not reject H_0
5	105.2	3.73e-17	Do not reject H_0	20	87.65	3.19e-15	Do not reject H_0
6	128.5	1.80e-19	Do not reject H_0	21	55.2	4.27e-11	Do not reject H_0
7	93.55	6.84e-16	Do not reject H_0	22	150.6	1.84e-21	Do not reject H_0
8	38.25	1.47e-08	Do not reject H_0	23	136.9	2.99e-20	Do not reject H_0
9	144.1	6.76e-21	Do not reject H_0	24	108.2	1.82e-17	Do not reject H_0
10	178.3	1.05e-23	Do not reject H_0	25	119.8	1.21e-18	Do not reject H_0
11	23.56	4.62e-06	Do not reject H_0	26	78.15	4.26e-14	Do not reject H_0
12	68.37	7.18e-13	Do not reject H_0	27	87.00	3.80e-15	Do not reject H_0
13	142.8	8.79e-21	Do not reject H_0	28	156.8	5.58e-22	Do not reject H_0
14	75.56	8.85e-14	Do not reject H_0	29	142.6	9.24e-21	Do not reject H_0
15	59.84	9.73e-12	Do not reject H_0	30	73.2	1.74e-13	Do not reject H_0

Table 10.68: Linearity test of the crossover rate in the range [0.9,0.99].

Crossover rate in the range [0.9,0.99]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	152.9	1.18e-21	Do not reject H_0	16	79.19	3.19e-14	Do not reject H_0
2	40.37	6.81e-09	Do not reject H_0	17	133.2	6.50e-20	Do not reject H_0
3	74.29	1.27e-13	Do not reject H_0	18	69.36	5.35e-13	Do not reject H_0
4	90.79	1.39e-15	Do not reject H_0	19	137.6	2.56e-20	Do not reject H_0
5	67.74	8.65e-13	Do not reject H_0	20	95.51	4.13e-16	Do not reject H_0
6	33.48	8.80e-08	Do not reject H_0	21	46.79	7.08e-10	Do not reject H_0
7	135.0	4.42e-20	Do not reject H_0	22	61.16	6.44e-12	Do not reject H_0
8	135.6	3.96e-20	Do not reject H_0	23	140.8	1.34e-20	Do not reject H_0
9	19.18	2.99e-05	Do not reject H_0	24	112.9	5.98e-18	Do not reject H_0
10	80.93	1.96e-14	Do not reject H_0	25	40.61	6.25e-09	Do not reject H_0
11	101.5	9.08e-17	Do not reject H_0	26	26.66	1.28e-06	Do not reject H_0
12	71.29	3.03e-13	Do not reject H_0	27	111.7	7.96e-18	Do not reject H_0
13	110.0	1.18e-17	Do not reject H_0	28	121.7	8.01e-19	Do not reject H_0
14	56.33	2.96e-11	Do not reject H_0	29	94.78	4.98e-16	Do not reject H_0
15	79.19	3.19e-14	Do not reject H_0	30	33.73	8.01e-08	Do not reject H_0

Table 10.69: Linearity test of the population size in the range [20,40].

Population size in the range [20,40]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	67.637	8.94e-13	Do not reject H_0	16	37.215	2.16e-08	Do not reject H_0
2	131.97	8.56e-20	Do not reject H_0	17	56.606	2.72e-11	Do not reject H_0
3	83.68	9.28e-15	Do not reject H_0	18	39.413	9.65e-09	Do not reject H_0
4	91.44	1.18e-15	Do not reject H_0	19	84.527	7.38e-15	Do not reject H_0
5	31.594	1.82e-07	Do not reject H_0	20	116.48	2.63e-18	Do not reject H_0
6	73.723	1.50e-13	Do not reject H_0	21	18.100	4.83e-05	Do not reject H_0
7	65.494	1.70e-12	Do not reject H_0	22	32.295	1.39e-07	Do not reject H_0
8	64.965	2.00e-12	Do not reject H_0	23	45.773	1.00e-09	Do not reject H_0
9	68.291	7.35e-13	Do not reject H_0	24	86.495	4.35e-15	Do not reject H_0
10	39.449	9.53e-09	Do not reject H_0	25	81.798	1.55e-14	Do not reject H_0
11	40.262	7.09e-09	Do not reject H_0	26	61.993	4.97e-12	Do not reject H_0
12	80.124	2.46e-14	Do not reject H_0	27	90.424	1.53e-15	Do not reject H_0
13	71.946	2.51e-13	Do not reject H_0	28	56.043	3.26e-11	Do not reject H_0
14	33.617	8.37e-08	Do not reject H_0	29	101.47	9.31e-17	Do not reject H_0
15	37.215	2.16e-08	Do not reject H_0	30	65.785	1.55e-12	Do not reject H_0

Table 10.70: Linearity test of the population size in the range [41,60].

Population size in the range [41,60]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	21.060	1.33e-05	Do not reject H_0	16	152.94	1.18e-21	Do not reject H_0
2	106.10	3.01e-17	Do not reject H_0	17	109.22	1.42e-17	Do not reject H_0
3	101.89	8.40e-17	Do not reject H_0	18	85.791	5.25e-15	Do not reject H_0
4	171.47	3.62e-23	Do not reject H_0	19	75.64	8.67e-14	Do not reject H_0
5	145.61	5.08e-21	Do not reject H_0	20	50.342	2.12e-10	Do not reject H_0
6	152.52	1.28e-21	Do not reject H_0	21	115.57	3.23e-18	Do not reject H_0
7	90.112	1.67e-15	Do not reject H_0	22	64.946	2.01e-12	Do not reject H_0
8	78.617	3.74e-14	Do not reject H_0	23	184.14	3.82e-24	Do not reject H_0
9	151.05	1.72e-21	Do not reject H_0	24	65.865	1.52e-12	Do not reject H_0
10	58.629	1.42e-11	Do not reject H_0	25	89.079	2.19e-15	Do not reject H_0
11	132.91	7.01e-20	Do not reject H_0	26	17.778	5.57e-05	Do not reject H_0
12	116.54	2.59e-18	Do not reject H_0	27	115.51	3.29e-18	Do not reject H_0
13	75.248	9.69e-14	Do not reject H_0	28	167.19	7.94e-23	Do not reject H_0
14	134.16	5.37e-20	Do not reject H_0	29	118.04	1.84e-18	Do not reject H_0
15	152.94	1.18e-21	Do not reject H_0	30	33.558	8.56e-08	Do not reject H_0

Table 10.71: Linearity test of the population size in the range [61,80].

Population size in the range [61,80]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	153.70	1.02e-21	Do not reject H_0	16	138.18	2.31e-20	Do not reject H_0
2	65.238	1.84e-12	Do not reject H_0	17	91.645	1.12e-15	Do not reject H_0
3	64.725	2.15e-12	Do not reject H_0	18	151.40	1.60e-21	Do not reject H_0
4	58.294	1.58e-11	Do not reject H_0	19	113.68	5.01e-18	Do not reject H_0
5	54.059	6.20e-11	Do not reject H_0	20	78.36	4.02e-14	Do not reject H_0
6	81.837	1.53e-14	Do not reject H_0	21	224.28	5.73e-27	Do not reject H_0
7	121.48	8.48e-19	Do not reject H_0	22	94.785	4.98e-16	Do not reject H_0
8	137.74	2.54e-20	Do not reject H_0	23	61.335	6.10e-12	Do not reject H_0
9	88.013	2.90e-15	Do not reject H_0	24	127.81	2.10e-19	Do not reject H_0
10	69.965	4.48e-13	Do not reject H_0	25	64.621	2.22e-12	Do not reject H_0
11	118.80	1.54e-18	Do not reject H_0	26	105.78	3.25e-17	Do not reject H_0
12	72.675	2.03e-13	Do not reject H_0	27	76.251	7.29e-14	Do not reject H_0
13	80.422	2.26e-14	Do not reject H_0	28	108.41	1.73e-17	Do not reject H_0
14	57.033	2.37e-11	Do not reject H_0	29	40.206	7.24e-09	Do not reject H_0
15	138.18	2.31e-20	Do not reject H_0	30	119.13	1.44e-18	Do not reject H_0

Table 10.72: Linearity test of the population size in the range [81,100].

Population size in the range [81,100]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	94.763	5.01e-16	Do not reject H_0	16	22.148	8.38e-06	Do not reject H_0
2	47.004	6.59e-10	Do not reject H_0	17	53.664	7.06e-11	Do not reject H_0
3	41.406	4.70e-09	Do not reject H_0	18	98.302	2.04e-16	Do not reject H_0
4	69.133	5.73e-13	Do not reject H_0	19	49.202	3.11e-10	Do not reject H_0
5	93.06	7.76e-16	Do not reject H_0	20	86.912	3.89e-15	Do not reject H_0
6	19.307	2.84e-05	Do not reject H_0	21	83.562	9.58e-15	Do not reject H_0
7	88.114	2.82e-15	Do not reject H_0	22	74.16	1.32e-13	Do not reject H_0
8	84.421	7.59e-15	Do not reject H_0	23	123.90	4.95e-19	Do not reject H_0
9	45.564	1.08e-09	Do not reject H_0	24	65.843	1.53e-12	Do not reject H_0
10	89.992	1.72e-15	Do not reject H_0	25	37.08	2.27e-08	Do not reject H_0
11	34.859	5.22e-08	Do not reject H_0	26	123.57	5.32e-19	Do not reject H_0
12	31.514	1.88e-07	Do not reject H_0	27	28.844	5.37e-07	Do not reject H_0
13	98.914	1.75e-16	Do not reject H_0	28	43.292	2.40e-09	Do not reject H_0
14	49.888	2.47e-10	Do not reject H_0	29	66.008	1.45e-12	Do not reject H_0
15	22.148	8.38e-06	Do not reject H_0	30	42.582	3.09e-09	Do not reject H_0

Table 10.73: Linearity test of the mating pool size in the range [0.1,0.249].

Population size in the range [0.1,0.249]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	115.0	3.65e-18	Do not reject H_0	16	148.1	3.07e-21	Do not reject H_0
2	146.7	4.02e-21	Do not reject H_0	17	81.31	1.77e-14	Do not reject H_0
3	79.66	2.80e-14	Do not reject H_0	18	46.31	8.35e-10	Do not reject H_0
4	100.8	1.08e-16	Do not reject H_0	19	122.4	6.87e-19	Do not reject H_0
5	61.61	5.60e-12	Do not reject H_0	20	135.4	4.12e-20	Do not reject H_0
6	94.00	6.08e-16	Do not reject H_0	21	55.05	4.49e-11	Do not reject H_0
7	89.08	2.18e-15	Do not reject H_0	22	12.99	0.000494	Do not reject H_0
8	17.05	7.70e-05	Do not reject H_0	23	12.22	0.000713	Do not reject H_0
9	66.44	1.27e-12	Do not reject H_0	24	116.7	2.49e-18	Do not reject H_0
10	89.39	2.01e-15	Do not reject H_0	25	107.4	2.20e-17	Do not reject H_0
11	36.08	3.29e-08	Do not reject H_0	26	34.52	5.92e-08	Do not reject H_0
12	45.12	1.26e-09	Do not reject H_0	27	120.8	9.78e-19	Do not reject H_0
13	65.90	1.50e-12	Do not reject H_0	28	54.05	6.21e-11	Do not reject H_0
14	172.4	3.06e-23	Do not reject H_0	29	51.59	1.39e-10	Do not reject H_0
15	148.1	3.07e-21	Do not reject H_0	30	50.63	1.92e-10	Do not reject H_0

Table 10.74: Linearity test of the mating pool size in the range [0.25,0.39].

Population size in the range [0.25,0.39]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	114.61	4.0426e-18	Do not reject H_0	16	84.586	7.2671e-15	Do not reject H_0
2	128.09	1.9778e-19	Do not reject H_0	17	127.61	2.191e-19	Do not reject H_0
3	143.99	7.0475e-21	Do not reject H_0	18	103.15	6.1648e-17	Do not reject H_0
4	151.53	1.5671e-21	Do not reject H_0	19	148.18	3.0367e-21	Do not reject H_0
5	104.58	4.3487e-17	Do not reject H_0	20	90.202	1.6321e-15	Do not reject H_0
6	145.17	5.5487e-21	Do not reject H_0	21	63.142	3.4905e-12	Do not reject H_0
7	176.94	1.3548e-23	Do not reject H_0	22	163.38	1.6122e-22	Do not reject H_0
8	179.26	8.9867e-24	Do not reject H_0	23	128.72	1.7218e-19	Do not reject H_0
9	157.89	4.5607e-22	Do not reject H_0	24	187.29	2.2210e-24	Do not reject H_0
10	86.487	4.3611e-15	Do not reject H_0	25	75.735	8.4427e-14	Do not reject H_0
11	26.132	1.6010e-06	Do not reject H_0	26	127.40	2.2967e-19	Do not reject H_0
12	69.847	4.641e-13	Do not reject H_0	27	54.596	5.2089e-11	Do not reject H_0
13	101.30	9.7113e-17	Do not reject H_0	28	99.968	1.3513e-16	Do not reject H_0
14	87.651	3.1978e-15	Do not reject H_0	29	82.028	1.4576e-14	Do not reject H_0
15	84.586	7.2671e-15	Do not reject H_0	30	100.24	1.2619e-16	Do not reject H_0

Table 10.75: Linearity test of the mating pool size in the range [0.4,0.549].

iiiiiii .mine

Population size in the range [0.4,0.549]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	77.046	5.82e-14	Do not reject H_0	16	36.481	2.84e-08	Do not reject H_0
2	118.35	1.71e-18	Do not reject H_0	17	243.67	3.30e-28	Do not reject H_0
3	122.60	6.60e-19	Do not reject H_0	18	98.8	1.80e-16	Do not reject H_0
4	111.41	8.52e-18	Do not reject H_0	19	85.528	5.63e-15	Do not reject H_0
5	71.627	2.75e-13	Do not reject H_0	20	139.2	1.87e-20	Do not reject H_0
6	72.23	2.31e-13	Do not reject H_0	21	164.55	1.29e-22	Do not reject H_0
7	183.71	4.12e-24	Do not reject H_0	22	132.82	7.15e-20	Do not reject H_0
8	183.78	4.07e-24	Do not reject H_0	23	232.29	1.72e-27	Do not reject H_0
9	80.157	2.44e-14	Do not reject H_0	24	51.792	1.30e-10	Do not reject H_0
10	101.17	1.00e-16	Do not reject H_0	25	70.398	3.94e-13	Do not reject H_0
11	72.43	2.18e-13	Do not reject H_0	26	104.46	4.47e-17	Do not reject H_0
12	84.04	8.42e-15	Do not reject H_0	27	55.108	4.41e-11	Do not reject H_0
13	134.12	5.43e-20	Do not reject H_0	28	39.587	9.06e-09	Do not reject H_0
14	20.859	1.45e-05	Do not reject H_0	29	101.52	9.19e-17	Do not reject H_0
15	36.481	2.84e-08	Do not reject H_0	30	45.883	9.70e-10	Do not reject H_0

Table 10.76: Linearity test of the mating pool size in the range [0.55,0.69].

Population size in the range [0.55,0.69]							
Run	f	p	Conclusion	Run	f	p	Conclusion
1	145.94	4.75e-21	Do not reject H_0	16	48.015	4.66e-10	Do not reject H_0
2	68.43	7.05e-13	Do not reject H_0	17	75.815	8.25e-14	Do not reject H_0
3	25.490	2.08e-06	Do not reject H_0	18	125.56	3.43e-19	Do not reject H_0
4	83.259	1.04e-14	Do not reject H_0	19	33.098	1.02e-07	Do not reject H_0
5	103.05	6.32e-17	Do not reject H_0	20	136.89	3.03e-20	Do not reject H_0
6	36.115	3.25e-08	Do not reject H_0	21	93.978	6.13e-16	Do not reject H_0
7	95.495	4.16e-16	Do not reject H_0	22	61.761	5.34e-12	Do not reject H_0
8	21.425	1.14e-05	Do not reject H_0	23	77.847	4.64e-14	Do not reject H_0
9	114.82	3.85e-18	Do not reject H_0	24	97.939	2.24e-16	Do not reject H_0
10	85.479	5.71e-15	Do not reject H_0	25	126.16	3.00e-19	Do not reject H_0
11	167.27	7.82e-23	Do not reject H_0	26	116.18	2.81e-18	Do not reject H_0
12	102.79	6.73e-17	Do not reject H_0	27	69.628	4.95e-13	Do not reject H_0
13	86.499	4.34e-15	Do not reject H_0	28	151.83	1.47e-21	Do not reject H_0
14	103.29	5.95e-17	Do not reject H_0	29	109.99	1.18e-17	Do not reject H_0
15	48.015	4.66e-10	Do not reject H_0	30	95.291	4.38e-16	Do not reject H_0

Normality

Table 10.77: Kolmogorov-Smirnov test of the mutation rate in the range [0.001,0.1249].

Mutation rate in the range [0.001,0.1249]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.06819	0.72051	Do not reject H_0	16	0.05466	0.91291	Do not reject H_0
2	0.07145	0.66599	Do not reject H_0	17	0.07574	0.59412	Do not reject H_0
3	0.06446	0.78068	Do not reject H_0	18	0.07480	0.60976	Do not reject H_0
4	0.07112	0.67152	Do not reject H_0	19	0.08879	0.39294	Do not reject H_0
5	0.16413	0.00846	Reject H_0	20	0.09800	0.27906	Do not reject H_0
6	0.07511	0.60462	Do not reject H_0	21	0.05287	0.93093	Do not reject H_0
7	0.04039	0.995	Do not reject H_0	22	0.05323	0.9275	Do not reject H_0
8	0.05935	0.85587	Do not reject H_0	23	0.05182	0.94041	Do not reject H_0
9	0.07059	0.68055	Do not reject H_0	24	0.08489	0.4489	Do not reject H_0
10	0.03583	0.99912	Do not reject H_0	25	0.07727	0.56878	Do not reject H_0
11	0.04227	0.9913	Do not reject H_0	26	0.04689	0.97433	Do not reject H_0
12	0.06183	0.82078	Do not reject H_0	27	0.04759	0.97056	Do not reject H_0
13	0.06472	0.77658	Do not reject H_0	28	0.06544	0.76527	Do not reject H_0
14	0.08065	0.5141	Do not reject H_0	29	0.04770	0.96996	Do not reject H_0
15	0.05466	0.91291	Do not reject H_0	30	0.06547	0.7647	Do not reject H_0

Table 10.78: Kolmogorov-Smirnov test of the mutation rate in the range [0.125,0.249].

Mutation rate in the range [0.125,0.249]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.06003	0.84657	Do not reject H_0	16	0.04158	0.99283	Do not reject H_0
2	0.04478	0.98366	Do not reject H_0	17	0.04254	0.99064	Do not reject H_0
3	0.04200	0.99192	Do not reject H_0	18	0.06054	0.83941	Do not reject H_0
4	0.04501	0.98278	Do not reject H_0	19	0.07015	0.68794	Do not reject H_0
5	0.04933	0.95971	Do not reject H_0	20	0.04737	0.97178	Do not reject H_0
6	0.05827	0.87014	Do not reject H_0	21	0.06874	0.71133	Do not reject H_0
7	0.05374	0.92251	Do not reject H_0	22	0.05460	0.91354	Do not reject H_0
8	0.07963	0.53047	Do not reject H_0	23	0.04062	0.99463	Do not reject H_0
9	0.05651	0.89204	Do not reject H_0	24	0.04941	0.95919	Do not reject H_0
10	0.05818	0.87128	Do not reject H_0	25	0.09944	0.26356	Do not reject H_0
11	0.09054	0.36926	Do not reject H_0	26	0.05447	0.91496	Do not reject H_0
12	0.0657	0.76112	Do not reject H_0	27	0.07143	0.66634	Do not reject H_0
13	0.0518	0.94066	Do not reject H_0	28	0.04303	0.98933	Do not reject H_0
14	0.05060	0.95046	Do not reject H_0	29	0.04299	0.98945	Do not reject H_0
15	0.04158	0.99283	Do not reject H_0	30	0.03610	0.999	Do not reject H_0

Table 10.79: Kolmogorov-Smirnov test of the mutation rate in the range [0.25,0.3749].

Mutation rate in the range [0.25,0.3749]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.06034	0.84222	Do not reject H_0	16	0.06180	0.82118	Do not reject H_0
2	0.04265	0.99035	Do not reject H_0	17	0.08206	0.49195	Do not reject H_0
3	0.05305	0.9292	Do not reject H_0	18	0.03631	0.9989	Do not reject H_0
4	0.05776	0.87667	Do not reject H_0	19	0.06365	0.79332	Do not reject H_0
5	0.08003	0.52393	Do not reject H_0	20	0.03997	0.99564	Do not reject H_0
6	0.04450	0.98468	Do not reject H_0	21	0.11844	0.11461	Do not reject H_0
7	0.08829	0.39987	Do not reject H_0	22	0.07674	0.57746	Do not reject H_0
8	0.07165	0.66262	Do not reject H_0	23	0.07660	0.57984	Do not reject H_0
9	0.08438	0.45643	Do not reject H_0	24	0.04755	0.9708	Do not reject H_0
10	0.04446	0.98483	Do not reject H_0	25	0.03981	0.99585	Do not reject H_0
11	0.04867	0.96411	Do not reject H_0	26	0.06684	0.7426	Do not reject H_0
12	0.05186	0.94012	Do not reject H_0	27	0.05262	0.93327	Do not reject H_0
13	0.07759	0.56351	Do not reject H_0	28	0.06284	0.80569	Do not reject H_0
14	0.06985	0.69283	Do not reject H_0	29	0.06792	0.72497	Do not reject H_0
15	0.06180	0.82118	Do not reject H_0	30	0.05649	0.89225	Do not reject H_0

Table 10.80: Kolmogorov-Smirnov test of the mutation rate in the range [0.375,0.49].

Mutation rate in the range [0.375,0.49]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.05404	0.91945	Do not reject H_0	16	0.04484	0.98343	Reject H_0
2	0.05701	0.886	Do not reject H_0	17	0.04210	0.9917	Reject H_0
3	0.07265	0.64589	Do not reject H_0	18	0.1003	0.25470	Do not reject H_0
4	0.06072	0.83677	Do not reject H_0	19	0.04950	0.95857	Reject H_0
5	0.07601	0.58962	Do not reject H_0	20	0.06962	0.69669	Do not reject H_0
6	0.11339	0.14513	Do not reject H_0	21	0.05881	0.8631	Do not reject H_0
7	0.04139	0.99322	Do not reject H_0	22	0.04786	0.969	Do not reject H_0
8	0.05371	0.92275	Do not reject H_0	23	0.05028	0.95293	Do not reject H_0
9	0.07043	0.68309	Do not reject H_0	24	0.10996	0.16928	Do not reject H_0
10	0.03675	0.99867	Do not reject H_0	25	0.05227	0.9365	Do not reject H_0
11	0.05318	0.92803	Do not reject H_0	26	0.05543	0.90452	Do not reject H_0
12	0.09619	0.29937	Do not reject H_0	27	0.07456	0.61388	Do not reject H_0
13	0.04194	0.99207	Do not reject H_0	28	0.08675	0.42166	Do not reject H_0
14	0.06870	0.71196	Do not reject H_0	29	0.05428	0.91694	Do not reject H_0
15	0.04484	0.98343	Do not reject H_0	30	0.06100	0.83275	Do not reject H_0

Table 10.81: Kolmogorov-Smirnov test of the crossover rate in the range [0.6,0.69].

Crossover rate in the range [0.6,0.69]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.05069	0.94973	Do not reject H_0	16	0.12268	0.09324	Do not reject H_0
2	0.06959	0.69723	Do not reject H_0	17	0.06202	0.818	Do not reject H_0
3	0.08026	0.52029	Do not reject H_0	18	0.04075	0.9944	Do not reject H_0
4	0.06015	0.84484	Do not reject H_0	19	0.05310	0.92872	Do not reject H_0
5	0.07583	0.59264	Do not reject H_0	20	0.05806	0.87291	Do not reject H_0
6	0.08288	0.47929	Do not reject H_0	21	0.05265	0.93304	Do not reject H_0
7	0.07783	0.55959	Do not reject H_0	22	0.06458	0.77884	Do not reject H_0
8	0.08633	0.42769	Do not reject H_0	23	0.05628	0.89472	Do not reject H_0
9	0.05422	0.91756	Do not reject H_0	24	0.08037	0.51849	Do not reject H_0
10	0.06563	0.76214	Do not reject H_0	25	0.07034	0.68462	Do not reject H_0
11	0.04362	0.98761	Do not reject H_0	26	0.04819	0.9671	Do not reject H_0
12	0.04852	0.96502	Do not reject H_0	27	0.05470	0.9125	Do not reject H_0
13	0.05966	0.85158	Do not reject H_0	28	0.10378	0.22088	Do not reject H_0
14	0.05655	0.89158	Do not reject H_0	29	0.03830	0.99756	Do not reject H_0
15	0.12268	0.09324	Do not reject H_0	30	0.06221	0.81519	Do not reject H_0

Table 10.82: Kolmogorov-Smirnov test of the crossover rate in the range [0.7,0.79].

Crossover rate in the range [0.7,0.79]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.05730	0.8824	Do not reject H_0	16	0.05206	0.93833	Do not reject H_0
2	0.05992	0.84802	Do not reject H_0	17	0.05575	0.90094	Do not reject H_0
3	0.04940	0.9593	Do not reject H_0	18	0.08470	0.45162	Do not reject H_0
4	0.07571	0.59459	Do not reject H_0	19	0.13325	0.05406	Do not reject H_0
5	0.08539	0.44152	Do not reject H_0	20	0.05642	0.89317	Do not reject H_0
6	0.06465	0.77778	Do not reject H_0	21	0.06891	0.70855	Do not reject H_0
7	0.06153	0.8252	Do not reject H_0	22	0.06210	0.81681	Do not reject H_0
8	0.05659	0.8911	Do not reject H_0	23	0.05845	0.86781	Do not reject H_0
9	0.05751	0.8798	Do not reject H_0	24	0.06260	0.80934	Do not reject H_0
10	0.06866	0.71265	Do not reject H_0	25	0.06903	0.7065	Do not reject H_0
11	0.06812	0.7216	Do not reject H_0	26	0.04800	0.9682	Do not reject H_0
12	0.05907	0.85955	Do not reject H_0	27	0.05439	0.91579	Do not reject H_0
13	0.05573	0.90112	Do not reject H_0	28	0.06770	0.72853	Do not reject H_0
14	0.05396	0.9202	Do not reject H_0	29	0.06132	0.82819	Do not reject H_0
15	0.05206	0.93833	Do not reject H_0	30	0.04210	0.9917	Do not reject H_0

Table 10.83: Kolmogorov-Smirnov test of the crossover rate in the range [0.8,0.89].

Crossover rate in the range [0.8,0.89]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.09576	0.30431	Do not reject H_0	16	0.03469	0.99948	Do not reject H_0
2	0.08176	0.49659	Do not reject H_0	17	0.07618	0.58673	Do not reject H_0
3	0.05223	0.9368	Do not reject H_0	18	0.05774	0.87696	Do not reject H_0
4	0.04414	0.98593	Do not reject H_0	19	0.05799	0.87374	Do not reject H_0
5	0.08429	0.45785	Do not reject H_0	20	0.04119	0.9936	Do not reject H_0
6	0.04815	0.9673	Do not reject H_0	21	0.05242	0.93514	Do not reject H_0
7	0.06094	0.83367	Do not reject H_0	22	0.04412	0.98598	Do not reject H_0
8	0.04710	0.97322	Do not reject H_0	23	0.04722	0.97257	Do not reject H_0
9	0.09225	0.34705	Do not reject H_0	24	0.06919	0.7039	Do not reject H_0
10	0.08007	0.52338	Do not reject H_0	25	0.07708	0.57186	Do not reject H_0
11	0.06337	0.79753	Do not reject H_0	26	0.06383	0.7906	Do not reject H_0
12	0.05889	0.86205	Do not reject H_0	27	0.07528	0.60178	Do not reject H_0
13	0.06994	0.69138	Do not reject H_0	28	0.10224	0.23539	Do not reject H_0
14	0.04225	0.99134	Do not reject H_0	29	0.07997	0.52499	Do not reject H_0
15	0.03469	0.99948	Do not reject H_0	30	0.05208	0.9382	Do not reject H_0

Table 10.84: Kolmogorov-Smirnov test of the crossover rate in the range [0.9,0.99].

Crossover rate in the range [0.9,0.99]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.04397	0.9865	Do not reject H_0	16	0.03440	0.99956	Do not reject H_0
2	0.08160	0.49913	Do not reject H_0	17	0.06001	0.84677	Do not reject H_0
3	0.09953	0.26263	Do not reject H_0	18	0.06070	0.83704	Do not reject H_0
4	0.06757	0.73067	Do not reject H_0	19	0.06780	0.72685	Do not reject H_0
5	0.03335	0.99975	Do not reject H_0	20	0.07079	0.67713	Do not reject H_0
6	0.07316	0.63728	Do not reject H_0	21	0.06531	0.76722	Do not reject H_0
7	0.04879	0.96337	Do not reject H_0	22	0.07474	0.61074	Do not reject H_0
8	0.04891	0.96258	Do not reject H_0	23	0.04197	0.99199	Do not reject H_0
9	0.05973	0.85068	Do not reject H_0	24	0.03552	0.99923	Do not reject H_0
10	0.05440	0.91574	Do not reject H_0	25	0.06603	0.75583	Do not reject H_0
11	0.05703	0.88573	Do not reject H_0	26	0.05861	0.8657	Do not reject H_0
12	0.04130	0.9934	Do not reject H_0	27	0.06319	0.80039	Do not reject H_0
13	0.04967	0.95737	Do not reject H_0	28	0.07285	0.64254	Do not reject H_0
14	0.0633	0.79874	Do not reject H_0	29	0.03970	0.996	Do not reject H_0
15	0.03440	0.99956	Do not reject H_0	30	0.09704	0.28966	Do not reject H_0

Table 10.85: Kolmogorov-Smirnov test of the population size in the range [20,40].

Population size in the range [20,40]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.12180	0.09738	Do not reject H_0	16	0.07002	0.6901	Do not reject H_0
2	0.04482	0.9835	Do not reject H_0	17	0.09535	0.30911	Do not reject H_0
3	0.05161	0.94226	Do not reject H_0	18	0.04774	0.96974	Do not reject H_0
4	0.06423	0.78436	Do not reject H_0	19	0.03851	0.99736	Do not reject H_0
5	0.08858	0.39583	Do not reject H_0	20	0.08427	0.45812	Do not reject H_0
6	0.06224	0.81466	Do not reject H_0	21	0.05920	0.85791	Do not reject H_0
7	0.06549	0.76446	Do not reject H_0	22	0.1136	0.14372	Do not reject H_0
8	0.08852	0.39674	Do not reject H_0	23	0.13860	0.04032	Reject H_0
9	0.08622	0.42926	Do not reject H_0	24	0.05975	0.85033	Do not reject H_0
10	0.12800	0.07129	Do not reject H_0	25	0.06441	0.78154	Do not reject H_0
11	0.08512	0.44540	Do not reject H_0	26	0.05934	0.85599	Do not reject H_0
12	0.08822	0.40091	Do not reject H_0	27	0.11973	0.10774	Do not reject H_0
13	0.05634	0.89403	Do not reject H_0	28	0.08117	0.50583	Do not reject H_0
14	0.05495	0.90978	Do not reject H_0	29	0.07302	0.63966	Do not reject H_0
15	0.07002	0.6901	Do not reject H_0	30	0.06609	0.75483	Do not reject H_0

Table 10.86: Kolmogorov-Smirnov test of the population size in the range [41,60].

Population size in the range [41,60]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.08475	0.451	Do not reject H_0	16	0.04787	0.96894	Do not reject H_0
2	0.07311	0.63814	Do not reject H_0	17	0.09799	0.27916	Do not reject H_0
3	0.05609	0.89696	Do not reject H_0	18	0.09099	0.36341	Do not reject H_0
4	0.06060	0.83851	Do not reject H_0	19	0.07147	0.66576	Do not reject H_0
5	0.06753	0.73142	Do not reject H_0	20	0.05400	0.91981	Do not reject H_0
6	0.04592	0.97897	Do not reject H_0	21	0.04413	0.98597	Do not reject H_0
7	0.07456	0.61389	Do not reject H_0	22	0.10110	0.24662	Do not reject H_0
8	0.07083	0.67642	Do not reject H_0	23	0.05284	0.93127	Do not reject H_0
9	0.05676	0.889	Do not reject H_0	24	0.08224	0.4892	Do not reject H_0
10	0.03841	0.99746	Do not reject H_0	25	0.05557	0.90292	Do not reject H_0
11	0.06837	0.71745	Do not reject H_0	26	0.09341	0.33255	Do not reject H_0
12	0.04576	0.97969	Do not reject H_0	27	0.06289	0.80485	Do not reject H_0
13	0.08131	0.50372	Do not reject H_0	28	0.04974	0.9569	Do not reject H_0
14	0.04633	0.97706	Do not reject H_0	29	0.05814	0.87188	Do not reject H_0
15	0.04787	0.96894	Do not reject H_0	30	0.08128	0.50416	Do not reject H_0

Table 10.87: Kolmogorov-Smirnov test of the population size in the range [61,80].

Population size in the range [61,80]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.05275	0.93204	Do not reject H_0	16	0.05749	0.88012	Do not reject H_0
2	0.09787	0.28040	Do not reject H_0	17	0.04522	0.98194	Do not reject H_0
3	0.05448	0.91489	Do not reject H_0	18	0.07373	0.62774	Do not reject H_0
4	0.06216	0.81587	Do not reject H_0	19	0.07970	0.52924	Do not reject H_0
5	0.04485	0.98337	Do not reject H_0	20	0.08565	0.43767	Do not reject H_0
6	0.04104	0.99388	Do not reject H_0	21	0.08552	0.43953	Do not reject H_0
7	0.08237	0.48711	Do not reject H_0	22	0.06419	0.78496	Do not reject H_0
8	0.06556	0.76328	Do not reject H_0	23	0.06415	0.78558	Do not reject H_0
9	0.06608	0.75491	Do not reject H_0	24	0.06402	0.78756	Do not reject H_0
10	0.04713	0.97307	Do not reject H_0	25	0.0701	0.68878	Do not reject H_0
11	0.06066	0.83765	Do not reject H_0	26	0.09470	0.31682	Do not reject H_0
12	0.07221	0.6533	Do not reject H_0	27	0.04631	0.97715	Do not reject H_0
13	0.05011	0.95415	Do not reject H_0	28	0.04808	0.96771	Do not reject H_0
14	0.05864	0.86536	Do not reject H_0	29	0.06884	0.70975	Do not reject H_0
15	0.05749	0.88012	Do not reject H_0	30	0.03565	0.99918	Do not reject H_0

Table 10.88: Kolmogorov-Smirnov test of the population size in the range [81,100].

Population size in the range [81,100]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.07464	0.6125	Do not reject H_0	16	0.09197	0.35072	Do not reject H_0
2	0.09636	0.29739	Do not reject H_0	17	0.06537	0.76625	Do not reject H_0
3	0.06623	0.75254	Do not reject H_0	18	0.08807	0.40303	Do not reject H_0
4	0.06474	0.77628	Do not reject H_0	19	0.06158	0.82436	Do not reject H_0
5	0.10093	0.24828	Do not reject H_0	20	0.06026	0.84337	Do not reject H_0
6	0.06273	0.80735	Do not reject H_0	21	0.08255	0.48427	Do not reject H_0
7	0.06414	0.78578	Do not reject H_0	22	0.05739	0.88137	Do not reject H_0
8	0.06580	0.7595	Do not reject H_0	23	0.04900	0.96195	Do not reject H_0
9	0.07348	0.63197	Do not reject H_0	24	0.05638	0.89357	Do not reject H_0
10	0.06672	0.74458	Do not reject H_0	25	0.07454	0.61407	Do not reject H_0
11	0.06415	0.78562	Do not reject H_0	26	0.07393	0.62443	Do not reject H_0
12	0.05850	0.86717	Do not reject H_0	27	0.05857	0.8662	Do not reject H_0
13	0.03336	0.99975	Do not reject H_0	28	0.06953	0.6982	Do not reject H_0
14	0.05783	0.87581	Do not reject H_0	29	0.05594	0.89872	Do not reject H_0
15	0.09197	0.35072	Do not reject H_0	30	0.04458	0.9844	Do not reject H_0

Table 10.89: Kolmogorov-Smirnov test of the mating pool size in the range [0.1,0.249].

Mating pool size in the range [0.1,0.249]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.06872	0.71176	Do not reject H_0	16	0.04698	0.97383	Do not reject H_0
2	0.04108	0.99382	Do not reject H_0	17	0.06581	0.75932	Do not reject H_0
3	0.04055	0.99475	Do not reject H_0	18	0.05250	0.93439	Do not reject H_0
4	0.05858	0.86617	Do not reject H_0	19	0.03819	0.99765	Do not reject H_0
5	0.06282	0.80601	Do not reject H_0	20	0.06123	0.82944	Do not reject H_0
6	0.03512	0.99936	Do not reject H_0	21	0.06937	0.70094	Do not reject H_0
7	0.04946	0.95884	Do not reject H_0	22	0.04777	0.96953	Do not reject H_0
8	0.06408	0.7866	Do not reject H_0	23	0.08180	0.49595	Do not reject H_0
9	0.07616	0.58718	Do not reject H_0	24	0.08679	0.42105	Do not reject H_0
10	0.08474	0.45115	Do not reject H_0	25	0.07580	0.59304	Do not reject H_0
11	0.05662	0.89076	Do not reject H_0	26	0.04533	0.98148	Do not reject H_0
12	0.04282	0.9899	Do not reject H_0	27	0.06378	0.79131	Do not reject H_0
13	0.05105	0.94693	Do not reject H_0	28	0.06995	0.69125	Do not reject H_0
14	0.08663	0.4234	Do not reject H_0	29	0.06972	0.69512	Do not reject H_0
15	0.04698	0.97383	Do not reject H_0	30	0.05505	0.90877	Do not reject H_0

Table 10.90: Kolmogorov-Smirnov test of the mating pool size in the range [0.25,0.39].

Mating pool size in the range [0.25,0.39]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.05167	0.9417	Do not reject H_0	16	0.09524	0.31047	Do not reject H_0
2	0.0763	0.58488	Do not reject H_0	17	0.04651	0.97623	Do not reject H_0
3	0.04719	0.97274	Do not reject H_0	18	0.06618	0.75337	Do not reject H_0
4	0.03951	0.99625	Do not reject H_0	19	0.04621	0.97764	Do not reject H_0
5	0.06689	0.74185	Do not reject H_0	20	0.06649	0.74825	Do not reject H_0
6	0.05700	0.88608	Do not reject H_0	21	0.05119	0.94574	Do not reject H_0
7	0.07438	0.61684	Do not reject H_0	22	0.07588	0.59183	Do not reject H_0
8	0.05170	0.94147	Do not reject H_0	23	0.05473	0.9122	Do not reject H_0
9	0.04179	0.99239	Do not reject H_0	24	0.05055	0.95087	Do not reject H_0
10	0.05781	0.87609	Do not reject H_0	25	0.06899	0.70725	Do not reject H_0
11	0.06731	0.73497	Do not reject H_0	26	0.04958	0.958	Do not reject H_0
12	0.07998	0.52474	Do not reject H_0	27	0.07219	0.65362	Do not reject H_0
13	0.05730	0.88245	Do not reject H_0	28	0.04440	0.98504	Do not reject H_0
14	0.08458	0.45345	Do not reject H_0	29	0.06604	0.75556	Do not reject H_0
15	0.09524	0.31047	Do not reject H_0	30	0.06649	0.74836	Do not reject H_0

Table 10.91: Kolmogorov-Smirnov test of the mating pool size in the range [0.4,0.549].

Mating pool size in the range [0.4,0.549]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.07439	0.61658	Do not reject H_0	16	0.07667	0.57876	Do not reject H_0
2	0.07788	0.55886	Do not reject H_0	17	0.04778	0.96947	Do not reject H_0
3	0.11871	0.11312	Do not reject H_0	18	0.04240	0.99098	Do not reject H_0
4	0.07710	0.57165	Do not reject H_0	19	0.04105	0.99388	Do not reject H_0
5	0.04603	0.97845	Do not reject H_0	20	0.05191	0.9397	Do not reject H_0
6	0.07456	0.61389	Do not reject H_0	21	0.06671	0.74472	Do not reject H_0
7	0.04010	0.99544	Do not reject H_0	22	0.03670	0.9987	Do not reject H_0
8	0.07910	0.53894	Do not reject H_0	23	0.05398	0.92007	Do not reject H_0
9	0.06623	0.75255	Do not reject H_0	24	0.05569	0.9016	Do not reject H_0
10	0.04441	0.985	Do not reject H_0	25	0.04724	0.97247	Do not reject H_0
11	0.05877	0.86361	Do not reject H_0	26	0.07744	0.56598	Do not reject H_0
12	0.04705	0.97347	Do not reject H_0	27	0.07764	0.5627	Do not reject H_0
13	0.06787	0.72568	Do not reject H_0	28	0.05011	0.95419	Do not reject H_0
14	0.05996	0.8475	Do not reject H_0	29	0.04336	0.98838	Do not reject H_0
15	0.07667	0.57876	Do not reject H_0	30	0.05655	0.89161	Do not reject H_0

Table 10.92: Kolmogorov-Smirnov test of the mating pool size in the range [0.55,0.69].

Mating pool size in the range [0.55,0.69]							
Run	d	p	Conclusion	Run	d	p	Conclusion
1	0.08352	0.46953	Do not reject H_0	16	0.09448	0.31944	Do not reject H_0
2	0.03891	0.99695	Do not reject H_0	17	0.04381	0.98701	Do not reject H_0
3	0.07594	0.59075	Do not reject H_0	18	0.06232	0.81354	Do not reject H_0
4	0.05565	0.90205	Do not reject H_0	19	0.05354	0.92452	Do not reject H_0
5	0.05316	0.9282	Do not reject H_0	20	0.06734	0.73448	Do not reject H_0
6	0.06586	0.75848	Do not reject H_0	21	0.09394	0.32608	Do not reject H_0
7	0.05786	0.87543	Do not reject H_0	22	0.11881	0.11261	Do not reject H_0
8	0.10013	0.25648	Do not reject H_0	23	0.11974	0.10768	Do not reject H_0
9	0.05422	0.9176	Do not reject H_0	24	0.05219	0.9372	Do not reject H_0
10	0.06317	0.80073	Do not reject H_0	25	0.06602	0.75584	Do not reject H_0
11	0.05394	0.92045	Do not reject H_0	26	0.04719	0.97276	Do not reject H_0
12	0.06376	0.79159	Do not reject H_0	27	0.04259	0.99052	Do not reject H_0
13	0.08249	0.48533	Do not reject H_0	28	0.04335	0.98841	Do not reject H_0
14	0.07404	0.62251	Do not reject H_0	29	0.08106	0.50759	Do not reject H_0
15	0.09448	0.31944	Do not reject H_0	30	0.08118	0.50571	Do not reject H_0

Independence

Table 10.93: Durbin-Watson test of the mutation rate in the range [0.001,0.1249].

Mutation rate in the range [0.001,0.1249]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.8434	0.81218	Do not reject H_0	16	1.7454	0.91608	Do not reject H_0
2	1.8271	0.8334	Do not reject H_0	17	2.0529	0.43836	Do not reject H_0
3	1.4804	0.99695	Do not reject H_0	18	1.8635	0.78383	Do not reject H_0
4	2.2241	0.15434	Do not reject H_0	19	1.883	0.7542	Do not reject H_0
5	2.0962	0.35387	Do not reject H_0	20	1.6935	0.9497	Do not reject H_0
6	1.5782	0.98722	Do not reject H_0	21	1.8902	0.74272	Do not reject H_0
7	1.7706	0.89473	Do not reject H_0	22	1.8248	0.83638	Do not reject H_0
8	1.9379	0.66028	Do not reject H_0	23	1.7981	0.86727	Do not reject H_0
9	1.8602	0.78879	Do not reject H_0	24	1.8098	0.8542	Do not reject H_0
10	2.0292	0.48304	Do not reject H_0	25	1.8804	0.75824	Do not reject H_0
11	1.5559	0.99057	Do not reject H_0	26	1.5159	0.9947	Do not reject H_0
12	2.0542	0.43582	Do not reject H_0	27	1.6351	0.9738	Do not reject H_0
13	1.6376	0.97303	Do not reject H_0	28	1.4285	0.99872	Do not reject H_0
14	1.3442	0.99974	Do not reject H_0	29	1.8322	0.82704	Do not reject H_0
15	1.7454	0.91608	Do not reject H_0	30	2.39	0.03137	Reject H_0

Table 10.94: Durbin-Watson test of the mutation rate in the range [0.125,0.249].

Mutation rate in the range [0.125,0.249]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.5298	0.9935	Do not reject H_0	16	2.3427	0.05272	Do not reject H_0
2	2.024	0.49341	Do not reject H_0	17	1.8461	0.8086	Do not reject H_0
3	1.2541	0.99996	Do not reject H_0	18	1.8894	0.74409	Do not reject H_0
4	1.7768	0.88889	Do not reject H_0	19	1.7502	0.91227	Do not reject H_0
5	1.9435	0.65008	Do not reject H_0	20	1.8487	0.80506	Do not reject H_0
6	1.8119	0.85187	Do not reject H_0	21	1.8788	0.76085	Do not reject H_0
7	1.4693	0.99745	Do not reject H_0	22	1.6184	0.9786	Do not reject H_0
8	1.4467	0.99825	Do not reject H_0	23	1.6510	0.96846	Do not reject H_0
9	1.9292	0.67603	Do not reject H_0	24	1.8095	0.85463	Do not reject H_0
10	2.0074	0.52647	Do not reject H_0	25	2.2596	0.11564	Do not reject H_0
11	1.7409	0.91953	Do not reject H_0	26	1.7791	0.88669	Do not reject H_0
12	1.6675	0.96203	Do not reject H_0	27	1.4382	0.99849	Do not reject H_0
13	1.7379	0.92181	Do not reject H_0	28	1.5281	0.99366	Do not reject H_0
14	1.5915	0.98477	Do not reject H_0	29	1.9528	0.63263	Do not reject H_0
15	2.3427	0.05272	Do not reject H_0	30	1.6799	0.9565	Do not reject H_0

Table 10.95: Durbin-Watson test of the mutation rate in the range [0.25,0.3749].

Mutation rate in the range [0.25,0.3749]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.7673	0.8977	Do not reject H_0	16	1.5720	0.98823	Do not reject H_0
2	1.7726	0.89286	Do not reject H_0	17	1.8961	0.73313	Do not reject H_0
3	2.0465	0.45129	Do not reject H_0	18	2.0609	0.42236	Do not reject H_0
4	1.9458	0.64577	Do not reject H_0	19	1.805	0.8597	Do not reject H_0
5	1.4063	0.99914	Do not reject H_0	20	2.1643	0.23668	Do not reject H_0
6	2.3772	0.03628	Reject H_0	21	1.6260	0.97653	Do not reject H_0
7	2.0927	0.36055	Do not reject H_0	22	1.5757	0.98764	Do not reject H_0
8	2.0042	0.53279	Do not reject H_0	23	2.0191	0.50317	Do not reject H_0
9	1.3751	0.99952	Do not reject H_0	24	1.8652	0.7814	Do not reject H_0
10	2.0966	0.35323	Do not reject H_0	25	1.7852	0.8807	Do not reject H_0
11	1.7327	0.92553	Do not reject H_0	26	2.2184	0.1612	Do not reject H_0
12	2.0029	0.53542	Do not reject H_0	27	2.0388	0.46709	Do not reject H_0
13	1.5965	0.98375	Do not reject H_0	28	2.0158	0.50977	Do not reject H_0
14	1.7500	0.91242	Do not reject H_0	29	2.0519	0.44045	Do not reject H_0
15	1.5720	0.98823	Do not reject H_0	30	1.8646	0.78235	Do not reject H_0

Table 10.96: Durbin-Watson test of the mutation rate in the range [0.375,0.49].

Mutation rate in the range [0.375,0.49]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.8866	0.74844	Do not reject H_0	16	2.1448	0.26788	Do not reject H_0
2	2.0386	0.46745	Do not reject H_0	17	1.8496	0.80377	Do not reject H_0
3	2.2367	0.13971	Do not reject H_0	18	2.2146	0.16601	Do not reject H_0
4	1.7077	0.9418	Do not reject H_0	19	1.7832	0.88272	Do not reject H_0
5	2.0295	0.48242	Do not reject H_0	20	1.9688	0.60225	Do not reject H_0
6	1.7136	0.93827	Do not reject H_0	21	1.8179	0.84468	Do not reject H_0
7	2.2344	0.14231	Do not reject H_0	22	1.9067	0.71534	Do not reject H_0
8	1.9329	0.66949	Do not reject H_0	23	1.8672	0.77849	Do not reject H_0
9	1.9571	0.62451	Do not reject H_0	24	2.0330	0.47904	Do not reject H_0
10	2.1339	0.28628	Do not reject H_0	25	2.4984	0.00769	Reject H_0
11	1.7645	0.90024	Do not reject H_0	26	2.0463	0.45163	Do not reject H_0
12	2.1919	0.19598	Do not reject H_0	27	1.8264	0.83428	Do not reject H_0
13	2.0612	0.42172	Do not reject H_0	28	2.2262	0.15189	Do not reject H_0
14	1.6680	0.96181	Do not reject H_0	29	1.9537	0.63103	Do not reject H_0
15	2.1448	0.26788	Do not reject H_0	30	1.9922	0.55653	Do not reject H_0

Table 10.97: Durbin-Watson test of the crossover rate in the range [0.6,0.69].

Crossover rate in the range [0.6,0.69]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.3671	0.04063	Reject H_0	16	1.8857	0.75002	Do not reject H_0
2	2.2618	0.11351	Do not reject H_0	17	1.8046	0.86015	Do not reject H_0
3	2.1443	0.26864	Do not reject H_0	18	1.5990	0.98322	Do not reject H_0
4	1.9421	0.65273	Do not reject H_0	19	1.8016	0.8635	Do not reject H_0
5	1.6917	0.95064	Do not reject H_0	20	1.8385	0.81883	Do not reject H_0
6	1.4991	0.9959	Do not reject H_0	21	1.8459	0.8089	Do not reject H_0
7	1.8436	0.8119	Do not reject H_0	22	1.9334	0.66846	Do not reject H_0
8	2.0829	0.37925	Do not reject H_0	23	1.6097	0.9808	Do not reject H_0
9	1.8562	0.79451	Do not reject H_0	24	1.9252	0.68334	Do not reject H_0
10	1.6644	0.96332	Do not reject H_0	25	2.1603	0.24284	Do not reject H_0
11	1.7464	0.91532	Do not reject H_0	26	2.3334	0.05801	Do not reject H_0
12	1.9312	0.67257	Do not reject H_0	27	1.7356	0.92345	Do not reject H_0
13	2.0728	0.39874	Do not reject H_0	28	2.5048	0.00701	Reject H_0
14	1.9797	0.5812	Do not reject H_0	29	1.8129	0.8507	Do not reject H_0
15	1.8857	0.75002	Do not reject H_0	30	1.9808	0.57895	Do not reject H_0

Table 10.98: Durbin-Watson test of the crossover rate in the range [0.7,0.79].

Crossover rate in the range [0.7,0.79]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.1185	0.31302	Do not reject H_0	16	1.9150	0.70113	Do not reject H_0
2	2.1243	0.30285	Do not reject H_0	17	1.5503	0.99128	Do not reject H_0
3	1.8135	0.8499	Do not reject H_0	18	2.3613	0.04325	Reject H_0
4	1.5356	0.99293	Do not reject H_0	19	2.0455	0.45324	Do not reject H_0
5	1.7537	0.90942	Do not reject H_0	20	1.9961	0.54882	Do not reject H_0
6	1.7657	0.89916	Do not reject H_0	21	1.7247	0.93106	Do not reject H_0
7	1.9475	0.6427	Do not reject H_0	22	1.8754	0.766	Do not reject H_0
8	1.8830	0.75418	Do not reject H_0	23	1.8648	0.78204	Do not reject H_0
9	2.1567	0.24855	Do not reject H_0	24	2.0019	0.53734	Do not reject H_0
10	1.7238	0.9317	Do not reject H_0	25	1.9106	0.7088	Do not reject H_0
11	1.8924	0.73923	Do not reject H_0	26	1.9576	0.62365	Do not reject H_0
12	2.0654	0.41343	Do not reject H_0	27	1.7805	0.88537	Do not reject H_0
13	1.9430	0.65103	Do not reject H_0	28	2.0465	0.45133	Do not reject H_0
14	1.5188	0.99447	Do not reject H_0	29	1.9101	0.70965	Do not reject H_0
15	1.9150	0.70113	Do not reject H_0	30	2.1079	0.33215	Do not reject H_0

Table 10.99: Durbin-Watson test of the crossover rate in the range [0.8,0.89].

Crossover rate in the range [0.8,0.89]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.744	0.91717	Do not reject H_0	16	1.8428	0.81298	Do not reject H_0
2	1.7931	0.87265	Do not reject H_0	17	1.6992	0.94664	Do not reject H_0
3	1.7157	0.93692	Do not reject H_0	18	2.0926	0.3607	Do not reject H_0
4	1.8288	0.83137	Do not reject H_0	19	2.1399	0.27606	Do not reject H_0
5	1.7117	0.93944	Do not reject H_0	20	1.7589	0.90508	Do not reject H_0
6	1.8512	0.8015	Do not reject H_0	21	1.5988	0.98326	Do not reject H_0
7	1.4891	0.9965	Do not reject H_0	22	1.8855	0.75022	Do not reject H_0
8	2.0330	0.47898	Do not reject H_0	23	1.8105	0.85339	Do not reject H_0
9	1.7715	0.8939	Do not reject H_0	24	2.0313	0.47895	Do not reject H_0
10	2.4614	0.01288	Do not reject H_0	25	2.2302	0.14716	Do not reject H_0
11	2.2973	0.08257	Do not reject H_0	26	1.6250	0.97681	Do not reject H_0
12	1.9001	0.72656	Do not reject H_0	27	1.9246	0.6844	Do not reject H_0
13	1.4533	0.99805	Do not reject H_0	28	2.0435	0.4575	Do not reject H_0
14	1.9941	0.55284	Do not reject H_0	29	2.4223	0.02131	Reject H_0
15	1.8428	0.81298	Do not reject H_0	30	1.6988	0.94688	Do not reject H_0

Table 10.100: Durbin-Watson test of the crossover rate in the range [0.9,0.99].

Crossover rate in the range [0.9,0.99]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.9700	0.59997	Do not reject H_0	16	1.9115	0.70716	Do not reject H_0
2	1.7083	0.94143	Do not reject H_0	17	2.2387	0.13753	Do not reject H_0
3	1.7581	0.90577	Do not reject H_0	18	1.7484	0.91373	Do not reject H_0
4	2.1214	0.30791	Do not reject H_0	19	2.0474	0.44945	Do not reject H_0
5	1.7475	0.91444	Do not reject H_0	20	1.4939	0.99622	Do not reject H_0
6	2.0643	0.41554	Do not reject H_0	21	1.8862	0.74907	Do not reject H_0
7	1.4563	0.99795	Do not reject H_0	22	1.9874	0.56607	Do not reject H_0
8	2.2177	0.16218	Do not reject H_0	23	1.9852	0.57038	Do not reject H_0
9	2.1445	0.26844	Do not reject H_0	24	1.9959	0.54933	Do not reject H_0
10	1.5818	0.98659	Do not reject H_0	25	2.1496	0.25995	Do not reject H_0
11	2.2173	0.16268	Do not reject H_0	26	1.8729	0.7698	Do not reject H_0
12	1.7574	0.90636	Do not reject H_0	27	2.0149	0.51157	Do not reject H_0
13	2.1059	0.33589	Do not reject H_0	28	1.9331	0.66906	Do not reject H_0
14	1.7815	0.88437	Do not reject H_0	29	1.7025	0.9448	Do not reject H_0
15	1.9115	0.70716	Do not reject H_0	30	2.017	0.50742	Do not reject H_0

Table 10.101: Durbin-Watson test of the population size in the range [20,40].

Population size in the range [20,40]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	0.85847	1	Do not reject H_0	16	1.8779	0.76215	Do not reject H_0
2	1.1252	1	Do not reject H_0	17	1.1544	1	Do not reject H_0
3	1.1438	1	Do not reject H_0	18	1.7977	0.86769	Do not reject H_0
4	1.2074	0.99999	Do not reject H_0	19	1.6969	0.9479	Do not reject H_0
5	1.6424	0.97146	Do not reject H_0	20	0.87244	1	Do not reject H_0
6	0.95813	1	Do not reject H_0	21	1.3376	0.99977	Do not reject H_0
7	1.2443	0.99997	Do not reject H_0	22	1.0582	1	Do not reject H_0
8	1.0801	1	Do not reject H_0	23	0.90157	1	Do not reject H_0
9	1.3039	0.99989	Do not reject H_0	24	1.5342	0.99307	Do not reject H_0
10	0.79873	1	Do not reject H_0	25	0.68786	1	Do not reject H_0
11	1.2829	0.99993	Do not reject H_0	26	0.9646	1	Do not reject H_0
12	1.7389	0.92101	Do not reject H_0	27	1.366	0.9996	Do not reject H_0
13	1.2475	0.99997	Do not reject H_0	28	1.3291	0.9998	Do not reject H_0
14	1.0224	1	Do not reject H_0	29	1.3384	0.99977	Do not reject H_0
15	1.8779	0.76215	Do not reject H_0	30	1.5493	0.9914	Do not reject H_0

Table 10.102: Durbin-Watson test of the population size in the range [41,60].

Population size in the range [41,60]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.6344	0.97404	Do not reject H_0	16	1.4780	0.99707	Do not reject H_0
2	1.9781	0.58421	Do not reject H_0	17	1.3563	0.99967	Do not reject H_0
3	1.7421	0.91866	Do not reject H_0	18	1.4475	0.99823	Do not reject H_0
4	1.9717	0.59661	Do not reject H_0	19	1.4529	0.99806	Do not reject H_0
5	1.2482	0.99997	Do not reject H_0	20	1.7406	0.91974	Do not reject H_0
6	1.5608	0.9899	Do not reject H_0	21	1.5328	0.9932	Do not reject H_0
7	1.3789	0.99949	Do not reject H_0	22	1.6751	0.95871	Do not reject H_0
8	1.5378	0.9927	Do not reject H_0	23	1.9228	0.68752	Do not reject H_0
9	1.6135	0.97987	Do not reject H_0	24	1.4908	0.9964	Do not reject H_0
10	1.7561	0.90742	Do not reject H_0	25	1.8309	0.8287	Do not reject H_0
11	1.4324	0.99863	Do not reject H_0	26	1.8488	0.80484	Do not reject H_0
12	2.1001	0.34666	Do not reject H_0	27	1.0425	1	Do not reject H_0
13	2.0243	0.49281	Do not reject H_0	28	1.5501	0.9913	Do not reject H_0
14	1.1289	1	Do not reject H_0	29	1.5320	0.99328	Do not reject H_0
15	1.4780	0.99707	Do not reject H_0	30	0.99795	1	Do not reject H_0

Table 10.103: Durbin-Watson test of the population size in the range [61,80].

Population size in the range [61,80]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.7491	0.91315	Do not reject H_0	16	1.6075	0.9813	Do not reject H_0
2	1.9816	0.5774	Do not reject H_0	17	1.9200	0.69253	Do not reject H_0
3	1.5871	0.98561	Do not reject H_0	18	1.6167	0.97906	Do not reject H_0
4	1.7651	0.89969	Do not reject H_0	19	1.5029	0.99566	Do not reject H_0
5	2.2474	0.12809	Do not reject H_0	20	1.7675	0.89752	Do not reject H_0
6	1.0225	1	Do not reject H_0	21	1.7781	0.88768	Do not reject H_0
7	1.3709	0.99956	Do not reject H_0	22	2.0097	0.52198	Do not reject H_0
8	2.1183	0.31344	Do not reject H_0	23	1.8574	0.7928	Do not reject H_0
9	1.9758	0.5887	Do not reject H_0	24	1.9234	0.6865	Do not reject H_0
10	1.5924	0.9846	Do not reject H_0	25	1.7226	0.93252	Do not reject H_0
11	1.5084	0.99528	Do not reject H_0	26	1.9927	0.55556	Do not reject H_0
12	1.8653	0.78126	Do not reject H_0	27	1.7116	0.93947	Do not reject H_0
13	2.0959	0.35451	Do not reject H_0	28	1.5832	0.98633	Do not reject H_0
14	1.7694	0.89584	Do not reject H_0	29	1.7937	0.87192	Do not reject H_0
15	1.6075	0.9813	Do not reject H_0	30	1.6745	0.95897	Do not reject H_0

Table 10.104: Durbin-Watson test of the population size in the range [81,100].

Population size in the range [81,100]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.4040	0.99918	Do not reject H_0	16	1.6143	0.97966	Do not reject H_0
2	1.3409	0.99976	Do not reject H_0	17	1.2464	0.99997	Do not reject H_0
3	1.3725	0.99955	Do not reject H_0	18	1.7035	0.94422	Do not reject H_0
4	1.3586	0.99965	Do not reject H_0	19	1.7344	0.92433	Do not reject H_0
5	2.1047	0.33809	Do not reject H_0	20	1.1552	1	Do not reject H_0
6	1.1632	1	Do not reject H_0	21	1.3824	0.99945	Do not reject H_0
7	1.6843	0.9544	Do not reject H_0	22	1.5732	0.98805	Do not reject H_0
8	1.9289	0.67657	Do not reject H_0	23	1.6289	0.9757	Do not reject H_0
9	1.3706	0.99956	Do not reject H_0	24	1.5274	0.99372	Do not reject H_0
10	1.3649	0.9996	Do not reject H_0	25	1.5016	0.99574	Do not reject H_0
11	1.8094	0.85465	Do not reject H_0	26	1.3078	0.99988	Do not reject H_0
12	2.0996	0.34747	Do not reject H_0	27	1.1079	1	Do not reject H_0
13	1.3401	0.99976	Do not reject H_0	28	1.2737	0.99994	Do not reject H_0
14	1.1019	1	Do not reject H_0	29	1.4984	0.99595	Do not reject H_0
15	1.6143	0.97966	Do not reject H_0	30	1.2362	0.99998	Do not reject H_0

Table 10.105: Durbin-Watson test of the mating pool size in the range [0.1,0.249].

Mating pool size in the range [0.1,0.249]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.3818	0.99946	Do not reject H_0	16	2.4908	0.00858	Reject H_0
2	1.7506	0.91194	Do not reject H_0	17	1.847	0.80734	Do not reject H_0
3	1.8652	0.78134	Do not reject H_0	18	1.7571	0.90664	Do not reject H_0
4	2.0963	0.35364	Do not reject H_0	19	1.7559	0.90759	Do not reject H_0
5	1.7908	0.87503	Do not reject H_0	20	1.6900	0.95155	Do not reject H_0
6	2.0486	0.447	Do not reject H_0	21	1.6787	0.95704	Do not reject H_0
7	1.6654	0.96288	Do not reject H_0	22	1.7455	0.91602	Do not reject H_0
8	2.0952	0.35579	Do not reject H_0	23	2.3767	0.03651	Reject H_0
9	2.0859	0.37345	Do not reject H_0	24	1.8738	0.76853	Do not reject H_0
10	2.1856	0.20493	Do not reject H_0	25	2.0791	0.38647	Do not reject H_0
11	2.1343	0.28558	Do not reject H_0	26	2.0835	0.37802	Do not reject H_0
12	1.7385	0.92134	Do not reject H_0	27	2.0972	0.35201	Do not reject H_0
13	1.7546	0.90874	Do not reject H_0	28	1.8787	0.76099	Do not reject H_0
14	2.2069	0.17588	Do not reject H_0	29	2.0029	0.53541	Do not reject H_0
15	2.4908	0.00858	Reject H_0	30	2.2356	0.14093	Do not reject H_0

Table 10.106: Durbin-Watson test of the mating pool size in the range [0.25,0.39].

Mating pool size in the range [0.25,0.39]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.4057	0.99915	Do not reject H_0	16	2.0579	0.42825	Do not reject H_0
2	1.9768	0.5867	Do not reject H_0	17	1.8516	0.80096	Do not reject H_0
3	1.6182	0.97865	Do not reject H_0	18	1.7223	0.93266	Do not reject H_0
4	2.1292	0.29426	Do not reject H_0	19	1.8350	0.82341	Do not reject H_0
5	1.9294	0.67577	Do not reject H_0	20	2.0182	0.50491	Do not reject H_0
6	2.1325	0.28861	Do not reject H_0	21	1.8054	0.85924	Do not reject H_0
7	2.1227	0.30563	Do not reject H_0	22	1.8763	0.76466	Do not reject H_0
8	2.1568	0.24832	Do not reject H_0	23	1.6862	0.95346	Do not reject H_0
9	1.8425	0.81344	Do not reject H_0	24	1.7005	0.94593	Do not reject H_0
10	1.9534	0.63149	Do not reject H_0	25	1.9052	0.71794	Do not reject H_0
11	2.0471	0.45016	Do not reject H_0	26	1.9398	0.65689	Do not reject H_0
12	1.8234	0.83807	Do not reject H_0	27	1.6316	0.97488	Do not reject H_0
13	1.687	0.95305	Do not reject H_0	28	1.9468	0.644	Do not reject H_0
14	2.3027	0.07848	Do not reject H_0	29	1.4701	0.99742	Do not reject H_0
15	2.0579	0.42825	Do not reject H_0	30	2.4752	0.01067	Reject H_0

Table 10.107: Durbin-Watson test of the mating pool size in the range [0.4,0.549].

Mating pool size in the range [0.4,0.549]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	2.266	0.10944	Do not reject H_0	16	1.4987	0.99593	Do not reject H_0
2	1.9997	0.54181	Do not reject H_0	17	1.5913	0.9848	Do not reject H_0
3	1.9354	0.66495	Do not reject H_0	18	2.0135	0.51445	Do not reject H_0
4	1.8797	0.75943	Do not reject H_0	19	2.0317	0.48168	Do not reject H_0
5	2.0321	0.48098	Do not reject H_0	20	1.8626	0.78522	Do not reject H_0
6	1.9521	0.63394	Do not reject H_0	21	1.7605	0.90374	Do not reject H_0
7	1.9337	0.66799	Do not reject H_0	22	1.9105	0.70895	Do not reject H_0
8	1.6688	0.96147	Do not reject H_0	23	1.8156	0.84751	Do not reject H_0
9	1.9431	0.65083	Do not reject H_0	24	1.9576	0.62365	Do not reject H_0
10	2.1315	0.29024	Do not reject H_0	25	1.7806	0.88526	Do not reject H_0
11	1.8053	0.85937	Do not reject H_0	26	1.9785	0.58352	Do not reject H_0
12	2.1925	0.19517	Do not reject H_0	27	1.6742	0.95912	Do not reject H_0
13	1.9324	0.67042	Do not reject H_0	28	2.0457	0.45287	Do not reject H_0
14	2.2627	0.1126	Do not reject H_0	29	1.9637	0.61211	Do not reject H_0
15	1.4987	0.99593	Do not reject H_0	30	2.2715	0.10433	Do not reject H_0

Table 10.108: Durbin-Watson test of the mating pool size in the range [0.55,0.69].

Mating pool size in the range [0.55,0.69]							
Run	dw	p	Conclusion	Run	dw	p	Conclusion
1	1.797	0.86848	Do not reject H_0	16	1.4578	0.9979	Do not reject H_0
2	2.2495	0.12590	Do not reject H_0	17	1.5671	0.98899	Do not reject H_0
3	1.7901	0.87575	Do not reject H_0	18	1.8211	0.84088	Do not reject H_0
4	2.0588	0.4265	Do not reject H_0	19	2.0743	0.39586	Do not reject H_0
5	1.6265	0.97638	Do not reject H_0	20	1.9839	0.57284	Do not reject H_0
6	2.011	0.51937	Do not reject H_0	21	1.9459	0.64569	Do not reject H_0
7	1.7639	0.9008	Do not reject H_0	22	2.1061	0.33548	Do not reject H_0
8	2.0227	0.49605	Do not reject H_0	23	1.7489	0.91337	Do not reject H_0
9	1.8874	0.74725	Do not reject H_0	24	2.1035	0.3404	Do not reject H_0
10	2.5455	0.00378	Reject H_0	25	1.4097	0.99909	Do not reject H_0
11	2.0465	0.45119	Do not reject H_0	26	1.5745	0.98783	Do not reject H_0
12	1.7187	0.935	Do not reject H_0	27	1.9567	0.62538	Do not reject H_0
13	2.3088	0.07403	Do not reject H_0	28	2.1062	0.33525	Do not reject H_0
14	1.9722	0.59569	Do not reject H_0	29	1.6541	0.96733	Do not reject H_0
15	1.4578	0.9979	Do not reject H_0	30	2.0621	0.41985	Do not reject H_0

Homoscedasticity

Table 10.109: Breusch-Pagan test of the mutation rate in the range [0.001,0.1249].

Mutation rate in the range [0.001,0.1249]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.33095	0.5651	Do not reject H_0	16	0.17138	0.67889	Do not reject H_0
2	2.9345	0.08670	Do not reject H_0	17	0.02282	0.8799	Do not reject H_0
3	12.562	0.00039	Reject H_0	18	1.9424	0.16340	Do not reject H_0
4	0.551	0.45791	Do not reject H_0	19	2.0232	0.15491	Do not reject H_0
5	1.0080	0.31539	Do not reject H_0	20	2.0443	0.15278	Do not reject H_0
6	0.07447	0.78493	Do not reject H_0	21	2.1561	0.14201	Do not reject H_0
7	0.61306	0.43364	Do not reject H_0	22	4.8403	0.02780	Reject H_0
8	0.27768	0.59823	Do not reject H_0	23	1.9151	0.1664	Do not reject H_0
9	0.32785	0.56693	Do not reject H_0	24	16.374	5.1e-05	Reject H_0
10	2.1687	0.14084	Do not reject H_0	25	3.0306	0.08170	Do not reject H_0
11	0.18596	0.6663	Do not reject H_0	26	0.02356	0.878	Do not reject H_0
12	3.9921	0.05571	Do not reject H_0	27	1.0966	0.29502	Do not reject H_0
13	0.21440	0.64334	Do not reject H_0	28	1.1531	0.2829	Do not reject H_0
14	4.0691	0.05116	Do not reject H_0	29	0.38433	0.5353	Do not reject H_0
15	0.17138	0.67889	Do not reject H_0	30	0.10605	0.74468	Do not reject H_0

Table 10.110: Breusch-Pagan test of the mutation rate in the range [0.125,0.249].

Mutation rate in the range [0.125,0.249]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	3.4148	0.06461	Do not reject H_0	16	1.9827	0.15910	Do not reject H_0
2	1.7e-05	0.9967	Do not reject H_0	17	0.46257	0.50642	Do not reject H_0
3	4.7201	0.02981	Reject H_0	18	5.0856	0.02412	Reject H_0
4	2.3077	0.12873	Do not reject H_0	19	0.17848	0.67268	Do not reject H_0
5	0.86845	0.35138	Do not reject H_0	20	0.53763	0.50342	Do not reject H_0
6	1.6407	0.20022	Do not reject H_0	21	3.9163	0.05078	Do not reject H_0
7	3.4598	0.05070	Do not reject H_0	22	0.37360	0.54105	Do not reject H_0
8	1.4943	0.22155	Do not reject H_0	23	0.43841	0.50789	Do not reject H_0
9	6.6148	0.01011	Reject H_0	24	2.0777	0.14947	Do not reject H_0
10	5.7708	0.01629	Reject H_0	25	2.1944	0.13852	Do not reject H_0
11	0.47308	0.50158	Do not reject H_0	26	0.97285	0.50934	Do not reject H_0
12	3.2846	0.05045	Do not reject H_0	27	3.0923	0.05307	Do not reject H_0
13	3.6361	0.05654	Do not reject H_0	28	5.1412	0.02336	Reject H_0
14	3.6366	0.05652	Do not reject H_0	29	2.4093	0.12062	Do not reject H_0
15	12.927	0.00032	Reject H_0	30	1.2210	0.26916	Do not reject H_0

Table 10.111: Breusch-Pagan test of the mutation rate in the range [0.25,0.3749].

Mutation rate in the range [0.25,0.3749]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	3.1496	0.07594	Do not reject H_0	16	1.2142	0.27049	Do not reject H_0
2	0.11566	0.73379	Do not reject H_0	17	0.02687	0.86978	Do not reject H_0
3	1.0313	0.30985	Do not reject H_0	18	6.0752	0.01370	Reject H_0
4	0.04405	0.83375	Do not reject H_0	19	3.0092	0.08279	Do not reject H_0
5	1.9229	0.16554	Do not reject H_0	20	2.7418	0.09775	Do not reject H_0
6	1.0770	0.29936	Do not reject H_0	21	2.4739	0.11575	Do not reject H_0
7	1.1904	0.27525	Do not reject H_0	22	0.57037	0.45011	Do not reject H_0
8	6.1821	0.01290	Reject H_0	23	8.1e-10	0.99998	Do not reject H_0
9	7.2811	0.00696	Reject H_0	24	0.51014	0.47508	Do not reject H_0
10	2.7285	0.09857	Do not reject H_0	25	1.9484	0.16276	Do not reject H_0
11	2.8482	0.09147	Do not reject H_0	26	0.46503	0.49528	Do not reject H_0
12	0.26921	0.60386	Do not reject H_0	27	1.9782	0.15958	Do not reject H_0
13	1.9483	0.16277	Do not reject H_0	28	2.3217	0.12758	Do not reject H_0
14	3.1429	0.07625	Do not reject H_0	29	3.9275	0.05075	Do not reject H_0
15	1.2142	0.27049	Do not reject H_0	30	1.2418	0.26513	Do not reject H_0

Table 10.112: Breusch-Pagan test of the mutation rate in the range [0.375,0.49].

Mutation rate in the range [0.375,0.49]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.99712	0.31801	Do not reject H_0	16	2.4284	0.11916	Do not reject H_0
2	0.12803	0.72048	Do not reject H_0	17	4.1032	0.05080	Do not reject H_0
3	7.0107	0.00810	Reject H_0	18	0.80054	0.37093	Do not reject H_0
4	5.659	0.01736	Reject H_0	19	7.8885	0.00497	Reject H_0
5	0.00235	0.96133	Do not reject H_0	20	2.8435	0.09174	Do not reject H_0
6	1.1522	0.2831	Do not reject H_0	21	0.22359	0.63632	Do not reject H_0
7	0.49162	0.48321	Do not reject H_0	22	0.26283	0.60818	Do not reject H_0
8	5.2075	0.02248	Reject H_0	23	2.0468	0.15253	Do not reject H_0
9	3.9445	0.05002	Do not reject H_0	24	1.2894	0.25616	Do not reject H_0
10	2.131	0.14435	Do not reject H_0	25	0.21419	0.6435	Do not reject H_0
11	0.43679	0.50868	Do not reject H_0	26	3.3481	0.06728	Do not reject H_0
12	1.4412	0.22994	Do not reject H_0	27	2.7221	0.09896	Do not reject H_0
13	1.681	0.19479	Do not reject H_0	28	0.46896	0.49347	Do not reject H_0
14	0.3652	0.54563	Do not reject H_0	29	0.36212	0.54733	Do not reject H_0
15	1.2647	0.26077	Do not reject H_0	30	2.2245	0.13584	Do not reject H_0

Table 10.113: Breusch-Pagan test of the crossover rate in the range [0.6,0.69].

Crossover rate in the range [0.6,0.69]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.00461	0.94583	Do not reject H_0	16	0.61822	0.43171	Do not reject H_0
2	3.3397	0.06762	Do not reject H_0	17	2.7692	0.09609	Do not reject H_0
3	2.9071	0.08818	Do not reject H_0	18	8.3102	0.00394	Reject H_0
4	7.8554	0.00506	Reject H_0	19	6.1718	0.01298	Reject H_0
5	0.93391	0.33385	Do not reject H_0	20	0.05567	0.81347	Do not reject H_0
6	3.3007	0.06925	Do not reject H_0	21	1.135	0.28671	Do not reject H_0
7	0.12494	0.72374	Do not reject H_0	22	6.4376	0.01117	Reject H_0
8	2.7409	0.09780	Do not reject H_0	23	0.95908	0.32742	Do not reject H_0
9	1.3689	0.24200	Do not reject H_0	24	7.3591	0.00667	Reject H_0
10	1.2345	0.26653	Do not reject H_0	25	3.1812	0.05087	Do not reject H_0
11	0.3883	0.5332	Do not reject H_0	26	0.97973	0.32227	Do not reject H_0
12	0.08214	0.77441	Do not reject H_0	27	0.07120	0.7896	Do not reject H_0
13	2.6225	0.10536	Do not reject H_0	28	0.12455	0.72415	Do not reject H_0
14	2.2468	0.13389	Do not reject H_0	29	1.5511	0.21297	Do not reject H_0
15	0.61822	0.43171	Do not reject H_0	30	3.778	0.05193	Do not reject H_0

Table 10.114: Breusch-Pagan test of the crossover rate in the range [0.7,0.79].

Crossover rate in the range [0.7,0.79]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	1.2912	0.25582	Do not reject H_0	16	0.00050	0.9821	Do not reject H_0
2	3.0309	0.08169	Do not reject H_0	17	0.74235	0.38891	Do not reject H_0
3	8.9167	0.00282	Reject H_0	18	3.6501	0.05606	Do not reject H_0
4	0.5981	0.4393	Do not reject H_0	19	0.17174	0.67857	Do not reject H_0
5	2.1275	0.14468	Do not reject H_0	20	2.0754	0.14969	Do not reject H_0
6	3.5748	0.05866	Do not reject H_0	21	0.99047	0.31963	Do not reject H_0
7	0.7151	0.39775	Do not reject H_0	22	1.4899	0.22223	Do not reject H_0
8	5.2768	0.02161	Do not reject H_0	23	3.6076	0.05183	Do not reject H_0
9	1.6624	0.19727	Do not reject H_0	24	0.05544	0.81385	Do not reject H_0
10	3.3468	0.05696	Do not reject H_0	25	0.16882	0.68117	Do not reject H_0
11	0.83685	0.3603	Do not reject H_0	26	1.4545	0.22781	Do not reject H_0
12	1.4298	0.2318	Do not reject H_0	27	0.04927	0.82433	Do not reject H_0
13	1.2314	0.26714	Do not reject H_0	28	7.0884	0.00775	Reject H_0
14	0.0216	0.88295	Do not reject H_0	29	6.7651	0.00929	Reject H_0
15	0.0005	0.9821	Do not reject H_0	30	3.8891	0.0567	Do not reject H_0

Table 10.115: Breusch-Pagan test of the crossover rate in the range [0.8,0.89].

Crossover rate in the range [0.8,0.89]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	1.2838	0.2572	Do not reject H_0	16	1.6636	0.19712	Do not reject H_0
2	0.67711	0.41058	Do not reject H_0	17	2.4668	0.11628	Do not reject H_0
3	1.9104	0.16692	Do not reject H_0	18	3.2012	0.05039	Do not reject H_0
4	0.69136	0.4057	Do not reject H_0	19	0.60476	0.43677	Do not reject H_0
5	0.74351	0.38854	Do not reject H_0	20	4.7185	0.02983	Do not reject H_0
6	0.86622	0.352	Do not reject H_0	21	8.9148	0.00282	Reject H_0
7	0.02054	0.88603	Do not reject H_0	22	2.1581	0.14182	Do not reject H_0
8	3.7162	0.05987	Do not reject H_0	23	8.7644	0.00307	Do not reject H_0
9	6.2422	0.01247	Reject H_0	24	2.4877	0.11474	Do not reject H_0
10	0.10895	0.74134	Do not reject H_0	25	2.8555	0.09106	Do not reject H_0
11	1.9592	0.16160	Do not reject H_0	26	0.33797	0.561	Do not reject H_0
12	8.4872	0.00357	Reject H_0	27	3.2845	0.05846	Do not reject H_0
13	2.5447	0.11066	Do not reject H_0	28	2.4143	0.12023	Do not reject H_0
14	7.0029	0.00813	Reject H_0	29	0.60476	0.43677	Do not reject H_0
15	1.6636	0.19712	Do not reject H_0	30	0.87073	0.35075	Do not reject H_0

Table 10.116: Breusch-Pagan test of the crossover rate in the range [0.9,0.99].

Crossover rate in the range [0.9,0.99]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	2.3036	0.12908	Do not reject H_0	16	8.015	0.00463	Reject H_0
2	0.29182	0.58905	Do not reject H_0	17	7.5958	0.00585	Reject H_0
3	0.28122	0.5959	Do not reject H_0	18	1.7266	0.18885	Do not reject H_0
4	3.1161	0.05247	Do not reject H_0	19	1.7054	0.19158	Do not reject H_0
5	3.0036	0.08308	Do not reject H_0	20	0.26785	0.60478	Do not reject H_0
6	1.8998	0.16810	Do not reject H_0	21	0.93363	0.33392	Do not reject H_0
7	0.06227	0.80294	Do not reject H_0	22	0.00012	0.9911	Do not reject H_0
8	6.5796	0.01031	Reject H_0	23	1.5673	0.2106	Do not reject H_0
9	0.44857	0.50301	Do not reject H_0	24	10.165	0.00143	Reject H_0
10	1.6362	0.20085	Do not reject H_0	25	2.0716	0.15006	Do not reject H_0
11	3.5483	0.05960	Do not reject H_0	26	1.1242	0.28901	Do not reject H_0
12	2.4204	0.11976	Do not reject H_0	27	1.5326	0.21573	Do not reject H_0
13	2.8456	0.09162	Do not reject H_0	28	3.5067	0.05894	Do not reject H_0
14	3.7889	0.05864	Do not reject H_0	29	0.96773	0.32525	Do not reject H_0
15	8.015	0.00463	Reject H_0	30	0.03033	0.86173	Do not reject H_0

Table 10.117: Breusch-Pagan test of the population size in the range [20,40].

Population size in the range [20,40]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	17.872	2.3e-05	Reject H_0	16	13.623	0.00022	Reject H_0
2	6.5991	0.01020	Reject H_0	17	1.3639	0.24287	Do not reject H_0
3	17.785	2.4e-05	Reject H_0	18	2.3354	0.12647	Do not reject H_0
4	1.3291	0.24896	Do not reject H_0	19	13.021	0.00030	Reject H_0
5	0.46919	0.49336	Do not reject H_0	20	10.549	0.00116	Reject H_0
6	4.6647	0.03078	Reject H_0	21	3.1957	0.07383	Do not reject H_0
7	7.5714	0.00593	Reject H_0	22	10.903	0.00096	Reject H_0
8	0.06502	0.79872	Do not reject H_0	23	11.977	0.00053	Reject H_0
9	2.1037	0.14695	Do not reject H_0	24	13.781	0.00020	Reject H_0
10	13.013	0.00030	Reject H_0	25	3.5618	0.05912	Do not reject H_0
11	11.253	0.00079	Reject H_0	26	0.88074	0.348	Do not reject H_0
12	0.21539	0.64258	Do not reject H_0	27	0.72967	0.39299	Do not reject H_0
13	4.4313	0.03528	Reject H_0	28	4.3007	0.03809	Reject H_0
14	8.124	0.00436	Reject H_0	29	6.9842	0.00822	Reject H_0
15	1.0276	0.31072	Do not reject H_0	30	4.7852	0.02870	Reject H_0

Table 10.118: Breusch-Pagan test of the population size in the range [41,60].

Population size in the range [41,60]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	17.872	2.3e-05	Reject H_0	16	13.623	0.00022	Reject H_0
2	6.5991	0.01020	Reject H_0	17	1.3639	0.24287	Do not reject H_0
3	17.785	2.4e-05	Reject H_0	18	2.3354	0.12647	Do not reject H_0
4	1.3291	0.24896	Do not reject H_0	19	13.021	0.00030	Reject H_0
5	0.46919	0.49336	Do not reject H_0	20	10.549	0.00116	Reject H_0
6	4.6647	0.03078	Reject H_0	21	3.1957	0.07383	Do not reject H_0
7	7.5714	0.00593	Reject H_0	22	10.903	0.00096	Reject H_0
8	0.06502	0.79872	Do not reject H_0	23	11.977	0.00053	Reject H_0
9	2.1037	0.14695	Do not reject H_0	24	13.781	0.00020	Reject H_0
10	13.013	0.00030	Reject H_0	25	3.5618	0.05912	Do not reject H_0
11	11.253	0.00079	Reject H_0	26	0.88074	0.348	Do not reject H_0
12	0.21539	0.64258	Do not reject H_0	27	0.72967	0.39299	Do not reject H_0
13	4.4313	0.03528	Reject H_0	28	4.3007	0.03809	Reject H_0
14	8.124	0.0043682	Reject H_0	29	6.9842	0.00822	Reject H_0
15	3.3926	0.06548	Do not reject H_0	30	13.716	0.00021	Reject H_0

Table 10.119: Breusch-Pagan test of the population size in the range [61,80].

Population size in the range [61,80]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	4.2e-05	0.99483	Do not reject H_0	16	5.7596	0.01639	Reject H_0
2	1.7587	0.18479	Do not reject H_0	17	10.573	0.00114	Reject H_0
3	0.0012	0.97163	Do not reject H_0	18	5.6065	0.01789	Reject H_0
4	9.988	0.00157	Reject H_0	19	0.94935	0.32989	Do not reject H_0
5	2.9940	0.08357	Do not reject H_0	20	5.3801	0.02036	Reject H_0
6	7.1645	0.00743	Reject H_0	21	1.5311	0.21595	Do not reject H_0
7	4.0835	0.05030	Do not reject H_0	22	0.41154	0.52119	Do not reject H_0
8	4.3887	0.03617	Reject H_0	23	4.2231	0.03987	Reject H_0
9	6.5346	0.01058	Reject H_0	24	3.9058	0.05012	Do not reject H_0
10	4.0481	0.05022	Do not reject H_0	25	14.533	0.00013	Reject H_0
11	2.7712	0.09597	Do not reject H_0	26	1.4732	0.22484	Do not reject H_0
12	1.2687	0.26002	Do not reject H_0	27	0.09639	0.7562	Do not reject H_0
13	1.0195	0.31263	Do not reject H_0	28	13.735	0.00021	Reject H_0
14	1.3742	0.24110	Do not reject H_0	29	4.5118	0.03366	Reject H_0
15	5.7596	0.01639	Reject H_0	30	3.2492	0.07145	Do not reject H_0

Table 10.120: Breusch-Pagan test of the population size in the range [81,100].

Population size in the range [81,100]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	3.4257	0.06419	Do not reject H_0	16	7.5629	0.00595	Reject H_0
2	0.00466	0.94555	Do not reject H_0	17	8.0127	0.00464	Reject H_0
3	2.6391	0.10426	Do not reject H_0	18	2.8465	0.09157	Do not reject H_0
4	8.566	0.00342	Reject H_0	19	2.4367	0.11852	Do not reject H_0
5	0.28128	0.59587	Do not reject H_0	20	1.8193	0.17739	Do not reject H_0
6	4.1544	0.04152	Reject H_0	21	3.7925	0.05148	Do not reject H_0
7	0.47661	0.48996	Do not reject H_0	22	2.6453	0.10386	Do not reject H_0
8	6.5258	0.01063	Reject H_0	23	0.81495	0.36666	Do not reject H_0
9	7.4832	0.00622	Reject H_0	24	7.4898	0.00620	Reject H_0
10	5.4566	0.01949	Reject H_0	25	0.43339	0.51033	Do not reject H_0
11	0.11447	0.73511	Do not reject H_0	26	2.0399	0.15321	Do not reject H_0
12	0.64132	0.42323	Do not reject H_0	27	1.4928	0.22178	Do not reject H_0
13	7.1744	0.00739	Reject H_0	28	4.9099	0.02670	Reject H_0
14	9.4299	0.00213	Reject H_0	29	5.885	0.01527	Reject H_0
15	7.5629	0.00595	Reject H_0	30	6.7616	0.00931	Reject H_0

Table 10.121: Breusch-Pagan test of the mating pool size in the range [0.1,0.249].

Mating pool size in the range [0.1,0.249]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.94497	0.331	Do not reject H_0	16	3.6802	0.05506	Do not reject H_0
2	1.9533	0.16224	Do not reject H_0	17	6.3022	0.01205	Reject H_0
3	0.29747	0.58547	Do not reject H_0	18	5.0454	0.02469	Reject H_0
4	1.702	0.19203	Do not reject H_0	19	6.8769	0.00873	Reject H_0
5	3.0305	0.08171	Do not reject H_0	20	0.06177	0.80372	Do not reject H_0
6	3.2763	0.05161	Do not reject H_0	21	0.44423	0.50509	Do not reject H_0
7	1.0648	0.30212	Do not reject H_0	22	1.2547	0.26266	Do not reject H_0
8	0.33393	0.56336	Do not reject H_0	23	1.4983	0.22093	Do not reject H_0
9	1.3193	0.25071	Do not reject H_0	24	0.07125	0.78951	Do not reject H_0
10	2.576	0.10849	Do not reject H_0	25	10.053	0.00152	Reject H_0
11	2.9878	0.08389	Do not reject H_0	26	3.8477	0.05768	Do not reject H_0
12	5.9055	0.01509	Reject H_0	27	3.4326	0.06392	Do not reject H_0
13	0.37783	0.53877	Do not reject H_0	28	4.2561	0.03910	Do not reject H_0
14	5.1845	0.02278	Reject H_0	29	1.6333	0.20125	Do not reject H_0
15	3.6802	0.05506	Do not reject H_0	30	3.4414	0.06358	Do not reject H_0

Table 10.122: Breusch-Pagan test of the mating pool size in the range [0.25,0.39].

Mating pool size in the range [0.25,0.39]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.16064	0.68857	Do not reject H_0	16	0.03851	0.84442	Do not reject H_0
2	0.61267	0.43378	Do not reject H_0	17	1.9837	0.159	Do not reject H_0
3	1.7597	0.18466	Do not reject H_0	18	5.7061	0.01690	Reject H_0
4	2.1424	0.14328	Do not reject H_0	19	8.4065	0.00373	Reject H_0
5	6.9687	0.00829	Reject H_0	20	0.38148	0.53681	Do not reject H_0
6	4.0187	0.05041	Do not reject H_0	21	1.9213	0.16572	Do not reject H_0
7	2.3033	0.1291	Do not reject H_0	22	6.614	0.01011	Reject H_0
8	2.1495	0.14262	Do not reject H_0	23	2.869	0.09030	Do not reject H_0
9	2.8412	0.09187	Do not reject H_0	24	1.1804	0.27727	Do not reject H_0
10	3.4227	0.06430	Do not reject H_0	25	0.03290	0.85605	Do not reject H_0
11	1.5518	0.21287	Do not reject H_0	26	1.0603	0.30315	Do not reject H_0
12	2.7696	0.09606	Do not reject H_0	27	0.79586	0.37233	Do not reject H_0
13	4.0049	0.05054	Do not reject H_0	28	5.5935	0.01802	Reject H_0
14	5.9114	0.01504	Do not reject H_0	29	2.0676	0.15046	Do not reject H_0
15	0.03851	0.84442	Do not reject H_0	30	2.8006	0.09423	Do not reject H_0

Table 10.123: Breusch-Pagan test of the mating pool size in the range [0.4,0.549].

Mating pool size in the range [0.4,0.549]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	2.5112	0.11304	Do not reject H_0	16	0.37360	0.54105	Do not reject H_0
2	3.3009	0.06924	Do not reject H_0	17	3.1480	0.05016	Do not reject H_0
3	3.8286	0.05038	Do not reject H_0	18	0.51974	0.47095	Do not reject H_0
4	3.1558	0.07565	Do not reject H_0	19	6.0653	0.01378	Reject H_0
5	1.2924	0.25561	Do not reject H_0	20	1.2846	0.25704	Do not reject H_0
6	1.8737	0.17105	Do not reject H_0	21	1.4404	0.23008	Do not reject H_0
7	0.40191	0.5261	Do not reject H_0	22	2.0777	0.14946	Do not reject H_0
8	7.9917	0.00469	Reject H_0	23	2.4709	0.11597	Do not reject H_0
9	3.6773	0.05515	Do not reject H_0	24	1.1329	0.28715	Do not reject H_0
10	8.388	0.00377	Reject H_0	25	1.2971	0.25475	Do not reject H_0
11	0.16582	0.68386	Do not reject H_0	26	3.6744	0.05525	Do not reject H_0
12	6.6838	0.00972	Reject H_0	27	13.804	0.00020	Reject H_0
13	1.0222	0.31201	Do not reject H_0	28	0.39939	0.52741	Do not reject H_0
14	3.3087	0.06891	Do not reject H_0	29	1.6912	0.19344	Do not reject H_0
15	6.0337	0.01403	Do not reject H_0	30	6.2603	0.01234	Reject H_0

Table 10.124: Breusch-Pagan test of the mating pool size in the range [0.55,0.69].

Mating pool size in the range [0.55,0.69]							
Run	bp	p	Conclusion	Run	bp	p	Conclusion
1	0.94497	0.331	Do not reject H_0	16	9.0625	0.00260	Reject H_0
2	4.0079	0.05096	Do not reject H_0	17	3.2826	0.07001	Do not reject H_0
3	0.73228	0.39215	Do not reject H_0	18	1.2178	0.26979	Do not reject H_0
4	4.0048	0.05035	Do not reject H_0	19	0.03222	0.85753	Do not reject H_0
5	4.5011	0.03387	Reject H_0	20	3.7364	0.05323	Do not reject H_0
6	5.3092	0.02121	Reject H_0	21	3.848	0.05080	Do not reject H_0
7	0.43849	0.50785	Do not reject H_0	22	0.41478	0.51955	Do not reject H_0
8	4.33	0.03744	Reject H_0	23	0.33627	0.56199	Do not reject H_0
9	2.1024	0.14707	Do not reject H_0	24	0.09525	0.7576	Do not reject H_0
10	5.7217	0.01675	Reject H_0	25	0.15222	0.69642	Do not reject H_0
11	2.9108	0.08798	Do not reject H_0	26	0.00279	0.95782	Do not reject H_0
12	0.04263	0.83642	Do not reject H_0	27	0.01523	0.90176	Do not reject H_0
13	3.0071	0.0829	Do not reject H_0	28	4.5426	0.03306	Do not reject H_0
14	2.595	0.10720	Do not reject H_0	29	0.40207	0.52602	Do not reject H_0
15	9.0625	0.00260	Reject H_0	30	3.2051	0.07341	Do not reject H_0

Stationarity

Table 10.125: KPSS test of the mutation rate in the range [0.001,0.1249].

Mutation rate in the range [0.001,0.1249]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.16265	0.03612	Reject H_0	16	0.04003	> 0.1	Do not reject H_0
2	0.08724	> 0.1	Do not reject H_0	17	0.02661	> 0.1	Do not reject H_0
3	0.09788	> 0.1	Do not reject H_0	18	0.13144	0.07697	Do not reject H_0
4	0.05469	> 0.1	Do not reject H_0	19	0.04268	> 0.1	Do not reject H_0
5	0.04446	> 0.1	Do not reject H_0	20	0.28369	0.01	Reject H_0
6	0.11654	> 0.1	Do not reject H_0	21	0.04845	> 0.1	Do not reject H_0
7	0.07631	> 0.1	Do not reject H_0	22	0.06317	> 0.1	Do not reject H_0
8	0.15647	0.05027	Do not reject H_0	23	0.05084	> 0.1	Do not reject H_0
9	0.04593	> 0.1	Do not reject H_0	24	0.06225	> 0.1	Do not reject H_0
10	0.04818	> 0.1	Do not reject H_0	25	0.04310	> 0.1	Do not reject H_0
11	0.04571	> 0.1	Do not reject H_0	26	0.15152	0.05039	Do not reject H_0
12	0.02902	> 0.1	Do not reject H_0	27	0.16983	0.03014	Reject H_0
13	0.21138	0.01173	Reject H_0	28	0.14833	0.05080	Do not reject H_0
14	0.23021	0.01	Reject H_0	29	0.31377	0.01	Reject H_0
15	0.04003	> 0.1	Do not reject H_0	30	0.02501	> 0.1	Do not reject H_0

Table 10.126: KPSS test of the mutation rate in the range [0.125,0.249].

Mutation rate in the range [0.125,0.249]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.15234	0.05047	Do not reject H_0	16	0.1312	0.07740	Do not reject H_0
2	0.12590	0.08721	Do not reject H_0	17	0.13924	0.06251	Do not reject H_0
3	0.25007	0.01	Reject H_0	18	0.04548	> 0.1	Do not reject H_0
4	0.08747	> 0.1	Do not reject H_0	19	0.15003	0.05030	Do not reject H_0
5	0.21131	0.01175	Reject H_0	20	0.18483	0.02168	Reject H_0
6	0.18672	0.02098	Reject H_0	21	0.17934	0.02374	Reject H_0
7	0.33137	0.01	Reject H_0	22	0.05205	> 0.1	Do not reject H_0
8	0.20675	0.01346	Reject H_0	23	0.07664	> 0.1	Do not reject H_0
9	0.08616	> 0.1	Do not reject H_0	24	0.18604	0.02123	Reject H_0
10	0.05749	> 0.1	Do not reject H_0	25	0.10257	> 0.1	Do not reject H_0
11	0.10896	> 0.1	Do not reject H_0	26	0.20818	0.01293	Reject H_0
12	0.17637	0.02486	Reject H_0	27	0.13447	0.07135	Do not reject H_0
13	0.12405	0.09065	Do not reject H_0	28	0.14415	0.05342	Do not reject H_0
14	0.10322	> 0.1	Do not reject H_0	29	0.21711	0.01	Reject H_0
15	0.1312	0.07740	Do not reject H_0	30	0.08919	> 0.1	Do not reject H_0

Table 10.127: KPSS test of the mutation rate in the range [0.25,0.3749].

Mutation rate in the range [0.25,0.3749]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.07288	> 0.1	Do not reject H_0	16	0.11866	> 0.1	Do not reject H_0
2	0.08451	> 0.1	Do not reject H_0	17	0.05941	> 0.1	Do not reject H_0
3	0.10637	> 0.1	Do not reject H_0	18	0.13056	0.05786	Do not reject H_0
4	0.18965	0.01988	Reject H_0	19	0.21891	0.01	Reject H_0
5	0.06851	> 0.1	Do not reject H_0	20	0.10131	> 0.1	Do not reject H_0
6	0.05077	> 0.1	Do not reject H_0	21	0.20526	0.01402	Reject H_0
7	0.12550	0.08795	Do not reject H_0	22	0.12394	0.09085	Do not reject H_0
8	0.10337	> 0.1	Do not reject H_0	23	0.35168	0.01	Do not reject H_0
9	0.23276	0.01	Reject H_0	24	0.09973	> 0.1	Do not reject H_0
10	0.04087	> 0.1	Do not reject H_0	25	0.12554	0.08788	Do not reject H_0
11	0.05237	> 0.1	Do not reject H_0	26	0.03901	> 0.1	Do not reject H_0
12	0.10060	> 0.1	Do not reject H_0	27	0.03994	> 0.1	Do not reject H_0
13	0.07378	> 0.1	Do not reject H_0	28	0.15151	0.05040	Do not reject H_0
14	0.12661	0.08589	Do not reject H_0	29	0.13577	0.06894	Do not reject H_0
15	0.11866	> 0.1	Do not reject H_0	30	0.07106	> 0.1	Do not reject H_0

Table 10.128: KPSS test of the mutation rate in the range [0.375,0.49].

Mutation rate in the range [0.375,0.49]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.16802	0.031648	Reject H_0	16	0.079227	> 0.1	Do not reject H_0
2	0.13072	0.0783	Do not reject H_0	17	0.24549	0.01	Reject H_0
3	0.12227	0.09395	Do not reject H_0	18	0.040092	> 0.1	Do not reject H_0
4	0.07440	> 0.1	Do not reject H_0	19	0.11682	> 0.1	Do not reject H_0
5	0.34922	0.01	Reject H_0	20	0.049547	> 0.1	Do not reject H_0
6	0.15589	0.0503424	Do not reject H_0	21	0.12188	0.09467	Do not reject H_0
7	0.09730	> 0.1	Do not reject H_0	22	0.15176	0.0502	Do not reject H_0
8	0.14151	0.058307	Do not reject H_0	23	0.20760	0.01315	Reject H_0
9	0.04277	> 0.1	Do not reject H_0	24	0.050193	> 0.1	Do not reject H_0
10	0.18418	0.021934	Reject H_0	25	0.028706	> 0.1	Do not reject H_0
11	0.08939	> 0.1	Do not reject H_0	26	0.14036	0.06044	Do not reject H_0
12	0.12949	0.080578	Do not reject H_0	27	0.15527	0.05022	Do not reject H_0
13	0.21166	0.011629	Reject H_0	28	0.038411	> 0.1	Do not reject H_0
14	0.13796	0.064892	Do not reject H_0	29	0.12029	0.09761	Do not reject H_0
15	0.07922	> 0.1	Do not reject H_0	30	0.087629	> 0.1	Do not reject H_0

Table 10.129: KPSS test of the crossover rate in the range [0.6,0.69].

Crossover rate in the range [0.6,0.69]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.10964	0.05697	Do not reject H_0	16	0.07534	> 0.1	Do not reject H_0
2	0.05626	> 0.1	Do not reject H_0	17	0.28315	0.01	Reject H_0
3	0.05198	> 0.1	Do not reject H_0	18	0.23634	0.01	Reject H_0
4	0.10722	0.05898	Do not reject H_0	19	0.1858	0.02132	Reject H_0
5	0.09440	> 0.1	Do not reject H_0	20	0.13713	0.06641	Do not reject H_0
6	0.07109	> 0.1	Do not reject H_0	21	0.13459	0.07112	Do not reject H_0
7	0.04609	> 0.1	Do not reject H_0	22	0.075262	> 0.1	Do not reject H_0
8	0.18120	0.02305	Reject H_0	23	0.19359	0.01840	Reject H_0
9	0.08158	> 0.1	Do not reject H_0	24	0.09824	> 0.1	Do not reject H_0
10	0.20176	0.01534	Reject H_0	25	0.11519	> 0.1	Do not reject H_0
11	0.09605	> 0.1	Do not reject H_0	26	0.080864	> 0.1	Do not reject H_0
12	0.0917	> 0.1	Do not reject H_0	27	0.081877	> 0.1	Do not reject H_0
13	0.08025	> 0.1	Do not reject H_0	28	0.10825	> 0.1	Do not reject H_0
14	0.10697	> 0.1	Do not reject H_0	29	0.10111	> 0.1	Do not reject H_0
15	0.07534	> 0.1	Do not reject H_0	30	0.036998	> 0.1	Do not reject H_0

Table 10.130: KPSS test of the crossover rate in the range [0.7,0.79].

Crossover rate in the range [0.7,0.79]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.10802	0.05832	Do not reject H_0	16	0.06333	> 0.1	Do not reject H_0
2	0.11418	> 0.1	Do not reject H_0	17	0.22928	0.01	Reject H_0
3	0.19353	0.01842	Reject H_0	18	0.10611	> 0.1	Do not reject H_0
4	0.43711	0.01	Reject H_0	19	0.22043	0.01	Reject H_0
5	0.10149	0.05542	Do not reject H_0	20	0.10578	> 0.1	Do not reject H_0
6	0.08382	> 0.1	Do not reject H_0	21	0.06789	> 0.1	Do not reject H_0
7	0.19621	0.01742	Reject H_0	22	0.06578	> 0.1	Do not reject H_0
8	0.03942	> 0.1	Do not reject H_0	23	0.24472	0.01	Reject H_0
9	0.11491	> 0.1	Do not reject H_0	24	0.03516	> 0.1	Do not reject H_0
10	0.05882	> 0.1	Do not reject H_0	25	0.06774	> 0.1	Do not reject H_0
11	0.08223	> 0.1	Do not reject H_0	26	0.14118	0.05893	Do not reject H_0
12	0.10908	0.05743	Do not reject H_0	27	0.17373	0.02689	Reject H_0
13	0.10986	0.05011	Do not reject H_0	28	0.07727	> 0.1	Do not reject H_0
14	0.18365	0.02213	Reject H_0	29	0.10298	> 0.1	Do not reject H_0
15	0.06333	> 0.1	Do not reject H_0	30	0.09122	> 0.1	Do not reject H_0

Table 10.131: KPSS test of the crossover rate in the range [0.8,0.89].

Crossover rate in the range [0.8,0.89]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.14148	0.05836	Do not reject H_0	16	0.05749	> 0.1	Do not reject H_0
2	0.12410	0.09055	Do not reject H_0	17	0.13077	0.05602	Do not reject H_0
3	0.12907	0.08134	Do not reject H_0	18	0.06414	> 0.1	Do not reject H_0
4	0.1196	0.09889	Do not reject H_0	19	0.19833	0.01662	Reject H_0
5	0.07163	> 0.1	Do not reject H_0	20	0.16055	0.03787	Reject H_0
6	0.03990	> 0.1	Do not reject H_0	21	0.12190	0.09463	Do not reject H_0
7	0.09896	> 0.1	Do not reject H_0	22	0.16194	0.03671	Reject H_0
8	0.08315	> 0.1	Do not reject H_0	23	0.16188	0.03677	Reject H_0
9	0.12020	0.09777	Do not reject H_0	24	0.16917	0.03069	Reject H_0
10	0.10399	> 0.1	Do not reject H_0	25	0.12172	0.09496	Do not reject H_0
11	0.04341	> 0.1	Do not reject H_0	26	0.12296	0.09266	Do not reject H_0
12	0.05027	> 0.1	Do not reject H_0	27	0.12866	0.08210	Do not reject H_0
13	0.20166	0.01537	Reject H_0	28	0.12033	0.09754	Do not reject H_0
14	0.08747	> 0.1	Do not reject H_0	29	0.12097	0.09634	Do not reject H_0
15	0.05749	> 0.1	Do not reject H_0	30	0.13894	0.06307	Do not reject H_0

Table 10.132: KPSS test of the crossover rate in the range [0.9,0.99].

Crossover rate in the range [0.9,0.99]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.07922	> 0.1	Do not reject H_0	16	0.02956	> 0.1	Do not reject H_0
2	0.13823	0.06438	Do not reject H_0	17	0.08515	> 0.1	Do not reject H_0
3	0.14713	0.05072	Do not reject H_0	18	0.12719	0.08482	Do not reject H_0
4	0.06111	> 0.1	Do not reject H_0	19	0.13002	0.07958	Do not reject H_0
5	0.11268	> 0.1	Do not reject H_0	20	0.08427	> 0.1	Do not reject H_0
6	0.09993	> 0.1	Do not reject H_0	21	0.05825	> 0.1	Do not reject H_0
7	0.14552	0.05089	Do not reject H_0	22	0.04363	> 0.1	Do not reject H_0
8	0.10397	> 0.1	Do not reject H_0	23	0.12342	0.09182	Do not reject H_0
9	0.05152	> 0.1	Do not reject H_0	24	0.13295	0.07416	Do not reject H_0
10	0.08416	> 0.1	Do not reject H_0	25	0.03553	> 0.1	Do not reject H_0
11	0.21760	0.01	Reject H_0	26	0.04902	> 0.1	Do not reject H_0
12	0.08742	> 0.1	Do not reject H_0	27	0.19796	0.01676	Reject H_0
13	0.08145	> 0.1	Do not reject H_0	28	0.04369	> 0.1	Do not reject H_0
14	0.14319	0.05520	Do not reject H_0	29	0.08917	> 0.1	Do not reject H_0
15	0.02956	> 0.1	Do not reject H_0	30	0.07057	> 0.1	Do not reject H_0

Table 10.133: KPSS test of the population size in the range [20,40].

Population size in the range [20,40]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.30711	0.01	Reject H_0	16	0.30209	0.01	Reject H_0
2	0.39549	0.01	Reject H_0	17	0.57824	0.01	Reject H_0
3	0.23875	0.01	Reject H_0	18	0.36562	0.01	Reject H_0
4	0.15338	0.04385	Reject H_0	19	0.46786	0.01	Reject H_0
5	0.33213	0.01	Reject H_0	20	0.44741	0.01	Reject H_0
6	0.38826	0.01	Reject H_0	21	0.34654	0.01	Reject H_0
7	0.09869	> 0.1	Do not reject H_0	22	0.53479	0.01	Reject H_0
8	0.41719	0.01	Reject H_0	23	0.32438	0.01	Reject H_0
9	0.29629	0.01	Reject H_0	24	0.4927	0.01	Reject H_0
10	0.12031	0.09757	Do not reject H_0	25	0.33462	0.01	Reject H_0
11	0.38525	0.01	Reject H_0	26	0.05351	> 0.1	Do not reject H_0
12	0.22160	0.01	Reject H_0	27	0.41098	0.01	Reject H_0
13	0.44008	0.01	Reject H_0	28	0.32131	0.01	Reject H_0
14	0.33700	0.01	Reject H_0	29	0.40164	0.01	Reject H_0
15	0.26018	0.01	Reject H_0	30	0.4663	0.01	Reject H_0

Table 10.134: KPSS test of the population size in the range [41,60].

Population size in the range [41,60]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.30455	0.01	Reject H_0	16	0.15020	0.04650	Reject H_0
2	0.22119	0.01	Reject H_0	17	0.23122	0.01	Reject H_0
3	0.40016	0.01	Reject H_0	18	0.24446	0.01	Reject H_0
4	0.41991	0.01	Reject H_0	19	0.26465	0.01	Reject H_0
5	0.15633	0.04139	Reject H_0	20	0.27724	0.01	Reject H_0
6	0.11170	> 0.1	Do not reject H_0	21	0.20618	0.01368	Reject H_0
7	0.4449	0.01	Reject H_0	22	0.24849	0.01	Reject H_0
8	0.1842	0.02192	Reject H_0	23	0.1968	0.0172	Reject H_0
9	0.29323	0.01	Reject H_0	24	0.20926	0.01252	Reject H_0
10	0.27789	0.01	Reject H_0	25	0.28091	0.01	Reject H_0
11	0.19337	0.01848	Reject H_0	26	0.49318	0.01	Reject H_0
12	0.42723	0.01	Reject H_0	27	0.2475	0.01	Reject H_0
13	0.15972	0.03856	Reject H_0	28	0.15443	0.04297	Reject H_0
14	0.28723	0.01	Reject H_0	29	0.17402	0.02665	Reject H_0
15	0.24526	0.01	Reject H_0	30	0.51235	0.01	Reject H_0

Table 10.135: KPSS test of the population size in the range [61,80].

Population size in the range [61,80]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.26521	0.01	Reject H_0	16	0.24856	0.01	Reject H_0
2	0.08768	> 0.1	Do not reject H_0	17	0.33024	0.01	Reject H_0
3	0.26671	0.01	Reject H_0	18	0.32051	0.01	Reject H_0
4	0.28079	0.01	Reject H_0	19	0.35646	0.01	Reject H_0
5	0.04504	> 0.1	Do not reject H_0	20	0.1729	0.02758	Reject H_0
6	0.14994	0.05	Do not reject H_0	21	0.34162	0.01	Reject H_0
7	0.3865	0.01	Reject H_0	22	0.33052	0.01	Reject H_0
8	0.12418	0.09041	Do not reject H_0	23	0.26031	0.01	Reject H_0
9	0.32584	0.01	Reject H_0	24	0.19790	0.01678	Reject H_0
10	0.21565	0.01013	Reject H_0	25	0.24326	0.01	Reject H_0
11	0.19197	0.01901	Reject H_0	26	0.44369	0.01	Reject H_0
12	0.23618	0.01	Reject H_0	27	0.26786	0.01	Reject H_0
13	0.21779	0.01	Reject H_0	28	0.14876	0.05070	Do not reject H_0
14	0.24935	0.01	Reject H_0	29	0.45822	0.01	Reject H_0
15	0.24856	0.01	Reject H_0	30	0.23207	0.01	Reject H_0

Table 10.136: KPSS test of the population size in the range [81,100].

Population size in the range [81,100]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.09473	> 0.1	Do not reject H_0	16	0.33264	0.01	Reject H_0
2	0.07325	> 0.1	Do not reject H_0	17	0.11081	> 0.1	Do not reject H_0
3	0.05209	> 0.1	Do not reject H_0	18	0.16165	0.03695	Reject H_0
4	0.16506	0.03411	Reject H_0	19	0.18540	0.02147	Reject H_0
5	0.09920	> 0.1	Do not reject H_0	20	0.22187	0.01	Reject H_0
6	0.14336	0.05488	Do not reject H_0	21	0.16693	0.03255	Reject H_0
7	0.15152	0.05039	Do not reject H_0	22	0.16353	0.03539	Reject H_0
8	0.07268	> 0.1	Do not reject H_0	23	0.35409	0.01	Reject H_0
9	0.41516	0.01	Reject H_0	24	0.07009	> 0.1	Do not reject H_0
10	0.10895	> 0.1	Do not reject H_0	25	0.09275	> 0.1	Do not reject H_0
11	0.22944	0.01	Reject H_0	26	0.23057	0.01	Reject H_0
12	0.09439	> 0.1	Do not reject H_0	27	0.07456	> 0.1	Do not reject H_0
13	0.02800	> 0.1	Do not reject H_0	28	0.17434	0.02638	Reject H_0
14	0.24904	0.01	Reject H_0	29	0.20846	0.01282	Reject H_0
15	0.33264	0.01	Reject H_0	30	0.15809	0.03992	Reject H_0

Table 10.137: KPSS test of the mating pool size in the range [0.1,0.249].

Mating pool size in the range [0.1,0.249]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.18219	0.02267	Reject H_0	16	0.06906	> 0.1	Do not reject H_0
2	0.14344	0.05474	Do not reject H_0	17	0.11387	> 0.1	Do not reject H_0
3	0.10656	> 0.1	Do not reject H_0	18	0.11303	> 0.1	Do not reject H_0
4	0.17943	0.02371	Reject H_0	19	0.1849	0.02166	Reject H_0
5	0.40736	0.01	Reject H_0	20	0.19804	0.01673	Reject H_0
6	0.06891	> 0.1	Do not reject H_0	21	0.11429	> 0.1	Do not reject H_0
7	0.08714	> 0.1	Do not reject H_0	22	0.05808	> 0.1	Do not reject H_0
8	0.14332	0.05496	Do not reject H_0	23	0.07322	> 0.1	Do not reject H_0
9	0.21602	0.01	Reject H_0	24	0.14610	0.05091	Do not reject H_0
10	0.17306	0.02744	Reject H_0	25	0.10310	> 0.1	Do not reject H_0
11	0.20056	0.01578	Reject H_0	26	0.20386	0.01455	Reject H_0
12	0.18733	0.02075	Reject H_0	27	0.08312	> 0.1	Do not reject H_0
13	0.24975	0.01	Reject H_0	28	0.093	> 0.1	Do not reject H_0
14	0.09285	> 0.1	Do not reject H_0	29	0.08282	> 0.1	Do not reject H_0
15	0.06906	> 0.1	Do not reject H_0	30	0.14047	0.06023	Do not reject H_0

Table 10.138: KPSS test of the mating pool size in the range [0.25,0.39].

Mating pool size in the range [0.25,0.39]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.12811	0.05024	Do not reject H_0	16	0.14372	0.05422	Do not reject H_0
2	0.07763	> 0.1	Do not reject H_0	17	0.05986	> 0.1	Do not reject H_0
3	0.05558	> 0.1	Do not reject H_0	18	0.10652	> 0.1	Do not reject H_0
4	0.09760	> 0.1	Do not reject H_0	19	0.05166	> 0.1	Do not reject H_0
5	0.14169	0.05798	Do not reject H_0	20	0.08567	> 0.1	Do not reject H_0
6	0.09805	> 0.1	Do not reject H_0	21	0.07759	> 0.1	Do not reject H_0
7	0.08948	> 0.1	Do not reject H_0	22	0.09420	> 0.1	Do not reject H_0
8	0.05716	> 0.1	Do not reject H_0	23	0.1263	0.08648	Do not reject H_0
9	0.09062	> 0.1	Do not reject H_0	24	0.06055	> 0.1	Do not reject H_0
10	0.06688	> 0.1	Do not reject H_0	25	0.05519	> 0.1	Do not reject H_0
11	0.03402	> 0.1	Do not reject H_0	26	0.12354	0.09158	Do not reject H_0
12	0.13192	0.07608	Do not reject H_0	27	0.13621	0.06813	Do not reject H_0
13	0.26866	0.01	Reject H_0	28	0.13338	0.07336	Do not reject H_0
14	0.05937	> 0.1	Do not reject H_0	29	0.12925	0.08101	Do not reject H_0
15	0.14372	0.05422	Do not reject H_0	30	0.03435	> 0.1	Do not reject H_0

Table 10.139: KPSS test of the mating pool size in the range [0.4,0.549].

Mating pool size in the range [0.4,0.549]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.05436	> 0.1	Do not reject H_0	16	0.21438	0.01060	Reject H_0
2	0.27486	0.01	Reject H_0	17	0.14849	0.05092	Do not reject H_0
3	0.21100	0.01187	Reject H_0	18	0.10645	> 0.1	Do not reject H_0
4	0.23396	0.01	Reject H_0	19	0.12496	0.08896	Do not reject H_0
5	0.09685	> 0.1	Do not reject H_0	20	0.07780	> 0.1	Do not reject H_0
6	0.03300	> 0.1	Do not reject H_0	21	0.18583	0.02131	Reject H_0
7	0.05396	> 0.1	Do not reject H_0	22	0.16346	0.03545	Reject H_0
8	0.21034	0.01212	Reject H_0	23	0.08378	> 0.1	Do not reject H_0
9	0.05131	> 0.1	Do not reject H_0	24	0.10665	> 0.1	Do not reject H_0
10	0.12755	0.08416	Do not reject H_0	25	0.07154	> 0.1	Do not reject H_0
11	0.13333	0.07345	Do not reject H_0	26	0.05245	> 0.1	Do not reject H_0
12	0.26486	0.01	Reject H_0	27	0.13887	0.06319	Do not reject H_0
13	0.17062	0.02948	Reject H_0	28	0.05559	> 0.1	Do not reject H_0
14	0.07737	> 0.1	Do not reject H_0	29	0.03566	> 0.1	Do not reject H_0
15	0.21438	0.01060	Reject H_0	30	0.10054	> 0.1	Do not reject H_0

Table 10.140: KPSS test of the mating pool size in the range [0.55,0.69].

Mating pool size in the range [0.55,0.69]							
Run	kpss	p	Conclusion	Run	kpss	p	Conclusion
1	0.16135	0.03720	Reject H_0	16	0.10032	> 0.1	Do not reject H_0
2	0.18514	0.02157	Reject H_0	17	0.11569	> 0.1	Do not reject H_0
3	0.08462	> 0.1	Do not reject H_0	18	0.04007	> 0.1	Do not reject H_0
4	0.05979	> 0.1	Do not reject H_0	19	0.13181	0.07627	Do not reject H_0
5	0.08239	> 0.1	Do not reject H_0	20	0.19672	0.01723	Reject H_0
6	0.09574	> 0.1	Do not reject H_0	21	0.14414	0.05032	Do not reject H_0
7	0.08822	> 0.1	Do not reject H_0	22	0.06138	> 0.1	Do not reject H_0
8	0.17311	0.02741	Reject H_0	23	0.17689	0.02466	Reject H_0
9	0.12379	0.09113	Do not reject H_0	24	0.07024	> 0.1	Do not reject H_0
10	0.03703	> 0.1	Do not reject H_0	25	0.11718	> 0.1	Do not reject H_0
11	0.16470	0.03442	Reject H_0	26	0.19053	0.01955	Reject H_0
12	0.06117	> 0.1	Do not reject H_0	27	0.08621	> 0.1	Do not reject H_0
13	0.03682	> 0.1	Do not reject H_0	28	0.12723	0.08475	Do not reject H_0
14	0.12978	0.08003	Do not reject H_0	29	0.10173	> 0.1	Do not reject H_0
15	0.10032	> 0.1	Do not reject H_0	30	0.15212	0.05049	Do not reject H_0

Bibliography

[1] Belarmino Adenso-Díaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.

[2] Mikael Åkerholm, Johan Fredriksson, Kristian Sandström, and Ivica Crnkovic. Quality attribute support in a component technology for vehicular software. In *Fourth Conference on Software Engineering Research and Practice in Sweden*, 2004.

[3] Aldeida Aleti, Stefan Björnander, Lars Grunske, and Indika Meedeniya. ArcheOpterix: An extendable tool for architecture optimization ofAADL models. In *Model-based Methodologies for Pervasive and Embedded Software (MOMPES)*, pages 61–71. ACM and IEEE Digital Libraries, 2009.

[4] Aldeida Aleti, Lars Grunske, Indika Meedeniya, and Irene Moser. Let the ants deploy your software - an ACO based deployment optimisation strategy. In *ASE*, pages 505–509. IEEE Computer Society, 2009.

[5] Aldeida Aleti and Irene Moser. Predictive parameter control. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 561–568, 2011.

[6] Peter J. Angeline. Adaptive and self-adaptive evolutionary computations. In Marimuthu Palaniswami and Yianni Attikiouzel, editors, *Computational*

Intelligence: A Dynamic Systems Perspective, pages 152–163. IEEE Press, 1995.

[7] Jaroslaw Arabas, Zbigniew Michalewicz, and Jan J. Mulawka. GAVaPS - A genetic algorithm with varying population size. In *International Conference on Evolutionary Computation*, pages 73–78, 1994.

[8] David A.Van Veldhuizen and Gary B. Lamont. On measuring multiobjective evolutionary algorithm performance. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 204–211. IEEE Service Center, 2000.

[9] Thomas Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Parallel Problem Solving from Nature 2,PPSN-II*, pages 87–96. Elsevier, 1992.

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

[11] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[12] Thomas Bäck. Introduction to the special issue: Self-adaptation. *Evolutionary Computation*, 9(2):iii–iv, 2001.

[13] Thomas Bäck, Ágoston Endre Eiben, and Nikolai A. L. van der Vaart. An empirical study on gas without parameters. In *Parallel Problem Solving from Nature – PPSN VI (6th PPSN'2000)*, volume 1917 of *Lecture Notes in Computer Science (LNCS)*, pages 315–324. Springer-Verlag (New York), 2000.

[14] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[15] Thomas Bäck and Martin Schütz. Intelligent mutation rate control in canonical genetic algorithms. *Lecture Notes in Computer Science*, 1079:158–167, 1996.

[16] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics, 4th International Workshop, Proceedings*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2007.

[17] Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss. Sequential parameter optimization. In *IEEE Congress on Evolutionary Computation*, pages 773–780. IEEE, 2005.

[18] M. Cecilia Bastarrica, Alexander A. Shvartsman, and Steven A. Demurjian. A binary integer programming model for optimal object distribution. In *OPODIS*, pages 211–226. Hermes, 1998.

[19] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies - A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[20] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kaufmann Publishers, 2002.

[21] Tobias Blickle, Jürgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *Design Automation for Embedded Systems*, 3(1):23–58, 1998.

[22] Mark F. Bramlette. Initialization, mutation and selection methods in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1991.

[23] H. J. Bremermann, M. Rogson, and S. Salaff. Global properties of evolution processes. In *Natural Automata and Useful Simulations*, pages 3–41, 1966.

[24] H.J. Bremermann. The evolution of intelligence. the nervous system as a model of its environment. Technical Report 477(17), Department of Mathematics, University of Washington, Seattle, 1958.

[25] Manfred Broy. Challenges in automotive software engineering. In *International Conference on Software Engineering (ICSE'06)*, pages 33–42. ACM, 2006.

[26] Ken Butts, Dave Bostic, Alongkrit Chutinan, Jeffrey Cook, Bill Milam, and Yanxin Wang. Usage scenarios for an automated model compiler. *Lecture Notes in Computer Science*, 2211:66–79, 2001.

[27] Radu Calinescu and Marta Kwiatkowska. Using quantitative analysis to implement autonomic it systems. In *International Conference on Software Engineering, ICSE*, pages 100–110. IEEE, 2009.

[28] Raphaël Cerf. Critical control of a genetic algorithm. *CoRR*, abs/1005.3390, 2010.

[29] Jorge Cervantes and Christopher R. Stephens. Limitations of existing mutation rate heuristics and how a rank GA overcomes them. *IEEE Trans. Evolutionary Computation*, 13(2):369–397, 2009.

[30] Maw-Sheng Chern. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*, 11(5):309315, June 1992.

[31] David W. Coit and Alice E. Smith. Reliability optimization of series-parallel systems using a genetic algorithm. *Reliability, IEEE Transactions on*, 45(2):254 – 260, 266, June 1996.

[32] David W. Coit and Alice E. Smith. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability*, 45(2):254–260, 1996.

[33] David Corne, Martin J. Oates, and Douglas B. Kell. On fitness distributions and expected fitness gain of mutation rates in parallel evolutionary algorithms. In *Parallel Problem Solving from Nature – PPSN VII (7th PPSN'02)*, volume 2439 of *Lecture Notes in Computer Science (LNCS)*, pages 132–141. Springer-Verlag, 2002.

[34] Vittorio Cortellessa, Fabrizio Marinelli, and Pasqualina Potena. Automated selection of software components based on cost/reliability tradeoff. In *Software Architecture,(EWSA'06)*, volume 4344, pages 66–81. Springer, 2006.

[35] Steven P. Coy, Bruce L. Golden, George C. Runger, and Edward A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001.

[36] Luis DaCosta, Alvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 913–920. ACM, 2008.

[37] Lawrence Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79. Morgan Kaufman, 1989.

[38] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reingold, 1991.

[39] Kenneth A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1995.

[40] Kalyanmoy Deb and Hans-Georg Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation*, 9(2):197–221, 2001.

[41] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm:NSGA-II. *IEEE-Evolutionary Computation*, 6:182–197, 2002.

[42] K. A. DeJong. *Analysis of Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, The University of Michigan, 1975.

[43] Kenneth DeJong. Parameter setting in EAs: a 30 year perspective. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 1–18. Springer, 2007.

[44] Jianguo Ding, Bernd J. Krämer, Yingcai Bai, and Hansheng Chen. Backward inference in bayesian networks for distributed systems management. *Network Systems Management*, 13(4):409–427, 2005.

[45] Ágoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transations on Evolutionary Computation*, 3(2):124–141, 2007.

[46] Ágoston Endre Eiben and M. Jelasity. A critical note on experimental research methodology in EC. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 582–587. IEEE Press, 2002.

[47] Ágoston Endre Eiben, E. Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 41–50. Springer-Verlag, 2004.

[48] Ágoston Endre Eiben and Martijn C. Schut. New ways to calibrate evolutionary algorithms. In Patrick Siarry and Zbigniew Michalewicz, editors, *Advances in Metaheuristics for Hard Optimization*, Natural Computing Series, pages 153–177. Springer, 2008.

[49] Ágoston Endre Eiben and Selmar K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[50] M. Eisenring, Lothar Thiele, and Eckart Zitzler. Conflicting Criteria in Embedded System Design. *IEEE Design and Test*, 17(2):51–59, 2000.

[51] Mohammed A. El-Beltagy, Prasanth B. Nair, and Andy J. Keane. Metamodeling techniques for evolutionary optimization of computationally expensive problems: Promises and limitations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 196–203. Morgan Kaufmann, 1999.

[52] Henrik Esbensen and Ernest S. Kuh. Design space exploration using the genetic algorithm. In *IEEE International Symposium on Circuits and Systems (ISCAS'96)*, pages 500–503. IEEE, 1996.

[53] Raziyeh Farmani and Jonathan A. Wright. Self-adaptive fitness formulation for constrained optimization. *IEEE Trans. Evolutionary Computation*, 7(5):445–455, 2003.

[54] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Extreme value based adaptive operator selection. In *Parallel Problem Solving from Nature (PPSN 08), 10th International Conference, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 2008.

[55] Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence - Special Issue on Learning and Intelligent Optimization*, 2010.

[56] Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 779–786. ACM, 2009.

[57] Terence C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 104–109. Morgan Kaufman, 1989.

[58] D. B. Fogel, L. J. Fogel, and J. W. Atmar. Meta-evolutionary programming. In R. R. Chen, editor, *Proceedings of 25th Asilomar Conference on Signals, Systems and Computers*, pages 540–545, 1991.

[59] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[60] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, 1966.

[61] Johan Fredriksson, Thomas Nolte, Mikael Nolin, and Heinz Schmidt. Contract-based reusable worst-case execution time estimate. In *The International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 39–46, 2007.

[62] Johan Fredriksson, Kristian Sandström, and MikaelÅkerholm. Optimizing resource usage in component-based real-time systems. In *Component-Based Software Engineering*, volume 3489 of *LNCS*, pages 49–65. Springer, 2005.

[63] T. Yokoyama G. Taguchi. *Taguchi methods: design of experiments*. ASI Press, 1993.

[64] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric statistical inference*. CRC Press, 2003.

[65] M. Giger, D. Keller, and P. Ermanni. Aorcea - an adaptive operator rate controlled evolutionary algorithm. *Computers and Structures*, 85(19-20):1547 – 1561, 2007.

[66] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[67] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93, San Mateo, 1991. Morgan Kaufmann.

[68] Katerina Goševa-Popstojanova and Kishor S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204, 2001.

[69] John J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1):122–128, 1986.

[70] John J. Grefenstette and James E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 20–27. Morgan Kaufmann Publishers, Inc., 1989.

[71] Lars Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In *International Conference on Software Engineering, ICSE*, pages 849–852. ACM, 2006.

[72] Lars Grunske. Early quality prediction of component-based systems - a generic framework. *Journal of Systems and Software*, 80(5):678–686, 2007.

[73] Lars Grunske, Peter A. Lindsay, Egor Bondarev, Yiannis Papadopoulos, and David Parker. An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In Rogério de Lemos, Cristina Gacek, and Alexander B. Romanovsky, editors, *Architecting Dependable Systems*, volume 4615 of *Lecture Notes in Computer Science*, pages 188–209. Springer, 2006.

[74] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[75] Günter Heiner and Thomas Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In *International Symposium on Fault-Tolerant Computing (FTCS'98)*, pages 402–407, 1998.

[76] Francisco Herrera and Manuel Lozano. Adaptive genetic operators based on coevolution with fuzzy behaviors. *IEEE-EC*, 5:149–165, April 2001.

[77] J. Hesser and R. Manner. Towards an optimal mutation probability for genetic algorithms. *Lecture Notes in Computer Science*, 496:23–32, 1991.

[78] Robert Hinterding, Zbigniew Michalewicz, and T. C. Peachey. Self-adaptive genetic algorithm for numeric functions. *Lecture Notes in Computer Science*, 1141:420–429, 1996.

[79] C. W. Ho, K. H. Lee, and K. S. Leung. A genetic algorithm based on mutation and crossover with adaptive probabilities. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 768–775. IEEE Press, 1999.

[80] Yosef Hochberg and Ajit C. Tamhane. Multiple comparisons in a mixed model. *The American Statistician*, 37(4):305–307, 1983.

[81] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan, 1975.

[82] Tzung-Pei Hong, Hong-Shung Wang, and Wei-Chou Chen. Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics*, 6(4):439–455, 2000.

[83] Tzung-Pei Hong, Hong-Shung Wang, Wen-Yang Lin, and Wen-Yuan Lee. Evolution of appropriate crossover and mutation operators in a genetic process. *Appl. Intell*, 16(1):7–17, 2002.

[84] Christian Igel and Martin Kreutz. Operator adaptation in evolutionary computation and its application to structure optimization of neural networks. *Neurocomputing*, 55(1-2):347–361, 2003.

[85] Christian Igel, Stefan Wiegand, and Frauke Friedrichs. Evolutionary optimization of neural systems: The use of strategy adaptation. In *Trends and Applications in Constructive Approximation*, volume 151 of *International Series of Numerical Mathematics*, pages 103–123. Birkhäuser Basel, 2005.

[86] Shariful Islam, Robert Lindstrom, and Neeraj Suri. Dependability driven integration of mixed criticalitySW components. In *International Symposium on object/component/service-oriented Real-time distributed Computing, ISORC*, pages 485–495. IEEE Computer Society, 2006.

[87] Shariful Islam and Neeraj Suri. A multi variable optimization approach for the design of integrated dependable real-time embedded systems. In *Embedded and Ubiquitous Computing, International Conference, Proceedings*, volume 4808 of *LNCS*, pages 517–530. Springer, 2007.

[88] ISO/IEC. IEEE International Standard 1471 2000 - Systems and software engineering - Recommended practice for architectural description of software-intensive systems, 2000.

[89] Cezary Z. Janikow and Zbigniew Michalwicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, pages 31–36. Morgan Kaufmann Publishers, 1991.

[90] Thomas Jansen and Kenneth De Jong. An analysis of the role of offspring population size in EAs. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 238–246. Morgan Kaufmann Publishers, 2002.

[91] Arshad Jhumka, Martin Hiller, and Neeraj Suri. Component-based synthesis of dependable embedded software. In Werner Damm and Ernst-Rüdiger Olderog,

editors, *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT*, volume 2469, pages 111–128, 2002.

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

[93] Terry Jones. A description of holland's royal road function. *Evolutionary Computation*, 2(4):409–415, 1994.

[94] Bryant A. Julstrom. What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 81–87, San Francisco, CA, 1995. Morgan Kaufmann.

[95] Bryant A. Julstrom. Adaptive operator probabilities in a genetic algorithm that applies three operators. In *ACM Symposium on Applied Computing (SAC)*, pages 233–238. ACM, 1997.

[96] Tatiana Kichkaylo, Anca-Andreea Ivan, and Vijay Karamcheti. Constrained component deployment in wide-area networks using ai planning techniques. In *17th International Parallel and Distributed Processing Symposium*, page 3, 2003.

[97] Tatiana Kichkaylo and Vijay Karamcheti. Optimal resource-aware deployment planning for component-based distributed applications. In *HPDC:High Performance Distributed Computing*, pages 150–159. IEEE Computer Society, 2004.

[98] Seong-Hee Kim and Barry L. Nelson. A fully sequential procedure for indifference-zone selection in simulation. *ACM Transactions on Modeling and Computer Simulation*, 11(3):251–273, July 2001.

[99] Joshua Knowles and David Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. In *Evolutionary Multi-Criterion Optimization, International Conference, EMO 2003*, volume 2632, pages 295–310. Springer. Lecture Notes in Computer Science., 2003.

[100] Joshua Knowles, Lothar Thiele, and Eckart Zitzler. A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, 2006.

[101] T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[102] Hermann Kopetz. *Real-Time Systems*. Kluwer Academic, 1997.

[103] Oliver Kramer. Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 3(2):51–65, 2010.

[104] Johannes W. Kruisselbrink, Rui Li, Edgar Reehuis, Jeroen Eggermont, and Thomas Bäck. On the log-normal self-adaptation of the mutation rate in binary search spaces. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceeding*, pages 893–900. ACM, 2011.

[105] Peter Kubat. Assessing reliability of modular software. *Operations Research Letters*, 8(1):35–41, 1989.

[106] Sadan Kulturel-Konak and Alice E Smith David W Coit. Efficiently solving the redundancy allocation problem using tabu search. *IIE Transactions*, 35(6):515–526, 2003.

[107] Sadan Kulturel-Konak, David W. Coit, and Fatema Baheranwala. Pruned pareto-optimal sets for the system redundancy allocation problem based on multiple prioritized objectives. *Journal of Heuristics*, 14(4):335–357, 2008.

[108] D. Kwiatkowski, P.C.B. Phillips, P. Schmidt, and Y. Shin. Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, 54:159–178, 1992.

[109] Xuyong Li, Jinhua Zheng, and Juan Xue. A diversity metric for multi-objective evolutionary algorithms. In *ICNC'05*, volume 3612 of *LNCS*, pages 68–73. Springer, 2005.

[110] Yun-Chia Liang and Alice E. Smith. An ant system approach to redundancy allocation. In *Congress on Evolutionary Computation*, pages 1478–1484. IEEE, 1999.

[111] Yun-Chia Liang and Alice E. Smith. An ant colony optimization algorithm for the redundancy allocation problem (RAP). *IEEE Transactions on Reliability*, 53(3):417–423, 2004.

[112] J. Lis and M. Lis. Self-adapting parallel genetic algorithm with the dynamic mutation probability, crossover rate and population size. In J. Arabas, editor, *Proceedings of the 1st Polish National Conference in Evolutionary Computation*, page 324329, 1996.

[113] Joanna Lis. Parallel genetic algorithm with the dynamic control parameter. In *International Conference on Evolutionary Computation*, pages 324–329, 1996.

[114] F. G. Lobo and David E. Goldberg. Decision making in a hybrid genetic algorithm. In *Proceedings of the Congress on Evolutionary Computation*, pages 121–125. IEEE Press, 1997.

[115] Fernando G. Lobo. Idealized dynamic population sizing for uniformly scaled problems. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings*, pages 917–924. ACM, 2011.

[116] Fernando G. Lobo and Cláudio F. Lima. A review of adaptive population sizing schemes in genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2005, Workshop Proceedings, Washington DC, USA, June 25-26, 2005*, pages 228–234. ACM, 2005.

[117] Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich. Efficient symbolic multi-objective design space exploration. In *ASP-DAC 2008*, pages 691–696. IEEE, 2008.

[118] Thilo Mahnig and Heinz Muhlenbein. A new adaptive boltzmann selection schedule SDS. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 183–190. IEEE Press, 2001.

[119] S. Malek, M. Mikic-Rakic, and N. Medvidovic. A decentralized redeployment algorithm for improving the availability of distributed systems. In *Comp. Dep.*, volume 3798 of *LNCS*, pages 99–114. Springer, 2005.

[120] Sam Malek. *A User-Centric Approach for Improving a Distributed Software System's Deployment Architecture.* PhD thesis, Faculty of the graduate schools, University of Southern California, 2007.

[121] Anne Martens and Heiko Koziolek. Automatic, model-based software performance improvement for component-based software designs. In *6th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA)*. Elsevier, 2009.

[122] Keith E. Mathias and Darrell Whitley. Initial performance comparisons for the delta coding algorithm. In *International Conference on Evolutionary Computation*, pages 433–438, 1994.

[123] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, and Michèle Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *IEEE Congress on Evolutionary Computation*, pages 365–372. IEEE, 2009.

[124] Jorge Maturana, Frédéric Lardeux, and Frédéric Saubion. Autonomous operator management for evolutionary algorithms. *J. Heuristics*, 16(6):881–909, 2010.

[125] Jorge Maturana and Frederic Saubion. A compass to guide genetic algorithms. In Gunter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *Parallel Problem Solving from Nature – (10th PPSN'08)*, volume 5199 of *Lecture Notes in Computer Science (LNCS)*, pages 256–265. Springer-Verlag, 2008.

[126] Nenad Medvidovic and Sam Malek. Software deployment architecture and quality-of-service in pervasive environments. In *Workshop on the Engineering of Software Services for Pervasive Environements, ESSPE*, pages 47–51. ACM, 2007.

[127] Indika Meedeniya, Aldeida Aleti, and Barbora Bühnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium of Avionics/Automotive Systems Engineering (SAASE'09), San Diego, CA*, 2009.

[128] Indika Meedeniya, Barbora Buhnova, Aldeida Aleti, and Lars Grunske. Architecture-Driven Reliability and Energy Optimization for Complex Em-

bedded Systems. In *Research into Practice - Reality and Gaps, 6th International Conference on the Quality of Software Architectures (QoSA 2010)*, pages 52–67. Springer, 2010.

[129] Indika Meedeniya, Barbora Buhnova, Aldeida Aleti, and Lars Grunske. Reliability-Driven Deployment Optimization for Embedded Systems. *Journal of Systems and Software*, 2011.

[130] Indika Meedeniya, Irene Moser, Aldeida Aleti, and Lars Grunske. Architecture-based reliability evaluation under uncertainty. In *7th International Conference on the Quality of Software Architectures, QoSA 2011 and 2nd International Symposium on Architecting Critical Systems, ISARCS 2011. Boulder, CO, USA, June 20-24, 2011, Proceedings*, pages 85–94. ACM, 2011.

[131] R.E. Mercer and J.R. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes*, 7:215–228, 1978.

[132] Silja Meyer-Nieberg and Hans-Georg Beyer. Self-adaptation in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 47–75. Springer, 2007.

[133] Efrn Mezura-Montes and Ana Palomeque-Ortiz. Self-adaptive and deterministic parameter control in differential evolution for constrained optimization. In Efrn Mezura-Montes, editor, *Constraint-Handling in Evolutionary Optimization*, volume 198 of *Studies in Computational Intelligence*, pages 95–120. Springer Berlin / Heidelberg, 2009.

[134] Zbigniew Michalewicz and David B. Fogel. *How to solve it: modern heuristics*. Springer-Verlag, 2004.

[135] Zbigniew Michalewicz and Martin Schmidt. Parameter control in practice. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 277–294. Springer, 2007.

[136] Marija Mikic-Rakic, Sam Malek, Nels Beckman, and Nenad Medvidovic. A tailorable environment for assessing the quality of deployment architectures in highly distributed settings. In *Component Deployment,CD'04*, volume 3083 of *LNCS*, pages 1–17. Springer, 2004.

[137] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Complex Adaptive Systems. MIT-Press, Cambridge, 1996.

[138] James Montgomery and Irene Moser. Parallel constraint handling in a multi-objective evolutionary algorithm for the automotive deployment problem. In *e-Science Workshops, 2010 Sixth IEEE International Conference on*, pages 104 –109, 2010.

[139] Sanaz Mostaghim. Parallel multi-objective optimization using self-organized heterogeneous resources. In *Parallel and Distributed Computational Intelligence*, volume 269 of *Studies in Computational Intelligence*, pages 165–179. Springer, 2010.

[140] Sanaz Mostaghim and Hartmut Schmeck. Self-organized parallel cooperation for solving optimization problems. In *Architecture of Computing Systems - ARCS 2009, 22nd International Conference*, volume 5455 of *Lecture Notes in Computer Science*, pages 135–145. Springer, 2009.

[141] Richard Myers and Edwin R. Hancock. Empirical modelling of genetic algorithms. *Evolutionary Computation*, 9(4):461–493, 2001.

[142] Farhad Nadi and Ahamad Tajudin Abdul Khader. A parameter-less genetic algorithm with customized crossover and mutation operators. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings*, pages 901–908. ACM, 2011.

[143] Volker Nannen and Ágoston Endre Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings*, pages 183–190. ACM, 2006.

[144] Volker Nannen and Ágoston Endre Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In Manuela M. Veloso, editor, *IJCAI'07, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 975–980, 2007.

[145] Mark Nicholson. *Selecting a Topology for Safety-Critical Real-Time Control Systems*. PhD thesis, Department of Computer Science, University of York, 1998.

[146] Barbara Paech and Thomas Wetter. Rational quality requirements for medical software. In *30th International Conference on Software Engineering (ICSE 2008)*, pages 633–638. ACM, 2008.

[147] Yiannis Papadopoulos and Christian Grante. Evolving car designs using model-based automated safety analysis and optimisation techniques. *The Journal of Systems and Software*, 76(1):77–89, 2005.

[148] Vilfredo Pareto. *CoursD'Economie Politique*. F. Rouge, 1896.

[149] A. N. Pettitt and M. A. Stephens. The kolmogorov-smirnov goodness-of-fit statistic with discrete and grouped data. *Technometrics*, 19(2):205–210, 1977.

[150] Alexander Pretschner, Manfred Broy, Ingolf H. Krüger, and Thomas Stauner. Software engineering for automotive systems: A roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 55–71. IEEE Computer Society, 2007.

[151] Alexander Pretschner, Manfred Broy, Ingolf H. Krüger, and Thomas Stauner. Software engineering for automotive systems:Aroadmap. In *International Conference on Software Engineering,ISCE'07*, pages 55–71, 2007.

[152] Ronald L. Rardin and Reha Uzsoy. Experimental evaluation of heuristic optimization algorithms:A tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[153] Ronald L. RardinBenjamin W. Lin. A short convergence proof for a class of ant colony optimization algorithms. *Management Science*, 25:1258–1271, 1980.

[154] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution.* Frommann-Holzboog, 1973.

[155] Ingo Rechenberg. Cybernetic solution path of an experimental problem. Technical report, Royal Air Force Establishment, 1965.

[156] Yosef Rinott. On two-stage selection procedures and related probability-inequalities. *Communications in Statistics - Theory and Methods*, 7(8):799–811, 1978.

[157] R. Rogenmoser, H. Kaeslin, and T. Blickle. Stochastic methods for transistor size optimization of CMOS VLSI circuits. *LNCS*, 1141:849–869, 1996.

[158] R. S. Rosenberg. *Simulation of genetic populations with biochemical properties.* PhD thesis, University of Michigan, Ann Harbor, Michigan, 1967.

[159] Günter Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Transaction Evolutionary Computation*, 5(4):410–414, 2001.

[160] J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufman, 1989.

[161] J. David Schaffer and Amy Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In John J. Grefenstette, editor, *Genetic Algorithms and their Applications (ICGA'87)*, pages 36–40. Lawrence Erlbaum Associates, 1987.

[162] Th. Scharnhorst, H. Heinecke, K. P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, P. Heitkmper, J. Leflour, J.-L. Mat, and K. Nishikawa. Autosar challenges and achievements. In *VDI Berichte Nr. 1907*, pages 395–408, 2005.

[163] D. Schlierkamp-Voosen and H. Mühlenbein. Strategy adaptation by competing subpopulations. *Lecture Notes in Computer Science*, 866:199–208, 1994.

[164] Bruce Schmeiser. *Simulation Experiments*. Mathematical-statistical framework for variance reduction / Barry L. Nelson and Bruce W. Schmeiser. Purdue University, Department of Statistics, 1990.

[165] Jason R. Schott. FaultTolerantDesignUsingSingle andMulticriteriaGeneticAlgorithmOptimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 1995.

[166] H. P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universitat Berlin, 1975.

[167] Mike Sewell, Jagath Samarab, Ranga Rodrigo, and Kenneth Mcisaac. The rank-scaled mutation rate for genetic algorithms. *International Journal of Information Technology*, 3(1):31–36, 2005.

[168] Craig G. Shaefer. The ARGOT strategy: Adaptive representation genetic optimizer technique. In John J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Publishers, 1987.

[169] Vibhu Saujanya Sharma, Pankaj Jalote, and Kishor S. Trivedi. Evaluating performance attributes of layered software architecture. In *Component-Based Software Engineering, 8th International Symposium, CBSE*, volume 3489 of *LNCS*, pages 66–81. Springer, 2005.

[170] Vibhu Saujanya Sharma and Kishor S. Trivedi. Quantifying software performance, reliability and security: An architecture-based approach. *Journal of Systems and Software*, 80(4):493–509, 2007.

[171] Sol M. Shatz, Jia-Ping Wang, and Masanori Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Transactions on Computers*, 41(9):1156–1168, 1992.

[172] Selmar K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *IEEE Congress on Evolutionary Computation*, pages 399–406. IEEE, 2009.

[173] Selmar K. Smit, Ágoston Endre Eiben, and Zoltán Szlávik. An MOEA-based method to tune EA parameters on multiple objective functions. In *Proceedings of the International Conference on Evolutionary Computation, (ICEC 2010)*, pages 261–268. SciTePress, 2010.

[174] Jim Smith and Terence C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *International Conference on Evolutionary Computation*, pages 318–323, 1996.

[175] Robert E. Smith. Adaptively resizing populations: An algorithm and analysis. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA'93)*, page 653, San Mateo, California, 1993. Morgan Kaufmann Publishers.

[176] William M. Spears. Adapting crossover in evolutionary algorithms. In *Evolutionary Programming*, pages 367–384. MIT Press, 1995.

[177] M. Srinivas and Lalit M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, 1994.

[178] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *EvolutionaryComputation*, 2(3):221–248, 1995.

[179] Christopher R. Stephens, I. Garcia Olmedo, J. Mora Vargas, and Henri Waelbroeck. Self-adaptation in evolving systems. *Artificial Life*, 4(2):183–201, 1998.

[180] Dirk Thierens. Adaptive mutation rate control schemes in genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 980–985. IEEE Press, 2002.

[181] Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In Hans-Georg Beyer and Una-May O'Reilly, editors, *Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 1539–1546. ACM, 2005.

[182] Dirk Thierens. Adaptive strategies for operator allocation. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 77–90. Springer, 2007.

[183] Andrew Tuson and Peter Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6(2):161–184, 1998.

[184] Péter Vajda, Ágoston Endre Eiben, and Wiebe Hordijk. Parameter control methods for selection operators in genetic algorithms. In *Parallel Problem Solving from Nature, 10th International Conference, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*, pages 620–630. Springer, 2008.

[185] David A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Air Force Institute of Technology, 1999.

[186] Peter Wallin, Joakim Froberg, and Jakob Axelsson. Making decisions in integration of automotive software and electronics: A method based on atam and ahp. In *SEAS '07: Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems*, page 5, 2007.

[187] R. Weinberg. *Computer Simulation of a Living Cell*. PhD thesis, The University of Michigan, 1970.

[188] James M. Whitacre. Adaptation and self-organization in evolutionary algorithms. *CoRR*, abs/0907.0516, 2009.

[189] James M. Whitacre, Tuan Q. Pham, and Ruhul A. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. *CoRR*, abs/0907.0595, 2009.

[190] Darrell Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 116–123, San Mateo, California, 1989. Morgan Kaufmann Publishers, Inc.

[191] Yuk-Yin Wong, Kin-Hong Lee, Kwong-Sak Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing*, 7(8):506–515, 2003.

[192] Bo Yuan and Marcus Gallagher. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 172–181. Springer-Verlag, 2004.

[193] Ruiqing Zhao and Baoding Liu. Redundancy optimization problems with uncertainty of combining randomness and fuzziness. *European Journal of Operational Research*, 157(3):716–735, 2004.

[194] Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization:Methods and Applications*. PhD thesis, Swiss Federal Institute of Tech. (ETH), 1999.

[195] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms:Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.

[196] Eckart Zitzler, Marco Laumanns, Lothar Thiele, Carlos M. Fonseca, and V.Grunert da Fonseca. Why Quality Assessment of Multiobjective Optimizers Is Difficult. In *GECCO'02*, pages 666–673. Morgan Kaufmann, 2002.

[197] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms:A comparative case study and the strength pareto approach. *IEEE-Evolutionary Computation*, 3(4):257–271, 1999.

[198] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and da V. G. Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Tran. on Evolutionary Comp.*, 7:117–132, 2003.